ABSTRACT

Title of dissertation: INTEGRATED FIELD INVERSION AND MACHINE LEARNING WITH EMBEDDED NEURAL NETWORK TRAINING FOR TURBULENCE MODELING

Jonathan Holland
Doctor of Philosophy, 2019

Dissertation directed by: Dr. James Baeder, Professor
Department of Aerospace Engineering
University of Maryland, College Park

Dr. Karthik Duraisamy, Associate Professor
Department of Aerospace Engineering
University of Michigan, Ann Arbor

A rich set of experimental and high fidelity simulation data is available to improve Reynolds Averaged Navier Stokes (RANS) models of turbulent flow. In practice, using this data is difficult, as measured quantities cannot be used to improve models directly. The Field Inversion and Machine Learning (FIML) approach addressed this challenge through an inference step, in the form of an inverse problem, which treats inconsistencies between the models and the data in a consistent manner. However, a separate learning algorithm is not always able to be learned from the generated inverse problem data accurately. Two new methods of incorporating higher fidelity data into RANS turbulence models via machine learning are proposed and applied for the first time in this thesis. Both build on the FIML

framework by performing learning during the inference step, instead of considering the inference and learning steps separately as in the classic FIML approach.

The first new approach embeds neural network learning into the RANS solver, and the second trains the weights of the neural network directly. Additionally, for the first time, the inverse problem can incorporate higher fidelity data from multiple cases simultaneously, promising to improve the generalization of the augmented model. The two new methods and the classic approach are demonstrated with a simple model problem, as well as a number of challenging RANS cases. For a 2D airfoil case, all three FIML augmentations are shown to improve predictions, with the new methods demonstrating increased regularization. Additionally, a model augmentation is generated by considering seven angles of attack of an airfoil in the inference step, and the augmentation is shown to improve predictions on a different airfoil. Additional cases are considered including a transonic shock wave boundary layer interaction and the NASA wall-mounted hump. In all cases, the inference is shown to improve predictions. For the first time, the inverse problem accounts for the limitations of the learning procedure, guaranteeing that the model discrepancy is optimal for the chosen learning algorithm. The results in this thesis prove that learning during the inference step provides additional regularization, and guarantees the inference produces *learnable* model discrepancy.

Integrated Field Inversion and Machine Learning With Embedded Neural Network Training for Turbulence Modeling

by

Jonathan Richard Holland

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Advisory Committee:
Dr. James Baeder, Chair/Advisor
Dr. Karthik Duraisamy, co-Advisor
Dr. Stuart Laurence
Dr. Kenneth Yu
Dr. Robert Sanner
Dr. Nikhil Chopra

Acknowledgments

This research was supported by the Vertical Lift Research Center Of Excellence (VLRCOE) at the University of Maryland and the ONR program *Advancing Predictive Strategies for Wall-Bounded Turbulence by Fundamental Studies and Data-driven Modeling* at the University of Michigan. Computational resources were provided by the University of Maryland Deepthought2 High Performance Computing cluster.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AoA	Angle of Attack
APG	Adverse Pressure Gradient
BFGS	Broyden Fletcher Goldfarb Shanno Quasi-Newton Minimization Algorithm
CFD	Computational Fluid Dynamics
DNS	Direct Numerical Simulation
DES	Detached Eddy Simulation
FIML	Field Inversion and Machine Learning
HPC	High Perfomance Computing
LES	Large Eddy Simulation
L-BFGS-B	Limited Memory BFGS Method
NASA	National Aeronautics and Space Administration
RANS	Reynolds Averaged Navier Stokes
SA	Spalart-Allmaras Turbulence Model
SA-BC	Spalart-Allmaras Bas Cakmakcioglu One Equation Algebraic Transition Model
SGD	Stochastic Gradient Descent
SSE	Sum Squared Error
SU2	Stanford University Unstructured
SWTBLI	Shock Wave Turbulent Boundary Layer Interaction

List of Symbols

β	Correction to Turbulent Production
$C_p,\ C_D,\ C_L$	Pressure, Drag, and Lift Coefficients
$c_p,\ c_v$	Specific Heat at Constant Pressure and Volume
D	Turbulent Destruction in Spalart-Allmaras Model
δ	Delta Criterion
e	Internal Engergy
γ	Intermittency
$J_c,\ J_e,\ J_d$	FIML-Classic, Embedded, and Direct Objective Functions
k	Thermal Conductivity
κ	Von Kármán Constant
$k_m,\ k_d$	Model Output and Higher Fidelity Data
λ	Objective Function Regularization Constant
M	Mach Number
$\mathcal{M}$	Model
ν	Kinematic Viscosity
ν_t	Eddy Viscosity
$\hat{\nu}$	Spalart-Allmaras Model Transported Variable
Ω	Vorticity
P	Turbulent Production in Spalart-Allmaras Model
p	Pressure

R	Ideal Gas Constant
Re	Reynolds Number
ρ	Density
S	Strain
T	Temperature
Tu_∞	Freestream Turbulence Intensity
τ_w	Shear Stress at Wall
U^+	Non-dimensional Boundary Layer Velocity
U_τ	Friction Velocity
x_i	Spatial Dimension
y^+	Non-dimensional Distance from Wall

Chapter 1: Introduction

1.1 Motivation

Many practical fluid dynamic applications require the accurate characterization of the dynamics of turbulent boundary layers. Even simple flows, such as incompressible flow over an airfoil, can require accurate modeling of boundary layer effects to accurately predict the lift and drag even at moderate angles of attack. Other examples are readily available, such as shock wave turbulent boundary layer interactions (SWTBLI). Computational Fluid Dynamics (CFD) is commonly selected to analyze these cases; but, unfortunately turbulence modeling for CFD is notoriously difficult. Despite immense effort and resources many practical engineering applications are still difficult to predict computationally. The continued accelerating growth of computational power predicted by Moore's law has enabled the development of modeling applications with higher fidelity and potential accuracy, such as Large Eddy Simulation (LES) and Direct Numerical Simulation (DNS). However, LES remains out of reach for most practical engineering applications, introduces additional modeling uncertainties, and still requires immense computational resources. DNS has incredible usefulness for providing the exact solution but will remain out of reach for anything but the simplest applications for decades. RANS

modeling will remain essential for fluid dynamics predictions, yet progress in RANS accuracy improvements has been frustratingly slow. Experiments, DNS, and LES have provided a wealth of information to leverage for turbulence modeling; unfortunately RANS models have yet to see substantial gains by incorporating inferences from these more fundamental sources of information.

Machine learning provides an exciting new avenue to incorporate inferences from these higher fidelity simulations and experiments into RANS simulations, to increase accuracy and reliability. It has been demonstrated, that by careful problem definition, corrections to RANS models can be generated for the fluid dynamic problem of interest to the modeler. These model discrepancies can be used by the modeler to improve RANS predictions. Additionally, by leveraging machine learning methods a machine learned model can be generated that corrects and augments the turbulence model in a predictive environment. The application of these machine learning methods is not intended to replace the modeler, but rather provide an avenue of generating inferences and incorporating patterns that are critical to RANS accuracy, but may not be apparent to the modeler through other traditional analysis techniques. In summary, the lack of higher order CFD techniques for high-Reynolds number flows and the difficulty of incorporating higher fidelity data into RANS models provides strong motivation for developing methods of inferring corrections and augmenting existing models using machine learning.

1.2 Physical Problem

Viscosity slows the fluid at the wall to zero when fluid flows over a wall. In a thin layer near the wall, a velocity profile develops where the velocity is zero at the wall and increases in velocity away from the wall until it reaches its freestream value. This thin layer where viscous effects are critical is known as the boundary layer and its concept is attributed to Prandtl [5]. Boundary layers can be divided into two types: laminar and turbulent. Laminar boundary layers are characterized by the streamlines of the fluid being largely parallel to the wall with little mixing between the layers of the boundary layer and negligible motion normal to the wall. Laminar boundary layers can be destabilized through a variety of mechanisms (surface roughness, freestream turbulence, curvature, etc) resulting in transition to turbulent boundary layers. Transition is the process by which the ordered, parallel streamlines of the laminar boundary layer destabilize into the disordered, chaotic structure of the turbulent boundary layer.

Despite being contained in a very thin layer near the wall, the turbulent boundary layer is responsible for numerous important fluid dynamic phenomena. Due to increased mixing in the turbulent boundary layer, the higher momentum from layers farther from the wall is thrown closer to the wall. This creates an increased velocity gradient near the wall, resulting in increased skin friction and heat transfer. Additionally, the boundary layer can separate when subjected to an adverse pressure gradient. Turbulent boundary layers are less susceptible to boundary layer separation due to the increased momentum near the wall, which counteracts an adverse

pressure gradient. This effect is important both for airfoils at high angles of attack and for shock boundary layer interactions, as the boundary layer in both cases is subjected to a strong adverse pressure gradient.

1.3 A Brief History of Turbulent Boundary Layer Analysis

Reynolds first distinguished between laminar and turbulent flows with his famous experiments in 1883. He observed dyed fluid in pipes and noted the point at which the dye became irregular (where the fluid transitions from laminar to turbulent flow). Through dimensional arguments and observations he discovered the now ubiquitous Reynolds number [6].

$$Re = \frac{\rho U L}{\mu}$$

This dimensionless quantity measures the ratio of the inertial forces (numerator) to the viscous forces (denominator). Reynolds observed that the flow in a pipe transitions to turbulence at particular Reynolds numbers, and additionally observed that the transition location is particularly dependent on the upstream flow conditions (i.e. transition location dependent on upstream disturbances) [6] [7].

The concept of the "law of the wall", first presented by von Kármán, was developed from first principles and observations and was quickly accepted. This foundational work presented a simple relation describing the velocity profile of turbulent boundary layers [8]. Subsequent experiments, such as the experiments of Coles, found experimental values for the constants in the relation to calibrate the

theory. The theory has been widely accepted and forms the basis and rationale for numerous computational models, including RANS turbulence models. Despite the ubiquitous nature of the law of the wall it is still the subject of some controversy. There is still significant debate over the constants of the theory, or even if the law of the wall is valid in a general sense. The debate is perhaps best explained by George [10].

Turbulent boundary layer analysis has increasingly made use of high fidelity simulations, due to the continuous advancement of computational resources. The first fully resolved DNS simulations of a turbulent channel were presented by Kim et al. [11]. This data is extremely valuable, and DNS data has in general proven to be the most accurate way of analyzing and understanding turbulent physics. Unfortunately, due to computational limitations DNS remains far out of reach for practical Reynolds numbers. Experiments can measure flows with Reynolds numbers far out of reach of DNS simulations, but physical limitations of experimental methods still prevent direct measurement of turbulent statistics from Reynolds numbers relevant to large aircraft [10].

1.4 Physics of Boundary Layers

Turbulent boundary layer analysis is a rich field with an immense volume of established research and thought. This section briefly introduces the terminology and underlying physical principles of turbulent boundary layer analysis and modeling.

1.4.1 Turbulent Boundary Layer Structure

The fluid dynamics of a turbulent boundary layer are extremely complex. It is often useful to analyze the turbulent boundary layer by looking at the mean velocity profile. The instantaneous velocity in a turbulent boundary layer is chaotic, but in a mean sense the velocity will be nearly parallel to the wall similar to the laminar boundary layer. Thus, we can simplify the velocity field to a mean velocity profile $\bar{U}(y)$ along the wall with y being the distance from the wall. These are dimensional variables but typically boundary layer velocity profiles are analyzed using the non-dimensional plus coordinates:

$$y^+ = \frac{U_\tau y}{\nu} \tag{1.1}$$

$$U^+ = \frac{\bar{U}}{U_\tau} \tag{1.2}$$

$$U_\tau \equiv \left(\frac{\tau_w}{\rho}\right) \tag{1.3}$$

Where ν is the kinematic viscosity and τ_w is the shear stress at the wall. By noting that the mean velocity profile near the wall will depend on the distance to the wall (y), the shear stress at the wall (τ_w), the kinematic viscosity (ν), and the fluid density (ρ) from dimensional analysis we can find the law of the wall, which concludes that the mean velocity profile ($\bar{U}$) will be a function of the distance from the wall [12].

$$U^+ = f(y^+) \tag{1.4}$$

Nearest to the wall the viscous forces dominate in a region commonly referred to as the viscous sublayer. In this region (in plus coordinates) the mean velocity is equivalent to the distance from the wall ($U^+ = y^+$). At approximately $y^+ \approx 5$ there begins a buffer layer which connects the viscous sublayer to the log-law region.

von Kármán first proposed the since widely accepted logarithmic law of the wall [8]. From first principles the law of the wall can be written:

$$U^+ = \frac{1}{\kappa} \ln y^+ + B_i \tag{1.5}$$

Conceptually, in plus coordinates, the turbulent boundary layer velocity profile is illustrated in Figure 1.1.

1.4.2 Factors Affecting Turbulent Boundary Layers

Boundary layer analysis is more difficult when considering more practical engineering flows, such as turbulent flow over an airfoil, or the reaction of the boundary layer to a sudden increase in pressure from a shock wave. Many factors can influence a turbulent boundary layer, and some of these factors are introduced in this section.

Real walls are not perfectly smooth, and naturally a viscous fluid interacting with a wall boundary will be influenced by the roughness of that boundary. Surface roughness primarily influences the viscous sublayer (nearest layer to the wall) and has little effect on the more outer layers. As the surface roughness increases the log

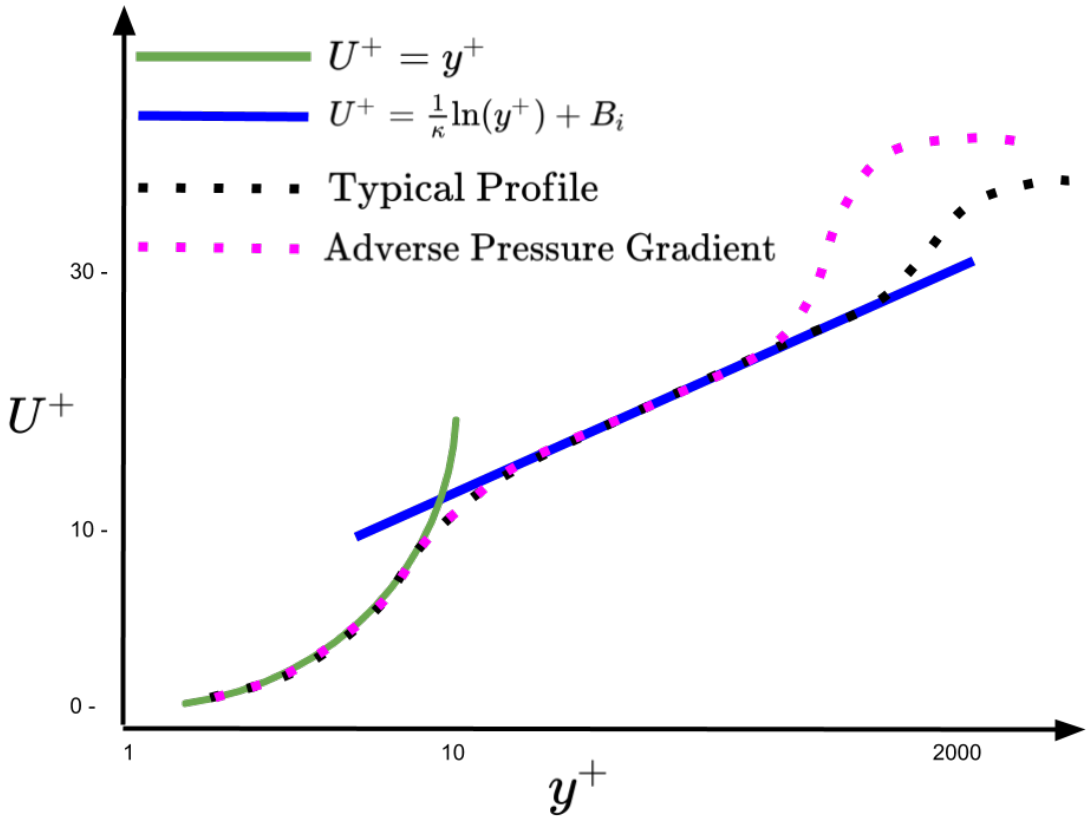

Figure 1.1: Illustration (not to scale) of a typical turbulent boundary layer velocity profile and a boundary layer subject to an adverse pressure gradient, adapted from [1, 2, 3].

region will move lower and to the right in Figure 1.1. This will increase the velocity gradient at the wall. Because the shear stress at the wall is proportional to this gradient (1.6), an increase in surface roughness is associated with an increase in τ_w and associated parameters, such as the skin friction ($C_f = \tau_w/(0.5\rho v^2)$) [1].

$$\tau_w = \mu \frac{\partial U}{\partial y} \tag{1.6}$$

If surface roughness is considered at all it is often simply correlated only to the

average roughness size, even though it is clear that this is not the only parameter required to capture the effects of surface roughness [1]. For all the cases considered in this thesis, the boundary layer data was obtained from experiments with models manufactured to be as smooth as possible. Therefore, as is typical, surface roughness effects are neglected.

Most importantly for the present work, boundary layers are strongly influenced by pressure gradients. Opposite to surface roughness, pressure gradients have the strongest effect on the outer layer. If subjected to an adverse pressure gradient ($dp/dx > 0$) the fluid will slow, and the boundary layer profile shown in Figure 1.1 will begin to show a slower velocity profile in the outer layer. It has been shown that the inner layer velocity profile is remarkably resilient to the influence of pressure gradients [1], however, in a strong adverse pressure gradient the defect layer begins closer to the wall as the outer layer velocity slows [3]. Conceptually, this is shown in Figure 1.1. Ultimately, given a strong enough adverse pressure gradient the momentum in the defect layer can slow to the point that the boundary layer separates. Therefore, accurately modeling the behavior of the boundary layer is critical for a variety of practical engineering flows.

1.4.3 Shock Wave Turbulent Boundary Layer Interactions

Another challenge for turbulent boundary layer analysis is shock wave boundary layer interaction. A shock wave marks a very rapid increase in pressure, temperature, and density. In many fluid dynamic problems of interest a shock wave

impinges on a turbulent boundary. For example, transonic flow over the suction side of a transonic airfoil, as well as supersonic inlets and isolators are common examples of flow conditions that result in shock boundary layer interactions. When a shock wave impinges on a turbulent boundary layer the boundary layer is subjected to a strong adverse pressure gradient. The reaction of the boundary layer to the shock wave can be very difficult to predict.

1.5 Challenges in Turbulent Boundary Layer Modeling

Turbulent boundary layer predictions for many practical engineering applications rely on boundary layer models. Experimental data is often expensive, or difficult to obtain. Computational fluid dynamic (CFD) models of turbulent boundary layers are often used to generate predictions. CFD seeks to numerically solve the physical equations that govern fluid flows: the Navier Stokes equations. Ideally, the exact equations could be solved (DNS), but for turbulent flows at practical Reynolds numbers it is not computationally possible to resolve the exceedingly small turbulent eddies that define the dynamics of these problems. The required grid sizes to do so are exceedingly small, requiring immense computational resources. LES relaxes this requirement somewhat, but still requires unattainable resources for high Reynolds number cases and introduces additional model uncertainties. Therefore, for practical turbulent CFD predictions it is common to apply the RANS equations. For RANS, the effects of turbulence are modeled, and the development, calibration, testing, uncertainty quantification, and validation of these models on all varieties

of turbulent flows has been an intense area of research for decades. This section will focus on the difficulties of RANS predictions for two broad classes of turbulent applications used in the current work: turbulent flow over airfoils and shock wave turbulent boundary layer interactions.

1.5.1 RANS Modeling of Turbulent Flows Over Airfoils

While RANS models yield excellent predictions for a variety of applications, there remain many common engineering applications for which RANS models remain surprisingly deficient. Specifically, for 2D flow over subsonic airfoils at low angles of attack predictions of the forces and moments on the airfoil are generally well predicted by a variety of RANS models. However, as the angle of attack is increased a strong adverse pressure gradient develops on the upper (suction) surface. Eventually, the boundary layer will separate, and a recirculating region will form. It has been well documented that RANS models have difficulty predicting the size of this recirculating region, and therefore have an inaccurate prediction of the forces and moments once the boundary layer separates [3, 13, 14]. Celic and Hirschel evaluated a variety of algebraic, one, and two equation eddy viscosity models on four adverse pressure gradient cases, including separated flow over 2D airfoils. They concluded that none of the eddy viscosity models showed a clear advantage for all the considered cases of adverse pressure gradient flows. In particular, it was shown that the prediction accuracy for the 2D airfoil case was generally far poorer than the other cases considered; although this could be contributed at least in part to

additional experimental uncertainty for this case [14]. Matyushenko et al. performed a detailed investigation into the prediction errors of RANS models for 2D airfoils at high angles of attack. A variety of turbulence models were examined for a set of 2D airfoils including the S809 wind turbine airfoil. Errors in the transition model were considered, as well as various sources of experimental error were examined parametrically, to evaluate the likely sources of observed RANS prediction errors in the separated angle of attack region of the airfoils. Ultimately it was concluded that the primary source of error must be the turbulence models themselves [15]. Figure 1.2 shows results for RANS predictions of a two dimensional airfoil designed for wind turbine applications (S809 airfoil). Experimentally derived lift coefficients for this airfoil are shown from Somers and Tangler [16], and are compared to RANS predictions for a range of angles of attack using the Spalart Allmaras turbulence model [17] with an algebraic transition model [18]. Note that at low angles of attack ($< 5°$) the RANS predicted lift coefficient agrees well with the experiment. At high angles of attack the result is quite poor. This result is typical for a variety of RANS models for 2D airfoils [14, 15].

All RANS closure models utilize assumptions and some level of empiricism to provide closure to the RANS equations. One of these assumptions present in the Spalart Allmaras turbulence model (and others) is the assumption of an equilibrium boundary layer in the log law region [17]. In the presence of an adverse pressure gradient this assumption is violated. The adverse pressure gradient slows the flow in the outer regions of the boundary layer, and the defect layer begins closer to the wall. As the adverse pressure gradient increases, the momentum in the log law region is

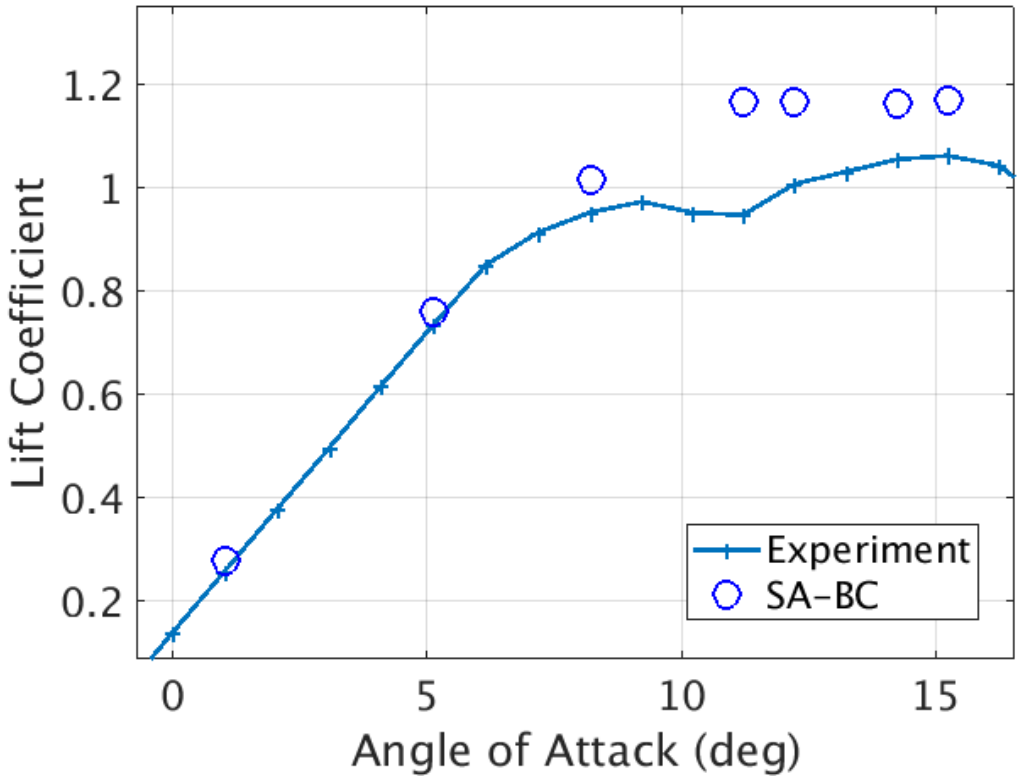

Figure 1.2: Spalart Allmaras Predicted Lift Coefficient for the S809 Airfoil Compared With Wind Tunnel Data.

decreased, and the equilibrium assumption of the eddy viscosity model is violated. As demonstrated by Medida, the eddy viscosity is over-predicted in the presence of an adverse pressure gradient [3, 13]. This over-prediction results in the RANS prediction of a boundary layer that is less susceptible to separation than observed in experiments. The resulting predicted separation location is then too far aft, and the separated region is too small. Ultimately this gives an over-prediction in the lift coefficient in the separated region. These observations are consistent with Spalart's observation in the original presentation of the model, where he noted that the model has a tendency to under-predict the thickness of the boundary layer under adverse

pressure gradients, and observed that this may lead to a prediction of a boundary layer less susceptible to separation [17].

Another violated assumption appears in the recirculating region on the suction side of the airfoil after the flow separates. RANS turbulence models are calibrated based on attached turbulent flow over simple geometries, such as flat plate boundary layers. In this regime it is natural to use the wall distance as a length scale. However, when the flow separates the wall distance is no longer an appropriate length scale, as the turbulent flow moves far from the wall. In this region it is more appropriate to use the length scale associated with the mesh size, as is used in large eddy simulation (LES) [19, 20]. Therefore, again (due to violated assumptions) it is expected that the prediction of the eddy viscosity will be inaccurate in the recirculating region. Figure 1.3 shows the eddy viscosity predicted for the S809 airfoil at a high angle of attack using the Spalart-Allmaras turbulence model. The eddy viscosity is over-predicted in the highlighted regions, due to the violated assumptions used in the formulation of the turbulence model.

1.5.2 RANS Predictions of Shock Wave Turbulent Boundary Layer Interactions

Another challenge for turbulent boundary layer analysis is shock wave boundary layer interaction. A shock wave marks a very rapid increase in pressure, temperature, and density. In many fluid dynamic problems of interest a shock wave impinges on a turbulent boundary layer. For example, transonic flow over the suc-

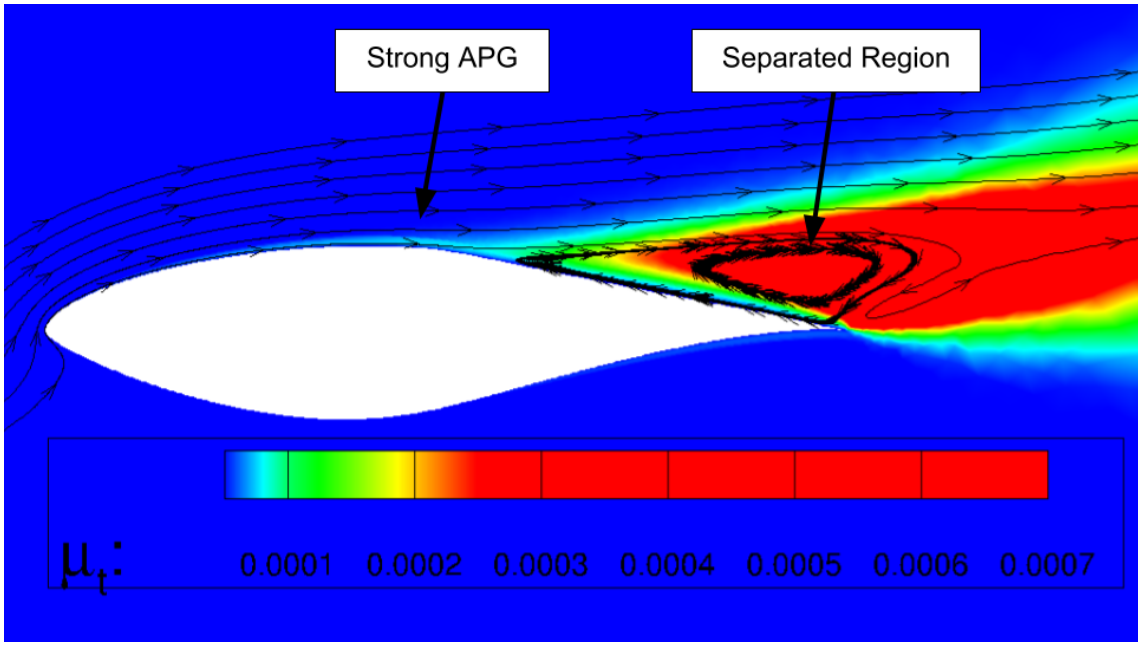

Figure 1.3: Spalart Allmaras Predicted Eddy viscosity for the S809 airfoil at 14.2° angle of attack.

tion side of a transonic airfoil, as well as supersonic inlets and isolators are common examples of flow conditions that result in shock boundary layer interactions. When a shock wave impinges on a turbulent boundary layer, the boundary layer is subjected to a strong adverse pressure gradient, and the reaction of the boundary layer to the shock wave can be very difficult to predict.

Two forms of shock boundary layer interaction will be examined in the current work. The first is a transonic airfoil as illustrated in Figure 1.4. As shown, the air accelerates over the suction side of the airfoil and a turbulent boundary layer develops. Despite the freestream Mach number being less than 1 there is a supersonic region above the airfoil. As described by Babinsky and Harvey, the upper surface of the airfoil is convex, which results in the creation of expansion waves from the

surface. These expansion waves reflect from the sonic line as compression waves, which coalesce and form a normal shock that ends the supersonic region [4]. This normal shock is characterized by a sudden increase in pressure and a decrease in velocity to subsonic conditions.

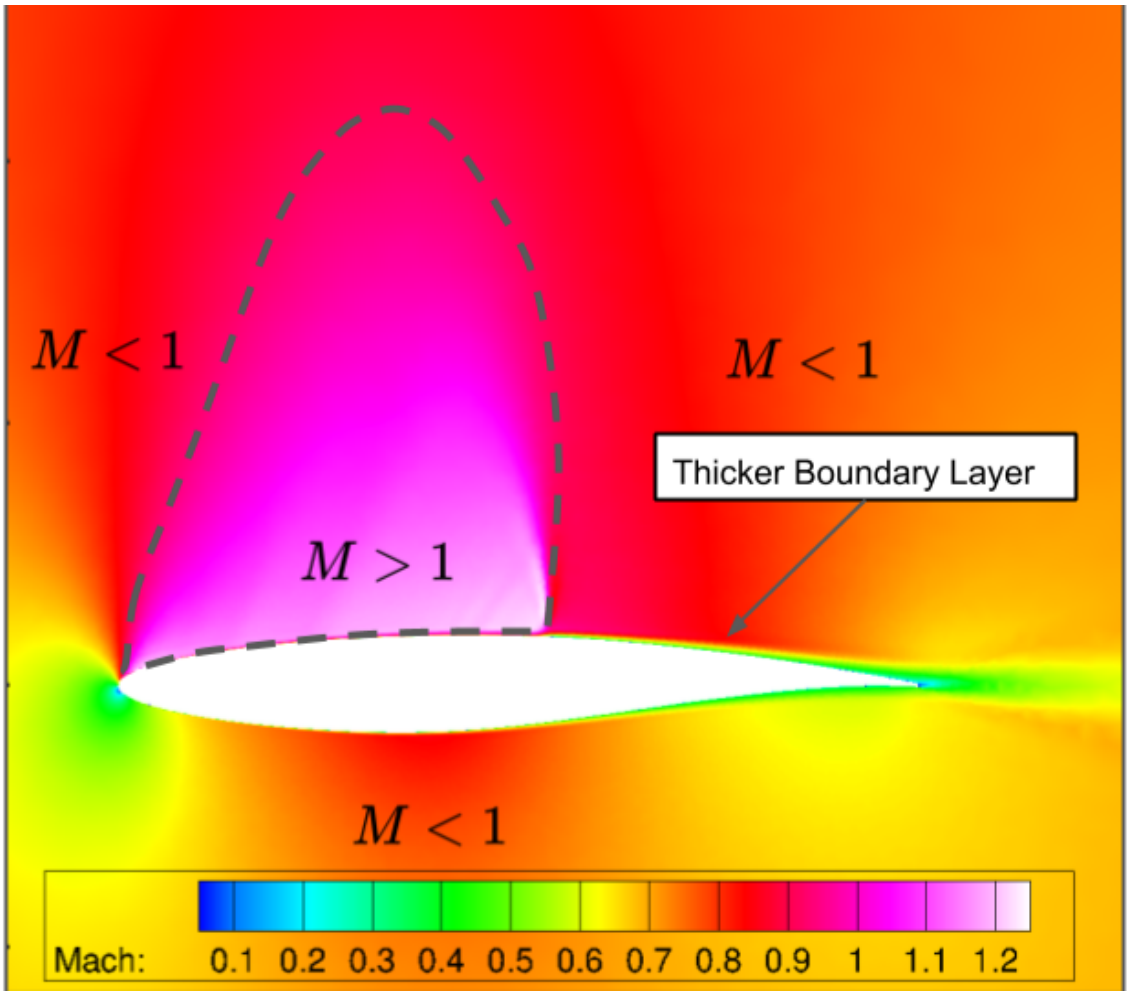

Figure 1.4: Mach contour of a transonic RAE2822 airfoil at 2.31° angle of attack and Mach 0.725.

When the shock impinges on the boundary layer a shock wave turbulent boundary layer interaction (SWTBLI) occurs. While for supersonic flow no information can travel upstream, there is a subsonic region at some point close to the wall,

separated by a sonic line, because the velocity must stagnate at the wall. For an attached interaction the subsonic region experiences an adverse pressure gradient and thickens the boundary layer not only downstream, but also some distance upstream of the shock due to the subsonic region in the boundary layer. The sonic line also moves further from the wall, which creates compression waves in the supersonic region just ahead of the impinging (nearly) normal shock wave. When visualized, either through CFD or experimental methods, these compression waves make the shock appear smeared over a small distance upstream of the interaction, which is the reason this effect is referred to as a smeared shock-foot [4]. A smeared shock-foot is illustrated in Figure 1.5.

Also note that unlike the subsonic airfoil, there is a region of favorable pressure gradient in the supersonic region because the airfoil is convex; therefore, the Mach number increases slightly in the supersonic region with a corresponding slight decrease in pressure. This favorable gradient is terminated by the strong adverse pressure gradient of the shock, and the adverse pressure gradient continues in the subsonic region following the shock. This is illustrated in Figure 1.6.

Additionally, strong transonic SWTBLI can result in flow separations that can have a substantial impact on airfoil or wing performance. A strong SWTBLI can result in a separation at the shock, which can either reattach downstream or remain separated. The increased boundary layer thickness downstream of the shock results in less momentum near the wall in the boundary layer. This gives the boundary layer downstream of a SWTBLI a decreased ability to resist separation, and can result in a separated region near the trailing edge of the airfoil [4]. Accurately predicting the

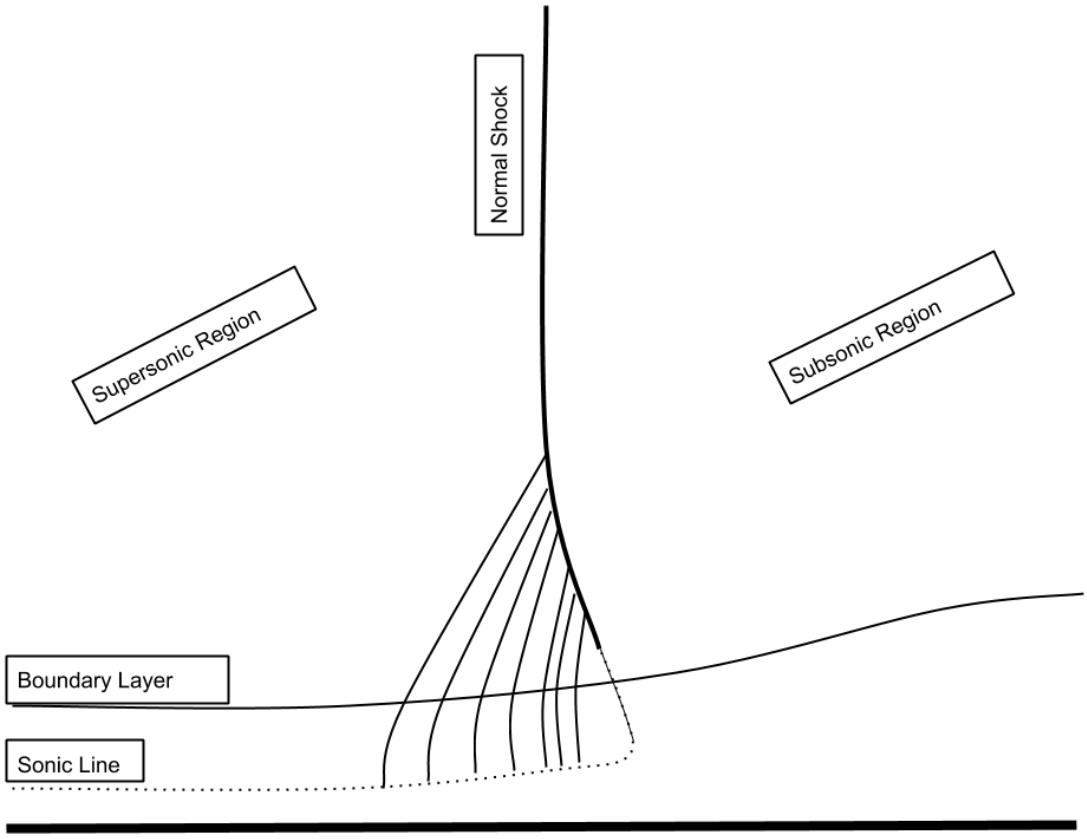

Figure 1.5: Illustration of a smeared shock foot, adapted from [4].

interaction location and flow separation remain a challenge for RANS models.

For similar reasoning as discussed for the subsonic airfoils, the boundary layer reaction to this impinging shock and associated adverse pressure gradient is difficult to predict with RANS models that are based on reasoning and calibrated for flat plate boundary layers. Note: it is expected that RANS turbulence models will perform better in the supersonic region with the favorable pressure gradient than in the strong adverse pressure gradient of the shock and the subsequent adverse pressure gradient in the subsonic region.

The second type of SWTBLI considered in this work is a turbulent boundary

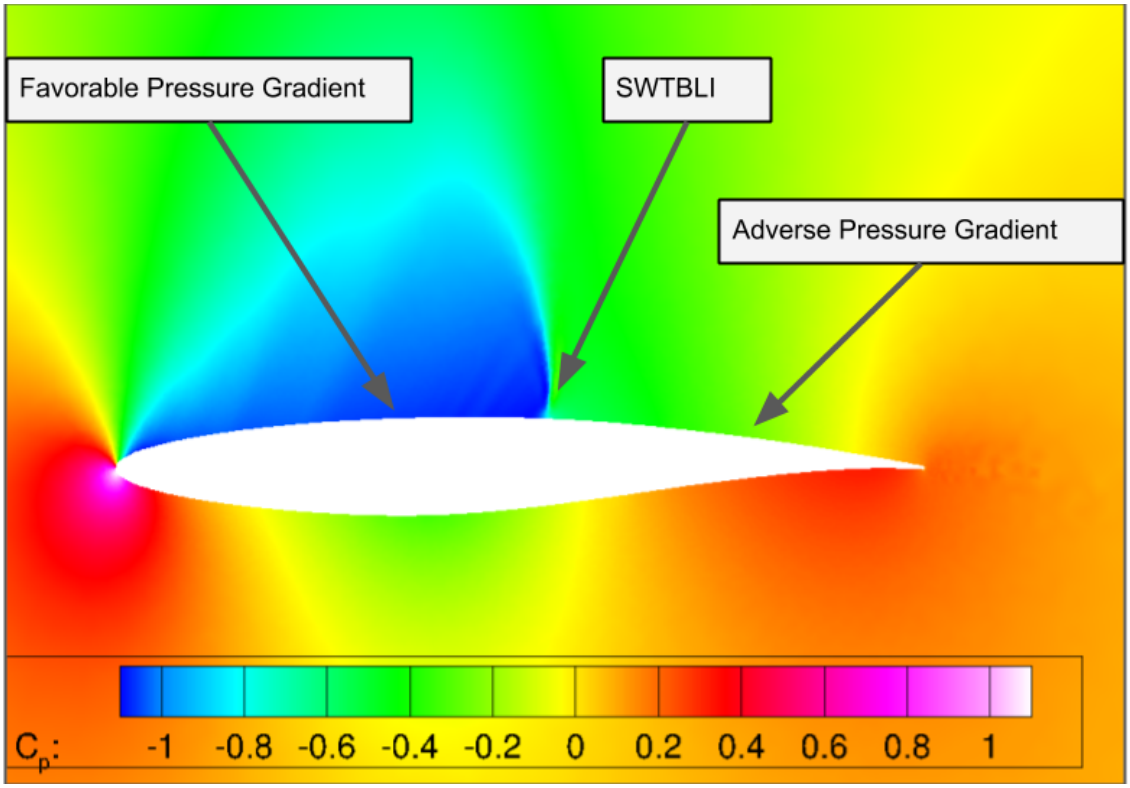

Figure 1.6: Pressure coefficient contour of a transonic RAE2822 airfoil at 2.31°angle of attack and Mach 0.725.

layer encountering a supersonic compression corner. An oblique shock forms when the supersonic air approaches a concave corner. If the compression angle is low enough, an oblique shock wave will form at the corner and turn the air to the new wall angle. This shock impinges on the boundary layer and creates a SWTBLI. If the shock is strong enough the boundary layer will separate, creating a separation bubble that extends upstream and downstream at the corner. Upstream, the separated bubble turns the supersonic air creating an oblique separation shock. The air above the bubble is characterized by a strong shear layer, which then reattaches downstream of the corner. The reattachment point generates a reattachment shock,

with a greater shock angle, such that downstream the separation and reattachment shocks impinge on each other generating a shock-shock interaction. For hypersonic cases, this interaction is characterized by very high heat fluxes with the peak heat flux occurring downstream of the reattachment point. Conceptually, the 2D separated ramp SWTBLI is shown in Figure 1.7.

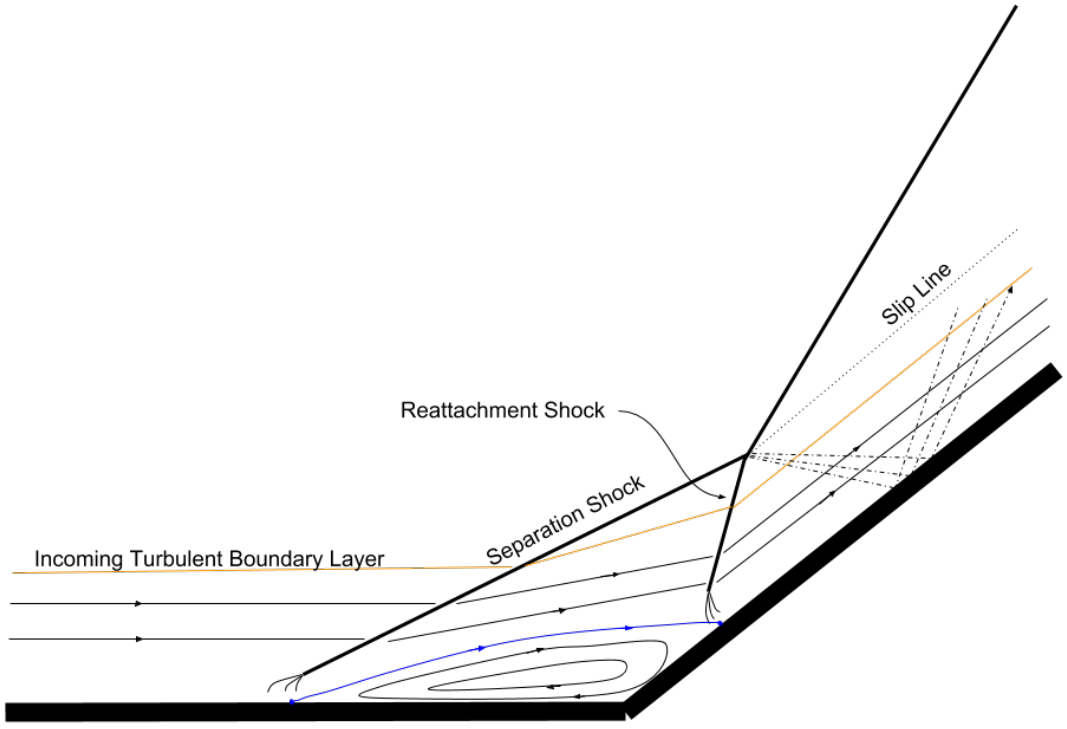

Figure 1.7: Flow structure of a ramp induced separated SWTBLI, adapted from [4].

There is a wealth of information from wind tunnel testing for 2D compression corners and transonic airfoils (and other SWTBLI) [21, 22, 23, 4, 24, 25, 26]. Despite this widely available experimental data for a variety of SBLI cases, RANS models are generally very poor at predicting the boundary layer reaction to an impinging shock.

Numerous studies have been devoted to investigating this error [27, 28, 29, 30, 21]. For transonic shock boundary layer interactions the shock/interaction location is generally poorly predicted [31, 32, 26], leading to an incorrect prediction of lift and drag on the airfoil. For compression corner interactions, RANS models are typically incapable of correctly predicting the separation length in the interaction region, and wall heat flux predictions are similarly poor [26, 21]. There does not appear to be a clear consensus whether any particular RANS turbulence model outperforms others.

1.6 Machine Learning with Neural Networks for Regression

Neural networks are commonly used in machine learning applications because of their flexibility, robustness, and relative ease of computation. Neural networks for regression allow modelers to connect inputs (features) with outputs, by generating a function based on example (training) data. What makes neural networks truly extraordinary is the ability to scale with numerous inputs[1], and their ability to be efficiently trained using massive quantities of data. Neural networks are not a new technology, but with the progress in modern computing there has been renewed interest in algorithms that can efficiently process the large quantity of data that is created by modern computing systems. Additionally, a property of neural networks that makes them particularly attractive is that they are universal approximators of nonlinear functions [33, 34].

Note that the properties of neural networks discussed here make them ideal

[1] or in other applications, numerous outputs as well

for augmenting RANS turbulence models (or other complex physical models). First, the required correction to augment a RANS turbulence model has an unknown, and non-linear functional form. It is difficult, therefore, to augment the model with a correction that could correct the model in a general sense.[2] Additionally, turbulence models utilize numerous variables in addition to the flow variables, all of which can influence the turbulent physics, and therefore meaningully affect the required correction. Neural networks can accept numerous inputs efficiently. In his book, Dreyfus states, "in general, neural networks make the best use of the available data for models with more than 2 inputs [33]." It is a safe assumption that any model aiming to augment a RANS equation will require more than two inputs due to the complexity of the problem. Due to the ability of neural networks to model general functions efficiently, incorporate a large quantity of data, and accept as many inputs as required by the problem, neural networks are employed in this work to produce augmented RANS turbulence models.

It is important to note here that neural networks do not eliminate the modeler from the learning process. In fact, like all machine learning methods there are a number of choices that the modeler must make to implement an effective neural network. These choices include the choice of inputs, scaling of the inputs, and the size and structure of the network itself. Additionally, there is a choice of how to introduce the nonlinearity of the network (activation function). These factors can

[2]Although some corrections are possible for specific applications, as the SA-APG correction proposed by Medida can be considered an augmented Spalart Allmaras model for adverse pressure gradient applications [3].

dramatically affect the performance of the learning and cannot be neglected.

1.7 Leveraging Machine Learning for Turbulence Modeling

Other researchers have presented the successful application of neural networks for physics-based simulations. Samareh and Wong used neural networks to model the dynamics of several complex physical simulations, including CFD, to predict the output of the simulation. In this way the user could be informed of the likelihood of success of the simulation before performing the computation [35]. These applications demonstrate the flexibility and efficiency that machine learning methods can have on complex physics based simulations.

Specifically for CFD, several researchers have utilized machine learning to either provide modeling inferences or enhanced models. Wang et al. utilized DNS solutions to train a machine learned correction to the Reynolds stress tensor using mean flow features. The resulting data-informed RANS predictions were shown to be substantially more accurate than the baseline model even for geometries different than the DNS training case [36]. Wu et al. used a similar methodology to produce a random forest model that provided an a-priori estimation of prediction confidence of a RANS solution [37]. Ling and Templeton used DNS and LES solutions to develop machine learned classifiers to identify where RANS turbulence models were likely to have large errors or violated assumptions [38].

Unfortunately, LES and DNS training data is unavailable at useful scales for even moderate Reynolds numbers. Additionally, directly learning from higher fi-

delity data is not likely to have much success, due to the lack of consistency between the higher fidelity data and the modeling environment. Measured data contains real, physical quantities, while RANS models employ empiricism and *modeled* quantities that are inconsistent with measured data. To resolve this inconsistency, Duraisamy and co-workers developed the Field Inversion and Machine Learning (FIML) approach [39]. Instead of learning directly from the data, an inverse problem is formulated such that information is generated that is consistent with the modeling environment in an inference procedure. A misfit function (J) between the data and the model output is formulated. One or more model discrepancy corrections (β) is (are) embedded within the model at each point on the computational domain and a gradient based optimization algorithm is used to find the optimal correction field. Since the discrepancy field is constrained by the model, the inference step resolves the inconsistency between the data and the model. Following the inference step, a feature set (η) can be constructed from the model variables and a machine learned model is trained separate (offline) from the inference step. It has been demonstrated that given appropriate training cases, the FIML approach produces a machine learned model that improves predictions [39, 40, 41, 13, 42, 43]. Despite these demonstrated successes, there remains a drawback to the FIML approach. While the inconsistency between the data and the modeling environment has been alleviated, it was not completely removed because there is no guarantee that the information generated by the inversion can be learned perfectly, if at all. This introduces an inconsistency between the information and the modeling environment due to the limitations of the chosen machine learning algorithm being neglected in

the inference step.

1.8 Objectives

The discussion in this chapter has identified several known areas where, despite a wealth of widely available experimental data, RANS models remain deficient. Additionally, machine learning with neural networks was introduced, along with a brief discussion of previous efforts to leverage machine learning methods to improve CFD predictions. Specifically, the FIML approach was shown to have a strong advantage over other machine learning augmentation methods, due to the inverse procedure; but a deficiency was noted in that imperfect training of the machine learning algorithm introduces an inconsistency between the information generated in the inversion and the model augmentation.

The objective of this thesis is to improve RANS predictions by leveraging machine learning approaches to augment RANS turbulence models. The goal is not to machine learn a new model, but to learn a correction to an existing model that improves predictions where the model is known to be deficient, but does not diminish the accuracy where the model already performs well. Additionally, the augmentation needs to be as consistent as possible to the modeling framework. To accomplish these broad goals this work builds on the FIML approach, but seeks to improve the consistency of the FIML framework by accounting for the limitations of the machine learning framework during the inversion. In doing so, the information generated by the inversion will be guaranteed to be learnable, optimal for the

chosen machine learning method, and as consistent as possible with the prediction environment. These methods must be able to incorporate enough data to produce a useful augmentation, and be efficient in order to be practically implemented on a modern high performance computing system.

1.9 Contributions of Thesis

The primary contributions of this thesis are as follows:

1. Implemented the Field Inversion and Machine Learning (FIML) framework inside the Stanford University Unstructured (SU2) code, a fully unstructured RANS solver, demonstrating robustness and practicality of the approach. For the first time, the SU2 code was adapted to perform the FIML approach.

2. Proposed and developed a new FIML approach that trains the neural network during inversion (FIML-Embedded) demonstrating improved regularization of the correction, guaranteeing that the correction can be learned, improving the consistency between the inversion and the model augmentation, and producing a neural network augmentation in the inversion step. This is a unique FIML method, entirely developed and applied for the first time in this thesis. The new method was demonstrated on both the simple model problem and several RANS applications.

3. Proposed and developed a new FIML method that utilizes a novel approach of training neural networks from physics based models (FIML-Direct). This procedure improves upon the FIML-Classic approach similarly to FIML-Embedded,

but with less complexity. This is also a unique FIML approach, entirely developed and applied for the first time in this thesis. This new method was also demonstrated on both the simple model problem and several RANS applications.

(a) Developed and applied a novel method of performing field inversions on numerous cases simultaneously, increasing generalization and preventing overtraining of the model augmentation. For the first time, the inference step can incorporate information from an unlimited number of datasets simultaneously, instead of solving multiple inverse problems as required by the classic approach.

(b) Developed efficient and scalable algorithms and demonstrated performance on high performance computing (HPC) architectures.

1.10 Scope and Organization of Thesis

Two new methods of improving upon the FIML framework were proposed, developed, and applied to both a simple model problem and a variety of RANS applications. Both new methods, wholly developed and applied for the first time in this thesis, improve on the classic FIML approach by accounting for the limitations of the chosen machine learning algorithm in the inversion process. The development, application, and results of these new approaches are documented in this thesis with the following organization.

- Chapter 1 provides an introduction and motivation to the physical problem,

as well as the major contributions of this thesis.

- Chapter 2 provides a detailed survey of previous work to augment RANS models with data via machine learning methods. Previous work on the FIML approach is discussed in particular detail.

- Chapter 3 introduces two new methods, developed for the first time in this effort, that improve upon the classic FIML approach by accounting for the limitations of the machine learning algorithm in the inversion process.

- Chapter 4 discusses the numerical methodology for the FIML approaches for both the RANS applications, and a simple 1D heat equation model problem.

- Chapter 5 presents the results for the FIML applications to the 1D model problem. The two new FIML approaches are applied here for the first time.

- Chapter 6 presents the results for the RANS applications. The FIML methods are demonstrated on a variety of canonical and practical RANS cases to explore the advantages/disadvantages of each approach. For the first time, the two new approaches are applied to several RANS applications.

- Chapter 7 presents the overall conclusions and summary for this effort, as well as some suggestions for future work.

- Appendix A presents inference results for a hypersonic wedge application.

- Appendix B presents inference results for the Onera M6 transonic wing.

Chapter 2: Progress in Data-Informed Turbulence Modeling

A wealth of experimental and high fidelity simulation data sets exist for turbulent flows, and naturally turbulence modelers have leveraged these data sets to either improve predictions or quantify uncertainties in turbulence models. In this chapter the various methods of leveraging data for turbulence modeling are examined. This chapter examines these previous works in two sections, based on the objective and approach of the effort. The first section reviews efforts to utilize data to calibrate models. These approaches do not attempt to modify the functional form of the underlying model, but instead calibrate closure constants such that the prediction matches observations. The next section discusses efforts to utilize data to improve predictions by modifying the functional form of the model itself.

2.1 Calibrating Turbulence Models From Data

RANS turbulence models contain a number of free parameters, known as closure constants, that can modify the behavior of the model. These closure constants were originally calibrated from simple flows such as flat plate boundary layers, and therefore many authors have attempted to recalibrate the various closure constants in order to improve predictions for specific applications. While many authors have

demonstrated success in improving predictions for various classes of flow, this simple approach to utilizing data is not likely to improve the model in general, as it does nothing to address the functional errors contained in the models themselves.

In a calibration exercise, higher fidelity data, θ is identified that the modeler expects could improve the model, $\mathcal{M}(c)$. Free parameters in the model, c, are identified by the modeler, preferably parameters that have considerable uncertainty or have violated assumptions for the chosen application [40]. The free parameters are then calibrated such that the output of the model more closely matches the chosen data after calibration.

Examples of RANS model calibration include Kato et al. who examined a single parameter of the $k-\omega$ SST turbulence model, and showed that calibrating the parameter based off of a backwards facing step would also somewhat improve predictions on other cases, such as the RAE2822 airfoil [44]. Lanzafame et al. calibrated closure constants in the $\gamma - Re_\theta - SST$ transition model to improve force predictions on wind turbine airfoils [45]. Note that this approach neglects the error present in the turbulence model itself entirely, and attempts to correct any error via manipulation of the transition model. Similarly, Mauro et al. calibrated the $\gamma - Re_\theta - SST$ transition model to improve force predictions on a S809 airfoil at a high angle of attack. This again assumes that the prediction error is due to a deficiency in the prediction of transition, and neglects the error in the turbulence model itself, as identified by the analysis of Matyushenko et al. for this airfoil and others [46, 15]. Rocha et al. examined the $k-\omega-SST$ turbulence model calibration for small wind turbines and demonstrated that modification of a closure constant

could improve performance for these cases [47, 48].

It is perhaps not surprising that modification of the closure constants can improve performance for some cases. It is, however, not likely a viable approach to improve the turbulence model in general because it inherently assumes that the functional form of the model is sufficient. Additionally, by recalibrating a new turbulence model variant is created, which can result in confusion and difficulty in assessing model accuracy [40]. Clearly, the models were originally calibrated based on certain simple flows, and recalibrating to different flows is unlikely to improve the accuracy of the model in general.

A more rigorous approach to calibration is provided by statistical inference. Bayesian analysis is a statistical inference technique that allows the calculation of posterior distributions of the parameter given the observations. In this approach both the data, θ, and the parameters of the model, c, have associated probability distributions. $P[c]$ is termed the prior distribution and represents the probability of the model without any observations. Given observations, θ, information about the probability of the model being correct can be determined. Typically this involves sampling methods such as Markov Chain Monte-Carlo (MCMC). Formally, the likelihood, $P[\theta|c]$, is the probability of the model being consistent with the observations which can be found through MCMC sampling. From Bayes' theorem the posterior probability, the probability of the parameters given the observations, $P[c|\theta]$, is proportional to the likelihood times the prior (2.1).

$$P[c|\theta] \propto P[\theta|c] \times P[c] \tag{2.1}$$

The posterior distribution gives statistical insight that can be used to calibrate the model (given by the values of the parameters that maximize the posterior probability distribution), along with the uncertainty of those values[1].

Several researchers have leveraged Bayesian techniques to recalibrate turbulence models. Ray et al. utilized Bayesian calibration techniques to calibrate closure constants of the $k - \varepsilon$ model, to improve predictions for a jet in crossflow. Another Bayesian calibration study was performed by Guillas et al., based on data for a street canyon. Of note, the best parameters identified by Guillas et al. varied substantially from those identified by Ray et al. [54, 53]. Oliver and Moser utilized DNS solutions of channel flow to perform uncertainty quantification and calibration of four common RANS models [55]. Papadimitriou and Papadimitriou used DNS solutions of flow over a backwards facing step to find posterior distributions of closure constants of the Spalart Allmaras model [56]. Edeling et al. used mean velocity profiles of flat plate boundary layers under various conditions to find posterior distributions for the closure constants of a number of popular RANS models. The results were remarkable in that the posterior probabilities differed substantially depending on the experimental data considered, despite the simple flat-plate application [57]. In summary, Edeling states: "These results suggest that there is no single best choice of turbulence model or closure coefficients, and no obvious way to

[1]This differs from traditional uncertainty quantification where the uncertainties are simply propagated through the model and to estimate the effect on the output. While this can be valuable information, it does not require observations. A good example of this manner of (data-free) analysis for the Spalart-Allmaras turbulence model is given by Schaefer et al. [49, 50, 51, 52]

choose an appropriate model a priori [57]."

2.2 Improving Turbulence Models with Data and Machine Learning

In this section, previous efforts to leverage data and machine learning to either replace model components with machine learned functions, or to augment the model to improve model accuracy. These approaches modify the functional form of the model itself to correct for errors in the functional form, as opposed to calibration exercises which simply modify the closure constants of the model and do not address errors in the functional form. The resulting data informed model is given by $\widetilde{\mathcal{M}} = \mathcal{M}(\theta)$. The general process these approaches utilize is first to identify higher fidelity data, θ, that could improve a deficient model, $\mathcal{M}$. A discrepancy is introduced, δ, to the model and is found. Machine learning is then applied to allow for the generalization of δ to additional cases.

The use of data to make predictions can be grouped into three separate overall approaches. The first is to simply use the data to directly make predictions. In this approach there is no model and the data is used directly. This, in general, is not practical as it would either require an extreme amount of data of perfect quality, or very simple physics such that modeling is not necessary. This is certainly not appropriate for RANS modeling, as turbulence is an extremely complex physical process, and there is certainly not enough data available to make predictions for all required cases.

The second category of utilizing data to inform predictions does involve mod-

eling. In this approach the higher fidelity data is injected directly into the model. While appealing in its simplicity, this approach is also not appropriate for RANS modeling. The higher fidelity data (typically from experiments, DNS, or LES) contain real, physical, quantities. The RANS models however, make use of *modeled* quantities that rely on some empiricism and have been carefully calibrated when the model was created. There is, therefore, a lack of consistency between the data injected into the model and the modeled quantities that will degrade the performance of the data-augmented prediction. An investigation by Raiesi et al. exemplifies the pitfalls of this approach. In order to evaluate and examine the underlying assumptions of various RANS models Raiesi et al. computed exact turbulent quantities from LES and DNS solutions. These exact quantities were then injected into the model and the performance compared to the baseline model. None of the turbulence models were improved by using the exact quantities and most showed inferior performance [58]. Attempts to inject exact quantities into turbulence models have not resulted in enhanced insights or improved models [59]. Even though the modeled quantities represent real, physical quantities, the models have been calibrated such that if exact quantities are injected in a portion of the model the predictive capability is reduced.

The third approach is appropriate for RANS modeling and other complex physics-based models. This approach is characterized by an inverse problem by which useful information is generated from the data. The data is inconsistent with the modeling environment. As noted, the data contains real, physical quantities, but RANS models contain numerous assumptions, calibrated quantities, and model

variables that are not consistent with the data itself. Therefore, to resolve this inconsistency, an inverse problem is formulated and solved by which information is generated that is consistent with the modeling environment. Note that in order to make predictions for new cases a model must be trained on this information to create an augmented model. This offline learning step introduces an additional opportunity for inconsistency, as the training introduces additional inconsistency between the information (from the inverse problem) and the resulting augmented model.

Emory et al. utilized an eigen-decomposition of the anisotropy tensor to decompose the Reynolds stress tensor and perturb the shape of the Reynolds stresses by modifying the eigenvalues. Eddy viscosity models typically only produce Reynolds stress tensor shapes in a very small region of the realizable range due to the Boussinesq assumption. Thus, by perturbing the shape of the Reynolds stresses the structural uncertainty of eddy viscosity models can be measured. This data-free uncertainty quantification has been applied to a variety of applications [60, 61], and has also been implemented in the SU2 package [62, 63].

To incorporate inferred Reynolds stress shape error from data, Xiao et al. utilized a sampling method to infer Reynolds stress anisotropy discrepancies from higher fidelity simulations. This discrepancy was spatially correlated $\delta(x)$ and the information was not able to be applied to a dissimilar application. Wang et al. built on this work and utilized machine learning methods (random forests) to build a function to reproduce the discrepancy from mean flow features, η. The resulting discrepancy function, $\delta(\eta)$, was then shown to improve RANS predictions on other

cases (aside from the case it was built from). Wu et al. used a similar Bayesian sampling method to infer posterior distributions of the model discrepancy, and then applied those inferences to improve predictions on similar flows in a predictive setting [36, 64, 66, 65]. Edeling et al. utilized two additional transport equations that perturbed the Reynolds stress anisotropy from the eddy viscosity model baseline. In the data-driven approach of this method Bayesian inference was used to infer the coefficients of these transport equations, such that the prediction of the model more closely matched the data [67]. Again, sampling methods were used, and the inference was not learned such that it could be applied in a predictive setting.

In an alternative approach to sampling methods, which can be prohibitively expensive for RANS applications, the inversion procedure was implemented by Wang and Dow in order to infer eddy viscosity fields from DNS data. A cost function was formulated that measured the distance of the RANS velocity field to the average velocity in the DNS solution. The eddy viscosity was treated as a parameter to be optimized, and the cost function was minimized, such that the optimal eddy viscosity was found that resulted in the RANS solution that most accurately matched the DNS solution [68]. This approach, however, completely neglected existing eddy viscosity models and is therefore not a complete methodology for improving them. Duraisamy et al. pioneered and developed the Field Inversion and Machine Learning (FIML) approach. This procedure utilizes a similar inverse problem; however, unlike the work of Wang and Dow the design variable is a discrepancy function introduced into the model. The inverse procedure finds the optimal discrepancy to minimize the distance between the output of the model and the data. This discrepancy

function is initially spatially correlated ($\delta(x)$), but then machine learning methods are utilized to create a discrepancy function from the flow features ($\delta(x) \rightarrow \delta(\eta)$) [40, 69, 43, 13, 70, 41, 71, 72, 73]. Additionally, the methodology was implemented in a Bayesian framework such that posterior distributions could be determined along with the optimal correction [41, 39]. A variety of applications have been demonstrated, including 2D airfoils and shock boundary layer interactions. Additionally, Singh et al. presented a methodology for incorporating information from a number of cases into a single machine learned model augmentation. It was shown that an augmentation built off of inverse information from a 2D S809 airfoil improved predictions on similar airfoils, demonstrating exciting generalization capabilities of the approach [72, 13]. It was demonstrated that the FIML augmentations are portable, and the augmentation was implemented and demonstrated on a separate prediction code than the implementation used for the inversion [13]. Parish and Duraisamy demonstrated the FIML approach on a simple model problem that will also be utilized in this work [39].

2.3 Summary

This chapter reviewed recent efforts by researchers to utilize higher fidelity data to improve turbulence models. The first section discussed efforts to calibrate models. In this approach the functional form of the model is not modified, but the various closure constants of the model are tuned in order to match data. There are numerous examples of successfully tuning models to match experimental or high

fidelity simulation data; however, it was noted that this method is unlikely to correct RANS turbulence models in general. RANS models were developed with simplifying assumptions, and calibrated based on simple flows such as flat plate boundary layers. Therefore, for more complicated applications, like flow over an airfoil, it is certain that the primary source of error in the model is the functional form of the model itself due to violated assumptions. Therefore, methods that do not seek to modify the functional form of the model are unlikely to show substantial improvement outside of the applications considered in the calibration process.

The next section discussed efforts to produce data-augmented turbulence models. The discussion was delineated by three approaches. The first, taking data and directly making predictions is the most simple but only appropriate for applications with exceedingly simple physics, or large quantities of perfect data. The second approach involves taking higher fidelity data and directly applying that data to the model. For RANS modeling, some researchers have demonstrated computing quantities from DNS or LES simulations and injecting those quantities directly into RANS models. The predictions are typically worse in this approach as there is an inconsistency between the real, physical quantities and the modeled quantities that have been calibrated in the model. The FIML approach addresses this inconsistency, by generating model-consistent information from higher fidelity data. This information can then be learned by a machine learning model and used in an augmented predictive model. This is the framework for the Field Inversion and Machine Learning approach, and the successes of this approach were briefly surveyed. The next chapter continues the FIML discussion, beginning with details about the framework

developed by previous researchers, and continuing with the new approaches that build on this body of work.

Chapter 3: Field Inversion and Machine Learning With Embedded Neural Network Training Development

3.1 Overview

This chapter introduces the field inversion and machine learning (FIML) framework. First, the methodology pioneered by Duraisamy et al. and utilized by previous researchers (referred to as FIML-Classic here) is discussed in detail. The motivation for this approach is discussed, as well as the procedure for incorporating information from multiple cases. To enable a discussion of the framework, a brief introduction to neural networks is also provided. Subsequent sections introduce two new methods proposed and developed for the first time in this effort: 1. FIML-Embedded, where the backpropagation algorithm was embedded into the solver, and 2. FIML-Direct, where the weights of the neural network are trained directly. The advantages and potential drawbacks of both approaches are discussed, as well as procedures to incorporate information from multiple cases. Specifically for FIML-Direct, a new approach to performing the inversion on numerous cases simultaneously is discussed, which provides an exciting path for increased generalization and regularization over the classic approach.

3.2 Field Inversion and Machine Learning: FIML-Classic

The Field Inversion and Machine Learning process was pioneered by Duraisamy et al. and has been shown to be a flexible framework to derive optimal RANS correction fields. This approach has been successfully applied to a variety of RANS problems [40, 43, 69, 13, 70, 41, 39]. Additionally, by casting the model in a Bayesian framework, posterior distributions can be computed. The discussion of the FIML process below will be restricted to the specific application in the current work, and in a non-Bayesian setting.

First an objective function (J) is defined to represent how closely the RANS solution matches the data. This cost function could take many forms, but typically J takes the form of a squared error of the model prediction (k_m) and the higher fidelity data (k_d). To differentiate the objective functions for different methods, we denote the objective function for FIML-Classic as J_c. For FIML-Classic and the cases used in the present work the objective function takes the following form:

$$\text{FIML-Classic:} \qquad J_c(\beta) = \|k_d - k_m(\beta)\|_2^2 + \lambda\|\beta - 1.0\|_2^2. \tag{3.1}$$

The first part of the objective function (J_c) represents the misfit between the current design and the higher fidelity data. The second term is the regularization term, and penalizes corrections far away from the prior (baseline) design and prevents corrections that are unnecessarily large. The regularization constant (λ) is set to a small value. With analogies to Bayesian inversion, the value of this constant has an intuitive meaning, and represents the confidence in the prior solution vs the

confidence in the posterior.

The inversion procedure takes the form of a minimization (optimization) problem to minimize the objective function: $\beta(x) = \arg\min_{\beta}(J_c)$. Following the inversion, the optimum correction field (β) is obtained. Note the optimal β is very useful to the modeler, as it shows the correction required to the model to match the data. If, however, the modeler wishes to incorporate this information that can be used to make predictions, a machine learned model, in this case feed-forward neural networks, is then trained so that the correction can be reproduced from the features (η) which are an appropriate selection of variables from the model. Previous works showed the capability of this approach, including the demonstration that the resulting data-augmented model can improve predictions in applications not in the training set, and that the augmented model is portable [13].

Conceptually, the FIML process is illustrated in Figure 3.1.

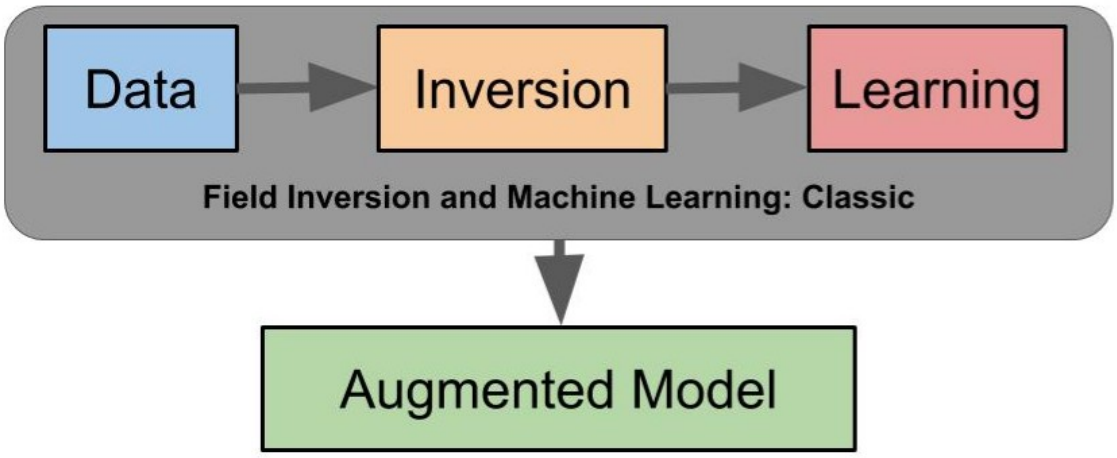

Figure 3.1: Chart of FIML-Classic procedure illustrating learning separate from inversion process.

In Figure 3.1 the data box represents the selection of appropriate higher fidelity data that the modeler has identified to improve the model. Examples of relevant data for RANS applications could include skin friction, surface pressure distributions, heat flux distributions, etc. The FIML process utilizes this data to produce an augmented model that better matches the chosen data. The inversion box for FIML-Classic represents the determination of an optimal $\beta(x)$ to minimize J_c. Adjoint methods efficiently compute the required gradients to employ gradient based optimization methods to solve the minimization problem. The learning box then represents the process of extracting features (η) from the solution to the inversion step, and training a machine learning model that can produce the correction $\beta(\eta)$ from the features. At the end of the FIML process, the model has been augmented with a machine learned model on-line so that the correction is applied to the model without the need to perform the expensive off-line inversion process.

Note that the offline training process introduces an inconsistency into the FIML-Classic approach. The inverse correction distribution is the solution of the optimization problem $\min_\beta(J_c(\beta))$ which will give the optimal[1] spatial distribution for the correction ($\beta(x)$). Regardless of the machine learning method to produce the function $\beta(\eta)$ there will always be residual training error as there is no guarantee that there is an algorithm that can produce $\beta(\eta)$ exactly. For neural networks, a large

[1]The solution to $\min_\beta(J_c(\beta))$ will be a local optimum. Typically due to the cost of evaluating $J_c(\beta)$ gradient based optimization methods are employed for this minimization and therefore the minimization will be susceptible to converging to local optima instead of the preferred global optimum.

neural network (a large number of nodes or hidden layers) may be required to reduce training error, but this network may be too costly to compute. Additionally, overly large neural networks suffer from *overtraining*, where the network represents the training data accurately but exhibits poor generalization to cases not in the training set. This training error is not accounted for in the inversion and therefore the evaluation of $\beta(\eta)$ will not perfectly reproduce the optimal distribution found in the inversion. In practice, this training error is significant and results in a degradation of the performance of the resulting augmented model.

Ideally, we want a model augmentation that improves the model prediction for a variety of cases. To accomplish this the machine learning model must be trained on data that sufficiently represents the entire feature space of interest. In this way, a neural network trained on sufficient data will be interpolating on the data it was trained on when asked to improve the prediction on a case not considered in the training set. Because there are features and a correction at every point in the computational domain in the FIML-Classic approach, there is a relatively large quantity of training data available per case in the RANS application. However, to cover the feature space in non-trivial problems it is required to train the neural network using information from multiple data sets.

In the FIML-Classic information learned from multiple cases is incorporated in a relatively straightforward manner. First the cases of interest are identified and higher fidelity data for each case (i) identified that the modeler expects will improve the model (k_d^i). Then the inversion is performed on each case individually, and the features η_k^i and optimal corrections β_k^i are compiled from each case into a single

training dataset $\{[\eta_k^1, \beta_k^1], [\eta_k^2, \beta_k^2], [\eta_k^i, \beta_k^i]\}$. The subscript k denotes that there is a feature and correction defined at every point in the computational domain. Note that the design variables used in the inversion are different for each case, which prevents us from performing the inversion on all cases simultaneously. The chosen neural network is then trained on the information from all cases in the training set. This generates a model augmentation that can reproduce the correction distribution from the features (training converts $\beta(x_j) \rightarrow \beta(\eta)$). This augmentation can be applied to cases not considered in the training set (*holdout* cases). Graphically, this is represented in Figure 3.2.

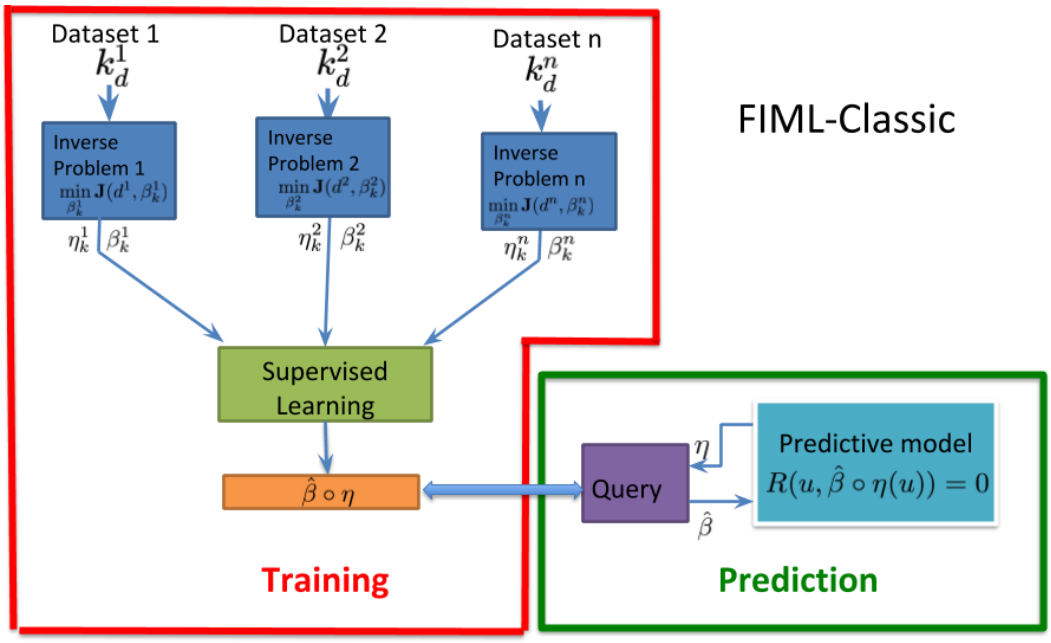

Figure 3.2: FIML-Classic Procedure for Producing Model Augmentations Incorporating Information Learned from Multiple Inversions.

3.2.1 Neural Networks for Regression

In order to better discuss the FIML framework a brief introduction to the numerical structure of neural networks is presented here, with much more amplifying detail in Chapter 4. For the FIML problem the authors utilized a fully connected, feed forward multilayer perceptron with a hyperbolic tangent activation function on the hidden layers. The input to the neural network is some subset of the flow variables (the features) at a node in the computational domain, and the output is the training correction at that node. In a fully connected feed forward neural network the input to the j-th neuron (x_j) is a linear combination of the output of the (i) neurons in the previous layer. So that:

$$x_j = \sum_i y_i w_{ji} \tag{3.2}$$

The output of node j is then computed using the activation function. For a hyperbolic tangent activation function:

$$y_j(x_j) = \tanh(x_j) = \frac{1 - e^{-2x_j}}{1 + e^{-2x_j}} \tag{3.3}$$

The output layer in this case is a single linear ($y_j = x_j$) neuron and the output of this neuron is the output of the neural network. The input layer has a number of neurons equal to the number of inputs to the neural network (number of features). Following the notation of Dreyfus, a single hidden layer network is given by (3.4) [33]:

$$y(x,w) = \sum_{i=1}^{N_c} \left[w_{N_c+1,i} \tanh \left(\sum_{j=0}^{n} w_{i,j} x_j \right) \right] + w_{N_c+1} \tag{3.4}$$

Where N_c is the number of hidden nodes and n is the number of inputs. Note that the neural network output depends on two variables: the features ($\eta = x$ for the input layer) and the weights (w). The weights are determined in the training process. Typically the data required for the training process is assembled prior to training, and subsequently the weights are obtained by training the network using the data via the backpropagation technique [74] or another algorithm. Following training the weights are fixed so that the neural network is a function of the features.

In backpropagation, the weights are first initialized to small random values and then iteratively updated to minimize a loss function, in this case the loss function was chosen to be the sum squared error over N training samples:

$$SSE = \frac{1}{2} \sum_{k=1}^{N} (y(w) - y_{T_k})^2, \tag{3.5}$$

where w are the weights of the neural network and y is the output of the neural network. y_T is the target output of the network (the output the neural network will reproduce if well trained).

The derivative of the loss function with respect to the weights is then efficiently computed using reverse mode differentiation, and the weights are updated in the steepest descent direction as indicated by the gradients. This algorithm is then repeated until the loss function is small enough or stops improving. Full documentation of the backpropagation algorithm is given by Rumelhart et al. [74], and the

algorithm is discussed in detail in Chapter 4.

3.3 Field Inversion and Machine Learning with Embedded Backpropagation: FIML-Embedded

A new approach was developed and applied for the first time in this thesis, termed FIML-Embedded, which trains the neural network by backpropagation simultaneously with the model such that the features are converging concurrently with the physics solver (and neural network inputs/features). FIML-Embedded involves the simultaneous solution of two minimization problems with different design variables. The first is the neural network cost function (3.5) which solves the minimization problem $\min_w(SSE(w))$ by backpropagation. The second problem is the minimization of the FIML objective function ($\min_{\beta_T}(J_e)$) using the training correction (β_T) as the design variable:

$$\text{FIML-Embedded:} \qquad J_e(\beta_T) = \|k_d - k_m(\beta_T)\|_2^2 + \lambda\|\beta_T - 1.0\|_2^2. \tag{3.6}$$

Note that both minimization problems must sufficiently converge, which results in added complexity to the inversion procedure as it is sometimes difficult to sufficiently minimize equation (3.5) throughout the inversion. If $\min_w(SSE(w))$ sufficiently converges however, the nonlinearity of $\beta(w)$ due to the neural network is effectively hidden from $\min_{\beta_T}(J_e(\beta_T))$ resulting in easier convergence of that minimization algorithm. Thus FIML-Embedded has the potential to be much more efficient than FIML-Direct (presented next section), especially for large and/or multi-

layered neural networks. Conceptually, the FIML-Embedded (and FIML-Direct) process is illustrated in Figure 3.3.

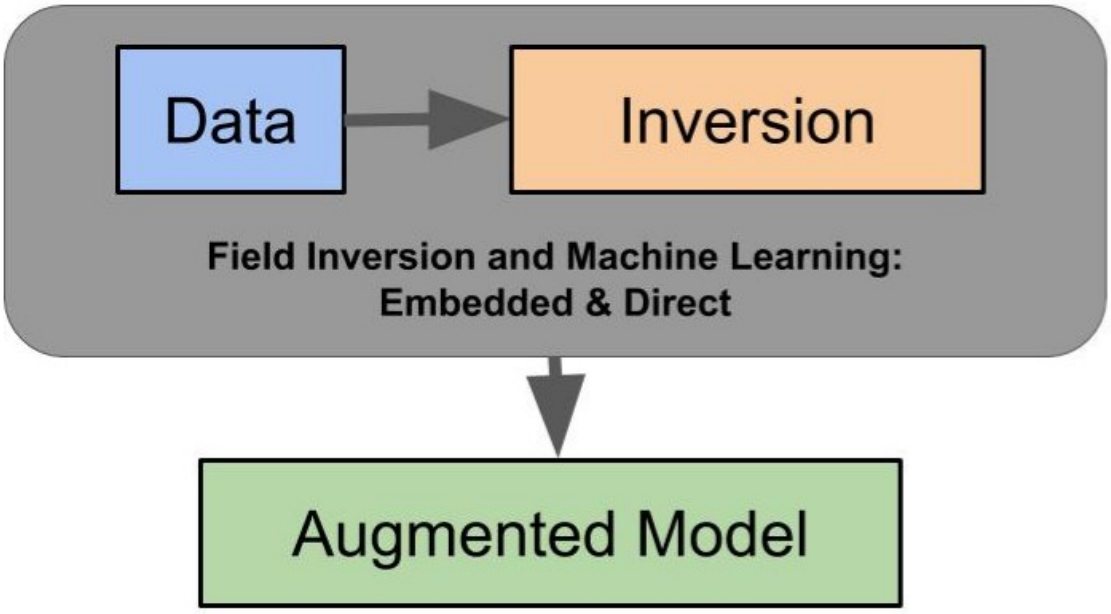

Figure 3.3: Chart of FIML-Embedded and FIML-Direct procedure illustrating that offline learning is not required for these approaches.

In Figure 3.3, the data and augmented model blocks are unchanged from the FIML-Classic chart (Figure 3.1). Now, however, the offline learning step is no longer required, because machine learning is included in the inversion step. For FIML-Embedded, the inversion block now represents the simultaneous solution of two minimization problems: $\min_w(SSE(w))$ and $\min_{\beta_T}(J_e(\beta_T))$. For both FIML-Embedded and FIML-Direct, the solution to the inversion procedure is the augmented model so offline training is no longer necessary.

Another potential drawback of the FIML-Embedded procedure is that is it is not as natural to account for the incorporation of multiple cases into a single

neural network. In order to provide more generalization to the resulting model augmentation it will be required to consider multiple cases. In other words, the model augmentation will need to incorporate information from multiple (perhaps numerous) inversions. For FIML-Classic the training data is simply assembled, and the learning is performed offline separate from the inversion. For FIML-Embedded the learning is performed in the inversion. This remains an unresolved question: how to best account for information from multiple inversions in a single model augmentation for FIML-Embedded. There are several possibilities that remain unexplored. There are concerns about all of these unexplored options:

Unexplored Option 1. Perform FIML-Embedded on many cases, which will effectively regularize the resulting β distributions by accounting for the limitations of the chosen machine learning algorithm for each case. Then, similar to the FIML-Classic procedure, assemble the features from each inversion and train a new model augmentation that incorporates all the inversions together. This approach is not ideal, as it is unsatisfying to simply throw out the augmentations for each case and train another. Additionally, while it would be guaranteed that each β distribution could be learned by the chosen machine learning algorithm, there would be no guarantee that the set of inverse solutions could be learned despite the increased regularization.

Unexplored Option 2. Perform FIML-Embedded inversions sequentially, with each inversion incorporating the inverse information learned from the previous inversions. This approach would be similar to Option 1 however it could potentially alleviate the issue with guaranteeing that the set of inverse solutions could be

learned. However, each subsequent inversion would become more and more costly as the training algorithm would be learning on more and more data. Therefore, this approach would likely become intractable as the number of cases grows.

Unexplored Option 3. Construct an ensemble model from each individual augmentation. Ensemble models for decision trees are utilized effectively in the random forests machine learning algorithm. The individual model augmentations could potentially be incorporated into an ensemble-averaged model that would incorporate the information learned from each inversion. This would be the most straightforward approach however it would still result in a lack of consistency between the inversion environment and the model augmentation as the augmentation would not reproduce precisely the same correction distribution.

Unexplored Option 4. Perform training offline (like FIML-Classic) but compute the derivative of the training algorithm so that the limitations of that algorithm are accounted for in the inversion. Option 4 is perhaps the most interesting of the unexplored FIML-Embedded options to incorporate multiple cases. This approach would account for the limitations in the inversion and it would also enable the *simultaneous* inversion of multiple cases (similar to FIML-Direct, discussed in the next section). In this approach the inversion would be performed using the current output of the neural network $\beta(\eta, w)$ to find that case's objective function j^i, and the derivative of the cost function would be computed via adjoint methods to find $dj^i/d\beta$. Then, the training data would be assembled from each case and a single neural network trained on the information from each case. The derivative of the training algorithm, $d\beta_T/d\beta$, could be computed by finite difference methods. Then,

the chain rule would be applied to each case to provide the required derivative, $dJ_e/d\beta_T$, in order to minimize the composite cost function $J_e = \sum_i^n j^i(\beta_T)$ where n is the total number of cases. There are two potential issues with this unexplored approach. The first is that the neural network is not defined for the first iteration, so the solver would be required to use $\beta = 1.0$ at the beginning of the inversion process. The second, and more complicating, is that adjoint methods or any reverse mode differentiation approach would not be advantageous for the neural network derivative computation $d\beta/d\beta_T$ because the total number of β_T elements is equivalent to the number of β elements. Still, this is an embarrassingly parallel problem that could be easily parallelized. However, for large neural networks, a large number of cases, or large computational domains this derivative computation could become costly.

3.4 Field Inversion and Machine Learning with Direct Training: FIML-Direct

Another new FIML approach was proposed, developed, and applied for the first time in this thesis: FIML-Direct. For FIML-Direct, we begin with a straightforward modification to the cost function. Now the design variables are the weights of the neural network, and the output of the neural network is the correction. So in (3.4), $y = \beta$. The minimization problem is now $\min_w(J_d)$

$$\text{FIML-Direct:} \quad J_d(w) = \|k_d - k_m(w)\|_2^2 + \lambda\|\beta(w) - 1.0\|_2^2. \tag{3.7}$$

The correction, β, is determined by computing the output of the neural network by applying (3.4). The immediate disadvantage of FIML-Direct is that the nonlinearity of the neural network is now fully exposed to the optimization algorithm ($\min_w(J_d)$). Typically neural networks are trained with static data and the computation of the gradient by backpropagation is very efficient and tens of thousands of backpropagation iterations can be quickly performed. With FIML-Direct, each gradient computation is far more costly because $k_m(w)$ must first be evaluated by a CFD simulation, followed by the evaluation of $dJ_d/d(w_j)$ by adjoint methods.

Despite the potentially challenging minimization, FIML-Direct has many advantages over FIML-Classic and FIML-Embedded. First, for turbulent correction applications, the use of FIML-Direct results in substantially fewer design variables for typical neural networks as the number of weights is substantially fewer than the number of points in the computational domain. Second, the optimum solution for the weights is independent of the computational grid and each particular flow solution. We are seeking one neural network that corrects the model for all applications of interest, and therefore the weights should be the same between cases. Because the weights are independent of the case and computational grid they can be applied to dissimilar cases (such as different airfoils).

To consider the simultaneous inversion of multiple cases, we first identify data that can improve each case, i, for n cases. The cost function is then defined as the composite cost function as the sum of the cost function from each case. So if each case, i, has objective function j^i then the total cost function is given by $J_d = \sum_{i=1}^{n} j^i$. Note there is no requirement that each j^i have the same functional form. This is

important as the type of data that improves case 1 may not be the same type that improves case 2. So dataset 1 could use the experimental lift coefficient, and dataset 2 could utilize the heat flux distribution. Both cases would have different functional forms for j^i, but the composite derivative would still simply be the sum of each case's derivative (Equation (3.8)). Additionally, the regularization constant, λ^i can be tuned for each case. So for a case where the model performs well and the modeler has high confidence in the result the λ^i can be increased for that case.

$$J_d = \sum_{i=1}^{n} j^i \tag{3.8}$$

$$\frac{dJ_d}{dw} = \sum_{i=1}^{n} \frac{dj^i(w)}{dw} \tag{3.9}$$

The inversion is then carried out as in the single case inversion, where we minimize the composite cost function with respect to the weights: $\min_w(J_d(w))$. The resulting augmentation is then optimal considering all the data (all k_d^i) and considering the limitations of the chosen machine learning algorithm. There is also no inconsistency between the inversion environment and the model augmentation as the same augmentation found in the inversion is used for prediction. Conceptually, this approach is illustrated in Figure 3.4.

3.5 Computational Cost

Modern CFD simulations are often extremely computationally expensive. The FIML-Classic approach requires an inversion utilizing a gradient based optimization procedure that results in a series of CFD simulations (direct and adjoint) and natu-

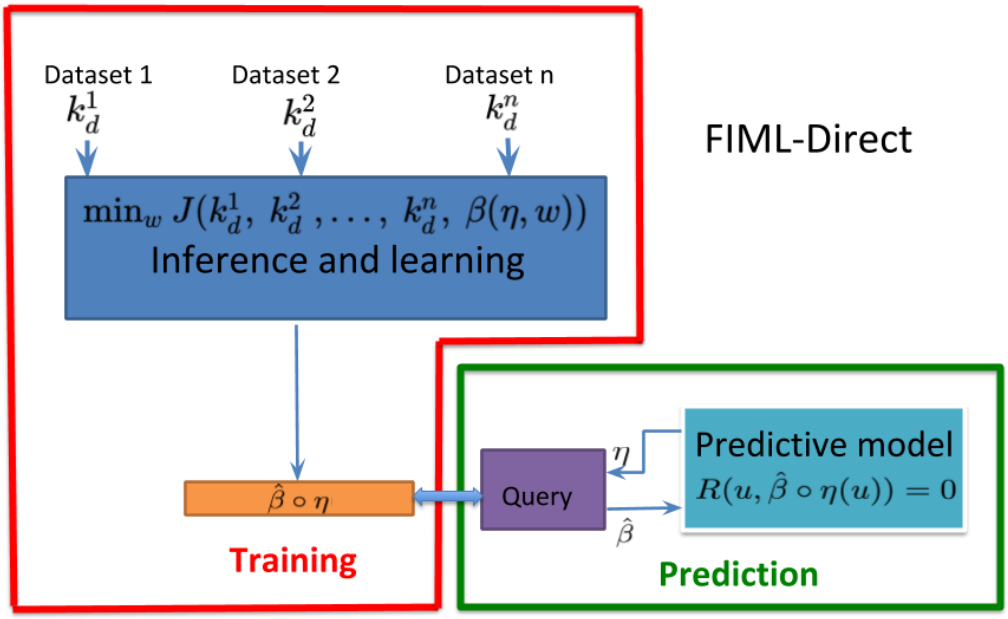

Figure 3.4: FIML-Direct procedure for simultaneous inversion of multiple cases to produce single augmentation.

rally for large cases this method can become quite expensive. We have proposed two novel approaches and here we discuss the computational cost of the new approaches and methods of mitigating the cost through efficient computation on modern high performance computing (HPC) architectures.

The minimization ($\min(J)$) for the FIML approaches could be performed using minimization algorithms that do not require gradient information, such as Powell's method [75], but in practice more evaluations are required for gradient-free approaches. Since efficiently computed gradient information is available via adjoint methods all FIML applications in this work utilize gradient-based minimization approaches[2]. Therefore, for the FIML-Classic inversion ($\min_\beta(J_c(\beta))$) each new design

[2]For applications where evaluations of the cost function are efficient, gradient-free genetic algorithms could be utilized [76], at dramatically increased expense, but without gradient information

requires at a minimum an updated objective function and gradient. So each iteration costs approximately two CFD simulations (Direct+Adjoint). The required number of optimizer iterations required to substantially minimize the cost function is application dependent but is typically on the order of 10-20 iterations corresponding to 20-40 simulations. Thus the inversion cost per case is 20-40 times the cost of the baseline SA model. The inversion must be performed on enough cases to cover the feature space of the problem of interest, so for n cases the cost is $20n - 40n$ simulations. The cost to perform the offline training is not trivial, but for the applications here was far less than the cost of the baseline CFD. Following offline training, the model has been augmented and the only increased cost to utilize the augmented model is the extra cost of the floating point operations in the neural network; which is trivial compared to solving the extra transport equation associated with the baseline SA model.

The added cost for FIML-Embedded is associated with the additional operations that must be performed per iteration inside the direct and adjoint solvers. Each iteration the features at each node are updated and scaled, the forward propagation is performed to obtain the current output of the neural network ($\beta(\eta, w)$), and then the backpropagation algorithm is performed to update the weights to minimize the error between β and β_T. Exact timing studies have not been performed to determine precisely how much these operations cost; however, in the applications considered in this work the additional computational cost was not noticeable. However, it was noted that under some conditions the backpropagation algorithm would not easily

and with an increased ability to locate the global optimum.

converge and would ultimately affect the convergence of the flow solver, requiring increased flow solver iterations to sufficiently converge. Improved robustness of the FIML-Embedded approach has been identified as an area deserving of future work. No offline training is required following the inversion as the training/augmentation have already been performed.

For FIML-Direct the added cost is primarily just the cost of assembling the features every solver iteration and forward propagating to obtain the current β. The added solver cost for this operation was not noticeable, and the convergence issues observed for FIML-Embedded were not an issue for the FIML-Direct applications in this thesis. The number of optimizer iterations required for FIML-Direct was comparable to that required for FIML-Classic, for the cases considered. Therefore, the computational cost of FIML-Direct is comparable to FIML-Classic except that there is no need for offline training following the inversion as the learning and augmentation has already been performed.

For FIML-Direct with multiple cases, again enough cases must be included to sufficiently cover the feature space for the application of interest. For n cases, each optimizer iteration for FIML-Direct will require $2n$ simulations ($n \times$ Direct + $n \times$ Adjoint) to evaluate the cost function $J_d(w) = \sum_{i=1}^{n} j_d^i(w)$ and its gradient $dJ_d(w)/dw = \sum_{i=1}^{n} dj_d^i(w)/dw$. So again, the cost is roughly comparable to FIML-Direct, with $20n - 40n$ simulations required. For FIML-Direct there is also the potential issue that the inversions are coupled such that all cases must be considered at the same time during the inversion process. This is not an issue on modern HPC systems as the computation of the composite cost function and the composite

gradient is an embarrassingly parallel problem. Additionally, as will be shown it is also not necessary to compute the full gradient, as a randomly selected subset of the cases can be considered a batch for that iteration, and only the partial gradient of the cases in the batch are computed for that iteration. The weights are updated with the partial gradient and a new batch is selected. This is similar to stochastic gradient descent and is demonstrated in Chapter 5.

3.6 Summary

The Field Inversion and Machine Learning methodology was introduced. The FIML-Classic procedure developed by Duraisamy and co-workers was discussed in detail including the methodology to incorporate numerous datasets (cases) into a single model augmentation. The overall advantage of the FIML procedure versus other machine learning approaches was discussed. Other machine learning approaches for physics based modeling suffer from an inconsistency between the modeling environment and the prediction environment. By using the field inversion approaches presented in this Chapter, information is generated by solving the the inverse problem that is consistent with the prediction/modeling environment. A potential shortcoming of the FIML-Classic procedure was also presented, in that it requires offline learning or training separate from the inversion. As discussed, this introduces an opportunity for inconsistency, as perfect learning is not possible in practice. This error is not accounted for in the inversion and therefore there is an inconsistency between the inversion and the prediction environment, albeit much improved over

not performing the inversion at all.

Two new FIML approaches, both developed and applied for the first time in this thesis, were introduced in this Chapter. Both new approaches address the identified deficiency in the FIML-Classic procedure. The first, FIML-Embedded was described, where the backpropagation algorithm is embedded into the model solver so that the limitations of the training algorithm would be implicitly considered in the inversion. Additionally, following inversion the neural network has been trained and the model augmented. The potential shortfalls of this approach were also discussed. The first being that there is considerable added complexity over the FIML-Classic and FIML-Direct approaches. The second is that it is not straightforward to include information learned from multiple cases. Potential methods to overcome this shortfall were proposed, but have not been developed and applied.

The second new FIML procedure, FIML-Direct, was introduced in this chapter. In this approach the weights are trained directly (without backpropagation or offline training) in the inversion. In this new approach the weights are considered the design variables of the cost function minimization. The advantages of this approach were discussed. For FIML-Direct the design variables are consistent across multiple cases. Therefore the inversion can be performed on numerous cases simultaneously. Thus, the machine learning model limitations are considered and the resulting augmentation incorporates limitations from numerous cases. Both of which are expected to improve the regularization and generalization of the resulting augmentation.

Finally, the computational cost of all three FIML approaches was discussed.

It was shown that the two new approaches do not present an unacceptable increase in computational cost and should be straightforward to perform on modern HPC architectures.

Chapter 4: Numerical Methodology

This chapter discusses the numerical methodology utilized in this effort. First the Navier Stokes equations are presented, followed by a discussion about the RANS approach. The turbulence models used are then presented. Neural networks, discussed only briefly in Chapter 3, are discussed in detail in this chapter. Methods to perform feature selection and scaling, procedures critical to machine learning, are also presented. The 1D heat equation model problem is presented, which provides a much simpler environment to examine the FIML methodology compared to the RANS applications. Finally, details on the numerical implementation of FIML-Classic, FIML-Embedded, and FIML-Direct are presented for both the model problem and the RANS applications.

4.1 Governing Equations

The governing equations for fluid flows are given by the Navier Stokes equations. Their derivation relies on the continuum hypothesis of an ideal gas and the assumption of a Newtonian fluid. From fundamental laws of conservation of mass, momentum, and energy the Navier Stokes equations on a rectilinear coordinate frame (x, y, z) are given by Equation 4.1 [77]

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} = J \tag{4.1}$$

$$U = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho(e + \frac{V^2}{2}) \end{Bmatrix} \tag{4.2}$$

$$F = \begin{Bmatrix} \rho u \\ \rho u^2 + p - \tau_{xx} \\ \rho v u - \tau_{xy} \\ \rho w u - \tau_{xz} \\ \rho(e + \frac{V^2}{2})u + pu - k\frac{\partial T}{\partial x} - u\tau_{xx} - v\tau_{xy} - w\tau_{xz} \end{Bmatrix} \tag{4.3}$$

$$G = \begin{Bmatrix} \rho v \\ \rho u v - \tau_{yx} \\ \rho v^2 + p - \tau_{yy} \\ \rho w v - \tau_{yz} \\ \rho(e + \frac{V^2}{2})v + pv - k\frac{\partial T}{\partial y} - u\tau_{yx} - v\tau_{yy} - w\tau_{yz} \end{Bmatrix} \tag{4.4}$$

$$H = \begin{Bmatrix} \rho w \\ \rho uw - \tau_{zx} \\ \rho vw - \tau_{zy} \\ \rho w^2 + p - \tau_{zz} \\ \rho(e + \frac{V^2}{2})w + pw - k\frac{\partial T}{\partial z} - u\tau_{zx} - v\tau_{zy} - w\tau_{zz} \end{Bmatrix} \tag{4.5}$$

$$J = \begin{Bmatrix} 0 \\ \rho f_x \\ \rho f_y \\ \rho f_z \\ \rho(uf_x + vf_y + wf_z) + \rho\dot{q} \end{Bmatrix} \tag{4.6}$$

The column vectors F, G, and H are the flux terms and J is a source term. U is referred to as a solution vector because typically in CFD solvers the elements of U are the variables that are solved for numerically [77]. The last element in U is the density times the total energy per unit mass E = internal energy + kinetic energy = $e + V^2/2$. The first equation (row) in (4.1) is derived from conservation of mass (or continuity), the next three from conservation of momentum in all three spatial dimensions, and the final from conservation of energy. ρ is the fluid density, $\{u, v, w\}$ are the components of velocity in the x, y, and z directions respectively, $V = \sqrt{u^2 + v^2 + w^2}$, e is the internal energy, and p is the pressure. The terms in J represent body forces and is zero if body forces are negligible. As a shorthand the spatial dimensions x, y, z will also be referred to as x_j where $i = 1, 2, 3$ in the

following discussion.

The $k\partial T/\partial x_j$ terms are the thermal conduction terms arising from temperature gradients in the fluid, where k is the thermal conductivity. Because k is typically constant the following relation is used for the thermal conductivity, where $Pr = 0.72$ is the Prandtl number for air [78]:

$$k = c_p \frac{\mu}{Pr} \tag{4.7}$$

τ_{ij} is the viscous stress tensor. For Newtonian fluids the shear stress is proportional to the time rate of strain and then the viscous stress tensor becomes:

$$\tau_{ij} = \mu \left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right] + \lambda \frac{\partial u_k}{x_k} \delta_{ij}, \quad \delta_{ij} = 1 \quad \text{if} \quad i = j; \quad \delta_{ij} = 0 \quad \text{if} \quad i \neq j \tag{4.8}$$

Where μ is the molecular viscosity of the fluid that can be easily computed as a function of temperature by Sutherland's law [79] given by Equation (4.9) along with the constants for air (4.10).

$$\mu = \mu_{ref} \left(\frac{T}{T_{ref}} \right) \frac{T_{ref} + S}{T + S} \tag{4.9}$$

$$\mu_{ref} = 1.716 \times 10^{-5} \quad \left[\frac{kg}{ms} \right], \quad T_{ref} = 273.15 \quad [K], \quad S = 110.4 \quad [K] \tag{4.10}$$

λ is the bulk viscosity and the Stokes hypothesis given by Equation (4.11). This hypothesis is widely used and accepted but has not been proven [77], although there is no evidence that contradicts it [78].

$$\lambda = -\frac{2}{3}\mu \tag{4.11}$$

For an ideal gas we have the following familiar relationship between pressure, density, and temperature:

$$P = \rho R T \tag{4.12}$$

Were R is the ideal gas constant. For a calorically perfect gas we have the following relationship with the specific heat at constant volume (c_v) and constant volume (c_p):

$$c_v = \frac{R}{\gamma - 1}, \quad c_p = \frac{\gamma R}{\gamma - 1} \tag{4.13}$$

We can also rewrite e in terms of the temperature:

$$e = c_v T \tag{4.14}$$

Thus we ultimately have seven unknowns $\{\rho, \rho u, \rho v, \rho w, \rho E, p, T\}$ that can be solved for with the five Navier Stokes equations (4.1), the equation of state (4.12) and equation (4.14) [77, 78].

For all cases considered in this work a calorically perfect gas is assumed where the ratio of specific heats ($\gamma = c_p/c_v$) is assumed constant $\gamma = 1.4$. For all of the applications with the exception of the hypersonic wedge this assumption is valid. The extreme temperature range encountered in the hypersonic wedge case somewhat challenges the calorically perfect gas assumption; however, even in this application

the variation in γ is expected to be small (about 5%).

4.2 Reynolds-Averaged Navier-Stokes Equations

Solving the Navier Stokes equations (4.1) is typically feasible for laminar flows, but far more involved and costly for turbulent flows. The issue arises from the need to accurately resolve all the small turbulent eddies, leading to the need for extremely spatially resolved grids as compared to the laminar case. This requirement becomes more and more challenging with increasing Reynolds number, as Blazek presents, the number of grid points to sufficiently resolve turbulent flows grows proportional to $Re^{9/4}$ and the solution time proportional to Re^3 [78]. With Reynolds numbers for typical engineering applications easily in the millions, the RANS equations are typically solved due to the difficulty of accurately resolving all turbulent length scales for DNS.

For the RANS equations the velocity components are decomposed into a mean component and a fluctuating component (4.15). Substituting Equation (4.15) into (4.1) the resulting RANS equations are almost identical, but now for the mean quantities, and with the addition of the *Reynolds stresses* in the momentum equations resulting from the fact that the mean of the product of two fluctuating components is not zero.

$$u_i = \bar{u_i} + u_i', \quad p = \bar{p} + p', \quad \rho = \bar{\rho} + \rho', \quad T = \bar{T} + T' \tag{4.15}$$

The Reynolds stress is given in (4.16), and represents the mean momentum

fluxes induced by the turbulence [80]:

$$\tau_{ij}^{R} = -\rho\overline{u_i' u_j'} \tag{4.16}$$

Boussinesq proposed the eddy viscosity concept which models the effect of turbulence as an additional effective viscosity, referred to as the eddy viscosity, ν_t. This reasoning leads to the Boussinesq hypothesis:

$$\tau_{ij}^{R} = -\rho\overline{u_i' u_j'} = \rho\nu_t \left[\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right] - \frac{\rho}{3}\overline{u_k' u_k'}\delta_{ij} \tag{4.17}$$

The second term in (4.17) is often rewritten in terms of the turbulent kinetic energy, $k = (1/2)\overline{u_k' u_k'}$. Turbulence models provide the missing terms in (4.17), namely ν_t and k.

4.2.1 The Spalart-Allmaras Turbulence Model

The Spalart-Allmaras (SA) turbulence model [17] is a popular one equation eddy viscosity model used to provide closure to the RANS equations for turbulent flow simulations. This model is chosen as the baseline to introduce the embedded model discrepancy.

The transport equation for the SA model is given below:

$$\frac{\partial \hat{v}}{\partial t} + u_j \frac{\partial \hat{v}}{\partial x_j} = c_{b1}(1 - f_{t2})\hat{S}\hat{v} - \left[c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2}\right]\left(\frac{\hat{v}}{d}\right)^2 + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_j}\left((v + \hat{v})\frac{\partial \hat{v}}{\partial x_j}\right) + c_{b2}\frac{\partial \hat{v}}{\partial x_i}\frac{\partial \hat{v}}{\partial x_i}\right] \tag{4.18}$$

The terms on the right hand side of (4.18) are referred to as the production (P), destruction (D), and diffusion terms of the SA model such that:

$$P = c_{b1}(1 - f_{t2})\hat{S}\hat{v} \tag{4.19}$$

$$D = \left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right]\left(\frac{\hat{v}}{d}\right)^2 \tag{4.20}$$

The eddy viscosity is given by $\mu_t = \rho\hat{v}f_{v1}$.

Additionally,

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \quad ; \quad \chi = \frac{\hat{v}}{v} \quad ; \quad c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma} \tag{4.21}$$

$$f_w = g\left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6}\right] \quad ; \quad g = r + c_{w2}(r^6 - r) \quad ; \quad r = \min\left[\frac{\hat{v}}{\hat{S}\kappa^2 d^2}, 10.0\right] \tag{4.22}$$

The f_{t2} term was originally implemented to model transition but is often neglected as was done in this application ($f_{t2} = 0$). This variant of the Spalart Allmaras turbulence model is referred to as the "SA-noft2" model.

The Spalart-Allmaras model uses some empiricism and a number of closure constants ($\kappa,\ \sigma,\ c_{bi},\ c_{wi},\ c_{ti}$) which have their own uncertainty. Of particular note is the equilibrium assumption in the log layer of the boundary layer. This assumes that the terms on the right hand side of equation (4.18) are in balance. In other words, the production, destruction, and diffusion terms all cancel each other in the log layer. This assumption is violated in strong adverse pressure gradients. Additionally, note

that the wall distance, d, is utilized as a length scale in both the production and destruction terms. This is natural for the problems for which the Spalart Allmaras was calibrated, namely attached boundary layers. However, this length scale is not appropriate far from the wall where d becomes large.

Recently, Schaefer et al. have performed sensitivity analysis on the SA model closure coefficients. The authors concluded that the uncertainty in the von Karman constant (κ) and the turbulent Prandtl number (σ) were the primary sources of uncertainty quantities of interest such as pressure coefficient (C_p) [51, 50]. Other efforts seeking to recalibrate the closure constants of turbulence models were discussed in detail in Chapter 2. While these calibration efforts consider parametric uncertainty, it is intuitive to expect that a larger source of uncertainty is a result of the structural form of the model.

4.3 The Spalart-Allmaras 1-equation BC Transitional Model (SA-BC)

For many cases of common engineering interest the effects of transition are significant. For cases where transition was anticipated to be significant the transition model proposed by Cakmakcioglu et al. was used [18]. Note that a correction to the original model is documented on the turbulence modeling resource website maintained by Rumsey, and additional comments and discussion of the model are also documented there [25]. In order to model laminar to turbulent transition an intermittency function, γ_{BC}, is applied to the production term of the SA-noft2 tur-

bulence model. The intermittency function approaches 0 for laminar regions and approaches 1.0 where the flow is predicted to be turbulent. In this way there is minimal turbulent production in the laminar regions and the standard SA-noft2 model in turbulent regions. The intermittency is computed as follows:

$$\gamma_{BC} = 1 - \exp(-\sqrt{\text{Term}_1} - \sqrt{\text{Term}_2}) \tag{4.23}$$

$$\text{Term}_1 = \frac{\max(Re_\theta - Re_{\theta_C}, 0.0)}{\chi_1 Re_{\theta_C}}, \quad \text{Term}_2 = \frac{\max(\nu_{BC} - \chi_2), 0.0}{\chi_2} \tag{4.24}$$

$$Re_\theta = \frac{Re_\nu}{2.193}, \quad Re_\nu = \frac{\rho d^2}{\mu}\Omega \tag{4.25}$$

$$Re_{\theta_C} = 803.73(Tu_\infty + 0.6067)^{-1.027} \tag{4.26}$$

$$\nu_{BC} = \frac{\nu_t}{Ud}, \quad \chi_1 = 0.002, \quad \chi_2 = \frac{5.0}{Re} \tag{4.27}$$

The variables χ_1 and χ_2 were calibrated to a flat plate test case and set to $\chi_1 = 0.002$, and $\chi_2 = 5.0/Re$, where Re is the freestream Reynolds number. Re_ν, Re_θ, $Re_{\theta c}$ are the vorticity, momentum thickness, and experimental transition onset critical momentum thickness Reynolds numbers respectively. Tu_∞ is the freestream turbulence intensity.

Like other RANS transition models [25, 81, 82, 3], the SA-BC model predicts transition by comparing the estimated momentum thickness Reynolds number to the

estimated critical momentum thickness Reynolds number. Unlike other transition models, the SA-BC model is an algebraic model meaning that there are no additional transport equations (no additional transported variables).

4.3.1 Strain Adaptive Spalart Allmaras Model: SA-SALSA

The SA-SALSA model was implemented in SU2 in order to better model hypersonic shock boundary layer interactions. This formulation of the SA model restates several terms with the intention of improving its predictive capability under non-equilibrium conditions [83]. This SA variant has shown an ability to better predict the separation length in hypersonic SWTBLI [26]. The modifications required for the SALSA variant of the SA model versus the previous model are documented below [25, 83, 17]. The $\hat{S}$ term is redefined (4.28) along with r (4.29):

$$\hat{S} = S^* \left[\frac{1}{\chi} + f_{v1} \right] \tag{4.28}$$

$$r = 1.6 \tanh \left[0.7 \sqrt{\frac{\rho_0}{\rho}} \left(\frac{\hat{\nu}}{\hat{S} \kappa^2 d^2} \right) \right] \tag{4.29}$$

ρ_0 is the freestream stagnation density, S^* is defined $S^* = \sqrt{2 S'_{ij} S'_{ij}}$, and S'_{ij} is similar (but not equivalent to) the vorticity (4.30):

$$S'_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{1}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \tag{4.30}$$

The term c_{b1} is replaced by c'_{b1} which is not constant and defined by (4.31):

$$c'_{b1} = 0.1355\sqrt{\Gamma}, \quad \Gamma = \min[1.25, \max(\gamma, 0.75)], \quad \gamma = \max(\alpha_1, \alpha_2) \tag{4.31}$$

The additional α terms are then defined as:

$$\alpha_1 = \left[1.1\left(\frac{\hat{\nu}}{S^*\kappa^2 d^2}\right)\right]^{0.65} \qquad \alpha_2 = \max\left[0, 1 - \tanh\left(\frac{\chi}{68}\right)\right]^{0.65} \tag{4.32}$$

Finally, unlike the standard SA model, the turbulent kinetic energy term in the Boussinesq equation (4.17) is not neglected, and k is given by:

$$k = \frac{\nu_t S^*}{\sqrt{0.09}} \tag{4.33}$$

4.4 Neural Network Structure and Training

At this point neural networks have been discussed in some detail but only a cursory discussion of the structure of the network or algorithm to determine the neural network weights has been presented. This section adds some detail to the neural network structure used in this thesis, and then presents the algorithm for training (determining the weights) of the network.

The first task in training a network is to assemble the training dataset. For all the applications in this thesis the output of the neural network is the correction to the model, β. The inputs of the neural network are the features (η). For the RANS applications the features are a subset of the local flow variables discussed in detail in later sections. Assembly of the training data is a critical step in neural network

training, as the features must have a strong functional relationship to the desired output or any attempt to train a network will be futile. Numerical methodology, feature selection, and feature scaling are discussed in detail in the next section.

Prior to learning, the neural network structure must be selected. In this application we exclusively use fully connected feed-forward neural networks but many other topologies and variations exist [33, 84, 85]. For this topology the primary parameters that define the topology (so-called "hyperparameters") are the selection of the number of layers, the number of nodes for each layer, and the choice of "activation function" on the nodes. A neural network is made up of "neurons[1]". A single neuron is illustrated in Figure 4.1 [84].

The neurons are placed in layers, such that the inputs are placed in the first "input layer". The subsequent layers are termed the hidden layers. If there are multiple hidden layers then the network can also be referred to as a "deep" network, although there is some disagreement about how many layers constitute a deep network. The final or output layer of the neural network constructs the output of the network. In all applications in the present work the network only has a single output, β. Each layer also utilizes bias nodes which are treated as additional inputs

[1]The term "neural network" is somewhat of a confusing nomenclature as the analogy of a neural network neuron to a neuron in a biological organism is poor at best. Dreyfus has an excellent discussion of this inconsistency and states clearly that the progress made in neural networks is due to understanding the mathematics and statistics and not due to an understanding of how the human brain functions [33]. Therefore the present work will not attempt to make any comparisons of the human brain to the functionality of a neural network, however the nomenclature of neural networks is well established and will be used here.

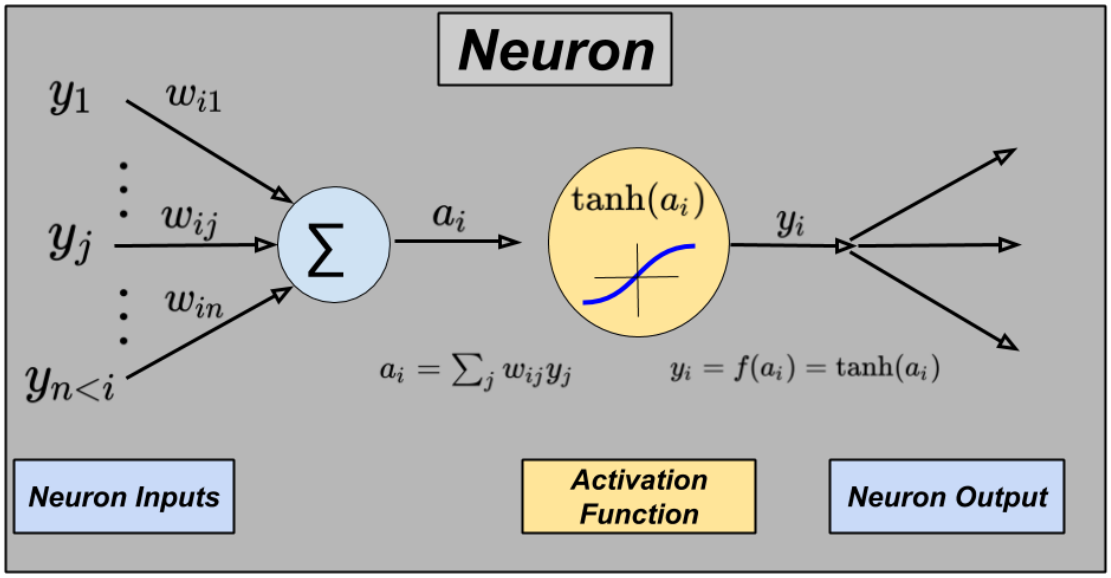

Figure 4.1: Illustration of a single neuron in a neural network.

(input of 1.0) to the neurons (each with an associated weight). This topology is illustrated in Figure 4.2.

Note that neural networks are often displayed graphically to illustrate the connections between neurons, but they can also be easily written as equations. For Figure 4.1 the inputs from the previous layer are denoted $y_j,\ j = 1, \ldots, n$ where n is the number of nodes in the previous layer. If the previous layer is the input layer, then n is the number of inputs and the y_j are the inputs. Each input to the neuron has a weight, w_{ij}, where i is the layer of the neuron. The weight multiplies its input, and then all the weighted inputs are summed to create the input to the neuron. The neuron applies an "activation function" to create the output of the neuron. There are many activation functions to choose from (Rectified Linear Unit (ReLU), Radial Basis Functions (RBF), etc.) but for all applications a hyperbolic tangent (sigmoid) activation function ($y_i = \tanh(a_i)$) was employed in the hidden layers. As is typical

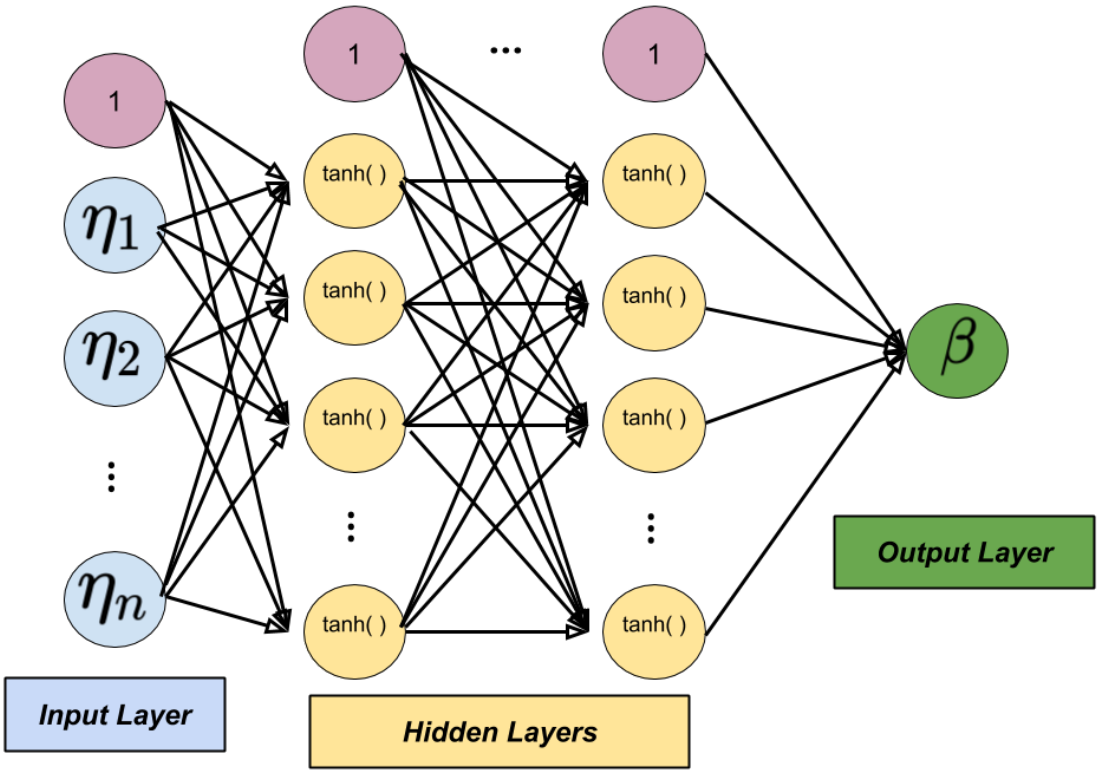

Figure 4.2: Illustration of chosen neural network structure.

for unbounded regression problems the activation function for the output layer was chosen to be linear ($y_i = a_i$). The activation function introduces nonlinearity, so for this neural network structure the output of the neural network (β) is nonlinear with respect to the inputs of the network, but linear to the output of the final hidden layer. Equation 3.4 is equivalent to the topology shown in Figure 4.2 for a single hidden layer.

The process of determining the weights of the neural network is referred to as "training" or "learning". The broad category of learning implemented in this thesis is called "supervised" learning which simply means that the model is being fitted

to assembled (known) training data in order to replicate the outputs as closely as possible given the same inputs. Now that the training data has been assembled and the structure of the network defined, the next task is to finally determine the weights of the network. Each line in Figure 4.1 represents a weight so naturally the number of weights grows substantially with the number of nodes and hidden layers. Typical neural networks have hundreds of weights that must be trained, and often there is a large quantity of training data, so the computational efficiency of the training algorithm is important.

To efficiently train the weights of the neural network gradient descent methods are typically used, in order to minimize a cost function that measures the distance of the current output of the neural network to the desired output defined by the training data. Gradient based methods require a gradient, and the gradient is efficiently computed by a reverse mode differentiation algorithm referred to as "backpropagation". This algorithm was first presented by Rumelhart et al. [74]; here we largely follow the notation of Reed and Marks [84], with minor changes to notation to reflect the more specific application in the present work. First, we define the cost function as the sum squared error in Equation 4.34. For P training samples (input/output pairs in the training set), E is the total sum squared error of the current output of the network ($\hat{\beta}$) compared to the target output in the training set, β.

$$E = \frac{1}{2}\sum_{p=1}^{P}(\beta_p - \hat{\beta}_p)^2 \tag{4.34}$$

As a reverse mode differentiation algorithm, the derivative computation starts

with the output of the neural network by computing the value δ_i for each node, i, as defined in Equation (4.35).

$$\delta_i = \frac{\partial E_p}{\partial a_i} = \frac{\partial E_p}{\partial y_i}\frac{\partial y_i}{\partial a_i} \tag{4.35}$$

For the output node, $\partial y_i/\partial a_i$ evaluates to 1 because this a regression problem using a linear activation function on the output node. The derivative of the error term for this pattern with respect to the output of the output node ($\partial E_p/\partial y_i$) is then simply the derivative of the cost function. For hidden nodes the term $\partial y_i/\partial a_i$ is evaluated by using the information from nodes upstream. For hidden nodes:

$$\delta_i = \frac{\partial E_p}{\partial a_i} = \sum_k \frac{\partial E_p}{\partial a_k}\frac{\partial a_k}{\partial a_i} = \sum_k \delta_k \frac{\partial a_k}{\partial a_i} \tag{4.36}$$

Then to evaluate $\partial a_k/\partial a_i$:

$$\frac{\partial a_k}{\partial a_i} = \begin{cases} f_i' \sum_k w_{ki}\delta_k & \text{If Node i Connects to Node k} \\ 0 & \text{Otherwise} \end{cases} \tag{4.37}$$

The term f_i' is the derivative of the activation function that captures the nonlinearity of the node at its current activation (a_k) value. For the $\tanh(a_i)$ we have $f_i' = 1 - f^2(a_i)$. So we ultimately arrive at the following algorithm to compute the δ in Equation (4.38). Note that the algorithm begins at the output layer and works backwards, such that the nodes k are in the next layer closer. So the output layer provides the k nodes to the last hidden layer (i nodes). The last hidden layer then provides the k nodes to the i nodes and so on.

$$\delta_i = \begin{cases} -(\beta_{pi} - \hat{\beta}_{pi}) & \text{For Output Node} \\ f_i' \sum_k w_{ki}\delta_k & \text{For Hidden Nodes} \end{cases} \tag{4.38}$$

Finally the derivative of the cost function with respect to each weight is given by Equation (4.39):

$$\frac{\partial E_p}{\partial w_{ij}} = \delta_i y_j \tag{4.39}$$

Summing Equation (4.39) across all patterns in the training set gives the required total error derivative with respect to that weight ($\partial E_p / \partial w_{ij}$). Note that while this algorithm is relatively easy to implement, it is possible to bypass the sometimes tricky coding of the backpropagation algorithm by using automatic differentiation [86]. This can greatly simplify the construction of large and/or complicated network connection structures by computing the gradient in Equation (4.39) with little or no user burden to implement the derivative computation. It also eliminates the requirement that the activation function has an analytical derivative. Additionally, the derivative computation could also be estimated by finite differences if the associated increase in cost and decrease in accuracy is tolerable (if using a real-valued step). In the present effort, the backpropagation algorithm was implemented as written above due to the relative simplicity of the neural networks implemented.

With the gradient information, the weights can be updated towards a descent direction in order to minimize the current error. This can be as simple as a constant step ("learning rate") in the steepest descent direction ($-\partial E_p / \partial w$). The update rule for the weights would then be given by Equation 4.40:

$$w^{(n+1)} = w^{(n)} - \alpha \frac{\partial E_p}{\partial w} \tag{4.40}$$

Where n is the current training iteration, the learning rate (α) is a small number (often 0.001 or smaller), and w is a vector containing the weights with corresponding gradient vector ($\partial E_p / \partial w$). For the first iteration the weights are initialized to small random numbers. The update algorithm given by Equation 4.40 is equivalent to steepest descent with a fixed line search step. This is trivial to implement, but will exhibit poor performance compared to more advanced algorithms such as BFGS or its limited memory variant: "L-BFGS-B" [87, 88]. Both BFGS variants construct estimates of the Hessian, giving superior performance over the steepest descent algorithm for minimization problems in high dimensions, such as neural network training, which will have hundreds of weights/dimensions even for small networks.

Additionally, we hope that neural network exhibits good generalization, so that that the model is capable of not only replicating the data it was trained on, but also cases that were not available in the training set (holdout data). The machine learning term for a network that performs well on the patterns in the training set but poorly on other patterns, is an "overtrained" network. The terminology refers to the phenomenon where a neural network exhibits good generalization for earlier training iterations, but diminished generalization to holdout data later in the training evolution. Therefore, it is standard practice to periodically test the network on holdout data in order to quantify the performance on cases not in the training

set. Overtraining is particularly troublesome for large networks and therefore it is advantageous to not over-parameterize the network, by using the smallest network that provides sufficient accuracy to the training data. Therefore, the modeler choice for the neural network topology is certainly not trivial and is often problem dependent. The selection of the number of nodes and layers is often set based on previous work on similar problems, by trial and error, or by a parametric study where numerous networks are constructed with varying hyperparameters and the resulting performance is examined.

4.4.1 Feature Selection and Scaling

Feature selection and scaling is an important consideration in the construction of useful neural networks that is sometimes neglected or overlooked. The features must have a strong functional relationship to the desired output of the model. For complex physics-based applications, such as RANS modeling, it is clear that there is a long list of variables that may influence the output, and it is not readily apparent which features are the most important. As noted, over-parameterizing the problem can result in over-training, so for generalization (and to minimize complexity) it is important to try to use the fewest features as possible that still accurately define the output. The process of determining the features is termed *feature selection*. Additionally, certain methods of training machine learning algorithms are sensitive to outliers and large differences in scales in the features. Neural networks trained by gradient descent methods are one such training algorithm, in which poor feature

scaling can result in networks that are difficult or impossible to sufficiently train.

Because of the importance of feature selection, a substantial effort was made to identify suitable features for the RANS applications in the current work. The numerical methodologies employed for feature selection are documented individually below.

Feature Selection Method: *Correlation Coefficients*

Pearson correlation coefficients[2] are a measure of the linear correlation between two quantities. The correlation coefficient between n samples of two variables x and y is computed by first finding the sample covariance between two variables and then dividing by $\sqrt{\sigma_X^2 \sigma_Y^2}$. The correlation coefficient is therefore given by Equation (4.41) [89]:

$$\rho_{XY} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sigma_X^2 \sigma_Y^2}} \tag{4.41}$$

Correlation coefficients can help expose strong linear correlations (ρ approaches 1) or strong inverse linear correlations (ρ approaches -1). Potential features with correlation coefficients with the output near 0 may not be the best choice. Similarly if two potential features are strongly correlated with each other using only one of the two in the feature set may provide the same information as using both [3]. To further examine relationships between features two dimensional scatter plots were examined

[2] Also known as Pearson's r, the Pearson product-moment correlation coefficient, or bivariate correlation

[3] Singular value decompositions were also used on potential features to help identify if features were providing unique information

between all features and the desired output (β). The features were assembled and correlation coefficients computed using the pandas library [90], and plots generated using the seaborn library which provides an interface to matplotlib [91] that enables the visualization of large quantities of data.

Feature Selection Method: *Sequential Back Selection*

Sequential back selection is a costly method of evaluating features but is straightforward to implement and can be applied to any machine learning algorithm. The machine learning model is trained using all but one of the features and is scored. This score could be any measure of accuracy such as mean squared error or maximum absolute error, but in this application the coefficient of determination was used, given by Equation (4.42).

$$R^2(\beta, \hat{\beta}) = 1 - \frac{\sum_{i=1}^{n}(\beta_i - \hat{\beta}_i)}{\sum_{i=1}^{n}(\beta_i - \bar{\beta})} \tag{4.42}$$

Values closer to 1 indicates that additional samples will be predicted well by the model. After models that have been trained holding each feature out, the feature set with the best score is carried forward and the feature not in that set is dropped. This continues until there is only one feature or the desired number of features is met. Often models *improve* when certain features are removed. This is an indication that there are features in the set that are not providing useful information, and are complicating training by inappropriately raising the dimensionality of the problem. Eventually the score begins to decrease when additional features are removed, indicating that these features are providing useful information to the

model and should likely be retained. Note that the sequential back selection method could quickly become impractical with large quantities of training data because so many models must be trained and compared.

Additionally, two methods of feature scaling were utilized.

Feature Scaling Method: *Z-Scaling Normalization*

For the RANS applications discussed later the features are of different scales, and there are very large outliers. Often the inputs have different physical units and scales, and if not normalized the smallest-scaled features will be ignored in training [33]. To mitigate the difference in scales Z-scaling was attempted, which is simply to scale each feature to a mean of 0 and a standard deviation of 1. So for each feature, i, and pattern, p, we perform the following normalization:

$$\eta_i^p = \frac{\eta_i^p - \bar{\eta}_i}{\sigma_{\eta_i}} \tag{4.43}$$

For the applications in this thesis this scaling method demonstrated less performance than mapping to a Gaussian distribution.

Feature Transformation Method: *Map to Normal Distribution*

Extremely skewed data can be mapped to a Gaussian distribution to mitigate the negative impact of outliers on neural network training performance. Mapping to a normal distribution tends to spread out closely spaced features, and reduce the impact or spread of the outliers. Two methods of accomplishing this mapping were used. For the offline training cases in the FIML-Classic procedure (discussed in this chapter) the quantile transformer was used in the scikit-learn package [92].

The second approach was implemented for the neural networks used inside the SU2 solver. First, the min and max of each feature were computed and bins created. Using the bins, the feature's probability distribution function was approximated by computing the estimated distribution function. Typically 10 bins were used, but testing showed the algorithm was not very sensitive to the number of bins. The estimated distribution function was then transformed by the logit function, which is a close approximator to the inverse cumulative distribution function for a normal distribution. Each feature was then mapped to the transformed estimated distribution function to find its transformed value. While somewhat cumbersome and costly to iterate through all the features several times to compute the transformed distribution, it was shown that this transformation dramatically improved the neural network training for the RANS applications due to its ability to mitigate the influence of marginal outliers.

4.5 Field Inversion and Machine Learning

Numerical considerations for the FIML methods, introduced in Chapter 3, are discussed here. Two different implementations were assembled for the present work: one for the RANS applications, and the other for the 1D heat equation model problem. The heat equation numerics are considerably more simple and are discussed later in this chapter. The SU2 framework is the main focus of the following sections.

The FIML approach for both the 1D heat equation and RANS implementation

relies on three major computations. The outer-most solver is the minimization (optimization) routine. The optimizer minimizes the cost function, J, by manipulating the design variables, α. For the applications in this work the BFGS optimization routine and the limited memory BFGS variant, L-BFGS-B were used. The optimizer initializes the design variables and calls the other components to compute the current value of the objective function, $J(\alpha)$, and the gradient vector, $dJ/d\alpha$. For the RANS applications the objective function is determined by solving the RANS equations using the modified SU2 package for the problem of interest, and comparing the output of the model $k_m(\alpha)$ to the data k_d. The derivative is computed by solving the corresponding adjoint equations. The modified SU2 autodifferentiated discrete adjoint solver was used to compute the gradient. The following sections detail this numerical framework for each FIML approach.

4.5.1 FIML-Classic

For FIML-Classic, the design variables are the correction to the production term of the SA model at each point in the computational domain ($\alpha \equiv \beta$). The objective function is given by Equation (3.1). The numerical solution procedure is illustrated in Figure 4.3.

The optimizer initializes the β field[4] and passes β to the flow (direct) solver. The direct solver[5] solves the RANS equations with the turbulence model with β

[4] Typically, $\beta = 1.0$ everywhere to start, corresponding to the baseline model solution for J_c

[5] "direct solver" here refers to the model solver, that is used to evaluate the objective function. This is also known as the "primal" solver, and the terms are used interchangeably throughout this

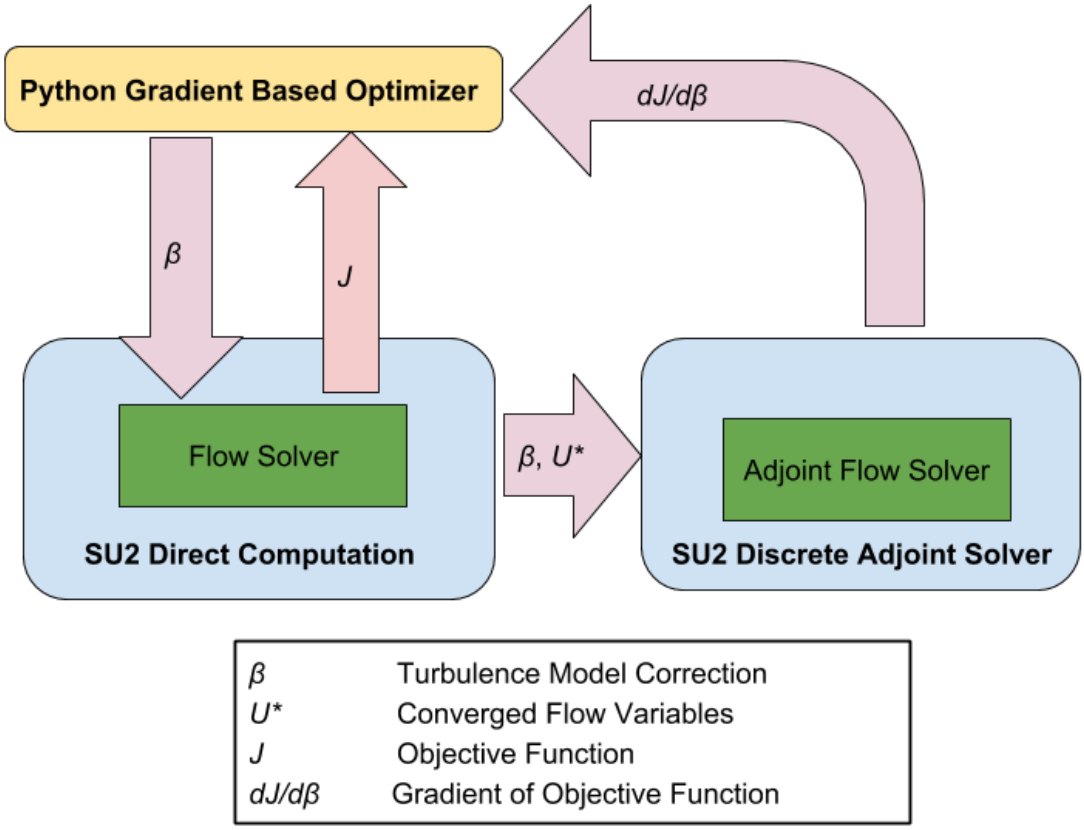

Figure 4.3: Flowchart of FIML-Classic RANS implementation in SU2 illustrating input and output of major modules.

applied to the production term. This gives the value of $k_m(\beta)$, and therefore also $J_c(\beta)$ which is returned to the optimizer. The adjoint solver requires the converged flow variables U^* from the direct solver as well as the current value of β set by the optimizer. The adjoint solver is executed and returns the gradient of the objective function with respect to the correction to the optimizer. With the current objective

thesis. For RANS applications the direct or primal solver is the flow solver used to evaluate J and obtain converged flow variables U^*. The adjoint solver refers to the solver used to obtain the gradient of the objective function, $dJ/d\alpha$, via the adjoint equations.

function value and gradient, the optimizer can then determine the search direction and update the correction field to minimize the objective function. This updated correction is passed to the flow solver, and the process is repeated until no new minimum can be found or the modeler terminates the algorithm because the objective function has been reduced sufficiently. The numerical details of both the optimizer and the adjoint solver are discussed in later sections of this chapter.

4.5.2 FIML-Embedded

For FIML-Embedded, the objective function (J_e) is given by Equation 3.6. The design variables in this case are the training correction ($\alpha \equiv \beta_T$). The backpropagation algorithm was embedded in the SU2 flow solver (embedded backpropagation). Every flow solver iteration the required features are assembled from the flow variables, are scaled, and the current values of the correction β are found at each node in the computational domain. The current output of the neural network is compared to the training correction and the weights are updated via backpropagation. Ideally, the backpropagation algorithm will converge well and the converged value of the weights w^* will replicate β_T well, so that $\beta^* \approx \beta_T$. The adjoint solver requires additional information from the direct solver for the FIML-Embedded case, and the converged values of the weights, flow variables, and β are passed to the adjoint solver. A flowchart of this method is shown in Figure 4.4.

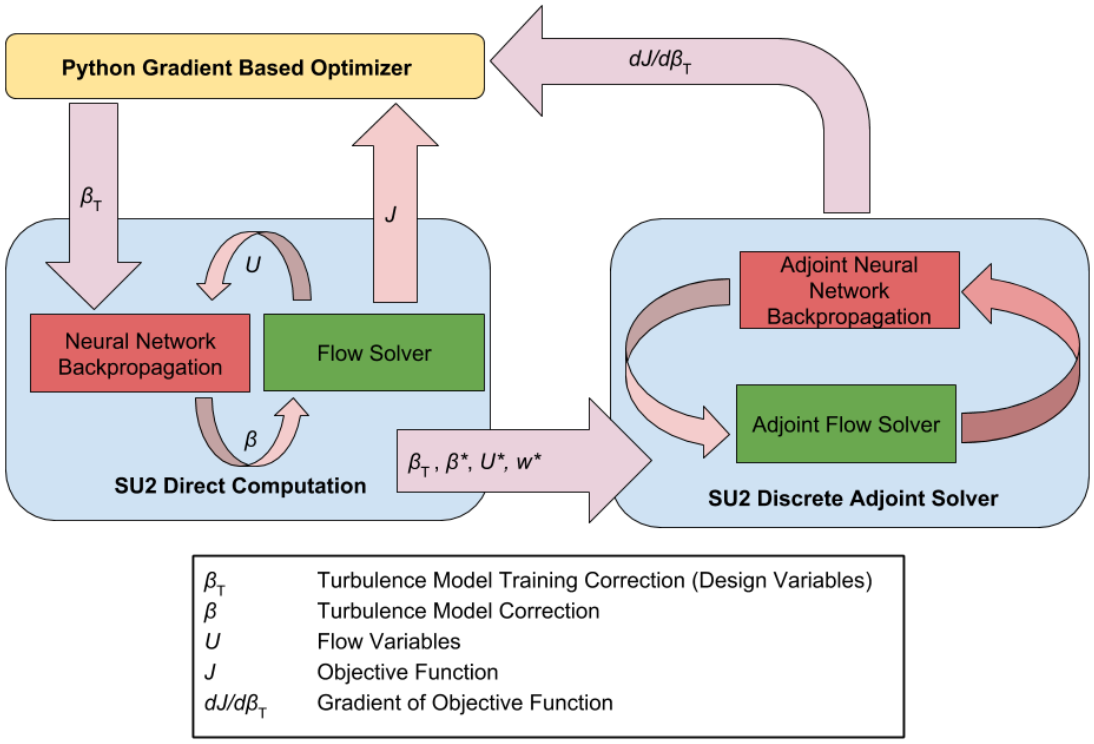

Figure 4.4: Flowchart of FIML-Embedded RANS implementation in SU2 illustrating input and output of major modules.

4.5.3 FIML-Direct

The flow structure of the FIML-Direct approach is largely similar to the FIML-Classic implementation just substituting the weights for the correction. So for FIML-Direct the objective function is given by Equation 3.7, and the weights of the neural network are the design variables ($\alpha \equiv w$). Inside the flow solver the flow features are assembled and scaled from the current values of the flow variables, and the current value of β is updated. The optimizer is therefore influencing the eddy viscosity by manipulating the weights of the neural network that augments the production

term of the SA model. The rest of the solver structure is largely similar to the FIML-Classic implementation, as illustrated in Figure 4.5.

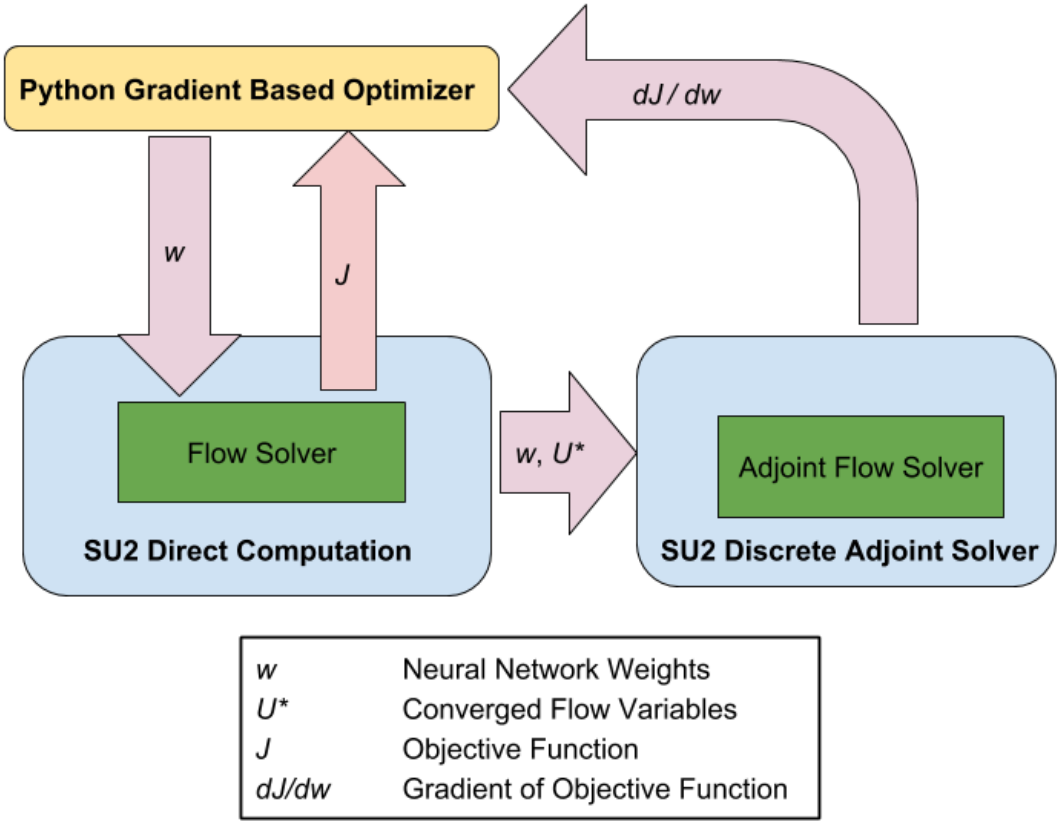

Figure 4.5: Flowchart of FIML-Direct RANS implementation in SU2 illustrating input and output of major modules.

When considering multiple cases, the flow structure is similar, except that each evaluation of the composite objective function (3.8) involves multiple flow solutions (one for each case), and each gradient evaluation involves multiple adjoint solution. Conceptually this is illustrated in Figure 4.6.

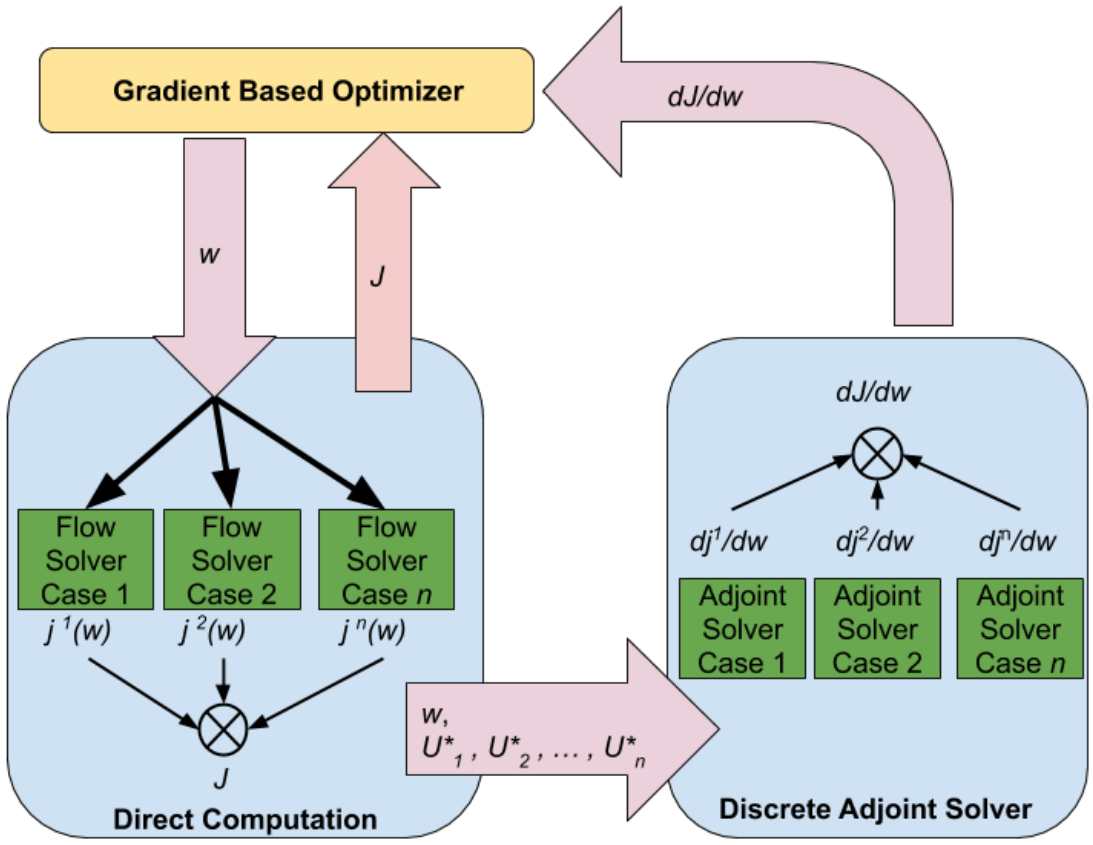

Figure 4.6: Flowchart of FIML-Direct RANS implementation for multiple cases.

4.5.4 Unconstrained Optimization By Gradient Descent

All three FIML methods require an unconstrained minimization algorithm in order to efficiently minimize the cost function, J, for each approach. As discussed for the backpropagation algorithm, it is possible to simply take a small fixed step in the steepest descent direction (defined by the negative of the gradient with respect to the design variables). This approach, however, is not efficient especially for problems such as the FIML approaches here, that are very highly dimensional. The BFGS[6]

[6]BFGS stands for four names: Broyden Fletcher Goldfarb and Shanno.

algorithm [93] is very efficient at minimizing functions with many dimensions, and most of the objective function minimizations in this thesis utilized either the BFGS algorithm or the limited memory variant (L-BFGS-B) [88] when constructing the full Hessian estimate was intractable due to the exceedingly high dimensionality of the application.

4.5.4.1 The BFGS Method

The gradient[7] of the cost function ∇J can be efficiently computed via adjoint methods. Hessian[8] information would be very valuable, if available, as efficient Newton methods could be used for the minimization of the cost function. Papadimitriou and Giannakoglou surveys different approaches for computing the Hessian for CFD applications via adjoint methods and, at best, the cost of computing the Hessian is $N + 1$ primal simulations [94, 95]. That cost is not achievable for applications with hundreds (potentially millions) of design variables. Therefore, the computational cost of computing the Hessian of J prevents its exact computation for the FIML RANS applications.

Quasi-Newton methods provide an alternative to the direct computation of the Hessian. This family of methods estimates the Hessian from the gradient evaluation at two points. The quasi-Newton condition relies on the approximation given by Equation (4.44), where J is the function to be minimized, x is a vector of design variables, k is an index indicating the current design, ∇J is the gradient of J, and

[7] Vector of first derivatives.

[8] Matrix of second derivatives.

H is the Hessian of J:

$$H_{(k+1)}(x_{k+1} - x_k) \approx \nabla J(x_{k+1}) - \nabla J(x_k) \tag{4.44}$$

The algorithm is summarized as follows. The estimate for the Hessian, B_k is initialized to the identity matrix for the first iteration only. The current search direction, p_k, is found by solving $B_k p_k = -\nabla J(x_k)$. Note that on the first iteration the search direction will be the steepest descent direction, but on subsequent iterations it will be informed by the estimate of the Hessian. If B_k were the exact Hessian we recover Newton's method. After p_k is found a line search is performed to minimize J along the search direction. This is a one-dimensional minimization problem to find $\alpha_k = \arg\min J(x_k + \alpha p_k)$. Once α_k is chosen by the line search algorithm the variable $s_k = \alpha_k p_k$ is set and the next design is defined by $x_{k+1} = x_k + \alpha_k p_k$. The variable y_k is defined as the difference between the gradient of the new design and the previous: $y_k = \nabla J(x_{k+1}) - \nabla J(x_k)$. Then the estimated Hessian can be updated with the new information as shown in Equation (4.45) [93, 87]:

$$B_{(k+1)} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k^T}{s_k^T B_k s_k} \tag{4.45}$$

Note that the Hessian matrix is a $n \times n$ matrix where n is the number of design variables (elements in x). Therefore the memory required to store H_k for problems with many design variables can be prohibitively large. For neural network training $n > 100$, and for FIML-Classic and FIML-Embedded $n > 50,000$ depending on the application. Thus for FIML-Classic and neural network training applications a

limited memory variant of BFGS was used, specifically the "L-BFGS-B" algorithm, which stores vectors containing the Hessian estimate information [88]. In this way a sparse estimate of the Hessian is used and the memory requirements are dramatically diminished. The L-BFGS-B algorithm can also incorporate design variable constraints. For the applications in this thesis design variable constraints were only used to prevent the optimizer from taking unnecessarily large line search steps and constraints were never active in the optimal designs. For the FIML results the SciPy implementations of BFGS and L-BFGS-B were utilized [96], and for offline training for FIML-Classic the L-BFGS-B implementation in the scikit-learn package to train the weights [92].

4.5.4.2 The Stochastic Gradient Descent Method

In stochastic gradient descent (SGD) method[9] is a natural extension of the steepest descent approach that is sometimes applied when multiple cases (or patterns) are being considered as a composite gradient. This is a common minimization procedure for weight updates in neural networks [84]. Instead of computing the exact gradient at each iteration only portions of the full gradient are computed, and a small step (h) in the partial gradient steepest descent direction is taken. In this effort the SGD method was implemented to minimize the composite cost function for the FIML-Direct approach, so the full cost function across n cases is given by $J = \sum_i^n j^i$. In the SGD method, $m < n$ cases are selected at random, and the

[9] also known as incremental gradient descent, and similar to "on-line" training as presented by Reed and Marks [84].

gradient is estimated by Equation 4.46:

$$\frac{dJ}{dw} \approx \sum_{l}^{m} \frac{dj^l}{dw} \tag{4.46}$$

The design variables are then updated by moving the design a small step in the steepest descent direction approximated by 4.46. Naturally, the gradient estimation will not be equivalent to the true steepest descent direction, but in some applications this is advantageous. First, it is less computationally expensive to compute the partial gradient. Second, the partial gradient introduces some randomness to the search direction, which in some applications can improve the chances of finding the true gradient and avoiding local minima [84].

4.6 Complex Step Method

Gradient descent methods require the computation of the gradient. For simple problems the gradient can often be computed analytically (and exactly). When the gradient is not available, the gradient is often estimated via finite differences. Writing the Taylor expansion of a function $F(x)$ about an arbitrary point $F(x_0)$ we have Equation 4.47.

$$F(x_0 + \Delta x) = F(x_0) + \Delta x F'(x_0) + \frac{\Delta x^2}{2} F''(x_0) - \frac{\Delta x^3}{3!} F^{(3)}(x_0) + \cdots + \frac{(\Delta x^n F(x_0)^{(n)}}{n!} \tag{4.47}$$

It is then straightforward to rearrange Equation 4.47 to obtain the first order accurate finite difference approximation for the first derivative:

$$F'(x_0) = \frac{F(x_0 + \Delta x) - F(x_0)}{\Delta x} + \mathcal{O}(\Delta x) \tag{4.48}$$

As Δx goes to 0 we recover the exact estimation of the first derivative, however, on computing systems using floating point math the accuracy is limited by floating point accuracy. Floating point numbers can represent extremely small numbers, but truncation error becomes significant when operating with large numbers. So when applying Equation (4.48) as Δx approaches 0 the numerator $(F(x_0 + \Delta x) - F(x_0))$ will also become very small, but the values of $F(x_0)$ will remain constant. If $F(x_0)$ is much larger than $F(x_0 + \Delta x) - F(x_0)$ the truncation error can be severe, and in practice there is a minimum value at which point Δx before the truncation error starts to dominate the error from neglecting higher order terms in Equation 4.47. Therefore, in practice, machine accurate estimations of the gradient are not possible with the finite difference method due to subtractive cancellation error. Note that this introduces another complication: the estimation of Δx will be dependent on the step size and the choice of Δx will be problem dependent.

Alternatively, the complex variable method (or complex step method) enables the computation of the gradient of real-valued functions to machine accuracy [97] by leveraging the additional information that is stored in floating point *complex* numbers. To compute the gradient we simply take a small complex step ($\Delta x \equiv ih$). The Taylor series expansion for this step is then:

$$F(x_0 + ih) = F(x_0) + ihF'(x_0) - \frac{1}{2}h^2F''(x_0) - \frac{1}{3!}ih^3F^{(3)}(x_0) + \cdots + \frac{(ih)^{(n)}F(x_0)^{(n)}}{n!} \quad (4.49)$$

Therefore, by adding a small complex step into any real-valued function the real part of the result is a $\mathcal{O}(h^2)$ accurate estimation of $F(x_0)$:

$$\mathrm{Re}\{F(x_0 + ih)\} = F(x_0) + \mathcal{O}(h^2) \quad (4.50)$$

The imaginary part can then then be used to estimate $F'(x_0)$ with $\mathcal{O}(h^2)$ accuracy:

$$F'(x_0) = \frac{\mathrm{Im}\{F(x_0 + ih)\}}{h} + \mathcal{O}(h^2) \quad (4.51)$$

The advantage of using a complex step over a real valued step is that the gradient of a real-valued function can be estimated to *machine accuracy*, because the small imaginary step is never added to a much larger number. This causes subtractive cancellation errors when using a real valued step and limits the smallest step size that is possible without unacceptably large round-off errors. With a complex step the subtractive cancellation error is avoided, enabling the use of extraordinarily small step sizes that can estimate the gradient so accurately that the missing terms of $\mathcal{O}(h^2)$ are below the round-off error. At this point the gradient is estimated to machine accuracy, and the result of the estimation is insensitive to the chosen step size.

The complex variable method is easily demonstrated, and we show the results of the same function used by Squire and Trapp [97]. We use the test function $f(x) = x^{9/2}$, and compare complex variable step method (4.51) with the first order forward finite difference method given by (4.48), which requires an additional function evaluation.

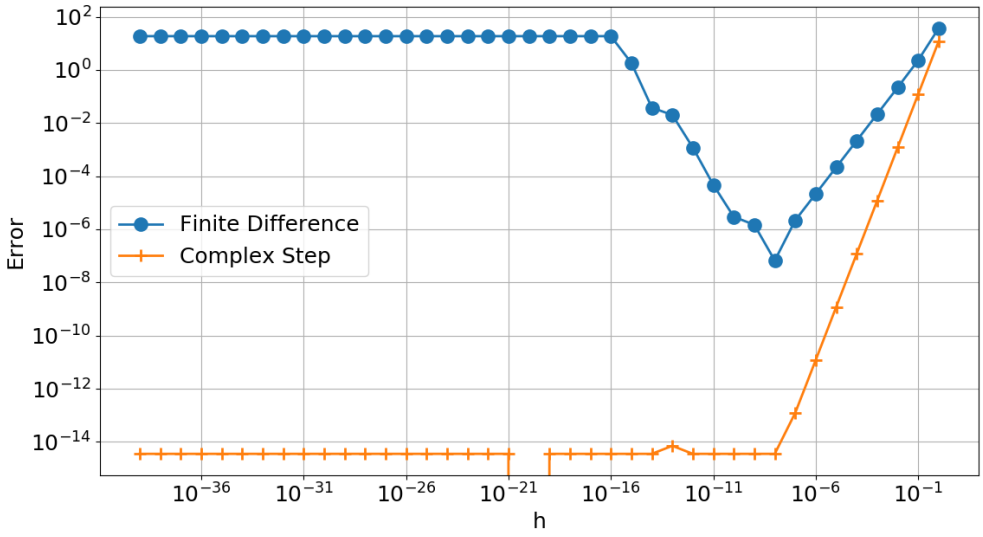

Figure 4.7: Comparison of the error as a function of step size for the finite difference method and complex variable step method.

As shown in Figure 4.7, the complex variable method has several advantages. First, the 2nd order accuracy of the complex variable method manifests as a steeper error convergence slope over the first order accurate real valued finite difference method. Most importantly, the finite difference method error never approaches a constant value, and instead grows as the step size is decreased past the point were subtractive cancellation error begins to dominate the error balance. In fact, the approximation returns 0.0 as the step size approaches zero because there is so much

subtractive cancellation error in the numerator of Equation (4.48). The complex variable method, however, does converge to a very small error at which point the gradient is estimated to machine accuracy. Note that the machine accurate gradient is returned for a wide range of step sizes indicating that the estimation is not sensitive to the step size in this region, which greatly simplifies the choice of step size.

Note that the complex variable method has two potential shortfalls: first, it only applies to real-valued functions. Second, like the finite difference method it is a forward differentiation approach. In other words, it can compute the gradient of any number of outputs for a single input. This can become costly for function with large numbers of inputs, as each element of the gradient requires a function evaluation. Despite these drawbacks, the approach is very practical for complicated and costly simulations as it is extremely easy to implement. Simply by changing all real type variables to complex type a real-valued simulation can return machine accurate gradients. The method is now widely used for a variety of physics based simulations due to its accuracy and ease of implementation [97, 98, 99, 100, 101].

4.7 Adjoint Methods

Adjoint methods allow the efficient computation of the exact gradient (to machine precision) that is required for gradient based optimization methods. Many excellent explanations of adjoint methods have been presented (Johnson, for example), and we follow the notation and explanation of Giles and Pierce here [103, 102]. Given a linear set of equations $Au = f$ where A is a matrix and f is a vector, it

is equivalent to solve the *dual form* where $A^T v = g$. It can be shown that these formulations are equivalent as shown in (4.52).

$$v^T f = v^T A u = (A^T v)^T u = g^T u \tag{4.52}$$

Now, given a set of equations that have variables, U and depend on α that satisfy:

$$N(U, \alpha) = 0 \tag{4.53}$$

For the Navier-Stokes equations the variables U would be the flow variables, but in the model problem discussed in later sections, U is equivalent to the temperature. α is a vector of design variables that we wish to modify to minimize the cost function, J, that depends on α and U. The gradient of the cost function is given by (4.54):

$$\frac{dJ}{d\alpha} = \frac{\partial J}{\partial U}\frac{dU}{d\alpha} + \frac{\partial J}{\partial \alpha} \tag{4.54}$$

In (4.54) the term $dU/d\alpha$ satisfies the linearized form of (4.53):

$$\frac{\partial N}{\partial U}\frac{dU}{d\alpha} + \frac{\partial N}{\partial \alpha} = 0 \tag{4.55}$$

Rewriting (4.55):

$$\frac{\partial N}{\partial U} \equiv A, \quad \frac{dU}{d\alpha} \equiv u, \quad \frac{\partial N}{\partial \alpha} \equiv -f \rightarrow Au = f \tag{4.56}$$

And we recover the conventional definition of the gradient of the cost function in (4.57):

$$g^T = \frac{\partial J}{\partial u} \rightarrow \frac{dJ}{d\alpha} = g^T u + \frac{\partial J}{\partial \alpha} \tag{4.57}$$

Note that the term $\partial J/\partial \alpha$ is typically easy to compute analytically. The first term involving $\partial J/\partial U$, however, can be costly. Note that $g^T u \equiv v^T f$. Therefore we can evaluate the first term in (4.57) either by evaluating the direct formulation $Au = f$ or the adjoint form $A^T v = g$. For a single design variable there is no difference in the cost and there is no benefit to evaluating the adjoint form. However for n design variables we must evaluate the direct formulation, $(Au = f)$, n times because each design variable has a different $f = \partial N/\partial \alpha$. This is not the case for the adjoint form as there is only a single $g = [\partial J/\partial u]^T$ common to all the design variables. Therefore, it is far less costly to evaluate the adjoint formulations when there are many design variables [103].

Alternatively, and perhaps more intuitively, adjoint variables can be derived by considering them as Lagrange multipliers [103]. Simply, because $N(U, \alpha) = 0$ the augmented objective function, $I(U, \alpha)$ can be written (4.58):

$$I(U,\alpha) = J(U,\alpha) - \lambda^T N(U,\alpha), \quad N(U,\alpha) = 0 \quad \rightarrow \quad I(U,\alpha) = J(U,\alpha) \tag{4.58}$$

Again λ are the adjoint variables. Differentiating:

$$dI = \left(\frac{\partial J}{\partial U} - \lambda^T \frac{\partial N}{\partial U}\right) dU + \left(\frac{\partial J}{\partial \alpha} - \lambda^T \frac{\partial N}{\partial \alpha}\right) d\alpha \tag{4.59}$$

Note that λ can be chosen to be any value without violating (4.58) so it is chosen to conveniently eliminate the first term in (4.59). We again then recover the adjoint equation (4.61) [103]:

$$\left(\frac{\partial N}{\partial U}\right)^T \lambda = \left(\frac{\partial J}{\partial U}\right)^T \tag{4.60}$$

And the derivative is computed:

$$\frac{dI}{d\alpha} = \frac{\partial J}{\partial \alpha} - \lambda^T \frac{\partial N}{\partial \alpha} \tag{4.61}$$

For FIML-Classic and FIML-Embedded we have a design variable at every spatial point in the domain. For RANS applications this typically requires tens of thousands of design variables and direct solutions requiring hours of computational time. In this framework it is not practical to evaluate $Au = f$ for each design variable and the adjoint formulation is required. For FIML-Direct the number of design variables is typically somewhat reduced, where even large neural networks will only have hundreds of weights. However it would still be far too costly to evaluate approach and therefore adjoint formulations are again required for the FIML-Direct approach.

4.7.1 Discrete Adjoint Implementation

In this section the discrete adjoint implementation in the chosen solver (SU2) is discussed, and the modifications required to implement the various FIML approaches in SU2 are presented. The SU2 open source CFD software suite was chosen, in part,

due to its implementation of the autodifferentiated discrete adjoint solver, which is presented and well explained by Albring et al.. This discrete adjoint solver was originally developed for aerodynamic design optimization problems, where the design variables modify an aerodynamic surface to minimize a cost function (minimize drag, for example). The flexibility of SU2 and the autodifferentiated discrete adjoint approach has enabled the extension of the SU2 package to solve a variety of optimization problems [104, 105, 106, 107, 108, 109]. Given that the present effort made use of the SU2 discrete adjoint solver, we follow the notation and explanation of the adjoint solver given by Albring et al. [104] here, with some modification to represent the changes for the FIML procedures. Given a cost function, J, design variables, α, and state variables, U, we have the following optimization problem (4.62):

$$\min_{\alpha} \; J(U(\alpha)) \tag{4.62}$$

$$\text{Subject to} \quad U(\alpha) = G(U(\alpha), X(\alpha)) \tag{4.63}$$

Where $G(\alpha)$ results from the discretization of and solution of $N(U(\alpha)) = 0$:

$$U^{n+1} = U^n + N(U(\alpha)^n) \tag{4.64}$$

$$G(U^n) \equiv U^n + N(U(\alpha)^n) \tag{4.65}$$

$N(U(\alpha))$ is the residual at the current iteration. If the system of equations has been solved, denoted by $*$, then $N(U^*) = 0$ which implies that with converged U we have $U^* = G(U^*)$.

This framework is sufficient for the FIML-Classic and FIML-Direct implementation. For these cases $\alpha = \beta$ and $\alpha = w$ respectively. However, for the FIML-Embedded implementation ($\alpha = \beta_T$) there is an additional constraint on the minimization. Let $X(U, \alpha) = H(\alpha)$ be the backpropagation algorithm which is an additional constraint on the minimization problem such that (4.62) becomes (4.66):

$$\min_{\alpha} J(U(\alpha), X(\alpha)) \tag{4.66}$$

$$\text{Subject to} \quad U(\alpha) = G(U(\alpha), X(\alpha)) \tag{4.67}$$

$$X(\alpha) = M(U(\alpha), \alpha) \tag{4.68}$$

Then, following the Lagrange multiplier explanation, we write the Lagrangian, where the additional adjoint variables, λ, are marked with a bar to denote the association to the variable in question ($\bar{U}$ is adjoint of flow variables, $\bar{X}$ is the adjoint of the neural network weights):

$$I(\alpha, U, X, \bar{U}, \bar{X}) = J(U, X) + [G(U, X) - U]^T \bar{U} + [M(\alpha) - X]^T \bar{X} \tag{4.69}$$

$$= O(U, \bar{U}, X) - U^T \bar{U} + [M(\alpha) - X]^T \bar{X} \tag{4.70}$$

Where $O(U, \bar{U}, X)$ is introduced as:

$$O(U, \bar{U}, X) \equiv J(U, X) + G^T(U, X)\bar{U} \tag{4.71}$$

The values of the adjoint variables are again chosen to eliminate troublesome variables from the derivative of (4.69), namely $\partial U/\partial\alpha$ and $\partial X/\partial\alpha$, giving:

$$\bar{U} = \frac{\partial}{\partial U} O(U, \bar{U}, X) = \frac{\partial}{\partial U} J^T(U, X) + \frac{\partial}{\partial U} G^T(U, X)\bar{U} \tag{4.72}$$

$$\bar{X} = \frac{\partial}{\partial X} O(U, \bar{U}, X) = \frac{\partial}{\partial X} J^T(U, X) + \frac{\partial}{\partial X} G^T(U, X)\bar{U} \tag{4.73}$$

And the required derivative is then:

$$\frac{dJ^T}{d\alpha} = \frac{d}{d\alpha} M^T(\alpha)\bar{X} \tag{4.74}$$

To solve for the adjoint variables, a fixed point iteration is performed such that:

$$\bar{U}^{n+1} = \frac{\partial}{\partial U} O(U^*, \bar{U}^n, X^*, \bar{X}^n) \tag{4.75}$$

$$\bar{X}^{n+1} = \frac{\partial}{\partial X} O(U^*, \bar{U}^n, X^*, \bar{X}^n) \tag{4.76}$$

Following convergence of (4.75), the adjoint variables have been found and the derivative can be computed. It can be shown that the adjoint fixed point iteration inherits the convergence properties of the forward (direct/primal) solution and therefore if the flow solver converges the adjoint solver will as well. The additional adjoint variables for the weights, $\bar{X}$ are required for the FIML-Embedded adjoint computation, but are dropped from the computation for the FIML-Classic approach, as well as the FIML-Direct approach because the weights are not converging (no backpropagation) and therefore the derivative information of the forward propagation in the FIML-Direct procedure can be computed via autodifferentiation without the additional adjoint variables.

Autodifferentiation tools were used to compute the required derivatives in (4.75). It is possible to compute the derivatives by hand, but autodifferentiation greatly simplifies the process. Essentially, the code that was used in the primal solver to produce U^* is recorded. For the derivative computation, the record is used to replace every elementary operation with its derivative. This dramatically simplifies the development process. SU2 makes use of the autodifferentiation tool CoDiPack [110].

Because of the use of autodifferentiation, the development effort was minimized. The primary effort of modifying SU2 to support the FIML approaches was to redefine the design variables, assemble and scale the features as required, and to implement the neural network forward and backpropagation algorithms. For FIML-Embedded, the additional modification of adding adjoint variables for the weights was required. For FIML-Direct, additional adjoint variables for the weights are not required as the weights are held fixed, and therefore the required derivative information is automatically accounted for by the autodifferentiation algorithm.

4.8 Model Problem: The 1D Heat Equation

To demonstrate all three methods we consider the 1D heat equation given by equation 4.77 and also presented by Parish and Duraisamy [39].

$$\frac{d^2T}{dz^2} = \varepsilon(T)(T_\infty^4 - T^4) + h(T_\infty - T), \quad z \in [0 \dots 1], \quad h = 0.5 \tag{4.77}$$

The emissivity, ε, is taken to be a nonlinear function of temperature:

$$\varepsilon(T) = \left[1 + 5\sin\left(\frac{3\pi}{200}T\right) + \exp(0.02T) + \mathcal{N}(0, 0.1^2)\right] \times 10^{-4} \tag{4.78}$$

To model this problem we use the following imperfect model of the true process. This imperfect model is missing the linear term, and therefore there is a deficiency in the functional form of the model. It is, therefore, a good analogy to the RANS application as the error in the RANS models are not likely due to the uncertainty in the closure constants, but are likely due to errors in the functional form of the model:

$$\frac{d^2T}{dz^2} = \varepsilon_o(T_\infty^4 - T^4), \quad z \in [0 \dots 1], \quad \varepsilon_o = 5 \times 10^{-4} \tag{4.79}$$

To correct the imperfect model through the FIML inversion process we multiply the right hand side of (4.79) by the function β. For FIML-Classic the inverse process first solves for $\beta(z)$ and then trains a neural network to determine $\beta(T, T_\infty)$. FIML-Embedded and FIML-Direct solve for $\beta(T, T_\infty)$ in the inverse procedure. For the 1D heat equation model problem we use the following objective functions:

$$\text{FIML-Classic:} \quad J_c(\beta) = \frac{1}{2}\sum_i^N (T_i(\beta) - T_{truth_i})^2 \tag{4.80}$$

$$\text{FIML-Embedded:} \quad J_e(\beta_T) = \frac{1}{2}\sum_i^N (T_i(\beta_T) - T_{truth_i})^2 \tag{4.81}$$

$$\text{FIML-Direct:} \quad J_d(w) = \frac{1}{2}\sum_i^N (T_i(w) - T_{truth_i})^2 \tag{4.82}$$

Note, that for this model problem there are essentially only two variables to use as inputs for an augmented model. Therefore, the features for our augmented model are chosen to be $\eta = \{T, T_\infty\}$ at each spatial point.

In this form, β has an analytical solution that, if found in the inversion process, will result in a fully corrected model.

$$\beta(T, T_\infty) = \frac{1}{\epsilon_0}\left[1 + 5\sin(\frac{3\pi}{200}T) + \exp(0.02T) + N(0, 0.1)\right] \times 10^{-4} + \frac{h}{\epsilon_0}\frac{T_\infty - T}{T_\infty^4 - T^4} \quad (4.83)$$

Note that it is not guaranteed that the chosen machine learning algorithm can sufficiently replicate (4.83). For FIML-Classic the procedure is to first perform the inversion to find the optimal correction $\beta(z) = \arg\min_\beta(J_c)$, and then train a machine learned model to replicate the solution of the inversion. It is not always possible to sufficiently replicate $\beta(z)$ with a machine learning algorithm ($\beta(\eta)$) which results in a degradation of performance of the augmented model due to imperfect learning. FIML-Embedded and FIML-Direct have a substantial advantage in this regard, as the training is performed in the inversion step. This will naturally restrict the solution of the inversion to what is learnable by the chosen algorithm, and by including the limitation of the algorithm in the inversion the optimization procedure will find the optimal solution that can be learned by the machine learning algorithm

The 1D heat equation model problem is considered for two important reasons. First, the 1D heat equation is substantially more simple than a RANS simulation. By using a single equation in a single dimension it is substantially easier to under-

stand the algorithms presented. Second, it demonstrates the applicability of all three FIML approaches to other physics-based systems. FIML-Classic, FIML-Embedded, and FIML-Direct are applicable to many physics-based simulations and by demonstrating these approaches on the 1D heat equation this flexibility is demonstrated.

4.8.1 Model Problem: FI-Classic Implementation

The 1D heat equation implementation is relatively straightforward. First the correction field $\beta(z)$ is uniformly initialized to $\beta(z) = 1.0$. The correction field is then applied to the right hand side of (4.79) to give (4.84):

$$\frac{d^2T}{dz^2} = \beta(z)\varepsilon_o(T_\infty^4 - T^4), \quad z \in [0 \dots 1], \quad \varepsilon_o = 5 \times 10^{-4} \tag{4.84}$$

Equation (4.84) is then solved by finite differences. In this application, second order central finite difference approximations were used for the second order derivatives with one sided second order finite difference approximations on the boundaries. This gives the following discretization:

$$B = \beta(z)\varepsilon_o(T_\infty^4 - T^4) \tag{4.85}$$

Where the matrix B is the discrete approximation of second order accurate spatial derivative of T. The system of equations is then solved by adding a pseudo-time and marching to the solution of (4.84). This results in the update equation in (4.87). Dirichlet boundary conditions of $T = 0$ are enforced at both ends of the spatial domain.

$$B \equiv \begin{bmatrix} 2 & -5 & 4 & -1 & 0 & \dots & \dots & 0 \\ 1 & -2 & 1 & 0 & & & & \vdots \\ 0 & 1 & -2 & 1 & & & & \\ \vdots & & \ddots & \ddots & \ddots & & & \vdots \\ & & & & 1 & -2 & 1 & 0 \\ \vdots & & & & 0 & 1 & -2 & 1 \\ 0 & \dots & \dots & 0 & -1 & 4 & -5 & 2 \end{bmatrix} \tag{4.86}$$

$$T^{(n+1)} = T^{(n)} + \Delta T, \quad \Delta T = \frac{\Delta t}{\Delta z^2} BT - \Delta t \beta(z) \varepsilon_o (T_\infty^4 - T^4) \tag{4.87}$$

To generate the truth temperature distribution, equation (4.77) is solved and sampled a number of times. This gives a temperature distribution that defines our data (k_d) for our problem. In the applications for this thesis only the mean of 100 samples is used, but in a Bayesian setting, as demonstrated by Parish and Duraisamy, the sampling procedure defines the prior distribution required for Bayesian inference [39]. After solving the model problem for the current design the objective function (4.80) is computed.

The gradient of the objective function J_c with respect to the design variables, $\beta(z)$ is efficiently computed via adjoint methods. First we set $u \equiv T(z)$ and solve for the adjoint variables (v) via the dual form given by $A^T v = g$. Here, using the linearized form of the equations to form the matrix A we have:

$$A = BT - 4\beta\varepsilon_o T^3 \tag{4.88}$$

Note that the temperature distribution ($T \equiv u$) has been found from the direct (primal) solver, so the adjoint solver requires a converged solution from the direct solver. For the model problem we use the objective function J_c given by (4.80) and therefore we have the following form for g:

$$g \equiv \left[\frac{\partial J}{\partial u}\right]^T = -(T_\infty - T) \tag{4.89}$$

The adjoint variables, v, are then solved similarly to the direct solution by solving for $A^T v = g$. Finally, the required derivative is computed from Equation (4.90):

$$\frac{dJ}{d\alpha} = \frac{\partial J}{\partial U}\frac{dU}{d\alpha} + \frac{\partial J}{\partial \alpha} = g^T u + \frac{\partial J}{\partial \alpha} g^T u \equiv v^T f \rightarrow \frac{dJ}{d\alpha} = v^T \varepsilon_0 (T_\infty^4 - T^4) \tag{4.90}$$

This gives the required gradient. For each design (each $\beta(z)$) the direct solver produces the temperature distribution $T(z)$ that satisfies the model problem and returns $T(z)$ and the current objective function value $J_c(\beta)$. The temperature distribution is then passed to the adjoint solver which computes the adjoint variables, v that satisfy $A^T v = g$ and returns the gradient, $dJ/d\beta$. The optimizer then generates a new β distribution that the gradient based optimizer (such as BFGS) expects will produce a lower cost function and the process is repeated. In this application the BFGS implementation provided in the open source library SciPy was utilized [96, 87].

4.8.2 Model Problem: FIML-Embedded

For FIML-Embedded, the minimization for J_e begins with the initialization of $\beta_T(z) = 1.0$. This is the "training" correction, and is passed to the direct (primal) solver to evaluate J_e. At the beginning of each new direct solve new weights, w, are initialized randomly. In other words, we train a new neural network every time J_e is evaluated. To solve for the current temperature distribution first the current feature values are collected and scaled, for the 1D heat equation the features are $\eta = \{T, T_\infty\}$. The weights and the features are then used to solve for the current value of the correction, $\beta(\eta, w)$ using Equation (3.4). The backpropagation algorithm is then performed to update the weights to minimize the sum squared error between the current correction ($\beta(\eta, w)$) and the training correction $\beta_T(z)$. Note that $\beta_T(z)$ is constant every J_e evaluation, however η, w, and β are updated every solver iteration. The temperatures are then updated using Equation (4.87). The process then repeats (gather and scale η, update $\beta(\eta, w)$, update w, update T) until the temperature distribution stops changing. In this fashion the temperature distribution converges with the weights. Following each J_e evaluation a neural network has been trained. Figure 4.8 displays the algorithm for evaluating the current value of J_e:

Complex step differentiation was utilized for the gradient computation in the 1D heat equation FIML-Embedded implementation. This greatly simplifies the algorithm development and still yields machine accurate gradient information, albeit at a much greater computational cost over the adjoint approach. The complex step differentiation algorithm is, however, an embarrassingly parallel problem for each

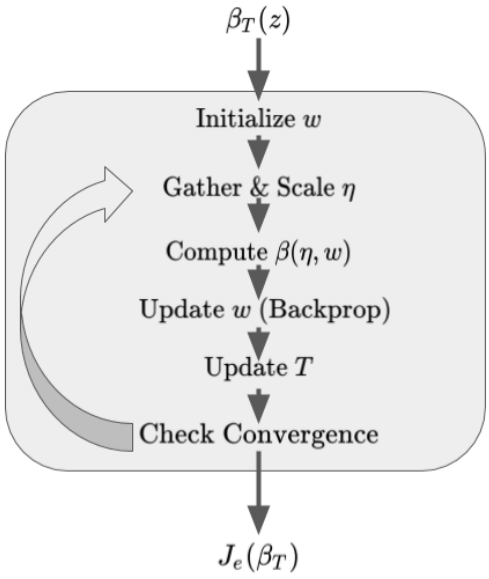

Figure 4.8: Illustration of the iteration process for 1D Heat direct (primal) solver for FIML-Embedded.

element of the gradient and therefore the process was parallelized to minimize the required run time.

With the gradient computation complete, the design variables, β_T, are updated using the BFGS algorithm in order to minimize the cost function. As is likely apparent from this discussion, the FIML-Embedded approach has more complexity than the FIML-Classic approach, as well as the FIML-Direct method discussed in the next section.

4.8.3 Model Problem: FIML-Direct

The solution process for FIML-Direct is largely the same as for FIML-Classic except the design variables are the weights of the neural network. This modification requires only minor changes to the direct and adjoint solver.

First the weights, w, are initialized randomly by the optimizer. This will initially give a very poor estimate to β, and the initial solution to $J_d(w)$ will not be equivalent to the baseline model ($\beta(w) \neq 1.0$ everywhere). In order to have the baseline solution for β be close to 1 everywhere for the initial solution we have the neural network learn $\Delta\beta$ so that $\beta = 1 + \Delta\beta$ and initialize the weights to small random values. This is equivalent to setting the output layer bias node to 1 for the first iteration, and therefore to simplify the notation β will be considered the output of the neural network in this discussion. This initialization detail is not strictly necessary but helps avoid potential numerical solution issues with an initial design far away from the baseline model. Initializing the weights to 0 everywhere is not possible as this would also result in $dJ_d/dw = 0$ because the weights would have no influence on the value of β.

The initialized weights are given to the direct (primal) solver in order to evaluate J_d. The features are initialized and scaled and the forward propagation algorithm is performed to find $\beta(w, \eta)$. The temperatures are updated and then the process is repeated (gather and scale features, update $\beta(w, \eta)$, update T) until the temperature distribution stops changing, and then β and T have converged to their final values. J_d is computed and passed to the optimizer.

To find the gradient the same adjoint numerical method discussed in the FIML-Classic section of this chapter is performed again with the converged value of $\beta(w, \eta)$ found in the direct solver. This produces the gradient $dJ_d/d\beta$. Then complex step differentiation is performed on the forward propagation algorithm to produce the gradient vector $d\beta/dw$. The chain rule is then applied to find the required gradient

for the optimizer: dJ_d/dw. The BFGS optimizer can then update the weights and the next evaluation of J_d begins until J_d has been sufficiently minimized.

4.9 Summary

In this chapter the numerical methodology was presented. The first section focused on the equations governing fluid flow and accommodations that must be made in order to efficiently solve for turbulent problems at practical Reynolds numbers. The turbulence models utilized in this work to provide closure to the RANS equations were then presented.

The next sections provided greater detail on machine learning with neural networks for regression. Neural networks, only discussed in passing up until this chapter, were explained along with the algorithmic documentation of the method for training these networks. A section was devoted to feature selection and scaling as this is a critical step that any modeler must consider in order to successfully construct a useful neural network.

Numerical details concerning the three FIML approaches used were then presented, along with the minimization algorithm used in all three approaches: the quasi-Newton BFGS method. This method is popular for highly dimensional minimization problems.

The next sections discussed methods of obtaining gradient information including the complex step finite difference method and adjoint methods. The adjoint equations were presented, along with a detailed explanation of the discrete adjoint

approach implemented and modified in the SU2 code.

Finally, the 1D heat equation model problem was presented. This simplified problem provides a simple modeling environment to test all three FIML approaches, and much of the numerical methodology presented in this Chapter. It also demonstrates that the FIML approach can be applied to other physics-based simulations (other than RANS simulations). The next chapter begins to present the results of this methodology, beginning with the 1D heat equation.

Chapter 5: One-Dimensional Heat Equation Simulations

5.1 Overview

This chapter presents the results of all three FIML methods to augment the 1D heat equation model problem. Results for the FIML-Embedded and FIML-Direct methods are presented here for the first time. First, the results of the 1D heat equation using FIML-Classic are presented for a single case. The inversion is then performed on four cases and a neural network augmentation trained on the information resulting from the inversion. The augmentation is then tested on holdout data. Then the FIML-Embedded results are presented for a single case. This demonstration is promising, but as discussed the methodology for considering multiple cases with FIML-Embedded has not yet been developed so results cannot be presented here. The results for FIML-Direct are presented for a single case, as well as the simultaneous inversion of multiple cases. The results from this model problem are very encouraging and are discussed. Finally, the FIML-Direct procedure is repeated using the stochastic gradient descent algorithm as an example demonstrating an algorithm that may be more favorable for considering large numbers of cases in certain computing environments.

5.2 FIML-Classic

First, the truth distribution (4.77) is sampled which gives the truth temperature distribution to which we compare our imperfect model given by (4.79). Then the minimization procedure is performed using gradient descent methods to minimize (4.80) with respect to β. In other words, the minimization is finding the spatially distributed $\beta(z) = \arg\min_{\beta}(J_c)$. This completes the field inversion process (FI-Classic)[1]. An example of the inversion results is shown in 5.1.

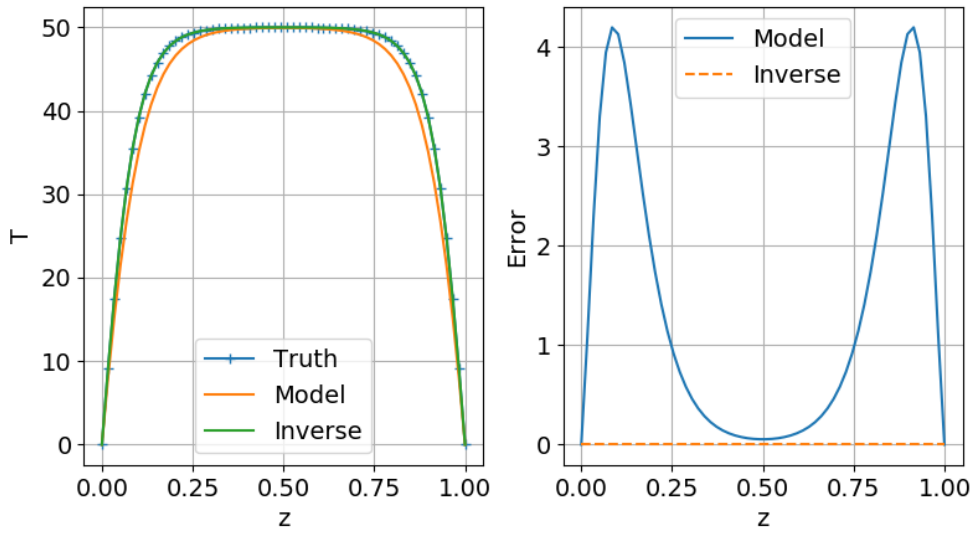

Figure 5.1: Figure of FI-Classic results for $T_\infty = 50$.

As shown in 5.1, the uncorrected model shows significant error. This error is almost completely eliminated following the inversion (right panel Figure 5.1). The

[1]Note the drop of the "ML" from FIML, this notation will be used to denote cases where the inversion has been performed to generate the model-consistent information, but the information has not been learned by a neural network (offline training not yet performed).

residual error is insignificant.

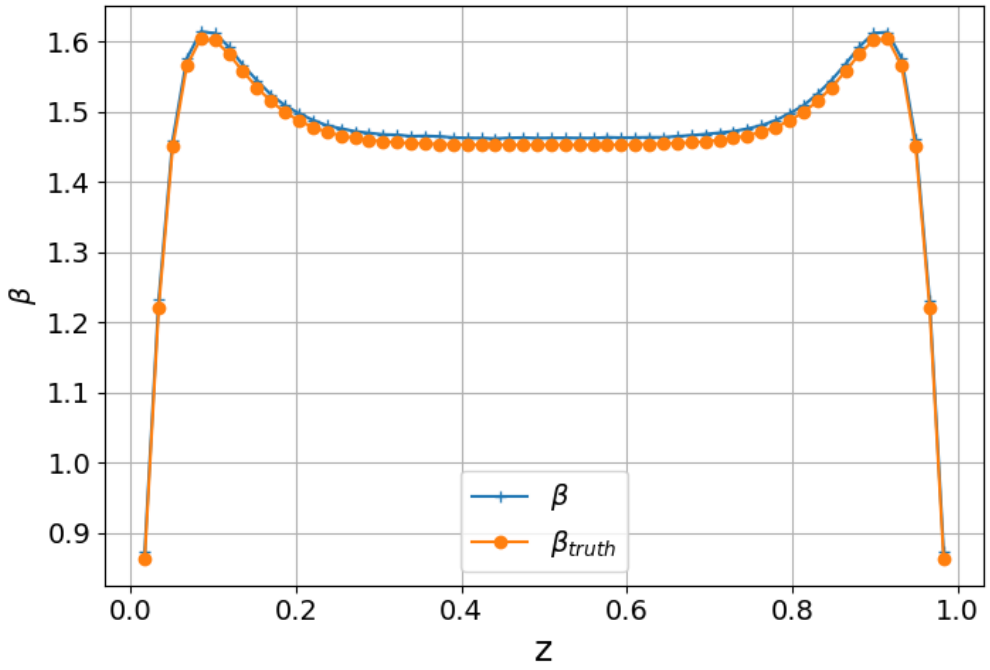

Figure 5.2: FI-Classic results displaying computed $\beta(z)$ for $T_\infty = 50$.

However, as shown in Figure 5.2 the inversion did not exactly produce the β distribution of the analytical solution. This is because there are portions of the solution for this T_∞ where the model performs well and is relatively insensitive to β. This residual error is not eliminated by additional optimizer iterations.

Figure 5.3 shows the inversion history of (4.80), J_c. The gradient-based optimization (BFGS) substantially reduced the value of J_c indicating that the optimization was successful in minimizing the error of the model. As with all gradient-based optimization procedures, there is no guarantee that the global optimum has been found. In this case, as there is an analytical solution for the optimum solution (4.83), we know that the final solution is very close, but not identical to the global

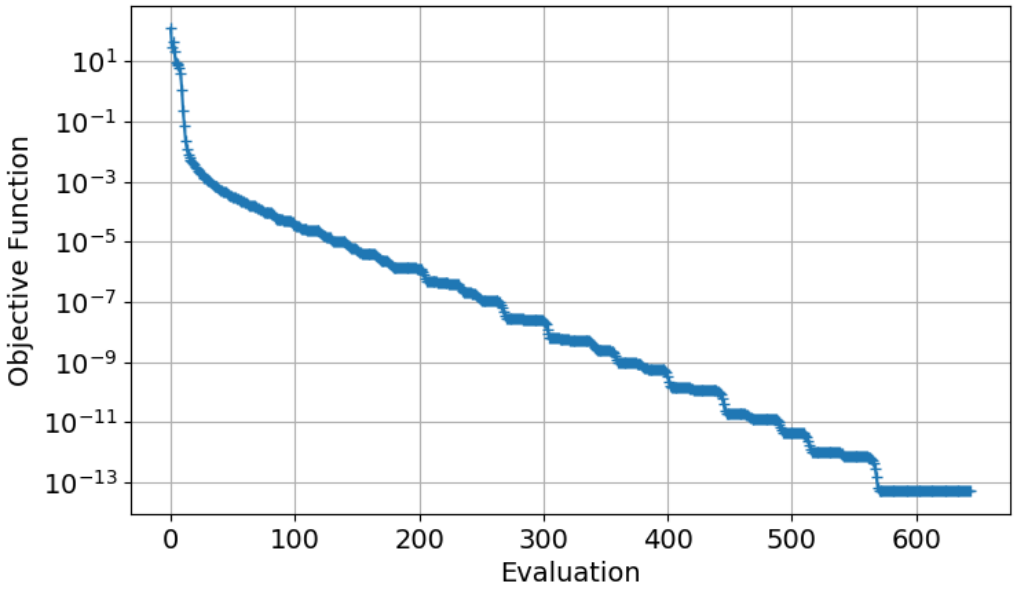

Figure 5.3: Objective function convergence history of (J_c) for $T_\infty = 50$.

(analytic) optimum. Also note, the inversion initially converges very quickly in the first 20 iterations. For this simulation it is relatively inexpensive to evaluate J_c and $dJ_c/d\beta$, however this convergence behavior could be very beneficial for more expensive applications such as RANS turbulence modeling as substantial improvement can be made with few optimizer iterations.

At this point only the field inversion process has been completed. The inversion is repeated for a number of cases (several T_∞) in order to generate training data that the machine learning algorithm will learn from. In this case we choose $T_\infty = \{20.0, 30.0, 40.0, 50.0\}$ as the training set. The inversion results for each of these cases is performed and saved.

From our training set we wish to create a machine learning model that can predict the β without performing an inversion. For this application we use fully

connected feed forward neural networks for regression. In this application there are only three variables (T, T_∞, z). The inversion created a spatially correlated correction distribution $\beta(z)$ and we now want to create a function $\beta(T, T_\infty)$. T and T_∞ are our "features" (η) that are the inputs to the neural network. The number of layers in the neural network and the number of nodes are free parameters that must be chosen by the modeler. In this application a single layer with 20 nodes was used. The hidden layer used a hyperbolic tangent activation function $(y = \tanh(x))$ and, as is typical for regression networks, the output node used a linear activation function $(y = x)$. The neural network was trained using the Scikit-Learn software [92]. The limited memory BFGS (L-BFGS-B) algorithm [88] was used to minimize the log loss function with respect to the weights of the neural network. 20% of the training data points were chosen at random and used as holdout points in order to test the algorithm, and guard against overfitting/overtraining, where the trained neural network performs very well on the training set but exhibits poor generalization for cases other than the training data (unseen data). The training results are shown in Figure 5.4. Note that for this neural network structure and activations some significant error remains following training. Though it is possible another neural network could perform better, there will always be residual error following learning. Note that this is error that was not considered during the inversion, and naturally, this will result in reduced performance of the augmented model due to imperfect training.

The truth temperature distribution, baseline model, and neural network augmented model results for each temperature in the training set are shown in Figure

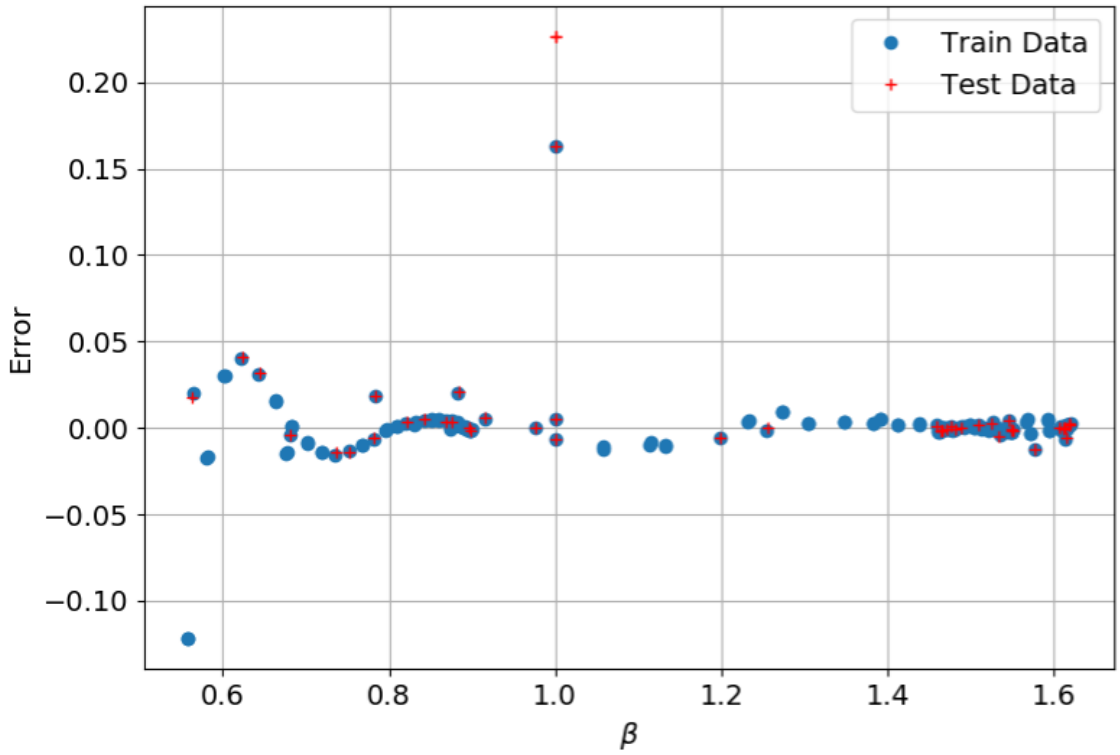

Figure 5.4: FIML-Classic Neural Network Error Following Training.

5.5. The error is almost completely eliminated for the temperatures in the training set indicating that the errors observed due to imperfect training (Figure 5.4) were not significant enough to produce significant errors in the augmented model. This is not true in general, as will be demonstrated in the FIML-Classic application to the S809 airfoil in Chapter 6.

To further explore the generalization capabilities the augmented model is tested for conditions that were not included in the training set (conditions that we did not perform an inversion for). Additionally, the augmented model was tested for a variable T_∞ case, while the neural network was trained using cases uniform T_∞ cases. The performance of the augmented model for these unseen conditions

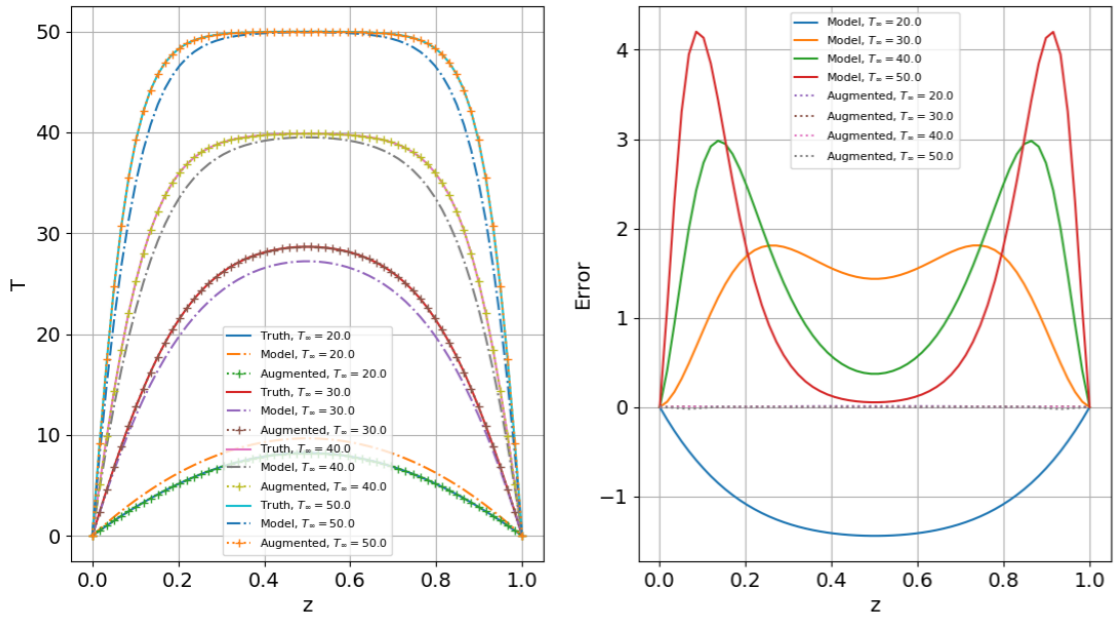

Figure 5.5: FIML-Classic results of model augmentation for training temperatures.

is shown in Figure 5.6. The correction $\beta(T, T_\infty)$ produced by the neural network augmentation is shown in Figure 5.7.

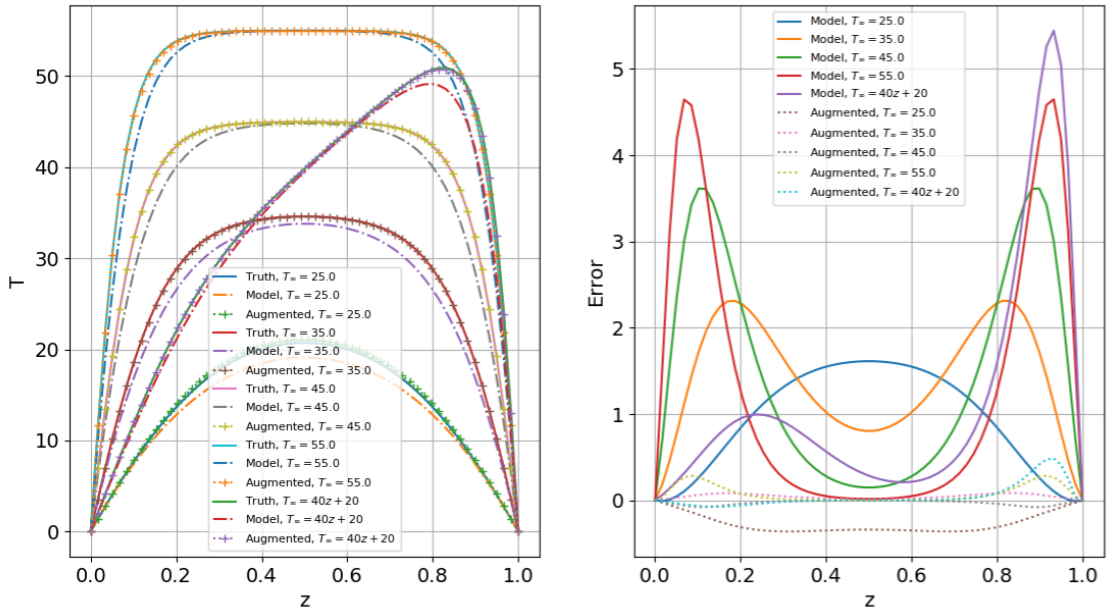

Figure 5.6: FIML-Classic results of model augmentation for holdout temperatures.

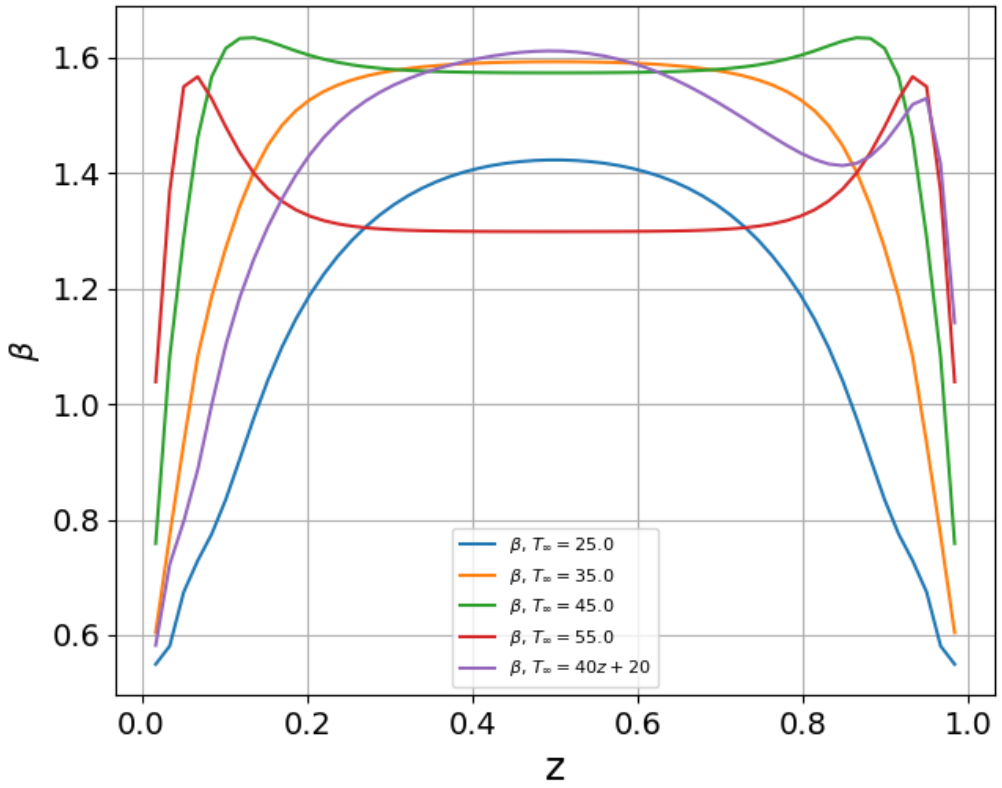

Figure 5.7: FIML-Classic neural network correction for holdout conditions.

As shown in Figure 5.6, there is some diminished performance for the holdout conditions compared to the training conditions. However the error is substantially reduced in the augmented model compared to the baseline, so the augmentation is certainly improving the model in all cases. The augmentation improved the prediction despite some extrapolation, which is illustrated in Figure 5.8. As shown, the holdout cases not only test the generalization of the network for interpolated cases, but also in some extrapolation conditions, demonstrating the robustness of the augmented model resulting from the FIML-Classic process.

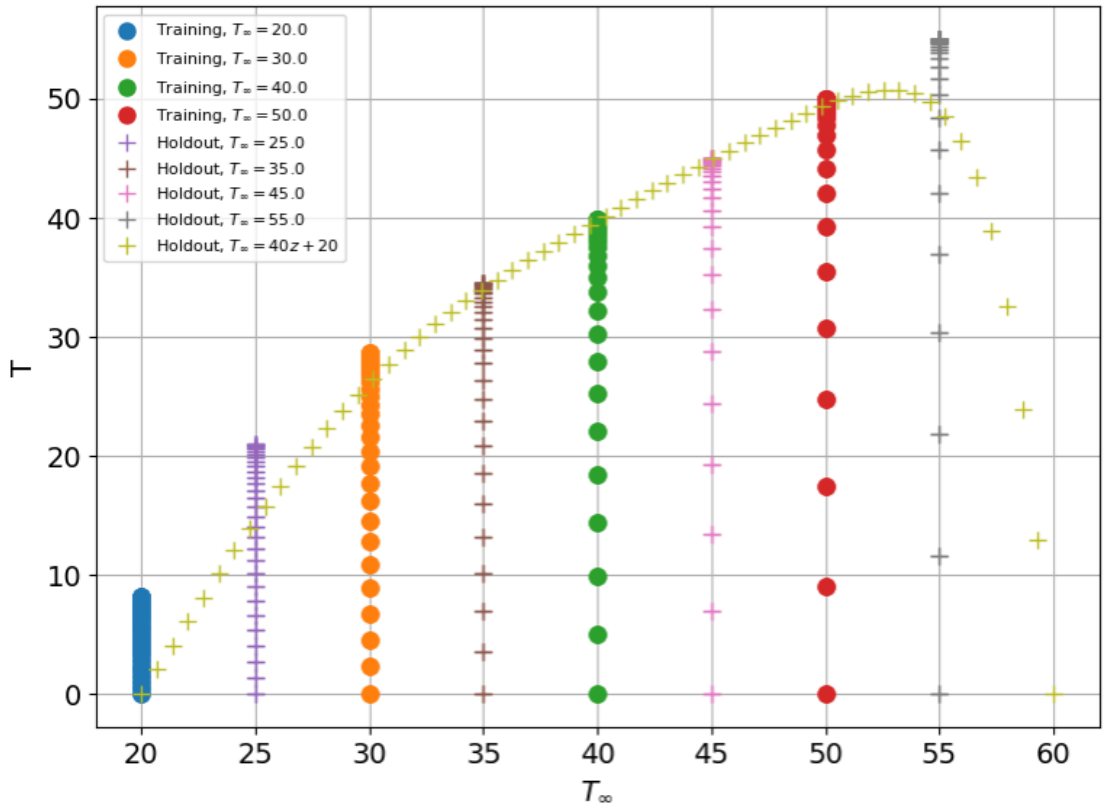

Figure 5.8: Features for FIML-Classic training and holdout cases.

5.3 FIML-Embedded

The FIML-Embedded procedure was also tested on the 1D heat equation model problem. Again, the truth distribution was sampled 100 times by solving the full "true" heat equation with the full right hand side (4.77). The weights of the neural network are initialized randomly at the start of the model (primal) solver and then updated via backpropagation. The weights converge to their final values with the temperature distribution. For the initial design the neural network is learning a uniform distribution ($\beta_T = 1.0$) and the initial temperature distribution will closely match that of the baseline model (4.79). The gradient is computed via the

complex step finite difference method which yields a machine-accurate estimation of the gradient, but is far less computationally efficient in this application compared to adjoint methods because it is a forward differentiation approach. This shortcoming is mitigated in this implementation by parallelizing the gradient computation. The gradient computation is embarrassingly parallel so this implementation is trivial and performs adequately well for a limited number of design variables. For FIML-Embedded a maximum of 60 nodes (equivalent to the number of design variables) was used. For RANS applications adjoint methods were used to compute the FIML-Embedded gradient due to the much larger number of design variables in that application.

The BFGS method was again used to perform the minimization of the cost function (4.81). The temperature distribution of the baseline model and the augmented model are shown in Figure 5.9. Note that unlike FIML-Classic, the neural network training is performed in the inversion. Therefore, following inversion, the model has already been augmented.

The error following inversion is slightly larger than that shown for the FI-Classic inversion (Figure 5.1). This is expected, because the FIML-Embedded approach accounts for the limitation of the chosen neural network in the inversion process. Therefore, it is guaranteed that the resulting inversion from the FIML-Embedded procedure can be learned. In other words, the inversion will not accept a solution that the backpropagation algorithm cannot learn.

FIML-Embedded results in a correction field, $\beta(T, T_\infty)$, that does not match the true distribution (4.83) as closely as the inverse to FI-Classic. This is due to

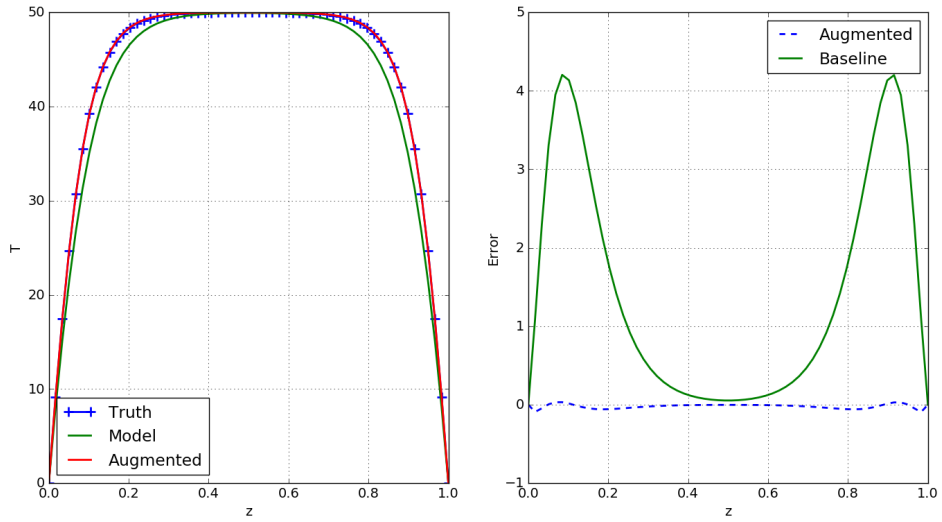

Figure 5.9: FIML-Embedded results of model augmentation.

the increased regularization of the FIML-Embedded approach. The correction field (output of the neural network augmentation) is shown in Figure 5.10.

Additionally, the history of the BFGS minimization of (4.81) is shown in Figure 5.11. This shows strong convergence and a substantial improvement over only a handful of iterations. By training the neural network inside the iterative solver we can train the neural network with only slightly increased computational effort in the model solver, and are still not required to perform excessive minimization iterations.

Note that unlike FIML-Classic and FIML-Direct it is not readily apparent how to incorporate data from multiple inversion cases (multiple T_∞) into a single augmented model. Additionally, FIML-Embedded relies on the sufficient convergence of two minimization problems: $\min_w(SSE)$ and $\min_{\beta_T}(J_e)$. The requirement to sufficiently converge the error term (SSE) by backpropagation inside the solver while the features are converging is sometimes difficult, and adds substantial complexity

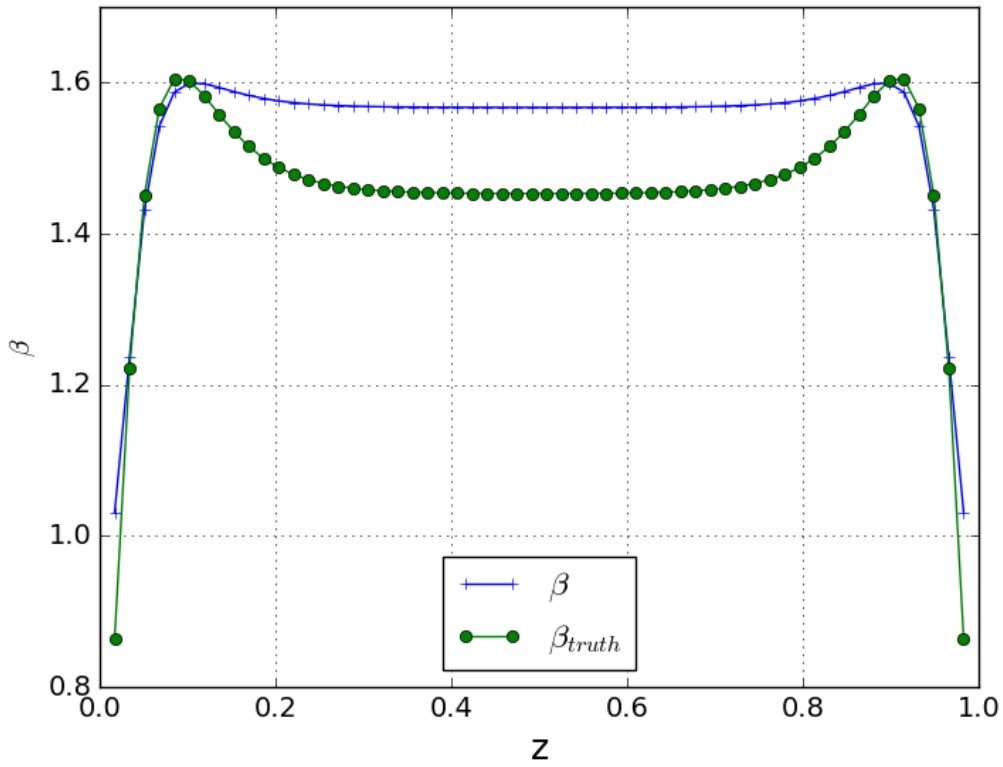

Figure 5.10: FIML-Embedded correction output from model augmentation.

to the evaluation of the current design (current J_e). These are both drawbacks of the FIML-Embedded approach. However, with FIML-Classic there is no guarantee that the chosen machine learning algorithm can learn the inverse solution. With FIML-Embedded the inverse solution is already learned, and there is no need for offline training. Additionally, FIML-Embedded utilizes the backpropagation algorithm to update the weights. This algorithm provides a very efficient weight update algorithm that enables the training of very large neural networks. Depending on the application, neural networks with multiple layers can easily have weights numbering in the hundreds. FIML-Embedded, like FIML-Classic, can efficiently train

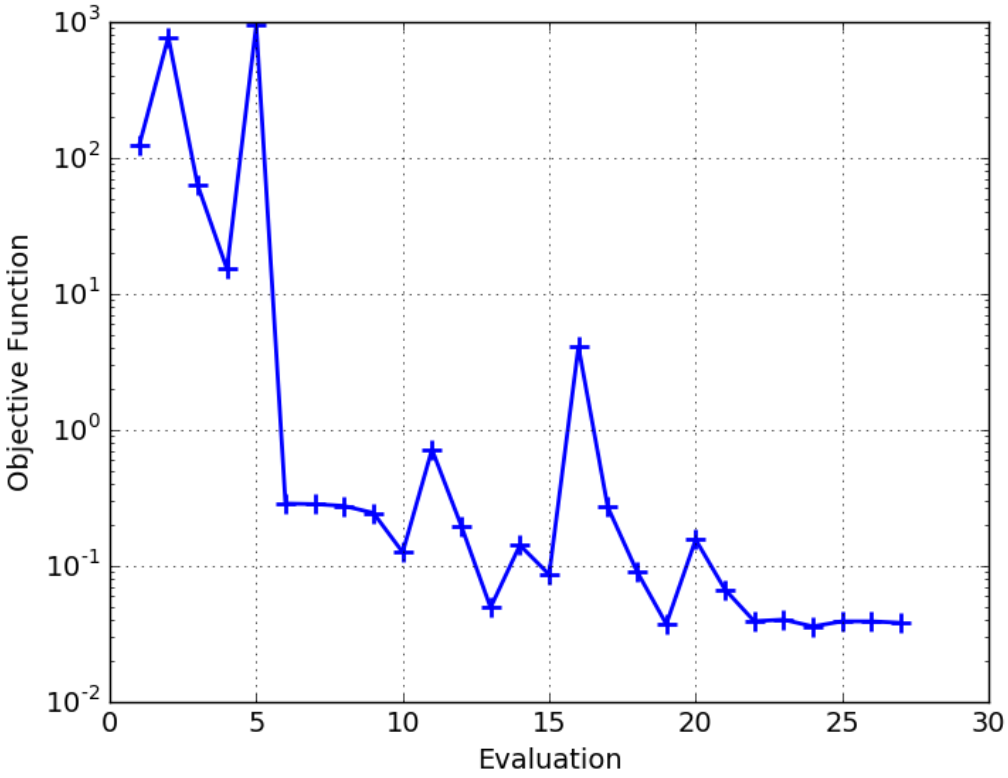

Figure 5.11: FIML-Embedded objective function minimization history.

these large networks. This may not be as straightforward when training the weights directly as discussed in the following section (FIML-Direct).

5.4 FIML-Direct

The FIML-Direct algorithm was also demonstrated using the model problem. Again the truth model was sampled 100 times to generate the truth distribution (k_d). The weights (FIML-Direct design variables) are initialized randomly. The model problem is solved by computing the current $\beta(w, T, T_\infty)$ and applying to the right hand side of the model equation. The cost function (J_d) is evaluated by

Equation (3.7). The gradient for FIML-Direct (dJ_d/dw) is evaluated through adjoint methods to compute $dJ_d/d\beta$ and $d\beta/dw$ is computed for each weight via complex step differentiation. The weights are then updated via the BFGS method in order to minimize the cost function J_d. Once the inversion is complete and a minimum J_d is found, the augmentation has been generated and is tested on holdout conditions. The weights are then held constant, and the correction is a function only of the flow features, $\beta(T, T_\infty)$. Figure 5.12 shows the results for FIML-Direct using a uniform $T_\infty = 50.0$. For this application, a single layer of five neurons using hyperbolic tangent activation functions.

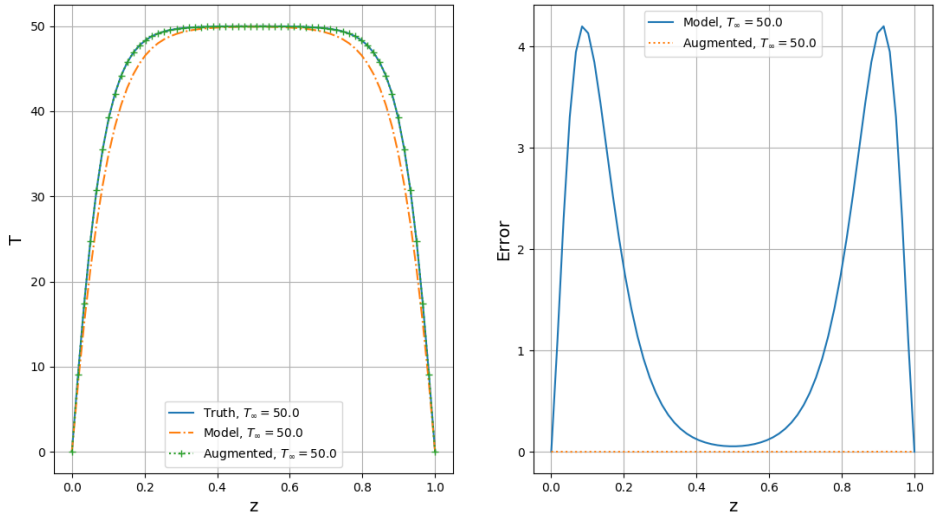

Figure 5.12: FIML-Direct results of model augmentation for a single training temperature.

The optimal correction, $\beta(T, T_\infty)$, is shown in Figure 5.13. The derived correction shows remarkable agreement with the true correction distribution for this temperature and neural network configuration. This agreement is not expected in a

general application because the limitations of the neural network are being considered in the inversion, and will limit the convergence of the neural network output to the true distribution in some cases. Despite this fact, the excellent convergence is very promising especially considering that neural networks are typically trained using large quantities of training data and thousands of weight updates. As shown in Figure 5.14, the objective function converged several orders of magnitude in a few iterations.

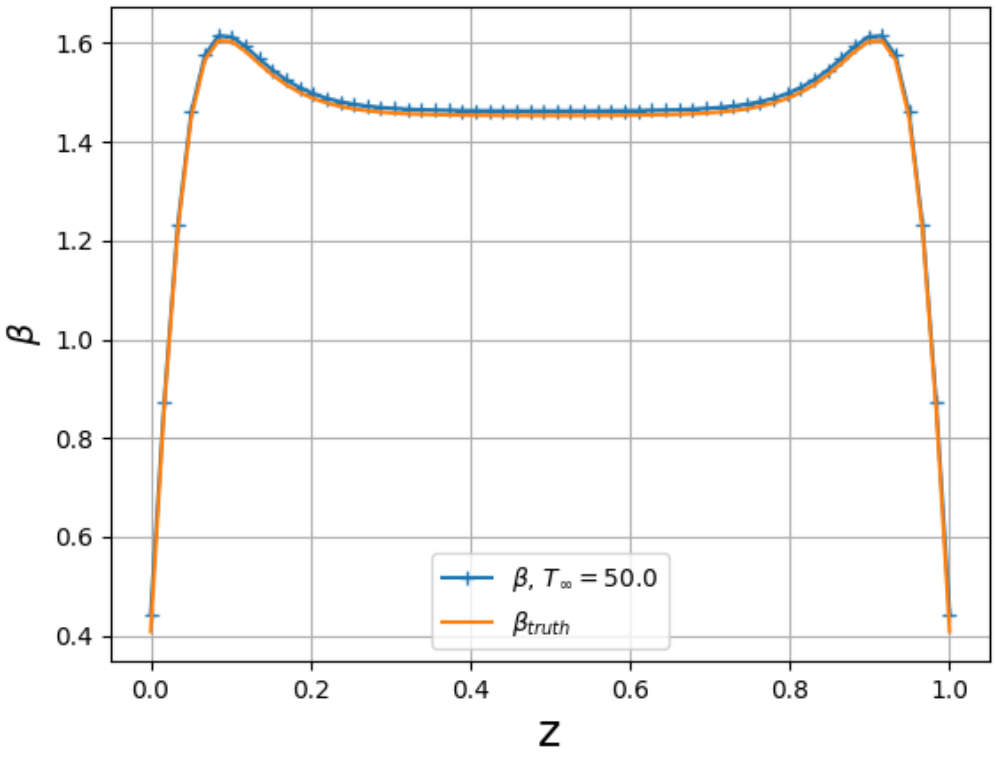

Figure 5.13: FIML-Direct correction for a single training temperature.

A major advantage of FIML-Direct over the FIML-Classic and FIML-Embedded approaches is that the design variables in the inversion minimization problem are

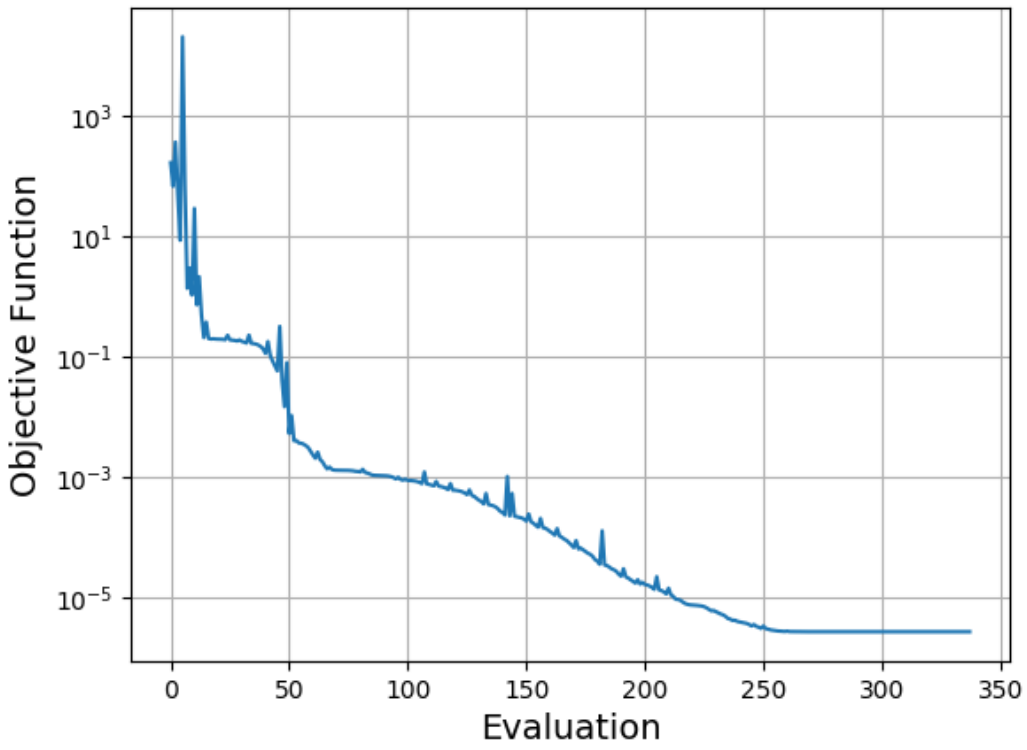

Figure 5.14: FIML-Direct correction for a single training temperature.

consistent across all cases. In this application: across T_∞ distributions. Therefore the inversion can be performed on numerous cases simultaneously simply by creating a composite objective function that is the sum of the cost function (J_d) from each case. The gradient is similarly just the sum of each case's gradient. This is demonstrated for the model problem and results are shown in Figure 5.15. Note that the augmentation is very successful with very little residual error shown for the training temperatures. This is confirmed by the excellent convergence of J_d shown in Figure 5.16.

Following the inversion/augmentation on four training temperatures the aug-

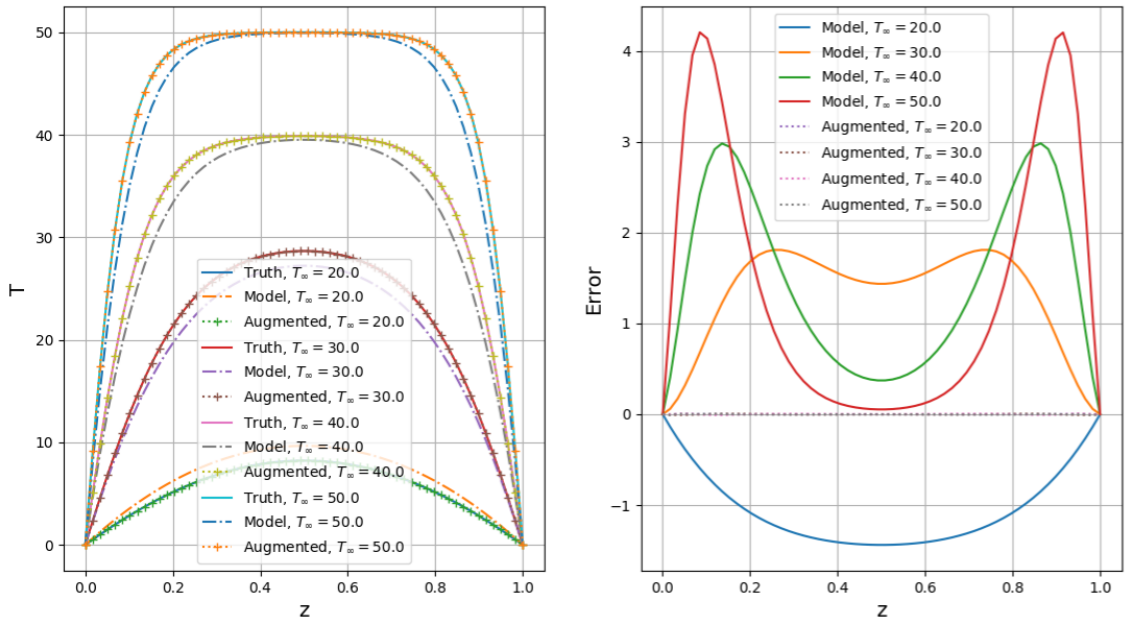

Figure 5.15: FIML-Direct results of model augmentation for four training temperatures.

mentation was tested on holdout (unseen) conditions, including a variable T_∞ distribution. As shown in Figure 5.18, the augmentation performs extremely well on the holdout conditions. This shows that the neural network is exhibiting good generalization capability, or using the machine learning terminology: the neural network is not overtrained. Despite only training on four uniform $T_\infty(z)$ distributions, the augmentation is able to substantially improve the prediction on a variable $T_\infty(z)$ holdout case. Additionally, some extrapolation is possible as the highest temperature in the training set was $T_\infty(z) = 50$, but the error is still dramatically reduced when the augmentation is tested on the holdout condition of $T_\infty(z) = 55$ and similarly on the variable T_∞ distribution that reaches $T_\infty = 60$ as z approaches 1.0.

The feature space is illustrated in Figure 5.20 and shows that the model was

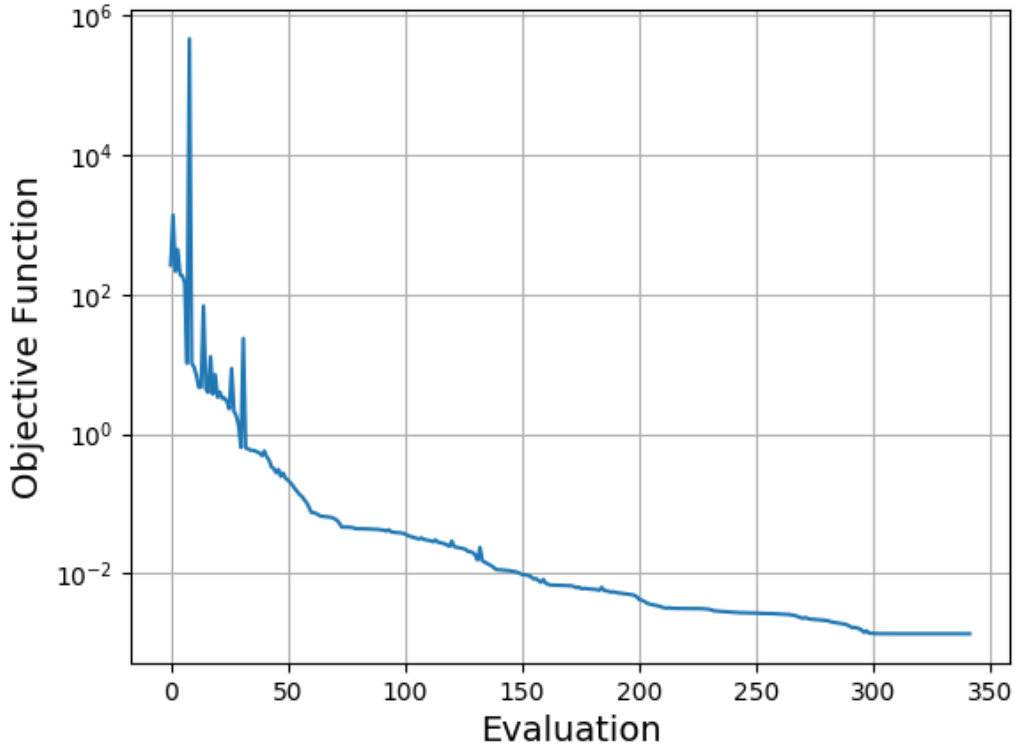

Figure 5.16: FIML-Direct Objective Function, J_d, Convergence using BFGS minimization for four training temperatures.

tested for both interpolations and modest extrapolation from the training set. Also note that excessive training conditions were not required to cover the feature space. This is a promising observation as we seek augmentations that apply to a broad variety of applications, and the results suggest that excessive sampling may not be required to cover the feature space of those applications in order to provide a substantial improvement to the performance of the augmented model.

The 1D heat equation was also tested using the FIML-Direct stochastic gradient descent approach. This method may be advantageous for performing the train-

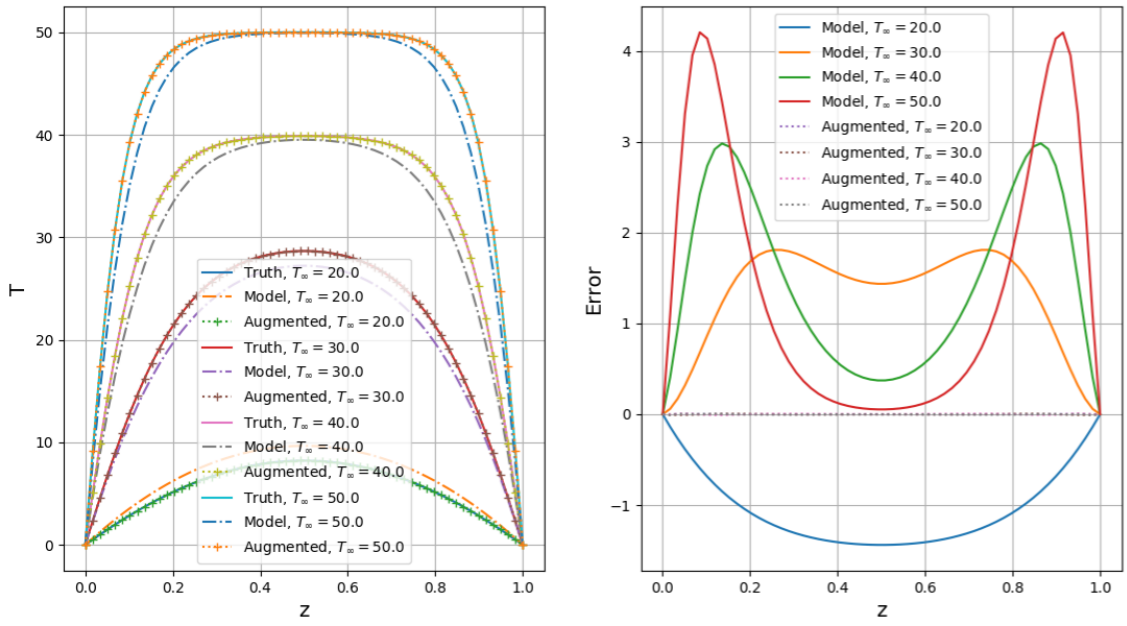

Figure 5.17: FIML-Direct correction, $\beta(T, T_\infty)$, for training using four temperatures.

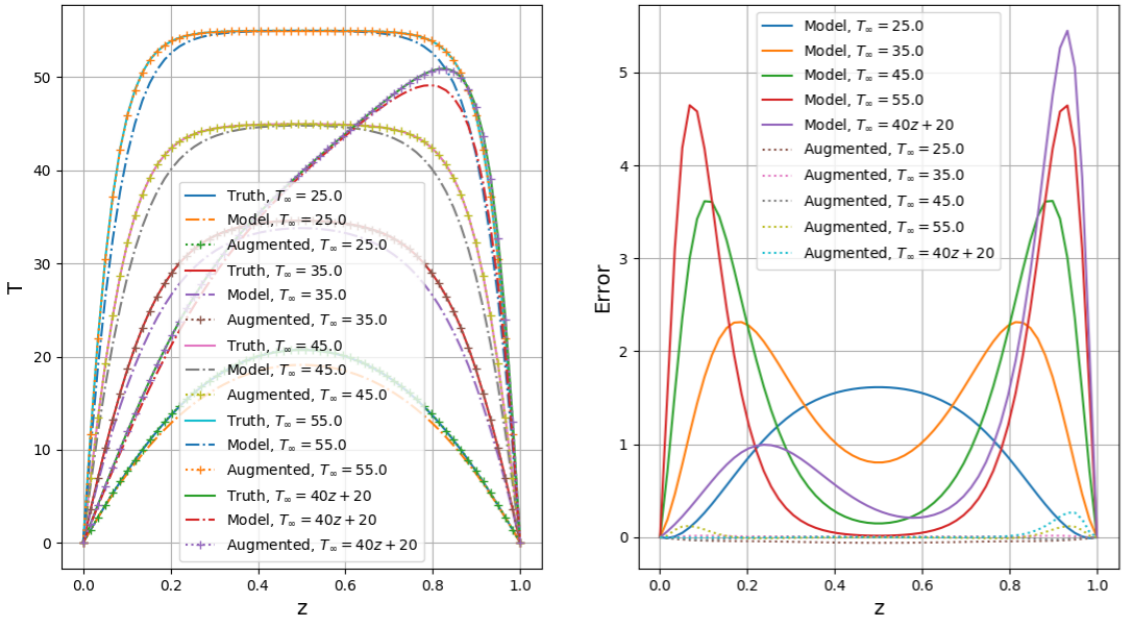

Figure 5.18: FIML-Direct results of model augmentation for four unseen $T_\infty(z)$ distributions.

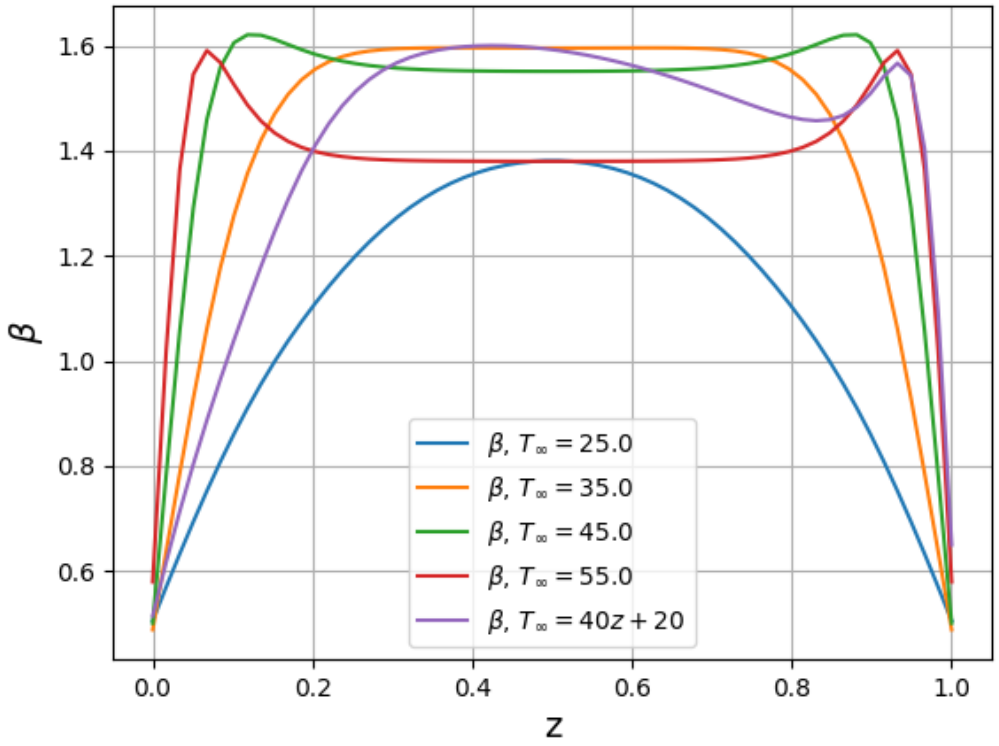

Figure 5.19: FIML-Direct correction, $\beta(T, T_\infty)$, for holdout T_∞ distributions.

ing/inversion on numerous cases when limited computational resources are available, as the gradient computation is substantially more efficient. For four training T_∞ distribution 8 simulations are required to compute the full gradient via adjoint methods (4 Direct + 4 Adjoint simulations). For the stochastic gradient descent demonstration on the 1D heat equation only a single case was considered the batch for the gradient computation. Therefore each gradient computation to update the weights was computed with the cost of two simulations (1 Direct + 1 Adjoint). Note that without the full gradient advanced gradient descent methods (like BFGS) should not be used for the direction finding, so a small step in the steepest descent

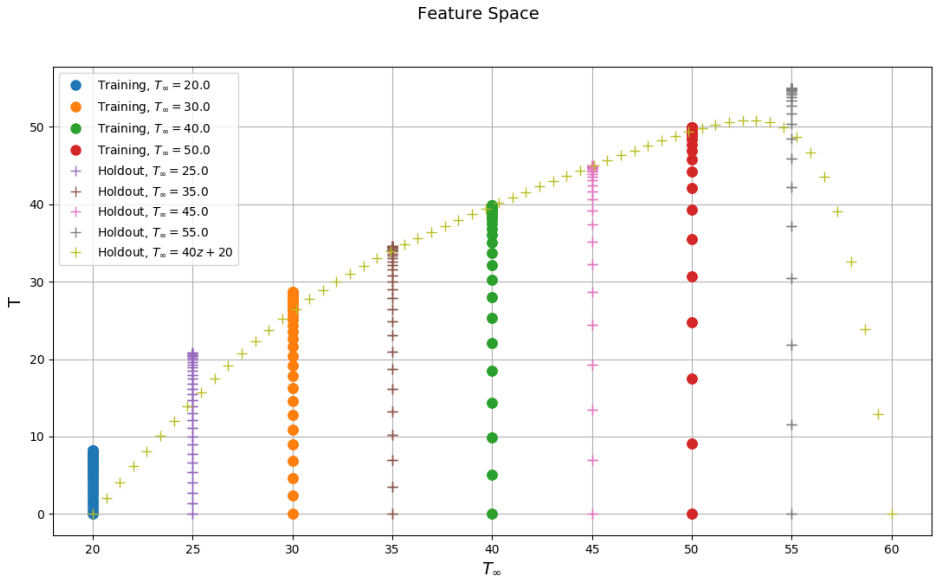

Figure 5.20: FIML-Direct feature space depiction for training and holdout $T_\infty(z)$ distributions.

direction is taken after the gradient evaluation for the stochastic descent method. It is expected, therefore that many more steps will be required for the stochastic implementation in order for an equivalent drop in the objective function compared to the full gradient approach, however, each step is far more efficient. Additionally, the computation of the full gradient is an embarrassingly parallel problem (since the computation of the partial gradient for each case does not require any information from the other cases), so given enough computational resources there should be no need to avoid the computation of the full gradient. The results of the model augmentation for the training temperatures after using stochastic gradient descent for the inversion/training are shown in Figure 5.21.

Similar to the FIML-Direct results when using the full gradient, the resulting model augmentation from using the partial gradient shows substantial improvement

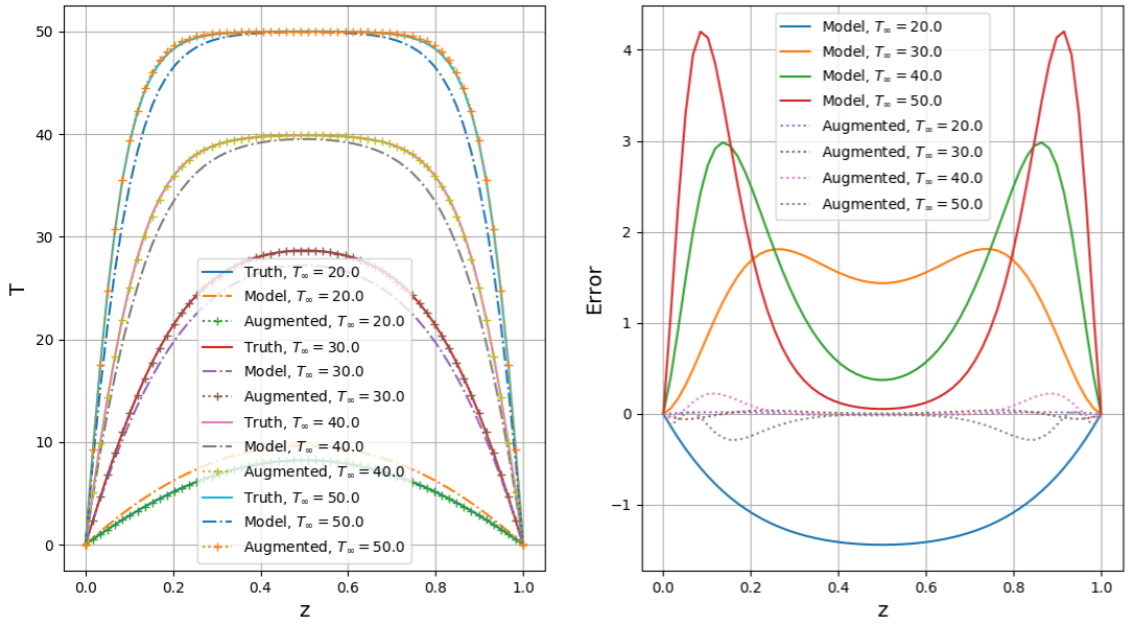

Figure 5.21: FIML-Direct stochastic gradient descent results for training temperatures.

on the holdout cases as (Figure 5.22). As expected, the convergence rate when using the full gradient is substantially better as compared to only using the partial gradient as shown in Figure 5.23. However, each evaluation for the SGD method is roughly 4 times more efficient than using the full gradient for this application.

5.4.1 Inversion Convergence Comparison

The convergence histories for the cases discussed for all three methods are shown in Figure 5.24. First note that because of the imperfect convergence of the backpropagation algorithm for FIML-Embedded and random initialization of the weights for FIML-Direct, the objective function values (J_c, J_e, J_d) are not equivalent on the first evaluation. The objective function values for the multi-temperature

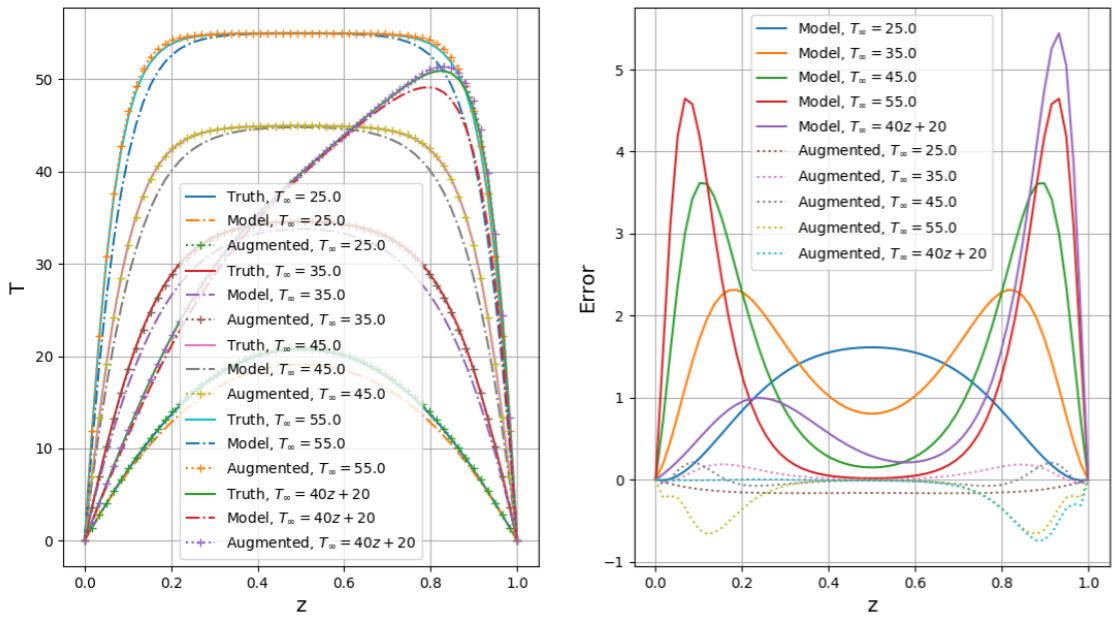

Figure 5.22: FIML-Direct stochastic gradient descent results for holdout $T_\infty(z)$ distributions.

inversions are normalized by the number of T_∞ distributions for comparison to the single temperature cases.

Note that all methods show strong convergence of the objective function (a drop of at least a few orders of magnitude). The FIML-Classic convergence of J_c is by far the best of all the methods tested. This is expected, as the FI-Classic approach does not account for the limitations of the chosen machine learning algorithm in the inversion process. The FIML-Embedded convergence indicates a successful augmentation, but is the worst compared to the other approaches. This is likely due to the imperfect convergence of the backpropagation algorithm that is limiting the achievable accuracy. As noted previously the robustness of the FIML-Embedded approach remains an issue, and the added complexity of the FIML-Embedded pro-

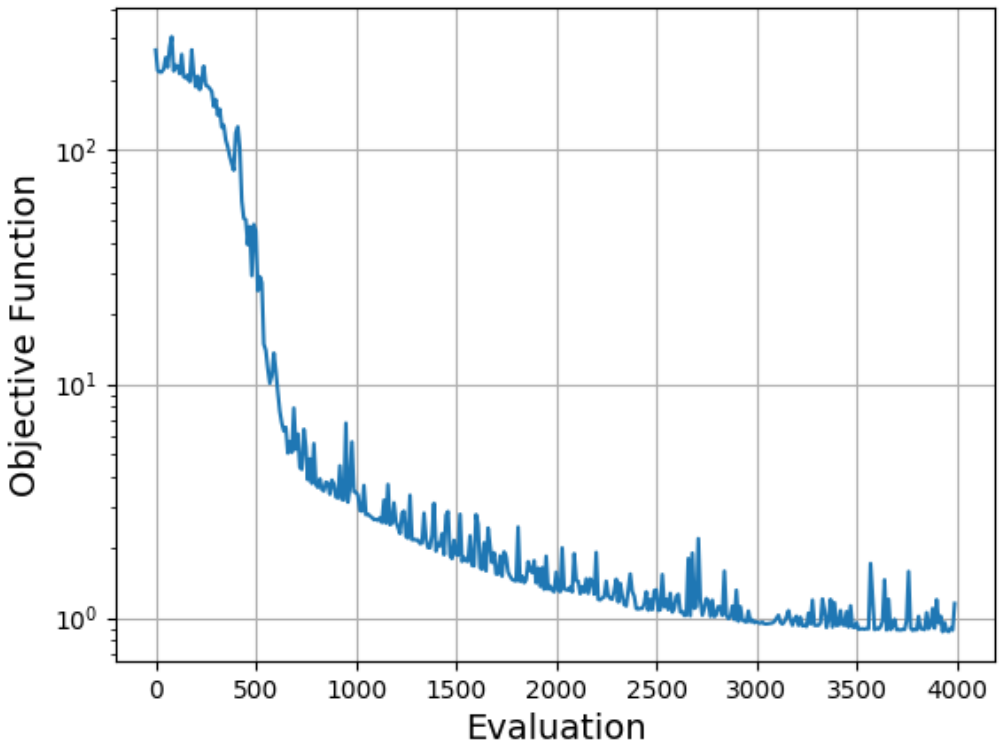

Figure 5.23: FIML-Direct stochastic gradient descent objective function J_d convergence history.

cedure is a drawback of this algorithm. Most remarkable is the convergence of the FIML-Direct cases. For a single temperature the FIML-Direct algorithm converges roughly six orders of magnitude; this is excellent convergence considering the algorithm is training the neural network in the inversion process. As more cases are considered the convergence is somewhat adversely affected when using the FIML-Direct approach, but the objective functions still show a drop of over five orders of magnitude. Also note that all methods show strong convergence early in the inversion history indicating that successful augmentations could still be obtained if

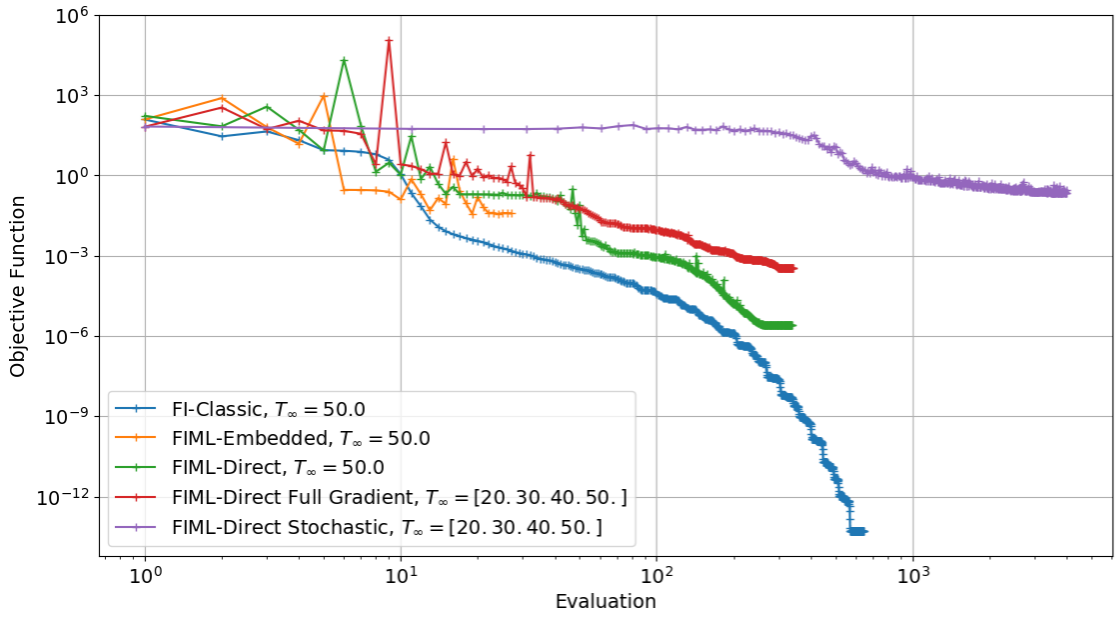

Figure 5.24: Convergence history of the objective functions for all three methods.

computational cost considerations preclude hundreds of inversion iterations. The FIML-Direct stochastic convergence is by far the worst on a per-iteration basis, but because each evaluation is efficient it is possible that the stochastic approach would be advantageous when considering more cases than considered here.

5.5 Summary

Results were presented for all three FIML methods on the 1D heat equation model problem. It was shown that all three methods can generate augmentations that correct the model for the training temperatures. The FIML-Classic and FIML-Direct methods were able to almost perfectly recover the truth β distribution for a single training temperature, while it was shown that the FIML-Embedded approach dramatically reduced the model error but did not converge to the true distribution.

The FIML-Classic and FIML-Direct methods were also demonstrated by using four $T_\infty(z)$ distributions, and it was shown that it was not possible to achieve perfect offline training for the FIML-Direct approach. This results in a lack of consistency between the inversion results and the model augmentation that can result in diminished performance. For this simple implementation, however, nearly perfect model augmentations were demonstrated for the training temperatures for both FIML-Classic and FIML-Direct.

It was demonstrated that the FIML-Direct method can efficiently perform the inversion on multiple cases simultaneously. This is a somewhat surprising result, as FIML-Direct is training a neural network in an unconventional way. FIML-Direct does not train a neural network by the typical approach: by acquiring large quantities of training data and utilizing the backpropagation algorithm (or other approach) to train the network to reproduce the training set. FIML-Direct operates on the weights directly by computing the gradient of the cost function (J_d) with respect to the weights. In this way there really is no training data in the classical sense (large quantity of input/output pairs). The algorithm is simply finding the weights that cause the augmentation to minimize the cost function. The only data that is being utilized is the truth distributions, but the output of the neural network is β and the inputs are the features, $\{T_\infty, T\}$. For FIML-Classic (and typical neural network training) we must assemble a representative dataset of inputs and outputs, but FIML-Direct completely avoids this sometimes cumbersome step. It was anticipated, but not observed, that the nonlinearity of the neural network may complicate the training and give poor inversion results. For the model problem this was not an

issue, as the FIML-Direct approach required a comparable number of iterations to perform the inversion and augmentation as the FI-Classic inversion.

In summary, the 1D heat equation provides a simple environment to evaluate and compare the FIML approaches. Of the two new methods presented in this thesis (FIML-Embedded and FIML-Direct), the FIML-Direct approach shows the most promising results for the model problem. The simultaneous inversion and training of multiple cases provides an exciting novel approach to train neural networks for physics based simulations. As demonstrated by the 1D heat equation cases considered in this Chapter, the FIML-Direct augmentation shows comparable or better accuracy than the FIML-Classic augmentation without the need for offline training. By including the limitations of the neural network in the inversion the modeler guarantees consistency between the inversion environment and the resulting augmentation.

Chapter 6: FIML Results for RANS Applications

6.1 Overview

This chapter presents results for all three FIML approaches on several historically challenging RANS applications. Results for the FIML-Embedded and FIML-Direct approaches for RANS applications are presented here for the first time. The FIML-Classic, FIML-Embedded, and FIML-Direct methods were implemented in a modified version of the Stanford University Unstructured code (SU2) [107] for the first time. This software suite is dramatically more complex than the 1D heat equation simulation in the previous chapter. This demonstrates the robustness and flexibility of the FIML approach, however it also complicates the discussion and conclusions as the physical problem and numerical methods involved are substantially more complex. In the first section, results for simulations concerning the S809 airfoil are presented, and all three methods are demonstrated. Of particular interest are the FIML-Direct results for this application, which demonstrate the capability of this new approach to perform the simultaneous inversion of multiple cases. A new augmentation is constructed from this inversion and it is demonstrated on unseen data: another airfoil, the S814.

To demonstrate that the methods can also improve predictions for compress-

ible cases, all three methods are demonstrated on a transonic airfoil with a shock wave turbulent boundary layer interaction (SWTBLI). Another historically difficult problem is the NASA Langley wall mounted hump [111]. These applications are presented in this chapter to demonstrate performance on a variety of problematic RANS applications. As will be shown in this chapter, the FIML approach demonstrates promising performance on all of these cases, with the 2D airfoil results being the most promising as these results are perhaps the most developed of all the applications presented.

This chapter only presents results for cases where one of the new novel methods (FIML-Embedded and FIML-Direct) was applied. Additional cases performed for only the FI-Classic approach are presented in Appendices A and B, which show results for a hypersonic wedge SWTBLI application and a 3D Onera M6 wing.

6.2 S809 Airfoil

The S809 airfoil is a thick airfoil designed for wind turbine applications. As discussed in Chapter 1, RANS predictions on 2D airfoils at a high angle of attack often suffer from an overprediction in eddy viscosity. This results in a boundary layer more resistant to separation than observed in experiments, leading to an overprediction in lift coefficient at angles of attack sufficient to result in separation on the suction surface [15, 3]. In general the SA model overpredicts eddy viscosity for adverse pressure gradients as noted by Spalart and Allmaras [17], and regions where the SA model is anticipated to perform poorly are illustrated in Figure 1.3.

The S809 airfoil was considered for all three FIML approaches. The data chosen to improve the baseline model was the experimental lift coefficient [16] ($k_d = C_{L_{exp}}$) for all cases. In the following sections first the results for the augmentation for a single angle of attack (single case) are presented for each method. Additionally, a discussion of the features selected for the neural network augmentation is presented. Finally, results for the simultaneous inversion of multiple cases using the FIML-Direct algorithm are presented. This augmentation, only trained using S809 data, is then tested on another 2D airfoil, the S814.

6.2.1 FI-Classic

The Classic method field inversion "FI-Classic" was performed on the S809 airfoil at $\alpha = 14.2°$ angle of attack. The baseline SA-BC model was run for a number of angles of attack and compared to the experiment. Note that the SA-BC transition model was used for all of the S809 airfoil cases as the natural transition experimental data was used. As shown in Figure 1.2, the lift coefficient is substantially over-predicted by the baseline SA-BC model, indicating that there is room for improvement that could be made by incorporating the information from the data into an augmented model.

The objective function for the FI-Classic inversion is given by 6.1, where n is the number of points in the computational domain:

$$J_c(\beta) = (C_{L_{exp}} - C_L)^2 + \frac{1}{2}\lambda \sum_{i=1}^{n} (\beta_i - 1.0)^2 \tag{6.1}$$

For this angle of attack we have $C_{L_{exp}} = 1.0546$ and $\lambda = 1.0 \times 10^{-4}$. Remember the regularization constant, λ, represents the confidence in the baseline solution relative to the data, such that higher λ indicates more confidence in the model. In this case λ was chosen to be similar to that explored by Singh [42]. To show the effect of the regularization constant values for the objective function without any regularization constant ($\lambda = 0$) will be shown and denoted J_c' such that $J_c \geq J_c'$.

The baseline SA-BC model solution (no correction, $\beta = 1.0$ everywhere) is shown in Figure 6.1. There is a large separated region at this angle of attack indicated by the recirculating streamlines. This solution corresponds to the first evaluation of J_c in the minimization.

Because there is a β at every node of the domain the gradients can be easily visualized. The gradients found by the adjoint solver for the first evaluation ($\beta = 1.0$ everywhere) are shown in Figure 6.2. Note that most of the design variables are inactive, meaning that the gradients are equal to 0, they do not affect the experimental lift coefficient, and therefore will not be modified by the optimization algorithm during the minimization. The remaining gradients are almost universally positive, meaning that the downhill direction is to decrease production on the upper surface of the airfoil, as expected.

The convergence history of the FI-Classic algorithm for the S809 airfoil is shown in Figure 6.3. Each "evaluation" represents a solution for a chosen $\beta(x)$ correction field. A direct (primal) and adjoint solution is found for each evaluation to find $J_c(\beta)$ and $dJ_c/d\beta$. Note that the convergence for this case is very good, with a substantial decrease in the lift coefficient error in a few iterations. The right side of

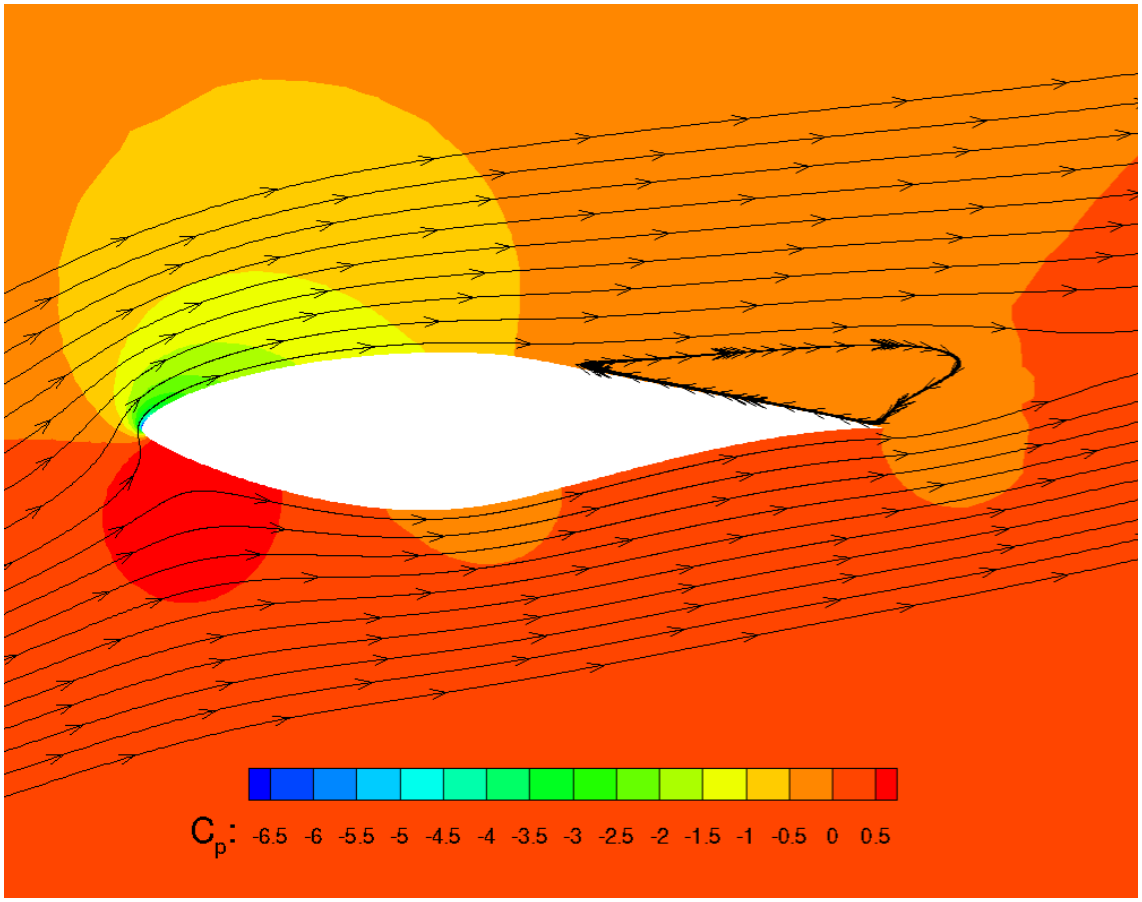

Figure 6.1: Baseline SA-BC streamlines and C_p contours for S809 airfoil.

Figure 6.3 shows good convergence and also shows the purpose of the regularization constant. Note that J_c' shows a three order of magnitude drop in the objective function for evaluation 3, however J_c is much higher indicating that the correction for evaluation 3 is excessively large. The minimum for J_c occurs on evaluation 11, which is a much less intrusive correction that still substantially corrects the lift coefficient. To further evaluate the inversion convergence history, the L2 norm of the gradient, $\|\nabla J_c\|_2$, is shown in Figure 6.5. Note that the magnitude of the gradient is substantially reduced throughout the inversion, exhibiting the expected behavior for convergence to a local minimum.

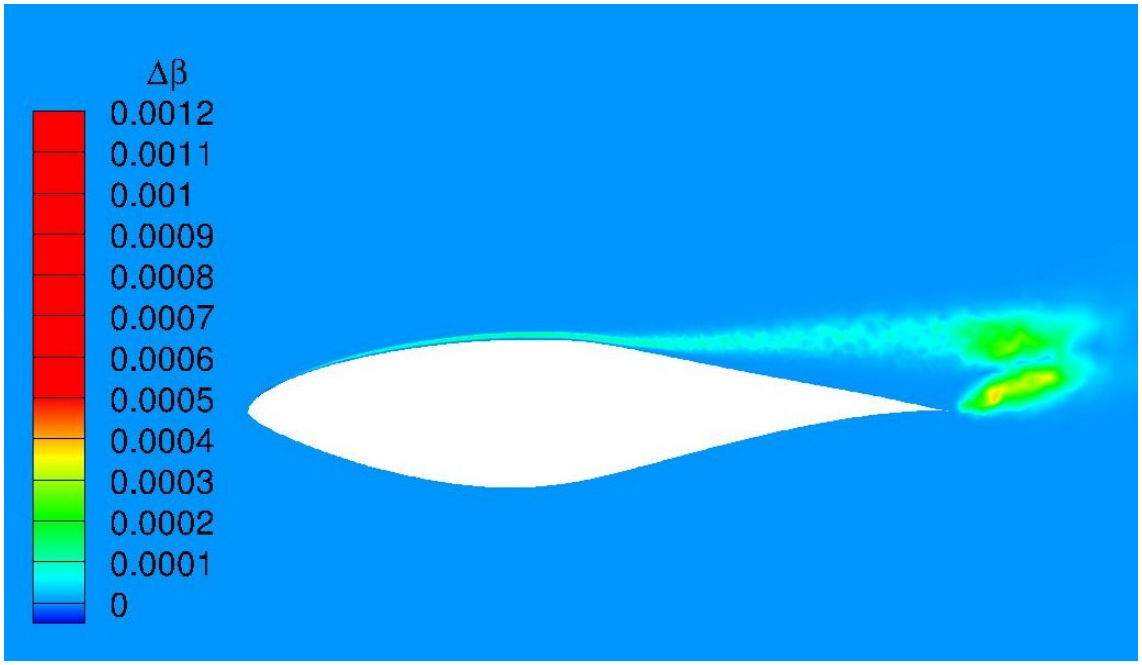

Figure 6.2: Initial gradient for FI-Classic.

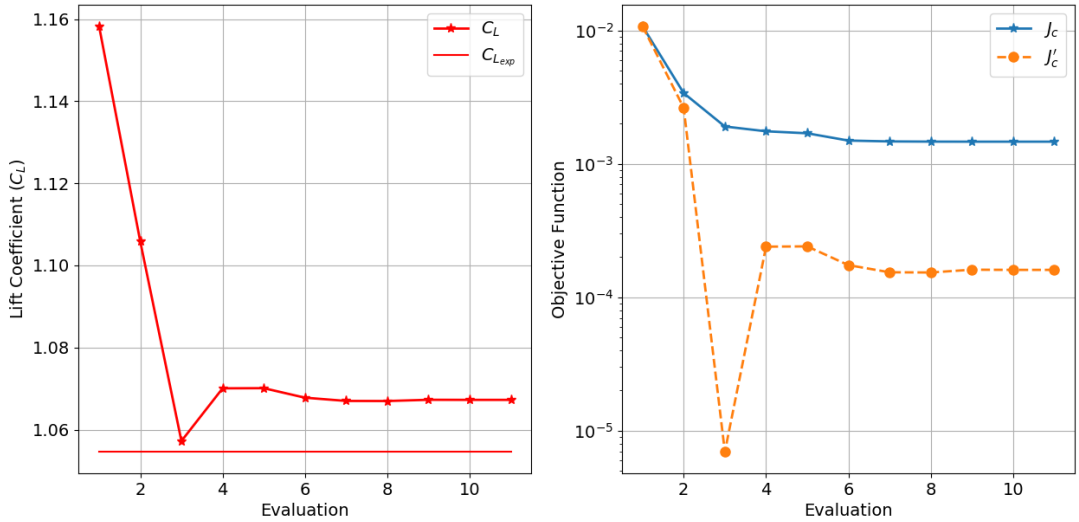

Figure 6.3: Convergence history of S809 airfoil for FI-Classic.

The optimal correction field, found in evaluation 11, is shown at the top of Figure 6.11. Note that this correction is rather substantial, and over a limited

domain in space. The sudden changes in β will complicate the determination of a function to reconstruct it from local flow features. At this point, the inversion is complete, but no learning has occurred. There remains the process of taking the spatially correlated correction field, and learning a function to reproduce it: $\beta(x) \rightarrow \beta(\eta)$.

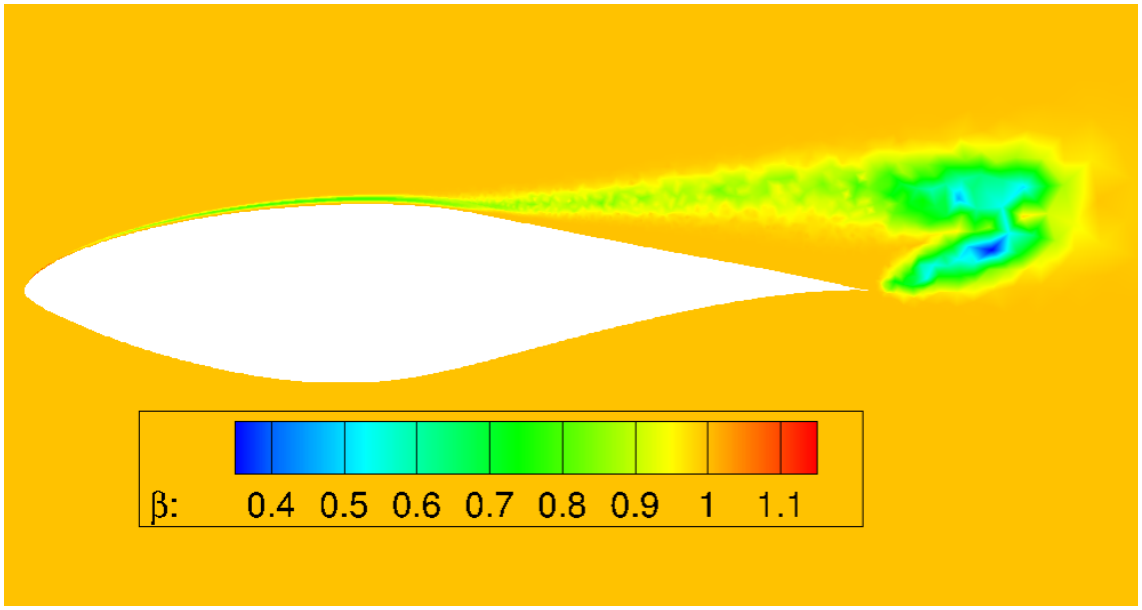

Figure 6.4: Optimal correction field found by FI-Classic.

6.2.2 Feature Selection and Scaling

The methodology used to examine potential features, η, is discussed in this section. In order to successfully train a neural network that can reproduce the correction field, the modeler must select appropriate features that exhibit a strong functional relationship to the output. Ideally these features will be local and easily computed in the flow solver so it will be available to the model augmentation. To start, the features were visualized with scatter plots and correlation coefficients. A

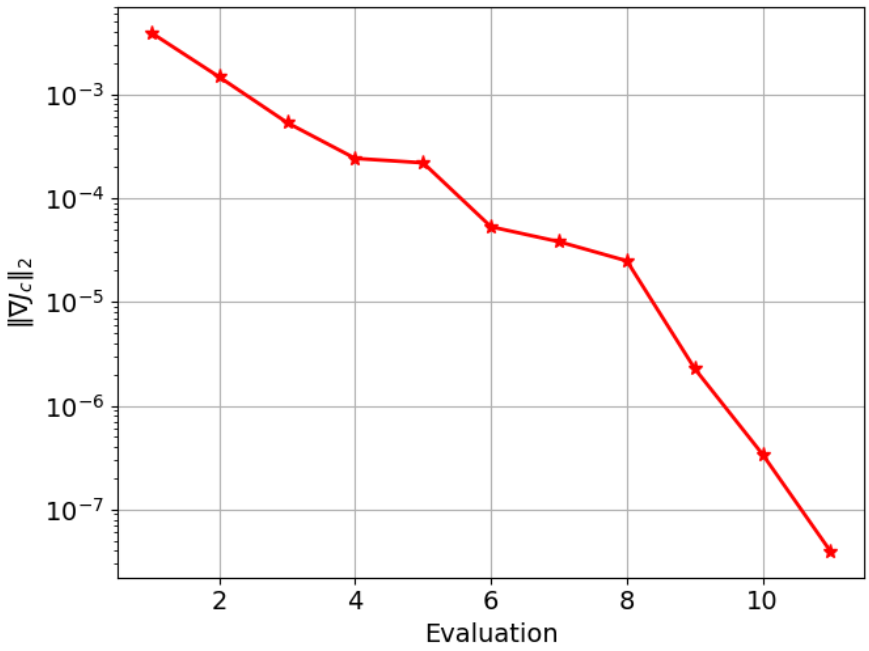

Figure 6.5: Magnitude of the gradient vector of the objective function during the inversion for FI-Classic.

large set of features already used in the SA turbulence model were examined. Intuitively, these are quantities that have already been identified by previous researchers as important to accurately model the eddy viscosity, and therefore it is likely that a subset of these features will be adequate to produce a function to predict the correction to the production term.

Figures 6.6 and 6.7 shows the scatterplot and correlation coefficient plot of a large subset of the features considered during this effort. P/D is the (dimensionless) ratio of the production to destruction terms of the SA model. $\hat{S}$ is the SA model variable. $|S|/|\Omega|$ is the ratio of the strain to vorticity magnitudes. δ is the ratio of the local turbulent strain to the shear stress at the wall that was also used by Medida

[3] in his PhD Thesis to create an adverse pressure gradient correction to the SA model 6.2. f_w is the SA wall function variable. The features have been scaled to unit mean and standard deviation. First note that some of the considered features have extreme outliers. As noted previously this can complicate neural network training. Additionally, the correlation coefficients are low (with the possible exception of δ), indicating that the features do not show a strong linear correlation to the correction.

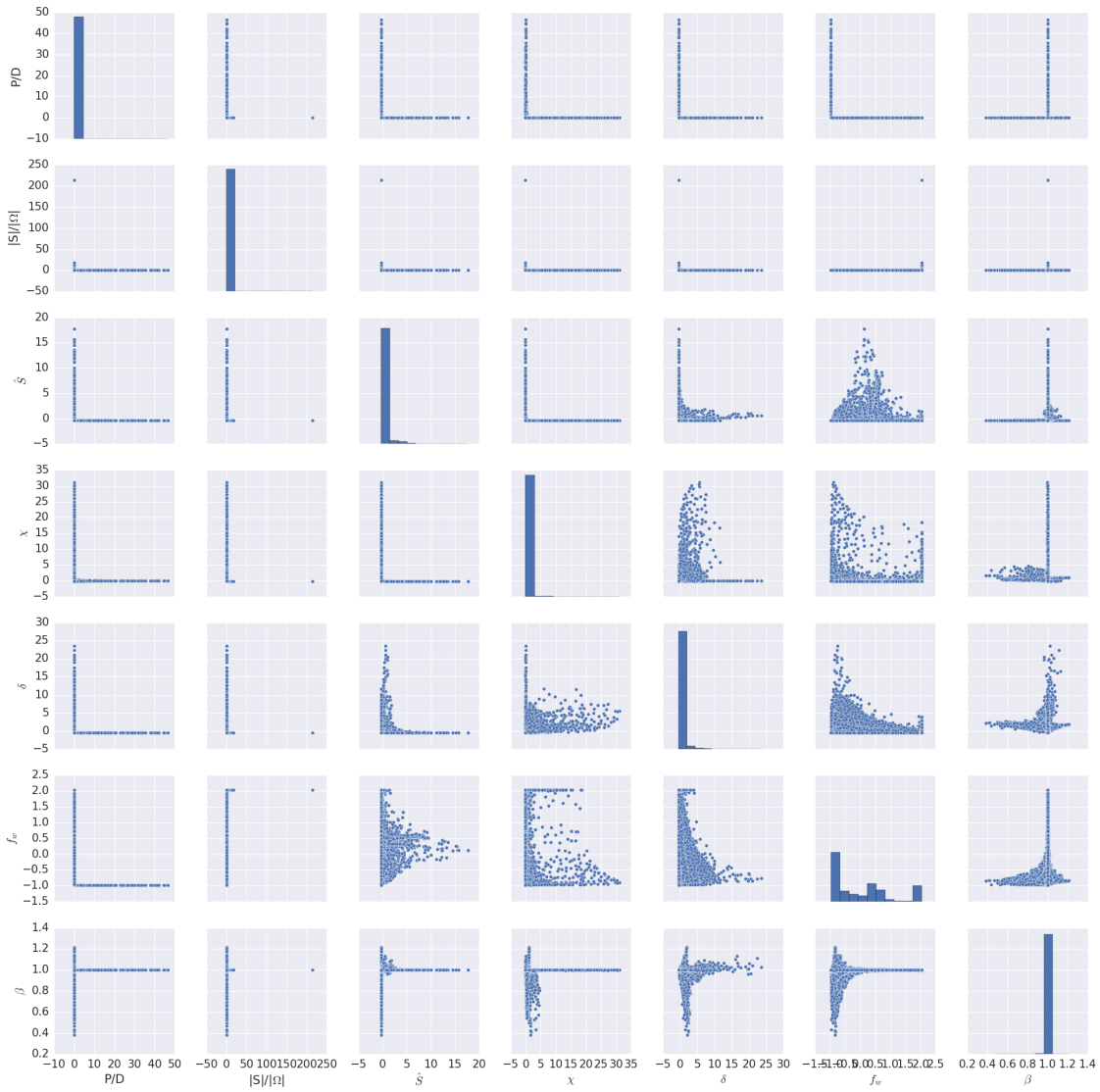

Figure 6.6: Scatterplot pairs for some considered features.

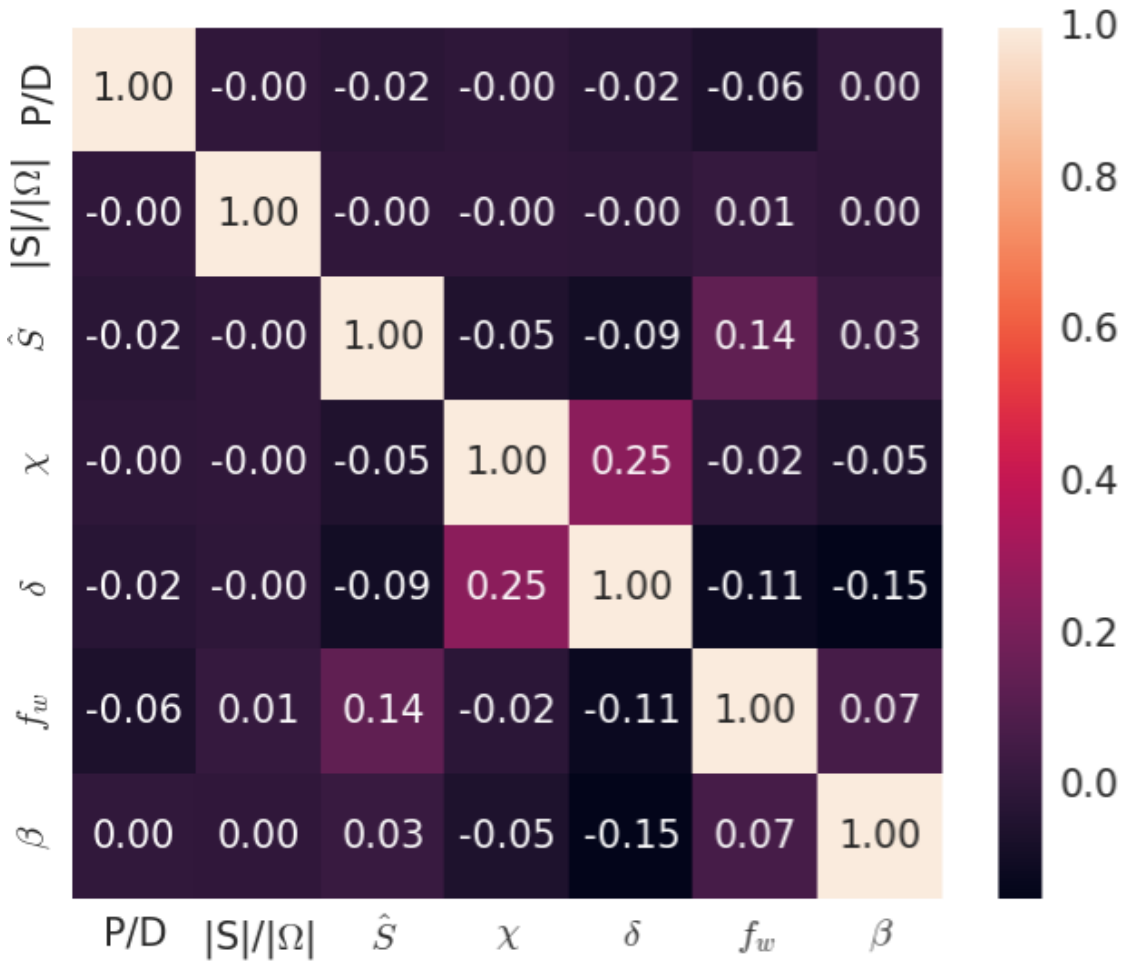

Figure 6.7: Correlation coefficient pairs for some considered features.

$$\delta = \frac{\mu_t |S_{ij}|}{1.5\tau_w} \tag{6.2}$$

A number of scaling operations were considered in order to account for large outliers. The most successful of these is shown in Figures 6.8 and 6.9. In this scaling the features have been mapped to a normal distribution. Note that the outliers have been dramatically reduced, and the correlation coefficients have also increased substantially. In practice, mapping the features to a normal distribution produced far better augmentations than scaling to unit mean and standard deviation due to

the more robust treatment of outliers. Also note that there is a strong relationship between f_w and the P/D suggesting that perhaps both features do not need to be included in the feature set.

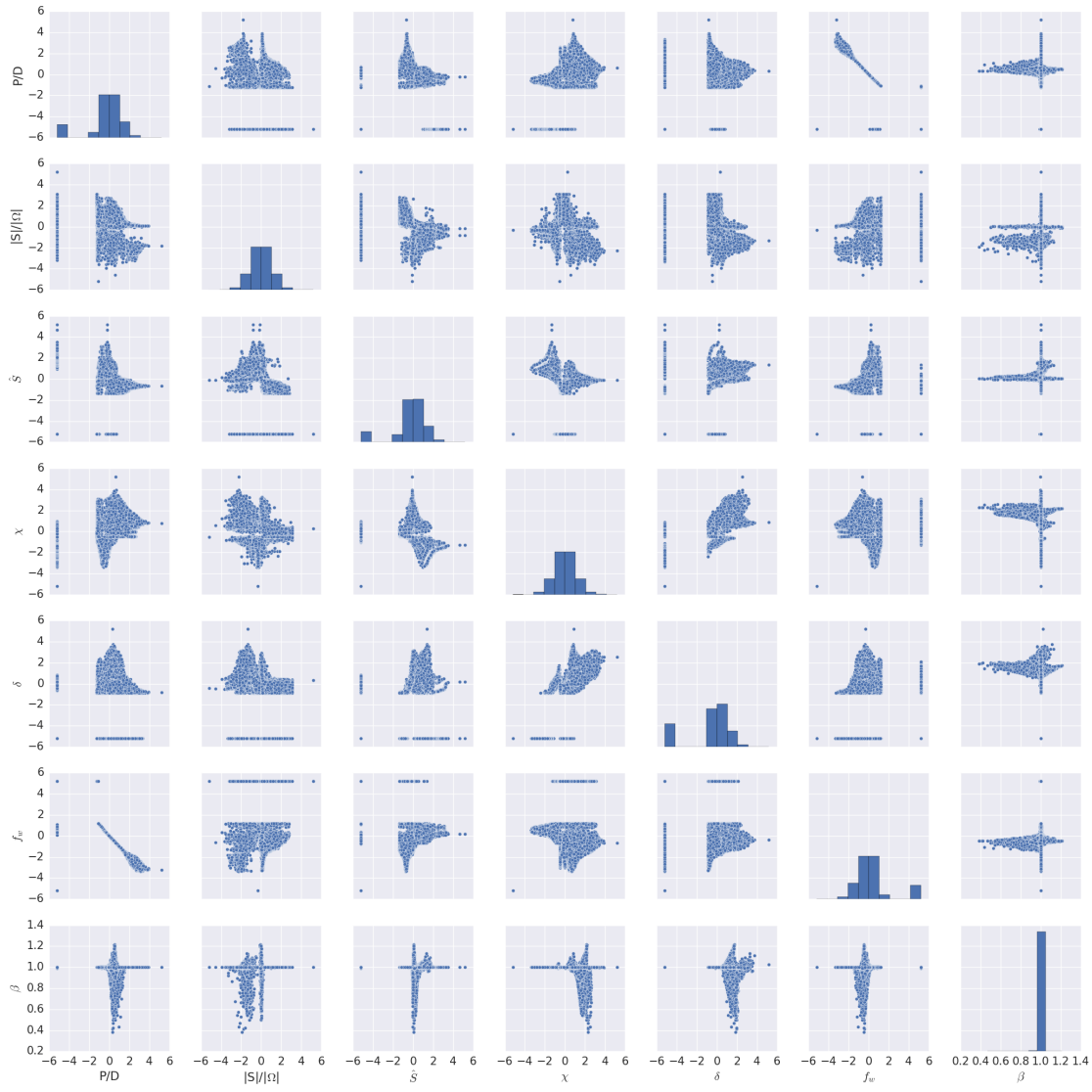

Figure 6.8: Scatterplot pairs for some considered features mapped to normal distribution.

The features were also ranked in order of importance in two ways. First, a random forest regressor was trained on the features. Random forests naturally

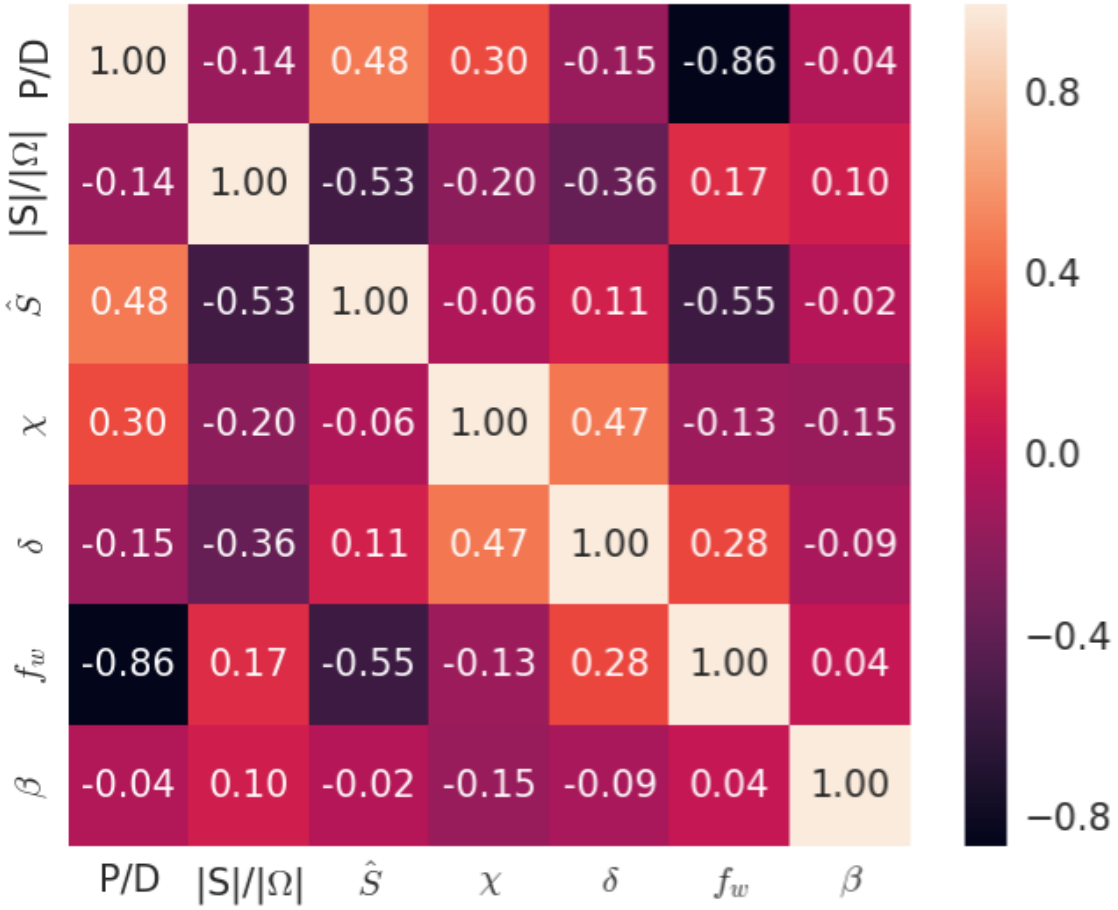

Figure 6.9: Correlation coefficient pairs for some considered features mapped to normal distribution.

produce a feature ranking as part of the learning process. This may give some indication of which features show the strongest functional relationship to the output, but since in this application we are using neural networks it is perhaps not the most appropriate approach. The feature ranking from the random forest regression training in order of most to least important was: 1. δ, 2. f_w, 3. χ, 4. $|S|/|\Omega|$, 5. $\hat{S}$, 6. P/D.

Next, the sequential back selection (SBS) algorithm was performed on the

potential features. This algorithm also ranks features, but in this case the proper learning algorithm is used (neural networks). Note, however, that the results are dependent on the chosen neural network structure, feature scaling, and specific training algorithm so again conclusions are difficult. The feature ranking for the sequential back selection algorithm was: 1. f_w, 2. P/D, 3. δ, 4. $|S|/|\Omega|$, 5. χ, 6. $\hat{S}$. Additionally, the R2 scores for each subset of features were plotted. This can help determine the selection of the optimal number of inputs for the neural network. Networks with too many features (features that do not provide additional useful information) suffer from overtraining that can diminish performance. The scores for each subset of features is shown in Figure 6.10. Note that the network for all 6 features shows a worse score than the 5 feature subset (P/D removed). This is likely because f_w and P/D are highly correlated (Figure 6.9) and therefore only one is needed in the feature set. Also note that the R2 score only starts decreasing for the 2 or 3 feature subsets indicating that perhaps 3-4 features are an appropriate number of inputs for this application, features selection, training algorithm, and neural network structure. Definitive conclusions concerning feature selection for machine learning are difficult to make, because the optimal features are so highly dependent on choices made by the modeler.

Ultimately, four features were selected given by Equation 6.3:

$$\eta = \left[\frac{P}{D},\ \chi,\ \frac{|S|}{|\Omega|},\ \delta\right] \tag{6.3}$$

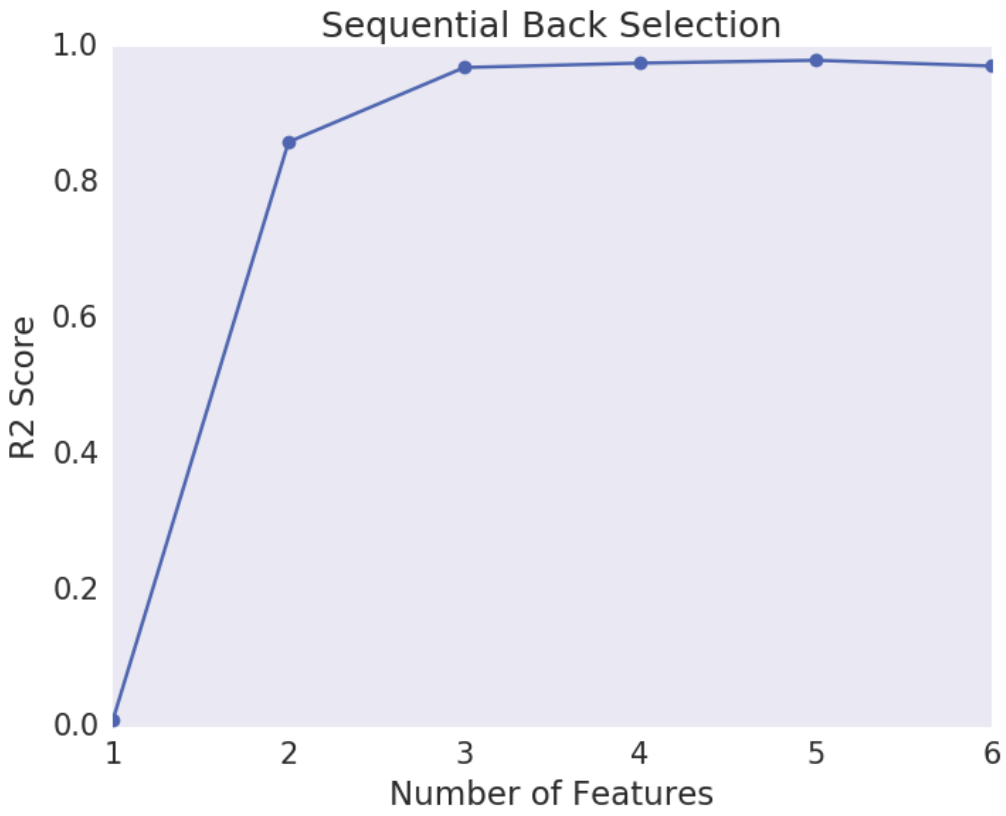

Figure 6.10: R2 scores for each subset of features during sequential back selection.

6.2.3 FIML-Classic

The four features selected from the feature selection section were used to train a neural network augmentation to the SA-BC model. This completes the FIML-Classic augmentation and creates a model that could be used to make predictions for additional (unseen) cases, without the need for the expensive inversion process (and the corresponding need for additional data).

The SciKit-Learn package was used to train the neural network. A rigorous hyperparameter search was not performed however a 3 hidden layer x 20 node network was selected. The network was trained via the L-BFGS-B algorithm, and the

resulting output of the neural network is shown in Figure 6.11.

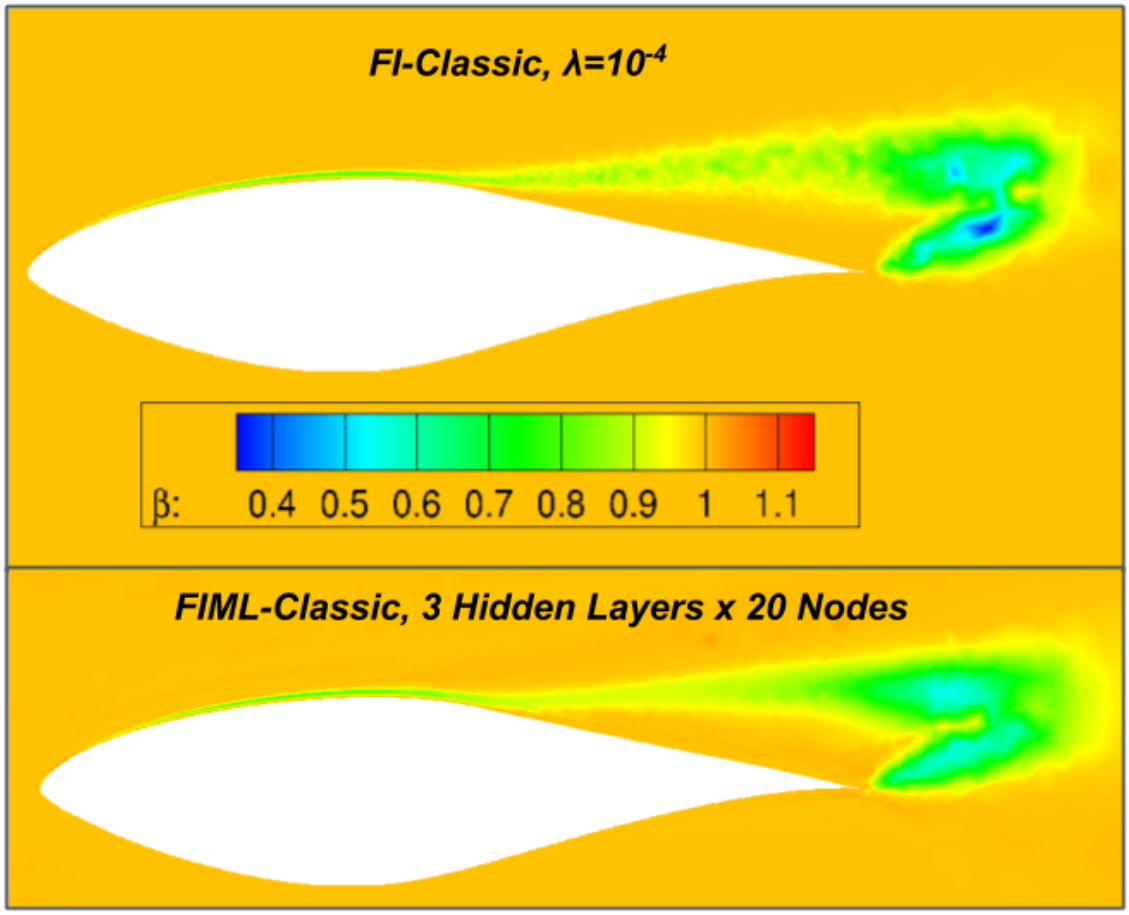

Figure 6.11: Comparison between correction found in inversion and output of neural network following learning.

Clearly, the residual training error shown in Figure 6.11 will have a negative effect on the accuracy of the resulting augmentation. While another neural network topology or other machine learning algorithm could likely perform better than shown [13], no learning algorithm will have perfect training. Therefore, for FIML-Classic there will always be some level of inconsistency between the inversion and the prediction environment. For this case, the penalty due to inconsistent augmentation

	C_L	Error
Target	1.0546	-
Baseline	1.15826	0.10366
Inverse	1.06726	0.01266
Augmentation	1.06914	0.01454

Table 6.1: Comparison of target (k_d), inverse, and augmented model lift coefficients.

is shown in Table 6.1. The penalty in this case is an error in the lift coefficient that is 15% greater than that of the inverse solution. Thus, the achievable performance for this neural network training increases the error by 15%. Fortunately, for this case the residual error is still quite low.

The pressure distributions from the experiment, baseline model, and FIML-Classic augmentation are shown in Figure 6.12. As shown, the improvement in prediction is due to better prediction of the separation location, consistent with the known deficiencies in the RANS models discussed previously. Note that for the FIML-Classic augmentation only the only higher fidelity data used was the experimental lift coefficient at 14.2° angle of attack: a single number. Despite this limited data set the augmentation has corrected the pressure distribution, indicating that the augmentation is making a physically relevant prediction.

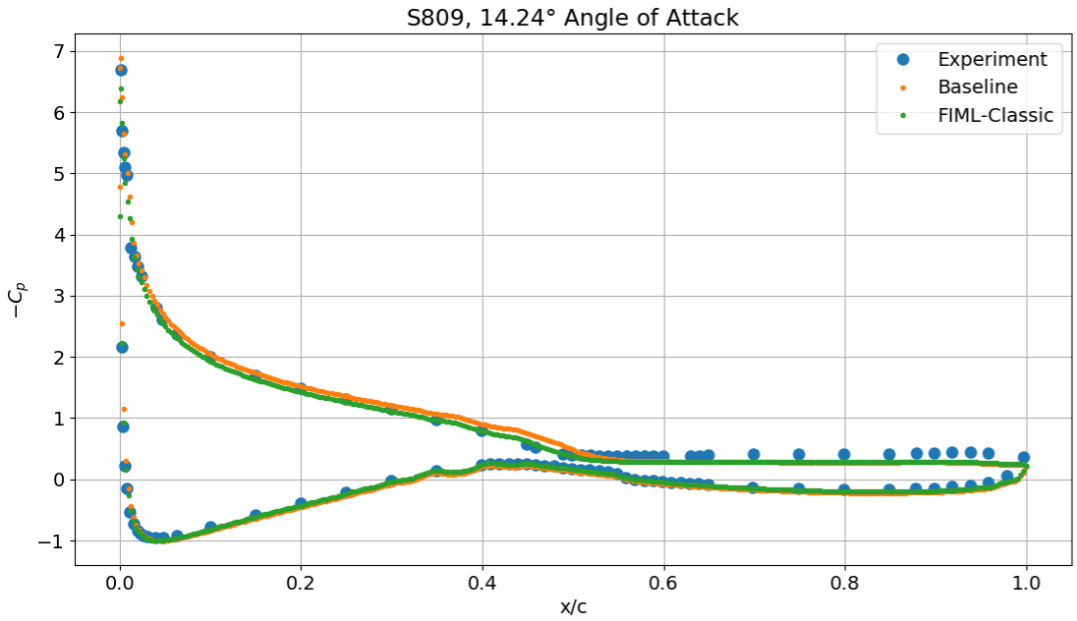

Figure 6.12: Pressure distribution plot showing improvement in prediction for the FIML-Classic augmentation.

6.2.4 FIML-Embedded

The FIML-Embedded approach was performed on the same application (S809 at 14.2° AoA). The same features were used however in this application a 2 hidden layer x 20 node neural network was used. A two hidden layer network proved sufficient to minimize the loss function in the flow solver. Again the hyperbolic tangent activation function was used. The features were again mapped to a normal distribution, however the method implemented in SU2 to accomplish the mapping is different than that implemented in the SciKit-Learn package [92]. The SU2 implementation computes the estimated distribution function of the feature by binning then applies the inverse cumulative distribution function of a normal distribution as described in Chapter 4. The objective function for the FIML-Embedded inversion

is given by 6.4:

$$J_e(\beta_T) = (C_{L_{exp}} - C_L)^2 + \frac{1}{2}\lambda \sum_{i=1}^{n} (\beta_{T_i} - 1.0)^2 \tag{6.4}$$

The minimization history of the cost function ($\min_{\beta_T} J_e(\beta_T)$) is shown in Figure 6.13. Note that the lift coefficient error has been substantially reduced as shown on the left hand side figure. On the right, the regularized cost function has been substantially reduced (J_e). The lowest value of J_e was obtained on evaluation 6.

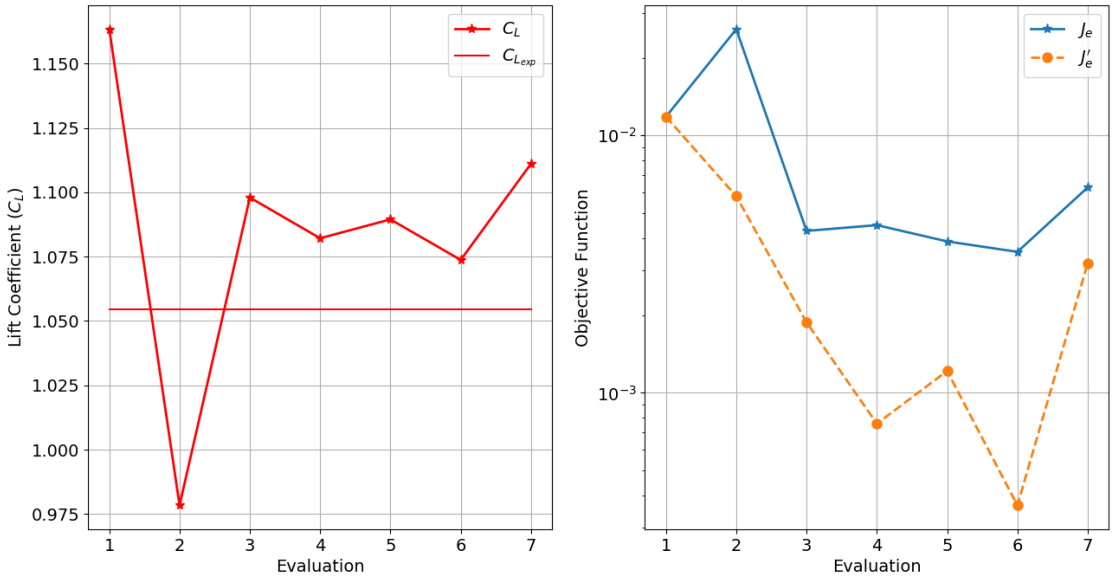

Figure 6.13: Convergence history of S809 airfoil field inversion.

As noted previously, a drawback to the FIML-Embedded approach is the additional complexity required to perform the simultaneous minimization associated with the backpropagation algorithm (minimize the SSE loss function) inside the flow solver, and the minimization of J_e. In practice, this sometimes resulted in poor convergence inside the flow solver. Improving the robustness of the FIML-Embedded

implementation remains an area for future work, especially for RANS applications. For this case, the robustness issue prevented further convergence of the minimization algorithm and the inversion terminated prematurely. This almost certainly is limiting the achievable performance of this FIML-Embedded case, so direct comparison to the other FIML algorithms is not appropriate until the robustness issues are resolved. Nevertheless, for this application Figures 6.14 and 6.15 show the lift convergence and SSE loss function convergence history for evaluation 6 indicating that the flow solver and backpropagation algorithm are showing sufficient convergence for this case. Therefore, regardless of the premature inversion termination, because the objective function is substantially smaller than the initial design, and the backpropagation algorithm converged, this design is valid and represents a successful augmentation that improves the prediction of the model. Additionally, Figure 6.16 shows shows the optimal β_T and β correction fields. Recall that β_T are the design variables for the optimizer, and β is the correction applied to the production term and also the output of the neural network being trained by the backpropagation algorithm.

Note that in Figure 6.16 the neural network training is satisfactory, but not perfect. The β field is slightly more dispersed and generally of lower magnitude indicating increased regularization over the β_T field. This is due to imperfect learning in the backpropagation algorithm and is expected. Since this training error is now inside the inversion, it is accounted for in the optimization process. This ensures that the optimal correction can be learned. Also, unlike FIML-Classic, this implementation trains a neural network in the inversion and therefore guarantees

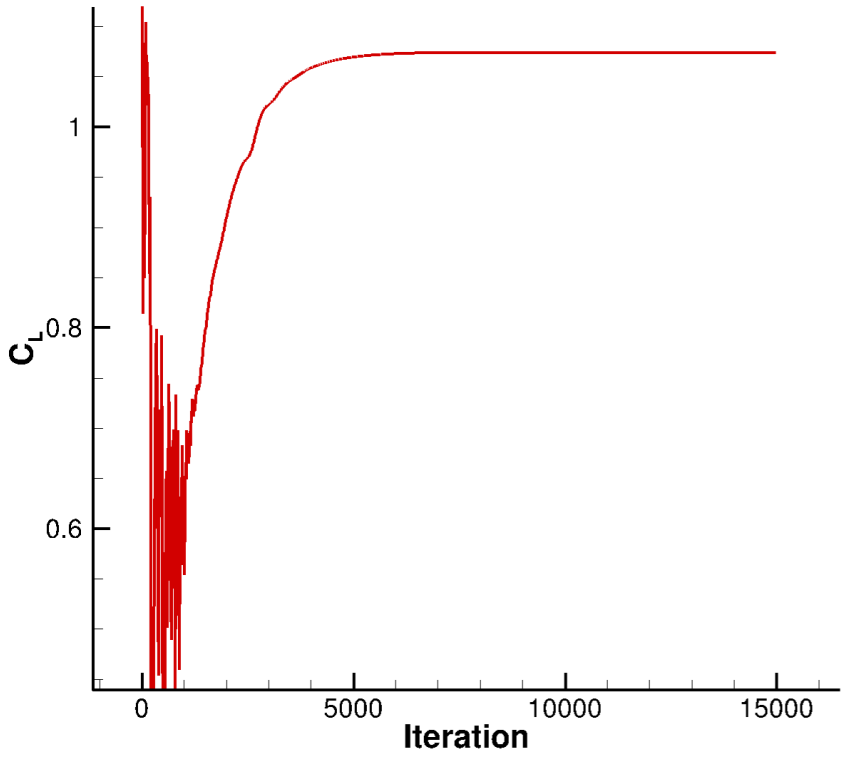

Figure 6.14: C_L convergence at minimum J_e evaluation.

consistency between the inversion solution and the resulting augmentation.

6.2.5 FIML-Direct

The FIML-Direct algorithm was also performed for the same case (S809 at 14.2° AoA). Compared to the FIML-Embedded implementation the FIML-Direct approach is substantially more straightforward. The weights of the neural network are now the design variables and are set by the optimizer at each $J_d(w)$ evaluation. The weights remain fixed for each evaluation. The forward propagation algorithm is still performed each flow solver iteration to find the current correction $\beta(w)$. For this application the same features were used, however a smaller network was

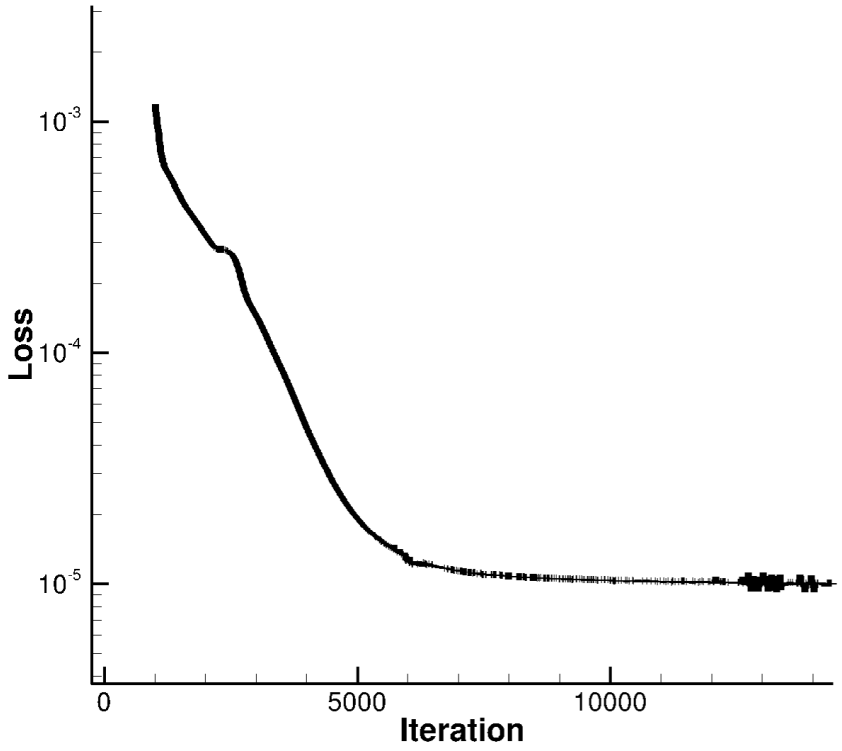

Figure 6.15: SSE convergence at minimum J_e evaluation.

selected using a single hidden layer with 20 nodes. More layers were tested but did not result in improved performance for this case. The objective function for this FIML-Direct case is given by 6.5 and initially $\lambda = 1.0 \times 10^{-4}$. Note that unlike the other methods, the gradient cannot be visualized in the same manner because the design variables are no longer defined at each node, but are now the weights of the neural network. Despite being not easily visualized, FIML-Direct substantially lowers the dimensionality of the inversion for this application (number of weights much less than the number of nodes in the computational domain). Due to the many fewer design variables the limited memory BFGS variant (L-BFGS-B) algorithm was not necessary and the BFGS algorithm was used. The convergence of $J_d(w)$ during the inversion/optimization procedure, $\min_w(J_d(w)$, is shown in Figure 6.17.

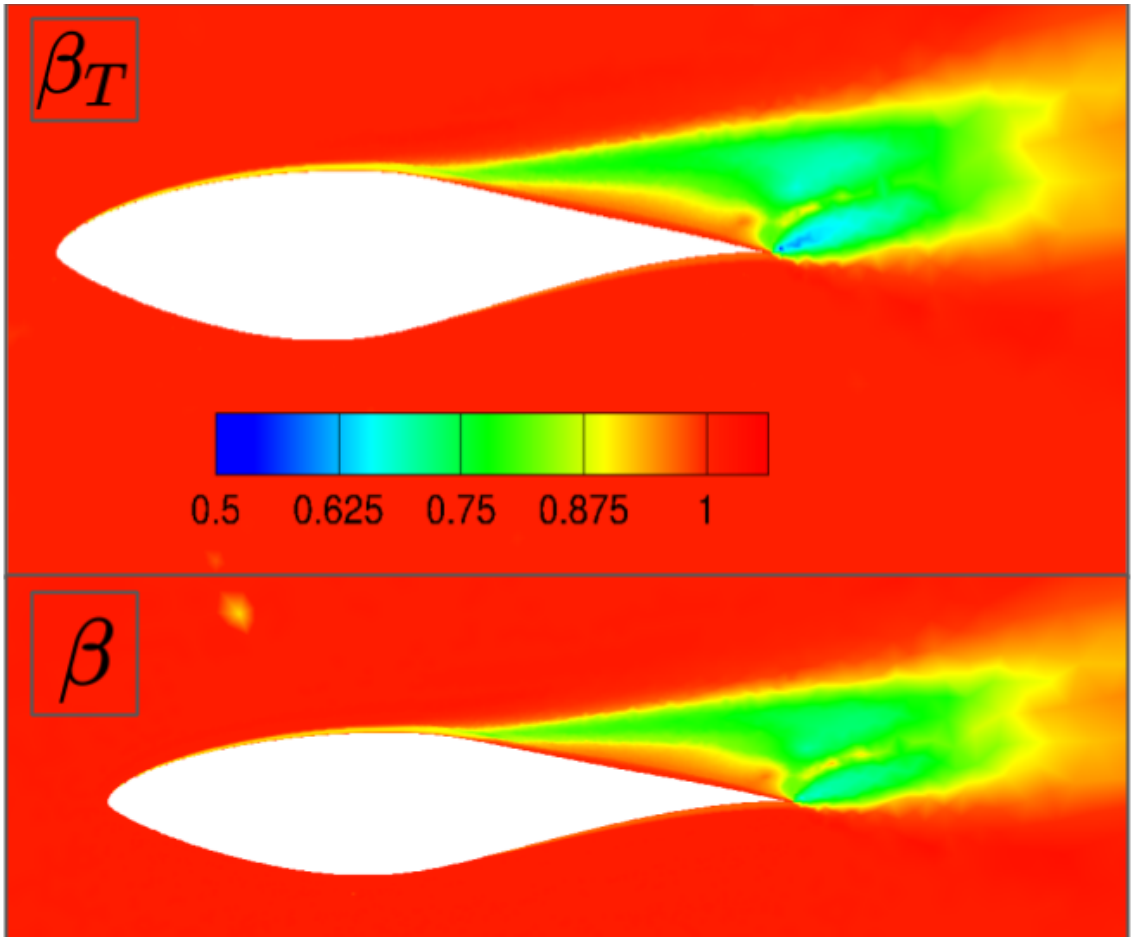

Figure 6.16: FIML-Embedded comparison of training correction β_T and correction output by converged neural network augmentation β.

$$J_d(w) = (C_{L_{exp}} - C_L)^2 + \frac{1}{2}\lambda \sum_{i=1}^{n} (\beta - 1.0)^2 \tag{6.5}$$

Note that the convergence shown in Figure 6.17 is relatively poor compared to FIML-Classic and FIML-Embedded. However, the FIML-Direct approach is resulting in substantially more regularization than the other methods, as shown in Figure 6.18. It is therefore appropriate to lower the regularization constant because the required regularization is being applied by the limitations of the neural network

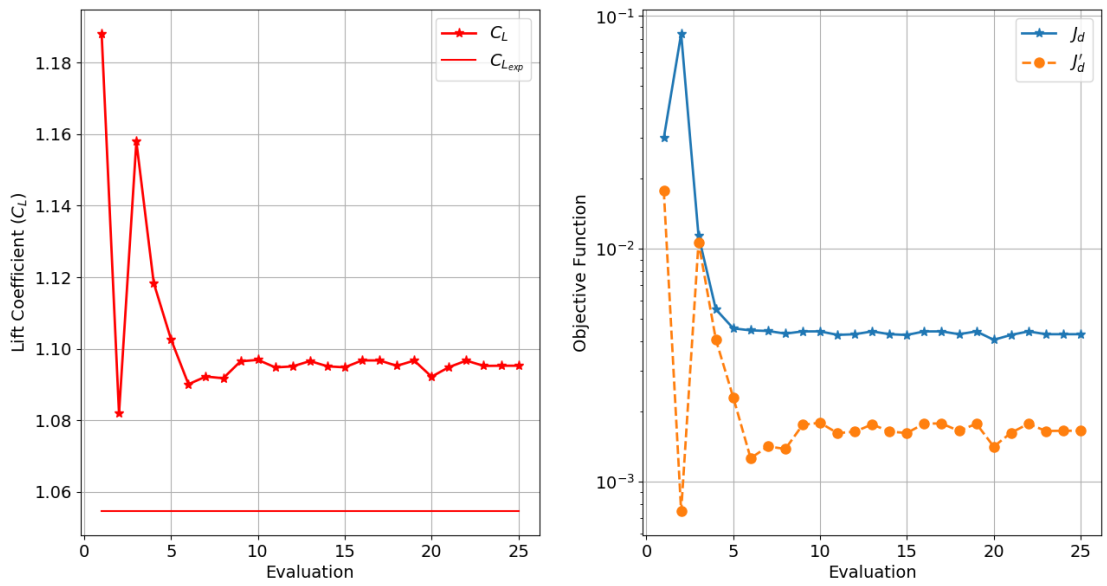

Figure 6.17: Convergence history of S809 airfoil FIML-Direct inversion for $\lambda = 10^{-4}$.

in this approach.

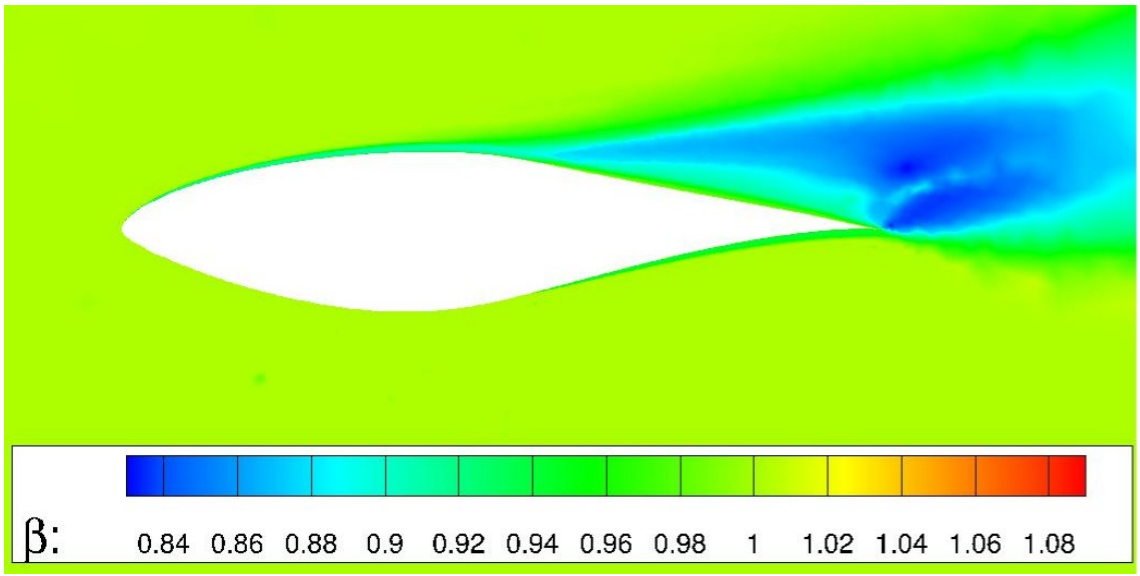

Figure 6.18: Correction field (β) for FIML-Direct with $\lambda = 10^{-4}$.

Figures 6.19 shows the correction for the same case with $\lambda = 10^{-5}$. Note that the correction has a larger magnitude, and the residual error in the lift coefficient

(discussed next section) is much lower with the decreased regularization constant. It is possible to lower the regularization constant for the other methods, with the risk of the increased difficulty of learning the de-regularized correction. Unlike the convergence history shown in Figure 6.17 for the $\lambda = 10^{-4}$ FIML-Direct case, the $\lambda = 10^{-5}$ results did not demonstrate strong (asymptotic) convergence during the inversion. However, the achieved minimum gives a much closer match to the data than the higher regularization constant solution, as expected, due to the decreased regularization constant allowing a more intrusive correction.

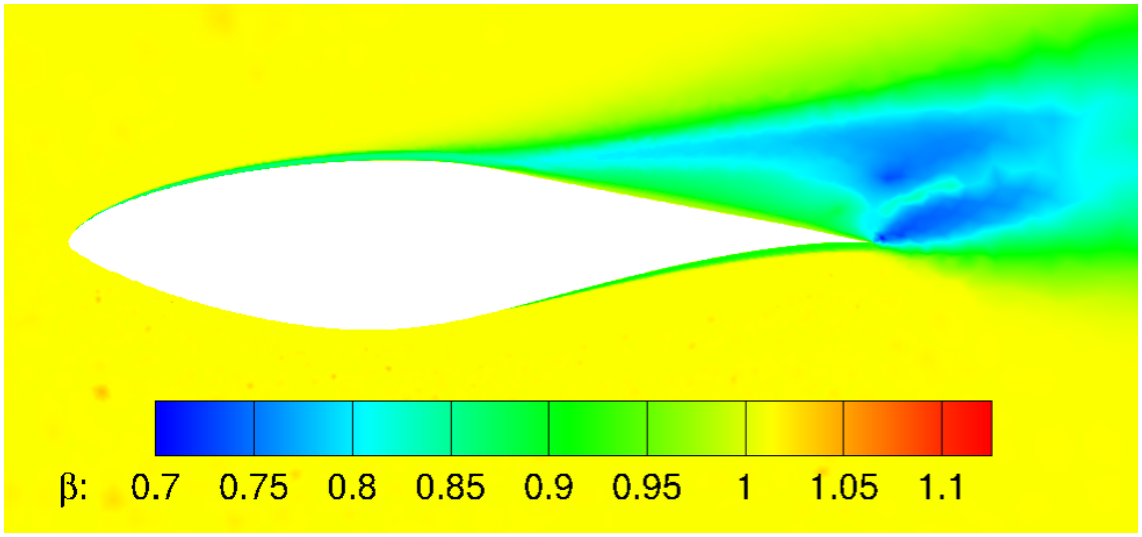

Figure 6.19: Correction field (β) for FIML-Direct with $\lambda = 10^{-5}$.

6.2.6 Comparison

Comparisons between the three FIML methods tested on the S809 airfoil for this single case are presented in this section. First, note that the three methods are using different neural network structures and numerical approaches so compar-

	C_L	Error
Target	1.0546	-
Baseline	1.15826	0.10366
FI-Classic, $\lambda = 10^{-4}$	1.06726	$\pm$ 0.01266
FIML-Classic, $\lambda = 10^{-4}$	1.06914	$\pm$ 0.01454
FIML-Embedded, $\lambda = 10^{-4}$	1.07372	$\pm$ 0.01912
FIML-Direct, $\lambda = 10^{-4}$	1.09483	$\pm$ 0.04023
FIML-Direct, $\lambda = 10^{-5}$	1.05834	$\pm$ 0.003740

Table 6.2: Comparison of C_L following inversion and/or model augmentation for FIML approaches on the S809 airfoil; the "$\pm$" is applied to emphasize that some results, as discussed, are not fully converged so confident conclusions about the relative performance of the three FIML approaches cannot be made.

isons must be made with caution. Additionally, the FIML-Embedded solution was not sufficiently converged due to robustness issues, as discussed. Convergence verification for the FIML-Direct application with $\lambda = 10^{-5}$ also remains an item for future work. Therefore, because of these considerations, confident comparisons of the performance of each method will not be made here, and the relative achievable performance of the methods remains uncertain. Table 6.2.6 compares the lift coefficients achieved for all three FIML approaches.

Note that all the FIML approaches result in augmentations that dramatically

reduce the error in the predicted lift coefficient. The penalty for imperfect training for the FIML-Classic approach is shown in Table 6.2.6. The difference in the FI-Classic and FIML-Classic result is simply due to the fact that the neural network used for the augmentation cannot perfectly replicate the correction field. FIML-Embedded for the same regularization constant performs somewhat worse than the FIML-Classic augmentation for this choice of neural network hyperparameters and regularization constant. It is certain that the performance of the FIML-Embedded algorithm would be improved if a solution to the observed robustness issues for the RANS applications is developed, enabling the inversion to converge further. The FIML-Direct method performs worse than the other methods for the same regularization constant, but as the regularization is being provided by considering the limitations of the neural network in the inversion it is appropriate to decrease the regularization. The lower FIML-Direct error bar shows the best performance of all augmentations ($\lambda = 10^{-5}$ for that case). Also note that, in general, it is possible that a different optimization algorithm or improved implementation could achieve better inversion performance than the results presented here.

The experimental lift coefficient at a single angle of attack was the only higher fidelity data used to perform the inversions: a single number. It is somewhat remarkable, therefore, that the entire pressure distribution on the airfoil is corrected by the augmentations as illustrated in Figure 6.20. Note that the baseline model shows substantial error in the experimentally observed separation location, but all the FIML approaches substantially reduce this error. Also note that the three FIML approaches result in different optimal solutions for the SA production term (β) as

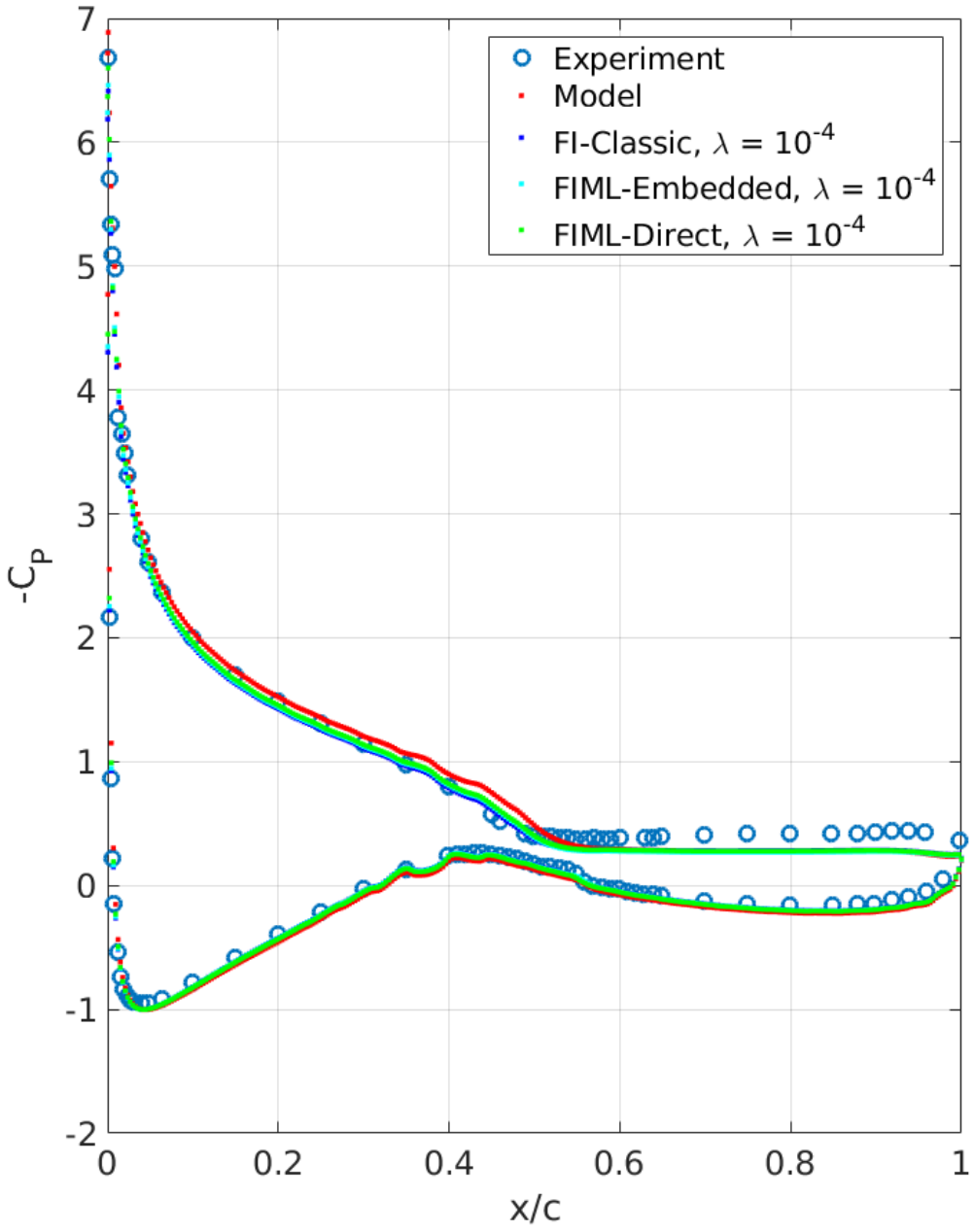

Figure 6.20: Comparison of pressure distributions for baseline SA-BC model and FIML augmentations.

shown in Figure 6.21. This is due to the differing ways of accounting for (or neglecting) the limitations of the learning algorithm in the inverse problem.

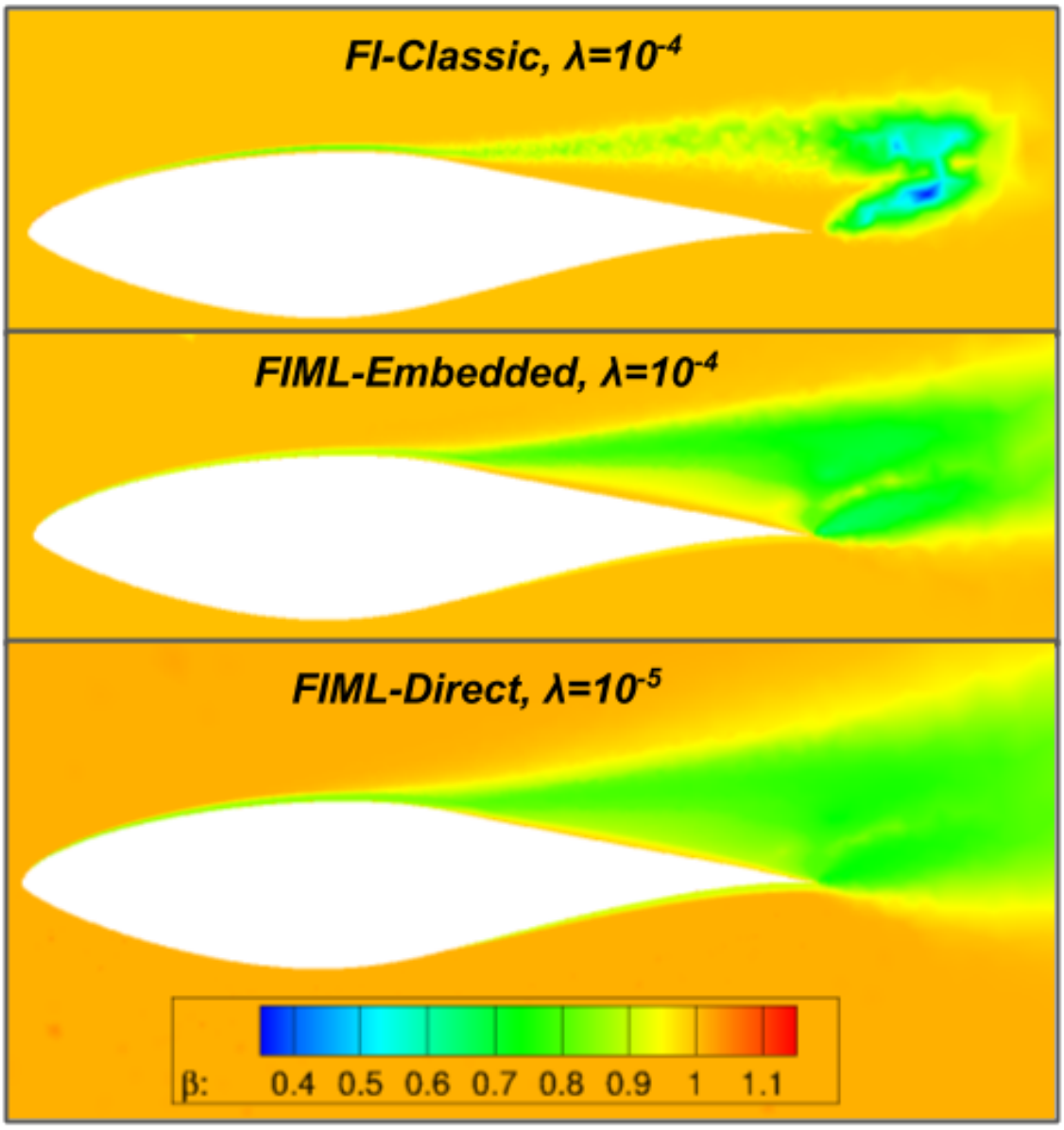

Figure 6.21: Comparison of production term correction (β) for FIML inversions.

6.2.7 FIML-Direct for Simultaneous Inversion of Multiple Cases

As discussed in Chapter 4, because the design variables are consistent across multiple cases for the FIML-Direct approach, the simultaneous inversion of multiple cases can be performed. In other words, because we are seeking a single neural network that augments the model for a variety of cases we can perform the FIML-Direct inversion on multiple cases simultaneously. Figure 3.4 illustrates this process. There is no requirement for the objective function to be the same for each case, meaning whatever data is available can be used for each case. Additionally, there is no requirement to use the same computational grid, and therefore multiple geometries can be considered simultaneously.

To demonstrate the simultaneous inversion of multiple cases the FIML-Direct inversion is performed on the S809 airfoil at three angles of attack, $\alpha_{train} = \{1.02°, 8.2°, 14.2°\}$. The individual objective functions were the same form as considered in the single angle of attack case; however, the regularization constants were different between the angles of attack considered. This is due to the difference in confidence between the conditions. At $\alpha = 1.02°$ the confidence in the prediction is high, and therefore the regularization constant was set at $\lambda = 10^{-4}$. For the high angles of attack the model prediction is much less confident due to the high error with respect to the experimental data. Therefore, the regularization constant for these cases was set at $\lambda = 10^{-5}$. The convergence for this case is shown in Figure 6.22. Note that each evaluation corresponds to the solution of 3 direct ($J_d(w) = \sum_i^3 j_i(w)$) and 3 adjoint simulations for roughly the equivalent cost of 6 times the cost of a single

baseline solution.

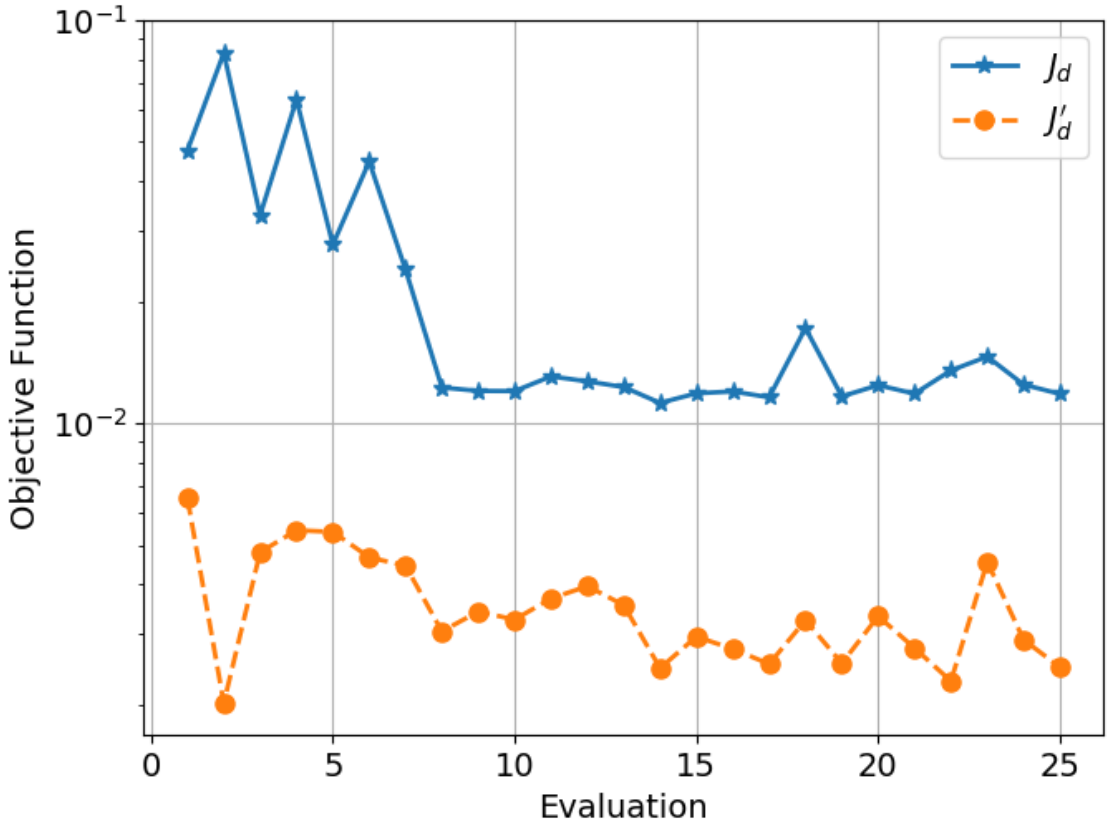

Figure 6.22: Convergence history of S809 airfoil FIML-Direct simultaneous inversion at 3 Angles of Attack.

The minimum was found on Evaluation 14 and the resulting correction for each angle of attack is shown in Figure 6.23.

To evaluate the generalization ability of the resulting augmentation the network was tested on four additional (unseen) angles of attack that were not included in the training set. These are referred to as the holdout cases, so $\alpha_{holdout} = \{5.13°, 11.21°, 12.22°, 15.24°\}$. The baseline results were then compared to the augmentation trained on the three α_{train} conditions. The results are summarized in Figure 6.24. Note that for all the angles of attack in the training and holdout cases, the

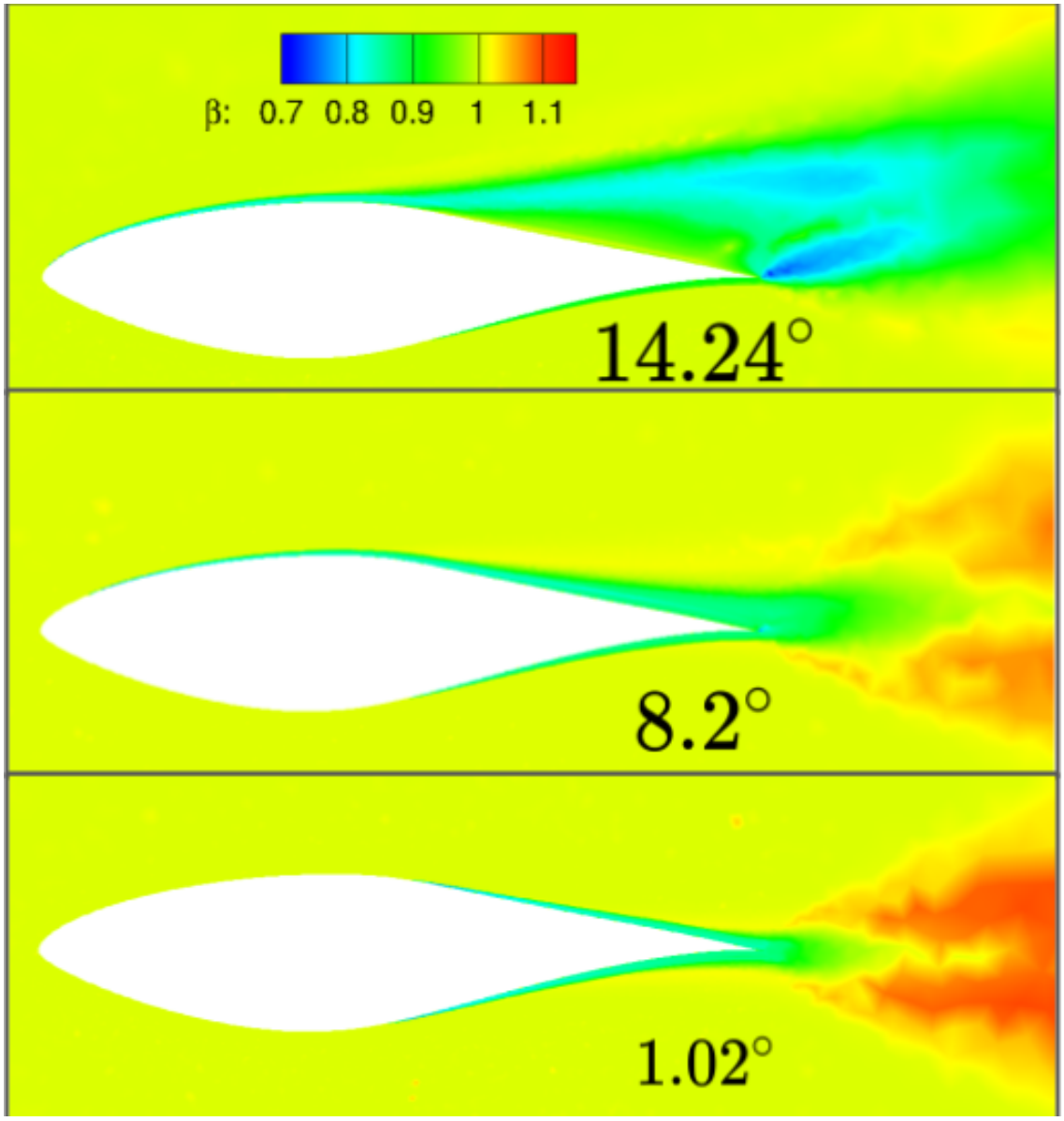

Figure 6.23: Correction for all three angles of attack at best evaluation.

predicted lift coefficient improved. Additionally, mild extrapolations in angle of attack from the training set were possible with the augmented model.

To further improve the augmentation the inversion was continued with all seven angles of attack. This highlights the flexibility of the FIML-Direct approach, as the modeler can choose to continue training with additional data as necessary,

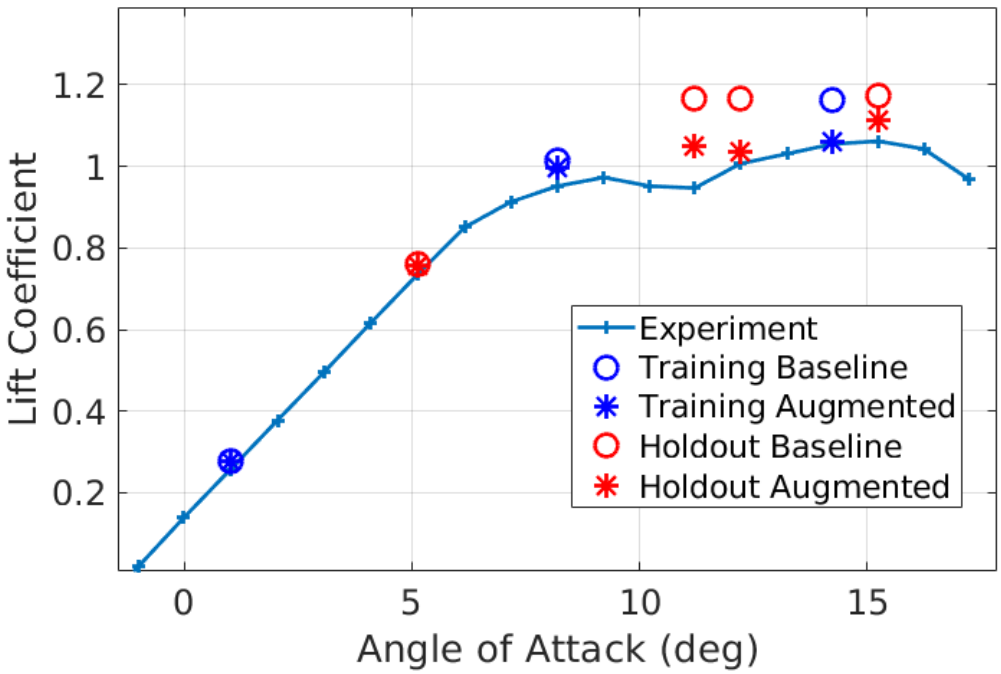

Figure 6.24: Summary of augmentation performance for training and holdout cases for augmentation trained on three angles of attack.

or an existing augmentation could incorporate new data that becomes available. Following the inclusion of the additional angles of attack the result of the inversion with all seven cases is shown in Figure 6.25. The drag prediction is also improved, as shown in Figure 6.26.

To test the augmentation created from S809 lift coefficient data, augmented predictions were made from the S814 airfoil. The Mach number contour for the baseline SA-BC prediction on the S814 airfoil at 15.25° angle of attack is shown in Figure 6.27.

The S809 augmentation also improves prediction on the S814 airfoil as shown in Figure 6.28. The corrections output by the neural network for three of the S814

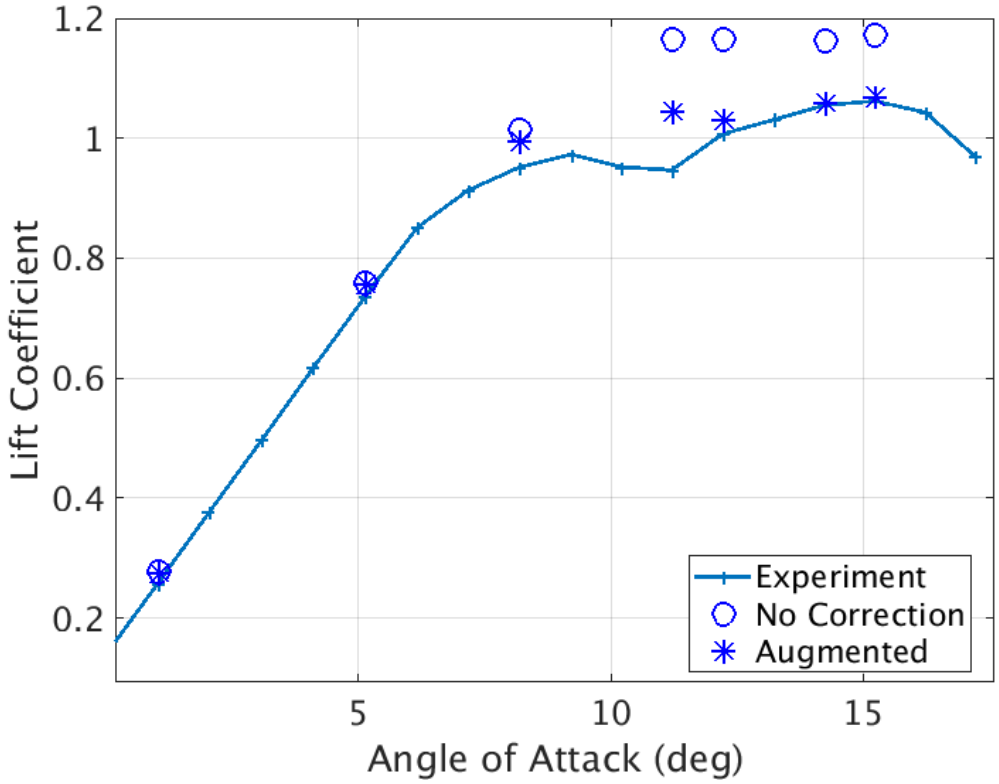

Figure 6.25: S809 augmentation lift performance for training set of all seven angles of attack.

angles of attack are shown in Figure 6.29. Additionally, the pressure distribution for the S814 at 16.2° angle of attack is compared to the experimentally obtained pressure distribution in Figure 6.30. The pressure distribution has been substantially improved by the augmentation.

The S814 results demonstrate an important characteristic of machine learning augmented models. Despite the S814 having a different shape than the S809, an augmentation trained using the FIML-Direct process on S809 information substantially improves predictions on the S814. Although the airfoils are different, the feature space of the S814 has been covered by the S809 cases; and, therefore the

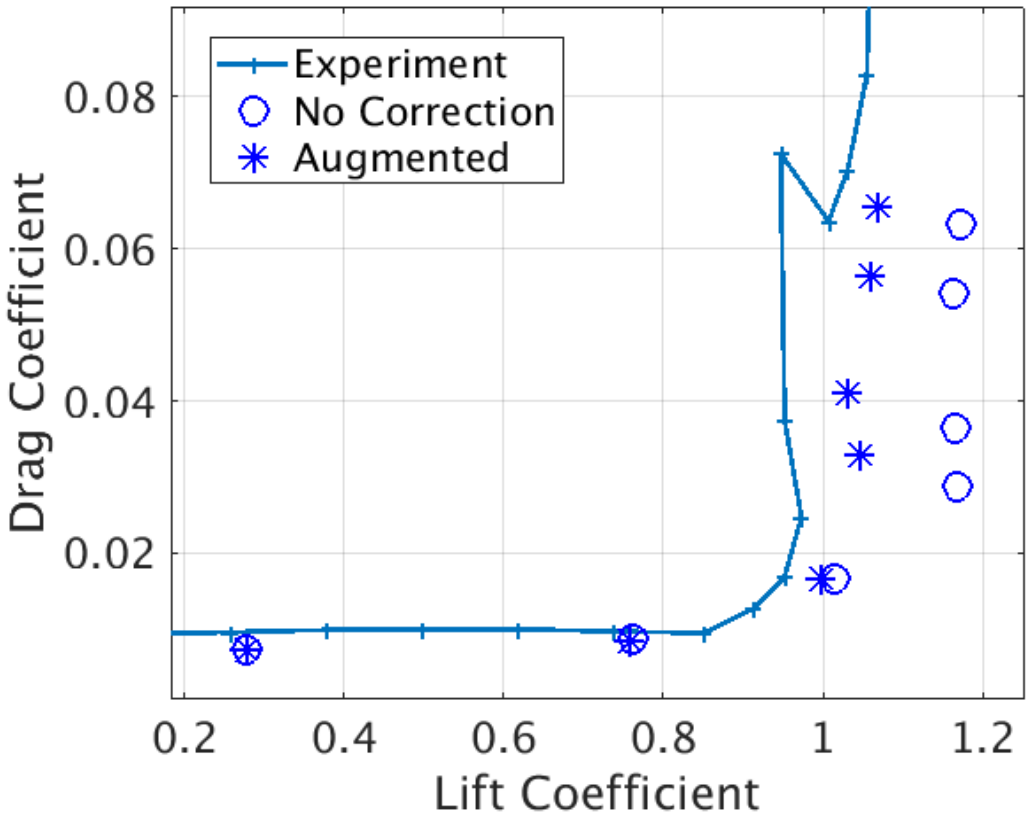

Figure 6.26: S809 augmentation drag performance for training set of all seven angles of attack.

model does not need information from the S814 cases to improve predictions. In other words, while two cases may be different, their features may be closer than expected and the information learned from one case may be surprisingly effective on a geometrically dissimilar case.

6.3 RAE2822 Airfoil

The RAE2822 transonic is another canonical RANS test case. This airfoil exhibits a shock boundary layer interaction on the suction surface that produces

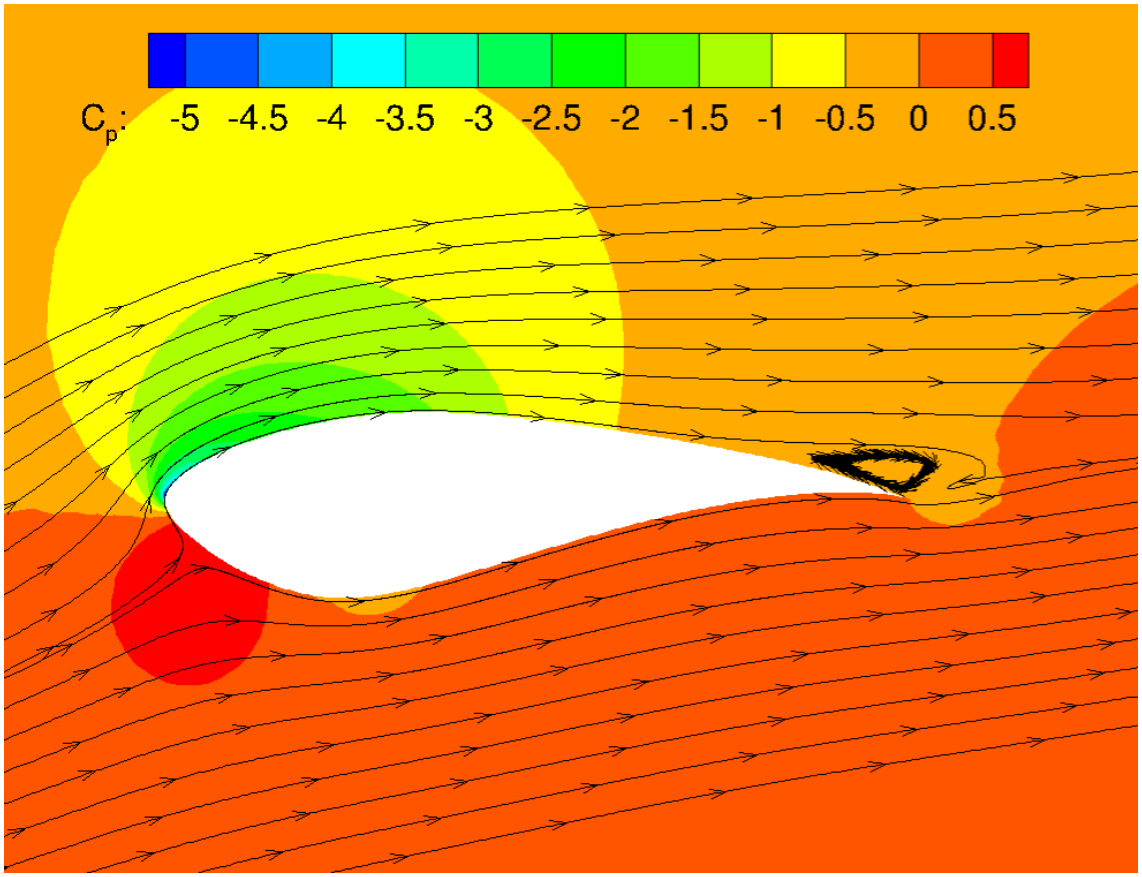

Figure 6.27: S814 baseline SA-BC pressure and streamline predictions for $\alpha = 16.2°$.

a strong adverse pressure gradient. This violates the assumptions in the RANS models, and the models have difficulty predicting the precise shock location as well as the response of the boundary layer to the large adverse pressure gradient of the shock. This error results in incorrect force predictions. RAE2822 wind tunnel predictions are provided by Cook et al. [24], and the higher fidelity data for the FIML applications was the experimentally derived pressure coefficient ($k_d = C_{P_{exp}}$). The freestream Mach number was $M_\infty = 0.729$ and the angle of attack was 2.31°. This corresponds to the conditions on the NPARC Wind validation website [112]. The experimental pressure distribution is interpolated onto the boundary points of

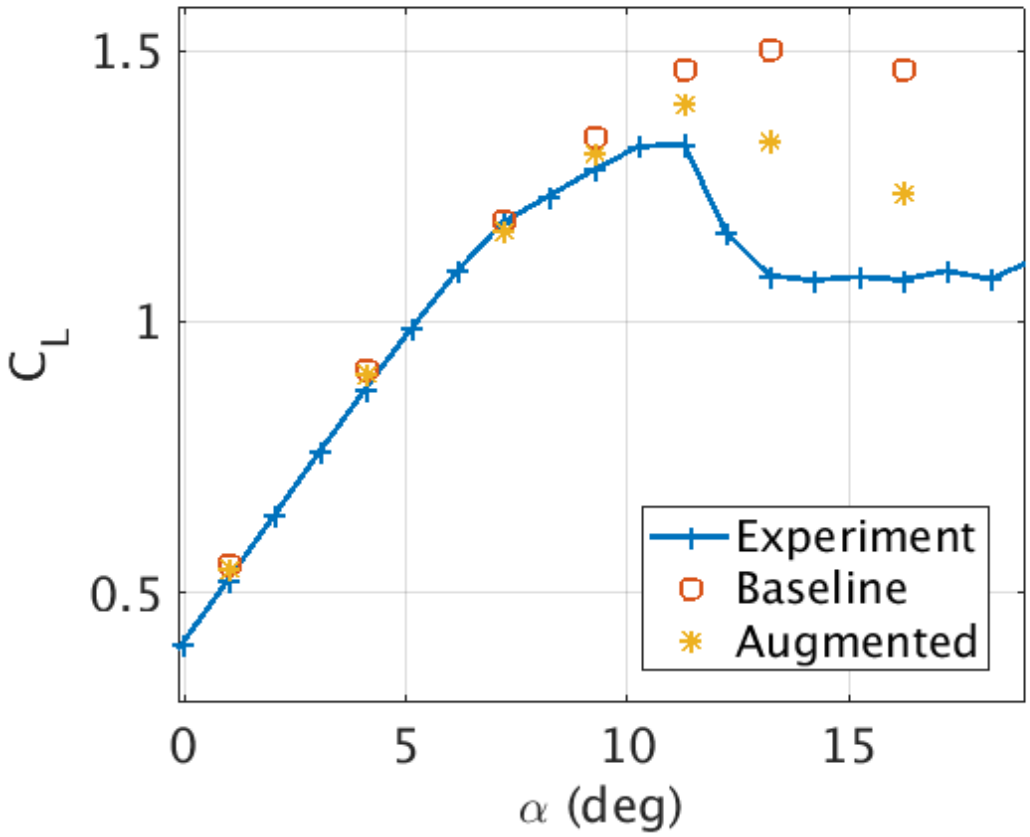

Figure 6.28: S814 C_L resulting from augmentation trained on S809 angles of attack.

the airfoil mesh.

6.3.1 FI-Classic

The objective function for the FI-Classic application is given by Equation 6.6, where k is the total number of boundary points defining the airfoil in the computational domain, n is the total number of points in the domain, and $\widehat{C}_{p_{exp}}$ represents the interpolated pressure coefficient from the experiment.

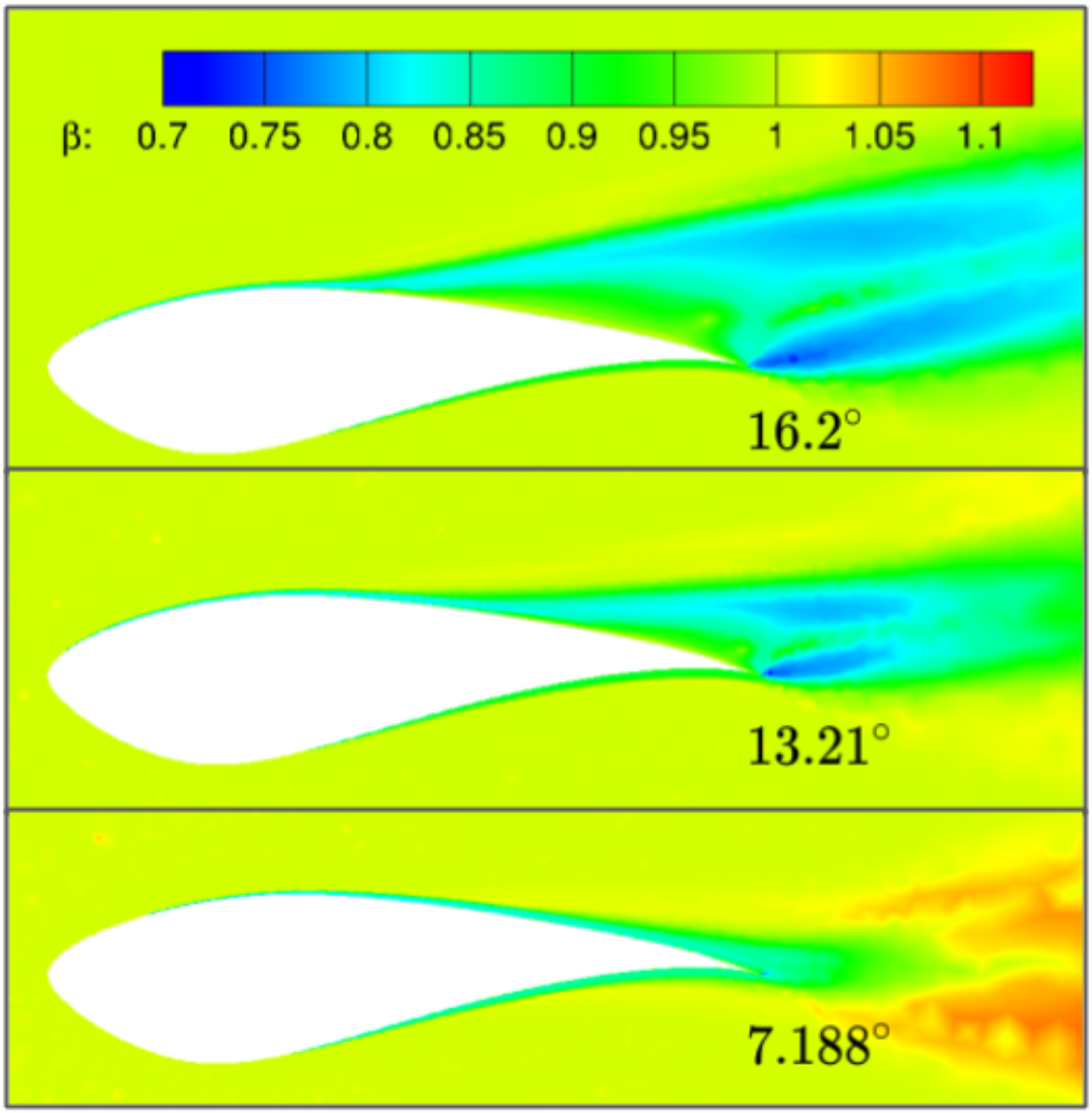

Figure 6.29: S814 β fields for three angles of attack.

$$J_c(\beta) = \sum_{i=1}^{k} (\widehat{C}_{p_{exp},i} - C_{P,i}(\beta))^2 + \frac{1}{2}\lambda \sum_{j=1}^{n} (\beta_j - 1.0)^2 \tag{6.6}$$

Again, the BFGS optimizer was used. The regularization constant was chosen to be $\lambda = 10^{-7}$. The initial solution is quite good, with the predicted shock location being relatively close to the experiment. The pressure immediately behind the

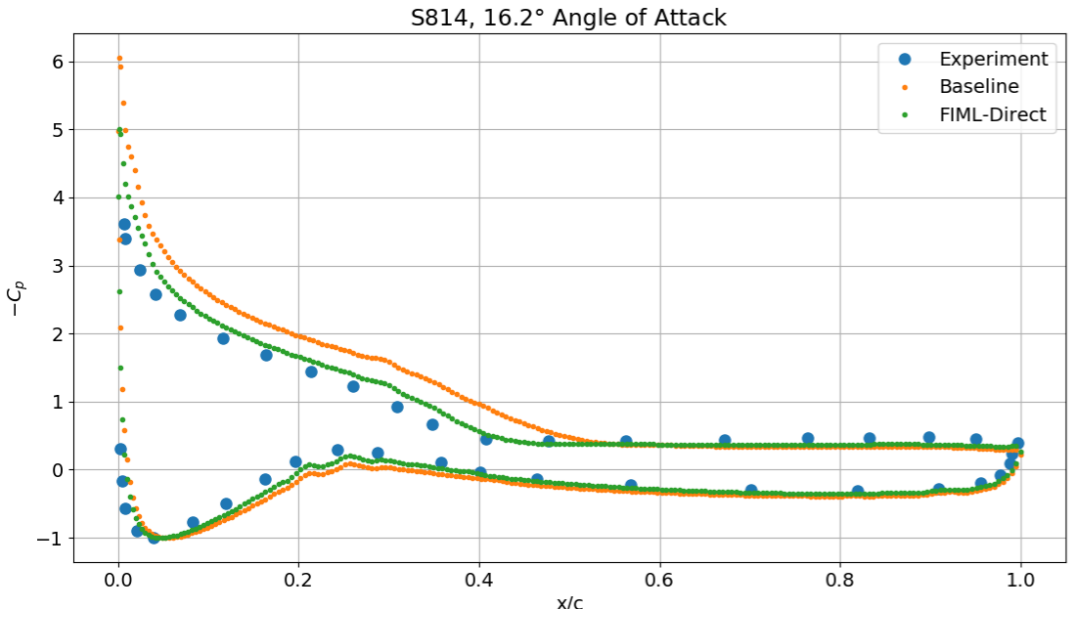

Figure 6.30: Pressure distribution plot for the S814 at 16.2° angle of attack showing improvement in prediction for the model augmentation.

shock is too low, indicating that the model is not predicting the correct boundary layer reaction to the strong adverse pressure gradient behind in the shock. The convergence of the objective function throughout the inversion is shown in Figure 6.31.

The pressure distribution of the experiment, baseline SA model, and FI-Classic inverse solutions are shown in Figure 6.32 and 6.33. As shown, the inversion has corrected the small error in predicted shock location (inverse solution shock slightly forward), and the inverse solution shock location agrees better with the experiment. Also note that the pressure rise through the smeared shock foot is now over a larger distance that more closely matches the experiment. As noted, it is not expected that the SA model will be very accurate in this region due to the strong adverse pressure gradient, and therefore it is expected that the inverse solution will show the

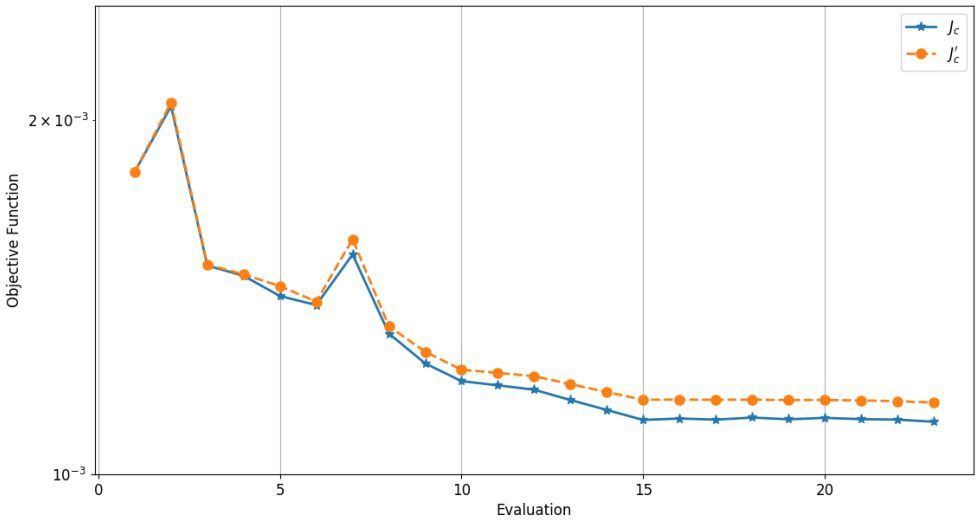

Figure 6.31: Convergence of RAE2822 FI-Classic objective function.

most improvement in this area. With the improved agreement in the shock location the pressure following the shock is higher than the baseline model, and in better agreement to the experiment.

The correction is also largest near the shock foot. Figure 6.34 shows the correction over the whole airfoil, and Figure 6.35 illustrates the correction and pressure distribution near the SWTBLI.

6.3.2 FIML-Direct

The FIML-Direct approach was also applied to the RAE2822 airfoil. The same experimental data was used giving the objective function in Equation (6.7). $\lambda = 10^{-6}$ was used for this case, as well as a neural network with two hidden layers of 20 neurons each. The same features were used as in the S809 airfoil case. The

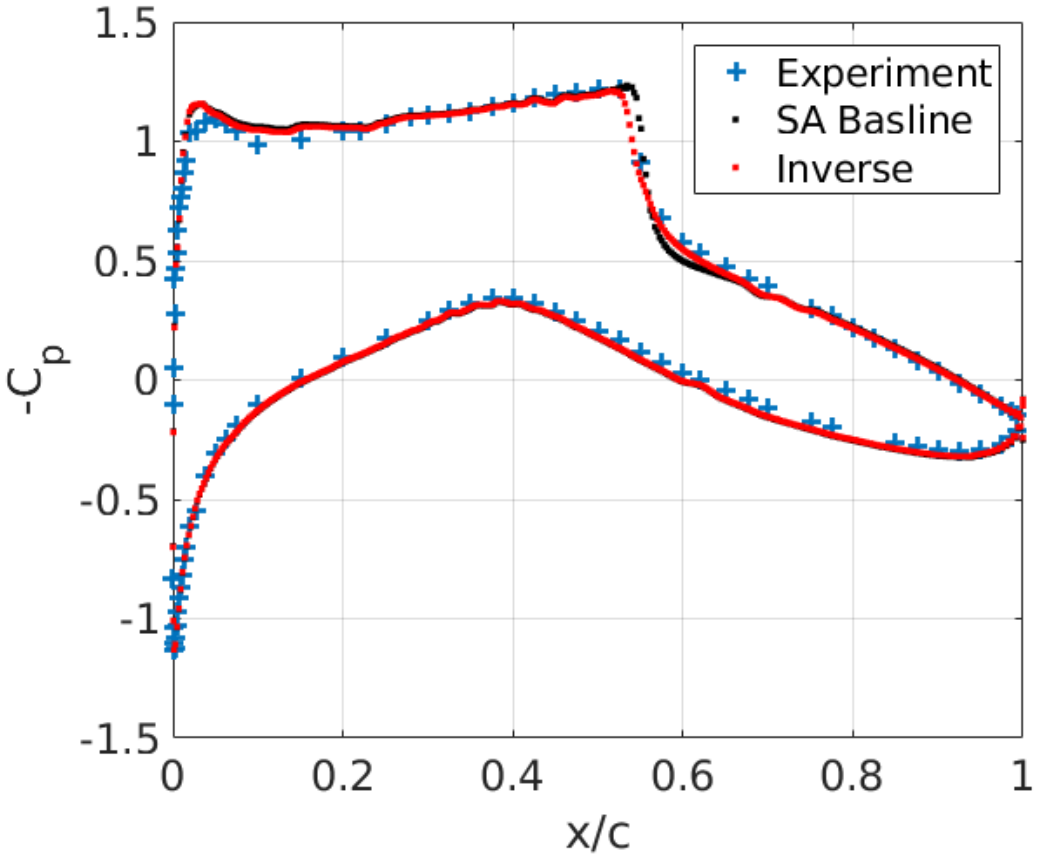

Figure 6.32: RAE2822 FI-Classic pressure distribution results.

convergence is shown in Figure 6.36.

$$J_d(\beta) = \sum_{i=1}^{k} (\widehat{C}_{p_{exp},i} - C_{P,i}(w))^2 + \frac{1}{2}\lambda \sum_{j=1}^{n} (\beta_j(w) - 1.0)^2 \tag{6.7}$$

Note that in Figure 6.36 there is little difference between J_d and J'_d indicating that the regularization term is not contributing much to the overall objective function despite setting the regularization term much higher than in the FI-Classic approach. The reason for this becomes apparent when looking at the optimal correction field output from the neural network as shown in Figures 6.37 and 6.38, showing the correction over the entire airfoil and at the interaction region respectively.

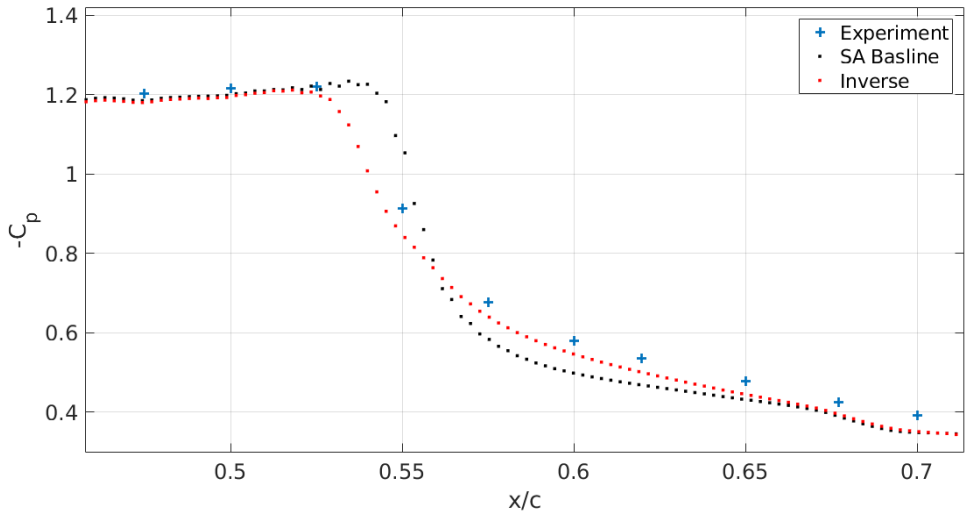

Figure 6.33: RAE2822 FI-Classic pressure distribution results near shock location.

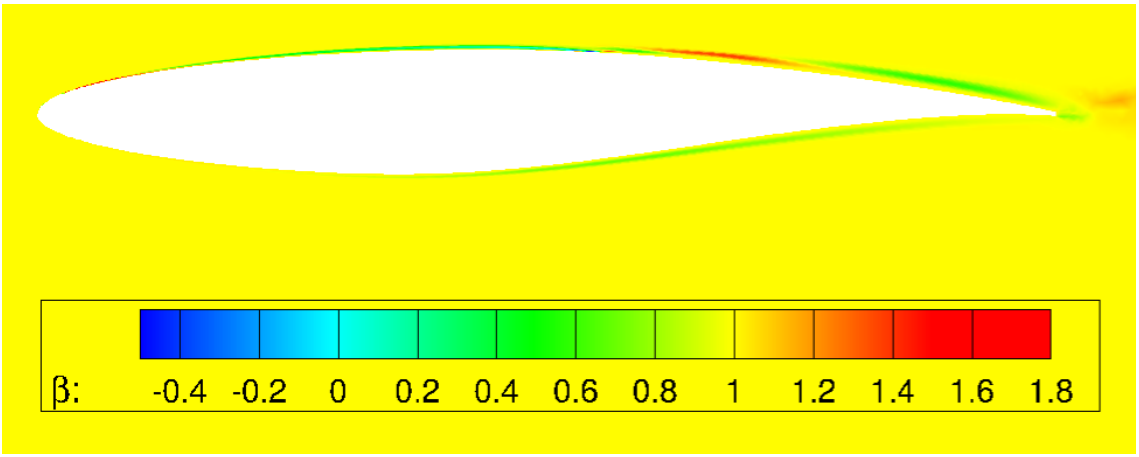

Figure 6.34: FI-Classic correction for RAE2822 application.

Similar to the S809 airfoil results, the FIML-Direct correction is much more regularized in that the magnitude of the correction is smaller and is spread over a larger area. This regularization is not being provided by the cost function regularization constant (λ), but is due to the limitations of the network that are now considered in the inversion.

The pressure distribution resulting from this inversion is shown in Figure 6.39.

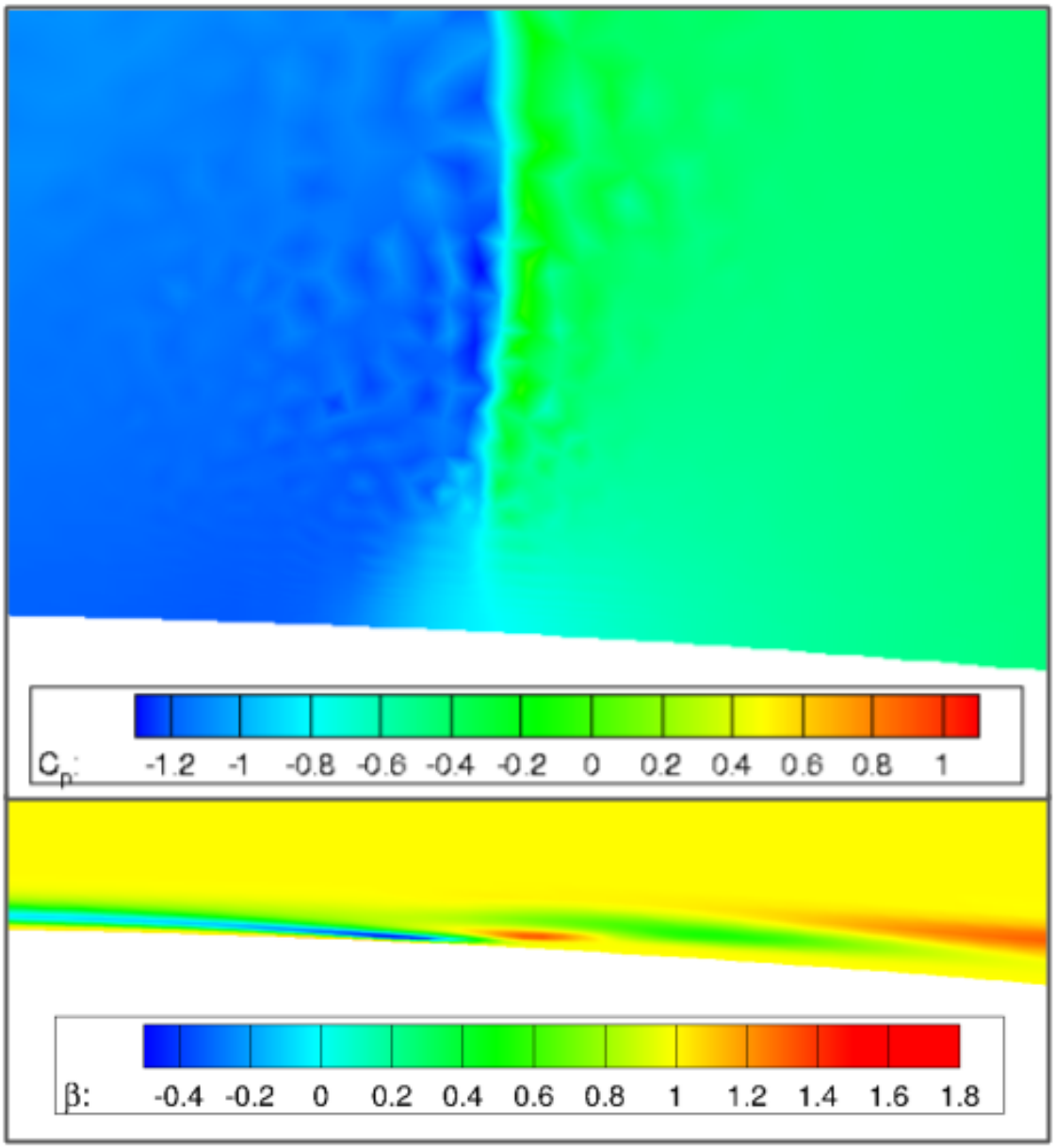

Figure 6.35: Pressure distribution and FI-Classic correction near the smeared shock foot.

Even with the increased regularization from the neural network the FIML-Direct inverse solution improved the shock location. This is better illustrated in Figure 6.40. Note, however, that the correction to the smeared shock-foot that was possible in the FI-Classic application is not shown in the FIML-Direct results. This is likely

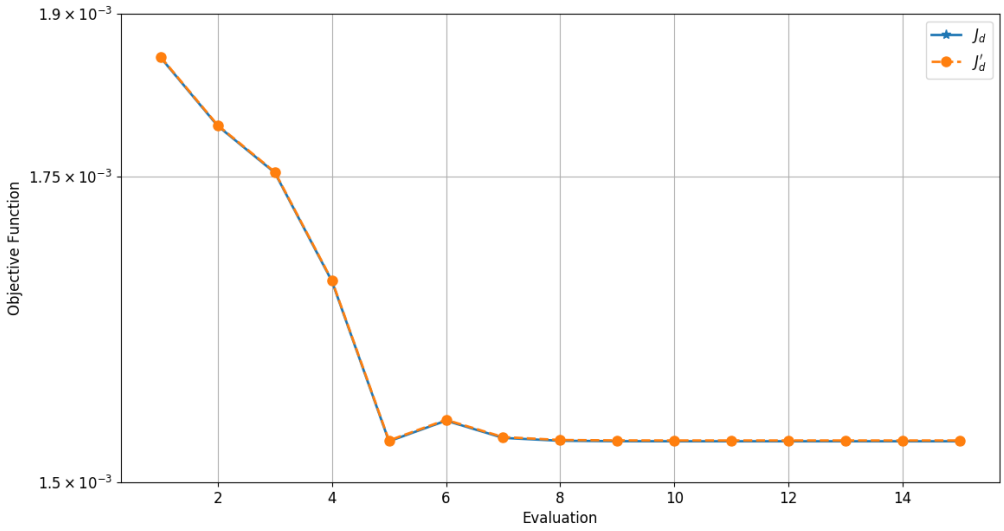

Figure 6.36: FIML-Direct convergence for RAE2822 airfoil test case with $\lambda = 10^{-6}$

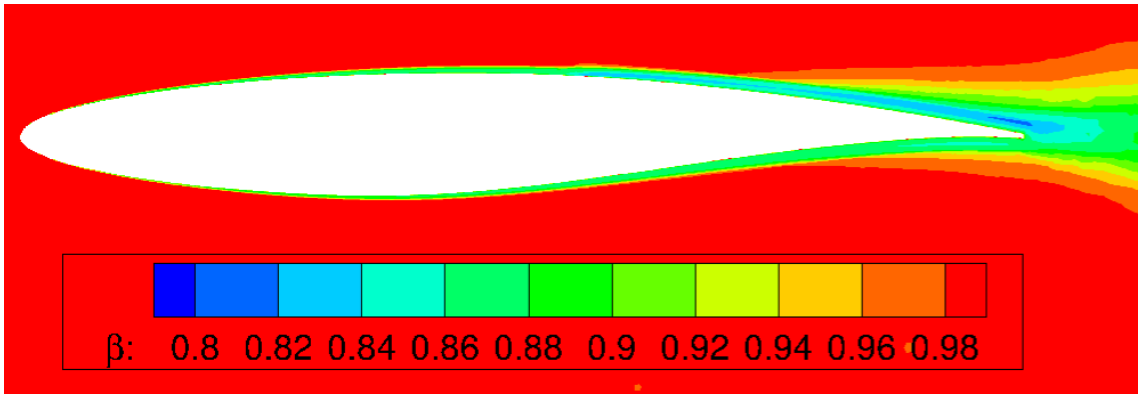

Figure 6.37: FIML-Direct correction ($\beta(w)$) for RAE2822 airfoil test case.

due to the correction near the shock being high in magnitude and in very small regions near the interaction region. This likely prevented learning of the required correction to adequately adjust the boundary layer reaction to the shock. It is possible that improved feature selection or a different neural network structure could improve the FIML-Direct performance for this case. Also note that to apply the information given by the FI-Classic inverse solution the information would still need

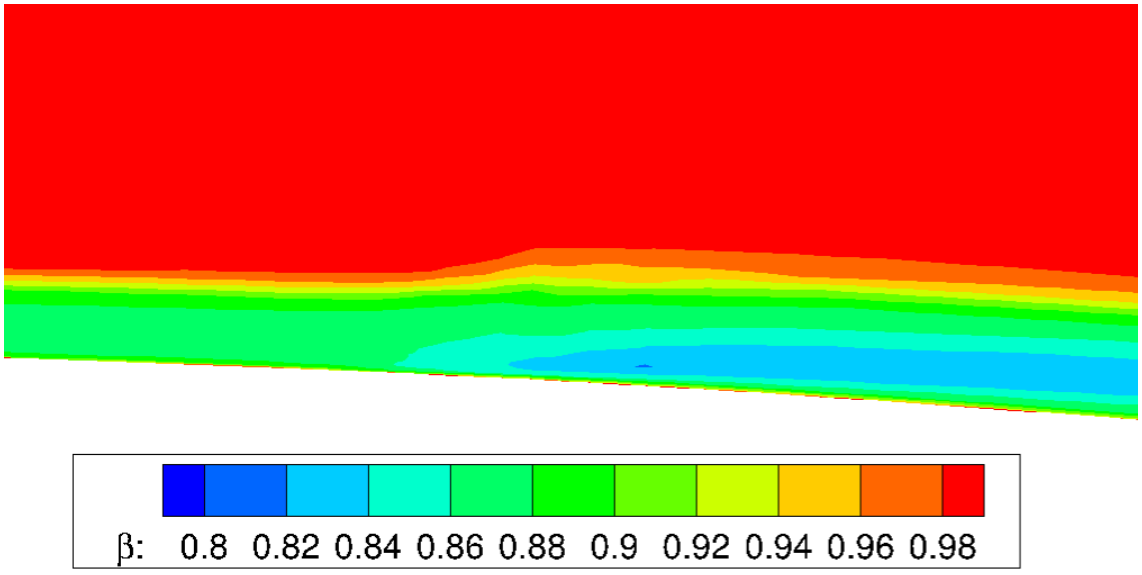

Figure 6.38: FIML-Direct correction at the shock interaction location for RAE2822 airfoil test case.

to be learned. For the FIML-Direct result the neural network has already been trained.

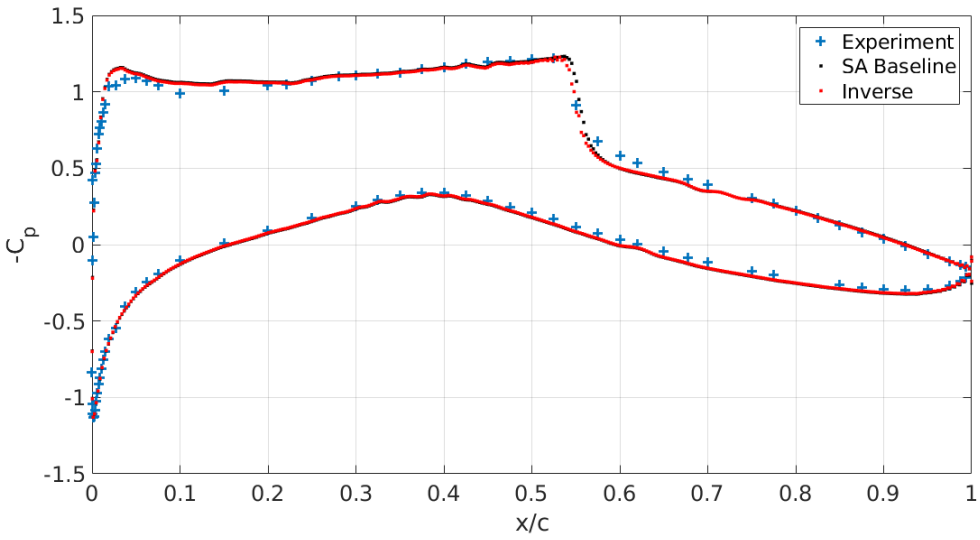

Figure 6.39: FIML-Direct pressure distribution results.

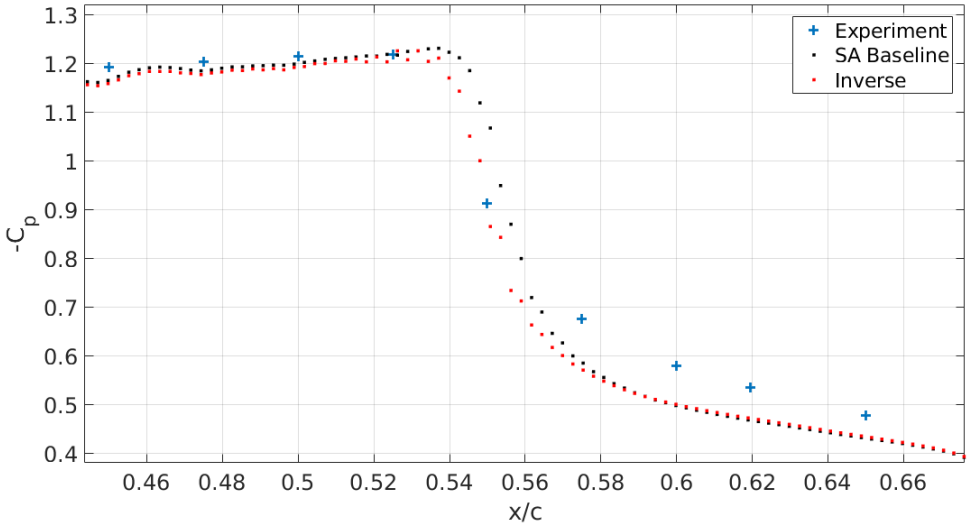

Figure 6.40: FIML-Direct pressure distribution results at the shock location.

6.4 NASA Langley Hump

6.4.1 Overview

The NASA Langley wall mounted hump was also examined. Similar to the S809 cases this is an incompressible, 2D, separated flow application. The case is based on a wind tunnel model hump that was placed directly on the tunnel wall. A fully developed turbulent boundary layer then passes over the hump and a large separation region develops on the leeward side. It has been shown that RANS models are unable to adequately predict the size of the separated region [111]. The computational domain was developed to mimic the conditions in the tunnel experiment. The top of the domain is an inviscid wall with a contoured shape to account for blockage effects in the wind tunnel experiment. The bottom of the domain is a viscous wall with sufficient length to develop the boundary layer consistent with

the experiment. Details concerning this computational setup are provided on the turbulence modeling resource website [25]. The pressure contours and streamlines predicted by the baseline SA model are shown in Figure 6.41.

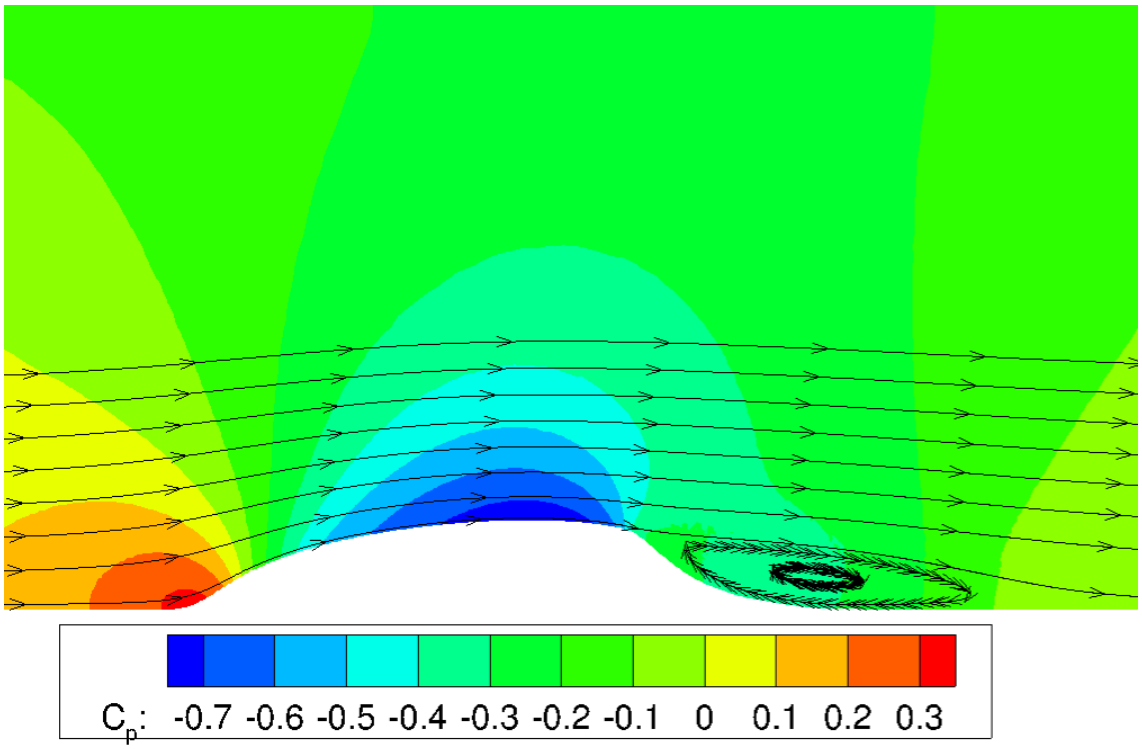

Figure 6.41: Baseline SA model predictions for pressure coefficient over the NASA Langley wall mounted hump.

As shown in Figure 6.41 there is a strong adverse pressure gradient just ahead of the separated region on the top of the hump. Similar to the S809 results we expect the eddy viscosity to be too high in this region. Additionally, in the separated region the assumption of the wall distance is invalid and therefore it is expected that the eddy viscosity is poorly modeled there.

6.4.2 FI-Classic

The FI-Classic method field inversion was performed on the NASA Langley 2D hump. The experimentally derived pressure distribution was used as the higher fidelity data $k_d = C_P$. Where experimental pressure was available, the pressure distribution was interpolated onto each point in the computational domain on the hump. This results in the following objective function for this application 6.8, where k is the total number of boundary points defining the hump in the computational domain, n is the total number of points in the domain, and $\widehat{C}_{p_{exp}}$ represents the interpolated pressure coefficient from the experiment.

$$J_c(\beta) = \sum_{i=1}^{k} (\hat{C}_{p_{exp},i} - C_{P,i}(\beta))^2 + \frac{1}{2}\lambda \sum_{j=1}^{n} (\beta_j - 1.0)^2 \tag{6.8}$$

First, a case was run with $\lambda = 0$. In general, this is not expected to produce a useful correction field because it is expected that the resulting correction will be excessively large. $\lambda = 0$ corresponds to *no* confidence in the baseline model. Clearly it is not believed that the baseline SA model is completely wrong in this case, however the resulting inversion illustrates the reaction of the inversion to different regularization constants. The convergence history of J_c with $\lambda = 0$ is shown in Figure 6.42.

Note the convergence is fantastic. The convergence likely would have continued however the inversion was terminated because the correction field was extreme, as shown in Figure 6.43. Despite this, the model shows an incredible match to the data over the baseline solution as shown in Figure 6.44. This illustrates the effect

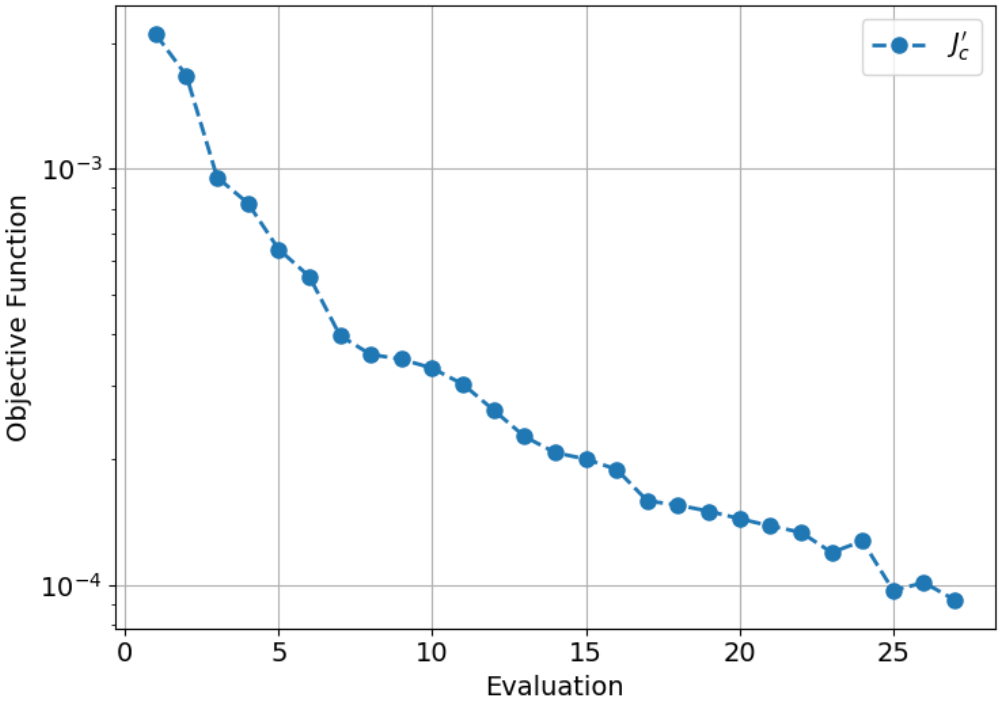

Figure 6.42: FI-Classic convergence for NASA Langley Hump test case with $\lambda = 0$

of the regularization constant: it is a compromise between matching the available data and the modeler's confidence in the model.

For a much more useful inversion, $\lambda = 10^{-6}$ was chosen. The convergence for this case is shown in Figure 6.45. The convergence is still quite good, and the resulting correction field is much more reasonable (Figures 6.46 and 6.47), as it is much lower in magnitude, and more closely localized to where there are violated assumptions in the SA model.

Note that in Figure 6.47 the correction is reducing eddy viscosity production in the strong adverse pressure gradient just before the boundary layer separates. This is similar to the behavior observed in the S809 cases. Then, also similar to the

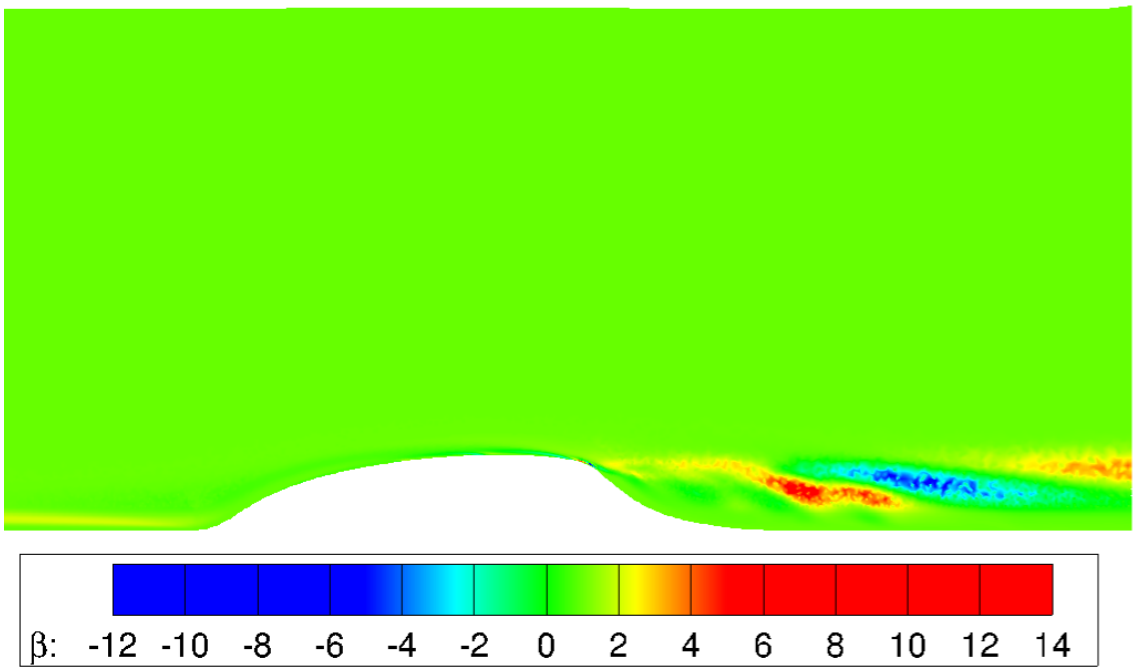

Figure 6.43: FI-Classic correction field β for case with $\lambda = 0$

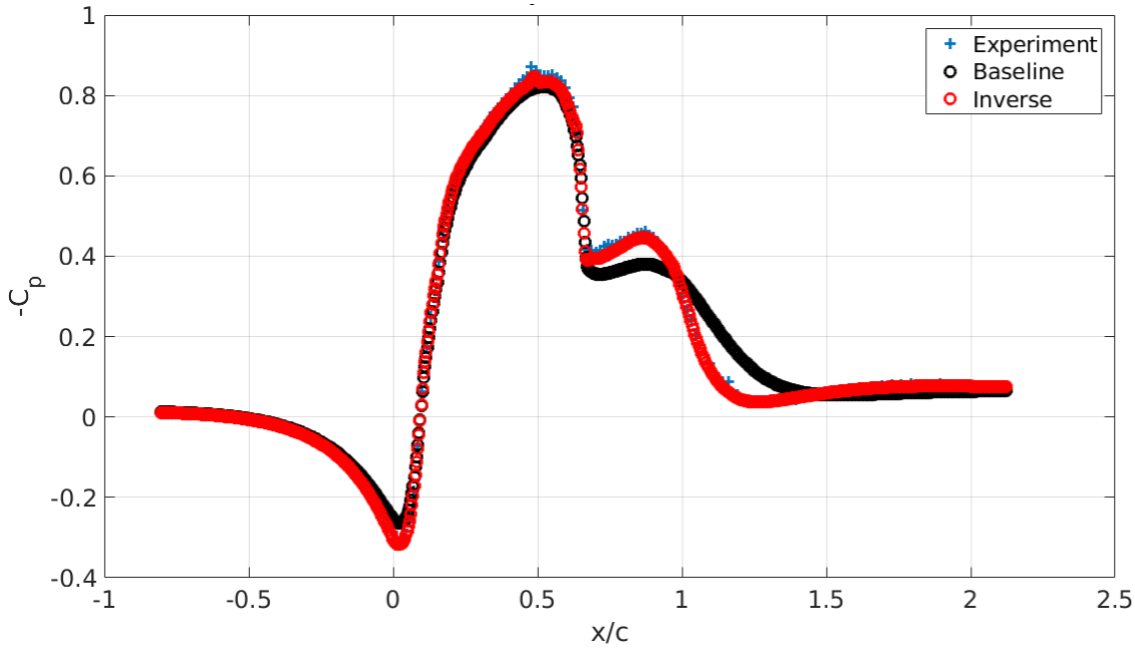

Figure 6.44: FI-Classic C_p comparison to available data k_d for test case with $\lambda = 0$

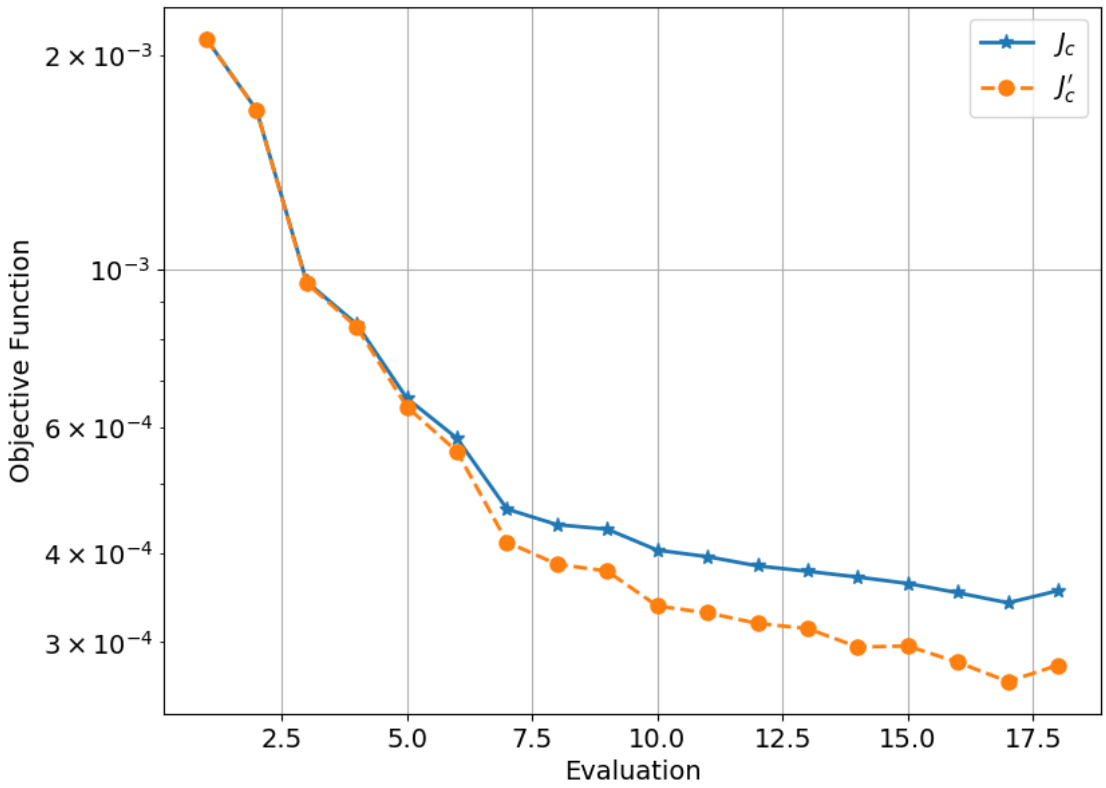

Figure 6.45: FI-Classic convergence for test case with $\lambda = 10^{-6}$

S809 cases the turbulent production is reduced in the separated region. Unlike the S809 cases there is also a large increase in production in the shear layer above the recirculating region.

Note that only the field inversion process has been performed. If an augmentation were to be created, offline training would be performed on the correction and flow features found in the inversion.

6.4.3 FIML-Direct

The FIML-Direct method was also performed on the NASA Langley hump. The computational domain was identical to that used for the FI-Classic application.

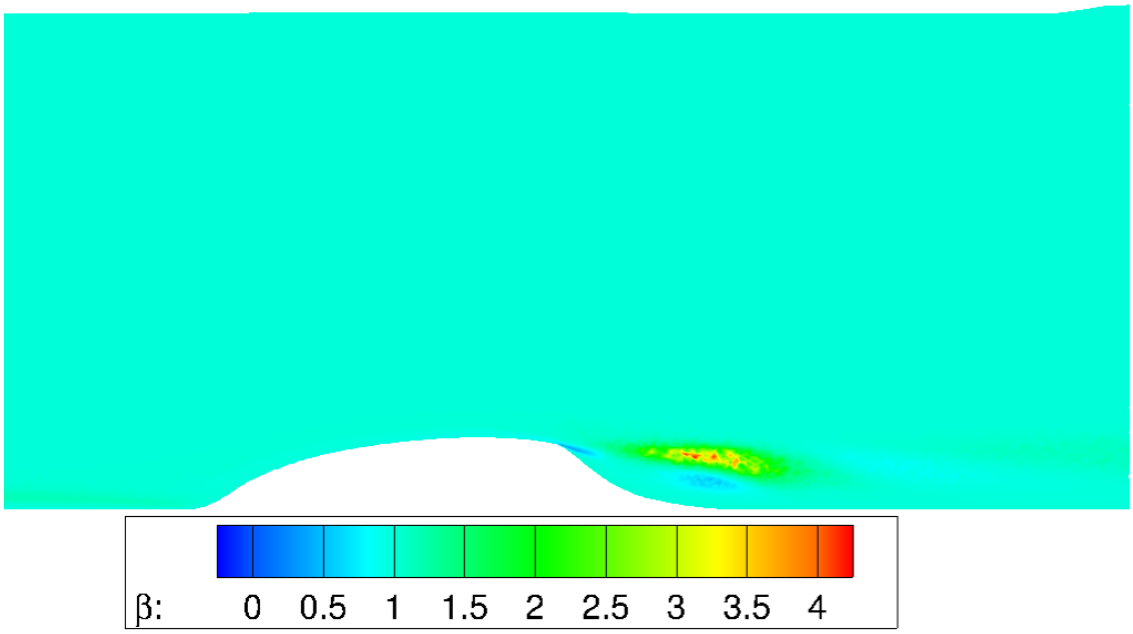

Figure 6.46: FI-Classic correction field β for $\lambda = 10^{-6}$

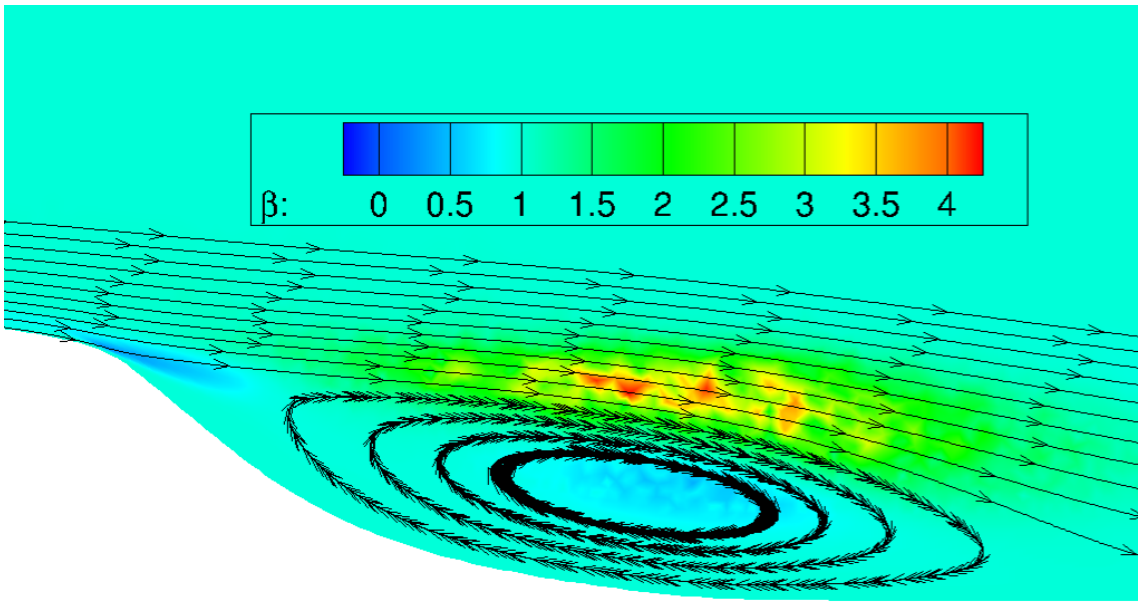

Figure 6.47: FI-Classic correction field β with streamlines for test case with $\lambda = 10^{-6}$

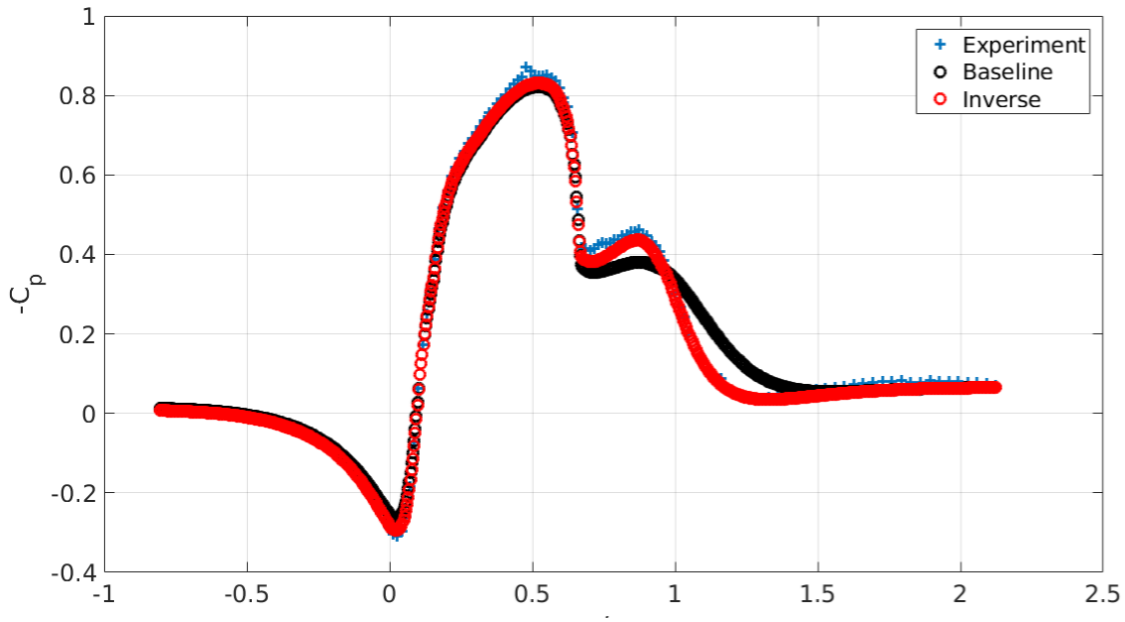

Figure 6.48: FI-Classic C_p comparison to available data k_d for test case with $\lambda = 10^{-6}$

This results in the following objective function for this case:

$$J_d(w) = \sum_{i=1}^{k} (\hat{C}_{p_{exp},i} - C_{P,i}(w))^2 + \frac{1}{2}\lambda \sum_{j=1}^{n} (\beta_j(w) - 1.0)^2 \tag{6.9}$$

Again, because of the increased regularization caused by the FIML-Direct method accounting for the limitations of the neural network in the inversion, the regularization constant was decreased to $\lambda = 10^{-7}$ for the FIML-Direct application. A single hidden layer of 20 hyperbolic tangent neurons was used. The convergence history of $\min_w J_d(w)$ is shown in Figure 6.49.

The minimum occurred on evaluation 19 with the optimizer subsequently struggling to find an appropriate line search step size near the minimum. Note that the regularization constant appears to not be active $J'_d \approx J_d$. Curiously, in-

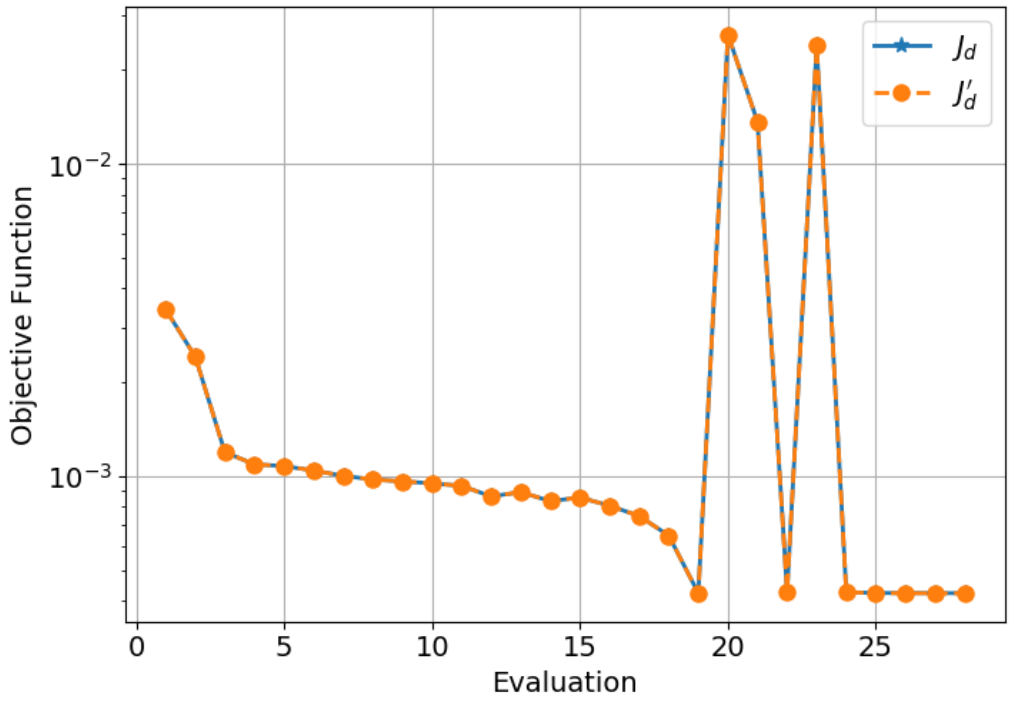

Figure 6.49: FIML-Direct convergence for test case with $\lambda = 10^{-7}$

creasing the regularization constant severely penalized the achievable match to the pressure data and therefore the regularization constant was left at 10^{-7}. Looking at the correction at the minimum (Figure 6.50) the correction has a much larger spatial extent than the equivalent FI-Classic case. It is possible that the features used could not sufficiently differentiate between the boundary layer and the rest of the domain, resulting in corrections far away from the separated region where it is not likely having much effect on the pressure distribution. Nevertheless, the achieved pressure distribution closely matches the experimental data indicating that the neural network augmentation has been successful in reducing the prediction error (Figure 6.51). Note that unlike the FI-Classic results, for the FIML-Direct inversion the

model has now been augmented, and that augmentation could be applied to other cases.

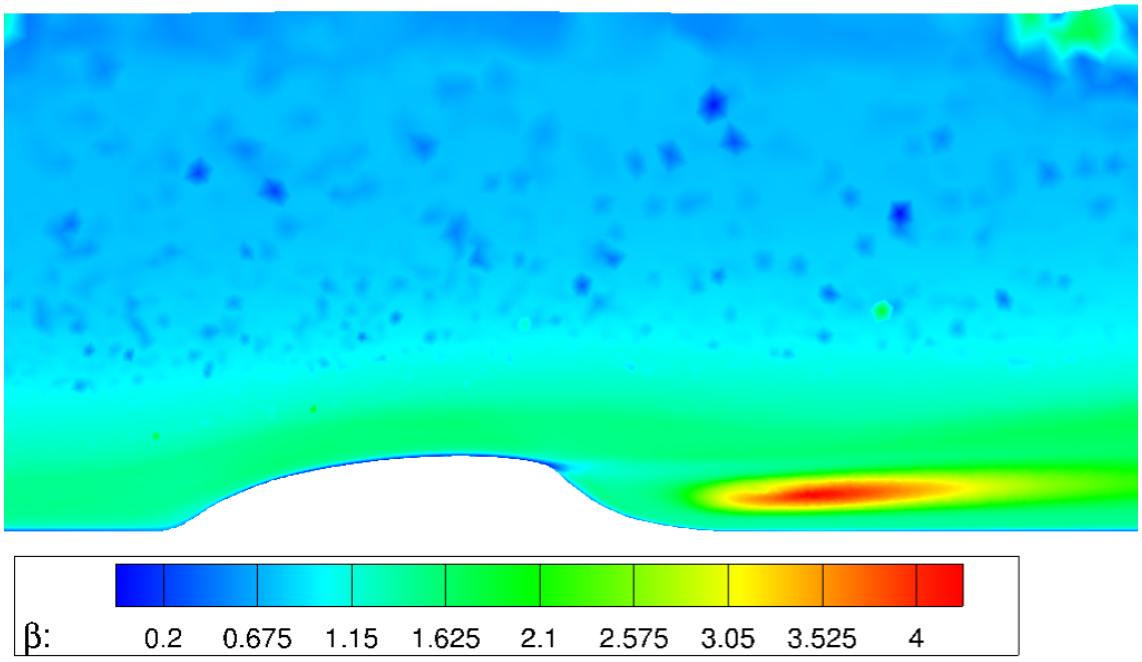

Figure 6.50: FIML-Direct correction field β with streamlines for test case with $\lambda = 10^{-7}$

Also note that the correction for the FIML-Direct application (Figure 6.50) is substantially different from the FIML-Classic application (Figure 6.47). Despite the different correction fields, the achieved pressure distributions both closely match the experiment. This suggests, as expected, that the optimal correction will be different when considering what correction can be learned by the chosen algorithm. In other words, the FIML-Direct correction is optimal for the chosen neural network features, scaling, and structure.

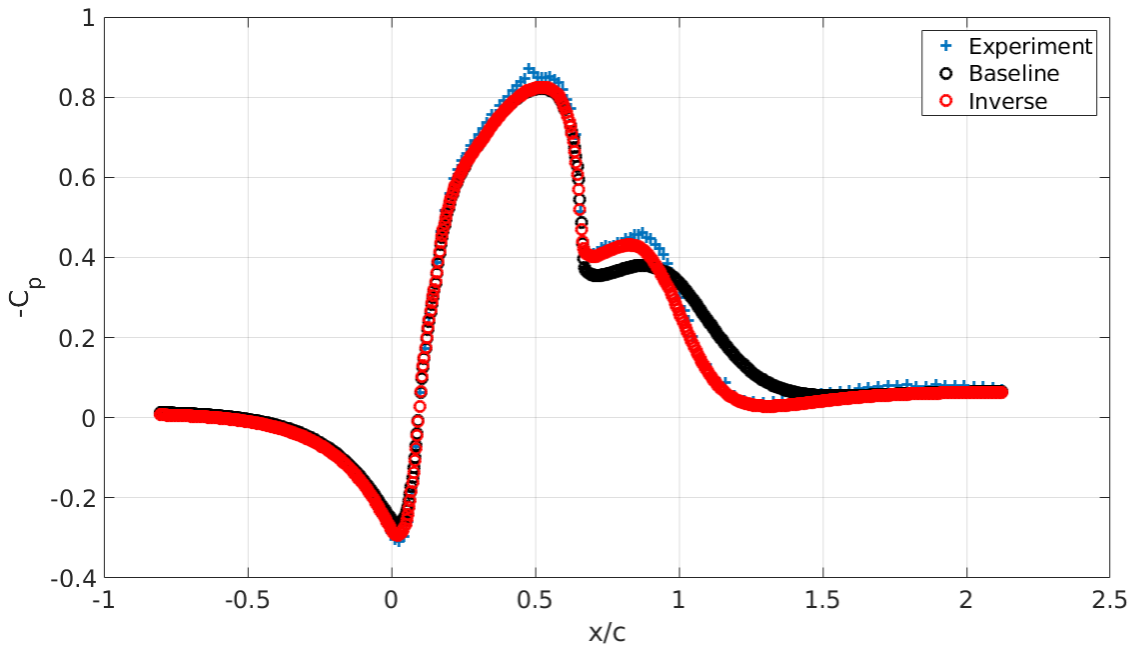

Figure 6.51: FIML-Direct C_p comparison to available data k_d for test case with $\lambda = 10^{-7}$

6.5 Summary

This chapter built on the 1D heat equation results by presenting the FIML-Classic, FIML-Embedded, and FIML-Direct results for RANS equations. All three methods were successfully demonstrated on the S809 airfoil using experimentally derived lift coefficient data. The full FIML-Classic methodology was performed for this application, followed by the FIML-Embedded, and FIML-Direct; all of which produced model augmentations based on the experimental data that improved predictions. Most exciting, the results of the simultaneous inversion of multiple S809 airfoils at various angles of attack was performed. Performing the inversion on three angles of attack produced an augmentation that corrected predictions for the en-

tire angle of attack range. This demonstrated the improved generalization provided by the FIML-Direct approach, especially when considering multiple conditions in a single inversion and augmentation. The inversion was then continued considering the S809 at seven angles of attack. This further improved the augmentation and demonstrated the improved flexibility of the FIML-Direct method since the design variables are consistent across any number of cases. The final augmentation, incorporating solely experimental lift coefficient data from the S809, was tested on a different airfoil: the S814. The augmented S814 airfoil predictions showed universal improvement across the angle of attack range, further demonstrating the generalization capabilities of the FIML-Direct approach.

Additionally, results for the FI-Classic and FIML-Direct approaches were applied to the NASA Langley wall mounted hump. The FI-Classic inversion was shown both with and without the regularization constant. This demonstrated the effect of the regularization constant on the result of the inversion. It was shown that without the regularization constant the inverse pressure distribution closely matched the data and objective function convergence was very good. However, it was shown that the inverse solution likely is not realistic for that case, as it was extremely high in magnitude and not localized close to where the SA model is likely inaccurate (adverse pressure gradients and separated regions). It was shown that the regularization constant allows the modeler to constrain the inversion to more reasonable corrections, without much penalty in accuracy with respect to the available data. The FIML-Direct results for the NASA Langley wall mounted hump were also presented in this chapter. It was shown that the FIML-Direct method can provide similar

inversion performance as the FI-Classic method, while simultaneously training the network and augmenting the model. The effect of considering the limitations of the network was also discussed. It was observed that the correction for FIML-Direct was substantially different than the FI-Classic results. This demonstrates the effect of considering the neural network limitations, as the inverse solution is now dependent on the feature selection, feature scaling, and network structure. Therefore, the optimal solution will likely differ in some applications such as the NASA hump, as the ideal solution adapts to the limitations of the chosen learning method.

Chapter 7: Conclusions

7.1 Summary

RANS turbulence modeling continues to be a challenging field. Despite decades of effort in the turbulence modeling community, RANS models continue to be deficient or unreliable for a variety of practical engineering applications. For example, RANS models typically fail to predict the correct forces on an airfoil at high angles of attack due to an incorrect prediction of the separation location. As DNS and LES are still decades away from being capable of simulating Reynolds numbers high enough for many applications, RANS models will remain the method of choice for the majority of engineering computations for some time. There has been, however, substantial investment in high fidelity numerical simulations and physical experiments. These efforts provide higher fidelity data that turbulence modelers can draw on to improve RANS models. Despite the availability of this data, progress is difficult, as measured quantities cannot be used to improve models directly. Thus, there is substantial motivation to utilize machine learning models to fully leverage this existing dataset of higher fidelity data to improve the RANS models where they are deficient. Several researchers have attempted to learn directly from the higher fidelity data, but this approach suffers from a lack of consistency between

the higher fidelity data and the model. The FIML approach addressed this concern by first performing an inference step (in the form of an inverse problem) to generate model consistent information from the higher fidelity data that can then be learned and applied in the form of an augmented model. Despite this improvement, there remained a lingering inconsistency with the FIML approach: the model consistent information was not always able to be easily learned, or learned sufficiently at all during offline training. It was this observation that motivated the present work; the disconnect between the model augmentation and the information generated by the inversion could be avoided if the learning algorithm were considered during the inversion.

The current effort attempted to resolve the inconsistency between the inverse problem and the model augmentation with the following approach, enumerated below as broad tasks.

1. The Stanford University Unstructured (SU2) open source CFD package was modified to perform the FIML process, referred to in this work as FIML-Classic. SU2 was selected due to its autodifferentiated (AD) discrete adjoint solver already configured for optimization problems. The use of AD for this effort enabled efficient experimentation for the new FIML approaches as much of the development effort for the adjoint solver is handled automatically through AD. The FIML method was implemented in SU2 for the first time, and the FIML-Classic approach was performed on a variety of canonically difficult RANS problems. Additionally, the FI-Classic information generated from 2D

airfoil simulation results was used to perform a feature selection analysis. This enabled the analysis of potential inputs to the model augmentation, and ultimately the selection of appropriate features and appropriate feature scaling for this problem.

2. A new FIML approach, termed FIML-Embedded, was proposed and analyzed. This unique approach was developed for this thesis and applied for the first time. The machine learning training algorithm was moved inside the model solver itself, such that the training is performed while the model variables are converging, and therefore the inverse procedure will account for the dynamics of the learning algorithm. Feedforward neural networks for regression were chosen as the machine learning algorithm, and the backpropagation training algorithm was implemented. The neural network is trained as the solver is converging, requiring several additional operations each solver iteration (feature scaling, backpropagation, and forward propagation). For the first time, this method was implemented both for a 1D heat equation model problem and in the modified SU2 code for RANS applications. The 1D heat equation provided a simple, easily implemented problem that fully demonstrates the method and numerics involved, while the SU2 implementation demonstrates that the method can be applied to complex RANS simulations.

3. Another new FIML approach, termed FIML-Direct, was proposed and analyzed. This new FIML approach was also developed in this thesis and applied for the first time. In this approach the weights of the neural network are con-

sidered the design variables of the inversion. This directly trains the neural network without the need for the backpropagation algorithm. The FIML-Direct method was also implemented for the 1D heat equation model problem and the modified SU2 package. Additionally, a methodology for incorporating information from multiple cases was developed for the FIML-Direct approach. For the first time multiple cases can be considered in the inference step simultaneously. For the 1D heat equation this enabled the simultaneous inversion of multiple temperature distributions, and for the 2D airfoil results the inversion was performed for up to seven cases simultaneously. The resulting augmentation was then shown to improve predictions on a different airfoil, not included in the training set. This effectively demonstrated the methodology for including multiple cases in the inversion and augmentation, which is a requirement for generating augmentations with adequate generalization for practical predictions. Additionally, for the first time this approach was also applied to other individual RANS cases, including the NASA Langley wall mounted hump and the RAE2822 transonic airfoil.

7.2 Key Observations

For the first time, it was shown that the FIML process can be improved by accounting for the limitations of the chosen machine learning algorithm inside the inversion process. By doing so, learning is performed during the inference step, the model correction will be optimally regularized for the chosen learning algorithm,

and there is the highest possible consistency between the inference step and the prediction environment. It was shown that the FIML-Classic inverse information can be difficult to learn due to insufficient regularization. This can be managed somewhat through careful construction of the objective function and the regularization constant (λ), but ultimately there is no guarantee that the optimal discrepancy is learnable by the learning algorithm selected by the modeler. Even when the inverse information can be learned, it was shown that there is a noticeable decrease in performance of the augmentation due to imperfect training, which illustrated the potential inconsistency between the inference and predictive environments in the FIML-Classic approach. Both new FIML approaches were shown to address this inconsistency by performing the learning in the inversion process.

First, general conclusions made for the two new FIML approaches and the FIML-Classic approach are discussed. Then, specific conclusions made from the FIML applications to the model problem and RANS cases are presented.

7.2.1 FIML-Embedded

The FIML-Embedded approach, developed and applied for the first time in this thesis, was shown to provide increased regularization over the FIML-Classic method. Additionally, by performing the backpropagation algorithm inside the solver during the inversion the inverse solution was guaranteed to be learnable. The optimal discrepancies for the FIML-Embedded applications showed increased regularization over the FIML-Classic procedure for all applications considered. It was also demon-

strated that the gradient could be efficiently computed through additional adjoint variables, with only a minor increase in computational cost over the FIML-Classic approach.

Unfortunately, there were also negative observations associated with this method. Performing the backpropagation algorithm in the solver itself increased the complexity of the simulation. This added complexity adversely affected the robustness of the inverse solution and is a drawback to the approach. Additionally, as currently formulated it is not readily apparent how to incorporate information learned from multiple cases with the FIML-Embedded approach. Possible solutions to this drawback were proposed, but have not yet been explored.

Nevertheless, it was observed that the FIML-Embedded approach could be performed with similar inverse problem convergence as the FIML-Classic approach. If the backpropagation algorithm converges sufficiently, the nonlinearity of the neural network is effectively hidden from the optimizer. By hiding this non-linearity it is expected that the FIML-Embedded approach could have substantial advantages over the FIML-Direct approach for large and/or complex network structures. For the attempted applications considered, however, the convergence of the FIML-Classic, FIML-Embedded, and FIML-Direct approaches were comparable.

7.2.2 FIML-Direct

The FIML-Direct approach, developed and applied for the first time in this thesis, was shown to give increased regularization over the FIML-Classic approach

for the RANS applications. More regularization, in fact, over the FIML-Embedded approach as well. It was demonstrated that the FIML-Direct approach could efficiently train the neural network by considering the weights as the design variables directly. This is somewhat of a surprising result, as typically thousands of iterations of the backpropagation algorithm are required to train a neural network from static data. It was shown that by considering the weights as the design variables in the FIML cost function the network could be minimized in a manageable number of evaluations (typically 10-20 observed). This is an exciting observation, and because of this result the FIML-Direct approach provides an alternative, efficient, and advantageous approach to training neural network augmentations that improve models with data.

It was also demonstrated that the FIML-Direct approach has a substantial advantage over the FIML-Embedded procedure because the design variables in the inversion are the same across multiple cases. The model discrepancy is not expected to be the same across multiple cases, but the weights of the neural network are (a single augmentation that improves predictions for multiple cases is desired). Therefore the inverse problem can be posed for an unlimited number of cases simultaneously by minimizing a composite cost function. For the first time, the FIML inversion was performed considering multiple cases simultaneously. It was observed, both for the model problem and the RANS applications, that the data from multiple cases could be successfully incorporated into a single augmentation via the FIML-Direct approach with simultaneous inversion. Additionally, it was demonstrated with the model problem that it is not necessary to compute the full gradient (which is costly),

but alternatively the partial gradient could be computed and the stochastic gradient descent method used to train the augmentation. This method could be advantageous when considering a large number of cases.

7.2.3 1D Heat Equation Model Problem

The 1D heat equation model problem was shown to be a computationally efficient problem to evaluate the FIML procedures and make direct comparisons between algorithms. The FIML-Classic, FIML-Embedded, and FIML-Direct algorithms were all implemented for this problem, and all produced model augmentations that dramatically reduced the error of the augmented model. Thus it was demonstrated that FIML-Embedded and FIML-Direct approaches can both produce useful model augmentations during the inversion step, unlike the FIML-Classic method which requires offline training following inversion. The exact analytical solution for the optimal correction is known for the 1D heat equation case. It was shown that the FIML-Classic and FIML-Direct methods, when applied to a single case, can find the optimal correction distribution. The FIML-Embedded implementation did not find the true distribution, but was still capable of producing a model augmentation that substantially reduced prediction error. The objective function convergence of the three methods was also compared, and it was shown that, as expected, the convergence of the FI-Classic approach was by far the best because the limitations of the learning algorithm are neglected in the inverse problem of FI-Classic. The convergence of the FIML-Direct method was perhaps most surprising, as it demonstrated

excellent convergence despite learning during the inversion step.

Model augmentations considering information from multiple cases were also generated and tested for the FIML-Classic and FIML-Direct methods. For the FIML-Classic method, this requires the solution of the inverse problem for each case to generate training data. The data from each case is then compiled into a single training set, and a model augmentation is trained offline. For the FIML-Direct case, it was shown that multiple cases could be considered *simultaneously*, such that the inversion not only learns during the inversion, but also learns information from multiple cases. For the first time, the inverse problem can incorporate information from multiple cases simultaneously. The augmentation was trained using several constant T_∞ distributions, and was shown to improve predictions for constant T_∞ distributions not in the training set, and also on variable T_∞ holdout data.

In summary, the conclusions from the 1D heat equations were:

1. Learning during the inversion process is feasible, either with FIML-Embedded or FIML-Direct with only a minor penalty on achievable accuracy of the resulting model augmentation over the FIML-Classic approach.

2. The FIML-Direct method exhibits convergence characteristics comparable to the FIML-Classic approach, while simultaneously learning the model augmentation. This is unexpected and exciting, given the number of iterations often required to train neural networks.

3. The FIML-Direct method can learn from multiple cases simultaneously, considering the limitations of the network during the inversion, and producing

the optimal model augmentation for the chosen network structure.

4. Only a limited number of training cases are required to produce model augmentations that improve predictions for cases that interpolate in, or mildly extrapolate from, the feature space of the training set.

5. For FIML-Direct, the full gradient computation is not required to produce useful augmentations. The stochastic gradient descent method was demonstrated using a single case as a batch, which may have practical advantages over the full gradient computation when considering a large number of cases.

7.2.4 2D Incompressible Airfoil Results

All three FIML approaches were applied to the S809 incompressible wind turbine airfoil. The Spalart-Allmaras turbulence model with the algebraic transition model was used (SA-BC). At high angles of attack it was shown that the baseline SA-BC model results in a substantial error in predicted lift coefficient, due to a smaller separated region than observed in experiments. The inverse solutions for all three methods demonstrate that this error is due to an over-production of eddy viscosity on the upper surface of the airfoil. It was shown that the FI-Classic approach could correct this error, but that there was a substantial penalty due to imperfect offline learning that resulted in increased error in the resulting augmentation. Thus an issue with FIML-Classic was demonstrated: by not accounting for the limitations of the neural network in the inversion there is an inconsistency between the inverse solution and the model augmentation. The FIML-Embedded approach was

performed and it was shown that this approach could produce a model augmentation in the inversion step that substantially reduces prediction error. However, it was also shown that the FIML-Embedded procedure has substantial robustness issues that prevented the inversion from fully converging. The FIML-Direct application for this single angle of attack, however, did not show this robustness issue. The resulting model augmentation for the FIML-Direct procedure provided a substantial reduction in error while regularizing the correction. It was shown that the regularization parameter could be decreased because the required regularization was being provided by considering the limitation of the neural network in the inversion process.

In summary, the conclusions from the S809 FIML applications at a single angle of attack were:

1. The FIML-Classic procedure can result in an inconsistency between the inverse solution and the model augmentation due to imperfect learning.

2. Both new methods, FIML-Embedded and FIML-Direct, were applied to RANS cases for the first time and were shown to produce useful model augmentations for the RANS problems considered.

3. The FIML-Embedded approach, as currently formulated, has significant robustness issues that can complicate the inversion procedure.

4. The FIML-Direct approach demonstrates surprisingly efficient training of the neural network on a RANS application, consistent with the heat equation conclusion.

7.2.5 2D Incompressible Airfoil Results for Multiple Cases

For the FIML-Direct method, a model augmentation was generated from the S809 airfoil considering experimentally observed lift coefficient data for up to 7 angles of attack simultaneously. A composite objective function was constructed, and initially an augmentation was produced considering three angles of attack. It was shown that the FIML-Direct algorithm could train a neural network that improved force predictions for the three angles of attack in the training set, and also for four additional angles of attack that were not. This further demonstrates that only a limited amount of higher fidelity data is required to produce useful model augmentations. In this case, the only higher fidelity data used was the experimental lift coefficient at three angles of attack, and the resulting model augmentation reduced model error for the entire S809 angle of attack range.

The model augmentation training was then continued using the FIML-Direct method and all seven S809 angles of attack. Minor improvement in the model augmentation was shown when considering this extra information. This model augmentation, trained exclusively on S809 airfoil data, was then tested on a different geometry: the S814 airfoil. It was shown that the model augmentation substantially improved the S814 predictions where the model was deficient (high angles of attack), but did not hurt accuracy where the model was already accurate (lower angles of attack).

In summary, the FIML-Direct application to the S809 at multiple angles of attack resulted in the following conclusions:

1. The simultaneous inversion of multiple RANS cases, performed for the first time in this thesis, produces model augmentations with good generalization capabilities.

2. The FIML-Direct algorithm exhibits good convergence during the inversion, even when considering multiple RANS cases.

3. Only a limited quantity of higher fidelity information is required to produce useful model augmentations via the FIML-Direct approach.

7.2.6 Transonic Airfoil Results

The RAE2822 transonic airfoil was considered for both the FI-Classic and FIML-Direct approaches. For the conditions considered this airfoil has a substantial supersonic region which terminates in a shock boundary layer interaction. Following the shock there is a subsonic region with an adverse pressure gradient. The baseline SA model showed a small error in predicted shock location, and the pressure distribution was poorly predicted in the vicinity of the SWTBLI.

Wind tunnel pressure coefficient data was used to formulate the objective function, and the FI-Classic and FIML-Direct methods were performed. The model results following the FI-Classic inversion compared very well with the experimental data, and there was a large correction in the SWTBLI region. This modified the location of the shock and the response of the boundary layer to the shock, which smeared the pressure rise over a larger area. This is consistent with the dynamics of a smeared shock-foot that is a characteristic of transonic airfoil SWTBLI. The

FIML-Direct method yielded a model augmentation that substantially improved the prediction of the shock location, but was not able to improve the pressure distribution through the SWTBLI as well as the FI-Classic inversion. This is likely due to the increased regularization of the FIML-Direct approach that prevented the precise correction required in the region of the SWTBLI. It is possible that the features used, chosen for the S809 case, are not optimal for SWTBLI cases. It is also possible that a more rigorous investigation into the neural network structure (size of hidden layers, number of neurons, choice of activation function, etc.) could produce better results.

In summary, the conclusions from the RAE2822 case were:

1. The SA baseline model incorrectly predicts the shock location, and does not accurately model the pressure distribution through the smeared shock-foot.

2. The inverse solution from the FI-Classic approach can accurately correct the observed SA model deficiencies, but this correction requires relatively intrusive corrections at very precise locations, indicating that the inverse solution is poorly regularized.

3. The FIML-Direct method can be successfully applied to SWTBLI and move the shock location, but it was not possible to adjust the pressure distribution through the SWTBLI for the chosen learning algorithm, neural network structure, and features.

7.2.7 NASA Langley Hump Results

The NASA Langley wall mounted hump was also considered. This is an additional 2D, incompressible, separated flow application that has been shown to be difficult to successfully model using RANS. Experimentally obtained pressure distribution was used as the higher fidelity data, and both the FI-Classic and FIML-Direct approaches were applied. Results for the FI-Classic approach both with and without the regularization constant were presented.

In summary the conclusions for the NASA Langley hump results were:

1. The regularization constant is critical to producing meaningful model augmentations; without this parameter the inverse solution can be unnecessarily intrusive.

2. When considering the limitations of the neural network, the model correction from the FIML-Direct approach can be substantially different than the correction resulting from FI-Classic method.

7.2.8 Other RANS Applications

Two other RANS applications were considered, the 2D hypersonic wedge and the 3D Onera M6 wing (results in Appendices A and B). The FI-Classic procedure was performed, and in both applications the inverse solution more closely matched the higher fidelity data. These cases further demonstrate the capability of the inverse procedure to generate corrections that improve the model for a wide range

of historically difficult RANS cases.

7.3 Recommendations for Future Work

In this section recommendations for future work are presented. The FIML approach up this point has been developed and demonstrated on a variety of difficult RANS applications. There remains the task of constructing a robust augmented model that is shown to improve predictions over a broad range of applications. Such an effort would not be trivial, as it would be necessary to construct inverse problems that adequately cover the feature space of the problem of interest, and despite the relative efficiency of the FIML approach over sampling methods substantial computational resources would still be required. Nevertheless the present work and the FIML approach in general has certainly demonstrated that such an effort could be successful.

For the two new FIML methods proposed, implemented, and analyzed in the present effort, the FIML-Direct approach is comparatively more developed than the FIML-Embedded method, as evidenced by the results in Chapters 5 and 6 and the discussion in detail below.

1. Feature Selection: Chapter 6 presented the feature selection approach used in the current effort, and discussed the reasons that conclusions concerning feature selection are difficult. In particular, the best features will be application dependent. The limited feature selection study in this effort focused on FI-Classic inversion results for the S809 airfoil, and it is not expected that

conclusions from this limited dataset apply to RANS applications in general. Additionally, the appropriate features may depend on the chosen machine learning algorithm. For FIML-Embedded and FIML-Direct feature selection is particularly important, as the inversion cannot be successful unless the features are appropriately chosen since the limitations of the learning algorithm are considered in the inversion. In particular, it was observed that the FIML-Direct correction for the NASA Langley hump case was particularly intrusive compared to the regularized FI-Classic result. It is likely that improved feature selection, tailored to this application, could produce a less intrusive augmentation than achieved in this effort.

2. Feature Scaling and Outlier Rejection: A related issue related to the features is feature scaling and outlier rejection. Extreme outliers were observed in the selected features that, if not properly accounted for, adversely affected the training of the neural network. In the current effort, the features were mapped to a normal distribution. This minimized the effect of outliers and enabled sufficient training for the FIML approaches presented in the current work. However, it is likely that the treatment of outliers could be improved, and this would likely improve the performance of all three FIML approaches. Perhaps a filtering algorithm could be implemented. The current effort tested a filtering method based on a shielding function often used in delayed detached eddy simulations (DDES)[1], but it was found that this did not sufficiently

[1]The SA-DDES shielding function, f_d, [113] was used to filter out all points not in the boundary layer for an S809 airfoil at high angle of attack. This method was found to not sufficiently minimize

reduce the effect of outliers and therefore was not used for any of the results presented in this thesis. A more robust treatment of outliers would almost certainly improve performance.

3. Learning Algorithm: As with all machine learning applications, the results are highly dependent on the choice of machine learning algorithm. Only neural networks were considered in the present effort, but many other algorithms could be applied. In particular for neural networks, the modeler has many choices including the number of hidden layers, number of nodes, choice of activation function, etc. There are also many types of neural networks, and only fully connected feedforward neural networks were considered. To fully explore the capability of both FIML-Embedded and FIML-Direct a more rigorous investigation into all of these choices is required.

4. FIML-Embedded Development: The FIML-Embedded approach was found to be difficult to perform for RANS applications. To succeed, the implementation of the backpropagation algorithm must not adversely affect the solver routines. Unfortunately, this was observed in the RANS applications presented in this thesis resulting in premature termination of the inversion. Feature scaling may have some influence on this issue, as it was found that mapping the features to a normal distribution substantially improved inversion performance. More robust treatment of outliers may keep intermediate weight values from

the effect of outliers, and also filtered out some of the separated region which could remove some areas from consideration that require correction.

predicting excessive corrections that prevent convergence of the flow solver. Thus, more effective outlier treatment may also resolve this robustness issue with the FIML-Embedded approach.

5. FIML-Embedded for Multiple Cases: The ability to perform the inversion incorporating the information from data for multiple cases has not been developed for the FIML-Embedded approach. Currently, this gives a strong incentive to using the FIML-Direct approach as this capability is likely required in order to adequately cover the feature space for a practical (useful) augmentation. Several possibilities for how to improve on the current FIML-Embedded methodology were proposed in Chapter 3, but these concepts remain unexplored.

6. Learning Without Forgetting: Perhaps the biggest obstacle to obtaining an augmentation that improves a RANS model for a wide range of applications is a practical issue. How can a single researcher or team practically or efficiently augment a model that applies to every case in an area of interest? Even if such a model is developed, how do you update such a model if it is found to be deficient in a new application? Currently, for FIML-Classic, to incorporate new information all the previous information is also required. It is certainly possible to retrain an existing augmentation with new data, but if the old data is not included in the updated training set the augmentation will "forget" the old data, such that the updated augmentation will have diminished performance on the old training set. Therefore, to retrain the augmentation with

new information, the new information must be added to the old and a new augmentation learned. This is also an issue with the FIML-Direct approach, as new data can be incorporated, but the inversion must be continued with all of the cases (old and new) as was demonstrated with the S809 airfoil augmentation in Chapter 6. Clearly, it will be impractical to share, store, and distribute large quantities of such data with the turbulence modeling community. Therefore, there is a need for algorithms that can incorporate new information without forgetting the old, such that a model augmentation can be updated efficiently without forgetting what has already been learned. This is termed "learning without forgetting", and is the concept of incorporating new data into an already trained model without access to the original data used to learn that model. Some algorithms exist [114, 115], and this capability in the FIML framework would greatly increase the ability for a community to develop useful model augmentations collaboratively.

7. Uncertainty Quantification: The FIML implementations in this work largely neglected formal uncertainty quantification. This is a major shortcoming, as the model and the data both have uncertainties that should be rigorously evaluated and quantified. In some applications, the FIML-Classic approach used a Bayesian formalism to evaluate model posterior uncertainties [39]. For the applications of the new methods in this thesis, the Bayesian framework was not used, and only a cursory accounting of the prior model confidence was utilized through the regularization constant. Additionally, neural networks can incor-

porate Bayesian uncertainty quantification by formally assigning uncertainty distributions to their parameters. Thus the model posterior probability can be evaluated. Utilizing Bayesian neural networks in the FIML approach could help avoid over-fitting and enable more successful learning from small datasets. Formal uncertainty quantification for the FIML-Direct and FIML-Embedded methods remains an area for future work.

Appendix A: Hypersonic Wedge

A.1 Overview

Accurate prediction of shock wave turbulent boundary layer interactions is also a difficult problem for the design of hypersonic vehicles. At these much higher Mach numbers concave corners can create very strong SWTBLI, and the prediction of the surface heat flux and pressures involved in these interactions can have a dramatic impact on design decisions for these vehicles. RANS models have difficulty predicting hypersonic SWTBLI due to the very strong adverse pressure gradients and strong nonequilibrium effects in the boundary layer.

To examine the FIML ability to improve RANS predictions for a hypersonic SWTBLI a 2D compression corner was modeled. The 36° compression corner presented by Holden et al. [116, 26] was modeled using the SA turbulence model. Incoming Mach 11.3 air encounters a flat plate at 0° incidence and a turbulent boundary layer develops[1]. This turbulent boundary layer encounters a 36° ramp which results in a SWTBLI that separates the boundary layer upstream of the

[1]The experiment used natural transition [116], but the SA-BC transition model was not used for this case due to the poor performance of this model for these hypersonic conditions. Therefore the CFD results are fully turbulent.

ramp corner. The experiment was performed at the CUBRC shock tunnel facilities, and the model was wide enough that three-dimensional effects (from finite width) are not expected to be significant. It has been shown [116, 26] that RANS models are unable to accurately predict the separation length, pressure distribution, and heat flux distribution in the interaction region and downstream on the ramp. The geometry for this experiment is illustrated in Figure A.1.

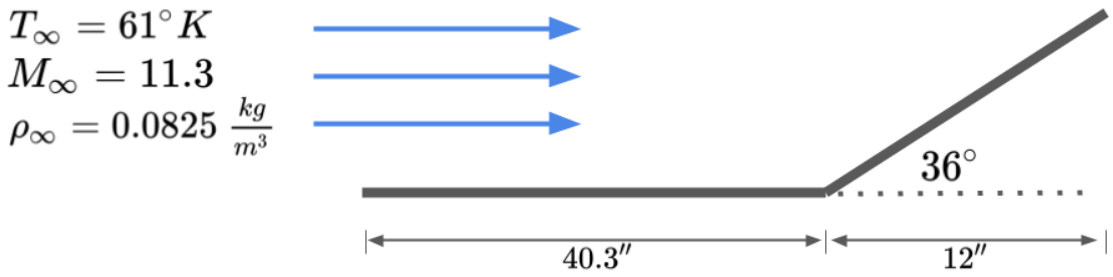

Figure A.1: Illustration of geometry for hypersonic wedge case.

The SA model used to this point (SA-noft2) [17] was found to produce poor results for this application. This model predicted no separated region and thus the physical dynamics of interest for this problem were not predicted at all. Another variant of the SA model, the SA-SALSA [83], was shown to produce better predictions for shock boundary layer interactions, and has been implemented in codes typically applied to hypersonic applications [117]. The SU2 v5.0 package did not include the SA-SALSA variant, but it is a relatively straightforward modification as outlined in Chapter 4, and it was implemented for use on this hypersonic SWTBLI application.

Figures A.2, A.3, and A.4 show the Mach, C_P, and temperature contour pre-

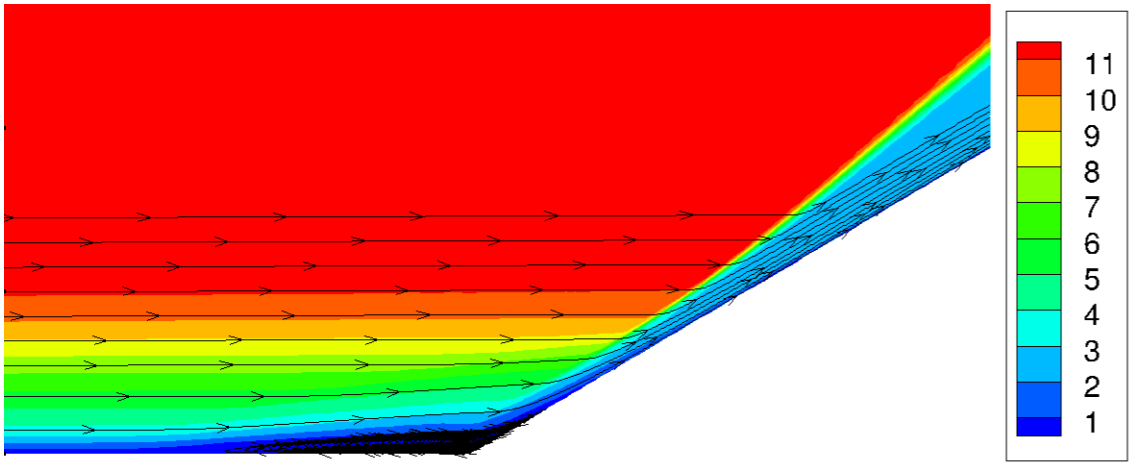

Figure A.2: Mach number contours for baseline SA-SALSA model.

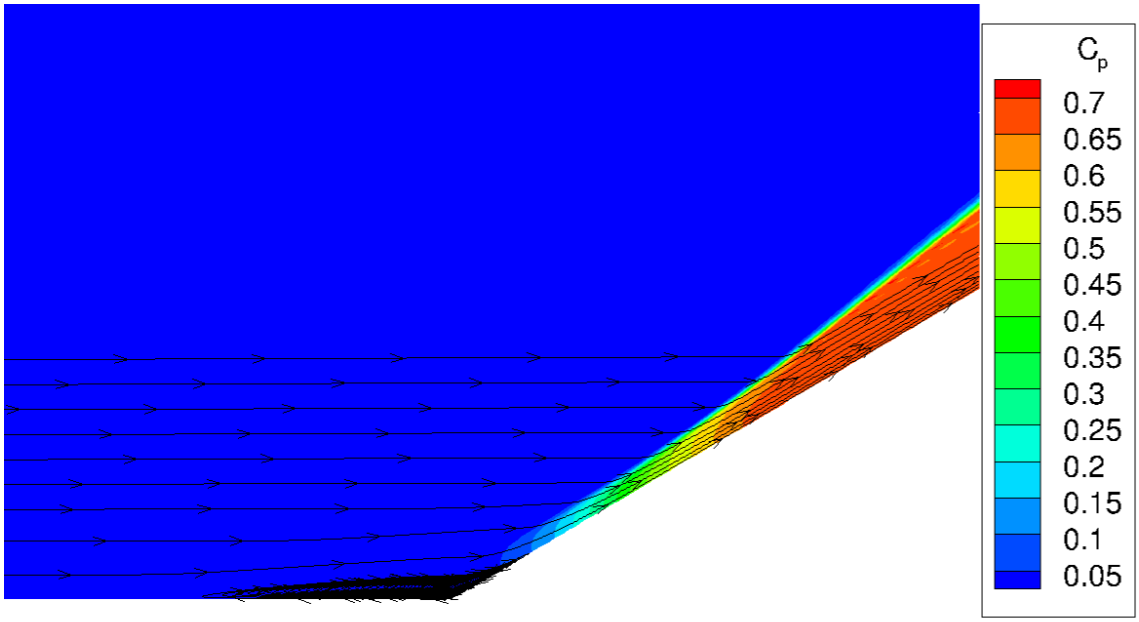

Figure A.3: Pressure coefficient contours for baseline SA-SALSA model.

dictions for the baseline SA-SALSA model at the interaction region. Note the computational domain extends much further in the left and right of the region shown in order to develop the incoming boundary layer from freestream conditions on the plate, and to predict the flow conditions downstream of the interaction region on the

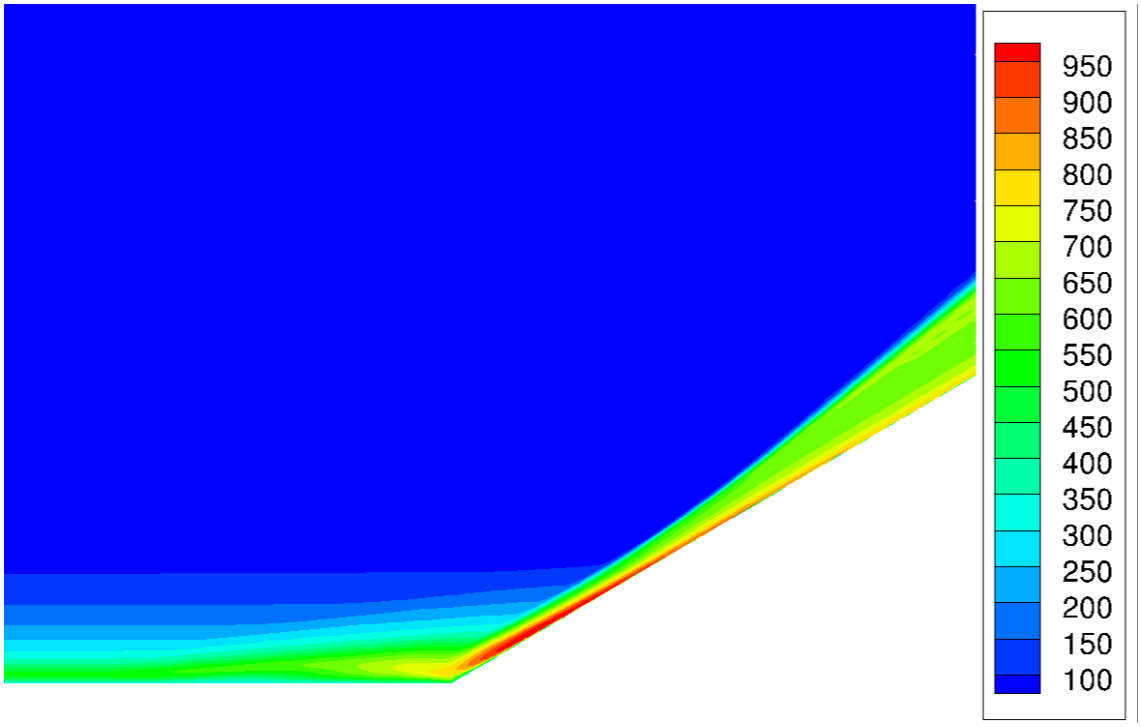

Figure A.4: Temperature ($^{\circ}K$) contours for baseline SA-SALSA model.

wedge. Note that in Figure A.2 the boundary layer separates in front of the corner and a recirculating region develops. A shock is shown in the boundary layer (separation shock) from the separation location that then impinges on the much stronger shock formed by the wedge. Note that there is a large range of pressure coefficient magnitudes as shown in Figure A.3. Additionally, extremely high temperatures are predicted in the interaction region as shown in Figure A.4. These temperatures are greatest near the reattachment location on the wedge. The wall is modeled as an isothermal boundary with $T_{wall} = 300^{\circ}K$, and this extremely high temperature produces very high heat flux at the wall.

A.2 FI-Classic

The FI-Classic procedure was applied to the hypersonic wedge case using experimental pressure coefficient information C_P. This pressure distribution was interpolated to the wall boundary nodes in the computational domain to produce the target pressure distribution $\widehat{C}_P$. Note that for this application the range of C_P values is very large. Upstream of the wedge the pressures will be low as there will only be a very weak compression from the boundary layer growth, and a stronger, but still relatively small compression from the separation shock upstream of the wedge corner. Downstream of the wedge corner the pressures will be dramatically higher from the compression of the very strong oblique shock to turn the air to the wedge angle. To account for this large magnitude range of the target (experimental) pressure distribution, the objective function was normalized by the target distribution at that point in the domain. The objective function is then given by Equation A.1.

$$J_c(\beta) = \sum_{i=1}^{k} \left(\frac{\widehat{C}_{p_{exp},i} - C_{P,i}(\beta)}{\widehat{C}_{p_{exp},i}} \right)^2 + \frac{1}{2}\lambda \sum_{j=1}^{n} (\beta_j - 1.0)^2 \tag{A.1}$$

The convergence history of Equation (A.1) is shown in Figure A.5. Note that the convergence is generally quite good, but requires more evaluations than other cases presented in this thesis. Additionally, the large increases observed in the intermediate evaluations (12-21) are due to line search steps that resulted in separation lengths larger than the experiment. This gives a large value for the numerator in the first term of Equation A.1 that is normalized by a very small value giving a very large objective function value. This is not an issue, as the optimizer

quickly found a lower objective value in the subsequent step in the line search. Despite the good trend in objective function value, the inversion was terminated after 30 evaluations because subsequent attempts to improve the objective further were not successful due to flow solver convergence issues.

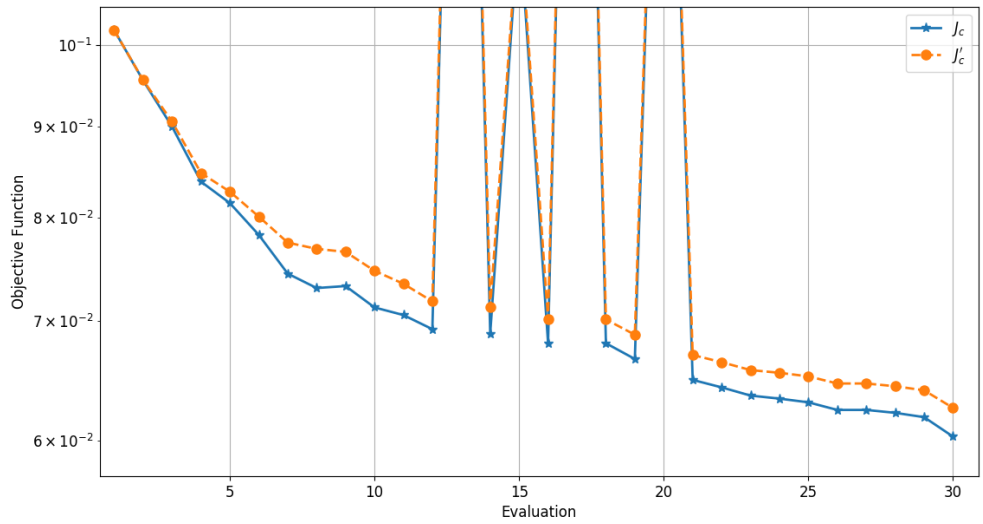

Figure A.5: Objective function convergence for FI-Classic hypersonic wedge application.

The cause of these issues is apparent when the correction is visualized (Figures A.6 and A.7). Note that at the interaction region (Figure A.6) the magnitude of the correction is very large ± 5 and is very localized. The correction in this case is required to be very large and in very specific locations in the interaction region. Note that there is a very large correction at the separation location, followed by a region where $\beta \approx 1.0$. Then, closer to the wall the shear layer above the separated region shows $\beta < 1.0$ corresponding to destruction of eddy viscosity, while the adjacent separated region is showing a very large positive correction indicating the

eddy viscosity should be increased there. Subsequent evaluations of the optimizer attempted to produce even larger magnitude corrections than shown in Figure A.6, however this resulted in numerical convergence issues of the flow solver and therefore the inversion was terminated.

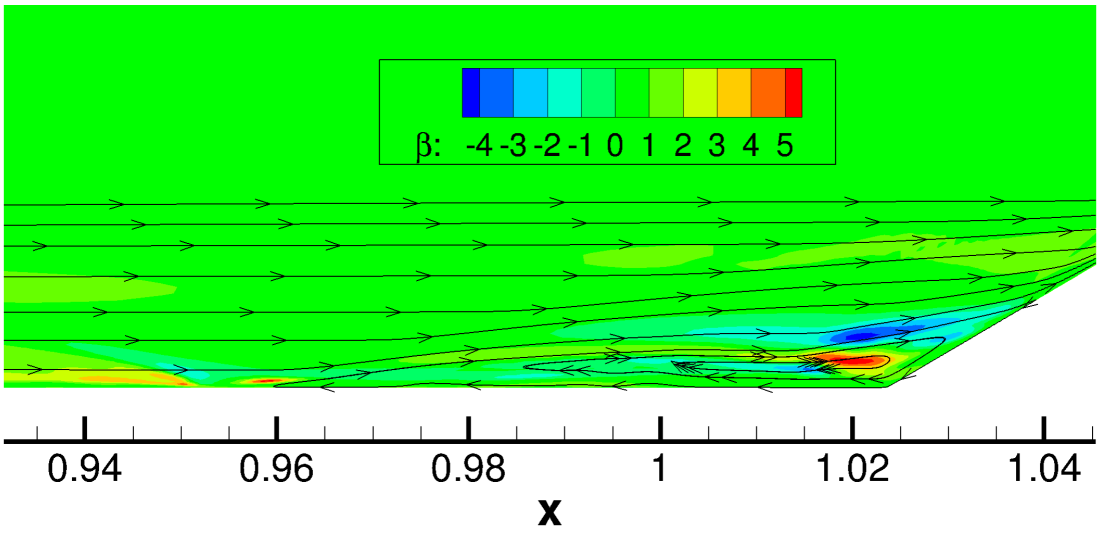

Figure A.6: Correction (β) at interaction region for hypersonic wedge FI-Classic application.

Nevertheless, the FI-Classic inversion substantially improved the pressure distribution, and the FI-Classic inverse solution more closely matches the experimental data as shown in Figure A.8. Note that the separation length is predicted very well by the inverse solution, and the resulting pressure distribution downstream of the interaction region is also improved. However, the magnitude of the pressure in the separated region is improved, but is still substantially lower than the experiment.

Experimental data for wall heat flux is also available for this experiment. The heat flux distribution results are shown in Figure A.9 in units of W/m^2

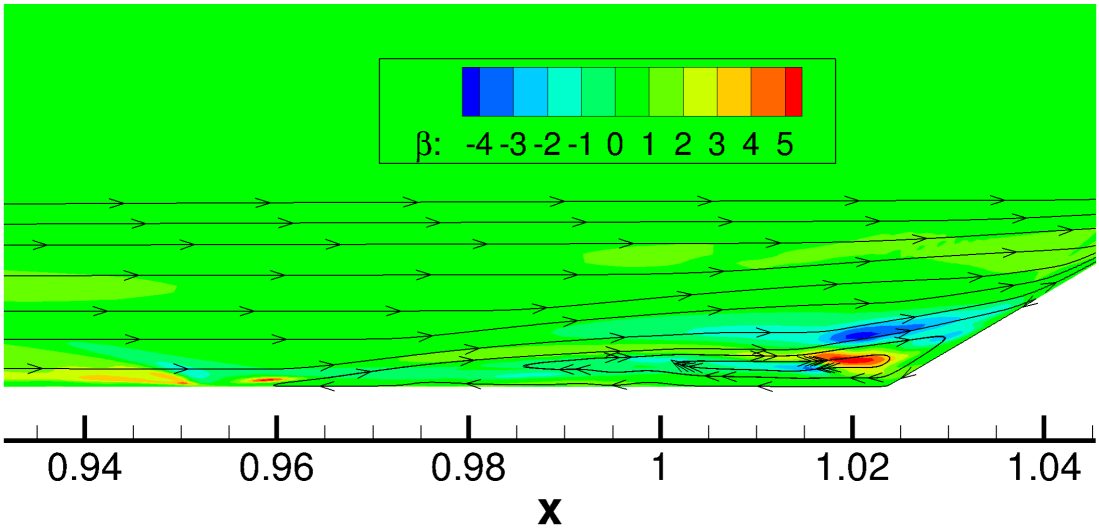

Figure A.7: Correction (β) upstream of interaction region for hypersonic wedge FI-Classic application.

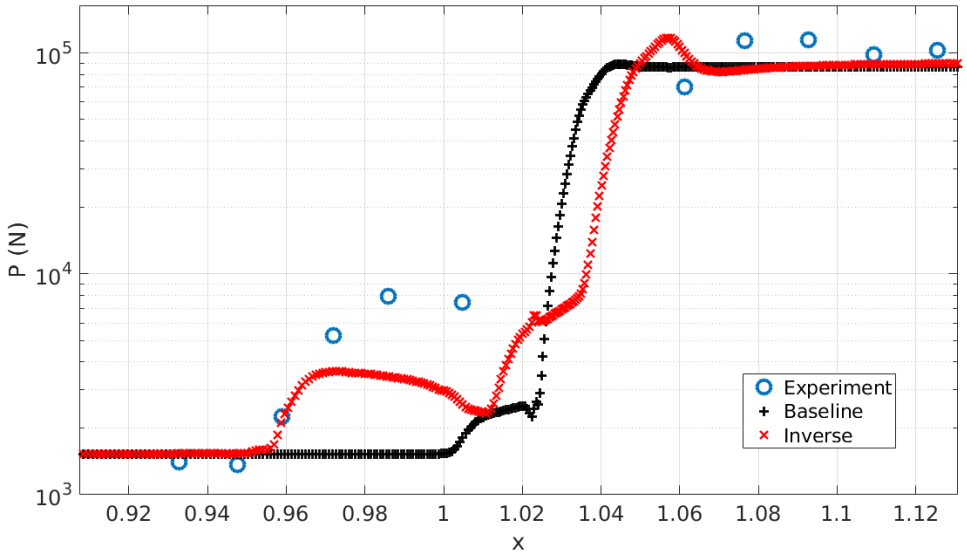

Figure A.8: Pressure distribution near the interaction region for FI-Classic hypersonic wedge application.

Note that the heat flux results show positive and negative trends. The baseline SA-SALSA model correctly predicts the heat flux on the flat plate ahead of the

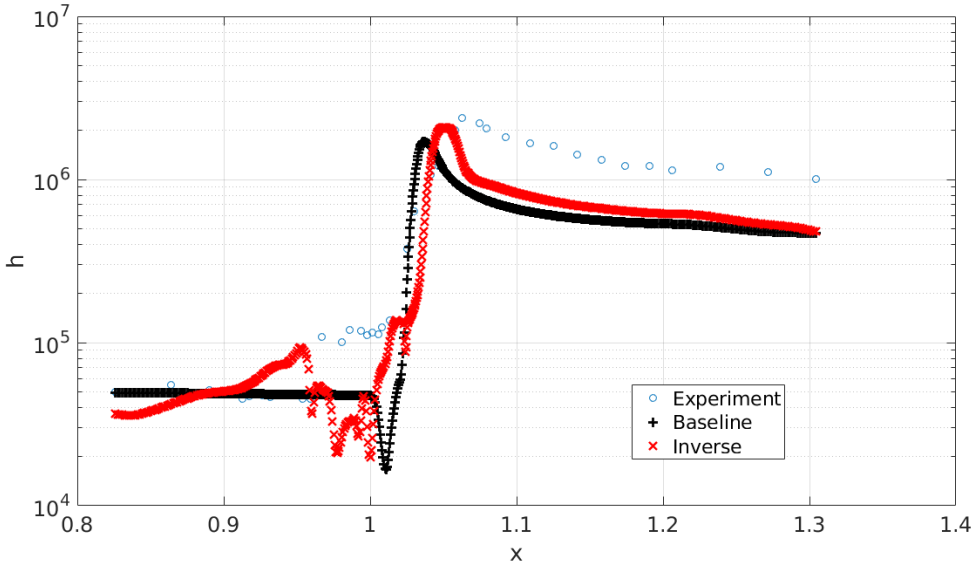

Figure A.9: Wall heat flux near the interaction region for FI-Classic hypersonic wedge application (W/m^2).

interaction region. The large corrections in this region, that are shown to help the pressure distribution, negatively impact the heat flux prediction on the flat plate. Additionally, the heat flux prediction was adversely affected in the separated region. Clearly this is problematic. Nevertheless, heat flux distribution downstream of the corner shows substantial improvement over the baseline model. The peak heat flux prediction is poor for the baseline SA-SALSA model, but much improved for the inverse solution.

A.3 Conclusions

A hypersonic wedge producing a SWTBLI interaction was considered. Compared to the other SWTBLI cases presented in this thesis, this case produces a

much stronger shock and therefore it is expected that the required correction to the baseline model will be comparatively larger. The baseline SA model was shown to produce poor results (no separated region), and therefore the SA-SALSA turbulence model was implemented in SU2. The SA-SALSA method predicted a separated region, but the separation length and pressure throughout the interaction region was still poorly predicted.

The FI-Classic method was applied. Experimentally measured pressure distribution was used to construct the cost function, and the error term was normalized by the local experiment C_P in order to account for the large range in C_P magnitude for this case. The inversion was performed and the objective function showed good improvement. However, it was found that eventually the required correction adversely affected the convergence of the flow solver, and therefore the inversion terminated while the cost function was still improving. Nevertheless, substantial improvement in the pressure distribution was observed prior to these numerical issues. The required correction was very large in magnitude. Specifically in the interaction region, the correction changes sign and magnitude at very specific locations in the separated region. This poor regularization could possibly make this correction difficult to learn, either offline in the FIML-Classic framework, or during the inversion for FIML-Direct applications.

Additionally, it was observed that the inverse solution negatively affected the prediction of the wall heat flux on the flat plate and in the interaction region. Despite this, the heat flux prediction downstream of the interaction region was improved. Overall, the heat flux results are disappointing, however, it is likely that

the inverse solution could be improved by including the experimental heat flux data in the objective function. Exploration for the best objective function for hypersonic SWTBLI cases remains an area for future work.

Appendix B: Onera M6

B.1 Overview

The FIML framework was applied to the Onera M6 transonic wing. Unlike the other RANS applications in this thesis, this case is three dimensional. This is a good demonstration of the advantage of implementing the FIML framework in the SU2 package, and demonstrates that the FIML approach can be applied to more complicated geometries, and to cases with large grid sizes sufficient to model practical engineering applications.

The Onera M6 transonic wing is a low aspect ratio design with experimental data presented by Schmitt and Charpin [118]. For the case considered, the freestream Mach number is $M_\infty = 0.84$ and the wing is at 3.06° angle of attack. Under these conditions, the wing is transonic and experimental data shows a "lambda" shock structure on the upper surface of the wing [119], with the name referring to the shape of the shock structure on the upper surface of the wing. Similar to the RAE2822 airfoil this establishes SWTBLIs on the upper surface that have strong adverse pressure gradients that are difficult to accurately predict with RANS models.

Predictions for the Onera M6 wing were made using the SA turbulence model. Figure B.1 displays the surface pressure contours for this case. Note the lambda

shock structure and the adverse pressure gradient behind the shocks similar to the RAE2822 case.

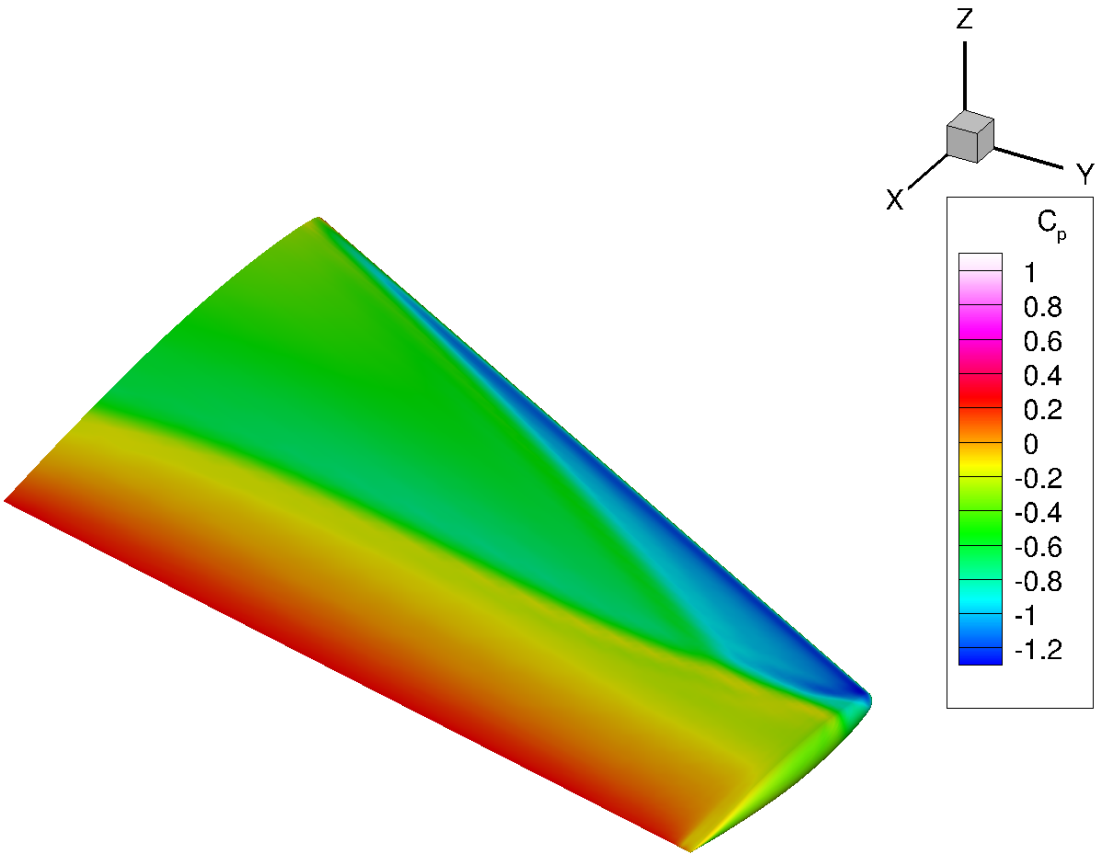

Figure B.1: Surface pressure C_P contours for baseline SA model on upper surface of Onera M6 wing.

B.2 FI-Classic

The FI-Classic procedure was applied to the Onera M6 wing. The experimental pressure distribution was used as the higher fidelity data. This data is relatively sparse and only available at 7 chordwise stations along the wing [120, 118]. A 2D

interpolation was performed to find the expected (interpolated) pressure coefficient at each node on the surface of the wing ($\widehat{C}_P$). Note that no extrapolations were performed, so a portion of the wing near the root and tip are excluded from the objective function calculation. The objective function for this case is the same as for the RAE2822 FI-Classic implementation, and given by Equation B.1. For this case, $\lambda = 10^{-6}$ was used.

$$J_c(\beta) = \sum_{i=1}^{k} (\widehat{C}_{p_{exp},i} - C_{P,i}(\beta))^2 + \frac{1}{2}\lambda \sum_{j=1}^{n} (\beta_j - 1.0)^2 \tag{B.1}$$

The objective function history during the inversion ($\min_\beta J_c(\beta)$) is shown in the bottom right panel of Figure B.2, along with the pressure distribution at span locations (η) where experimental data is available.

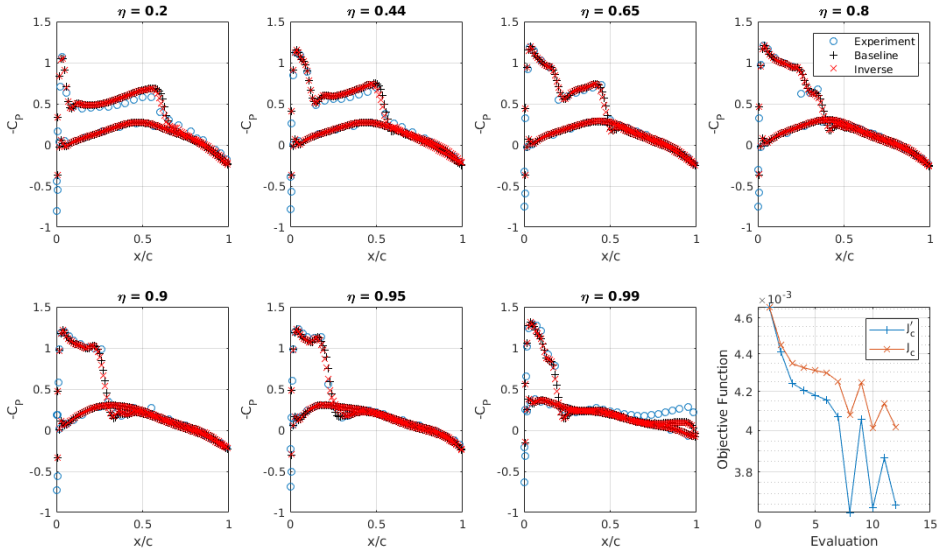

Figure B.2: FI-Classic results for Onera M6 wing.

The largest improvement is shown at stations $\eta = 0.2,\ 0.44$, and 0.95, better

illustrated in in Figures B.3, B.4, and B.5. This improvement is primarily due to moving the aft shock location slightly more forward at these locations.

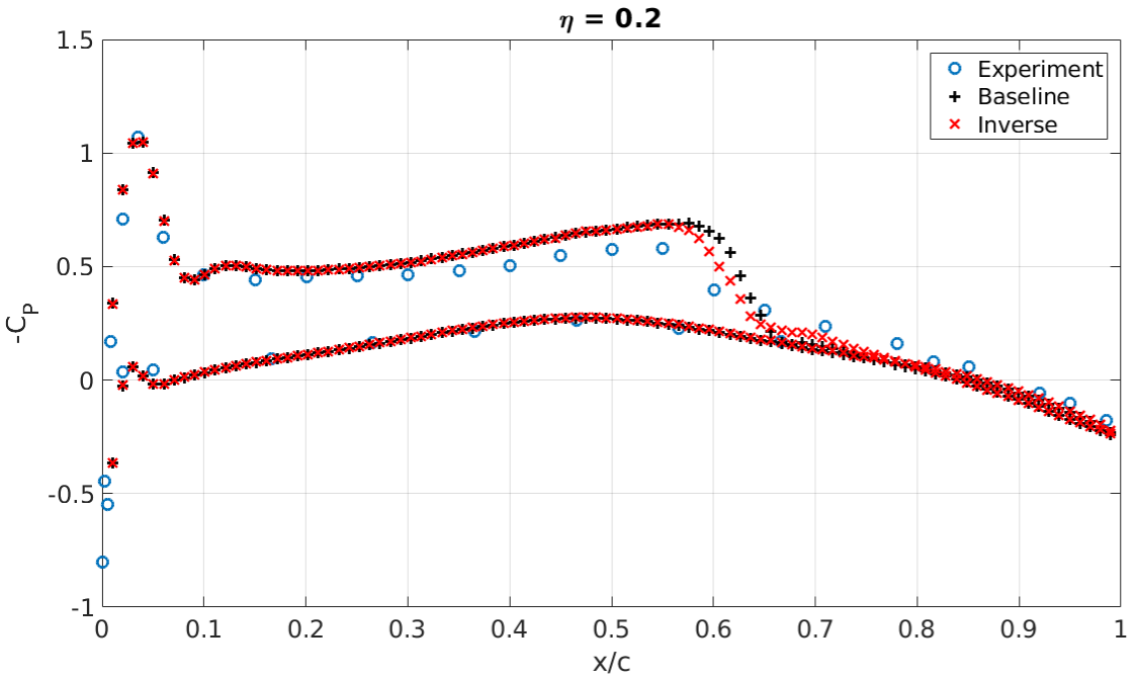

Figure B.3: Inverse results at span $\eta = 0.2$.

Additionally, the improvement in prediction can be visualized by looking at the error scatterplot shown in Figure B.6, which again shows that the most improvement is near the mid-chord of the wing closer to the root of the span.

In 3D it is more difficult to visualize the correction (β) but in this case the correction is largely localized near $\eta = 0.2$ as shown in Figures B.7 and B.8 with a small correction at $\eta = 0.95$ as shown in Figure B.9. Note that at $\eta = 0.2$ the correction is almost universally decreasing eddy viscosity production, which is consistent with the observations for the RAE2822 airfoil. The SA model tends to over-predict eddy viscosity in adverse pressure gradients, and the FI-Classic inversion shows that corrections are improved if the eddy viscosity is decreased in these regions.

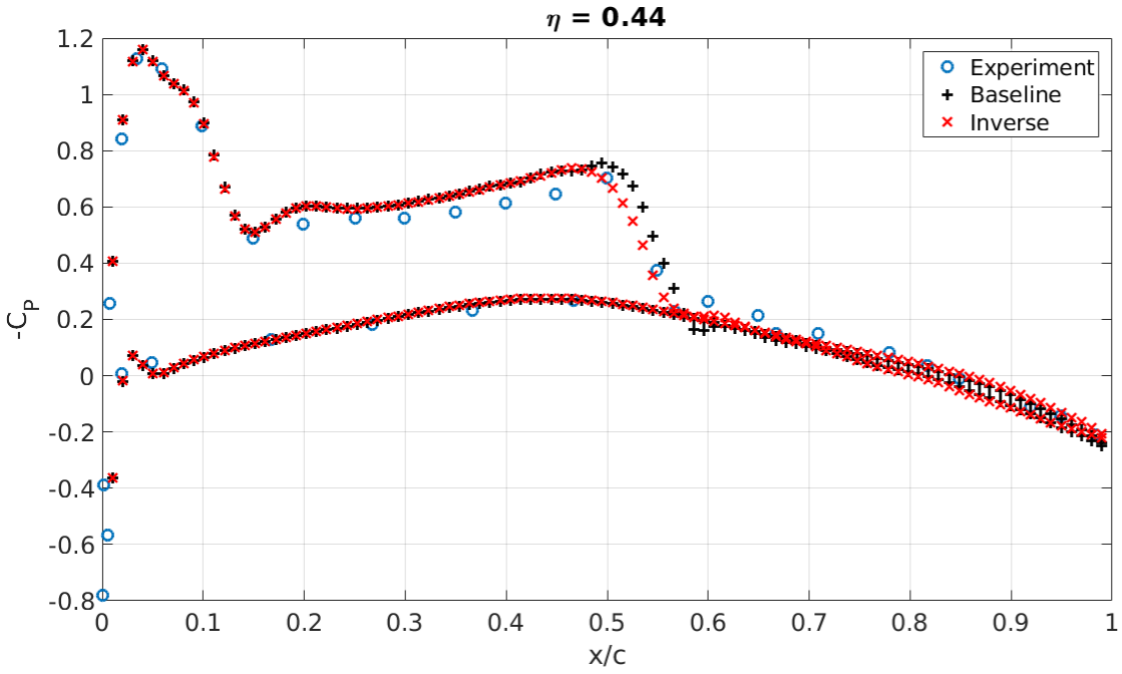

Figure B.4: Inverse results at span $\eta = 0.44$.

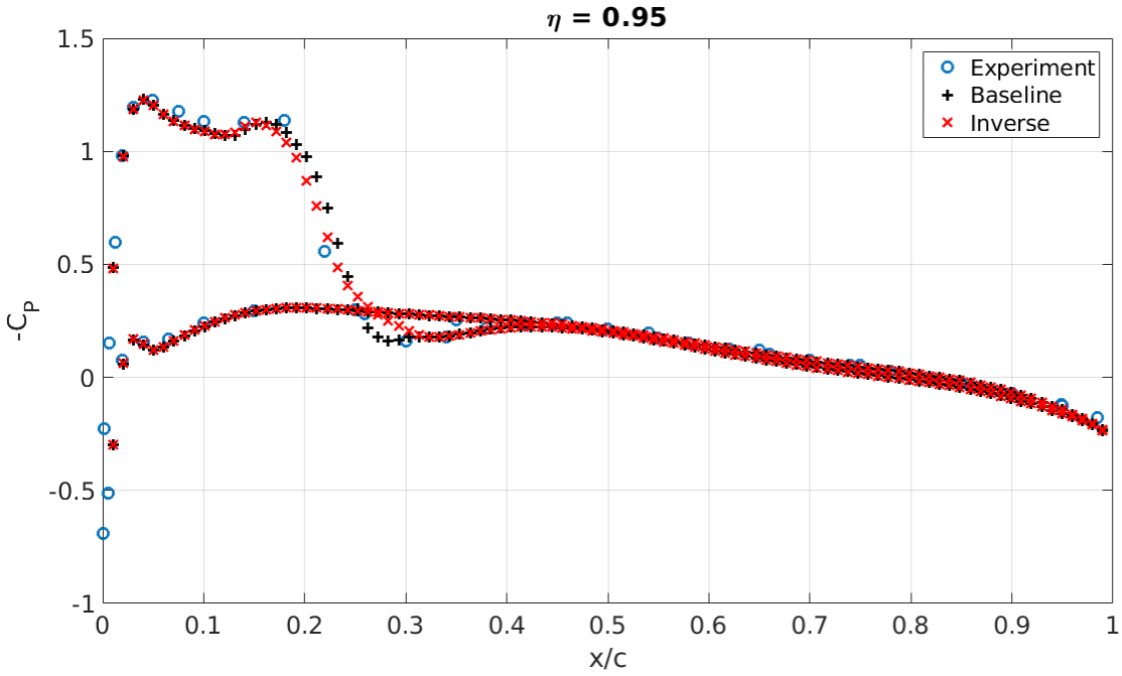

Figure B.5: Inverse results at span $\eta = 0.95$.

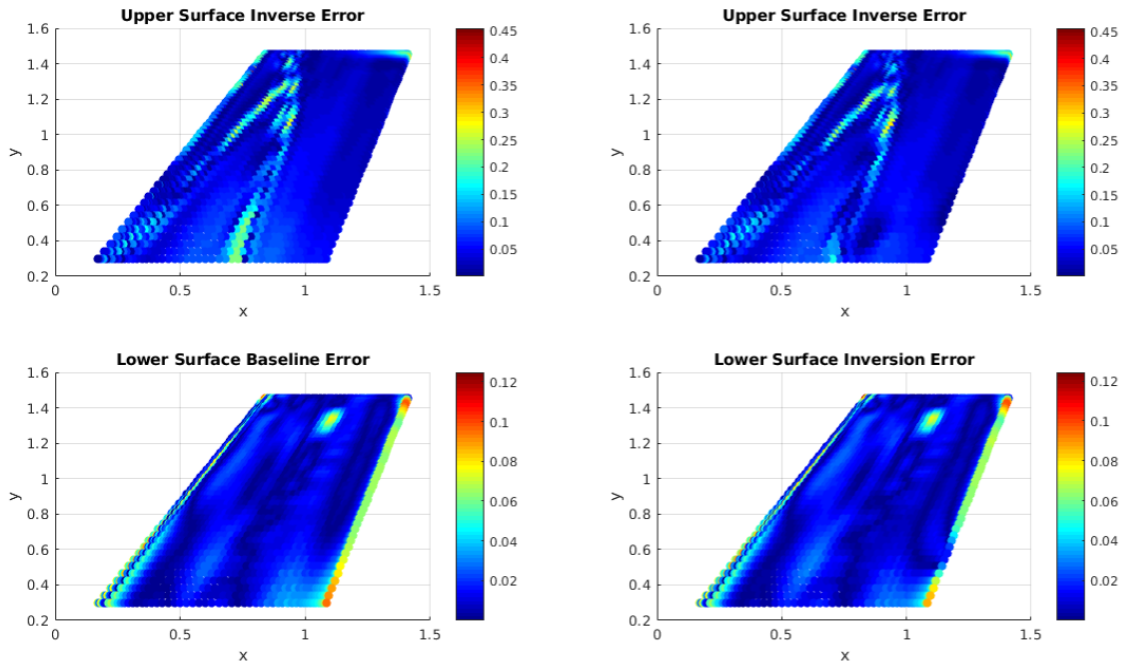

Figure B.6: Scatterplot colored by error in pressure coefficient ($|\widehat{C}_P - C_P|$).

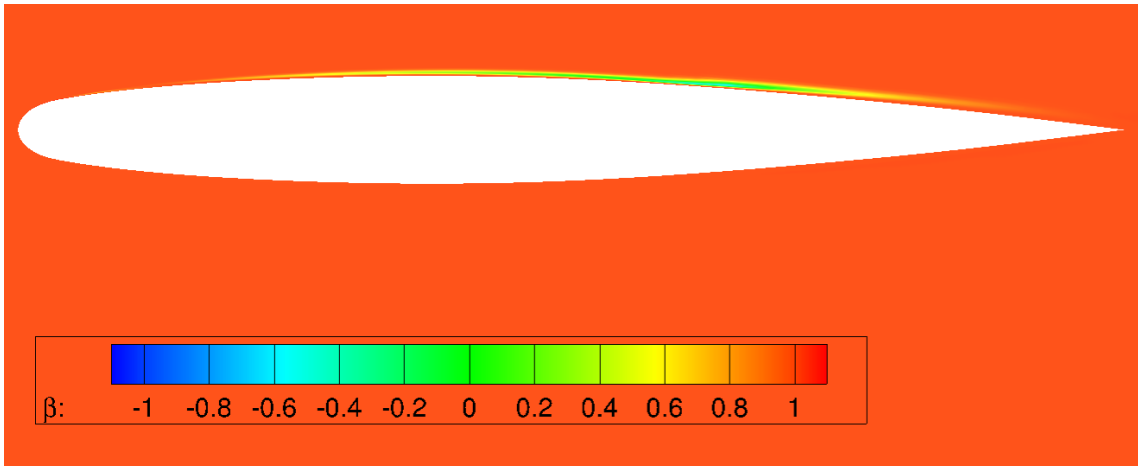

Figure B.7: Correction (β) at span $\eta = 0.2$.

B.3 Conclusions

FI-Classic results for the Onera M6 wing were presented in this appendix. These results demonstrate that the FIML approach can also be applied to 3D RANS

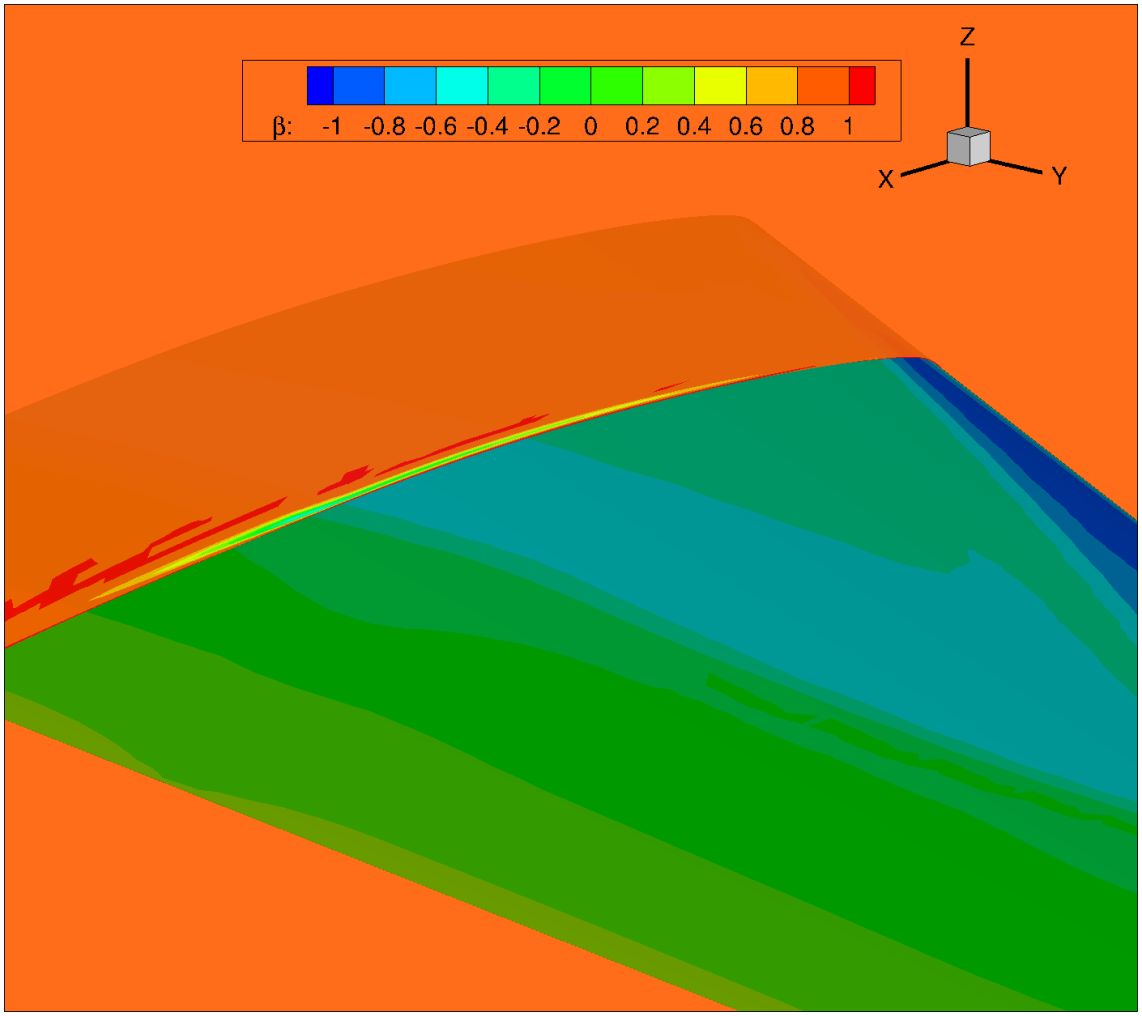

Figure B.8: Correction (β) at span $\eta = 0.2$, with wing surface contoured by C_p.

applications at a practical scale. Naturally the computational expense increases, however, this demonstrates proves that the added cost of considering 3D cases is not insurmountable. For this application, experimental pressure data were interpolated onto the surface of the wing at every node in the computational domain.

The inversion had good convergence, and the resulting error distribution showed that the primary increase in accuracy was towards the root of the wing. The location of the second shock was moved forward slightly, which improved the pressure

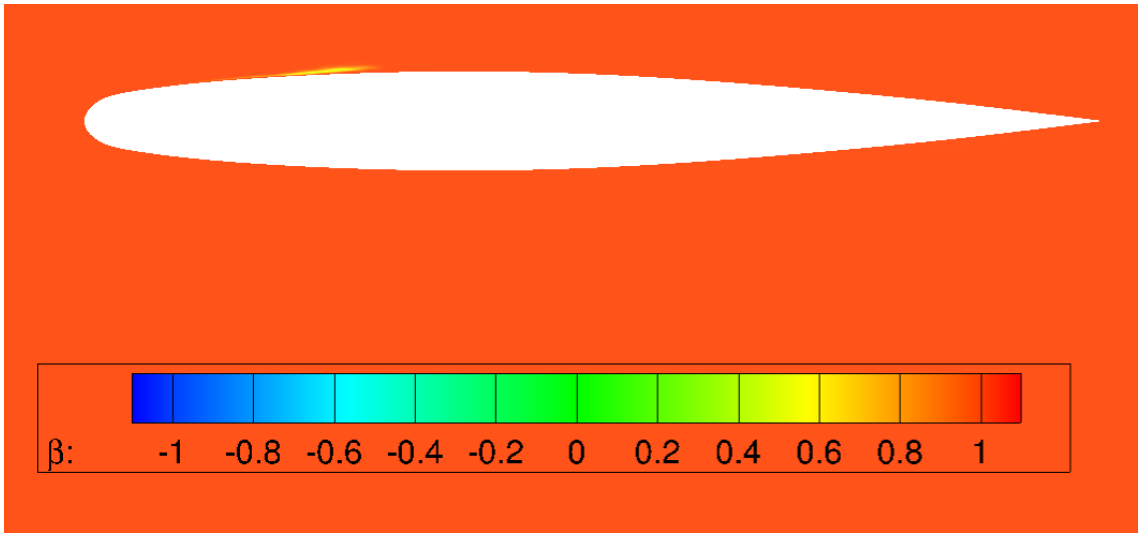

Figure B.9: Correction (β) at span $\eta = 0.95$.

distribution. When visualizing the correction, it was shown that the largest correction was in this region, and that the correction almost universally decreased eddy viscosity production. This result is consistent with the 2D RAE2822 airfoil result in that the eddy viscosity tends to be over-produced by the baseline SA model. This 3D application demonstrates that the FIML procedure can be applied to more complicated geometries and larger computational domains that will be required for practical engineering applications.

Bibliography

[1] Joseph A Schetz and Rodney DW Bowersox. *Boundary layer analysis*. American Institute of Aeronautics and Astronautics, 2011.

[2] Francis H Clauser. The turbulent boundary layer. In *Advances in applied mechanics*, volume 4, pages 1–51. Elsevier, 1956.

[3] Shivaji Medida. *Correlation-based Transition Modeling for External Aerodynamic Flows*. PhD thesis, University of Maryland, College Park, University of Maryland, College Park, MD, 20742, USA, 7 2014. URL `http://hdl.handle.net/1903/15150`.

[4] Holger Babinsky and John K Harvey. *Shock wave-boundary-layer interactions*, volume 32. Cambridge University Press, 2011.

[5] Ludwig Prandtl. Motions of fluids with very little viscosity. *National Advisory Committee for Aeronautics*, (452), 1928. URL `https://ntrs.nasa.gov/search.jsp?R=19930090813`.

[6] Osborne Reynolds. Iii. an experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel channels. *Proceedings of the Royal Society of London*, 35(224-226):84–99, 1883. doi: 10.1098/rspl.1883.0018. URL `http://rspl.royalsocietypublishing.org/content/35/224-226/84.abstract`.

[7] Bruno Eckhardt. Introduction. turbulence transition in pipe flow: 125th anniversary of the publication of reynolds' paper. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 367(1888):449–455, 2009. ISSN 1364-503X. doi: 10.1098/rsta.2008.0217. URL `http://rsta.royalsocietypublishing.org/content/367/1888/449`.

[8] Theodore von Kármán. Mechanical similitude and turbulence. *Nachrichten von der Gesellschaft der Wissenschaften zu Gottingen - Fachgruppe I (Mathematik)*, 5:58–76, 3 1930. URL `https://ntrs.nasa.gov/search.jsp?R=19930094805`.

[9] Donald Coles. The law of the wake in the turbulent boundary layer. *Journal of Fluid Mechanics*, 1(2):191226, 1956. doi: 10.1017/S0022112056000135.

[10] William K George. Is there a universal log law for turbulent wall-bounded flows? *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 365(1852):789–806, 2007. ISSN 1364-503X. doi: 10.1098/rsta.2006.1941. URL `http://rsta.royalsocietypublishing.org/content/365/1852/789`.

[11] John Kim, Parviz Moin, and Robert Moser. Turbulence statistics in fully developed channel flow at low reynolds number. *Journal of Fluid Mechanics*, 177:133166, 1987. doi: 10.1017/S0022112087000892.

[12] P.S. Bernard and J.M. Wallace. *Turbulent Flow: Analysis, Measurement, and Prediction*, chapter 4, pages 111–112. John Wiley & Sons, 2002. ISBN 9780471332190. URL `https://books.google.com/books?id=N44_L3SCN6EC`.

[13] Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA Journal*, 55(7):2215–2227, 2017. URL `https://doi.org/10.2514/1.J055595`.

[14] Alan Celic and Ernst H Hirschel. Comparison of eddy-viscosity turbulence models in flows with adverse pressure gradient. *Aiaa Journal*, 44(10):2156–2169, 2006.

[15] Aleksey A Matyushenko, Eugeniy V Kotov, and Andrey V Garbaruk. Calculations of flow around airfoils using two-dimensional rans: an analysis of the reduction in accuracy. *St. Petersburg Polytechnical University Journal: Physics and Mathematics*, 3(1):15–21, 2017.

[16] Dan M. Somers and James Tangler. Design and experimental results for the s809 airfoil. Technical report, National Renewable Energy Laboratory, 01 1997. URL `https://www.nrel.gov/docs/legosti/old/6918.pdf`.

[17] P. R. Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th Aerospace Sciences Meeting and Exhibit*, 1992. URL `https://doi.org/10.2514/6.1992-439`.

[18] Samet Caka Cakmakcioglu, Onur Bas, and Unver Kaynak. A correlation-based algebraic transition model. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 232(21):3915–3929, 2018. doi: 10.1177/0954406217743537. URL `https://doi.org/10.1177/0954406217743537`.

[19] PR Spalart, WH Jou, M Strelets, and SR Allmaras. Comments on the feasibility of les for wings, and on a hybrid rans/les approach. 1st afosr int. In *Symp. Eng. Turb. Modelling and Measurements, May*, pages 24–26, 1997.

[20] Philippe R Spalart. Detached-eddy simulation. *Annual review of fluid mechanics*, 41:181–202, 2009.

[21] Michael S Holden, Timothy P Wadhams, John K Harvey, and Graham V Candler. Comparisons between measurements in regions of laminar shock wave boundary layer interaction in hypersonic flows with navier-stokes and dsmc solutions. Technical report, AEROTHERMAL AND AERO-OPTICS EVALUATION CENTER BUFFALO NEW YORK, 2006.

[22] Michael S Holden, Tim P Wadhams, and Matthew G MacLean. Measurements in regions of shock wave/turbulent boundary layer interaction from mach 4 to 10 for open and blind code evaluation/validation. In *21st AIAA Computational Fluid Dynamics Conference*, page 2836, 2013.

[23] Michael Holden. Measurements in regions of shock wave/turbulent boundary layer interaction from mach 4 to 7 at flight duplicated velocities to evaluate and improve the models of turbulence in cfd codes. In *2018 Fluid Dynamics Conference*, page 3706, 2018.

[24] PH Cook, MCP Firmin, and MA McDonald. *Aerofoil RAE 2822: pressure distributions, and boundary layer and wake measurements*. RAE, 1977.

[25] Christopher L Rumsey. Recent developments on the turbulence modeling resource website. In *22nd AIAA Computational Fluid Dynamics Conference*, page 2927, 2015. URL `https://turbmodels.larc.nasa.gov/`.

[26] Joseph G Marvin, James L Brown, and Peter A Gnoffo. Experimental database with baseline cfd solutions: 2-d and axisymmetric hypersonic shock-wave/turbulent-boundary-layer interactions. 2013. URL `https://turbmodels.larc.nasa.gov/Other_exp_Data/SBLI_various_marvin/NASATM_2013_216604.pdf`.

[27] Francis K Acquaye. Evaluation of various turbulence models for shock-wave boundary layer interaction flows. Master's thesis, Washington University in St. Louis, 2016. URL `https://doi.org/10.7936/K7RJ4GS6`.

[28] Sergio Pirozzoli, Alexandre Beer, Matteo Bernardini, and Francesco Grasso. Computational analysis of impinging shock-wave boundary layer interaction under conditions of incipient separation. *Shock Waves*, 19(6):487, 2009.

[29] Dennis C Jespersen, Thomas H Pulliam, and Marissa Lynn Childs. Overflow turbulence modeling resource validation results. 2016. URL `https://ntrs.nasa.gov/search.jsp?R=20190000252`.

[30] Brandon Morgan, K Duraisamy, N Nguyen, S Kawai, and SK Lele. Flow physics and rans modelling of oblique shock/turbulent boundary layer interaction. *Journal of Fluid Mechanics*, 729:231–284, 2013.

[31] Pietro Catalano and Marcello Amato. An evaluation of rans turbulence modelling for aerodynamic applications. *Aerospace science and Technology*, 7(7): 493–509, 2003.

[32] Keerti K Bhamidipati, Daniel A Reasor, and Crystal L Pasiliao. Unstructured grid simulations of transonic shockwave-boundary layer interaction-induced oscillations. In *22nd AIAA Computational Fluid Dynamics Conference*, page 2287, 2015.

[33] Gérard Dreyfus. *Neural Networks: Methodology and Applications*. Springer-Verlag, Berlin Heidelberg, 1 edition, 2005.

[34] Kurt Hornik, Maxwell Stinchcombe, and Halber White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:451460, 3 1989.

[35] Jamshid A. Samareh and Jay Ming Wong. Training knowledge bots for physics-based simulations using artificial neural networks. Technical report, NASA, 11 2014. URL `https://ntrs.nasa.gov/search.jsp?R=20150000596`.

[36] Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. A physics informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Physical Review Fluids*, 2(3), 2017.

[37] Jin-Long Wu, Jian-Xun Wang, Heng Xiao, and Julia Ling. A priori assessment of prediction confidence for data-driven turbulence modeling. *Flow, Turbulence and Combustion*, 99(1):25–46, 2017. URL `https://doi.org/10.1007/s10494-017-9807-0`.

[38] J. Ling and J. Templeton. Evaluation of machine learning algorithms for prediction of regions of high reynolds averaged navier stokes uncertainty. *Physics of Fluids*, 27(8):085103, 2015. doi: 10.1063/1.4927765. URL `https://aip.scitation.org/doi/abs/10.1063/1.4927765`.

[39] Eric J. Parish and Karthik Duraisamy. A paradigm for data-driven predictive modeling using field inversion and machine learning. *Journal of Computational Physics*, 305:758 – 774, 2016. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2015.11.012. URL `http://www.sciencedirect.com/science/article/pii/S0021999115007524`.

[40] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, 51:357–377, 2019.

[41] Anand Pratap Singh and Karthik Duraisamy. Using field inversion to quantify functional errors in turbulence closures. *Physics of Fluids*, 28(4):045110, 2016. doi: 10.1063/1.4947045. URL `https://doi.org/10.1063/1.4947045`.

[42] Anand Pratap Singh. *A Framework to Improve Turbulence Models Using Full-field Inversion and Machine Learning*. PhD thesis, University of Michigan, University of Michigan, Ann Arbor, MI, 48109, USA, 6 2018. URL `http://hdl.handle.net/2027.42/144034`.

[43] Karthik Duraisamy, Ze Jia XZhang, and Anand Pratap Singh. New approaches in turbulence and transition modeling using data-driven techniques. In *53rd AIAA Aerospace Sciences Meeting, AIAA SciTech Forum*, 2015. URL `https://doi.org/10.2514/6.2015-1284`.

[44] Hiroshi Kato, Keiichi Ishiko, and Akira Yoshizawa. Optimization of parameter values in the turbulence model aided by data assimilation. *AIAA Journal*, pages 1512–1523, 2016.

[45] Rosario Lanzafame, Stefano Mauro, and Michele Messina. 2d cfd modeling of h-darrieus wind turbines using a transition turbulence model. *Energy Procedia*, 45:131–140, 2014.

[46] S Mauro, R Lanzafame, M Messina, and D Pirrello. Transition turbulence model calibration for wind turbine airfoil characterization through the use of a micro-genetic algorithm. *International Journal of Energy and Environmental Engineering*, 8(4):359–374, 2017.

[47] PA Costa Rocha, HH Barbosa Rocha, FO Moura Carneiro, ME Vieira da Silva, and A Valente Bueno. k–ω sst (shear stress transport) turbulence model calibration: A case study on a small scale horizontal axis wind turbine. *Energy*, 65:412–418, 2014.

[48] PA Costa Rocha, HH Barbosa Rocha, FO Moura Carneiro, ME Vieira da Silva, and C Freitas de Andrade. A case study on the calibration of the k–ω sst (shear stress transport) turbulence model for small scale wind turbines designed with cambered and symmetrical airfoils. *Energy*, 97:144–150, 2016.

[49] John Schaefer, Serhat Hosder, Thomas West, Christopher Rumsey, Jan-Renee Carlson, and William Kleb. Uncertainty quantification of turbulence model closure coefficients for transonic wall-bounded flows. *AIAA Journal*, 55(1): 195–213, 2016.

[50] John Schaefer, Andrew Cary, Mori Mani, and Philippe Splart. Uncertainty quantification and sensitivity analysis of sa turbulence model coefficients in two and three dimensions. In *55th AIAA Aerospace Sciences Meeting, AIAA SciTech Forum*, 2017. URL `https://doi.org/10.2514/6.2017-1710`.

[51] John Schaefer, Serhat Hosder, Thomas West, Christopher Rumsey, Jan-Renee Carlson, and William Kleb. Uncertainty quantificaiton of turbulence model closure coefficients for transonic wall-bounded flows. *AIAA Journal*, 55(1): 195–213, 2017. URL `https://doi.org/10.2514/1.J054902`.

[52] John Anthony Schaefer. Uncertainty quantification of turbulence model closure coefficients for transonic wall-bounded flows. Master's thesis, Missouri University of Science and Technology, Missouri S&T, Rolla, MO 65409, USA, 2014. URL `http://scholarsmine.mst.edu/masters_theses/7480/`.

[53] Jaideep Ray, Sophia Lefantzi, Srinivasan Arunajatesan, and Lawrence J DeChant. Bayesian calibration of a k-ε turbulence model for predictive jet-in-crossflow simulations. In *44th AIAA Fluid Dynamics Conference*, page 2085, 2014.

[54] Serge Guillas, Nina Glover, and Liora Malki-Epshtein. Bayesian calibration of the constants of the k–ε turbulence model for a cfd model of street canyon flow. *Computer methods in applied mechanics and engineering*, 279:536–553, 2014.

[55] Todd A Oliver and Robert D Moser. Bayesian uncertainty quantification applied to rans turbulence models. In *Journal of Physics: Conference Series*, volume 318, page 042032. IOP Publishing, 2011.

[56] Dimitrios I Papadimitriou and Costas Papadimitriou. Bayesian uncertainty quantification of turbulence models based on high-order adjoint. *Computers & Fluids*, 120:82–97, 2015.

[57] WN Edeling, Pasquale Cinnella, and Richard P Dwight. Predictive rans simulations via bayesian model-scenario averaging. *Journal of Computational Physics*, 275:65–91, 2014.

[58] Hassan Raiesi, Ugo Piomelli, and Andrew Pollard. Evaluation of turbulence models using direct numerical and large-eddy simulation data. *Journal of Fluids Engineering*, 133(2):021203, 2011.

[59] Paul A Durbin. Some recent developments in turbulence closure modeling. *Annual Review of Fluid Mechanics*, 50:77–103, 2018.

[60] Michael Emory, Johan Larsson, and Gianluca Iaccarino. Modeling of structural uncertainties in reynolds-averaged navier-stokes closures. *Physics of Fluids*, 25(11):110822, 2013.

[61] Michael Emory, Rene Pecnik, and Gianluca Iaccarino. Modeling structural uncertainties in reynolds-averaged computations of shock/boundary layer interactions. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 479, 2011.

[62] Aashwin Mishra, J Mukhopadhaya, Gianluca Iaccarino, and Juan Alonso. Uncertainty quantification of turbulence models for complex aerospace flows, 12 2018.

[63] Aashwin Ananda Mishra, Jayant Mukhopadhaya, Gianluca Iaccarino, and Juan Alonso. Uncertainty estimation module for turbulence model predictions in su2. *AIAA Journal*, 57(3):1066–1077, 2018.

[64] Heng Xiao, J-L Wu, J-X Wang, Rui Sun, and CJ Roy. Quantifying and reducing model-form uncertainties in reynolds-averaged navier–stokes simulations: A data-driven, physics-informed bayesian approach. *Journal of Computational Physics*, 324:115–136, 2016.

[65] Jin-Long Wu, Jian-Xun Wang, and Heng Xiao. A bayesian calibration–prediction method for reducing model-form uncertainties with application in rans simulations. *Flow, Turbulence and Combustion*, 97(3):761–786, 2016.

[66] Heng Xiao and Paola Cinnella. Quantification of model uncertainty in rans simulations: a review. *Progress in Aerospace Sciences*, 2019.

[67] WN Edeling, G Iaccarino, and P Cinnella. Data-free and data-driven rans predictions with quantified uncertainty. *Flow, Turbulence and Combustion*, 100(3):593–616, 2018.

[68] Q Wang and EA Dow. Quantification of structural uncertainties in the k- ω turbulence model. In *Proceedings of the Summer Program*, page 41, 2010.

[69] Brendan D Tracey, Karthikeyan Duraisamy, and Juan J Alonso. A machine learning strategy to assist turbulence model development. In *53rd AIAA aerospace sciences meeting*, page 1287, 2015.

[70] Anand Pratap Singh, Shaowu Pan, and Karthikeyan Duraisamy. Characterizing and improving predictive accuracy in shock-turbulent boundary layer interactions using data-driven models. In *55th AIAA Aerospace Sciences Meeting, AIAA SciTech Forum*, 2017. URL `https://doi.org/10.2514/6.2017-0314`.

[71] Anand Pratap Singh, Karthikeyan Duraisamy, and Ze Jia Zhang. Augmentation of turbulence models using field inversion and machine learning. In *55th AIAA Aerospace Sciences Meeting*, page 0993, 2017.

[72] Anand Pratap Singh, Racheet Matai, Asitav Mishra, Karthikeyan Duraisamy, and Paul A Durbin. Data-driven augmentation of turbulence models for adverse pressure gradient flows. In *23rd AIAA Computational Fluid Dynamics Conference*, page 3626, 2017.

[73] Jonathan R Holland, James D Baeder, and Karthik Duraisamy. Towards integrated field inversion and machine learning with embedded neural networks for rans modeling. In *AIAA Scitech 2019 Forum*, page 1884, 2019.

[74] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. doi: 10.1038/323533a0. URL `http://dx.doi.org/10.1038/323533a0`.

[75] Michael JD Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7 (2):155–162, 1964.

[76] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[77] John David Anderson and J Wendt. *Computational fluid dynamics*, volume 206. Springer, 1995.

[78] Jiri Blazek. *Computational fluid dynamics: principles and applications*. Butterworth-Heinemann, 2015.

[79] W Sutherland LII. The viscosity of gases and molecular force. *Lond. Edinburgh Dublin Philos. Mag. J. Sci.*, 36(223):507–531, 1893.

[80] Peter Davidson. *Turbulence: an introduction for scientists and engineers*. Oxford University Press, 2015.

[81] Robin B Langtry and Florian R Menter. Correlation-based transition modeling for unstructured parallelized computational fluid dynamics codes. *AIAA journal*, 47(12):2894–2906, 2009.

[82] FR Menter, R Langtry, and S Völker. Transition modelling for general purpose cfd codes. *Flow, turbulence and combustion*, 77(1-4):277–303, 2006.

[83] T Rung, U Bunge, M Schatz, and F Thiele. Restatement of the spalart-allmaras eddy-viscosity model in strain-adaptive formulation. *AIAA journal*, 41(7):1396–1399, 2003.

[84] Russell Reed and Robert J Marks. *Neural smithing: supervised learning in feedforward artificial neural networks*. 1999.

[85] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[86] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[87] Josef Stoer and Roland Bulirsch. *Introduction to numerical analysis*, volume 12. Springer Science & Business Media, 2013.

[88] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[89] Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

[90] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[91] D Hunter John. Matplotlib: a 2d graphics environment comput. *Sci. Eng*, 9: 90–5, 2007.

[92] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[93] Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[94] DI Papadimitriou and KC Giannakoglou. Direct, adjoint and mixed approaches for the computation of hessian in airfoil design problems. *International journal for numerical methods in fluids*, 56(10):1929–1943, 2008.

[95] Devendra Ghate and Michael Giles. Efficient hessian calculation using automatic differentiation. In *25th AIAA Applied Aerodynamics Conference*, page 4059, 2007.

[96] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`.

[97] William Squire and George Trapp. Using complex variables to estimate derivatives of real functions. *SIAM review*, 40(1):110–112, 1998. URL `https://doi.org/10.1137/S003614459631241X`.

[98] James C Newman, W Kyle Anderson, and David L Whitfield. Multidisciplinary sensitivity derivatives using complex variables. *Mississippi State University Publication, MSSU-EIRS-ERC-98-08*, 1998.

[99] Robert T Biedron, Jan-Reneé Carlson, Joseph M Derlaga, Peter A Gnoffo, Dana P Hammond, William T Jones, Bill Kleb, Elizabeth M Lee-Rausch, Eric J Nielsen, Michael A Park, et al. Fun3d manual: 13.4. 2018.

[100] Martin S Ridout. Statistical applications of the complex-step method of numerical differentiation. *The American Statistician*, 63(1):66–74, 2009.

[101] Ravi Kiran and Kapil Khandelwal. Complex step derivative approximation for numerical evaluation of tangent moduli. *Computers & Structures*, 140:1–13, 2014.

[102] Steven G. Johnson. Notes on adjoint methods for 18.336. Online Posting, 12 2012. URL `https://math.mit.edu/~stevenj/18.336/adjoint.pdf`.

[103] Michael B Giles and Niles A Pierce. An introduction to the adjoint approach to design. *Flow, turbulence and combustion*, 65(3-4):393–415, 2000.

[104] Tim A Albring, Max Sagebaum, and Nicolas R Gauger. Efficient aerodynamic design using the discrete adjoint method in su2. In *17th AIAA/ISSMO multidisciplinary analysis and optimization conference*, page 3518, 2016.

[105] Beckett Yx Zhou, Tim A Albring, Nicolas R Gauger, Thomas D Economon, and Juan J Alonso. An efficient unsteady aerodynamic and aeroacoustic design framework using discrete adjoint. In *17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 3369, 2016.

[106] Thomas D Economon, Juan J Alonso, Tim A Albring, and Nicolas R Gauger. Adjoint formulation investigations of benchmark aerodynamic design cases in su2. In *35th AIAA Applied Aerodynamics Conference*, page 4363, 2017.

[107] Thomas D. Economon, Francisco Palacios, Sean R. Copeland, Trent W. Lukaczyk, and Juan Alonso. Su2: An open-source suite for multiphysics simulation and design. *AIAA Journal*, 54(3):828–846, 2016. URL `https://doi.org/10.2514/1.J053813`.

[108] Beckett Yx Zhou, Tim A Albring, Nicolas R Gauger, Thomas D Economon, Francisco Palacios, and Juan J Alonso. A discrete adjoint framework for unsteady aerodynamic and aeroacoustic optimization. In *16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 3355, 2015.

[109] Beckett Yx Zhou, Tim Albring, Nicolas R Gauger, Carlos R Ilario da Silva, Thomas D Economon, and Juan J Alonso. A discrete adjoint approach for jet-flap interaction noise reduction. In *58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, page 0130, 2017.

[110] M. Sagebaum, T. Albring, and N. R. Gauger. High-performance derivative computations using codipack. *arXiv preprint arXiv:1709.07229*, 2017. URL `https://arxiv.org/abs/1709.07229`.

[111] Christopher L Rumsey. Proceedings of the 2004 workshop on cfd validation of synthetic jets and turbulent separation control. 2007.

[112] R Bush, G Power, and C Towne. Wind-the production flow solver of the nparc alliance. In *36th AIAA Aerospace Sciences Meeting and Exhibit*, page 935, 1998. URL `https://www.grc.nasa.gov/WWW/wind/valid/raetaf/raetaf05/raetaf05.html`.

[113] Philippe R Spalart, Shur Deck, Michael L Shur, Kyle D Squires, M Kh Strelets, and Andrei Travin. A new version of detached-eddy simulation, resistant to ambiguous grid densities. *Theoretical and computational fluid dynamics*, 20 (3):181, 2006.

[114] Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508, 2001.

[115] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2018.

[116] Michael Holden, Timothy Wadhams, Matthew MacLean, and Erik Mundy. Experimental studies of shock wave/turbulent boundary layer interaction in high reynolds number supersonic and hypersonic flows to evaluate the performance of cfd codes. In *40th Fluid Dynamics Conference and Exhibit*, page 4468, 2010.

[117] Michael J Wright, Todd White, and Nancy Mangini. Data parallel line relaxation (dplr) code user manual: Acadia-version 4.01. 1. 2009.

[118] V Schmitt and F Charpin. Pressure distributions on the onera-m6-wing at transonic mach numbers. *Experimental data base for computer program assessment*, 4, 1979.

[119] Alexander Kuzmin. On the lambda-shock formation on onera m6 wing. *International Journal of Applied Engineering Research*, 9(20):7029–7038, 2014.

[120] Daniel Destarac and Antoine Dumont. Onera m6 wing test-case, original and tmr. 2016. URL `https://turbmodels.larc.nasa.gov/Onerawingnumerics_val/ONERA_M6_Test_Case_TMR.pdf`.