Metric-based, goal-oriented mesh adaptation using machine learning

Krzysztof J. Fidkowski*, Guodong Chen

Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109

Abstract

This paper presents a machine-learning approach for determining the optimal anisotropy in a computational mesh, in the context of an output-based adaptive solution procedure. Artificial neural networks are used to predict the desired element aspect ratio from readily accessible features of the primal and adjoint solutions. Whereas the sizing of the element is still based on an adjoint-weighted residual error estimate, the network augments this information with element stretching magnitude and direction, at lower computational and implementation costs compared to a more rigorous approach: mesh optimization through error sampling and synthesis (MOESS). The network is trained to provide anisotropy information in the form of a normalized metric field, computed from primal, adjoint, and error indicator features. MOESS-optimized meshes for a variety of steady aerodynamic flows governed by the Reynolds-averaged Navier-Stokes equations in two dimensions provide data for training several multi-layer perceptron networks, which differ in size and inputs. The networks are then deployed and tested by driving complete mesh adaptation sequences, and the results show improvements in mesh efficiency compared to pure primal Hessian-based anisotropy detection and in many cases to MOESS itself.

Keywords: Output Error Estimation, Neural Network, Mesh Optimization, Metric, Adjoint, Discontinuous Galerkin

1. Introduction

Growth in computational resources and advancements in numerical methods have made possible simulations of complex physical phenomena on large, intricate domains. The foundation for most such simulations is the computational mesh, which affects both the accuracy of the solutions and the efficiency of the computations. The construction of an optimal mesh, one that maximizes accuracy for a given cost, is not trivial and has been the subject of much previous research. Most of this research has focused on adaptive methods, in which the construction of the mesh proceeds iteratively, concurrently with the solver [1]. In this work, we tackle one aspect of adaptive mesh generation, determination of anisotropy, with the intent of simplifying the process, in the context of computational fluid dynamics simulations.

Mesh anisotropy refers to direction-dependent sizing of the elements that constitute the mesh. Anisotropic meshes are important for efficiently resolving certain flow features, such as boundary layers, wakes, and shocks, that appear in computational fluid dynamics. However, determining the optimal mesh anisotropy, i.e. one that produces the most cost-effective meshes for a chosen error measure, is not a trivial task. Heuristics based on flow features, such as the Hessian of a scalar quantity [2, 3, 4, 5, 6, 7, 8, 9, 10, 11], can perform well for many cases but are not generally optimal for a wide range of flow fields and approximation orders. They are also difficult to extend to systems of equations, in which multiple quantities are approximated.

A more rigorous approach for determining mesh anisotropy and sizing involves sampling the effects of element subdivision and regressing a model for the error behavior [12, 13]. Such an approach directly addresses the error/cost optimization problem in the presence of unknown local convergence rates but is computationally more complex and requires non-trivial mesh operations for its implementation.

*Corresponding author
Email address: kfid@umich.edu (Krzysztof J. Fidkowski)

In this work, we investigate the idea of using a machine-learning approach to predict the optimal mesh anisotropy. The goal is to design an algorithm that produces results similar to MOESS, at lower online implementation and computational costs, on par with feature-based detection methods. Machine learning techniques have the potential to accurately and efficiently model responses of highly-nonlinear problems over a wide range of parameters. A machine-learning algorithm must be trained on a large amount of data, and for this we use element-specific results of many MOESS iterations. The solution features used as inputs to the machine-learning algorithm are based on both the primal and the adjoint solution, motivated by the fact that this is the same information that is used to compute the error indicator.

For the machine-learning method, we use an artificial neural network (ANN). ANNs are a popular scientific machine learning approach for modeling nonlinear mappings between vector inputs and outputs [14, 15, 16, 17]. They emulate biological systems through connected layers of neurons that are activated under sufficiently-strong input signals. They have been used for many scientific and engineering purposes, primarily for interpolation and data-driven modeling, both of which involve mapping inputs to desired outputs [18, 19, 20, 21, 22, 23, 24, 25, 26]. Such a map does not contain physics of the problem and is therefore often viewed as "black box." Nevertheless, the map can effectively "learn" relationships between inputs and outputs that may be difficult to discern mathematically from first principles.

The outline for the remainder of this paper is as follows. Section 2 presents the numerical approach used in this work, a discontinuous finite-element discretization. Section 3 summarizes output-based error estimation, which is used to drive the adaptation algorithms presented in Section 4. Section 5 introduces the new machine-learning approach for identifying the correct mesh anisotropy during adaptation. Finally, Sections 6 and 7 present results of the training and deployment of the neural network, and Section 8 concludes with a summary and a discussion of future directions.

2. Discretization

We present the adaptive approach in the context of a discontinuous Galerkin (DG) finite element discretization [27, 28, 29]. We consider a system of steady partial differential equations in conservative form,

$$\nabla \cdot \vec{\mathbf{F}}(\mathbf{u}, \nabla\mathbf{u}) + \mathbf{S}(\mathbf{u}, \nabla\mathbf{u}) \quad = \quad \mathbf{0}, \tag{1}$$

where $\mathbf{u} \in \mathbb{R}^s$ is the s-component state vector, $\vec{\mathbf{F}} \in \mathbb{R}^{d\times s}$ is the flux vector, d is the spatial dimension, and $\mathbf{S}$ is a source term associated with turbulence modeling closure equations, in this work RANS-SA [30, 31].

We assume a subdivision, T_h, of the computational domain, Ω, into N_e non-overlapping elements, Ω_e, and on each element we approximate the state by an order p polynomial. Specifically, the state approximation is $\mathbf{u}_h^p \in \boldsymbol{\mathcal{V}}_h^p = [\mathcal{V}_h^p]^s$, where

$$\mathcal{V}_h^p \equiv \{u \in L_2(\Omega) : u|_{\Omega_e} \in \mathcal{P}^p \ \forall \Omega_e \in T_h\}. \tag{2}$$

The subscript h indicates a specific domain subdivision, i.e. the collective size/shape distribution of all of the elements, and $\mathcal{P}^p$ is the set of order-p polynomials [1].

Multiplying (1) by test functions $\mathbf{v}_h^p \in \boldsymbol{\mathcal{V}}_h^p$, integrating by parts on each element, and using the Roe [32] convective flux and the second form of Bassi and Rebay (BR2) [33] for the viscous treatment, we obtain the following semilinear weak form:

$$\mathcal{R}_h^p(\mathbf{u}_h^p, \mathbf{v}_h^p) \quad = \quad 0. \tag{3}$$

Choosing bases for the trial and test spaces results in a system of nonlinear equations that is solved using a Newton-Raphson procedure. The sparse Jacobian matrix is fully stored, and the linear solves are preformed using GMRES, preconditioned by an element-line iterative solver [29].

[1] The polynomials are defined in reference space and for curved elements they may not remain order p polynomials after the mapping to physical space.

3. Output Error Estimation

We use an adjoint-based output error estimate to drive the mesh optimization. Theoretical details can be found in previous works [34, 1, 35]. For a scalar output, $\mathcal{J}(\mathbf{u}_h^p)$, the discrete adjoint field $\boldsymbol{\psi}_h^p \in \boldsymbol{\mathcal{V}}_h^p$ is the linearized sensitivity of $\mathcal{J}$ to residual source perturbations, $\delta\mathcal{R}(\cdot) : \boldsymbol{\mathcal{V}}_h^p \to \mathbb{R}$ added to the left-hand side of (3),

$$\delta\mathcal{J} = \delta\mathcal{R}(\boldsymbol{\psi}_h^p), \tag{4}$$

where the equality assumes infinitesimal perturbations. We obtain the adjoint field by solving the following linear equation,

$$\mathcal{R}_h'^{\,p}[\mathbf{u}_h^p](\mathbf{v}_h^p, \boldsymbol{\psi}_h^p) + \mathcal{J}'[\mathbf{u}_h^p](\mathbf{v}_h^p) \quad = \quad 0, \quad \forall \mathbf{v}_h^p \in \boldsymbol{\mathcal{V}}_h^p, \tag{5}$$

where the prime denotes Fréchét linearization with respect to the argument in square brackets.

Output error estimation relies on a finer discretization space, which in this work is $\boldsymbol{\mathcal{V}}_h^{p+1}$: order $p+1$ approximation on elements of the same domain subdivision, T_h. The original (coarse) primal state $\mathbf{u}_h^p$ does not generally satisfy the fine-space weak form; instead it satisfies a perturbed weak form, with a residual source of $\delta\mathcal{R}(\cdot) = -\mathcal{R}_h^{p+1}(\mathbf{u}_h^p, \cdot)$. If we have a fine-space adjoint, $\boldsymbol{\psi}_h^{p+1}$, we can then estimate the error in $\mathcal{J}$ due to using the coarse primal instead of the (never calculated) fine-space primal,

$$\text{output error } = \delta\mathcal{J} \equiv \mathcal{J}(\mathbf{u}_h^p) - \mathcal{J}(\mathbf{u}_h^{p+1}) \approx -\mathcal{R}_h^{p+1}(\mathbf{u}_h^p, \boldsymbol{\psi}_h^{p+1}). \tag{6}$$

The approximation sign indicates that the adjoint-weighted residual (last term) is an *estimate* of the error between the spaces when the equations or output are not linear, since we are using linear sensitivities with non-infinitesimal perturbations. The calculation in (6) is also only an estimate of the true numerical error, i.e. compared to the exact solution, since it uses an approximate, finite-dimensional, fine space. The richer the fine space, the better the error estimate. We rely on the fact that both the primal and the adjoint solutions are used to compute the error estimate when selecting features for the machine-learning approach presented in Section 5.

4. Adaptation

We note that (6) can be localized to elemental contributions,

$$\delta\mathcal{J} \approx \sum_{e=1}^{N_e} -\mathcal{R}_h^{p+1}(\mathbf{u}_h^p, \boldsymbol{\psi}_h^{p+1}\big|_{\Omega_e}), \qquad \mathcal{E}_e \equiv |\mathcal{R}_h^{p+1}(\mathbf{u}_h^p, \boldsymbol{\psi}_h^{p+1}\big|_{\Omega_e})|. \tag{7}$$

$\mathcal{E}_e \geq 0$ in the above equation is the error indicator for element e. During mesh adaptation, this indicator provides information to adapt the computational mesh. However, as just one scalar quantity per element, the error indicator is not sufficient to provide information about mesh anisotropy. This information can come from heuristics, or from a sampling approach, as described below.

4.1. Hessian-Based Anisotropy Detection

One dominant approach for detecting the anisotropy is to estimate the directional interpolation error of the solution [36, 2], and we describe here an extension of such an approach that incorporates output error adaptive indicators [6, 10]. For linear approximation, i.e., $p = 1$, the interpolation error of a scalar solution u over an edge E in the mesh, with unit tangent vector $\vec{s}$ and length h, is given by

$$\delta_{u,E} \propto |\vec{s}^T \mathbf{H} \vec{s}| h^2, \tag{8}$$

where $\mathbf{H}$ is the solution Hessian matrix,

$$H_{i,j} = \frac{\partial u^2}{\partial x_i \partial x_j}, \quad i, j \in [1, ..., d], \tag{9}$$

and d is the spatial dimension. The second derivatives can be estimated by a quadratic reconstruction of the linear solution. The scalar u used in this work is the Mach number as it has been found to be effective for many types of flows, although more sophisticated quantities can also be used.

A geometric interpretation of (8) is that the interpolation error over an edge depends on the squared edge length under the measure of $|\mathbf{H}|$, which is a symmetric positive definite (SPD) matrix defined by taking the absolute values of its eigenvalues $\mathbf{\Lambda}$ while keeping the eigenvectors $\boldsymbol{Q}$ the same, $|\mathbf{H}| = \mathbf{Q}^T|\mathbf{\Lambda}|\mathbf{Q}$. For higher-order approximations, the interpolation error is characterized by the $p+1^{\text{st}}$ derivatives, and the first d largest directional derivatives can be used to determine the principal directions $\mathbf{Q}$ and the corresponding stretching $\mathbf{\Lambda}$ [10]. In this work, however, we only consider using the Hessian matrix of second derivatives, regardless of the solution approximation order.

The principle of error equidistribution dictates that an optimal mesh is obtained by equally distributing the squared edge length under $|\mathbf{H}|$. Consider two principal directions $\vec{e}_i$ and $\vec{e}_j$ from $\mathbf{Q}$. Error equidistribution yields the mesh stretching as

$$\frac{h_i}{h_j} = \left(|u_{\vec{e}_j}|/|u_{\vec{e}_i}|\right)^{1/(p+1)} = \left(|\lambda_j|/|\lambda_i|\right)^{1/(p+1)}, \tag{10}$$

where $u_{\vec{e}_i}$ is the $p+1^{\text{st}}$ derivate along the direction $\vec{e}_i$, whose mangitude is characterized by the i^{th} eigenvalue λ_i. The key idea of Hessian-based mesh adaptation with output error estimation is to use (10) to control the mesh stretching (relative mesh size) while using the output error indicator in (7) to determine the absolute mesh sizes.

4.2. Element Sizing using A Priori Rate Estimates

In order to perform the mesh adaptation, we need to predict the desired element sizes, or the number of the elements N^f in the adapted (fine) mesh. Let n_e, not necessarily an integer, estimate the number of adapted mesh elements contained in element e of the original mesh. Denoting the current element size by h_i^c and the requested element size by h_i^f, where i again indexes the principal directions, n_e can be approximated as

$$n_e = \prod_{i=1}^{d} (h_i^c/h_i^f). \tag{11}$$

The current sizes h_i^c are calculated as the singular values of the mapping from a unit equilateral triangle to element e [6]. Given an output error tolerance $\mathcal{E}^f$, to satisfy the error equidistribution, each fine-mesh element is allowed an error $\mathcal{E}^f/N^f$, which means that element e is allowed an error of $n_e\mathcal{E}^f/N^f$. We relate the growth in the number of elements to an error reduction factor through an *a priori* estimate

$$\underbrace{n_e \frac{\mathcal{E}^f}{N^f}}_{\text{allowable error}} = \underbrace{\mathcal{E}_e^c \left(\frac{h_{\text{ref}}^f}{h_{\text{ref}}^c}\right)^{\bar{p}_e+1}}_{\text{a priori estimate}} = \mathcal{E}_e^c \left[\prod_{i=1}^{d} \left(\frac{h_i^f}{h_i^c}\right)\right]^{(\bar{p}_e+1)/d}, \tag{12}$$

where h_{ref}^c and h_{ref}^f are the current and the adapted reference element sizes characterizing the error convergence, $\mathcal{E}_e^c$ indicates the current error indicator in element e, $\bar{p}_e = \min(p_e, \gamma_e)$, and γ_e is the lowest order of any singularity within element e. Substituting (11) into (12) yields a relation between n_e and N^f

$$n_e \frac{\mathcal{E}^f}{N^f} = \mathcal{E}_e^c \left[\prod_{i=1}^{d} \left(\frac{h_i^f}{h_i^c}\right)\right]^{(\bar{p}_e+1)/d} = \mathcal{E}_e^c n_e^{-(\bar{p}_e+1)/d} \quad \Rightarrow \quad n_e^{1+(\bar{p}_e+1)/d} = \frac{\mathcal{E}_e^c}{\mathcal{E}^f/N^f}. \tag{13}$$

Substituting (13) into $N^f = \sum_{e=1}^{N_e} n_e$, we can solve for N^f if $\bar{p}_e$ is constant on the entire mesh, otherwise it is solved iteratively. In this work, all of the test cases use a constant order, $p_e = p$, and for simplicity, no regularity estimate is employed, so that $\gamma_e = p$. Adaptation can be terminated when the error tolerance is met, $\mathcal{E} \equiv \sum_{e=1}^{N_e} \mathcal{E}_e \leq \mathcal{E}^f$. In practice, the error tolerance can be modified or a fixed-growth refinement strategy can be adopted to control the mesh adaptation [10].

Alternatively, the adaptation algorithm can be modified for the case when we are not given $\mathcal{E}^f$ but instead the target number of elements, N_f. This is typical in an adaptive setting where the growth of the cost is controlled and we seek

the best possible solution for a given cost. Combining (12) with the constraint that $\sum_{e=1}^{N_e} n_e = N^f$, and approximating $n_e \approx (h^c_{\text{ref}}/h^f_{\text{ref}})^d$, we can solve for $\mathcal{E}^f$,

$$\mathcal{E}^f = N^f \left(\frac{1}{N^f} \sum_{e=1}^{N_e} (\mathcal{E}^c_e)^{r'} \right)^{1/r'}, \tag{14}$$

where $r' = d/(p+1+d)$.

With both $\mathcal{E}^f$ and N^f known, the last part of the adaptive algorithm is the identification of the scaling factor in element size, h^f_i/h^c_i. This is necessary because the anisotropy information, consisting of principal directions and desired aspect ratios, comes from an arbitrarily-scaled metric, e.g. the Hessian, which cannot be used to obtain the element size. In this work, we determine the absolute mesh sizing through the smallest principal stretching and the corresponding aspect ratios along other principal directions. Starting from the expression in (13), for element e we have,

$$n_e^{1+(p+1)/d} = \left[\prod_{i=1}^{d} \left(\frac{h^c_i}{h^f_i} \right) \right]^{1+(p+1)/d} = \left[\left(\frac{h^c_1}{h^f_1} \right)^d \prod_{i=1}^{d} \left(\frac{AR^c_i}{AR^f_i} \right) \right]^{1+(p+1)/d} = \frac{\mathcal{E}^c_e}{\mathcal{E}^f/N^f}, \tag{15}$$

where $AR_i = h_i/h_1$ denotes the aspect ratio along the i^{th} principal direction with respect to the smallest stretching direction, h_1. Therefore, the desired element size can be predicted as

$$\frac{h^f_1}{h^c_1} = \left[\frac{\mathcal{E}^f}{N^f} \frac{1}{\mathcal{E}^c_e} \right]^{1/(p+1+d)} \left[\prod_{i=1}^{d} \left(\frac{AR^c_i}{AR^f_i} \right) \right]^{1/d}, \quad h^f_i = h^f_1 AR^f_i. \tag{16}$$

During adaptation, the new metric for element e is constructed by combining the principal directions and stretching information from the desired metric with the above scaling in the element size.

4.3. Sampling-Based Mesh Optimization

The goal in this method, MOESS, is to optimize the computational mesh, T_h, in order to minimize the output error at a prescribed computational cost. We present the approach introduced by Yano [37], which iteratively determines the optimal change in the mesh metric field given a prescribed metric-cost relationship and a sampling-inferred metric-error relationship.

4.3.1. Metric-Based Meshing

A Riemannian metric field, $\mathcal{M}(\vec{x})$, is a field of symmetric positive definite (SPD) tensors that can be used to encode information about the desired size and stretching of a computational mesh. At each point in physical space, $\vec{x}$, the metric tensor $\mathcal{M}(\vec{x})$ provides a "yardstick" for measuring the distance from $\vec{x}$ to another point infinitesimally far away, $\vec{x} + \delta\vec{x}$. After choosing a Cartesian coordinate system and basis for physical space, $\mathcal{M}$ can be represented as a $d \times d$ SPD matrix. The set of points at unit metric distance from $\vec{x}$ is an ellipse: eigenvectors of $\mathcal{M}$ give directions along the principal axes, while the length of each axis (stretching) is the inverse square root of the corresponding eigenvalue. The aspect ratio is the ratio of the largest stretching magnitude to the smalleset.

A mesh that *conforms* to a metric field is one in which each edge has the same length, to some tolerance, when measured with the metric. An example of a two-dimensional metric-conforming mesher is the Bi-dimensional Anisotropic Mesh Generator (BAMG) [38], and this is used to obtain the results in the present work.

BAMG generates a mesh given a metric field, which is specified at nodes of a background mesh – the current mesh in an adaptive setting. The optimization determines *changes* to the current, mesh-implied, metric, $\mathcal{M}_0(\vec{x})$. Affine-invariant [39] changes to the metric field are made via a symmetric *step matrix*, $\mathcal{S} \in \mathbb{R}^{d \times d}$, according to

$$\mathcal{M} = \mathcal{M}_0^{\frac{1}{2}} \exp(\mathcal{S}) \mathcal{M}_0^{\frac{1}{2}}. \tag{17}$$

Note that $\mathcal{S} = 0$ leaves the metric unchanged, while diagonal values in $\mathcal{S}$ of $\pm 2 \log 2$ halve/double the metric stretching sizes.

4.3.2. Error Convergence Model

The mesh optimization algorithm requires a model for how the error changes as the metric changes. We consider one element, Ω_e, with a current error $\mathcal{E}_{e0}$, the absolute value of the element's contribution to (7), and a proposed metric step matrix of $\mathcal{S}_e$. The error on Ω_e following refinement with this step matrix is given by

$$\mathcal{E}_e = \mathcal{E}_{e0} \exp\left[\mathrm{tr}(\mathcal{R}_e \mathcal{S}_e)\right], \tag{18}$$

where $\mathcal{R}_e$ is a symmetric *rate tensor*. The total error over the mesh is the sum of the elemental errors, $\mathcal{E} = \sum_{e=1}^{N_e} \mathcal{E}_e$. The rate tensor, $\mathcal{R}_e$, is determined separately for each element through a sampling procedure [13].

4.3.3. Cost Model

To measure the cost of refinement, we use degrees of freedom, `dof`, which on each element just depends on the approximation order p, assumed constant over the elements. By (17) and properties of the metric tensor, when the step matrix S_e is applied to the metric of element e, the area of the element decreases by $\exp\left[\frac{1}{2}\mathrm{tr}(S_e)\right]$. Equivalently, the number of new elements, and hence degrees of freedom, occupying the original area Ω_e *increases* by this factor. So the elemental cost model is

$$C_e = C_{e0} \exp\left[\frac{1}{2}\mathrm{tr}(\mathcal{S}_e)\right], \tag{19}$$

where $C_{e0} = \mathtt{dof}_{e0}$ is the current number of degrees of freedom on element e. The total cost over the mesh is the sum of the elemental costs, $C = \sum_{e=1}^{N_e} C_e$.

4.3.4. Metric Optimization Algorithm

Given a current mesh with its mesh-implied metric ($\mathcal{M}_0(\vec{x})$), elemental error indicators $\mathcal{E}_{e0}$, and elemental rate tensor estimates, $\mathcal{R}_e$, the goal of the metric optimization algorithm is to determine the step matrix field, $\mathcal{S}(\vec{x})$, that minimizes the error at a fixed cost.

The step matrix field is approximated by values at the mesh vertices, $\mathcal{S}_v$, which are arithmetically-averaged to adjacent elements[2]:

$$\mathcal{S}_e = \frac{1}{|V_e|} \sum_{v \in V_e} \mathcal{S}_v, \tag{20}$$

where V_e is the set of vertices ($|V_e|$ is the number of them) adjacent to element e. The optimization problem is to determine $\mathcal{S}_v$ such that the total error $\mathcal{E}$ is minimized at a prescribed total cost C.

First-order optimality conditions require derivatives of the error and cost with respect to $\mathcal{S}_v$. We note that the cost only depends on the *trace* of the step matrix; i.e. the trace-free part of $\mathcal{S}_e$ stretches an element but does not alter its area. We therefore separate the vertex step matrices into trace ($s_v \mathcal{I}$) and trace-free ($\widetilde{\mathcal{S}}_v$) parts, with $\mathcal{I}$ the identity tensor,

$$\mathcal{S}_v = s_v \mathcal{I} + \widetilde{\mathcal{S}}_v. \tag{21}$$

The optimization algorithm is then the same as presented by Yano [37]:

1. Given a mesh, solution, and adjoint, calculate $\mathcal{E}_e, C_e, \mathcal{R}_e$ for each element e.
2. Set $\delta s = \delta s_{\max}/n_{\text{step}}$, $\mathcal{S}_v = 0$.
3. Begin loop: $i = 1 \ldots n_{\text{step}}$
   (a) Calculate $\mathcal{S}_e$ from (20), $\frac{\partial \mathcal{E}_e}{\partial \mathcal{S}_e}$ from (18), and $\frac{\partial C_e}{\partial \mathcal{S}_e}$ from (19).
   (b) Calculate derivatives of $\mathcal{E}$ and C with respect to s_v and $\widetilde{\mathcal{S}}_v$.
   (c) At each vertex form the ratio $\lambda_v = \frac{\partial \mathcal{E}/\partial s_v}{\partial C/\partial s_v}$ and

[2]There is no need for an affine-invariant average because entries of $\mathcal{S}$ are coordinate system independent.

- Refine the metric for 30% of the vertices with the largest $|\lambda_v|$: $\mathcal{S}_v = \mathcal{S}_v + \delta s \mathcal{I}$
- Coarsen the metric for 30% of the vertices with the smallest $|\lambda_v|$: $\mathcal{S}_v = \mathcal{S}_v - \delta s \mathcal{I}$

(d) Update the trace-free part of S_v to enforce stationarity with respect to shape changes at fixed area: $S_v = S_v + \delta s(\partial\mathcal{E}/\partial\widetilde{\mathcal{S}}_v)/(\partial\mathcal{E}/\partial s_v)$.

(e) Rescale $S_v \to S_v + \beta\mathcal{I}$, where β is a global constant calculated from (19) to constrain the total cost to the desired `dof` value: $\beta = \frac{2}{d}\log\frac{C_{\text{target}}}{C}$, where C_{target} is the target cost.

Note, λ_v is a Lagrange multiplier in the optimization. It is the ratio of the marginal error to marginal cost of a step matrix trace increase (i.e. mesh refinement). The above algorithm iteratively equidistributes λ_v globally so that, at optimum, all elements have the same marginal error to cost ratio. Constant values that work generally well in the above algorithm are $n_{\text{step}} = 20$ and $\delta s_{\max} = 2\log 2$.

In practice, the mesh optimization and flow/adjoint solution are performed several times at a given target cost, C_{target}, until the error stops changing. Then the target cost is increased to reduce the error further if desired.

5. Machine-Learning Anisotropy Detection

As an alternative to mesh subdivision sampling and regression to determine the anisotropy information, we present an approach that uses a neural network to determine anisotropy from relevant features of the primal and adjoint solution. A key choice in the design of the neural network is the set of features from which the anisotropy information can be detected. For generality across problems, the features should be insensitive to scaling of the problem and the choice of physical units. In the Hessian-based anisotropy detection approach, the Mach number Hessian matrix provides such information from the primal solution. In MOESS both the primal and adjoint solutions are combined via sampling of the adjoint-weighted residual to produce the most efficient anisotropy distribution. Note that in both methods, the element size is obtained from the output error estimate.

The output error estimation formula involves both the primal (via the residual) and the adjoint (as a weight) solutions, and we therefore seek to incorporate both sets of features into the anisotropy measure. Our choice for the features is motivated by the success of the Mach number Hessian, even for high-order solutions. Properly normalized, the Hessian matrix provides information about the relative importance of principal orthogonal directions computed from a scalar field. In order to not exclude any primal or adjoint information, we take as features the Hessian matrices computed from all primal and adjoint state components.

5.1. Primal and Adjoint Hessian Features

For element e, denote by $\mathbf{H}_k^u$ and $\mathbf{H}_k^\psi$ the Hessian matrices of the k^{th} state and adjoint variables, respectively,

$$\left[\mathbf{H}_k^u\right]_{i,j} = \frac{\partial u_k^2}{\partial x_i \partial x_j}, \quad \left[\mathbf{H}_k^\psi\right]_{i,j} = \frac{\partial \psi_k^2}{\partial x_i \partial x_j}, \quad i,j \in [1,...,d], \tag{22}$$

where $k \in [1,\ldots,s]$ ranges over the state rank. We do not include the subscript e to denote that this is an elemental quantity, as this will be assumed for all calculations in this section. The second derivatives in the Hessian are computed from the polynomial approximation of the state inside element e. When the approximation order is $p = 2$, the Hessian is uniquely defined for the entire element. To obtain a unique definition for orders $p > 2$, we project the field, via least squares, to $p = 2$. For $p < 2$, which we do not consider in this work, we would obtain a $p = 2$ field using patch reconstruction from neighboring elements.

In two spatial dimensions, to normalize the Hessian, we compute the aspect ratio and direction of the first eigenvector,

$$AR = \sqrt{\lambda_2^H/\lambda_1^H}, \quad \theta = \arg(\vec{v}_1), \tag{23}$$

where $\lambda_1^H \le \lambda_2^H$ are the Hessian eigenvalues and $\vec{v}_1$ is the eigenvector corresponding to λ_1^H. The normalized elemental Hessian metric, for either a primal or adjoint field k, is then defined as

$$\overline{\mathbf{H}}_k = \begin{bmatrix} \lambda_1\cos^2\theta + \lambda_2\sin^2\theta & (\lambda_1-\lambda_2)\sin\theta\cos\theta \\ (\lambda_1-\lambda_2)\sin\theta\cos\theta & \lambda_1\sin^2\theta + \lambda_2\cos^2\theta \end{bmatrix}, \quad \text{where } \lambda_1 = \frac{1}{AR},\ \lambda_2 = AR. \tag{24}$$

Note that this matrix does not encode sizing, which is not a meaningful quantity from the Hessian matrix in the context of output-based adaptation, and hence has only two independent quantities (e.g. AR and θ). This gives a total of $2s$ primal features and $2s$ adjoint features for use in the machine-learning algorithm.

Inputs into a neural network should be normalized so as not to bias the training towards extreme cases, e.g. in this case to elements of large aspect ratio. In addition, care must be taken when using an angle as an input, due to the equivalence of 0 and 2π. To address both problems, we do not use the metric as defined in (24) directly as an input. Nor do we use AR and θ, which can have different scales. Instead, we take the natural logarithm of the metric, which is equivalent to a step matrix computed relative to the identity matrix. Since the eigenvalues of $\overline{\mathbf{H}}_k$ are multiplicative inverses, the resultant step matrix has zero trace and hence only two independent components in two dimensions. Without loss of generality, we use the first row values as the independent components:

$$\mathbf{S}_k = \log\left(\overline{\mathbf{H}}_k\right) \quad \Rightarrow \quad S_{k,1} \equiv (\mathbf{S}_k)_{1,1}, \; S_{k,2} \equiv (\mathbf{S}_k)_{1,2}. \tag{25}$$

These two values per scalar field result in a total of $4s$ inputs from the primal and adjoint anisotropy information.

5.2. Error Indicator Features

In addition to the primal and adjoint features, state-component contributions to the error indicator can also inform the element anisotropy calculation. The elemental error indicator $\mathcal{E}$ in (7) (again, we drop the subscript e for conciseness) is a scalar that is already used to determine element size. However, for systems of equations, this indicator can be broken down into contributions from each equation,

$$\mathcal{E} = \sum_{k=1}^{s} \mathcal{E}_k, \qquad \mathcal{E}_k = \left|\mathcal{R}_h^{p+1}(\mathbf{u}_h^p, \mathbf{Z}_k \boldsymbol{\psi}_h^{p+1}\big|_{\Omega_e})\right|, \tag{26}$$

where $\mathbf{Z}_k \in \mathbb{R}^{s\times s}$ is a mask matrix of all zeros except for a 1 on the main diagonal, at row k. The relative magnitudes of $\mathcal{E}_k$ can then be incorporated into the anisotropy calculation, as these values could arguably influence the importance of Hessian anisotropy information from the various state and adjoint components. Note that we do not assume a direct weighting by these indicators, but rather let the neural network determine the effect of these values on the output. For input into the neural network, we define the normalized error contributions as

$$\overline{\mathcal{E}}_k \equiv \mathcal{E}_k \left(\sum_{k=1}^{s} \mathcal{E}_k^2\right)^{-1/2}. \tag{27}$$

5.3. Neural Network Architectures

In this work we consider four neural network architectures for mapping input features into the desired output. In all networks, the output layer, $\mathbf{x}_{\text{out}}$ contains the desired anisotropy information, as encoded by a step matrix that is the natural logarithm of the normalized output metric, $\overline{\mathbf{M}}_{\text{out}}$,

$$\mathbf{S}_{\text{out}} = \log\left(\overline{\mathbf{M}}_{\text{out}}\right) \quad \Rightarrow \quad \mathbf{x}_{\text{out}} = \left[(\mathbf{S}_{\text{out}})_{1,:}\right]^T. \tag{28}$$

As the output step matrix is trace-free, the number of outputs is only 2, $\mathbf{x}_{\text{out}} \in \mathbb{R}^2$. By the above equation, the output layer is the first row of the output step matrix. We note that using the metric logarithm ensures validity of the output metric regardless of the sign and magnitude of the network output.

Figure 1 shows the structure of the networks, which is a multi-layer perceptron. It consists of one or two hidden layers, $\mathbf{x}_1, (\mathbf{x}_2)$, between the input layer, i.e. the features, $\mathbf{x}_0$, and the output layer, $\mathbf{x}_{\text{out}}$. The networks are fully-connected, and the formulas for computing the states within each layer are shown in Figure 1. The map from the input to the hidden layer involves an entry-wise sigmoid activation function, $\sigma(x) = 1/(1+e^{-x})$, whereas no activation function is used for the output layer calculation. The parameters associated with the network consist of the weights and biases,

$$\mathbf{W}_i \in \mathbb{R}^{n_i \times n_{i-1}}, \quad \mathbf{b}_i \in \mathbb{R}^{n_i},$$

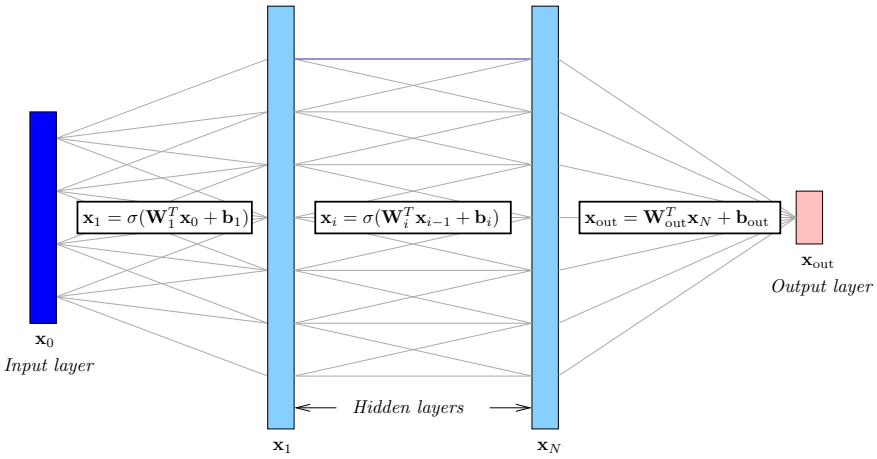


Figure 1: Structure of the artificial neural networks used to predict element anisotropy and orientation. The number of hidden layers, N, is 1 or 2.

where n_i is the number of neurons in layer i.

The four networks, labelled $A-D$, differ in the size of the input layer and the number and size of the hidden layers. Table 1 gives the specifications of each network. The inputs to all networks contain the primal and adjoint Hessian information, through the $4s$ values in $\mathbf{S}^u_{k,1,:}$ and $\mathbf{S}^\psi_{k,1,:}$. Networks A-C also include the s relative errors, $\mathcal{E}_k$ in the input. Networks A and D both have one hidden layer of $10s$ neurons, network B has two hidden layers of $40s$ neurons each, and network C has one hidden layer of $2s$ neurons. The associated numbers of weights and biases are given in the last two columns of Table 1. We designate network A as the *baseline* as this was the initial network that was found, based on preliminary tests, to show accurate performance on the training cases without being excessively large. Relative to A, B is a large/fine network, C is a small/coarse network, and D is the baseline without the relative error indicators.

Table 1: Neural network specifications

Network	**Input Layer**	**Hidden Layers**	**# Weights**	**# Biases**
A	$\mathbf{x}_0 \in \{\mathbf{S}^u_{k,1,:}, \mathbf{S}^\psi_{k,1,:}, \mathcal{E}_k\} \in \mathbb{R}^{5s}$	$\mathbf{x}_1 \in \mathbb{R}^{10s}$	$50s^2 + 20s$	$10s + 2$
B	$\mathbf{x}_0 \in \{\mathbf{S}^u_{k,1,:}, \mathbf{S}^\psi_{k,1,:}, \mathcal{E}_k\} \in \mathbb{R}^{5s}$	$\mathbf{x}_1 \in \mathbb{R}^{40s}, \mathbf{x}_2 \in \mathbb{R}^{40s}$	$1800s^2 + 80s$	$80s + 2$
C	$\mathbf{x}_0 \in \{\mathbf{S}^u_{k,1,:}, \mathbf{S}^\psi_{k,1,:}, \mathcal{E}_k\} \in \mathbb{R}^{5s}$	$\mathbf{x}_1 \in \mathbb{R}^{2s}$	$10s^2 + 4s$	$2s + 2$
D	$\mathbf{x}_0 \in \{\mathbf{S}^u_{k,1,:}, \mathbf{S}^\psi_{k,1,:}\} \in \mathbb{R}^{4s}$	$\mathbf{x}_1 \in \mathbb{R}^{10s}$	$40s^2 + 20s$	$10s + 2$

The parameters associated with the network consist of all of the weights and biases. The values of these parameters are determined using an optimization procedure, the adaptive moment (Adam) estimation algorithm in TensorFlow [40], that minimizes the mean squared error loss function between predicted and actual output layer values. The actual values come from training data, which are obtained from multiple MOESS iterations on prototypical cases. Each MOESS iteration produces one "training point" for each element of the mesh, so for meshes with many elements, a large amount of data can be obtained with a relatively small number of MOESS iterations.

Once a network is trained, it is implemented in the adaptive code as a replacement for the Hessian-only metric anisotropy and stretching calculation in the a priori output-based mesh refinement method described in Section 4.2. Figure 2 illustrates the calculation process from the features: primal, adjoint, and error indicators, to the desired element anisotropy, encoded by the metric $\overline{\mathbf{M}}_{\text{out}}$ presented on the lower-left in the figure. This metric then replaces the Mach Hessian in an otherwise equivalent adaptive procedure. In particular, note that MOESS is not used with the neural network model.

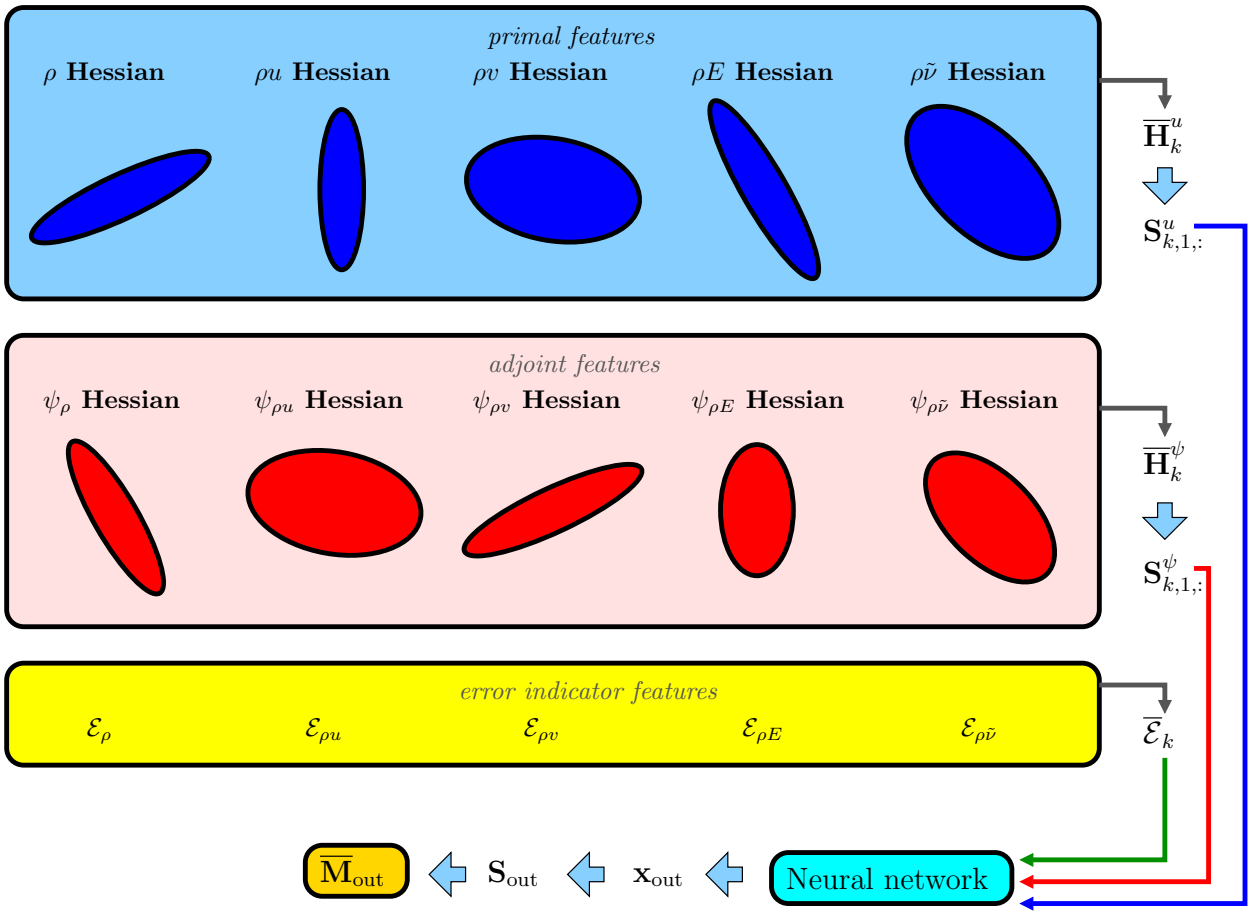


Figure 2: Flowchart of the neural-network implementation.

6. Neural Network Training

Several prototypical aerodynamic flow cases are run to provide training data for the anisotropy prediction neural network. Table 2 describes these cases and includes figures of the primal and adjoint solutions, and of the optimized meshes. All cases are two-dimensional and use the Reynolds-averaged Navier-Stokes equations. A variety of flow Mach numbers are included, ranging from subcritical to supersonic. The Reynolds numbers are representative of aircraft flight conditions, $O(10^6)$. Force outputs, drag and lift, are considered for error estimation and adaptation. In some cases, for a given flow condition, both outputs yield two different adaptation sequences.

Adapted meshes are generated at a chosen target degree-of-freedom cost using MOESS, at a solution approximation order of $p = 2$ and separately at $p = 3$ (for use in the results in Section 7.6). The meshes for the different cases are not all made to be of the same size, but the number of adaptive iterations collected is chosen such that the total amount of training data (product of mesh size and number of iterations) is approximately the same among the cases. Data collection begins once the mesh size and outputs stabilize following initialization of MOESS iterations with a user-generated, sub-optimal mesh. Specifically, data from ten MOESS iterations are discarded before collection.

The input data, which consist of the normalized primal and adjoint variable Hessian matrices, are computed from the current-space primal and adjoint solutions for each element. For $p = 2$, these matrices are constant across elements, whereas for $p > 2$, the matrices would be averaged across the element interiors. The output data are computed using the normalized grid-implied metric, since we are using MOESS to generate the meshes, converted to a step matrix via a natural logarithm. We note that no changes are made to MOESS to enable training of the networks. Instead MOESS is used as a black-box algorithm, and only the resulting meshes are used as training outputs.

The total number of training data points (elements) over all cases and adaptive iterations is 121,840. These data are randomized and split 70%/30% into training and validation categories. The training data are used to drive the optimization, whereas the validation data are used to monitor the loss on untrained data. The training data are broken into mini-batches of size 1000 for the optimizer, and the learning rate is set to .001. Prior to training, the weights and biases are initialized randomly from a unit normal distribution. Several tens of thousands of optimization iterations typically lead to a stabilization of the mean-squared error, as shown in Figure 3. Typically, an order of magnitude drop in the loss is observed, without a significant difference between training and test data loss. This indicates that we are not over-fitting the data, which would be difficult to do with the small neural network size presently considered. Furthermore, the results of the training were not found to be overly sensitive to the choices of the mini-batch size, learning rate, or the initialization.

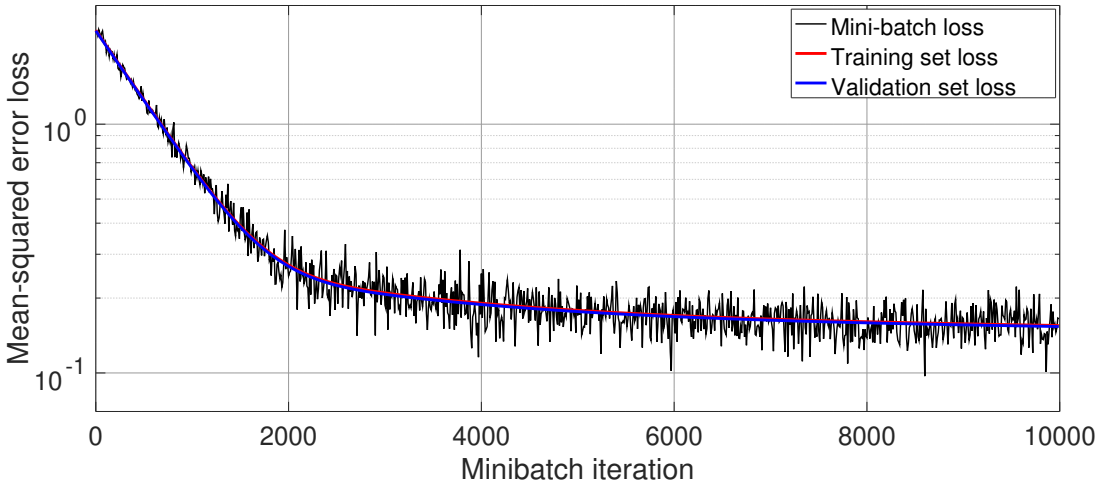
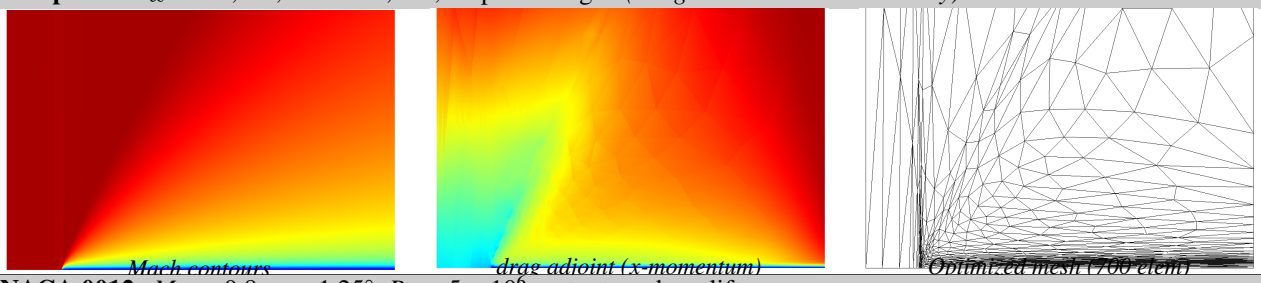


Figure 3: Neural network A training loss history, using a 70% training, 30% validation split.

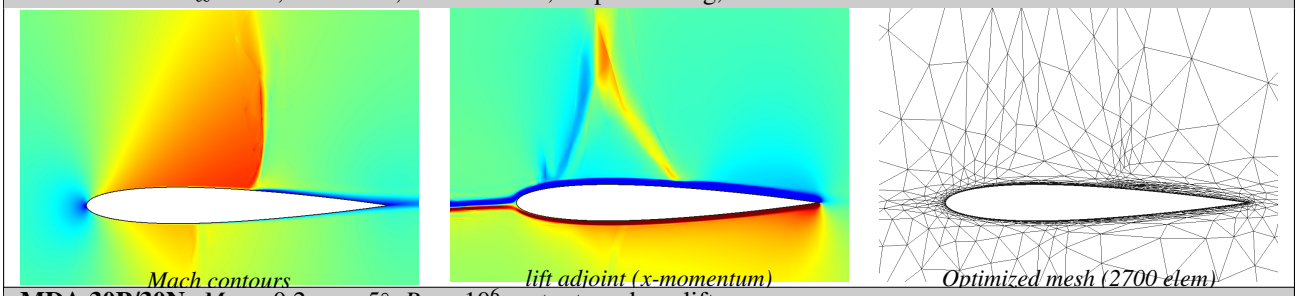
Training of the networks yields weight matrices and bias vectors. Interpretation of these values is not straightforward, especially for large networks. However, some insight can be gained from a graphical representation, as shown

Table 2: Neural network training cases.

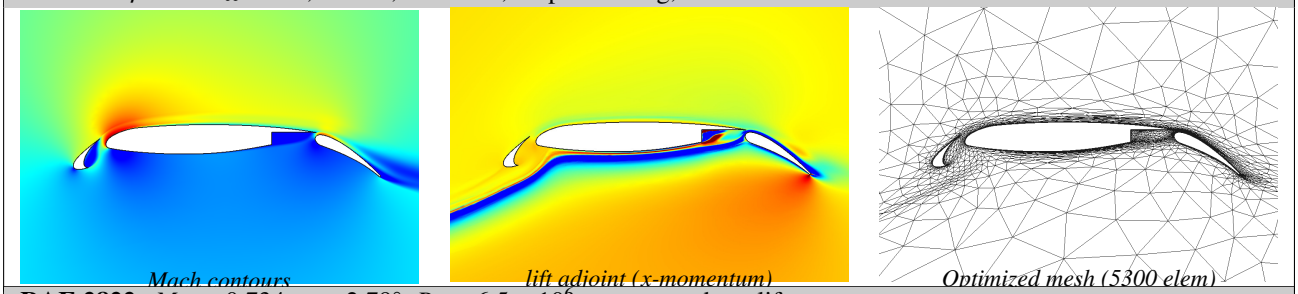
Flat plate: $M_\infty = 0.2, 0.5$, $Re = 10^5, 10^6$, output = drag *(images scaled* 100x *vertically)*



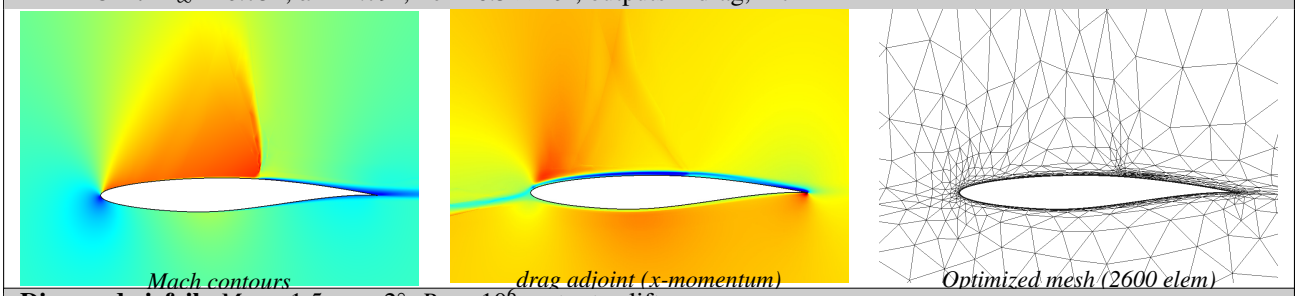
NACA 0012: $M_\infty = 0.8$, $\alpha = 1.25°$, $Re = 5 \times 10^6$, outputs = drag, lift



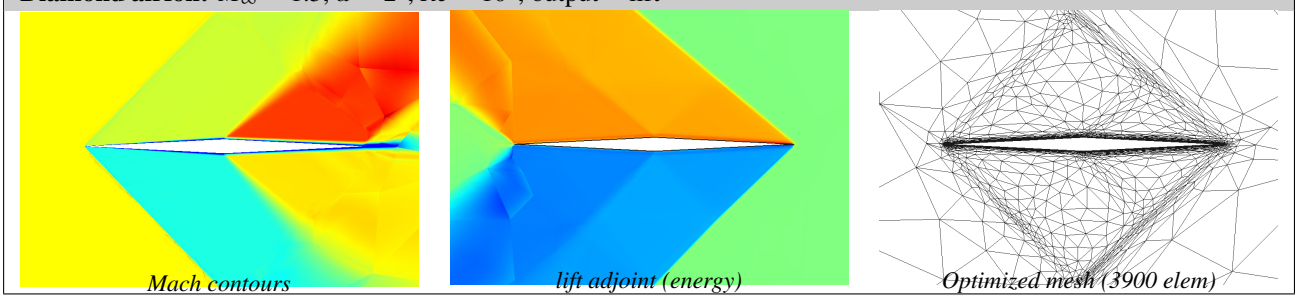
MDA 30P/30N: $M_\infty = 0.2$, $\alpha = 5°$, $Re = 10^6$, outputs = drag, lift



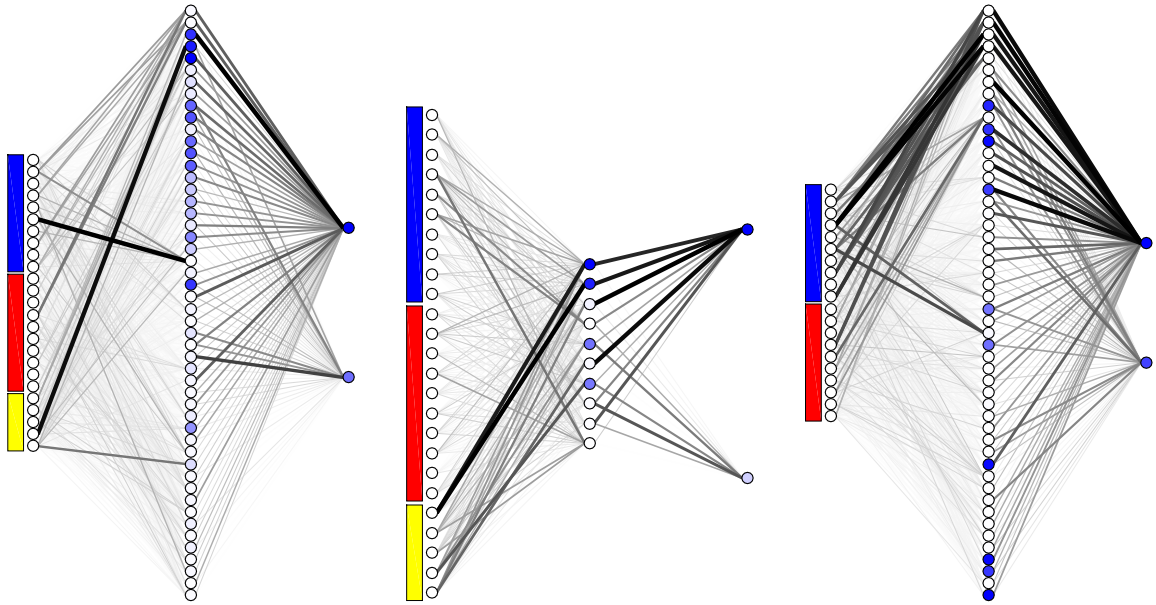
RAE 2822: $M_\infty = 0.734$, $\alpha = 2.79°$, $Re = 6.5 \times 10^6$, outputs = drag, lift



Diamond airfoil: $M_\infty = 1.5$, $\alpha = 2°$, $Re = 10^6$, output = lift



in Figure 4 for networks A,C, and D (network B is too large to clearly visualize). These have all been trained using $p = 2$ solutions. Note that the hidden neurons have been sorted based on importance, calculated from incoming and outgoing weight magnitudes, with the most important/active neurons at the top. We see that in network A, the primal and adjoint features both contribute to the final output (based on the weights of connections emanating from the input blocks), with the primal features showing more significance compared to the adjoint, and that the error indicators (particularly the fourth, energy component) are active. In network C, which has a small hidden layer, the primal features contribute somewhat more than the adjoint features, and the error indicators are quite active (large magnitude weights). Finally, for network D, which does not use the error indicators, the primal features are more active than the adjoint features. In addition, in networks A and C, the bias magnitudes are large for the important hidden layer neurons, whereas this is not the case for network D.



(a) Network A (b) Network C (c) Network D

Figure 4: Visualization of the trained neural networks. The input highlighting indicates primal (blue), adjoint (red), and error indicator (yellow) neurons. The darkness and width of connecting lines indicate weight magnitudes, and the color of the neurons indicates bias magnitude. The hidden layer neurons are sorted by importance, using the product of the sum of incoming weight magnitudes and the sum of outgoing weight magnitudes. The two outputs are $S_{\text{out},1,1}$ and $S_{\text{out},1,2}$.

7. Adaptive Simulation Results

This section presents results obtained by implementing the trained neural network in an adaptive solution process and using it to generate adapted meshes for various flow cases at different degrees of freedom. When using the network to determine anisotropy information, the same a-priori element sizing technique, described in Section 4.2, is used as in the Hessian-based approach. The neural network results are compared to both Mach number Hessian-based adaptation and MOESS. In all cases, metric-based re-meshing is performed using BAMG [38], and mesh elements are curved in the presence of curved boundaries through a linear elasticity approach [41].

7.1. NACA 0012 airfoil

This test case consists of RANS fully-turbulent flow over a NACA 0012 airfoil at $M_\infty = 0.8$, $\alpha = 1.25°$, and $Re = 5 \times 10^6$. The computational domain consists of a square farfield boundary 100 chords away from the airfoil. An adiabatic no-slip wall boundary conditions is imposed on the airfoil. Shock capturing is performed using element-based artificial viscosity [42], and the output of interest is the drag coefficient on the airfoil. A hand-generated initial

isotropic mesh of 2800 elements is constructed with sufficient boundary-layer resolution to enable a converged RANS solution.

Adaptive simulations are performed at $p = 2$ solution approximation order for three target degree-of-freedom (`dof`) costs: $8000, 16000, 32000$. At each target `dof`, 10 adaptive iterations are run, using the final mesh from the previous target `dof` as the starting mesh at each iteration. The adaptive methods compared are MOESS, Mach-number Hessian, and the neural-network approaches.

Figure 5 presents the output convergence histories obtained from the adaptive runs. For each method, only one data point is shown at a given `dof` target: this is the average output at the average `dof` value computed over the last 4 adaptive iterations at that target `dof`. A reference value is also indicated, obtained by running a simulation at an approximation order of $p = 3$ on a uniformly-refined version of the final MOESS adapted mesh.

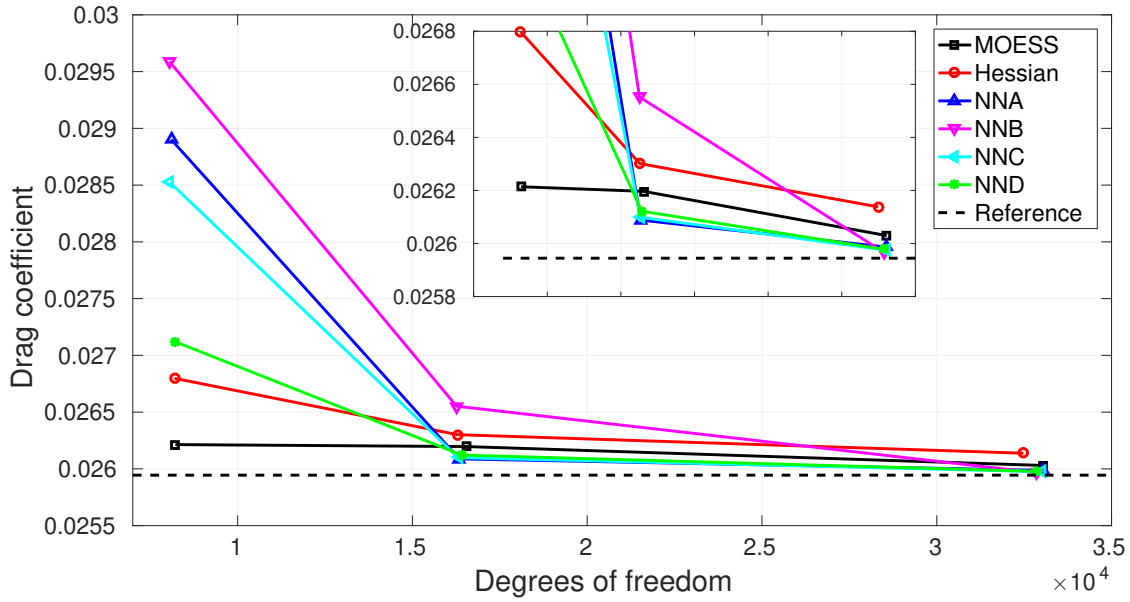


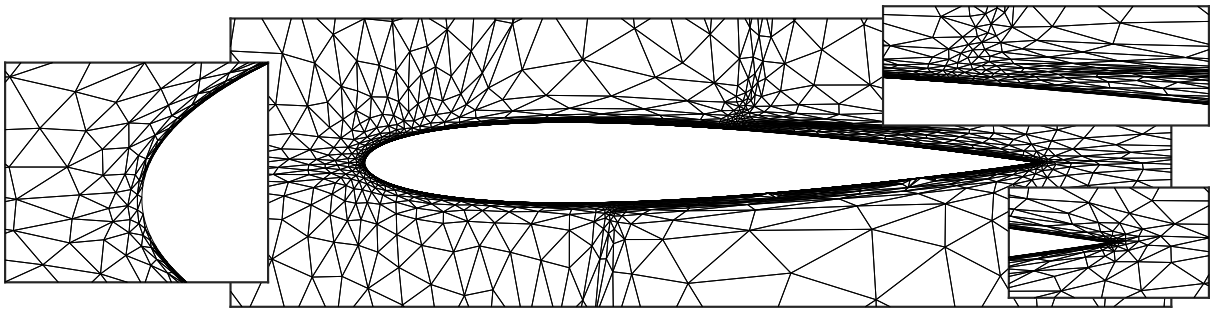
Figure 5: NACA 0012, $M_\infty = 0.8$, $\alpha = 1.25°$, $Re = 5 \times 10^6$: output convergence history.

We see that after the coarsest target `dof`, the adaptive results using most of the neural-network methods are closer to the reference value compared to those using the Mach number Hessian anisotropy. By the finest target `dof`, this is true for all of the methods. As both methods are driven by the same output-based element sizing information, the observed error reduction is made possible by more efficient meshes resulting from improved anisotropy information. Specifically, correct anisotropy identification allows for optimal use of degrees of freedom to reduce the output error.

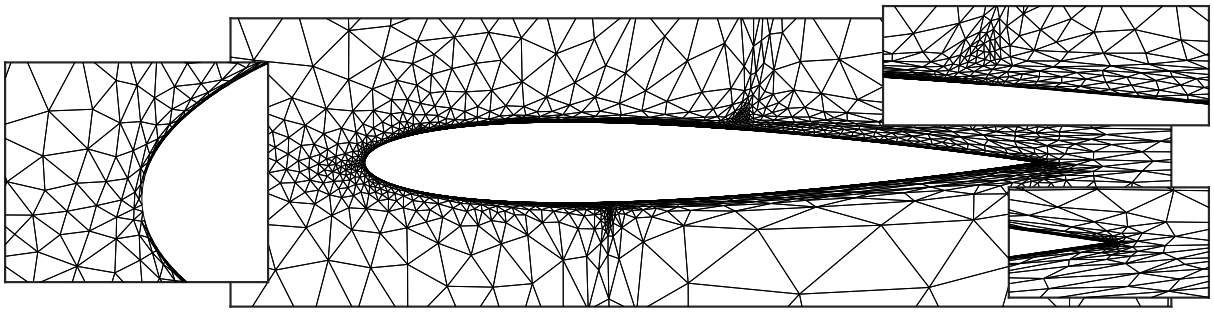
The neural network approaches can achieve more optimal mesh anisotropy because they incorporates information not only from the primal solution, but also from the adjoint. The networks were trained to reproduce the element stretching obtained from MOESS, but they also include a degree of regularization, which leads to a slightly-improved performance over MOESS. This point is discussed further in the subsequent results. We note that this regularization stems from the small size of the network, rather than any explicit regularization techniques during training, which were not used.

Figure 6 shows the final adapted meshes at the highest target `dof` for three of the adaptive approaches tested. The neural network meshes are similar, and hence only results from one network (A) are shown. The flow is transonic with a strong shock on the upper surface, a weaker shock on the lower surface, and rapid boundary-layer growth in the vicinity of the shocks. From the three figures, we see that the resolution of the shock, which is minimal to start with in the output-based setting, is similar among the methods, with the neural-network indicator exhibiting slightly less primal-based anisotropy compared to the Hessian indicator and MOESS.

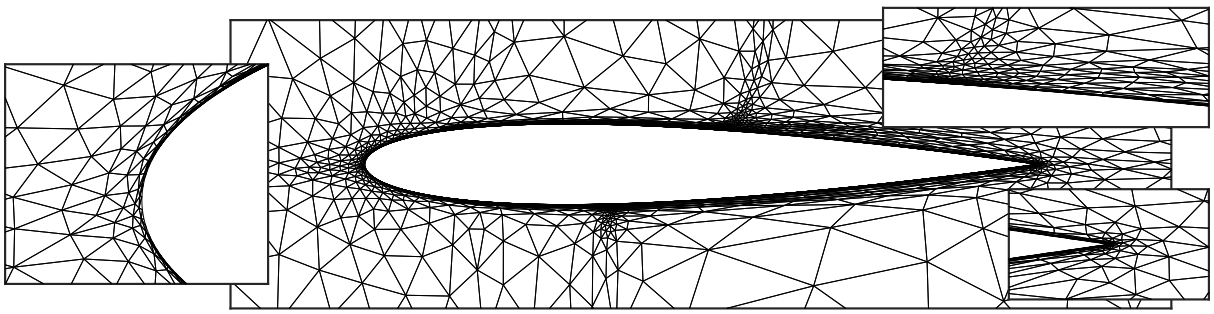
As shown in Table 2, the adjoint variable possesses a λ structure variation on the upper surface, and both the neural-network anisotropy measure and MOESS show evidence of its resolution. The Mach number Hessian measure



(a) MOESS



(b) Hessian



(c) Neural network A

Figure 6: NACA 0012: final adapted meshes.

does not align elements to this structure, which is specific to the adjoint. The most notable difference is the leading-edge stagnation streamline anisotropy, which is again an adjoint feature that is resolved by MOESS and the neural networks. Although the extent to which this feature needs to be resolved for accurate output prediction is still a point of debate, it is reassuring to see the neural network reproduce the MOESS behavior, for which it was trained. Finally, near the trailing edge, the Hessian-based measure shows more flow-aligned anisotropy, due to the primal wake, which is not as apparent in MOESS and the neural-network approach.

7.2. Diamond airfoil

This test case consists of RANS supersonic flow over a diamond airfoil at $M_\infty = 1.5$, $\alpha = 2°$, and $Re = 1 \times 10^6$. The leading and trailing edge corner angles are both $2\tan^{-1}(.05)$. The computational domain consists of a square farfield boundary 100 chords away from the diamond. An adiabatic no-slip wall boundary conditions is imposed on the airfoil. Shock capturing is again performed using element-based artificial viscosity [42]. The output of interest is the lift coefficient on the airfoil. A hand-generated initial isotropic mesh of 1500 elements is constructed with sufficient boundary-layer resolution to enable a converged RANS solution.

Adaptive simulations are performed at $p = 2$ solution approximation order for three target `dof` costs: 10000, 20000, 40000. At each target `dof`, 10 adaptive iterations are performed, using the final mesh from the previous target `dof` as the starting mesh. As in the previous case, MOESS, Mach-number Hessian, and the neural-network anisotropy prediction methods are compared.

Figure 7 presents the output convergence history obtained from the adaptive runs, where again averaging has been performed over the last 4 iterations at each target `dof`. The reference value is also indicated, obtained from a $p = 3$ run on a uniformly-refined version of the final MOESS adapted mesh.

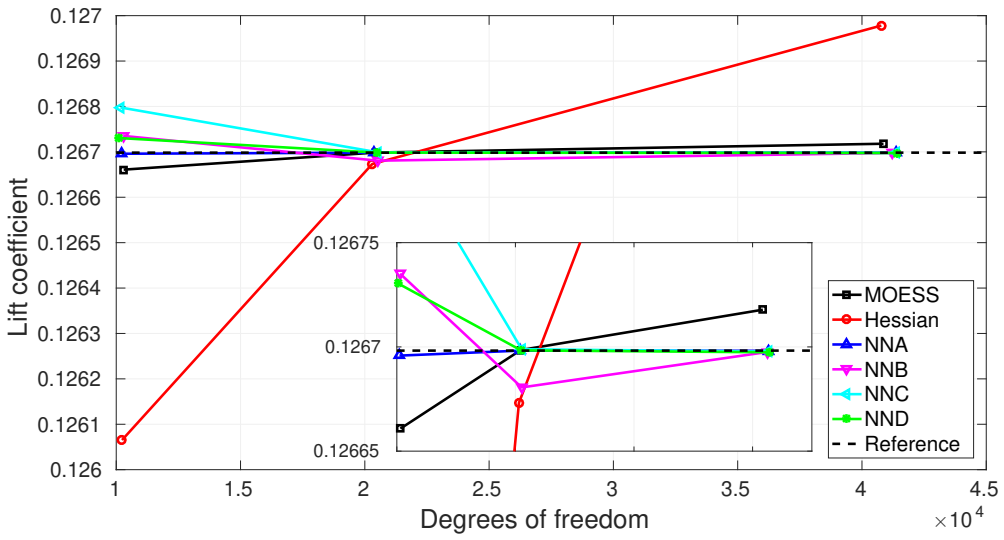
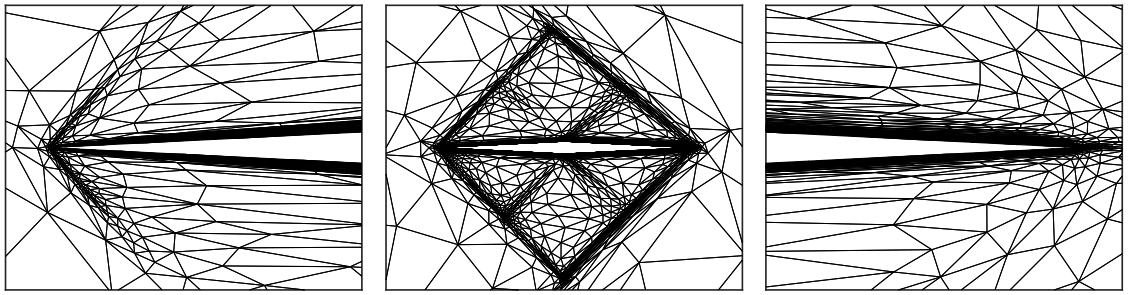


Figure 7: Diamond airfoil, $M_\infty = 1.5$, $\alpha = 2°$, $Re = 1 \times 10^6$: output convergence history.

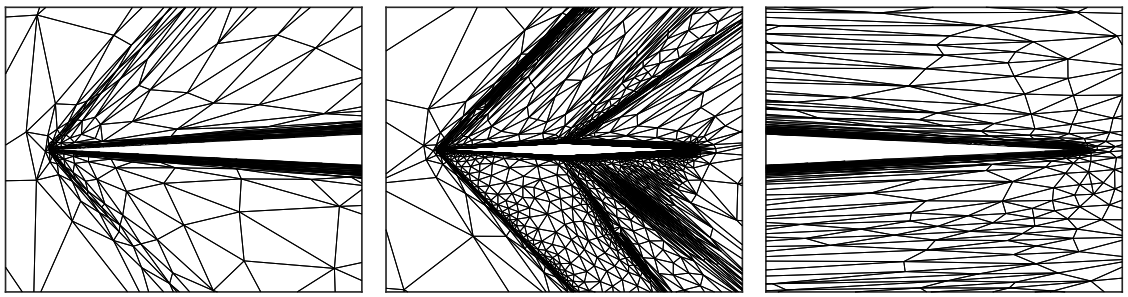
We see that at all target `dof`, the adaptive results using neural-network anisotropy are much closer to the reference value compared to those using the Mach number Hessian anisotropy. In this case, the difference between Hessian-based anisotropy and MOESS is significant, and the machine-learning approaches follow the MOESS results closely. As in the previous case, the accuracy improvement is made possible by a more optimal distribution of degrees of freedom due to correct anisotropy in regions where the output error is more sensitive to resolution in one direction than another. In this case, the performance of MOESS and the neural-network are close in terms of output accuracy versus degrees of freedom, and in fact, on the finest mesh, the neural networks perform better, as shown in the zoomed-in portion of Figure 7. This behavior will be seen in subsequent results, and it is likely due to a regularization effect of the neural networks: their anisotropy predictions are less sensitive to outlier primal and adjoint input features, such as lines of primal or adjoint discontinuity.

Specifically, in an under-resolved setting, where much of the adaptation occurs, errors due to imperfect numerical approximations pollute both the primal and adjoint approximations. This pollution can take the form of inter-element jumps or intra-element oscillations, particularly for high-order. In MOESS, which uses the primal and adjoint solutions locally on each individual element, these errors propagate through the error sampling procedure and limited-data regression to yield potentially noisy metric information. The overall MOESS optimization procedure, which involves averaging to nodes, reduces this noise, but effects of strong singularities, such as shocks (primal) and stagnation streamlines (adjoints) persist in the final sizing and stretching, such that elements may be overly small or skewed relative to an optimal shape based on their actual impact on the functional of interest. That is, the adjoint-weighted residual approximation is susceptible to inaccuracies arising from under-resolution. The better performance of the neural network sizing technique observed in some of the cases can then be attributed to the regularization effect of the network having been trained on many feature-to-metric maps: noisy sizing predictions arising from under-resolution are averaged out during the regression. In addition, the size of the network plays a role in the regularization: the network should not be too large such that it over-fits the training data and reproduces the outlier behavior. Presently, this restriction is favorable as it keeps the network size and associated cost down.

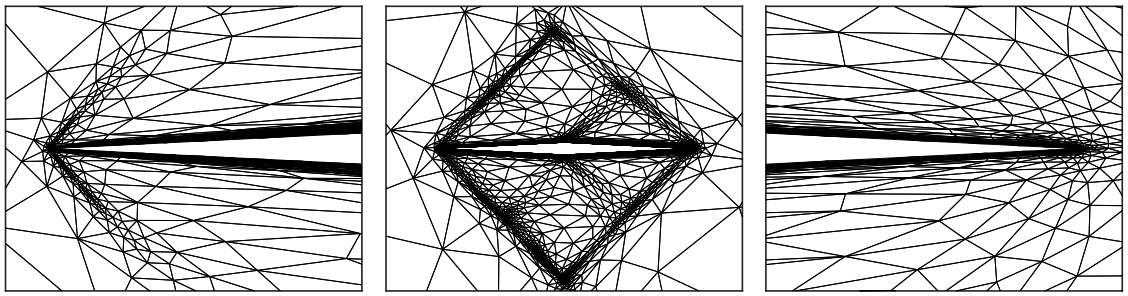
Figure 8 shows the final adapted meshes at the highest target `dof` for the adaptive approaches tested. The neural network adapted meshes are similar, and again only one (A) is given. The flow over the diamond airfoil is supersonic,



(a) MOESS



(b) Hessian



(c) Neural network A

Figure 8: Diamond airfoil: final adapted meshes.

with oblique shocks emanating from the leading and trailing edges, and expansions originating from the mid-chord corners. The Hessian-based primal anisotropy detection approach places shock/expansion-aligned stretched elements in these areas, as expected based on the Mach number variations. In contrast, MOESS places anisotropic elements in not only these areas, but also the corresponding adjoint features, which mimic the primal shock/expansion structure but in reverse (right to left instead of left to right). The result is a primal/adjoint resolving anisotropic mesh, with somewhat higher emphasis evident for the adjoint features. Finally, the neural network anisotropy detection methods, like MOESS, also target the primal and adjoint features, but with a lower degree of anisotropy in the primal and adjoint shocks/discontinuities – the anisotropy remains high in the boundary layer.

We note top/bottom asymmetry of the resolution is expected due to the presence of a nonzero angle of attack. In addition, all methods target the boundary layer with anisotropic elements, although the Hessian-based approach places more emphasis on anisotropy away from the wall near the trailing edge/wake compared to the other two methods. Combined with the excessive resolution of the shocks and expansions due to a primal-only anisotropy measure, particularly on the bottom aft portion of the airfoil, where the predicted anisotropy is nearly orthogonal to the required one, this leads to a less efficient distribution of degrees of freedom for the Hessian approach, and hence larger error for a given target `dof`.

7.3. MDA 30P/30N airfoil

The third test case consists of subcritical RANS flow over the McDonnell Douglas Aerospace (MDA) 30P/30N three-element airfoil at $M_\infty = 0.2$, $\alpha = 5°$, and $Re = 5{\times}10^6$. The computational domain consists of a C-shaped farfield boundary that is 100-200 main-element chords away from the airfoil. An adiabatic no-slip wall boundary conditions is imposed on the airfoil main element, the leading-edge slat, and the trailing-edge flap. The flow is subcritical and hence no shock capturing is employed. The output of interest is the lift coefficient on the airfoil. An Euler-flow adapted initial isotropic mesh of 3000 elements is constructed with sufficient boundary-layer resolution to enable a converged RANS solution.

Adaptive simulations are performed at $p = 2$ solution approximation order for three target `dof` costs: 8000, 16000, 32000. At each target `dof`, 10 adaptive iterations are performed, using the final mesh from the previous target `dof` as the starting mesh. Again, MOESS, Mach-number Hessian anisotropy, and the neural-network anisotropy methods are compared.

Figure 9 presents the output convergence history obtained from the adaptive runs, where averaging has been performed over the last 4 iterations at each target `dof`. The reference value is also indicated, obtained from a $p = 3$ run on the final MOESS adapted mesh, uniformly-refined.

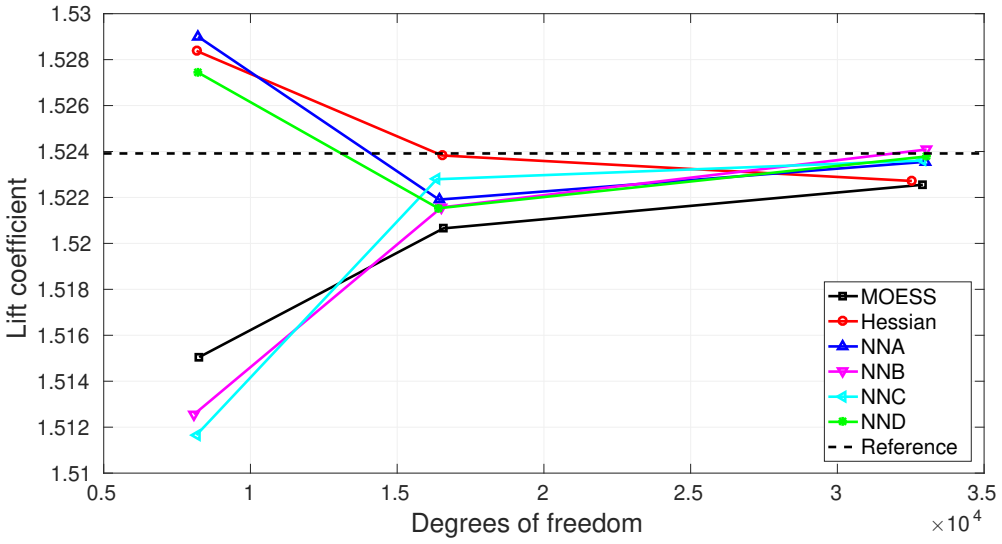
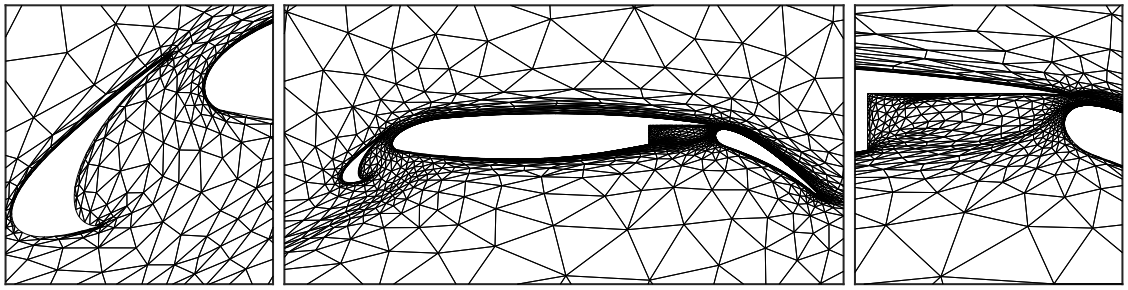


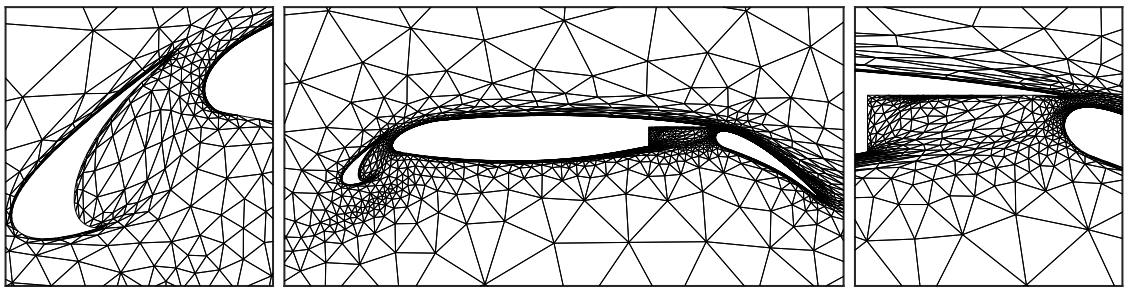
Figure 9: MDA 30P/30N, $M_\infty = 0.2$, $\alpha = 5°$, $Re = 5 \times 10^6$: output convergence history.

We see that the Hessian-based anisotropy detection method yields meshes that have the lift output closer to the reference value compared to MOESS, although the prediction undershoots the reference value on the way to convergence. On the other hand, the neural network approaches all perform better than MOESS by the second `dof` target and hone in closest to the reference value on the finest meshes compared to Hessian and MOESS anisotropy. We note that the neural network results are not identical, and that the output differences on the coarsest `dof` target are large.

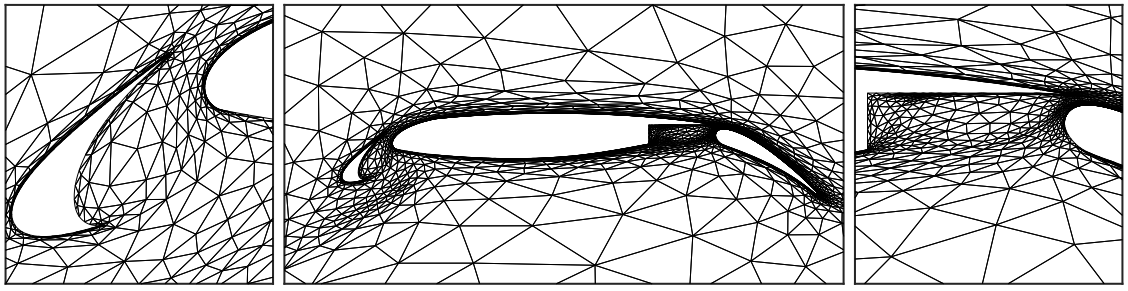
Figure 10 shows the final adapted meshes at the highest target `dof` for the adaptive approaches tested. The most notable differences are in the placement of anisotropy on the leading-edge stagnation streamline and near the lower corner of the main-element cove. Anisotropy on the leading-edge streamline is strongest with MOESS, diminishes slightly with the neural-network method, and is nonexistent for the Hessian approach. This is consistent with the observation that the anisotropy in this region is driven by features of the adjoint, to which the Mach-number Hessian is agnostic. On the other hand, the Hessian approach places anisotropic elements in the region of flow coming off the lower surface of the main element, into the cove. This anisotropy is driven by the edge of the boundary layer, in which the Mach number variation is large. MOESS and the neural network approaches do not fixate on this region, indicating that high element stretching is not required there.



(a) MOESS



(b) Hessian



(c) Neural network A

Figure 10: MDA 30P/30N airfoil: final adapted meshes.

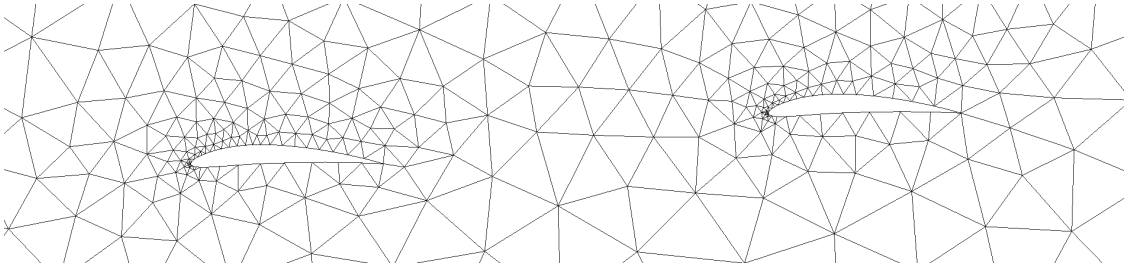
7.4. First extrapolation test: tandem NACA 5410 airfoils at $M_\infty = 0.3$

The previous tests demonstrated the ability of the neural network approaches to correctly detect anisotropy in cases used for training. Given the complexity of the training flowfields and the relatively small sizes of the networks, the successful performance of the networks was by no means guaranteed. Nevertheless, the generalizability of the network must be assessed on an extrapolation case, which is one for which the network was not trained.

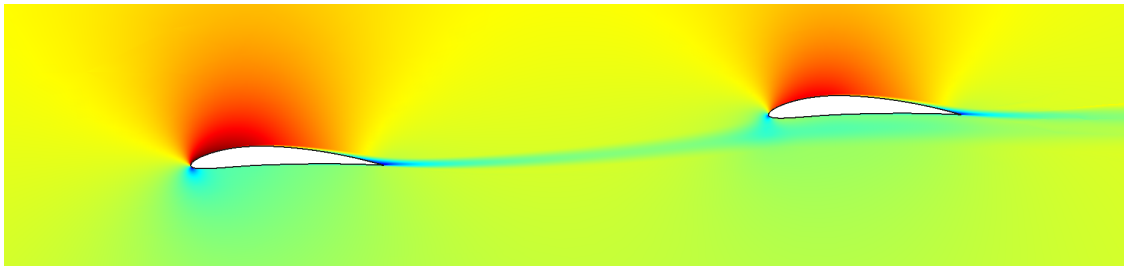
For this test, we choose a completely new geometry: two NACA 5410 airfoils separated by approximately two chord lengths, as shown in Figure 11. The farfield boundary is a square, 100 chords away from the airfoils. The flow conditions are also different from the previous tests: $M_\infty = 0.3$, $\alpha = 5°$, and $Re = 2 \times 10^6$. A hand-generated initial isotropic mesh, shown in Figure 11a, of 1250 elements is constructed with sufficient boundary-layer resolution to enable a converged RANS solution.

7.4.1. Output convergence

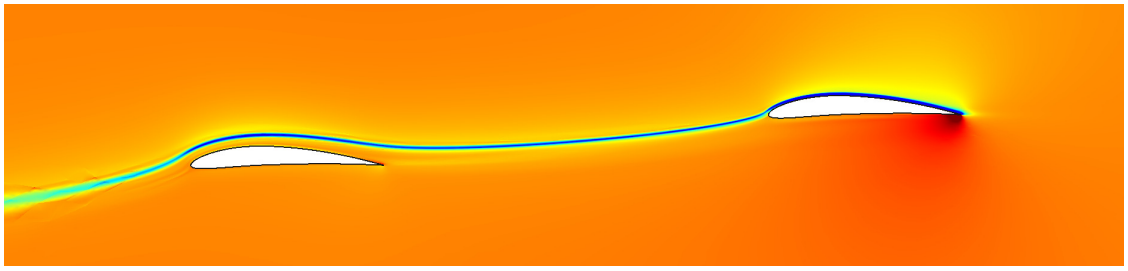
The output of interest is the lift coefficient on the second airfoil, and Figure 11c shows the conservation of x-momentum component of the corresponding adjoint. Note the high variation of the adjoint along the leading-edge stagnation streamline of the second airfoil.



(a) Initial mesh



(b) Mach number



(c) Conservation of x-momentum adjoint

Figure 11: Tandem NACA 5410 airfoils: initial mesh and primal/adjoint solutions.

Adaptive simulations are performed at $p = 2$ solution approximation order for three target `dof` costs: 8000, 16000, 32000. At each target `dof`, 10 adaptive iterations are performed, using the final mesh from the previous target `dof` as the start-

ing mesh. As in the previous cases, we compare MOESS, Mach-number Hessian anisotropy, and the neural-network approaches.

Figure 12 presents the output convergence history obtained from the adaptive runs, where again averaging has been performed over the last 4 iterations at each target `dof`. The reference value is also indicated, obtained from a four-times finer-mesh run at a higher approximation order of $p = 3$.

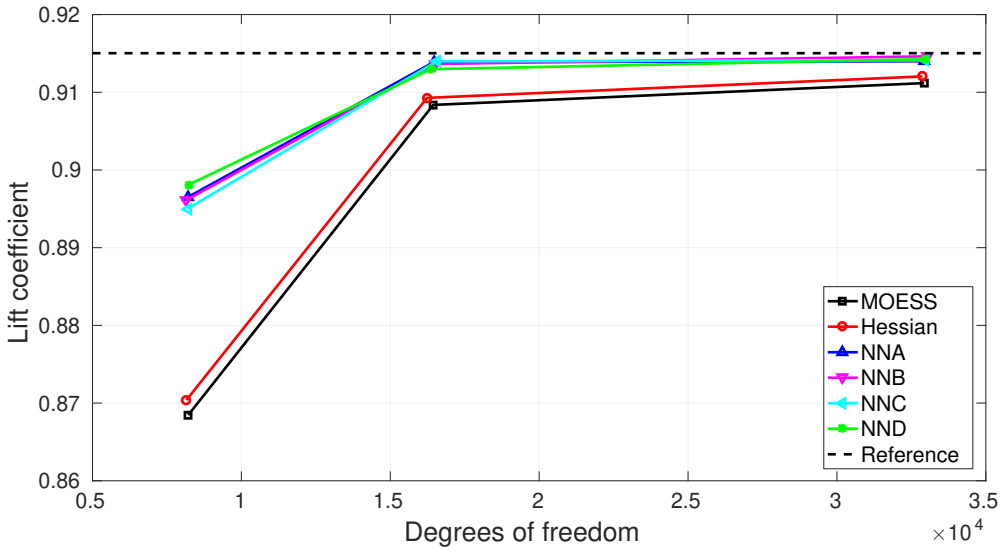


Figure 12: Tandem NACA 5410 airfoils: output convergence history.

We see that at all target `dof`, the adaptive results using the neural-network approaches are closer to the reference value compared to those using MOESS and the Mach number Hessian anisotropy. Indeed, in this case, both MOESS and the Hessian-based methods perform comparably, at least in terms of output convergence, even though the adapted meshes, shown next, are different. The clustering of the neural network approaches indicates similar performance.

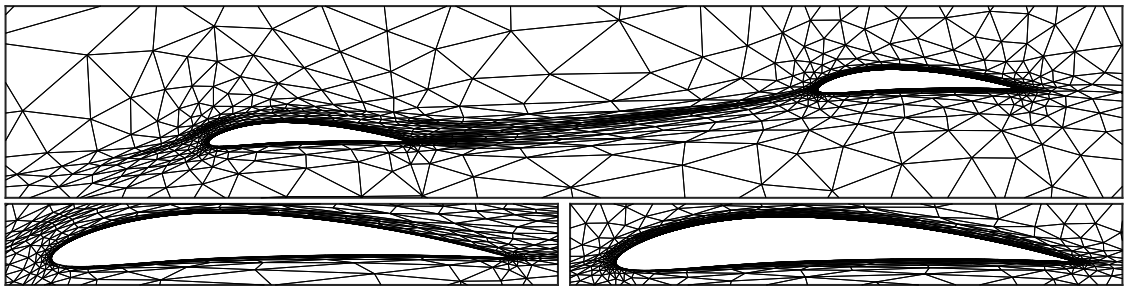
Figure 13 shows the final adapted meshes at the highest target `dof` for several of the adaptive approaches tested. As in the previous tests, the meshes among the approaches differ in their placement of anisotropic elements. The Mach number Hessian approach focuses on the primal anisotropy, in the boundary layer and wake regions. It particularly targets with anisotropy the initial wake of the second airfoil, an area not as much resolved by the other methods. On the other hand, MOESS addresses both the primal and adjoint anisotropy features. MOESS particularly targets the leading-edge stagnation streamline of the aft airfoil, shown in the adjoint contours in Figure 11c, with anisotropic elements. The Hessian-based method places isotropic elements in these regions, with anisotropy between the airfoils driven only by the Mach number variation in the wake of the first airfoil.

Finally, the neural network approaches, which yield similar meshes among themselves, produce anisotropy distributions that are similar to MOESS, but with somewhat lower emphasis on the adjoint anisotropy in the leading-edge stagnation streamline. This is likely due to a regularization property of the neural networks, which are small relative to the amount of training data, and which employ an activation function that saturates. As a result, their predictions are not as sensitive to outlier inputs, such as very high adjoint or primal anisotropy.

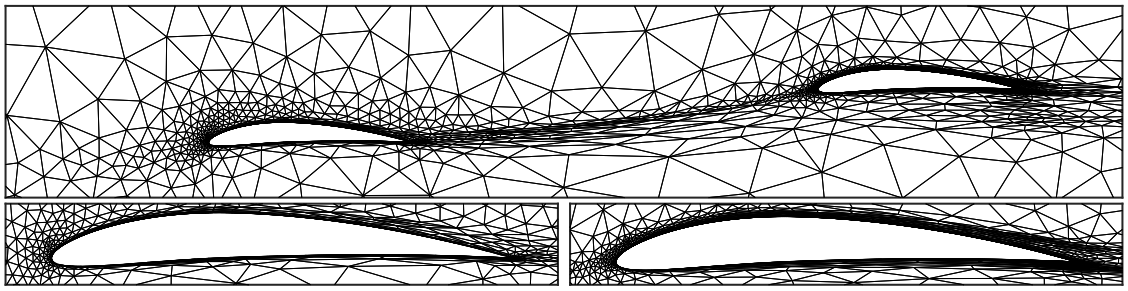
7.4.2. Adaptive iteration comparison

A practical consideration of adaptive methods is their cost. The final meshes produced can be highly optimized, but their generation requires multiple adaptive iterations. Each such iteration requires primal and adjoint solutions, which are relatively inexpensive on coarse meshes but grow in cost as refinement proceeds. Ideally, an adaptive method should not need many solutions on fine meshes before honing in on the optimal mesh. In this section, we compare the various adaptive methods in terms of this cost measure.

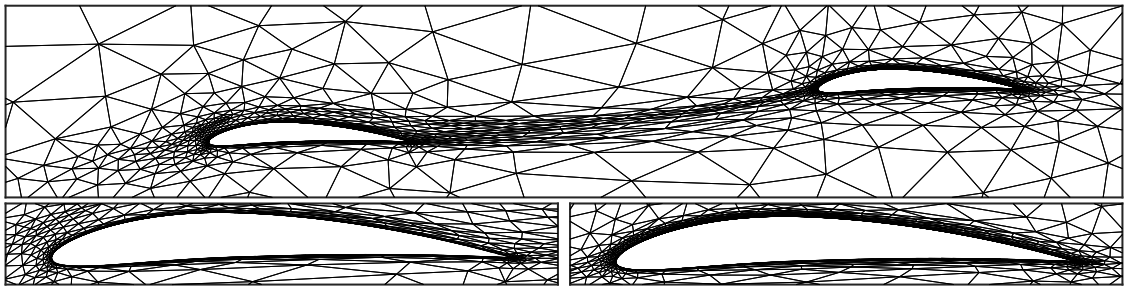
The solver settings, target degrees of freedom, and mesh growth factors are the same for all methods tested. Therefore, the measure of cost reduces to the number of iterations required to attain a certain error level. In the results



(a) MOESS



(b) Hessian



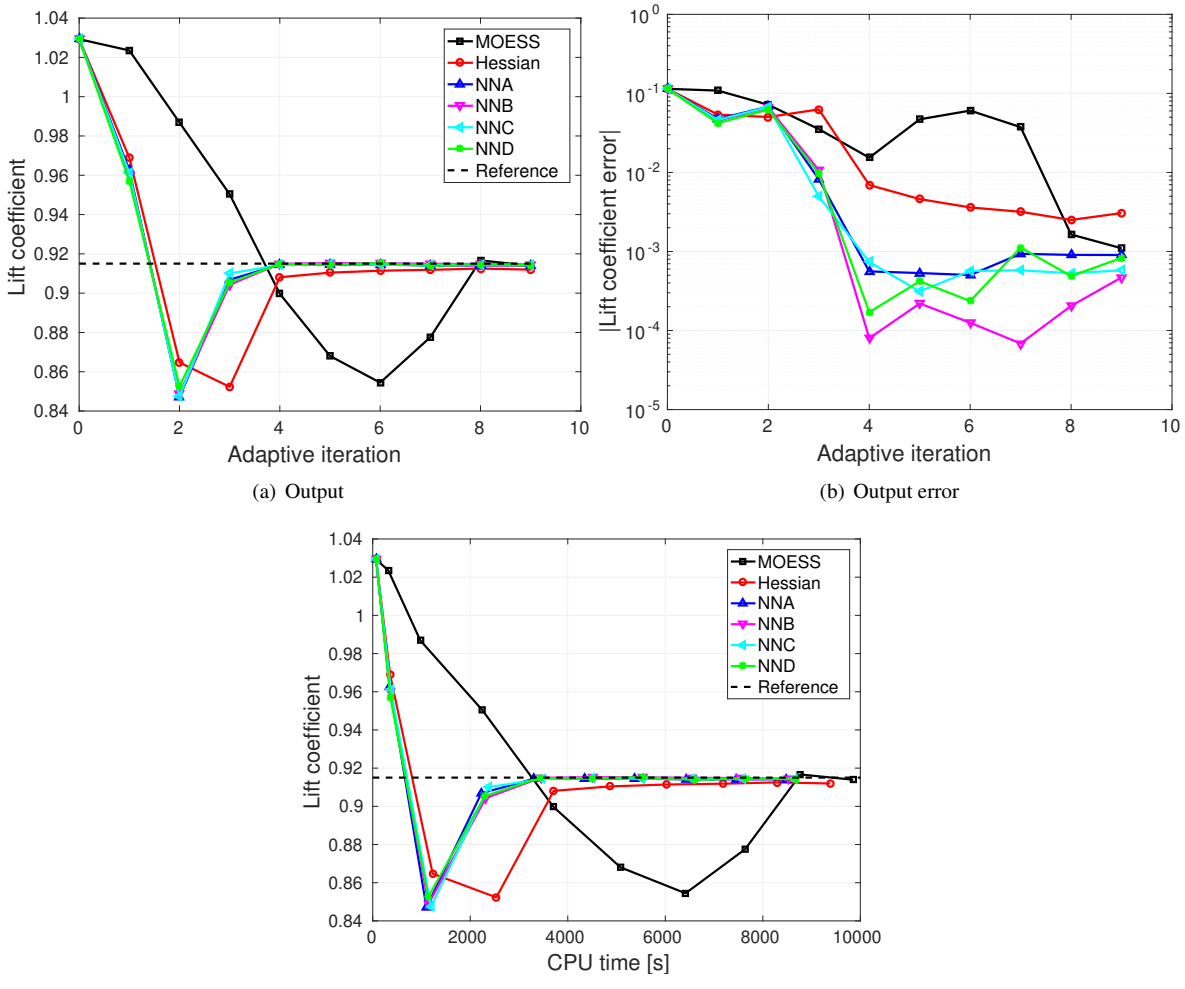
(c) Neural network A

Figure 13: Tandem NACA 5410 airfoils: final adapted meshes.

thus far, 10 iterations were chosen empirically by monitoring outputs and ensuring that they had stabilized by the point of data collection, which was over the last four iterations. Restarts from previous target degrees of freedom provide good starting meshes, which limit the number of extra adaptive iterations needed.

In a production setting, of practical interest is the expense of generating the final adapted solution starting from a coarse initial mesh. We study this cost in terms of the number of adaptive iterations, using the same mesh growth factors for all methods, for the tandem airfoils simulation with the lift on the second airfoil as the output. The starting mesh is the coarse one shown in Figure 11a, which has 7.5k `dof` for $p = 2$, the target mesh size is 32k `dof`, and the growth factor set to 2.0.

Figure 14 shows the convergence of the output with adaptive iterations for all of the methods. Both the actual output and the error relative to a reference/truth solution are shown. The output has not stabilized by the last four iterations for all methods, as this calculation proceeds straight from the initial mesh to the finest `dof` target.



(a) Output

(b) Output error

(c) Output versus CPU time

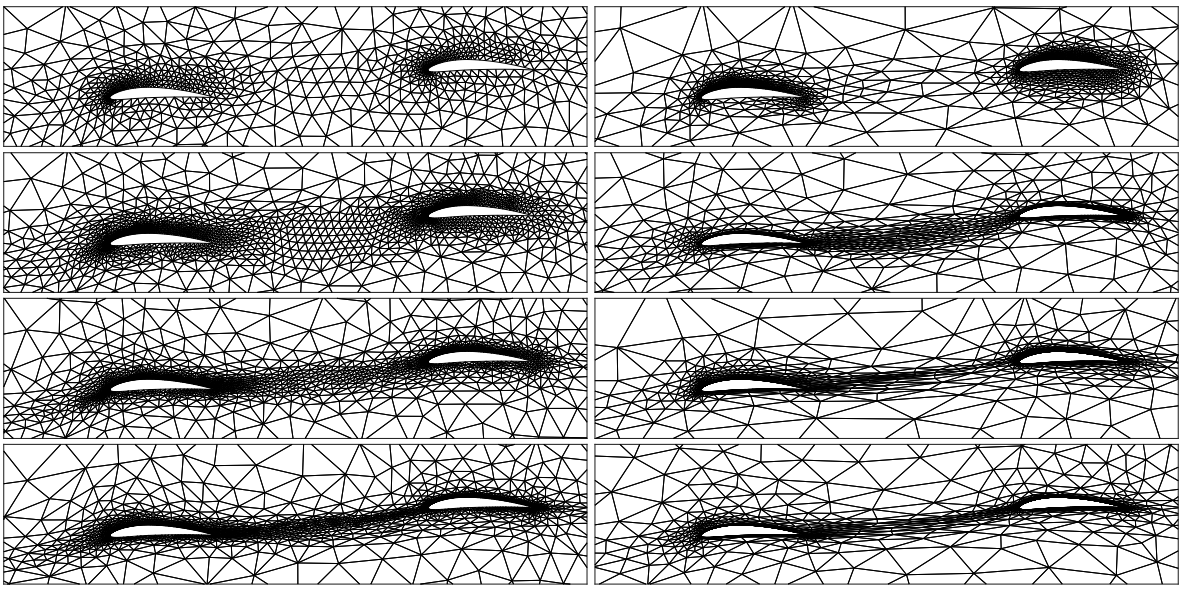
Figure 14: Tandem NACA 5410 airfoils: adaptive iteration and CPU time performance.

From the figure, we see that MOESS exhibits the slowest output convergence. MOESS discovers the error model, specifically the rate tensor that governs anisotropy, through a sampling procedure. This discovery takes multiple adaptive iterations, as the error model is based on asymptotic analysis, and samples are computed relative to an incrementally finer space. On the other hand, both the Hessian and neural-network-based approaches discern the anisotropy from a direct mapping of the primal/adjoint solution features. As soon as the fields are reasonably resolved,

the anisotropy prediction becomes accurate and drives the meshes more quickly to the final optimized mesh. We note, however, that the Hessian-based approach converges to a larger error compared to the neural network approaches, as expected from the data in the previous section. The neural network methods hone in on the optimal meshes quickly, by the fourth adaptive iteration in this test case, minimizing the number of solutions required at the finest `dof` target.

Figure 14 also presents the wall-clock computational (CPU) time required to obtain the solution at each of the adaptive iterations. This includes the primal and adjoint solves, the error estimation, mesh optimization, and adaptation. We see that the relative behavior of the methods resembles the plot versus adaptive iteration. This is because the CPU cost is dominated by the nonlinear primal solve and the linear fine-space adjoint solve, so that given similar `dof` targets, the CPU times per adaptive iteration among the methods will be similar. We note that the costs of the neural network approaches are similar and slightly lower than that of the Hessian approach, which is slightly lower than MOESS. These differences can be largely attributed to an increased stiffness of systems on meshes with non-ideal anisotropy, resulting is more time spent in the primal and adjoint solvers. In addition, MOESS requires slightly more expensive regressions and multiple optimization iterations, and is therefore more expensive. However, for all methods, the costs associated with adaptation are small relative to the primal/adjoint solve times, and hence are not discernible in the CPU plot.

Figure 15 compares the first four adapted meshes between MOESS and the neural network A approach. The side-by side meshes in each row are at similar `dof`. We see that MOESS discovery of anisotropy proceeds more slowly, with largely isotropic elements in the joining wake until the fourth adaptive iteration. On the other hand, anisotropic elements are already present in the neural network meshes by the second adaptive iteration. Combined with the higher resolution in the vicinity of the airfoils, the result is faster iterative convergence to the optimal mesh.



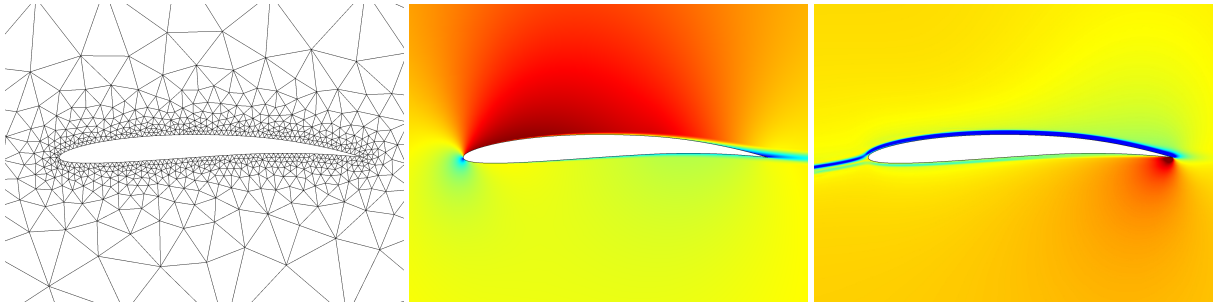
(a) MOESS (b) Neural network A

Figure 15: Tandem NACA 5410 airfoils: meshes at adaptive iterations 1 (top) through 4 (bottom).

7.5. Second extrapolation test: moment prediction on a NACA 4608 airfoil at $M_\infty = 0.6$

As a second extrapolation test, we run a case with not only a different geometry, the NACA 4608 airfoil, but also using an output that was not included in the training set. Whereas only lift and drag outputs were considered during training, the output of interest in this test is the moment coefficient, measured about the leading edge. Since the choice of output enters the network only through features of the adjoint, the networks should generalize to different outputs, as long as features of the adjoint in the extrapolation case are not radically different from those observed in the training. The purpose of this example is to test this claim.

The geometry for this test is a NACA 4608 airfoil in a flow at $M_\infty = 0.6$, $Re = 2 \times 10^6$, and $\alpha = 2°$. The farfield is a square, 100 chords away from the airfoils. Figure 16 shows the initial mesh of 1419 elements and primal and adjoint solutions on a refined mesh.



(a) Initial mesh (b) Mach number (range 0 to 0.9) (c) Conservation of x-momentum adjoint

Figure 16: NACA 4608: initial mesh and primal/adjoint solutions.

The output of interest is the moment coefficient on the airfoil, computed about the z-axis coming out of the page through the leading-edge of the airfoil. As in the previous cases, adaptive simulations are performed at $p = 2$ solution approximation order for three target `dof` costs: $8000, 16000, 32000$. At each target `dof`, 10 adaptive iterations are performed with each method, using the final mesh from the previous target `dof` as the starting mesh.

Figure 17a presents the output convergence history obtained from the adaptive runs, where again averaging has been performed over the last 4 iterations at each target `dof`. The reference value is also indicated, obtained from a four-times finer-mesh run at a higher approximation order of $p = 3$.

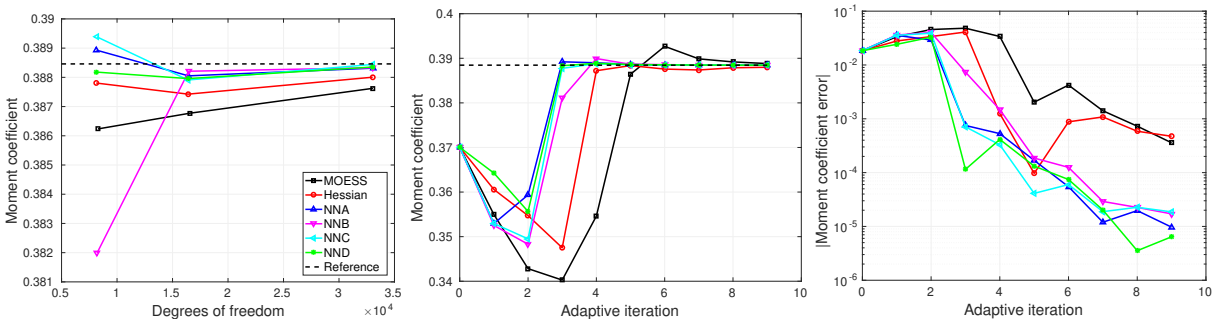
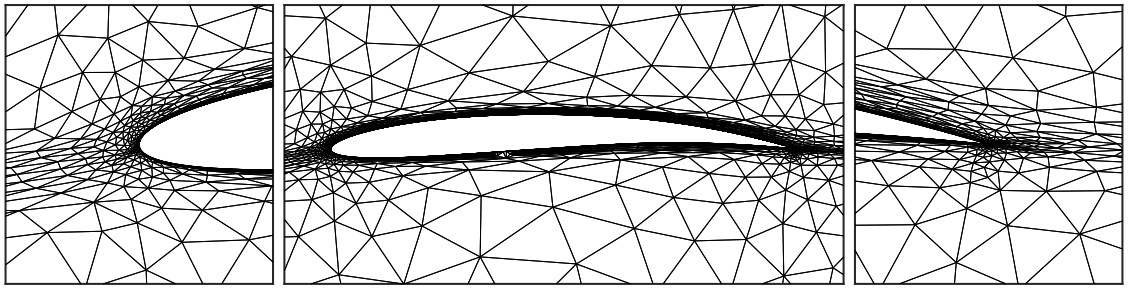


Figure 17: NACA 4608: output convergence history and adaptive iteration performance.

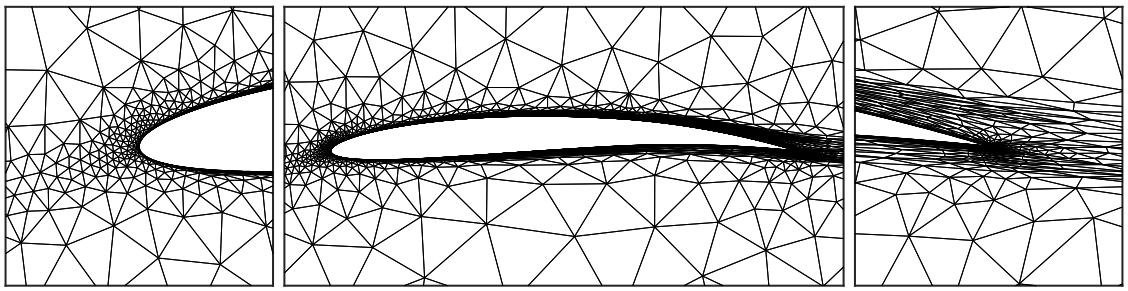
We see that at all but the coarsest target `dof`, the adaptive results using the neural-network approaches are closer to the reference value compared to those using MOESS and the Mach number Hessian anisotropy. In this case, the Hessian adaptation actually outperforms MOESS for the degrees of freedom tested. This initially seems counter-intuitive, as the networks were trained using MOESS. However, the reason for the better performance of the networks remains the same as discussed in Section 7.2: lower propensity of the networks to produce outlier sizing outputs compared to MOESS, which operates on limited local data and can be sensitive to noisy primal and adjoint solutions. This is another example of the benefit of regularization effect of the small network. Finally, clustering of the network approaches indicates similar performance.

Figure 17 also shows the adaptive iteration performance of the different approaches, starting with the coarsest mesh and ramping up to a target size of 32k `dof` with a growth factor of 2.0. We see that the networks are again the quickest to converge to the reference output, requiring the fewest adaptive iterations before the output stabilizes. The error plot in Figure 17c clarifies the differences in the latter adaptive iterations, showing that whereas the Hessian-based approach gets close to the output rapidly initially, in the latter adaptive iterations the error rises and is on par with MOESS.

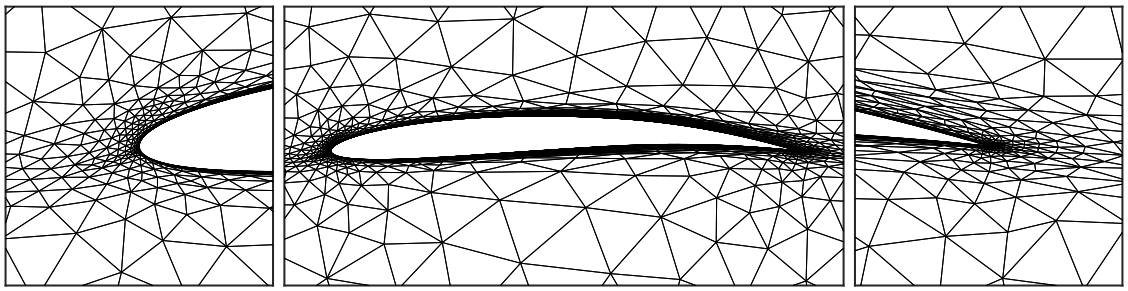
Figure 18 shows the final adapted meshes at the highest target `dof` for the adaptive approaches tested (only A out of the four networks). The regions targeted for refinement are similar among the methods, but the anisotropy predictions differ. Adjoint anisotropy heavily influences MOESS, the Hessian approach only picks up the Mach number anisotropy, and the neural network approach visually falls somewhere between these two, closer to MOESS, particularly near the trailing edge.



(a) MOESS



(b) Hessian



(c) Neural network A

Figure 18: NACA 4608 airfoil : final adapted meshes.

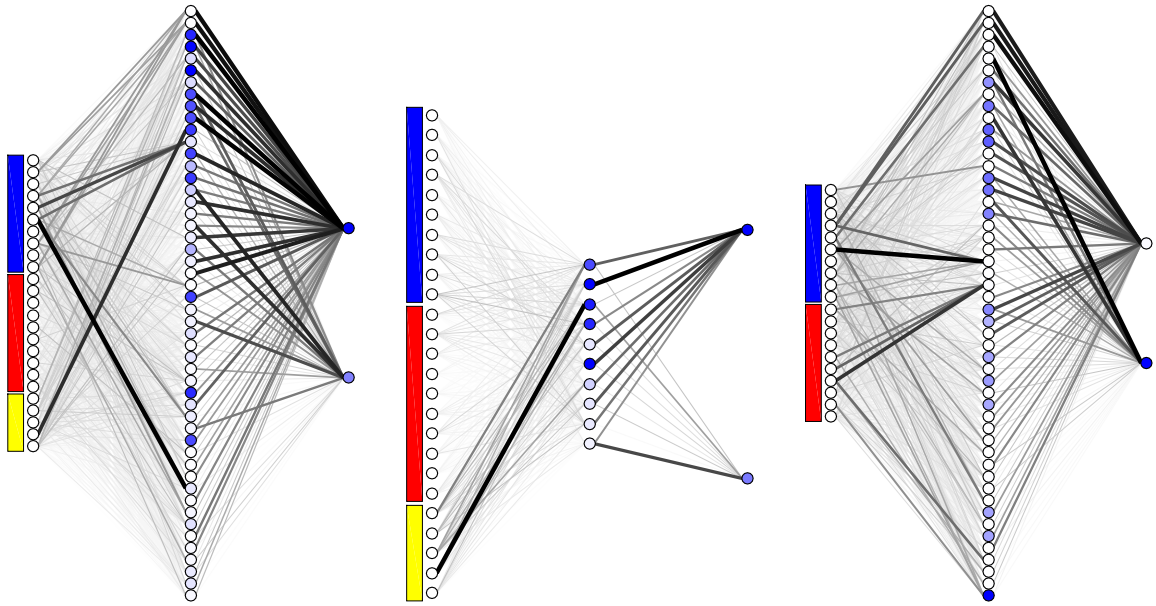
7.6. $p = 3$ Results

All results so far were obtained using an approximation order of $p = 2$ for both training and deployment simulations. In this section we present, as an extension, a subset of corresponding $p = 3$ results. Recall that for $p > 2$, the state and adjoint fields are first least-squares projected to $p = 2$ approximation order within each element, in order to obtain constant elemental Hessian matrices. This occurs during training and deployment feature calculations. The relative error indicators remain unaltered from the standard adjoint-weighted residuals, computed at order $p + 1$.

7.6.1. Network Training

The same flow cases that were used for training at $p = 2$, described in Section 6, are also used at $p = 3$. The training algorithm, including the optimizer, batch size, learning rate, and number of iterations, also remains the same.

The resulting training loss history is similar to that shown in Figure 3. Figure 19 illustrates three out of the four trained networks, A, C, D. Note that the number of inputs, outputs, and hidden layer sizes does not change with p.



(a) Network A (b) Network C (c) Network D

Figure 19: Visualization of the neural networks for $p = 3$. The format is the same as the $p = 2$ version in Figure 4.

Based on the connecting weights, we see that in network A, the baseline, the primal and error indicator features contribute relatively more to the output prediction than the adjoint features. In network C, which has the small hidden layer, the error indicator features become more important. Finally, in network D, which does not use the error indicators, the contributions from the primal and adjoint features are similar.

7.6.2. Tandem NACA 5410 airfoil test

We present one $p = 3$ result, the same extrapolation test case as in Section 7.4. The flow conditions, starting mesh, output of interest, and `dof` targets remain unchanged. Figure 20 shows the convergence of the output, the lift coefficient on the aft airfoil averaged over the last four adaptive iterations, with degrees of freedom. The reference value is computed at $p = 4$ on a uniformly-refined version of the finest MOESS-adapted mesh.

We see that the neural network results are similar and that they yield outputs closer to the reference result for all `dof` targets. Both MOESS and the Hessian-based methods perform comparably, even though the adapted meshes, are different.

Figure 21 shows the final adapted meshes at the highest target `dof`. These are coarser than the $p = 2$ ones in Figure 13, due to the higher `dof` per element for $p = 3$. We see similar differences in the anisotropy placement. First, MOESS heavily targets the stagnation streamline for the aft airfoil, with high aspect ratio elements. The neural network mesh exhibits less anisotropy in this area, and the Hessian mesh places isotropic elements there due to a lack of anisotropy in the Mach number ahead of the airfoil. At the trailing edge of the aft airfoil, the Hessian-based mesh exhibits heavy anisotropic refinement, due to Mach number variations in the boundary layer and wake. On the other hand, the meshes from MOESS and the neural network show less refinement due to lower anisotropy. Overall, as for $p = 2$, the neural network approaches, which again yield similar meshes among themselves, produce anisotropy distributions that are similar to MOESS, but with lower emphasis on the adjoint anisotropy features.

As in Section 7.4.2, we also test the rate at which the different methods converge to the final meshes for $p = 3$. The initial mesh has 12.5k `dof` for $p = 3$, and the target mesh size is 32k `dof`, with a growth factor of 2.0. Figure 22 shows the convergence of the aft airfoil lift output with adaptive iterations. We again see a more rapid convergence of the

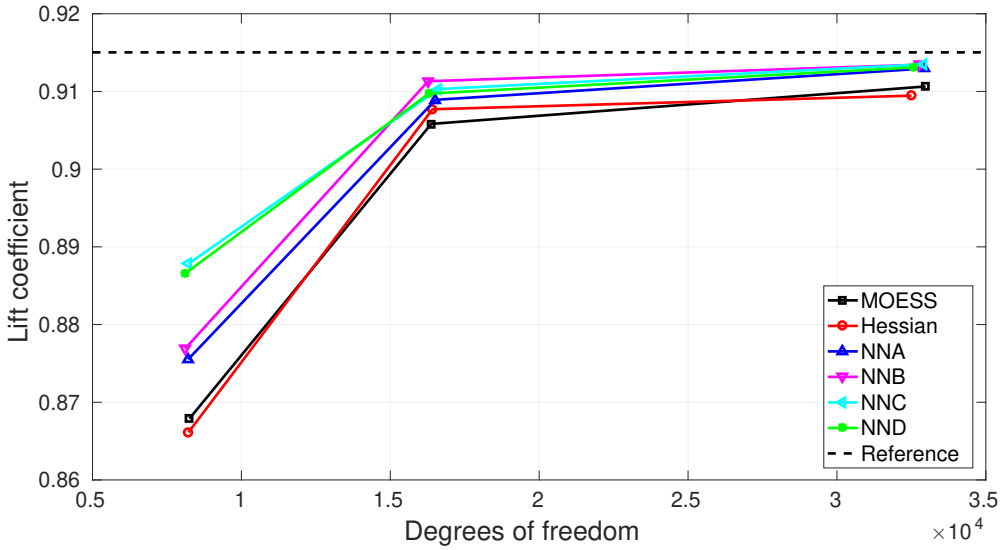


Figure 20: Tandem NACA 5410 airfoils: $p = 3$ output convergence history.

output with the neural network methods compared to MOESS, and we attribute this to the direct feature-to-anisotropy map in the neural-network methods, as opposed to the error model discovery of MOESS.

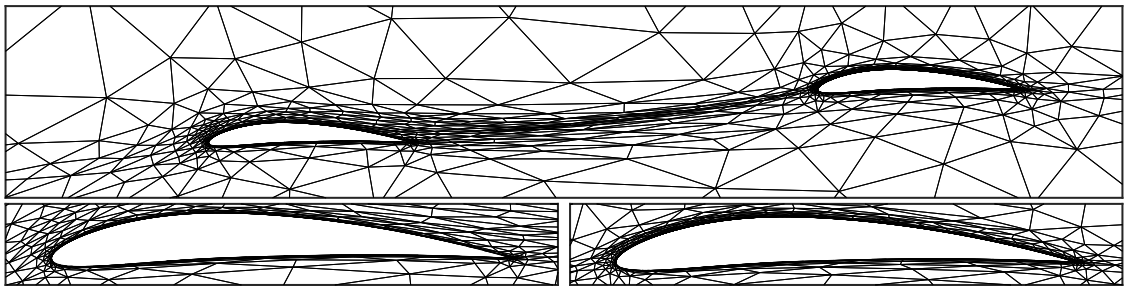
Figure 23 shows a comparison of the adapted meshes for MOESS and the neural network A approach. As in the case of $p = 2$, we see that MOESS discovery of anisotropy proceeds more slowly, with largely isotropic elements in the joining wake until the fourth adaptive iteration. In contrast, anisotropic elements are present in the neural network meshes by the second adaptive iteration, resulting in faster iterative convergence to the optimal mesh.

8. Conclusions

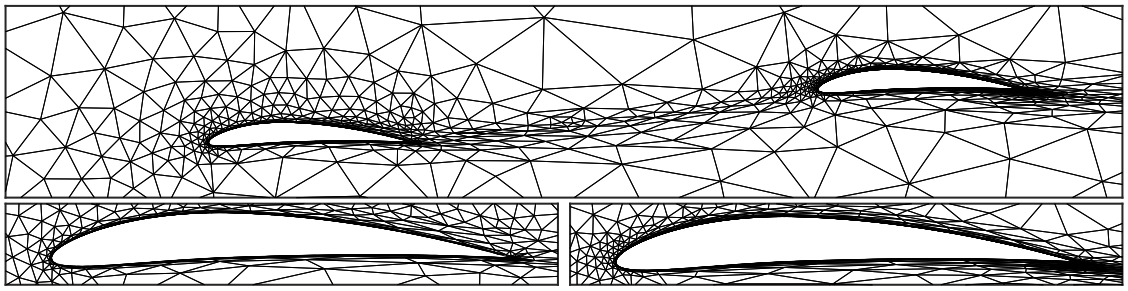
This paper introduces a machine-learning approach for determining the optimal anisotropy in a mesh, in the context of an output-based adaptive solution procedure. An artificial neural network is used to predict the desired element stretching ratio and direction from features of the primal and adjoint solutions. The network is trained to reproduce anisotropy calculated by MOESS, using features that are more easily calculated than the mesh-refinement sampling procedure of MOESS. These features consist of aspect ratio and direction data computed from the Hessian of each of the primal and adjoint variables, and from normalized error indicators of each of the state components. The features are invariant to scaling of the equations or outputs, and hence generalizable across different flow conditions, outputs, and unit choices.

A variety of RANS cases, including transonic and supersonic flows, provides training data for a set of four neural networks, which consists of a simple structure with one or two hidden layers and an output layer of two neurons. Training of the networks yields a reduction in the mean-squared loss of about one order of magnitude, where the loss is measured over two independent components of the non-dimensional step matrix, which is the logarithm of the normalized output metric.

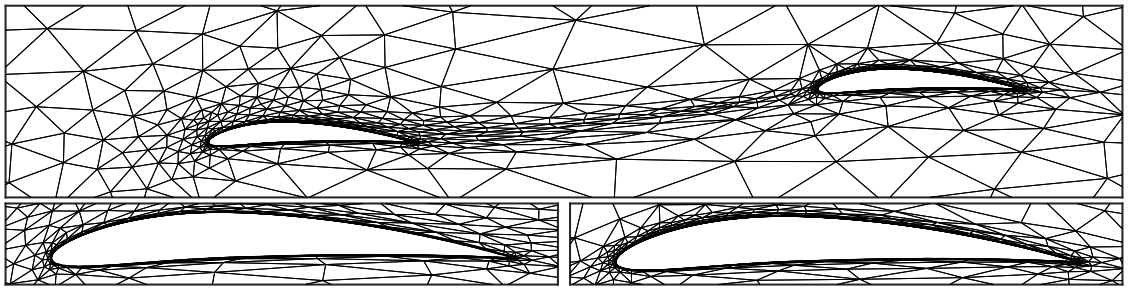
The test cases compare the proposed neural-network anisotropy detection approach to the Mach number Hessian anistotropy detection and to MOESS. When used to drive adaptation, the neural-network approaches yields output errors that more closely follow the MOESS results, and which are generally more accurate than the Hessian anisotropy approach. Among the four networks tested, the output convergence histories and resulting meshes were similar. When the error indicator was not available, in network D, the network was still able to predict the proper anisotropy information using just the primal and adjoint Hessians. Furthermore, network C, which a small hidden layer of size just twice the state rank, performed comparably to the other networks, which suggests that the prediction of anisotropy from the primal and adjoint Hessian features is not an overly complex function. Indeed, given a limited number of inputs and outputs, a small network can be expected to perform well. However, this performance is not guaranteed



(a) MOESS

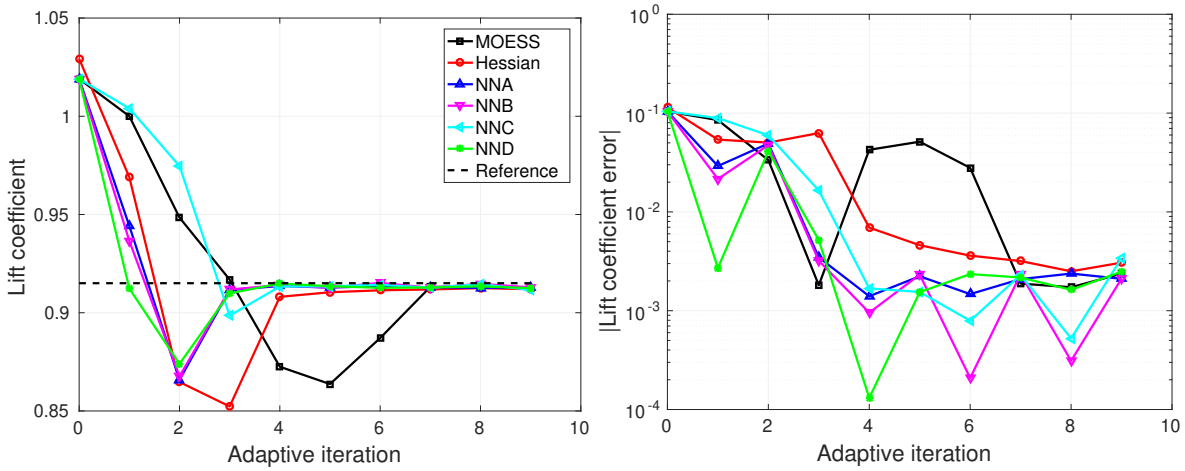


(b) Hessian



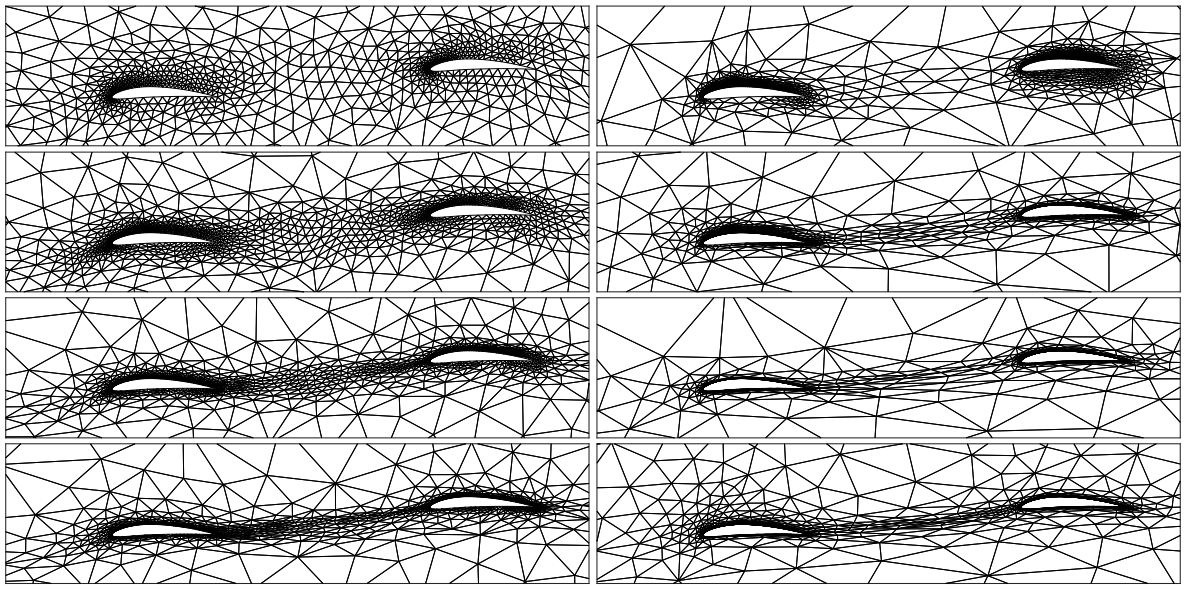
(c) Neural network A

Figure 21: Tandem NACA 5410 airfoils: $p = 3$ final adapted meshes.



(a) Output (b) Output error

Figure 22: Tandem NACA 5410 airfoils: $p = 3$ adaptive iteration performance.



(a) MOESS (b) Neural network A

Figure 23: Tandem NACA 5410 airfoils: $p = 3$ meshes at adaptive iterations 1 (top) through 4 (bottom).

and is subject to the proper choice of inputs and outputs. Choosing irrelevant features or outputs that do not smoothly map to the desired quantity of interest, an optimal metric in the present study, can severely deteriorate the network performance. The proper choice of inputs and parametrization of the output, as described in this study, are critical to enabling a useful sizing prediction.

In many of the cases tested, the neural networks yielded outputs that, for a given degree-of-freedom target, were more accurate than MOESS. This is in spite of the fact that the networks were trained with MOESS data. The explanation for this is that the networks possess a regularization property: the networks are small relative to the amount of training data and employ an activation function that saturates. As a result, their predictions are not as sensitive to outlier inputs, such as very high adjoint or primal anisotropy. This generally results in lower aspect ratios in regions of large primal or adjoint variation.

Based on the analysis of the adapted meshes, we observe that MOESS appears to over-emphasize regions of adjoint anisotropy, such as leading-edge stagnation streamlines, whereas the Mach number Hessian-based approach appears to over-emphasize regions of primal anisotropy, such as boundary layers and wakes. Although the element sizing information is driven by a common error estimate, a sub-optimal anisotropy prediction will yield inefficient flowfield resolution in these regions, requiring more mesh elements, or equivalently higher error for a fixed degrees of freedom. The neural network approaches, being less sensitive to sharp primal/adjoint features, can then perform better than either MOESS or the Hessian approach.

Furthermore, the neural network methods can more rapidly converge to the final adapted meshes compared to MOESS. Whereas MOESS discovers the error model, specifically the rate tensor that governs anisotropy, through a sampling procedure, which can take multiple adaptive iterations, both the Hessian and neural-network-based approaches discern the anisotropy from a direct mapping of the primal/adjoint solution features. When these fields are reasonably resolved, the anisotropy prediction becomes accurate and drives the meshes more quickly to the final optimized mesh. These final meshes are more accurate for the neural network methods compared to the Hessian approach.

In addition to accuracy, computational cost is a relevant metric for comparing adaptive methods. The online CPU time of the MOESS metric calculation exceeds that of both the Hessian and network approaches, although all of the costs associated with adaptation are low compared to the computational time required to obtain the primal and adjoint solutions. An additional cost associated with the networks is the offline training cost. The most time consuming portion of this training is running a representative set of samples of adaptation, in this case MOESS, to generate the feature-to-metric maps. This is not an excessive cost, however, and in the present study, only nine cases were considered, with 5-10 adaptive iterations per case. As every element from a mesh yields one data point, the complete data set is rich even for the small number of cases. The actual training of the weights and biases is much cheaper than the data collection. Furthermore, just as in model reduction, this offline cost is amortized over many online runs. In a many-query setting, such as optimization or uncertainty quantification, the offline cost associated with each particular evaluation can be made arbitrarily small. Finally, the networks are independent and portable, meaning that once generated, they can be used without MOESS, e.g. in other codes that have adjoint capability and Hessian-based meshing.

The results in this paper show that the neural-network models not only reproduce the correct anisotropy for cases in the training data set, but that they also generalize to flowfields not considered during training. The performance of the networks on these extrapolation tests is similar to the other test cases. In addition, the networks are not tied to a specific output, as they are trained on normalized adjoint features. We note that the choice of output affects the overall result, but only through the features of the adjoint, which are used as inputs into the network. Specifically, just as the structure of the MOESS algorithm does not change as the output changes, the structure of the neural network is also not restricted to a particular output. Changing the output will change the adjoint, which will change the result of the network, but the network is meant to be a general map from primal and adjoint features to element sizing information. A caveat is that the outputs used during training need to excite a set of primal and adjoint features that is representative of those to be observed in the deployment. In particular, if an output considered in deployment creates adjoint features not present in training data, such as discontinuities or strong anisotropy, the neural network will likely not yield optimal element sizing information. This observation highlights the importance of using a diverse set of flows and outputs during training. As shown in the example in Section 7.5, extrapolation of the method to an unseen output is possible, and the networks continue to perform well.

Finally, the methods were demonstrated for both $p = 2$ and $p = 3$ approximation orders, where the $p = 3$ features

were computed after projection to $p = 2$. Natural extensions of this work include three-dimensional flow tests and analysis of even more varied output predictions. We do not expect fundamental challenges in the extension to three dimensions, as the Hessian features that serve as network inputs are well defined, and MOESS and Hessian-based adaptation have both been demonstrated in three dimensions, so that training data can be created. The size of the networks will increase, but as these are currently very small, the impact of the online evaluation on the cost of the adaptive solution process is expected to remain negligible.

References

References

[1] K. J. Fidkowski, D. L. Darmofal, Review of output-based error estimation and mesh adaptation in computational fluid dynamics, AIAA Journal 49 (4) (2011) 673–694. doi:10.2514/1.J050073.

[2] M. J. Castro-Diaz, F. Hecht, B. Mohammadi, O. Pironneau, Anisotropic unstructured mesh adaptation for flow simulations, International Journal for Numerical Methods in Fluids 25 (1997) 475–491.

[3] G. C. Buscaglia, E. A. Dari, Anisotropic mesh optimization and its application in adaptivity, International Journal for Numerical Methods in Engineering 40 (22) (1997) 4119–4136.

[4] W. G. Habashi, J. Dompierre, Y. Bourgault, D. Ait-Ali-Yahia, M. Fortin, M.-G. Vallet, Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. Part I: general principles, International Journal for Numerical Methods in Fluids 32 (2000) 725–744.

[5] G. Xia, D. Li, C. L. Merkle, Anisotropic grid adaptation on unstructured meshes, AIAA Paper 2001-0443 (2001).

[6] D. A. Venditti, D. L. Darmofal, Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows, Journal of Computational Physics 187 (1) (2003) 22–46.

[7] M. A. Park, Three–dimensional turbulent RANS adjoint–based error correction, AIAA Paper 2003-3849 (2003).

[8] E. Schall, D. Leservoisier, A. Dervieux, B. Koobus, Mesh adaptation as a tool for certified computational aerodynamics, International Journal for Numerical Methods in Fluids 45 (2) (2004) 179–196.

[9] L. Formaggia, S. Perotto, P. Zunino, An anisotropic a posteriori error estimate for a convection-diffusion problem, Computing and Visualization in Science 4 (2001) 99–104.

[10] K. J. Fidkowski, D. L. Darmofal, A triangular cut-cell adaptive method for high-order discretizations of the compressible Navier-Stokes equations, Journal of Computational Physics 225 (2007) 1653–1672. doi:10.1016/j.jcp.2007.02.007.

[11] M. Yano, J. Modisette, D. Darmofal, The importance of mesh adaptation for higher-order discretizations of aerodynamics flows, AIAA Paper 2011-3852 (2011).

[12] M. Yano, D. Darmofal, An optimization-based framework for anisotropic simplex mesh adaptation, Journal of Computational Physics 231 (22) (2012) 7626–7649. doi:10.1016/j.jcp.2012.06.040.

[13] K. J. Fidkowski, A local sampling approach to anisotropic metric-based mesh optimization, AIAA Paper 2016–0835 (2016). doi:10.2514/6.2016-0835.

[14] P. Werbos, Beyond regression: New tools for prediction and analysis in the behavioral science, Ph.D. thesis, Harvard University (1974).

[15] Y. LeCun, Y. Bengio, G. Hinton, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.

[16] A. Khosravi, S. Nahavandi, D. Creighton, A. F. Atiya, Comprehensive review of neural network-based prediction intervals and new advances, IEEE Transactions on Neural Networks 22 (9) (2011) 1341–1356. doi:10.1109/TNN.2011.2162110.

[17] S. Shanmuganathan, S. Samarasinghe, Artificial Neural Network Modelling, Springer International Publishing, Switzerland, 2016. doi:10.1007/978-3-319-28495-8.

[18] W. E. Faller, S. J. Schreck, Unsteady fluid mechanics applications of neural networks, Journal of Aircraft 34 (1) (1997) 48–55. doi:10.2514/2.2134.

[19] R. Huang, H. Hu, Y. Zhao, Nonlinear reduced-order modeling for multiple-input/multiple-output aerodynamic systems, AIAA Journal 52 (6) (2014) 1219–1231. doi:10.2514/1.J052323.

[20] M. Milano, P. Koumoutsakos, Neural network modeling for near wall turbulent flow, Journal of Computational Physics 182 (1) (2002) 1–26. doi:10.1006/jcph.2002.7146.

[21] N. Smaoui, S. Al-Enezi, Modelling the dynamics of nonlinear partial differential equations using neural networks, Journal of Computational and Applied Mathematics 170 (1) (2004) 27–58. doi:10.1016/j.cam.2003.12.045.

[22] L. Manevitz, A. Bitar, D. Givoli, Neural network time series forecasting of finite-element mesh adaptation, Neurocomputing 63 (2005) 447–463. doi:10.1016/j.neucom.2004.06.009.

[23] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, Journal of Fluid Mechanics 807 (2016) 155–166. doi:10.1017/jfm.2016.615.

[24] A. P. Singh, S. Medida, K. Duraisamy, Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils, AIAA Journal 55 (7) (2017) 2215–2227. doi:10.2514/1.j055595.

[25] S. Pan, K. Duraisamy, Long-time predictive modeling of nonlinear dynamical systems using neural networks, Complexity 2018 (2018) 1–26. doi:10.1155/2018/4801012.

[26] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics 378 (2019) 686–707. doi:10.1016/j.jcp.2018.10.045.

[27] W. Reed, T. Hill, Triangular mesh methods for the neutron transport equation, Los Alamos Scientific Laboratory Technical Report LA-UR-73-479 (1973).

[28] B. Cockburn, C.-W. Shu, Runge-Kutta discontinuous Galerkin methods for convection-dominated problems, Journal of Scientific Computing 16 (3) (2001) 173–261.
[29] K. J. Fidkowski, T. A. Oliver, J. Lu, D. L. Darmofal, *p*-Multigrid solution of high–order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations, Journal of Computational Physics 207 (2005) 92–113. doi:10.1016/j.jcp.2005.01.005.
[30] S. Allmaras, F. Johnson, P. Spalart, Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model, Seventh International Conference on Computational Fluid Dynamics (ICCFD7) 1902 (2012).
[31] M. A. Ceze, K. J. Fidkowski, High-order output-based adaptive simulations of turbulent flow in two dimensions, AIAA Paper 2015–1532 (2015). doi:10.2514/6.2015-1532.
[32] P. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, Journal of Computational Physics 43 (1981) 357–372.
[33] F. Bassi, S. Rebay, Numerical evaluation of two discontinuous Galerkin methods for the compressible Navier-Stokes, equations, International Journal for Numerical Methods in Fluids 40 (2002) 197–207.
[34] R. Becker, R. Rannacher, An optimal control approach to a posteriori error estimation in finite element methods, in: A. Iserles (Ed.), Acta Numerica, Cambridge University Press, 2001, pp. 1–102.
[35] K. J. Fidkowski, High-order output-based adaptive methods for steady and unsteady aerodynamics, in: H. Deconinck, R. Abgrall (Eds.), 37$^{\text{th}}$ Advanced CFD Lectures series; Von Karman Institute for Fluid Dynamics (December 9–12 2013), von Karman Institute for Fluid Dynamics, 2013.
[36] J. Peraire, M. Vahdati, K. Morgan, O. C. Zienkiewicz, Adaptive remeshing for compressible flow computations, Journal of Computational Physics 72 (1987) 449–466.
[37] M. Yano, An optimization framework for adaptive higher-order discretizations of partial differential equations on anisotropic simplex meshes, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts (2012).
[38] H. Borouchaki, P. George, F. Hecht, P. Laug, E. Saltel, Mailleur bidimensionnel de Delaunay gouverné par une carte de métriques. Partie I: Algorithmes, INRIA-Rocquencourt, France. Tech Report No. 2741 (1995).
[39] X. Pennec, P. Fillard, N. Ayache, A Riemannian framework for tensor computing, International Journal of Computer Vision 66 (1) (2006) 41–66.
[40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL http://tensorflow.org/
[41] P.-O. Persson, J. Peraire, Curved mesh generation and mesh refinement using Lagrangian solid mechanics, AIAA Paper 2009-0949 (2009).
[42] P.-O. Persson, J. Peraire., Sub-cell shock capturing for discontinuous Galerkin methods, AIAA Paper 2006-112 (2006).