# Mesh Adaptation for Large Eddy Simulation with the FR/CPR Method



Jeremy H. Ims

B.S. Physics, University of Kansas, 2013

B.S. Astronomy, University of Kansas, 2013

B.S. Mathematics, University of Kansas, 2013

M.S. Aerospace Engineering, University of Kansas 2015

Submitted to the graduate degree program in Aerospace Engineering and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Committee members

ZJ Wang, Chairperson

Craig McLaughlin

Ray Taghavi

Cheng Huang

Suzanne Shontz

Erik Van Vleck

Date defended: November 16, 2022

The Dissertation Committee for Jeremy H. Ims certifies
that this is the approved version of the following dissertation :

Mesh Adaptation for Large Eddy Simulation with the FR/CPR Method

ZJ Wang, Chairperson

Date approved: December 1, 2022

# Abstract

The computational mesh for a Computational Fluid Dynamics (CFD) simulation must provide sufficient cell density at proper locations in the domain in order to resolve the flow physics impactful to the targeted engineering parameters. The key locations for resolution are often imprecisely known and so must be found by trial and error. This dissertation discusses CFD mesh adaptation—the computer guided adjustment of the mesh in response to the simulation. Specifically, this document addresses mesh adaptation for Large Eddy Simulation (LES) with the Flux Reconstruction / Correction Procedure via Reconstruction (FR/CPR) method. It presents a computer program to adaptively refine 3D unstructured hexahedral meshes, guided by the distribution of error within the flow field, estimated by an error indicator algorithm integrated into the flow solver. Furthermore, it introduces four error indicator algorithms for tracking the location and the amount of under-resolution in turbulent flow fields. The error estimators are derived, mathematically analyzed, and numerically tested upon two well-known benchmark LES simulations. The analysis leads to a performance evaluation of the error indicators, judged on their ability to drive the CFD simulation toward truth. The four error indicators are: 1) The Unsteady Residual Indicator (${}^{\mathbf{unStd}}\mathcal{E}$), based on the unsteady residual from the FR/CPR calculation, 2) The Smoothness Indicator (${}^{\mathbf{smth}}\mathcal{E}$), based on a local smoothness indicator, 3) The Adapted Toosi-Larson Indicator (${}^{\mathbf{T.L.err}}\mathcal{E}$), based on the estimated small scale turbulent kinetic energy, and 4) The Average Toosi-Larson Indicator ($\overline{{}^{\mathbf{T.L.err}}\mathcal{E}}$), conceptually the same as ${}^{\mathbf{T.L.err}}\mathcal{E}$ but formulated to be less costly to compute. Upon LES simulations that model transitional flow past the T106 low pressure turbine blade, all of the error indicators demonstrate ability to boost resolution of the flow field, improving simulation accuracy of force coefficients, vortex structure, Reynolds stresses, and the energy spectra. ${}^{\mathbf{unStd}}\mathcal{E}$ and ${}^{\mathbf{T.L.err}}\mathcal{E}$ are found to be the fastest to bring improvement to coarse-meshed simulations. Of the two, ${}^{\mathbf{unStd}}\mathcal{E}$ is mathematically simpler and easier to compute.

# Acknowledgements

First, I wish to thank my advisor and academic mentor Dr. ZJ Wang for his guidance, support, and the learning environment he fostered and maintained in the Computational Thermo-Fluids Laboratory. From the beginning, he shared his contagious enthusiasm for the challenge and significance of the CFD quest. For me, that quest began with him pointing to a video of a failing CFD simulation and proclaiming: "Look! It's losing the physics." This dissertation, focused on boosting simulation resolution, is my attempt to preserve the physics.

Thank you to my committee members Dr. McLaughlin, Dr. Taghavi, Dr. Huang, Dr. Shontz, and Dr. Van Vleck. Thank you to Dr. Van Vleck & Dr. Shontz for essential coursework in numerical analysis & parallel programming, to Dr. Taghavi for his riveting teaching of fluid mechanics, to Dr. McLaughlin for his valuable astrodynamics coursework that has proved instrumental in my work beyond this dissertation, and to the newest committee member Dr. Huang for contributing CFD expertise. I also wish to thank my enthusiastic vector calculus professor Dr. Bozenna Pasik-Duncan, who inspired me to add both math and physics to my astronomy undergraduate major, paving the way for my graduate study. I also thank my family for the many sacrifices made to support my work, especially my mother for collaborative editing and organizational support. I also wish to thank my engineering work colleagues at Garmin and especially my supervisor Brandon Guttersohn, who tolerated my odd working hours and remote work so I would have the schedule flexibility to attend CFD research meetings and to push this dissertation over the finish line.

Finally, I wish to acknowledge research financial support from the Air Force Office of Scientific Research under Award Number FA9550-20-1-0315, from the Army Research Office under Award Number W911NF-20-1-0065, and from the National Science Foundation Graduate Research Fellowship Program under grant number NSF0064451.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

It is no exaggeration to say that the modern world is dependent on the control of fluid in motion. We rely on high-speed vehicular transport, which involves movement through one or more fluids, either air or water. The engines that propel our vehicles as well as the fuel pipes and the pumps are also fluid based. Whether we seek to prototype, build, or operate machinery, we must understand and predict the motion of objects in realistic flow conditions.

Computational fluid dynamics (CFD) answers that need numerically by solving a set of equations to model the physics of fluid motion. These equations are called the Navier-Stokes equations, named after mathematician / physicist George Gabriel Stokes (1819-1903) and French engineer / physicist Claude-Louis Navier (1785-1836). The equations express the conservation of mass, momentum, and energy for a freely deformable viscous medium (a fluid), which experiences internal stress force proportional to both the pressure and the gradient of velocity. As a predictive model for fluid motion, Navier-Stokes is combined with a thermodynamic equation of state, relating pressure, temperature, and density.[1]

In principle, those equations are all that is required to predict any type of flow, but in practice, it's a more complicated matter. Turbulence—motion characterized by chaos—is one of those complications. Random fluctuations from eddies that swirl out of the bulk flow, splintering, cascading into whorls that diffuse and scatter the energy. Predicting the effect of such motion is difficult due to the coupling between large and small scales.

The modeling of turbulence requires an enormous number of calculations, and that modeling effort has inspired several techniques, with the most accurate methods requiring the most effort. At

[1] For the equations, see (Hirsch, 2007, chap. 1 & table 1).

the far extreme of effort would be the brute force DNS (*Direct Numerical Simulation*), which aims to resolve all parts of the flow, from the largest currents to the smallest eddies. The less intense LES (*Large Eddy Simulation*) sidesteps the difficulty of resolving the small eddies by deferring to a semi-empirical subgrid scale model. RANS (*Reynolds Averaged Navier-Stokes*) doesn't even try to predict any eddies, aiming only for the statistical average flow by corralling the turbulent element into random variables, modeled statistically. Other techniques, less mainstream, conceptually split the flow field into interacting regions of unique dominant behavior, which are treated individually by dissimilar approximation.[2]

With all solution procedures, the result is a large system of equations in need of calculation. CFD is the numerical art of solving the equations by a high-speed computer program, yielding a predictive flow field. In spite of the difficulties of modeling turbulence, CFD technology is of great practical interest to engineers. Since the development of CFD in the 1960s,[3] CFD has become a major tool for engineering companies since it allows for in-depth analyses when experimental tests would be impractical or too costly.

A variety of numerical methods exist to solve the Navier-Stokes system of equations. In an engineering environment, the usual procedures involve a grid, which subdivides the physical domain into a set of small, non-overlapping cells. Local fluid conditions within the cells are approximated, separate and apart from the fluid interactions between the cells.

The accuracy of the calculation greatly depends on the quality of the mesh. If the mesh is too coarse, the flow field will be incorrectly captured. If the mesh is too fine, the computation will be too long to execute in a practical time frame. Finding the proper balance between accuracy and execution length generally requires the hand of an experienced analyst to iteratively refine a starting mesh to achieve the needed accuracy. As highlighted by NASA's *CFD Vision 2030*, the meshing process is commonly acknowledged by CFD practitioners to be the primary obstacle and the most time-consuming step of CFD analysis (Slotnick et al., 2014).

Mesh creation is especially onerous when the simulation is new to the analyst. To produce

[2]For a summary of the techniques, see (Hirsch, 2007, chap. 2).

[3]circa 1955-1980. See for example (Richardson, 1922; Harlow et al., 1955; Jameson et al., 1981).

a mesh, knowledge of the flow field is required. But to simulate the flow field, a mesh must be supplied. So the practical work of CFD devolves into a tedious "guess and check" exercise. At this point, the frustrated analyst may speculate that the process of mesh turning could be more profitably done by a suitably equipped computer program. Such a program could iterate between the mesh and simulation, adjusting the former to supply the later, much like an adaptive numerical integrator will redistribute quadrature points to maintain a target resolution.[4]

## 1.1 Adaptive CFD

There are three adaptive procedures by which a CFD simulation's resolution may be adjusted after the numerical solver has been established on its initial mesh: **1)** R-adaptation, **2)** P-adaptation, and **3)** H-adaptation. *R-adaptation*, also known as the "moving mesh method" refers to mesh modification that either re-distributes the cells or alters the cell sizes without change to their number. *P-adaptation* refers to modification of the CFD solver's internal degree of numerical approximation through either the addition or the subtraction of terms to the modeling polynomials. *H-adaptation* refers to direct mesh modification that adds or subtracts cells.[5] All three types of adaptation require instruction to guide the process of redistributing the simulation's degrees of freedom. That information can come from either the CFD analyst designing the simulation, or it can come from an algorithm estimating a spatial error distribution.

## 1.2 Approaches to Adaptive CFD

Early approaches to adaptation, as well as some later approaches, were focused upon minimizing the error of interpolation. In other words, the best mesh would be optimally designed for interpolation of the solution field. The work of Diaz et al. (1997) is an example. The theory, borrowed from the finite element procedure, holds that interpolation error is proportionate to the second derivative of the interpolant upon the mesh (or to the Hessian matrix when more than one spatial dimension

[4] adaptive quadrature: (Shampine et al., 1997, chap. 5.2).
[5] For more detail, see (Li et al., 2001).

is at play). The goal is to rearrange the cells and edges of the mesh to minimize the error of interpolation. One challenge, considered by Diaz et al., is the need to choose a quantity for optimal interpolation. But as the authors observed:

> ...one variable cannot encapsulate all the physics of the system (Diaz et al., 1997, pg. 476).

Therefore, they proposed pooling the flow field variables. Since the variables have dissimilar units, they non-dimensionalized the variables, scaling by local maxima.

That pooling idea was criticized by Habashi et al. (2000), whose authors argued that the former idea of the "intersection metric" was likely to produce isotropic grids when anisotropic grids would do better. In the first of a three part series of papers on the topic, Habashi et al. (2000) proposed a "Mesh Optimization Methodology" (acronym "MOM") utilizing a single variable Hessian, evaluated upon the cellular edges of a 2D triangular mesh. Armed with the edge-based error distribution, the MOM procedure would iteratively apply mixed H-adaptation and R-adaptation to add, subtract, and move the triangle edges, smoothing the error distribution. The final mesh would be highly anisotropic, optimally tuned to the chosen flow field variable, following the contours of shock waves, etc.

In later works,[6] the MOM procedure was applied to more advanced CFD problems with both structured and unstructured 2D meshes. The authors demonstrated remarkable consistency at being able to recover the same optimized mesh from dissimilar starting points. Also, upon these optimized meshes, dissimilar CFD solution techniques would produce surprisingly similar flow fields. And typically unstable CFD calculations would become robust, matching the stability of the complex CFD solvers. The discovery highlighted the intertwined relationship between the mesh and the simulation:

> It brings one to question the major effort of the last decade into more refined algorithms, all the while ignoring the impact of the meshes on the numerical solution (Dompierre et al., 2002).

[6] (Ait-Ali-Yahia et al., 2002; Dompierre et al., 2002)

As impressive as the MOM results were, the approach had drawbacks. One documented weakness was the difficulty with extending the procedure to unsteady flows. As the authors point out:

> Our adaptation strategy, by being an a posteriori one, cannot predict in advance the movement of the vortices (Habashi et al., 2000, pg. 741).

In an attempt to extend the MOM to unsteady flow, they manually adjusted a global size parameter in the program to artificially produce small cells in the appropriate area. While that maneuver did produce a viable mesh, it hardly solved the bigger conceptual problem, which is that a perfectly optimized mesh for $t = 1$ will not be optimal for $t = 2$, nor will a mesh tuned for variable $X$ necessarily work for variable $Y$. Another disadvantage is the complexity of the re-meshing process, which involves node addition, node subtraction, edge swapping, and repositioning of the nodes. The strong coupling between the meshing process and the CFD solution is clearly advantageous, but it would be nice if the process was less complicated. That may be one of the reasons the series of papers[7] did not extend to 3D simulations. Nevertheless, variants of the MOM idea continue to be explored (now that computer power has increased), even in 3D (as in Frey & Alauzet, 2005). The Hessian matrix also continues to play a central role in error estimation, even when the target error metric is not directly related to interpolation (as in Toosi & Larsson, 2017a). So too does interpolatory error continue to be used, separate from Hessian matrix or the MOM procedure.

A recent and novel example of interpolation driven adaptation can be found in the work of Foti et al. (2020), which approaches unsteady flow optimized meshing from a unique vantage point, best described as the merger of video compression and CFD. With a process reminiscent of movie codec image compression, the CFD procedure employs linear algebra (singular value decomposition and rank-revealing QR decomposition) to calculate optimal spatiotemporal sampling points for the flow field's time history.

Interpolation is not the only criteria that has been used to drive mesh adaptation. Roy (2009), classifies the other procedures into four categories:

[7] *Anisotropic Mesh Adaptation: Towards User-Independent, Mesh-Independent and Solver-Independent CFD*: (Habashi et al., 2000; Ait-Ali-Yahia et al., 2002; Dompierre et al., 2002).

- **Feature-based** methods adapt on the basis of a solution field attribute, such as a gradient or a pressure value.
- **Adjoint-based** methods adapt according to an uncertainty estimate propagated backwards.
- **Discretization-error-based** methods adapt according to the difference between the solutions of the exact PDE and the discretized PDE. Generally, the true solution isn't known (otherwise there would be no need to simulate it), so estimation is used to proxy the discretization error.
- **Truncation-error-based** methods adapt according to the difference between the exact and the approximate *PDEs* (the equations themselves, not the solutions as with discretization-error-based methods).

Roy reports feature-based adaptation as the most popular approach at the time but notes that it does not always work. He attributes the failure to lack of mathematical justification. In regards to the adjoint procedure, he notes interesting results in the literature, but he dismisses the method as overly complex. The other two methods, he reasons, are related mathematically. In a final analysis with the Burgers Equation[8], he concludes that truncation-error adaptation is the best, explaining that:

> ...the truncation error serves as the local source for [the] discretization error transport equations (Roy, 2009, pg. 19).

In other words, *truncation-error* is an upstream version of *discretization-error*. Truncation-error, coming from omissions in the modeling equations, is the easier quantity to estimate. Discretization-error, the accumulated inaccuracy at the end of the simulation, is more obviously relevant but difficult to estimate. Roy's advice is to target the source of the inaccuracies, choosing truncation-error as the basis for H-adaptation[9].

Fidkowski & Darmofal (2011) express a different opinion of error transport and the adjoint procedure. They highlight the fundamental difficulty of quantifying the effect of transported error:

[8]Burgers Equation: (Morton & Mayers, 2005, pg. 105-106)
[9]For more on the error transport equation, see (Qin et al., 2004).

> For hyperbolic problems, [like Navier-Stokes] the residual and discretization error may not necessarily be large in certain crucial areas that significantly affect the solution downstream and the computed outputs [of final engineering interest] (Fidkowski & Darmofal, 2011, pg. 8).

As evidence, they cite the example of separated flow over an airfoil, pointing out that only the adjoint method is specifically formulated to quantify the actual importance of the upstream inaccuracies. The authors provide the definitive review of the adjoint method, starting from its mathematical foundation, progressing to its use with H-adaptation & R-adaptation, moving to key tests, and concluding with a summary of both its strengths and its weaknesses. The adjoint method is perhaps best described as a process of reverse engineering to identify the cause of output perturbations. The method assigns upstream blame for the downstream vacillation of any chosen output parameter $\delta J$. For example, it might determine which parts of the flow field are responsible for the lift coefficient of a wing. Such knowledge is of immense value for adaptation because it directly reveals where the simulation's degrees of freedom ought to be placed for maximum effect. This strength of the adjoint method is in direct contrast to all other methods of error estimation, which cannot consider the goal of the simulation in the assessment and are therefore liable to misallocate resources, needlessly resolving useless parts of the flow while missing the truly important details. To demonstrate this point, Fidkowski & Darmofal (2011) reference an adaptive RANS simulation of an EET ("energy efficient transport") airfoil intended to predict the lift coefficient (originally from Venditti & Darmofal, 2002, 2003). Whereas adjoint-assisted adaptation was able to match the experimentally known lift (experiment by Lin & Dominik, 1995), a pure Hessian-based adaptation, predicated on Mach number, was unable to do so. The Hessian-based adaptation drove the simulation to an incorrect answer by allocating too little resolution to the inviscid regions of the flow field.

The conclusion of (Fidkowski & Darmofal, 2011) is that adjoint adaptation is the most efficient way to obtain mesh-independent lift, drag, and moment coefficients — at least for CFD simulations of the steady RANS variety. Other researchers have also reported impressive success with

the adjoint procedure, finding that it can provide reasonable error estimates by which to improve simulation accuracy. Comparisons against fixed grid approaches indicate that CPU time can be reduced by orders of magnitude (as demonstrated by Yano & Darmofal, 2012; Zhou et al., 2017).

The mathematical essence of the adjoint method may be gleaned from its defining equation, the contemplation of which will reveal not only how the adjoint method works, but also why the procedure is — at least for the present — ill suited for general use with *unsteady* 3D simulations, especially turbulent ones.

To demonstrate the unsuitability, let $J(\mathbf{u})$ represent a target output quantity of interest, a function of flow field $\mathbf{u}$. Assume that $\mathbf{u}$ satisfies the residual equation $\mathbf{R}(\mathbf{u}) = 0$, as would be the case in a steady-state CFD problem. The defining equation for the adjoint is

$$\delta J(\mathbf{u}) \equiv \psi^{\mathrm{T}} \, \delta \mathbf{R}(\mathbf{u}).$$

Field $\psi$ is the desirable adjoint vector. It must be solved by an inversion of the simulation, via the discrete adjoint equation:

$$\left( \frac{\partial \mathbf{R}}{\partial \mathbf{u}} \right)^{\mathrm{T}} \psi + \left( \frac{\partial J}{\partial \mathbf{u}} \right)^{\mathrm{T}} = 0,$$

The disadvantage begins with the procurement of $\psi$, a computational feat in its own right. Also troublesome is the nature of the coupling between $\psi$ and $\mathbf{u}$. When the math is extended to time dependent flows, not only does the adjoint procedure require the *final* version of $\mathbf{u}$, it also requires *the entire time history* (Fidkowski & Darmofal, 2011). Memory space, potentially massive, is a prerequisite. The adjoint equations must be solved in reverse time, from the simulation's end to its start. However, for the most accurate result, it's not enough to replay the timesteps like a movie in reverse, frame-by-frame. The mesh actually needs to be adapted between the frames, so the trip backward through $\mathbf{u}$'s history will not necessarily coincide with the trip forward (Fidkowski, 2017).

Adaptation with the adjoint is computationally expensive, but it is still tractable in some useful cases, for instance when the CFD problem can be reduced to two dimensions (as in Wang

& Mavriplis, 2009), or the CFD problem can be made time independent (as in Park, 2004), or the effect of viscosity can be ignored (as in Alauzet & Loseille, 2016). Turbulent simulation via RANS is amenable (as in Hartmann et al., 2011) because RANS is inherently time independent. But high fidelity unsteady turbulent simulation via LES—simultaneously 3D, time dependent, and viscous—is not compatible with adjoint adaptation at this time. The computational cost is prohibitively high. Even worse, the adjoint math has problems with unsteady turbulence. The chaos threatens the mathematical integrity of the procedure because the instantaneous flow field quantities are so irregular as to be mathematically unsound (shown by Wang et al., 2014).

The chaos, it might be argued, ought not to be insurmountable because the engineer is likely only interested in time averaged quantities. There has been work in that direction (by Blonigan et al., 2018), smoothing the chaos via a procedure known as Least Squares Shadowing. However, the statistics require many timesteps to stabilize, so the price is formidable. Overall and for the present, output adjoint-based mesh adaptation is impractical for LES. That is problematic because of the need for high-fidelity turbulence simulation. To quote NASA's *CFD Vision 2030* report:

> Perhaps the single, most critical area in CFD simulation capability that will remain a pacing item by 2030 in the analysis and design of aerospace systems is the ability to adequately predict viscous turbulent flows with possible boundary layer transition and flow separation present (Slotnick et al., 2014, pg. 12).

Alauzet & Loseille (2016) echo this same unmet need in their review paper *A Decade of Progress on Anisotropic Mesh Adaptation for Computational Fluid Dynamics*. They report that most existing adaptation algorithms are restricted to steady, inviscid flow fields. The authors address the problem of unsteady flow by proposing a time-extended version of the adjoint procedure, executed periodically at fixed points along the simulated timeline. They conclude by highlighting the unsolved challenges presented by: 1) viscous flow, 2) boundary layer flow, and 3) turbulence.

With adjoint adaptation unavailable for LES, *physics based* adaptation criteria have been developed instead. Adaptation methods in the *physics* category utilize physical attributes of the simulated turbulence to judge the realism and veracity of the CFD simulation. A physics based

method might, for example, compare turbulent length scales, or the balance of energy, or the size of viscosity subcomponents, or the statistical correlation between nearby points in space.[10] Celik et al. (2005), the inventors of the LES quality index LES_IQ, recommend that 80% of the turbulent kinetic energy be resolved. Often, the physics-based adaptation strategy will utilize a quantity comparison between a resolved and an estimated element of the turbulence, above and below the threshold of the subgrid scale model.

Benard et al. (2016) takes the subgrid approach while also incorporating the advice of Celik et al.. His proposal is for two-step adaptation, first to resolve the mean flow, second to resolve the turbulence to the 80% threshold. The first step utilizes an interpolation-based procedure with a Hessian matrix. The second step uses a physics based procedure, comparing unresolved to resolved turbulent kinetic energy. Of course, the amount of unresolved kinetic energy is not actually a known quantity, but it is circuitously estimated through the combination of a subgrid scale model and a theoretical math relationship connecting the subgrid model's predicted eddy viscosity and the LES algorithm's filter width. For Benard et al. the filter width was assumed equal to the mesh cell size, justifying the decision to base mesh adaptation upon it.

Slightly earlier than Benard et al., the approach proposed by Antepara et al. (2015) aims at roughly the same physics target. Instead of kinetic energy, velocity magnitude serves as the metric, specifically the magnitude of the residual velocity after application of a...

> ...non-uniform Laplace filter based on a Gaussian filter, [normalized], [conservative], and ... self-adjoint (Antepara et al., 2015).

Breaking with the pattern, Abbà et al. (2020) uses a structure function that is "widely used to study turbulence statistics and ... known to be directly related to subgrid stresses": $D_{i,j} = \langle \Delta u_i \Delta u_j \rangle$, with $u$ a directional component of velocity and $\Delta u \equiv [u(\vec{x} + \vec{r}) - u(\vec{x})]$. Insufficient resolution is signaled by abnormally large magnitudes of the $D_{i,j}$ values, judged by comparison against corresponding $D_{i,j}$ values from isotropic flow.

[10] For examples, see Benard et al. (2016) and references therein.

## 1.3 The Goals of this Dissertation

Many recent attempts to increase the resolution of LES have focused upon P-adaptation: (Naddei et al., 2019; Abbà et al., 2020; Wang et al., 2020). However, this dissertation focuses upon H-adaptation, with the claim that H-adaptation alone has more potential than P-adaptation alone. With a poor mesh, several iterations of pure H-adaptation can produce a reasonable solution, while unassisted P-adaptation may fail at flow field discontinuities. The phenomenon exists because H-adaptation provides a direct boost to the spatial sampling, which is needed in the discontinuous situation.

P-adaptation provides benefit in other contexts by enhancing the dexterity of the modeling equations. Both H-adaptation and P-adaptation have their place within the larger goal to improve the quality of the CFD simulation. A suitably powerful CFD computer program can actually switch between H-adaptation and P-adaptation at the push of a button — provided there exists an error estimator algorithm to provide guidance. So in the big picture, the error estimator is more important than any difference between P & H adaptation. So although the focus of this dissertation is upon H-adaptation, the aim is to better the error estimators.

As indicated by NASA's *CFD Vision 2030* report, there is a growing need for predictive modeling of high-fidelity turbulence. That need motivates the concentrated focus of this dissertation, not just upon error estimators, but turbulent-capable error estimators, specifically LES capable ones — the simpler the better, given the already enormous computational costs associated with the calculation.

Besides the error estimators, it is additionally the intent of this dissertation to provide both blueprint and high quality implementation of the computational infrastructure to actually *do* 3D unstructured mesh adaption for industrial scale modeling problems of unsteady turbulence.

## 1.4 Structure of this Dissertation

Chapter 2 covers the mathematical underpinnings of all algorithms in this dissertation, starting with the CFD numerical scheme and extending through the error estimators featured in this work. The CFD scheme is FR/CPR, a high-order discontinuous procedure that solves the unsteady Navier-Stokes equations. Four error indicators are presented, the first three of which are considered capable enough for detailed testing.

1. $^{\mathbf{unStd}}\mathcal{E}$, an estimator based on the high-order unsteady residual operator and inspired by the residual-based adaptation criterion used for steady flow adaptation. In the terminology of (Roy, 2009), this error estimator would be classified as a *truncation-error* method.

2. $^{\mathbf{smth}}\mathcal{E}$, an estimator based on local geometric smoothness of the flow field, inspired by prior work in shock capturing (Persson & Peraire, 2006) and error estimation for non-turbulent flow (Naddei et al., 2019). This estimator would be classified as a *feature-based* approach.

3. $^{\mathbf{T.L.err}}\mathcal{E}$, a *physics based* estimator translated to the FR/CPR simulation methodology from the work of Toosi & Larsson (2017a). This error indicator is predicated on sub-grid energy density.

4. $\overline{^{\mathbf{T.L.err}}\mathcal{E}}$, a simplified version of $^{\mathbf{T.L.err}}\mathcal{E}$, less capable but far easier to compute.

Chapter 3 blueprints both algorithms and computer program architecture to produce error-adapted 3D unstructured meshes from an unsteady fluid simulation. The proposed architecture hooks into a highly parallelized implementation of FR/CPR for execution in tandem.

Chapter 4 numerically tests the proposed error estimators, implemented per the blueprints of Chapter 3. Well-known benchmark LES simulations of the T106 turbine blade (Hillewaert & JS., 2016, 2018) are used to compare and contrast the adaptation performance. Simulation accuracy is assessed by flow field comparison against DNS (Alhawwary & Wang, 2019).

Chapter 5 provides a glimpse of the future of error-guided mesh adaptation, discussing the burgeoning research area combining Machine Learning with CFD mesh adaptation. This chapter

completes the joint research project first reported upon by Phommachanh (2021), which sought to trial the artificial intelligence of a neural network in place of an error estimator.

Finally, Chapter 6 summarizes and presents areas for future study.

# Chapter 2

# Numerical Methods[1]

Navier-Stokes is the system of partial differential equations embodying the physics of fluid substance. No general procedure exists to solve Navier-Stokes analytically, but the equations can be approached numerically, discretized, and cast into a form amenable to computer calculation. CFD is the numerical art of actuating the Navier-Stokes calculation to obtain a predictive flow field.

This chapter describes the CFD calculation used in this research. It also describes the mathematics of each proposed error estimator algorithm for trial and testing for boosting simulation resolution by subdividing the simulation's supporting mesh.

## 2.1 FR/CPR: The Discretization of Navier-Stokes

Navier-Stokes depicts the conservation of mass, momentum, and energy for the fluid medium. Let $Q$ denote the conservative flow variables within a differential volume of space. That is, let $Q$ be the column matrix $[\rho, \vec{m}, E]^{\mathrm{T}}$, with $\rho$ representing mass (per unit volume), $\vec{m}$ representing momentum (per unit volume), and $E$ representing total energy (per unit volume). Let function $\vec{F}$ denote the *flux vector*, i.e. the peripheral exchange of fluid over the boundary of the differential volume, both its convective and its dissipative parts (ref. Hirsch, 2007). With the defined symbols, the Navier-Stokes equations, written in conservative form, are expressed thus:

[1] This chapter extends the analysis from:

- Ims, J. & Wang, Z. J. (2022). A comparison of three error indicators for adaptive high-order large eddy simulation. In *AIAA SciTech 2022 Forum* (pp. 1201)
- Ims, J. & Wang, Z. J. (under review). A comparison of three error indicators for adaptive high-order large eddy simulation. *(submitted to) Journal of Computational Physics*.

$$\frac{\partial Q}{\partial t} = -\nabla \cdot \vec{F}(Q, \nabla Q). \tag{2.1}$$

In words: Absent any source, the given quantity of fluid will only change in the amount it flows into or out of the region.

There is another way to mathematically express the same truth, which will be more useful for upcoming calculation:

$$\int_V \left( \frac{\partial Q}{\partial t} + \nabla \cdot \vec{F}(Q, \nabla Q) \right) \mathrm{d}\Omega = 0. \tag{2.2}$$

To solve Navier-Stokes, this research employs the FR/CPR method, short for *Flux Reconstruction or Correction Procedure via Reconstruction*. The method was conceptualized by Huynh (2007), with further development by Huynh et al. (2014) and Wang & Huynh (2016). FR/CPR begins with the Navier-Stokes in the form of equation (2.2), discretizing the equation in a manner similar to the Discontinuous Galerkin method (Bassi et al., 2006).

The first step is to define the volume elements, represented by symbol $V$. For that task, a tessellated grid is established, covering the domain. The integral of equation (2.2) is then reinterpreted to apply to each cell individually upon that mesh. In so doing, the integrand is multiplied by a test function and a corrective field is added to counteract modeling inaccuracy at the cell-to-cell interface. The result is:

$$\int_{V_i} W \left( \frac{\partial Q}{\partial t} + \nabla \cdot \vec{F}(Q, \nabla Q) + \delta \right) \mathrm{d}\Omega = 0, \tag{2.3}$$

where $V_i$ is now the volume of a single cell (index $i$), $W$ is the test function — arbitrary beyond a requirement that it be zero outside $V_i$ — and $\delta$ is the spatial correction field, defined by the relation:

$$\int_{V_i} W\delta \, \mathrm{d}\Omega = \oint_{\partial V_i} W \left( \vec{F}_{com} - \vec{F}(Q, \nabla Q) \right) \cdot \mathrm{d}\vec{S}. \tag{2.4}$$

$\vec{F}_{com}$ is the "common flux," the virtual flux that exists on the discontinuous border between neighboring cells. $\vec{F}_{com}$ is computable via a Riemann solver (see for example Roe, 1981).

Although the correction field $\delta$ is introduced in (2.3) as a volume field, equation (2.4) shows that it is more appropriate to consider it a source term upon the boundary $\partial V_i$. It arises from (and corrects for) the mismatch between $\vec{F}$ *inside* the cell and the jump discontinuity $\vec{F}_{com}$ *surrounding* the cell. Another way to interpret $\delta$ is as a mathematical anchor for the integrand at the cell-to-cell interface; as such, it defines a unique, single-valued field where there would otherwise be an ambiguous function with multiple values.

The next step will be to remove the integral of equation (2.3), leaving a system of per-cell equations that relate the internal action of $Q$ to the peripheral action of $\vec{F}$. However, before proceeding further, it's helpful to explain the overall strategy of FR/CPR. The goal is to model $Q$ in a per-cell manner, via polynomials. Every cell is to receive a unique modeling polynomial, and every cell is to equilibrate its polynomial among the neighbors, working through the boundary fluxes $\vec{F}$. Inevitably, there will be discontinuities between neighbors, but the discontinuities are to be absorbed by correction field $\delta$ and forced back upon $Q$ so that the individual modeling polynomials will coordinate. With the strategy established, we may continue.

From equation (2.3), the volume integral is stripped away via judicious choice of the test function $W$ and via the expansion of the integrand terms as polynomials. A system of equations remains, which relates discrete values of the polynomials on a set of nodal points ($j$) within the cell ($i$) that comprises volume $V_i$

$$\left.\frac{\partial Q_p}{\partial t}\right|_{i,j} + \underset{\boldsymbol{p}}{\curvearrowright}\left[\nabla \cdot \vec{F}\right]_{i,j} + \delta_{i,j} = 0. \tag{2.5}$$

In this final version of the Navier-Stokes equations, subscript $p$ denotes the polynomial order. Operator $\underset{\boldsymbol{p}}{\curvearrowright}\left[\cdot\right]$ denotes projection onto the abstract space of the modeling polynomials.

The nodal values of the correction field $\delta_{i,j}$ may be obtained by discretizing equation (2.4), using a procedure similar to that just employed for $Q$. Step 1 introduces an interpolatory polynomial along the cell boundary to model the normal component of the flux jump

$$\left(\vec{F}_{com} - \vec{F}\right) \cdot \hat{n}. \tag{2.6}$$

The action of Step 1 will allow the surface integral to be expanded. Meanwhile, the volume integral can be expanded as Step 2, using the polynomial representation of $\delta$. Step 3 works out the integrals, resulting in the following general expression for the discretized correction field for a linear simplex element:

$$\delta_{i,j} = \frac{1}{|V_i|} \sum_{f \in \partial V_i} \sum_{l} \alpha_{j,f,l} \, F^n_{f,l} \, S_f . \tag{2.7}$$

As before, $V_i$ represents cell volume. The '$\alpha$'s are "lifting coefficients," independent of the flow field. $S$ represents boundary face surface area. $F^n$ represents the per-node & per-face value of the modeling polynomial for the normal flux jump (equation (2.6)). Index $l$ cycles through the chosen interpolation flux points, and index $f$ cycles through the set of boundary faces.

There are a few additional details unique to the formulation of FR/CPR used in this research. For viscous flow simulations, the flux term in equation (2.5) has a dependence on $\nabla Q_p$. The computation of that gradient uses the Bassi-Rebay 2 scheme (Bassi & Rebay, 2000). For numerical stabilization of severely under resolved areas of the flow field, use is made of an accuracy preserving limiter (Li & Wang, 2017). Time integration is possible via both an explicit or an implicit scheme. When explicit, the scheme is the three stage SSP Runge-Kutta (Shu, 1988). When implicit, the scheme is the A-stable, second-order accurate, BDF2OPT procedure (Vatsa et al., 2010). Finally, when the CFD solver is being used for Large Eddy Simulation (LES) the implicit approach is taken. That is, the numerical scheme is used as is, without a subgrid scale model.

## 2.2 The Error Estimators

### 2.2.1 The Unsteady Residual Indicator

Error indicator ${}^{\mathbf{unStd}}\mathcal{E}$ is inspired by the "residual-based" error indicator from Gao & Wang (2011). The original error indicator was used for two-dimensional non-turbulent flow problems. This work extends and adapts the indicator for use with three dimensional LES.

"Error" is defined as the difference in resolution afforded by a +1 order increase in the sim-

ulation's modeling polynomial. Specifically, it is the difference $\left(\frac{\partial Q_{p+1}}{\partial t} - \frac{\partial Q_p}{\partial t}\right)$, where $Q_p$ is the per-cell modeling polynomial of the flow field from equation (2.5). The motivation for such a definition springs from the observation that equation (2.5) — the equation that the fluid simulation actually solves — is but an approximation of the original physics equation (2.1). Likewise, $Q_p$ is only an approximate (and local) imitation of the true flow field $Q$. Presumably, this approximation can be improved by using a higher order modeling polynomial, with its extra degrees of freedom better mimicking the undulations of the flow. That being the case, the difference between a $(p+1)^{\text{st}}$-order approximation and the original $p^{\text{th}}$-order approximation can presumably gauge how far from the original $Q_p$ the true flow field was. But it is not the plain terms of $Q_p$ and $Q_{p+1}$ that appear in the error formula; it is their time derivatives. A derivation will show why.

Before proceeding, it's instructive to pause and anticipate why this particular error definition may be expected to do well at driving autonomous mesh refinement within the context of FR/CPR. In this type of simulation, spatial resolution is jointly set by the polynomial order and the cell tessellation density. An increase to either will boost spatial resolution by increasing the number and density of sampling points. Since a polynomial order increase (P-refinement) will act similarly to a cell division (H-refinement), it can be expected that an error indicator based on the former will serve well for making judgments about the latter.

The derivation of ${}^{\textbf{unStd}}\mathcal{E}$ begins with equation (2.5), the discretized Navier-Stokes equation. Here it is again, rearranged to a form that will be useful to our purpose.

$$\left.\frac{\partial Q_p}{\partial t}\right|_{i,j} = \left.\operatorname*{Res}_p\left(Q_p, Q_p^{\text{neighbors}}\right)\right|_{i,j} \tag{2.8}$$

$\text{Res}(\,\cdot\,)$ is the "residual operator." It is the sum of the boundary flux terms and the spatial correction field, both from the original equation. The residual operator will be the primary focus of further calculations, necessitating simplification of its notation. Parameters $Q_p$ and $Q_p^{\text{neighbors}}$ will

be combined into the joint symbol $\{Q_p\}$.

$$\underset{p}{\text{Res}}\Big( Q_p, Q_p^{\text{neighbors}} \Big) \quad \Longrightarrow \quad \underset{p}{\text{Res}}\big( \{Q_p\} \big)$$

The next step in the derivation is to consider the quantity $( \mathrm{d}Q_{p+1} - \mathrm{d}Q_p)$, i.e., the differential "error" that accumulates during a single tick of the simulation's clock. This is the parent quantity giving rise to the error estimate $\epsilon$ we are after.

$$\begin{aligned} \mathrm{d}(\,\text{error}\,) &\equiv \big( \mathrm{d}Q_{p+1} - \mathrm{d}Q_p \big) \\ &= \left( \frac{\partial Q_{p+1}}{\partial t} - \frac{\partial Q_p}{\partial t} \right) \mathrm{d}t \\ &= \left( \underset{p+1}{\text{Res}}\big( \{Q_{p+1}\} \big) - \underset{p}{\text{Res}}\big( \{Q_p\} \big) \right) \mathrm{d}t \\ &= \epsilon \; \mathrm{d}t. \end{aligned} \tag{2.9}$$

As can be seen from (2.9), the quantity $\epsilon$ is directly proportional to the accumulating error at each "tick" of the simulation's clock, with the proportionality constant being $\mathrm{d}t$. In other words, $\epsilon$ is the time derivative of the polynomial difference, or, equivalently, the error generation rate. The calculation of $\epsilon$ can be approximated as:

$$\epsilon_{i,j} \approx \left\{ \underset{p+1}{\text{Res}}\left( \underset{\boldsymbol{p+1}}{\overset{\frown}{\cdot}}\Big[ \{Q_p\} \Big] \right) - \underset{\boldsymbol{p+1}}{\overset{\frown}{\cdot}}\Big[ \underset{p}{\text{Res}}\big( \{Q_p\} \big) \Big] \right\}_{i,j} . \tag{2.10}$$

Operator $\underset{\cdot}{\overset{\frown}{\cdot}}\big[ \cdot \big]$ denotes projection onto the polynomial space ($\mathcal{P}$) with the subscripted order. Indices $(i, j)$ number the cells ($i$) and their polynomial interpolation nodes ($j$). In words, the calculation procedure is to:

1. Project $Q_p$ to space $\mathcal{P}_{p+1}$ and apply the FR/CPR residual calculation.

2. Apply the FR/CPR residual calculation to $Q_p$. Then project the result to space $\mathcal{P}_{p+1}$.

3. Do the final subtraction in space $\mathcal{P}_{p+1}$.

Like $Q_{p+1}$, the error quantity $\epsilon_{i,j}$ is a multidimensional object within the $\mathcal{P}_{p+1}$ piecewise polynomial space. It spatially varies across each cell $i$, attaining a different value at each interpolation node $j$. For it to become useful as a mesh refinement flag, the variation must be condensed down to a single number per cell. The flattening approach is to volume-average within each cell, using numerical quadrature.

$$\mathcal{E}_i = \frac{\sum_j |\epsilon_{i,j}| V_{i;\,j}}{\sum_j V_{i;\,j}} \approx \frac{\iiint_{V_i} |\epsilon_i| \, \mathrm{d}V_i}{\iiint_{V_i} \mathrm{d}V_i}. \tag{2.11}$$

Thus far, $\mathcal{E}_i$ is a multi-value object, with one dimension corresponding to each of the original conservative flow field variables within $Q$. That is, $\mathcal{E}$ is the column matrix $[\mathcal{E}_\rho, \mathcal{E}_{\vec{m}}, \mathcal{E}_E]^{\mathrm{T}}$. Any one of these '$\mathcal{E}$'s could reasonably be selected to control mesh refinement, but since the variables carry dissimilar information, it is undesirable to select one over the other, so the three are blended. Each one is normalized, made unitless via division of its global mean across the flow domain. Then the '$\mathcal{E}$'s are combined in a three-way average.

At the end, the time-average is computed, $\langle \cdot \rangle$, collapsing the temporal variation of the error to a single number per cell.

Thus, the final formula for ${}^{\mathbf{unStd}}\mathcal{E}$ is

$${}^{\mathbf{unStd}}\mathcal{E}_i \equiv \frac{1}{3} \cdot \left\langle \frac{\mathcal{E}_{\rho;\,i}}{\overline{\mathcal{E}_\rho}} + \frac{\|\mathcal{E}_{\vec{m};\,i}\|}{\overline{\|\mathcal{E}_{\vec{m}}\|}} + \frac{\mathcal{E}_{E;\,i}}{\overline{\mathcal{E}_E}} \right\rangle \tag{2.12}$$

### 2.2.2 The Smoothness Indicator

Indicator ${}^{\mathbf{smth}}\mathcal{E}$ derives from a shock-capturing algorithm of the same name (Persson & Peraire, 2006), originally intended to identify flow field regions at risk of triggering solver divergence. The repurposed algorithm now goes by a variety of names in the literature. Naddei, in (Naddei et al., 2019), dubs it the "spectral decay indicator" (SED). Testing its capabilities on steady flow simulations, Naddei concluded that its performance and relative simplicity make it a potentially worthwhile option to explore for use with unsteady or even turbulent simulations.

The theoretical motivation to use ${}^{\mathbf{smth}}\mathcal{E}$ for autonomous mesh refinement comes from the fact

that this indicator is purpose-built to find un-smooth regions of the flow field, areas where the gradient is sharp and under-resolved, areas where the spatial change is too rapid for the solver to track. Knowing this, we may anticipate that ${}^{\mathbf{smth}}\mathcal{E}$ will seek to apply mesh refinement in the areas where flow conditions change dramatically over short distances, examples being the boundary layers above friction generating surfaces, the vortex tangle inside turbulent wakes, and the lines along shocks. All seem like reasonable areas to target if the goal is to improve a simulation's prediction of lift, drag, and friction.

The derivation of our version of this indicator begins with $Q_p$, the polynomial approximation of the flow field from the discretized Navier-Stokes equation, equation (2.5). The goal is to evaluate the amount of spatial fluctuation within $Q_p$ upon a change in the local spatial resolution. For that task, $Q_p$ is projected down one polynomial order, obtaining $Q_{p-1}$. Then $Q_{p-1}$ differenced relative to $Q_p$, squared to produce a positive, error-like quantity.

$$\epsilon_{i,j} = \left.\frac{(Q_p - Q_{p-1})^2}{\|Q_p\|^2}\right|_{i,j} \equiv \left[\frac{(\rho_p - \rho_{p-1})^2}{\rho_p^2} \quad \frac{(\vec{m}_p - \vec{m}_{p-1}) \cdot (\vec{m}_p - \vec{m}_{p-1})}{\|\vec{m}_p\|^2} \quad \frac{(E_p - E_{p-1})^2}{E_p^2}\right]_{i,j} \tag{2.13}$$

As usual, subscripts $i$ and $j$ are per-cell, per-interpolation node indices, respectively.

$Q_p$ and $Q_{p-1}$ are within different polynomial spaces, so properly speaking, they do not have coincident interpolation nodes $j$ and cannot be directly subtracted. That difficulty is overcome by projecting $Q_{p-1}$ back to polynomial space $\mathcal{P}_p$, so subtractions within (2.13) should be interpreted as

$$Q_p - Q_{p-1} \quad \Longrightarrow \quad Q_p - \overset{\curvearrowleft}{\boldsymbol{p}}\left[\ \overset{\curvearrowleft}{\boldsymbol{p-1}}\left[\ Q_p\ \right]\ \right].$$

As with $Q_p$, quantity $\epsilon_{i,j}$ is a multidimensional object, with one component corresponding to each of the original conservative flow field variables. Any single component could legitimately be selected as the decision variable for mesh refinement. In this work, we combine the influence of

all components in a three-way average.

$$\widetilde{\epsilon}_{i,j} = \frac{1}{3}(\epsilon_{\rho;\,i,j} + \epsilon_{\vec{m};\,i,j} + \epsilon_{E;\,i,j}) \tag{2.14}$$

Like $Q_p$, quantity $\widetilde{\epsilon}_{i,j}$ possesses both spatial variation across each cell and temporal variation over the simulation's duration. To flatten, we volume-average within each cell and time-average over the duration, obtaining a single number per cell in the end.

$$\mathcal{E}_i = \frac{\sum_j \widetilde{\epsilon}_{i,j} V_{i;\,j}}{\sum_j V_{i;\,j}} \approx \frac{\iiint_{V_i} \widetilde{\epsilon}_i \,\mathrm{d}V_i}{\iiint_{V_i} \mathrm{d}V_i} \quad \Longrightarrow \quad {}^{\mathbf{smth}}\mathcal{E}_i \equiv \langle \mathcal{E}_i \rangle \tag{2.15}$$

### 2.2.3 The Toosi-Larson Indicator

${}^{\mathbf{T.L.err}}\mathcal{E}$ is our variant of an anisotropic grid adaptation scheme developed by Toosi and Larson, specifically designed for LES (Toosi & Larsson, 2017b,a). The motivating philosophy for this error indicator differs significantly from the prior two indicators we've discussed. The prior two were founded upon measures of sensitivity to a local change in the resolution of the flow field. In contrast, ${}^{\mathbf{T.L.err}}\mathcal{E}$ is based upon a measure of turbulent kinetic energy at the smallest resolved length scale. The idea is that an energy buildup at the small scale limit signifies the presence of under-resolved turbulence. Toosi and Larson formulated their original error indicator for Cartesian coordinates. Since FR/CPR simulation operates in curvilinear coordinates, it's necessary to modify the error formula.

The precursor formula for this error indicator, as presented Toosi and Larson (Toosi & Larsson, 2017b,a), is

$$\begin{cases} \vec{u}^{\,*} = -\dfrac{\Delta_n^2}{4}\,\hat{\mathrm{n}}^{\mathrm{T}}(\,\nabla\nabla^{\mathrm{T}}\vec{u}\,)\hat{\mathrm{n}} & \text{(2.16a)} \\ A = \sqrt{\langle \vec{u}^{\,*} \cdot \vec{u}^{\,*} \rangle} & \text{(2.16b)} \end{cases}$$

Sub-equation (2.16a) is a directional high-pass differential filter of the velocity field $\vec{u}$, in the di-

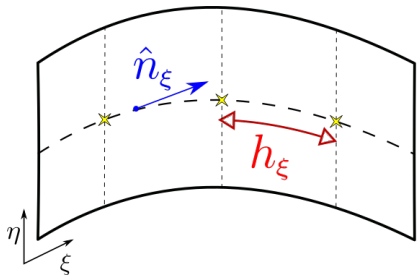



Figure 2.1: A curved cell with computational coordinates $(\eta, \xi)$.

rection specified by unit normal $\hat{n}$, with length scale $\Delta_n$. Sub-equation (2.16b) is the error value $A$, a proxy of the average, small-scale, turbulent kinetic energy that lies along the filter direction. The angle brackets $\langle \cdot \rangle$ represent an average over both space and time. The doubly projected Hessian operator $\hat{\mathrm{n}}^{\mathrm{T}}(\nabla\nabla^{\mathrm{T}})\hat{\mathrm{n}}$ can equivalently be written as $\sum_{k=1}^{3}\sum_{l=1}^{3} n_k n_l \, {}^{\partial^2}/_{\partial x_k \partial x_l}$, which perhaps better illustrates its mathematical meaning: a directional second derivative in the direction of vector $\hat{n}$.

To use this error indicator, one first has to choose a filter direction. Toosi & Larsson (2017b,a) discuss this choice, recommending that it be based on the type of mesh and the method of refinement. For this work, the mesh is of type unstructured, and cell splitting is the method of refinement. The corresponding recommendation is to filter in the direction perpendicular to each cell cut line being evaluated, making quantity $A$ "point" in every direction that spatial degrees-of-freedom could potentially be added. This guidance is given in the context of a Cartesian coordinate system with the implicit assumption that the mesh is linear, but the meshes here are nonlinear, so it is at this juncture that the adjustments begin.

Figure 2.1 depicts a generic high-order curvilinear cell, drawn with two dimensions for clarity. The cell is curved, parameterized by computational coordinates $(\eta, \xi)$. If split, the cut lines will be perpendicular to the coordinate curves of $(\eta, \xi)$, so it is along these curves that we evidently ought to filter. The challenge is how to modify formula (2.16) to best accomplish that. There are two approaches we could adopt. Approach 1 is to apply a coordinate transformation to the derivatives of the Hessian operator, mapping the original equation from the physical space to the computational space, preserving it exactly. Approach 2 is to apply physical reasoning to the problem and seek a new formula, directly within computational space itself. Approach 2 is the one we will ultimately

adopt even though Approach 1 seems, at first glance, to be the superior choice. To show why Approach 2 is better, we will first discuss Approach 1.

Referring to Figure 2.1 for guidance, we seek the second directional derivative in the direction of $\hat{\xi}$, a (normalized) basis vector in computational space. We begin by calculating the first directional derivative, using symbol $\vec{x}$ to represent the position vector in Cartesian space.

$$\mathcal{D}_{\hat{\xi}}(\vec{u}) \;=\; \nabla\vec{u}\cdot\hat{\xi} \;=\; \frac{\partial\vec{u}}{\partial\vec{x}}\cdot\frac{{}^{\partial\vec{x}}/_{\partial\xi}}{\left\|{}^{\partial\vec{x}}/_{\partial\xi}\right\|} \;=\; \left\|\frac{\partial\vec{x}}{\partial\xi}\right\|^{-1}\left(\frac{\partial\vec{u}}{\partial\vec{x}}\cdot\frac{\partial\vec{x}}{\partial\xi}\right) \;=\; \left\|\frac{\partial\vec{x}}{\partial\xi}\right\|^{-1}\frac{\partial\vec{u}}{\partial\xi}$$

Therefore

$$\implies \qquad \mathcal{D}_{\hat{\xi}}(\vec{u}) = s_\xi^{-1}\frac{\partial\vec{u}}{\partial\xi}, \quad \text{with } s_\xi \equiv \left\|\frac{\partial\vec{x}}{\partial\xi}\right\| \tag{2.17}$$

Thus, the first directional derivative equals the partial derivative of $\vec{u}$ with respect to coordinate $\xi$, multiplied by a scale factor. The result is similar for the second directional derivative.

$$\begin{aligned}\mathcal{D}^2_{\hat{\xi}}(\vec{u}) \;&=\; \mathcal{D}_{\hat{\xi}}\left(s_\xi^{-1}\frac{\partial\vec{u}}{\partial\xi}\right) \;=\; s_\xi^{-1}\frac{\partial}{\partial\xi}\left(s_\xi^{-1}\frac{\partial\vec{u}}{\partial\xi}\right) \;=\; s_\xi^{-1}\left(-s_\xi^{-2}\frac{\partial s_\xi}{\partial\xi}\frac{\partial\vec{u}}{\partial\xi}+s_\xi^{-1}\frac{\partial^2\vec{u}}{\partial\xi^2}\right)\\ &=\; -s_\xi^{-2}\frac{\partial s_\xi}{\partial\xi}\left(s_\xi^{-1}\frac{\partial\vec{u}}{\partial\xi}\right)+s_\xi^{-2}\frac{\partial^2\vec{u}}{\partial\xi^2} \;=\; -s_\xi^{-2}\frac{\partial s_\xi}{\partial\xi}\,\mathcal{D}_{\hat{\xi}}(\vec{u})+s_\xi^{-2}\frac{\partial^2\vec{u}}{\partial\xi^2}\\ &=\; s_\xi^{-2}\left(-\frac{\partial s_\xi}{\partial\xi}\,\mathcal{D}_{\hat{\xi}}(\vec{u})+\frac{\partial^2\vec{u}}{\partial\xi^2}\right)\end{aligned}$$

Therefore

$$\implies \qquad \mathcal{D}^2_{\hat{\xi}}(\vec{u}) = s_\xi^{-2}\left(-\frac{\partial s_\xi}{\partial\xi}\,\mathcal{D}_{\hat{\xi}}(\vec{u})+\frac{\partial^2\vec{u}}{\partial\xi^2}\right) \tag{2.18}$$

The second directional derivative is the sum of two terms. The first accounts for the variation due to fluctuation in the length scale of coordinate $\xi$ (as seen from physical space). The second accounts for variation in the value of $\vec{u}$ itself. As before, there is a leading scale factor.

The length scale is going to be more or less constant unless the mesh is unevenly curved, so in equation (2.18), we look past the first term to make a key observation about the second. The

second term, $\partial^2\vec{u}/\partial\xi^2$, is precisely the quantity that we would have calculated *had we simply defined computational space as home base* from the beginning.

As it turns out, computational space can legitimately be interpreted as the native coordinate system of the simulation. Even though the flow field physics may play out in physical space, the modeling polynomials themselves inhabit computational space. Consequently, computational space is a perfectly suitable location to apply filtering for error estimation. With that observation, we switch approaches. Instead of translating (2.16) from physical space to computational space, we simply start there, declaring the error formula to be

$$\begin{cases} \vec{u} = \dfrac{\vec{m}}{\rho} & \text{(2.19a)} \\ \vec{u}^*_\xi = -\dfrac{h_\xi^2}{4}\dfrac{\partial^2\vec{u}}{\partial\xi^2} & \text{(2.19b)} \\ A_\xi = \sqrt{\left\langle \vec{u}^*_\xi \cdot \vec{u}^*_\xi \right\rangle} & \text{(2.19c)} \end{cases}$$

For the filter width $\Delta_n$, we choose $h_\xi$, the distance — in computational space — between successive polynomial interpolation nodes. The nodes are uniformly distributed across the width of the cell, and that width — in computational space — is constant and always equal to 1. Therefore, $h_\xi$ is defined to be $1/(p+1)$, where $p$ is the order of the flow field modeling polynomial. Of course, in physical space, the cells are *not* evenly sized, so the chosen $h_\xi$ will mean that $\Delta_n$ will also not be evenly sized. The filter width will physically vary per cell, in accordance with cell size and the amount by which the computational coordinate is stretched due to local curvature. Said another way, $\Delta_n$ will self adjust to the local resolution in physical space.

If you recall the definition of the $Q$ from Section 2.1 ( i.e. $Q \equiv [\rho, \vec{m}, E]^{\mathrm{T}}$ ), you will see that $\vec{u}$ is not among the variables actually being tracked by the simulation. That's not a problem because $\vec{u}$ is calculatable from the variables that are being tracked. Actually, instead of $\vec{u}$, the calculation

procedure favored here is to jump directly to $\frac{\partial^2 \vec{u}}{\partial x^2}$ via the chain rule:

$$\frac{\partial \vec{u}^2}{\partial \xi^2} = \frac{1}{\rho}\frac{\partial^2 \vec{m}}{\partial \xi^2} - \frac{1}{\rho^2}\left(2\frac{\partial \vec{m}}{\partial \xi}\frac{\partial \rho}{\partial \xi} + \vec{m}\frac{\partial^2 \rho}{\partial \xi^2}\right) + \frac{2\vec{m}}{\rho^3}\left(\frac{\partial \rho}{\partial \xi}\right)^2 \tag{2.20}$$

The conservative variable derivatives are computed on a cell-by-cell basis straight from the FR/CPR modeling polynomials of $Q$. Section 2.1 also informs us that the modeling polynomials are discontinuous at the periphery of each cell, where they intersect and clash with the polynomials of neighbor cells. Because of the discontinuities, the derivatives within equation (2.20) should technically require correction terms at the periphery of each cell. However, in this dissertation, no correction terms are applied. The decision saves a bit of computational effort and does not produce any noticeable deleterious effect. Something else worth noting is that equation (2.20) imparts non-linearity to $\frac{\partial \vec{u}^2}{\partial \xi^2}$ beyond whatever non-linearity the conservative variables themselves may contribute. Thus, even when $Q$ is linearly modeled at $p = 1$, the second derivative of velocity will still exhibit a high-order character. Through the velocity factor, so too will ${}^{\mathbf{T.L.err}}\mathcal{E}$.

The last piece of (2.19) that needs explanation is the angle brackets. The brackets represent a double average, first spatial over the volume of the cell, second temporal over the simulation's timeline.

The final error metric is obtained by cycling over the computational coordinate directions, treating subscript $\xi$ as an index, to select the biggest error for the cell. Symbolically, the calculation is:

- Point-wise error, per cell ($i$), per node ($j$), per coordinate direction ($\xi$):

$$\epsilon_{\xi;\,i,j} \;= A_\xi^2 = \vec{u}^*_{\xi;\,i,j} \cdot \vec{u}^*_{\xi;\,i,j}$$

- Average directional error, per cell ($i$), per coordinate ($\xi$):

$$\mathcal{E}_{\xi;\,i} = \frac{\sum_j \epsilon_{\xi;\,i,j} V_{i;\,j}}{\sum_j V_{i;\,j}} \approx \frac{\iiint_{V_i} \epsilon_{\xi;\,i}\,\mathrm{d}V_i}{\iiint_{V_i}\,\mathrm{d}V_i}$$

- Final error, per cell ($i$):

$$^{\textbf{T.L.err}}\mathcal{E}_i \equiv \sqrt{\max_{\xi}\Big(\big\langle \mathcal{E}_{\xi;\,i} \big\rangle\Big)} \tag{2.21}$$

### 2.2.4 The Average Toosi-Larson Indicator

The Average Toosi-Larson Indicator, $\overline{^{\textbf{T.L.err}}\mathcal{E}}$, is a modified version of $^{\textbf{T.L.err}}\mathcal{E}$, differing only in the position of the time average operator $\langle\,\cdot\,\rangle$. While the calculation of $^{\textbf{T.L.err}}\mathcal{E}$ ends with the time average, the calculation of $\overline{^{\textbf{T.L.err}}\mathcal{E}}$ begins with it, using $\langle\vec{u}\rangle$ as its velocity field rather than $\vec{u}$. Since the calculation feeds off time averaged velocity, there is no need for additional time averaging at the end.

The benefit of $\overline{^{\textbf{T.L.err}}\mathcal{E}}$ is a reduction in the computational effort. $\langle\vec{u}\rangle$ is a commonly desired output variable, so it is typically available from the simulation for free. With $\langle\vec{u}\rangle$ in hand, there are no further time dependent parts of $\overline{^{\textbf{T.L.err}}\mathcal{E}}$, so its entire calculation may be done in post-processing, with a single step procedure. The defining equations for this error estimator are as follows.

- Starting point:

$$\begin{cases} \vec{a}^{\,*}_{\xi} = -\dfrac{h_\xi^2}{4}\dfrac{\partial^2\langle\vec{u}\rangle}{\partial\xi^2} & \text{(2.22a)} \\ B_\xi = \sqrt{\vec{a}^{\,*}_{\xi}\cdot\vec{a}^{\,*}_{\xi}} & \text{(2.22b)} \end{cases}$$

- Pointwise error, per cell ($i$), per node ($j$), per coordinate direction ($\xi$):

$$\epsilon_{\xi;\,i,j} \;= B_\xi^2 = \vec{a}^{\,*}_{\xi;\,i,j}\cdot\vec{a}^{\,*}_{\xi;\,i,j} \tag{2.23}$$

- Average directional error, per cell ($i$), per coordinate ($\xi$):

$$\mathcal{E}_{\xi;\,i} = \frac{\sum_j \epsilon_{\xi;\,i,j} V_{i;\,j}}{\sum_j V_{i;\,j}} \approx \frac{\iiint_{V_i} \epsilon_{\xi;\,i}\,\mathrm{d}V_i}{\iiint_{V_i}\,\mathrm{d}V_i}$$

- Final error, per cell ($i$):

$$\overline{\text{T.L.err}}\mathcal{E}_i \equiv \sqrt{\max_{\xi}\left(\mathcal{E}_{\xi;\,i}\right)}$$

Since $\langle\,\cdot\,\rangle$ is a linear operator that commutes with $\frac{\partial^2}{\partial\xi^2}$, it may be tempting to think that $\overline{\text{T.L.err}}\mathcal{E}$ is nothing but a clever way to calculate $^{\text{T.L.err}}\mathcal{E}$ and that the two similar error estimators will be equivalent. However, no such equivalency exists because operator $\langle\,\cdot\,\rangle$ does not commute with the dot operator in equation (2.19c).

An unfortunate consequence of the differing position of $\langle\,\cdot\,\rangle$ is that $\overline{\text{T.L.err}}\mathcal{E}$ has less informational access to the velocity jitter of the underlying turbulence than its counterpart estimator $^{\text{T.L.err}}\mathcal{E}$. While $^{\text{T.L.err}}\mathcal{E}$ will witnesses the root-mean-square of $\vec{u}^*_\xi$, estimator $\overline{\text{T.L.err}}\mathcal{E}$ will only observe a much smoother square-of-the-average. Therefore, it's possible to predict that $\overline{\text{T.L.err}}\mathcal{E}$ will be less adept at detecting temporally unstable parts of the flow field. It may still have a chance to detect areas of high spatial change.

Error estimator $\overline{\text{T.L.err}}\mathcal{E}$ was a late addition to the error estimator lineup, and due to the mathematically lower expectations for its success, it was tested apart from the main three error estimators $^{\text{smth}}\mathcal{E}$, $^{\text{unStd}}\mathcal{E}$, and $^{\text{T.L.err}}\mathcal{E}$. Consequently, $\overline{\text{T.L.err}}\mathcal{E}$ will not be discussed by Chapter 4, covering numerical tests. $\overline{\text{T.L.err}}\mathcal{E}$ will be discussed by Chapter 5, in the context of a machine learning experiment that incorporated it.

## 2.3 Mesh Splitting

To ensure that the new mesh will have a smooth progression of cell size between refinement zones, we impose a few constraints on the cell marking process. First, there is a one-level size constraint;

any doubly split cell will force a mark upon any un-split neighbor cells. Second, there is a clumping constraint; any cell that is at least 50% flanked by marked neighbors will automatically get marked too, regardless of error value. These extra rules are applied iteratively until the proposed cell markings cease to change in response.

# Chapter 3

# Methodology[1]

The error indicators have a common operating procedure. Built to run in parallel with an FR/CPR flow solver, they accumulate a per-cell error quantity over the course of a simulation, exporting it at the end. In post-processing, the errors may be used by a separate algorithm to refine the mesh.

## 3.1 Procedure for Adaptive Mesh Refinement

Since the guiding error values are natively per-cell quantities, the most straightforward refinement approach is to cleave cells into parts. The process begins, naturally enough, by ranking the cells by error magnitude, highest to lowest. Next, a preset fraction of the topmost cells are flagged for cleavage. Were the process to stop here, the flagged cells would not necessarily yield a smooth progression of size between refinement zones. A smooth size progression is desirable for the flow solver, so after the flagging step, additional processing is done to smooth the refinement list. Two rules are applied iteratively, until the flagged cell list stabilizes.

**Rule 1:** Marked cells in close proximity must clump together. When an un-flagged cell is more than 50% surrounded by flagged cells, it becomes flagged too.

**Rule 2:** No more than a one-level size difference is permitted between neighbor cells. When a size jump is discovered, the coarser neighbor is flagged for splitting.

[1] Mesh tooling from this chapter contributed to:

- Jia, F., Ims, J., Wang, Z. J., Kopriva, J., & Laskowski, G. M. (2018). An evaluation of a commercial and a high order FR/CPR flow solvers for industrial large eddy simulation. In *2018 AIAA Aerospace Sciences Meeting* (pp. 0827)
- Jia, F., Ims, J., Wang, Z., Kopriva, J., & Laskowski, G. M. (2019). Evaluation of second-and high-order solvers in wall-resolved large-eddy simulation. *AIAA Journal*, 57(4), 1636–1648.

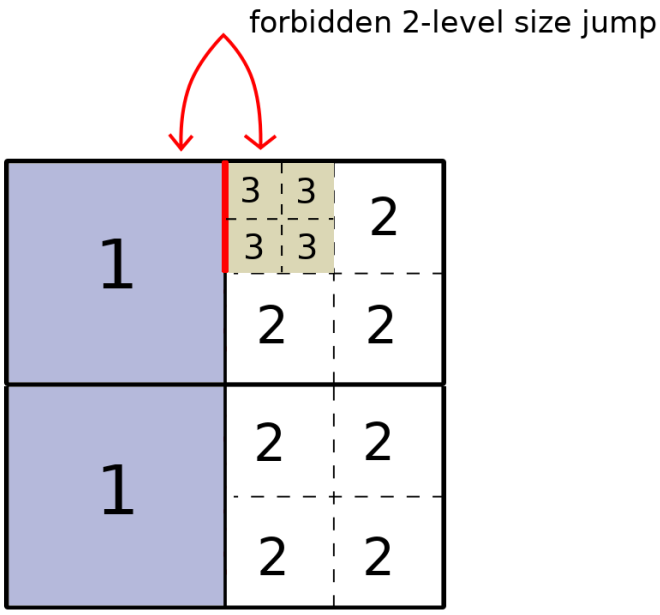



Figure 3.1: Illustration of a forbidden 2-level size jump.

Algorithm Box 1 expresses the full process. The algorithm exports a list of cells that are to be subdivided. Figure 3.1 illustrates a 2-level size jump, forbidden by Rule 2.

Applying the exported list of cell subdivisions is conceptually straightforward, but the process has one tricky element: treatment of the domain boundaries. Ideally, the subdivision of a boundary surface will add geometric detail to the mesh. However, this is easier said than done because the extra geometry information has to come from somewhere other than the mesh itself.

Of course, when the shape of the domain is composed of simple geometric shapes, it is possible to hard code in this information, but in a practical use scenario with 3D meshes, this is no longer possible to do. In that case, the source of information must be the original CAD files. Use of the CAD files adds considerably complexity, transforming what was simple cell subdivision into localized remeshing. Moreover, in a production CFD environment, original CAD files are not always available since the CFD solver doesn't require them.

Without CAD, the mesh surface cannot be enhanced, but its existing shape can still be preserved. Shape preservation is the minimum requirement that a practical mesh subdivider must meet. In some contexts, it may actually be the more desirable goal. Alteration of a domain bound-

**Algorithm 1** Generation of the Mesh Refinement Map

**Require:** Per-cell error values $\mathcal{E}_i$ and a target refinement fraction $T$.

Rank cells by $\mathcal{E}$, largest to smallest.
Mark the top $T$ fraction for refinement.
**repeat until** changes cease. ▷ *Aggregate into refinement zones.*
    **for all** $c \in$ unmarked cells **do**
        Mark $c$ if more than 50% of '$c$'s neighbors are marked.
**repeat until** changes cease. ▷ *Remove sharp size transitions.*
    **for all** $m \in$ marked cells **do**
        **for all** cell faces $f \in m$ **do**
            **if** upon $f$, there's a greater than 1-level refinement mismatch **then**
                Mark the coarsest cell abutting $f$.
**return** Per-cell refinement flags.

ary will inevitably alter the flow field, independent of any boost to solver resolution. When the goal is simply to increase the resolution as in the present research, the boundaries should be left as they are, subdivided but not enhanced.

There are two ways to subdivide while preserving boundary shape. Option #1 is to reposition new boundary cells according to a local extrapolation of the surface geometry, using an approach akin to the program MeshCurve (Ims & Wang, 2019; Ims et al., 2015). Option #2 is to align new cells according to the mesh's own embedded high-order geometry. If the mesh is high-order, option #2 is the better approach because it truly preserves the existing shape. The former option risks corrupting the shape by way of estimation. But the later option is not without risk. The exact preservation of boundary shape combined with an increase in mesh resolution will exacerbate any existing geometry errors in the starting mesh by making them more visible to the flow solver.

In the present work, the boundary shape is preserved by using the high-order geometry of the mesh as a scaffold to place new boundary cells. The calculation is done cell-by-cell. Uniquely curved cut lines are produced to conform to the cell's high-order curved exterior, using the natural coordinate system of the cell.

Figure 3.2 illustrates the natural coordinate system, showing how it makes the calculation simple. The natural coordinate system is relative to the cell's unique geometry. Distance is measured in fractions of arch length while the grid lines follow the cell's shape. With size and shape both

abstracted away, the cell is made straight-sided, non-skewed, and of unit area. For such a regular cell, generating cut lines is easy. In fact, since all cells of common shape map to exact same unit cell, the cut lines can be calculated ahead of time and pre-programmed for efficient reference at run time. The only challenge is to transform the pre-computed cut lines out of the natural coordinate system into physical space, where they take on the desired curvature.

The coordinate transformation is nonlinear. It's done by way of interpolation with the classical finite element formula. Shape functions at the cell's nodes spatially interpolate the $(x, y, z)$ coordinates for the cut lines. The equation is (3.1).

$$\vec{r} = \sum_i \vec{c}_i \phi_i(\eta, \zeta, \xi) \tag{3.1}$$

$\vec{c}_i$ is the position vector of the $i^{\text{th}}$ node in physical space, i.e. its $(x_i, y_i, z_i)$ coordinates. $\phi_i$ is the basis function. $\vec{r}_i$ is the sought after position vector in physical space that corresponds to the natural coordinates $(\eta, \zeta, \xi)$. The formula has no restriction on the type of basis function. This work uses the classic Lagarange basis.

Figure 3.3 is an illustration of the resulting cell division in action. On the figure's left is a strongly curved high-order hexahedral cell. On the right is the resulting mesh after two rounds of uniform refinement, with cut lines placed at the midpoint for each spatial dimension. As the rightmost image makes clear, the new sub-cells are following the boundary curves of the original parent cell, preserving its shape.

## 3.2 Hanging Nodes

When a hexahedral or quadrilateral cell is cut into pieces but its neighbor is *not* into pieces, then a hanging node is produced between the cells. The hanging node breaks the one-to-one correspondence between the left and right neighbor faces. The FR/CPR calculations of the flow solver (see equations (2.5) and (2.7)) depend upon the one-to-one match between neighbor faces. When hanging nodes exist, the flow solver is no longer able to calculate fluid motion variables upon the

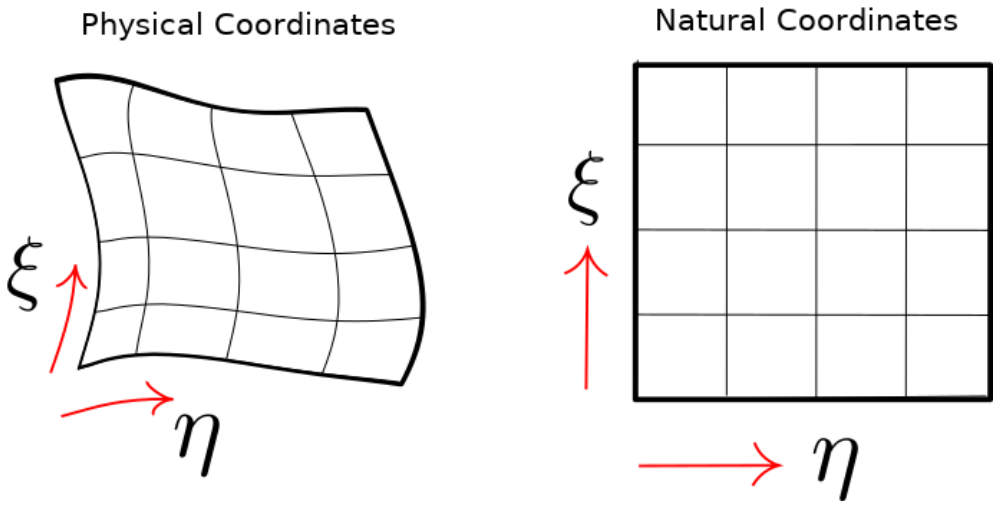



Figure 3.2: The physical coordinate system versus the natural coordinate system.

cell interface.

The destructive presence of hanging nodes is a major impediment to the version of mesh refinement so far described. It prevents the generated meshes from being used in subsequent CFD calculations. To make hanging nodes viable with FR/CPR, it is necessary to bridge the gap between the non-corresponding cells, using some type of interpolation. In this way, the un-split neighbor may be supplied with an interface that is unbroken while the opposing conglomeration of subcells can exist and operate normally.

The chosen interpolation method here is the mortar method (Shi et al., 2016; Kopriva, 1996), illustrated by Figure 3.4. At the hanging node juncture, a bridging "mortar" face is inserted, which fuses the sub-faces from the broken side to match the unbroken side. All FR/CPR calculations that would normally occur upon the interface happen upon the mortar face, at the higher of the two discontinuous resolutions. Input data is sourced from abutting cells, copied directly from the higher resolution side and interpolated from the lower resolution side. Output data transfers back by a direct copy to the higher resolution side and by an $L_2$ projection down to lower resolution side. Figure 3.5 shows the viability of the computational technique with a velocity field snapshot from an FR/CPR simulation using mortar method.

Two of the pieces of the input data required by the FR/CPR calculation are the pointwise normal

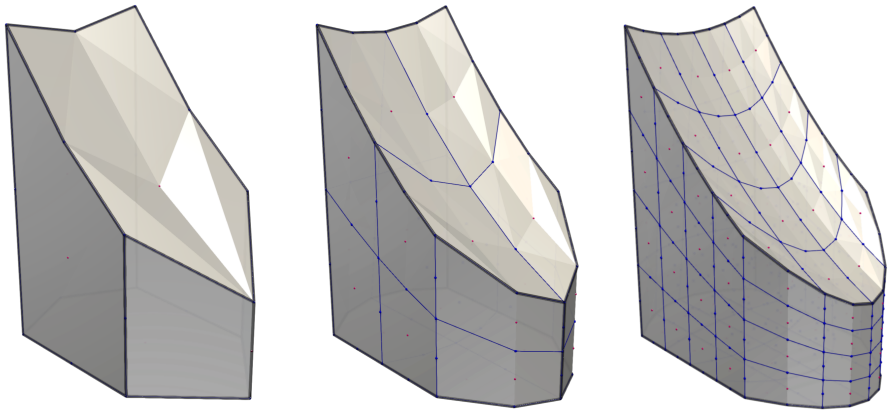


Figure 3.3: Curvature preserving cell division. On the left is a strongly curved single high-order hexahedral cell. To its right are two levels of uniform refinement, showing how the cell division process is able to preserve the boundary curvature.

vectors and the pointwise area vectors from the interface. These are required for variables $F^n_{f,l}$ and $S_f$ in equation (2.7). A subtle quirk of the mortar calculation is that these geometry vectors must source from the lower resolution side when the flow solver is operating at polynomial order $\mathcal{P}_1$. However, when the flow solver is operating at higher polynomial orders, better results are achieved by sourcing the geometry vectors from the higher resolution side. This quirky requirement is due to the limited spectral bandwidth available to the flow field at $\mathcal{P}_1$. A $\mathcal{P}_1$ flow field upon the low resolution side is linear across the cell, so it cannot appropriately respond to a hanging node "corner" upon the edge. Even though the "corner" is supposed to be perfectly flat, roundoff error will make it slightly pointed, resulting in an unstable calculation if the geometry vectors are allowed to reveal the hanging node's existence.

## 3.3 Computer Implementation

The flow solver used in this work is hpMusic, the high-order FR/CPR flow solver developed and maintained by the Computational Thermal-Fluids Laboratory at the University of Kansas (Bhaskaran et al., 2017; Wang et al., 2017). The error estimators were implemented as add-on modules to the program, executing their calculations as part of the flow solver's time-stepping

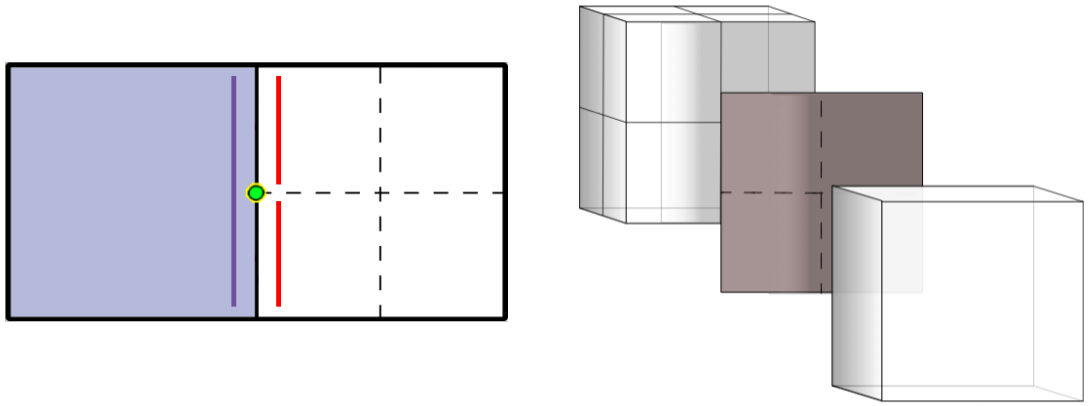


Figure 3.4: Hanging nodes exist at the juncture between refined and unrefined cells, which require special treatment in the flow solver. At such junctures, the flux calculations are done on a virtual "mortar face," using projected data from the abutting cells Shi et al. (2016); Kopriva (1996).

sequence. The cell flagging sequence (algorithm box 1) was added to the program as a post-processing step. Cell-splitting and flow-field interpolation from old mesh to new was implemented as an independent utility program.

Figure 3.6 presents the program structure and execution sequence for the full system. The key calculation steps, added for this work, are colored in green. The flow solver takes as input a configuration file (red), a mesh (blue), and—optionally—an initializing flow field (purple). The flow solver exports, upon its completion, the flow field restart file, and the list of cells marked for splitting (yellow). The list of flagged cells, the starting mesh, and the flow field restart file move to program 2, the mesh adapter. Here, the cells are split and the flow field is interpolated from old mesh to new mesh. The new mesh and new flow field may then be used to restart the flow solver where it left off.

All programs were written in C++11, with distributed parallelization via MPI and localized parallelization via threading and SIMD vectorization via openmp (OpenMP Architecture Review Board, 2015).

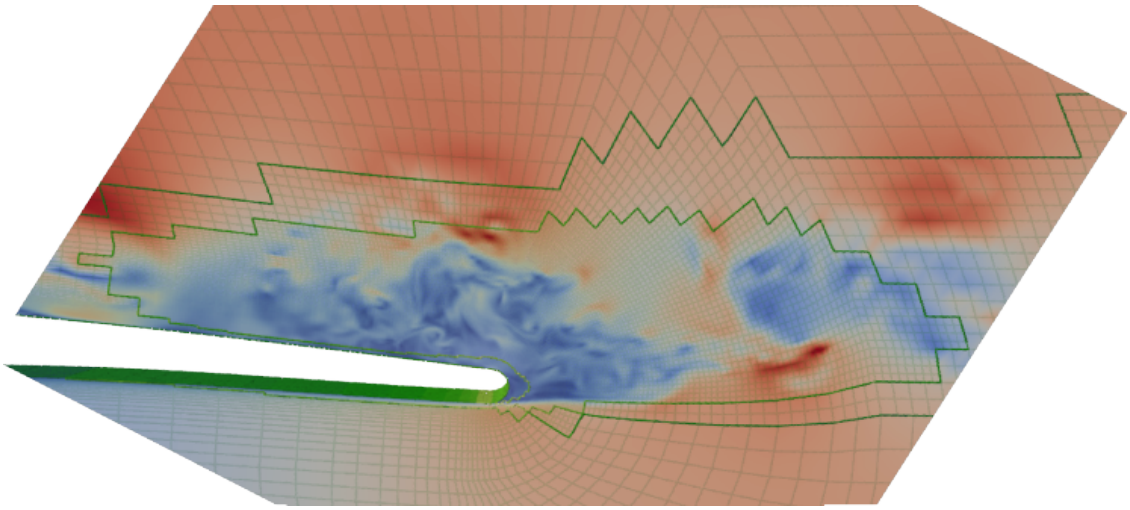


Figure 3.5: Velocity field across hanging node interfaces, marked by thick green lines. At these interfaces, the mesh resolution changes discontinuously. To bridge the resolution gap, the mortar method is used. The resulting flow field is realistic and the simulation is stable.

### 3.3.1 Implementation of the Error Estimators

The error estimators are built utilizing C++ class inheritance. Each estimator is implemented as a child of an abstract base class `ErrEst`. The base class provides function placeholders and data structures common to all of the error estimators:

- data structure initialization at boot
- loading saved state from disk
- saving state to disk
- accumulation of quantities at each iteration of the flow solver
- a post-processing calculation
- pointers to the flow solver's own data structures, through which the flow field information may be retrieved

The individual error estimators inherit from `ErrEst`, supplying code for the virtual functions and for any functionality unique to the given error estimator. The class for error indicator ${}^{\mathbf{T.L.err}}\mathcal{E}$,

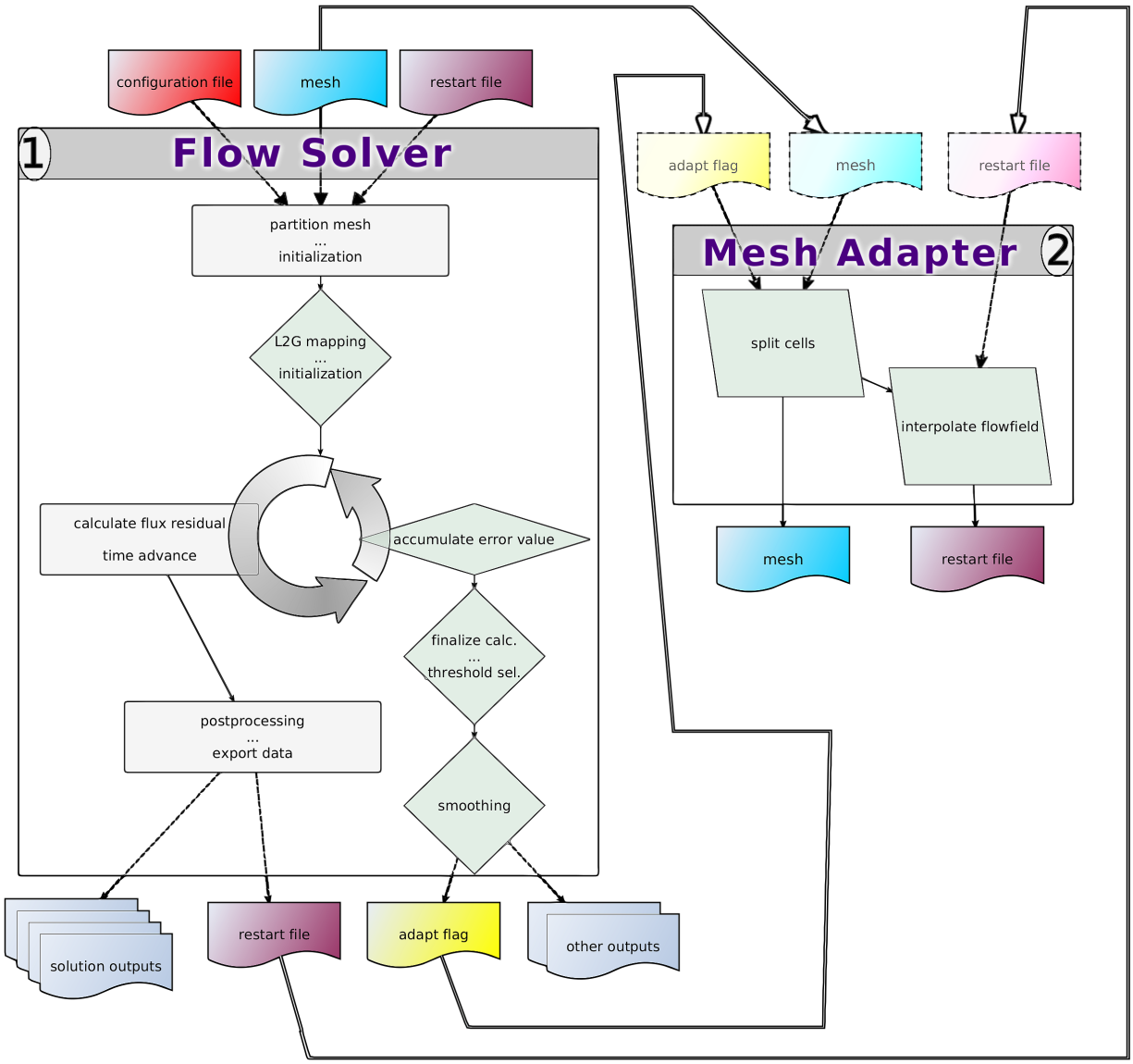



Figure 3.6: Program structure and execution sequence.

described in subsection 2.2.3, has a special (vectorized) function to compute $\frac{\partial^2 \vec{u}}{\partial \xi^2}$ from the conservative flow field variables $[\rho, \vec{m}, E]^{\mathrm{T}}$.

The estimators are fully parallel. Just like the flow solver itself, the error estimators execute distributed over MPI CPUs, one copy per mesh partition, as shown by Figure 3.7. A second layer of parallelization exists within the distributed copies. Upon each CPU, the error estimators utilize openmp threading and SIMD vectorization.

Parallelized file I/O is implemented at the solver level by way of functions that wrap the commands `MPI_Gatherv` and `MPI_Scatterv`. The call to function `save` automatically invokes machinery to make the distributed error estimators synchronize, sending data to the root processor, where it is rearranged according to the cell sequence in the un-partitioned mesh before export to disk. The `load` function follows the same procedure in reverse. Because the data is always rearranged to match the un-partitioned mesh, the files are independent of the number of MPI pro-

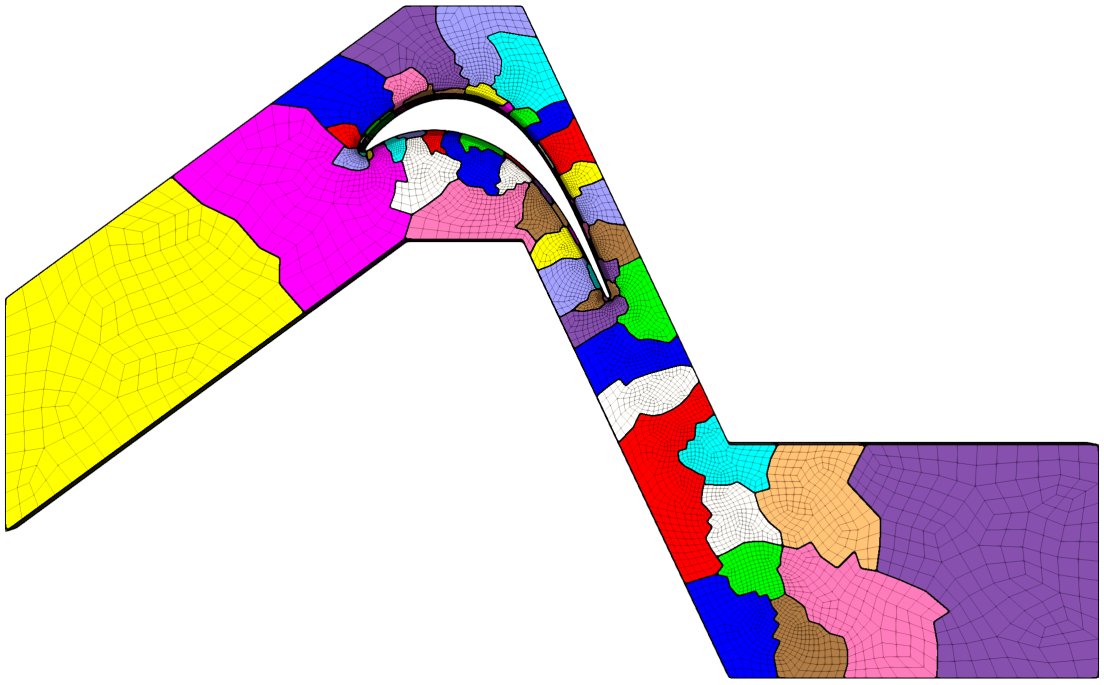


Figure 3.7: Like the flow solver itself, the error estimators are highly parallelized with MPI. Each MPI processor has its own instance of the C++ error estimator object, with dominion over the assigned mesh partition.

cessors. Thus it is possible for the program to be halted and restarted on a different number of CPUs. The serialized files have another benefit too. They are usable in the mesh adapter program, which is serial.

The mesh adapter is basically a series of `for` loops, proceeded and followed by functions to read and write both the mesh file and the saved flow field. The `for` loops cycle over the mesh's cells one-by-one, modifying each according to instructions exported by the error estimator. The central calculation is a matrix multiplication, which maps the pre-computed cell cut lines from natural coordinates to physical coordinates. A second matrix multiplication interpolates the flow field from old mesh to new, cell-by-cell.

# Chapter 4

# Test Cases[1]

## 4.1 T106c

The first test case for the error estimators is T106c, a benchmark LES challenge problem featured at both the 4th and the 5th International Workshops on High-Order CFD Methods (Hillewaert & JS., 2016, 2018). This simulation models air flowing through an angled turbine blade. The simulation has several characteristics to recommend its use as a test case:

- **There is a readily-available DNS simulation to compare against, simulated by the University of Kansas's Computational Thermo-Fluids Laboratory.**
- **It has the appropriate degree of complexity.**

  It is a realistically complex problem, reminiscent of turbo-machinery simulations that might be conducted in an industrial setting. Yet it is also simple enough for the flow field to be dissected and analyzed.
- **It features a mix of flow conditions.**

  The incoming air is laminar, but the flow transitions to turbulent as it passes over the turbine blade. Modeling the flow transition is a challenge for the CFD solver.

[1] Portions of this chapter report results from:

- Ims, J. & Wang, Z. J. (2022). A comparison of three error indicators for adaptive high-order large eddy simulation. In *AIAA SciTech 2022 Forum* (pp. 1201)
- Ims, J. & Wang, Z. J. (under review). A comparison of three error indicators for adaptive high-order large eddy simulation. *(submitted to) Journal of Computational Physics*.

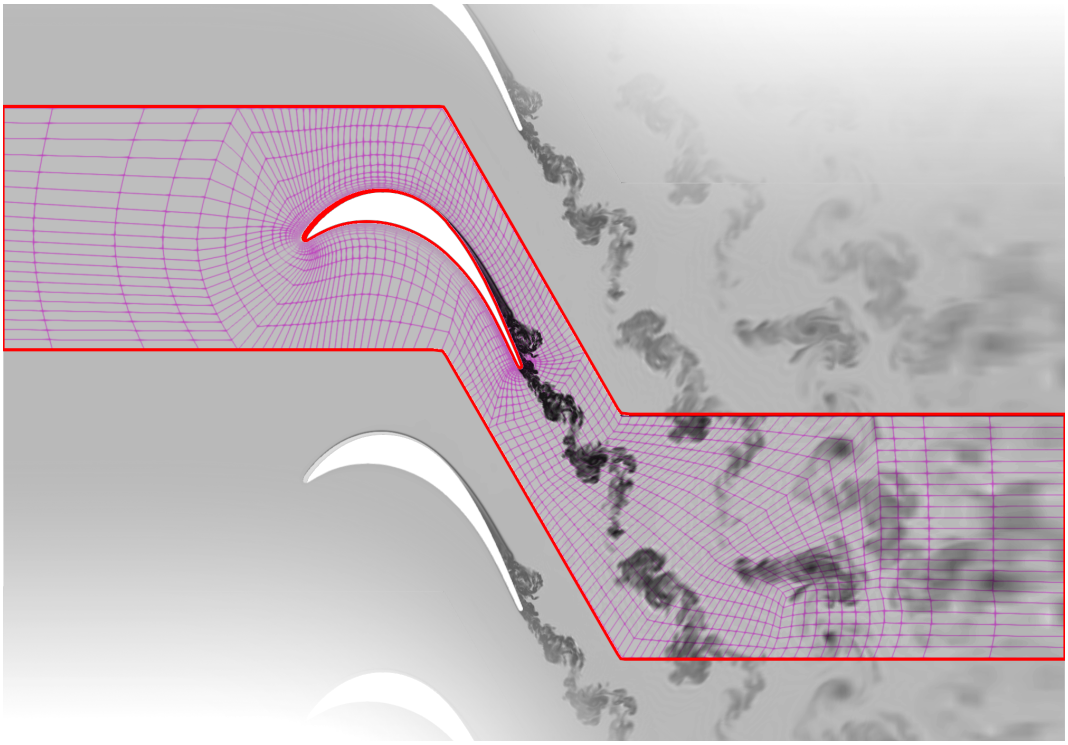


Figure 4.1: The simulation domain for the T106c turbine blade.

Figure 4.1 illustrates the geometry of the T106c flow field. The simulation domain consists of the highlighted part of the image, encircling a single blade from an infinite stack, produced with periodic boundary conditions above and below in the y-direction. All elements of the geometry are spaced relative to the blade's chord length $C$. The vertical distance between the stacked blades is $0.95C$. The spanwise depth (in to & out of the page) is $0.1C$, but periodic boundary conditions are also depth-wise to give the illusion that the blade is infinitely wide. The incoming airflow is angled upward at 32.7° with a state sufficient to produce exit isentropic Mach and Reynolds numbers of $M_{is} = 0.65$ and $Re_{is} = 80,000$, respectively, at scale $C = 1$ m.

The test pits the three primary error estimators ($^{\mathbf{smth}}\mathcal{E}$, $^{\mathbf{unStd}}\mathcal{E}$, and $^{\mathbf{T.L.err}}\mathcal{E}$) against each other, competing to improve simulation quality. The error estimators will begin with a coarsened mesh for T106c, topologically similar to the mesh used in a published DNS simulation of T106c by Alhawwary & Wang (2019). The DNS flow field was simulated at polynomial order $\mathcal{P}_3$, but the error estimators will run at $\mathcal{P}_2$, exacerbating the loss of resolution from the coarsened mesh. After executing upon the coarsened mesh, each error estimator will be permitted two rounds of

H-adaptation to improve the resolution. At each round of H-adaptation, the refinement budget will be 20%; that is, at every round, the error estimators will each select the 20% most erroneous cells to subdivide. After each H-adaptation, the quality of the simulation will be gauged by its comparison against the DNS simulation. Examination will also be made of the cell refinement pattern, elucidating the characteristic behaviors of the error estimators.

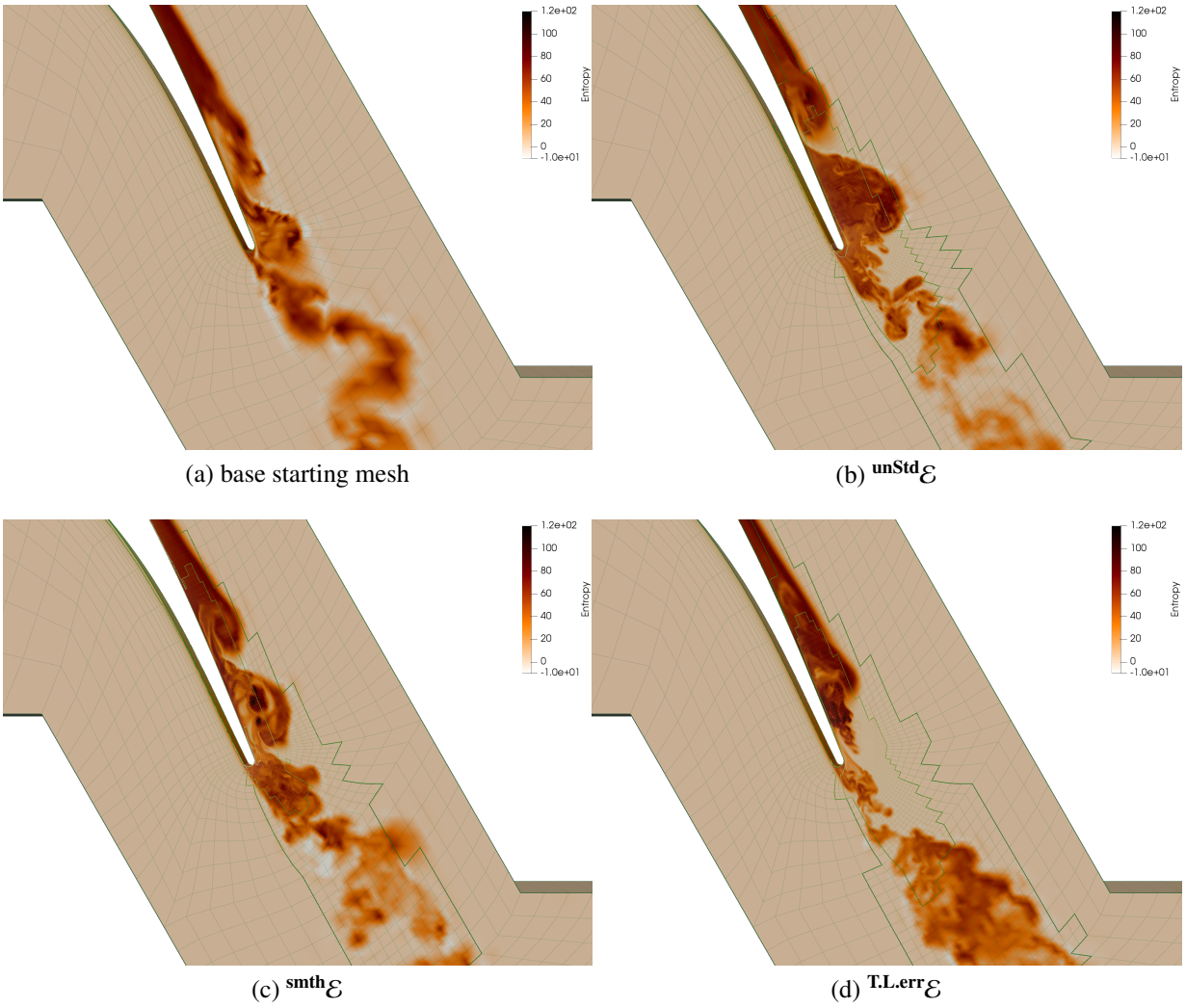
### 4.1.1 Visual Quality Assessment

As a visual basis for the flow field quality improvement produced by the error estimators, Figure 4.2 contrasts ending snapshots of the turbulent wake, immediately aft of the turbine blade. The images render the entropy increase relative to the far field, making the vortex pattern visible. In each image, the underlying mesh has been superimposed on the flow field, with thick lines denoting the hanging node boundaries. An examination of the images reveals the first two items of interest pertinent to the planned analysis. 1) All of the error indicators chose to refine both near the blade surface and also within the more active parts of the turbulent wake. 2) All three error indicators were successful at boosting the vortex resolution, uncovering detail not present in the starting simulation upon the base mesh.

### 4.1.2 The Mesh Refinements

Figure 4.3 directly contrasts the mesh refinements themselves, using a color code to show which cells were marked at each round of refinement. Blue denotes cells that were marked upon the base mesh. Yellow denotes cells that were marked upon the once-refined mesh. Orange denotes cells that were marked upon the twice-refined mesh. Zoomed views of the refinements are provided by Figures 4.3, 4.4, and 4.5.

For correct interpretation of the figure, it is critical to remember that the refinement budget is quantified in terms of cell quantity (i.e. the 20% most erroneous cells), not cell volume. Accordingly, the size of the colorized regions does not communicate how many cells were refined. It communicates how large those cells were. This fact is key to spotting the major difference between




(a) base starting mesh

(b) $^{\mathbf{unStd}}\mathcal{E}$

(c) $^{\mathbf{smth}}\mathcal{E}$

(d) $^{\mathbf{T.L.err}}\mathcal{E}$

Figure 4.2: Visual quality comparison of the T106c turbulent wake after refinement #2. The vortices have been made visible by rendering the instantaneous entropy increase relative to the far field.

the error indicators, namely that some of the indicators preferentially targeted small cells near the blade ($^{\mathbf{smth}}\mathcal{E}$) while others preferentially targeted large cells in the wake ($^{\mathbf{T.L.err}}\mathcal{E}$). Likewise, the shrinkage in color areas at later refinement steps does not indicate a reduction in the number of flagged cells. It indicates that some cells were split multiple times into tiny fragments.

The figure shows that each of the error indicators traced out an arced path from the suction side of the blade, starting from the point of flow separation and following the detached stream into the turbulent area at the rear of the blade. In the wake behind the blade, the error indicators made similar resolution requests. First, they marked cells in the general area around and downstream of the turbulence, following the vortex train. Second, they marked cells at the center of the turbulent zone, where the whirling motion was greatest. Third, they marked cells in two disconnected areas: 1) deep within the turbulent zone, and 2) just outside the region marked in step #2. This second action hints that the consequence of refinement #2 may have been an enhancement of the vortices, streaming from the blade, through the turbulence zone and out the opposite side.

An alternative view of the mesh refinement is presented by Figures 4.6 through 4.8. These figures graph $y^+ = \frac{yu_\tau}{\nu} = \frac{y}{\nu}\sqrt{\frac{\tau_w}{\rho}}$ of the first cell layer as a function of distance along the turbine blade. ($y$ is the distance to the wall; $\nu$ is the kinematic viscosity; and $u_\tau = \sqrt{\frac{\tau_w}{\rho}}$ is the friction velocity, with $\tau_w$ being the wall shear stress and $\rho$ the fluid density.) Horizontal axis coordinate $X = 0$ corresponds to the leading tip of the blade. Horizontal coordinates $X = \pm 1$ correspond to the blade's tail. Positive $X$ refers to the blade's suction side. Negative $X$ refers to the pressure side. $y^+$ is a non-dimensionalized coordinate, measuring relative distance into the flow boundary layer, so the plots show the level of boundary resolution provided by the flow simulation and the effect of the error indicators. (Smaller $y^+$ is better.)

The plots reveal a clear difference between the indicators. Figure 4.6 shows indicator $^{\mathbf{smth}}\mathcal{E}$ lowered $y^+$ across most of the surface. In contrast, indicator $^{\mathbf{T.L.err}}\mathcal{E}$ (Figure 4.8) left the original $y^+$ mostly unaltered, adding resolution only to the upper rear of the blade. Indicator $^{\mathbf{unStd}}\mathcal{E}$ (Figure 4.7) took a hybrid approach. It lowered $y^+$ at the rear of the blade and also in an upstream span from $X/C_{\text{axial}} \approx 0.15$ to $X/C_{\text{axial}} \approx 0.525$.

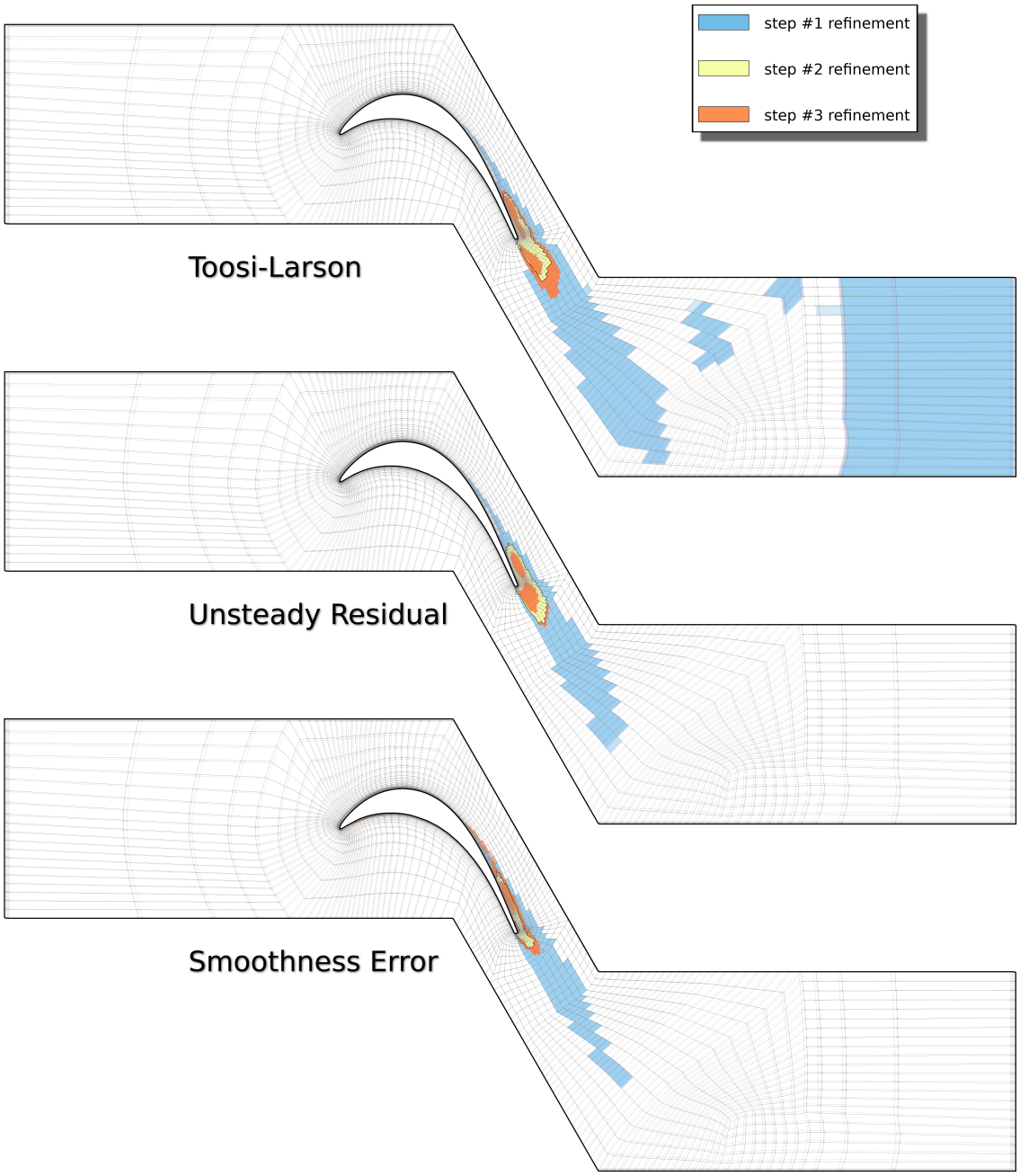



Figure 4.3: Comparison of the mesh refinement requested for simulation T106c. With each error indicator, the simulation was executed three times in sequence, starting with the same base mesh. Blue marks cells that were were flagged for refinement round #1. Yellow marks cells that were flagged for refinement after round #2. Orange marks cells that were flagged after round #3. All error indicators were granted the same refinement budget: the 20% most erroneous cells, plus as many additional cells as it takes to meet the two smoothness requirements.

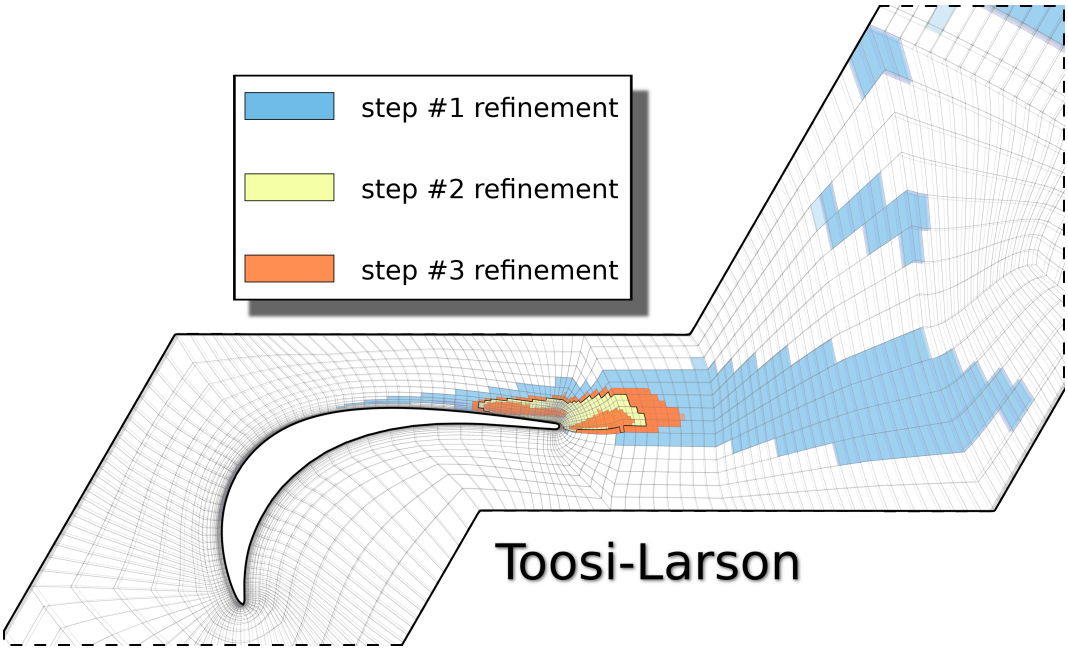



Figure 4.3: Zoomed view of the cells flagged from simulation T106c by $^{\mathbf{T.L.err}}\mathcal{E}$.

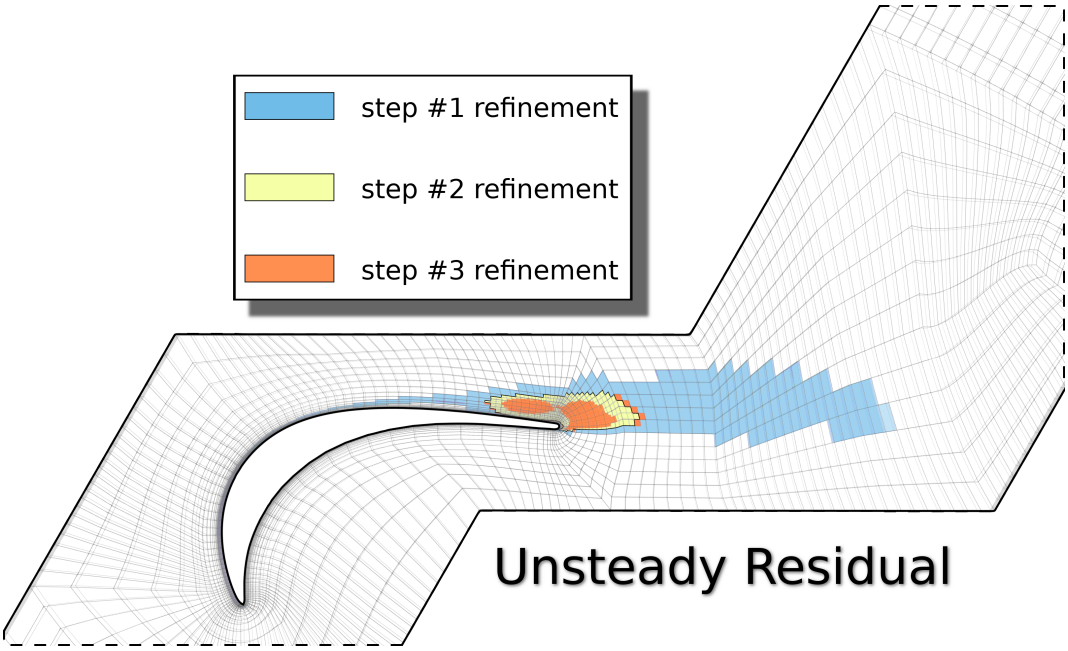



Figure 4.4: Zoomed view of the cells flagged from simulation T106c by $^{\mathbf{unStd}}\mathcal{E}$.

What's special about that upstream area? Well, flow detachment occurs at roughly $X/C_{\text{axial}} \approx 0.1$, so error indicator ${}^{\mathbf{unStd}}\mathcal{E}$ was evidently trying to enhance the resolution of the departing stream and the point of departure. ${}^{\mathbf{smth}}\mathcal{E}$ did similar, starting its own refinement at $X/C_{\text{axial}} \approx 0.08$, but whereas ${}^{\mathbf{smth}}\mathcal{E}$ simply refined everything downstream of the detachment point, ${}^{\mathbf{unStd}}\mathcal{E}$ broke off early, picking up again where the detached jet reconnected, circa $X/C_{\text{axial}} = 0.8$. Unlike the other two error indicators, ${}^{\mathbf{T.L.err}}\mathcal{E}$ did not enhance the surface resolution except for the tail of the blade, inside the turbulent zone. Does that mean that error indicator ${}^{\mathbf{T.L.err}}\mathcal{E}$ failed to detect the point of flow detachment and reattachment? No. When Figure 4.8 is put in the broader context of the surrounding cell refinements, shown by Figure 4.3, it is possible to see that ${}^{\mathbf{T.L.err}}\mathcal{E}$ was flagging off-surface cells near the same locations that ${}^{\mathbf{unStd}}\mathcal{E}$ and ${}^{\mathbf{smth}}\mathcal{E}$ were flagging on-surface cells.

In summary, all three error estimators detected the critical points of the flow field, but in refining the mesh, they ascribed differing importance to the adjoining cells. ${}^{\mathbf{smth}}\mathcal{E}$ boosted resolution of nearby WALLs. ${}^{\mathbf{T.L.err}}\mathcal{E}$ boosted resolution of nearby vortex paths. ${}^{\mathbf{unStd}}\mathcal{E}$ did both.

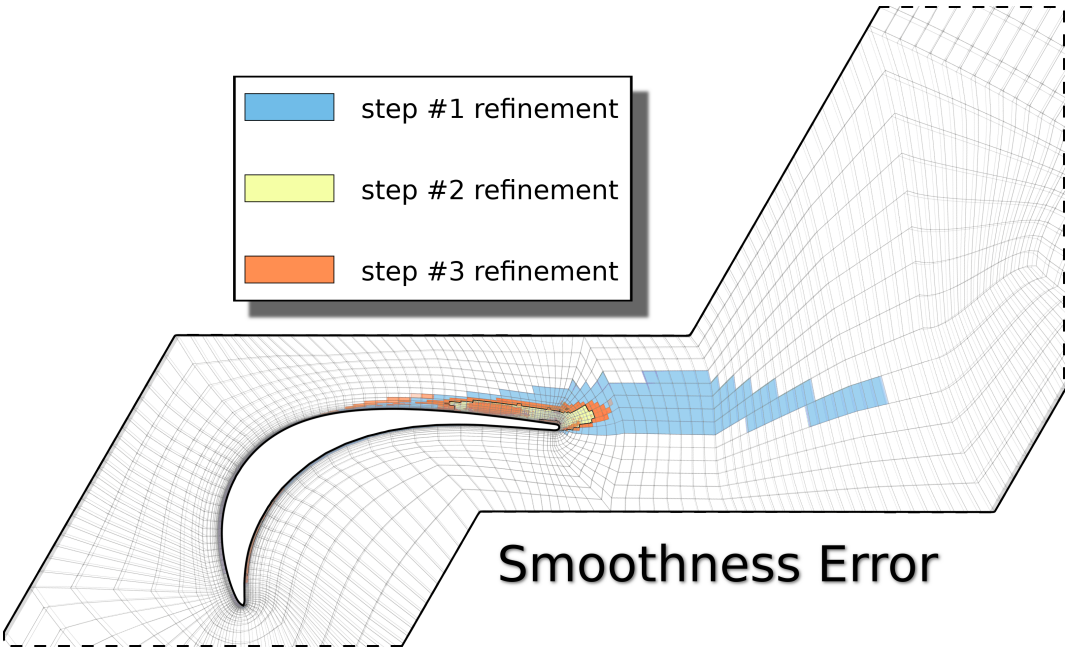



Figure 4.5: Zoomed view of the cells flagged from simulation T106c by ${}^{\mathbf{smth}}\mathcal{E}$.

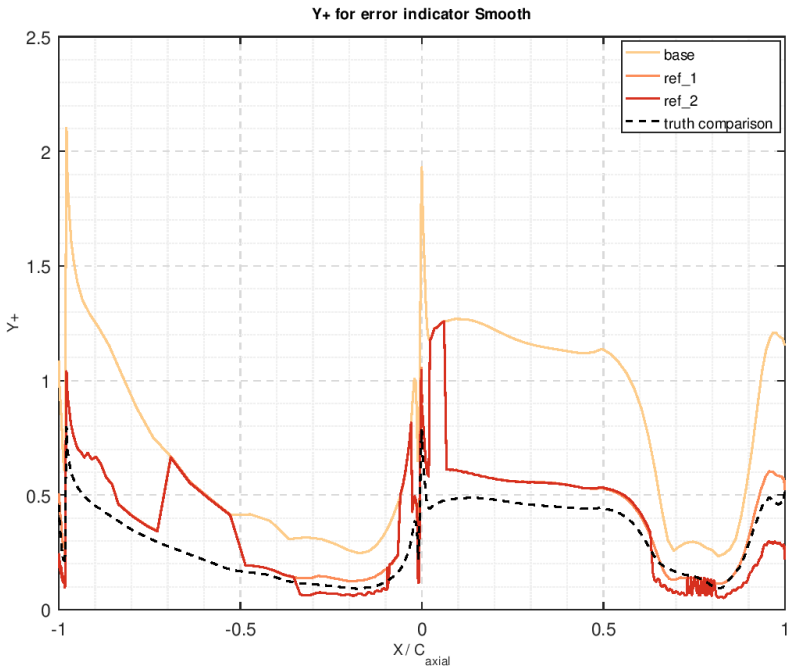



Figure 4.6: Progression of $y^+$ for ${}^{\mathbf{smth}}\mathcal{E}$

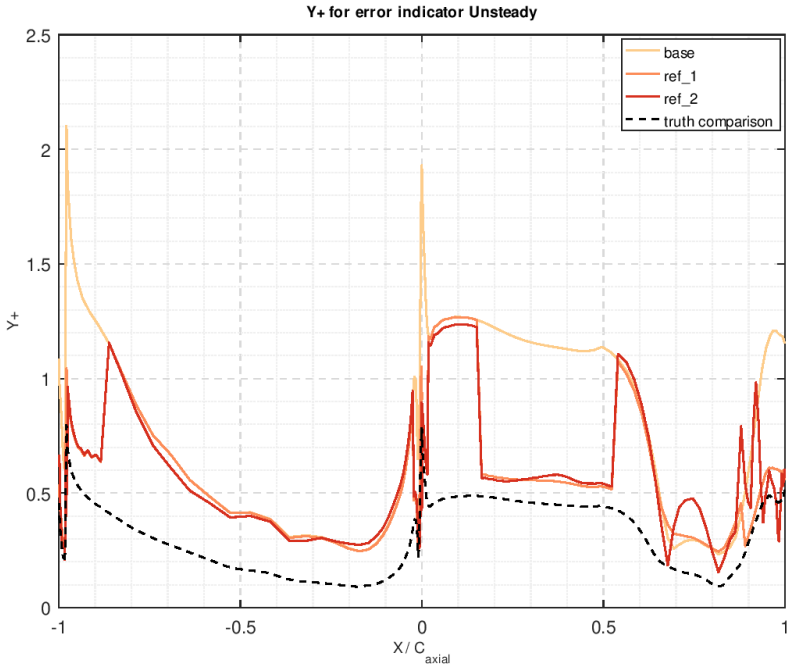



Figure 4.7: Progression of $y^+$ for ${}^{\mathbf{unStd}}\mathcal{E}$

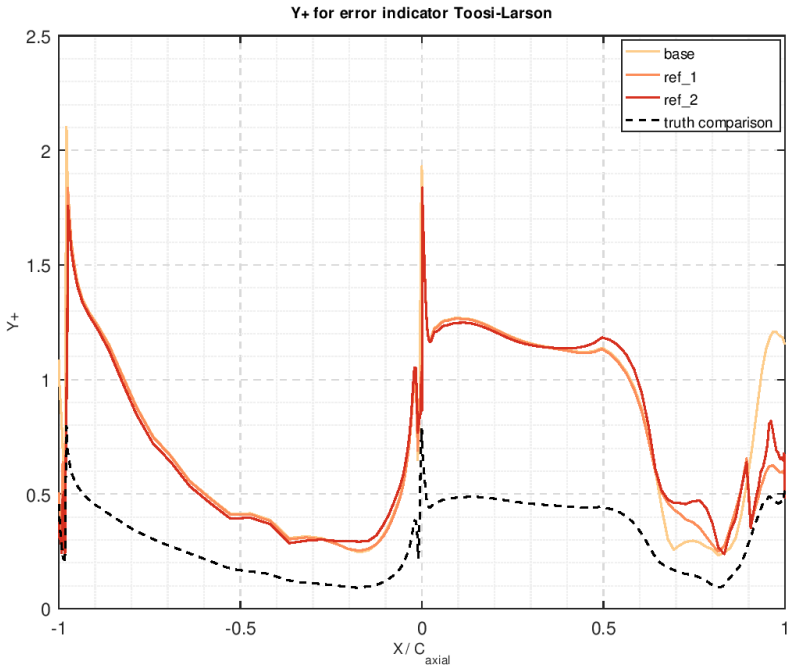



Figure 4.8: Progression of $y^+$ for ${}^{\mathbf{T.L.err}}\mathcal{E}$

### 4.1.3 Force Analysis: lift, drag, and pressure

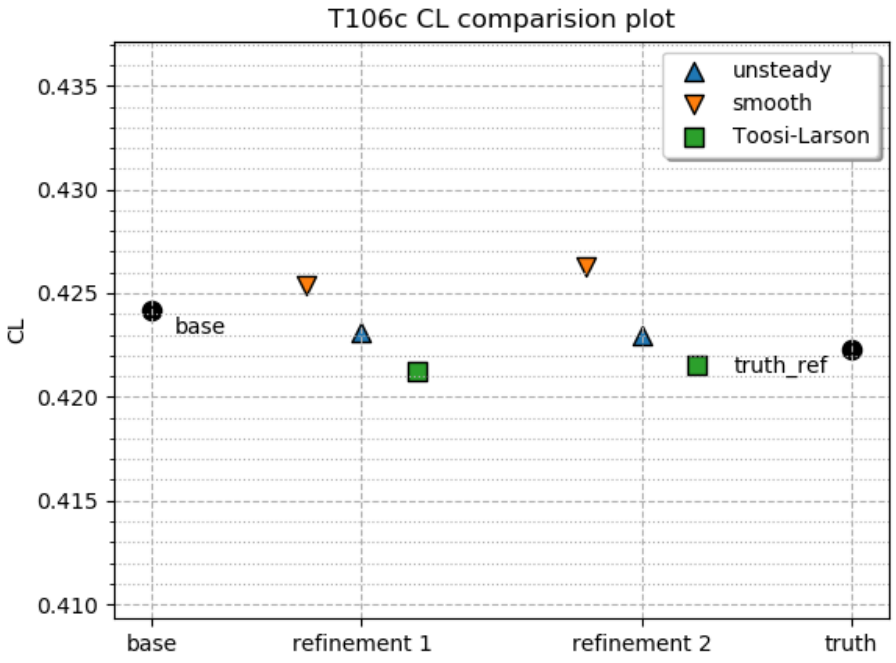
To assess the impact of the increasing mesh resolution, we first look at the force coefficients, starting with lift and drag.

Figures 4.9a and 4.9b plot the progression of the lift and the drag coefficients while Tables 4.1a and 4.1b list the plotted numbers. These coefficients are time-averaged quantities, with statistical uncertainty due to the turbulent fluntuation in the time domain. The tables provide $1\sigma$ statistical bounds on the means. These are calculated in the typical fashion (e.g. sample standard deviaiton divided by the square root of the number of measurements) with a correction factor for the short-term temporal correlation of the time sampling. The correction procedure used an $AR(1)$ autoregressive model and is documented in Donath (2020).

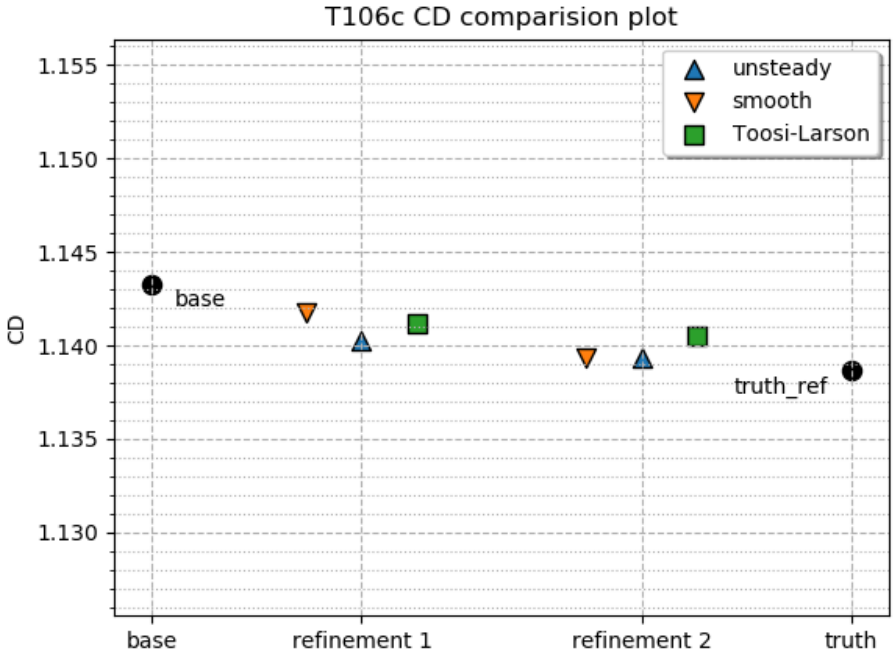
From the plots and the tables, it appears that both ${}^{\mathbf{unStd}}\mathcal{E}$ and ${}^{\mathbf{T.L.err}}\mathcal{E}$ were successful at driving the lift and the drag coefficients toward truth. ${}^{\mathbf{smth}}\mathcal{E}$ also drove $C_D$ toward truth, but with $C_L$, it seems to have slightly overestimated, and it is not clear that the progression is converging to the correct value. The discrepancy is not large, and it is within the error bars, so the failure may not actually indicate a problem, but it does hint that ${}^{\mathbf{smth}}\mathcal{E}$ may not be as performant as the other two error estimators.

For the next assessment, we look at the pressure distribution along blade. Figure 4.10 plots the pressure coefficient, one error indicator per row. The center plots cover the whole blade, while the left and right columns show zoomed views. The x-axis is set up just as in the earlier $y^+$ plots. $X/C_{\text{axial}} = 0$ corresponds to the blade tip and $X/C_{\text{axial}} = \pm 1$ corresponds to the blade's tail. The suction side of the blade is to the right (positive x) and the pressure side is to the left (negative x).

The pressure distribution on the base mesh is already close to the truth solution, but all error estimators managed to improve it by providing increased spatial sampling in the most unstable regions, such as the underside of the blade's tail. Another subtle improvement is the position of the dips and peaks. A feature that did *not* noticeably improve is the amplitude of the profile. The amplitude matches well with DNS for *non*-turbulent areas of the surface, but the amplitude is over or under estimated upon the turbulent portions of the blade. ${}^{\mathbf{T.L.err}}\mathcal{E}$ and ${}^{\mathbf{unStd}}\mathcal{E}$ got closest to the




(a) Lift coefficients for T106c




(b) Drag coefficients for T106c

Figure 4.9: Progression of the lift and drag coefficients for simulation T106c

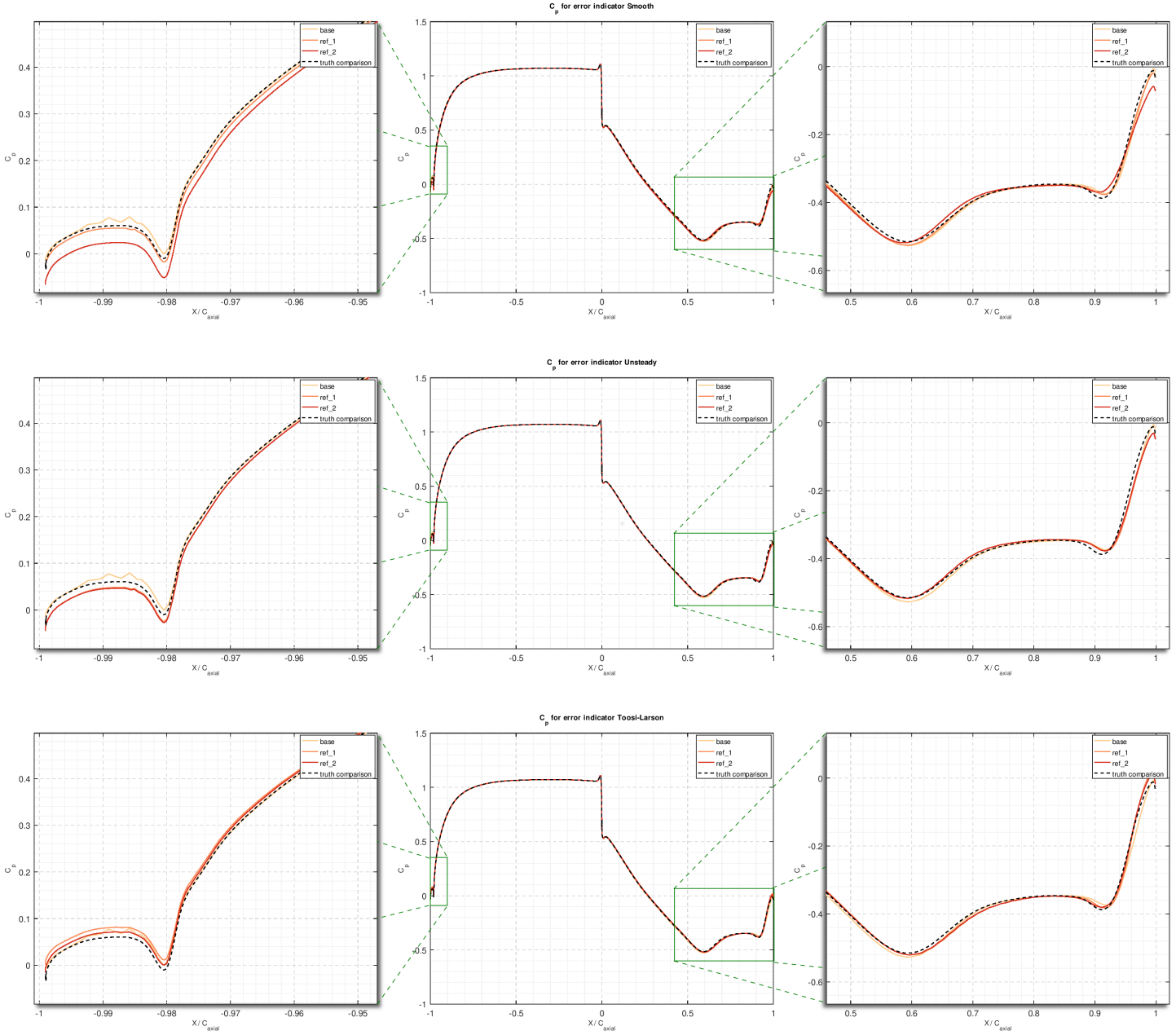



Figure 4.10: Progression of the pressure coefficients. $^{\mathbf{smth}}\mathcal{E}$ is at top. $^{\mathbf{unStd}}\mathcal{E}$ is in the middle. $^{\mathbf{T.L.err}}\mathcal{E}$ is at bottom.

| | $C_L$ common | $C_L$ for $^{\mathbf{smth}}\mathcal{E}$ | $C_L$ for $^{\mathbf{unStd}}\mathcal{E}$ | $C_L$ for $^{\mathbf{T.L.err}}\mathcal{E}$ |
|---|---|---|---|---|
| base mesh | 0.4242 ± 0.002 | | | |
| refinement #1 | | 0.4254 ± 0.003 | 0.4231 ± 0.003 | 0.4212 ± 0.005 |
| refinement #2 | | 0.4262 ± 0.003 | 0.4239 ± 0.004 | 0.4225 ± 0.005 |
| truth ref. | 0.4223 ± 0.005 | | | |

(a) T106c lift coefficient

| | $C_D$ common | $C_D$ for $^{\mathbf{smth}}\mathcal{E}$ | $C_D$ for $^{\mathbf{unStd}}\mathcal{E}$ | $C_D$ for $^{\mathbf{T.L.err}}\mathcal{E}$ |
|---|---|---|---|---|
| base mesh | 1.1433 ± 0.003 | | | |
| refinement #1 | | 1.1417 ± 0.004 | 1.1402 ± 0.005 | 1.1412 ± 0.007 |
| refinement #2 | | 1.1393 ± 0.005 | 1.1393 ± 0.006 | 1.1405 ± 0.007 |
| truth ref. | 1.1387 ± 0.008 | | | |

(b) T106c drag coefficient

Table 4.1: progression of the lift coefficient (table 4.1a) and the drag coefficient (table 4.1b) for simulation T106c

correct amplitude, with the former slightly overestimating and the later slightly underestimating. Curiously, $^{\mathbf{smth}}\mathcal{E}$, at refinement round #1, was even closer to the correct amplitude, but it moved away at refinement #2.

### 4.1.4 Turbulence Analysis: Reynolds stress and energy spectra

Figure 4.12 shows how the mesh refinement affected the simulations' accuracy of the turbulence, using the uv cross term of Reynolds stress tensor as a proxy. The plots are grouped by row, one error indicator along each, with the steps of refinement increasing left to right. $^{\mathbf{unStd}}\mathcal{E}$ occupies the first row, subfigure 4.12a. $^{\mathbf{smth}}\mathcal{E}$ occupies the second row, subfigure 4.12b. And $^{\mathbf{T.L.err}}\mathcal{E}$ occupies the third row, subfigure 4.12c. The base mesh is at the left (identical down the column). Refinement #1 is in the middle. And refinement #2 is at the right. The final row, subfigure 4.12d, holds the DNS truth solution.

This collection of plots is particularly revealing, showing not only a universal quality improvement from all of the indicators but also a more definitive superiority of some relative to others, judged on the basis of how quickly and accurately they conformed to the DNS truth. $^{\mathbf{smth}}\mathcal{E}$ lagged

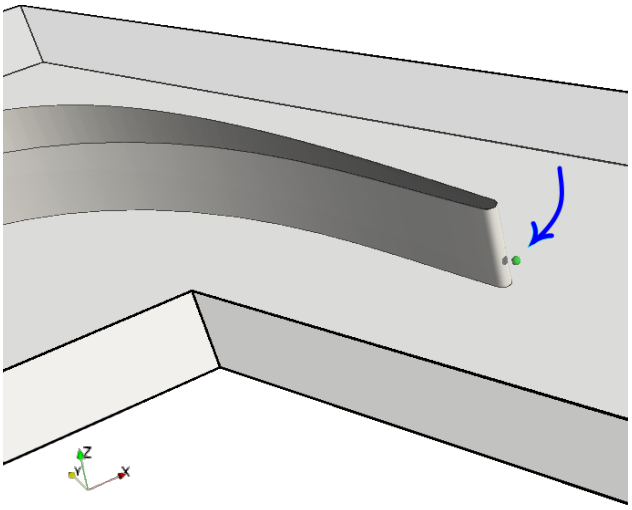



Figure 4.11: The monitor point (green dot) for the pressure and turbulent kinetic energy spectra.

behind the others at both the first and the second refinements, ending with the poorest match to DNS. Meanwhile, both ${}^{\mathbf{unStd}}\mathcal{E}$ and ${}^{\mathbf{T.L.err}}\mathcal{E}$ converged rapidly, nearly in lockstep. A very close examination of their final plots shows that ${}^{\mathbf{T.L.err}}\mathcal{E}$ slightly bested ${}^{\mathbf{unStd}}\mathcal{E}$ near the most downstream point of the blade and along the trailing upper surface.

As a final assessment, we turn to the frequency spectra of the pressure and the turbulent kinetic energy, fluctuating in the turbulent wake immediately behind the blade. Figure 4.11 shows the location of the monitor point relative to the blade.

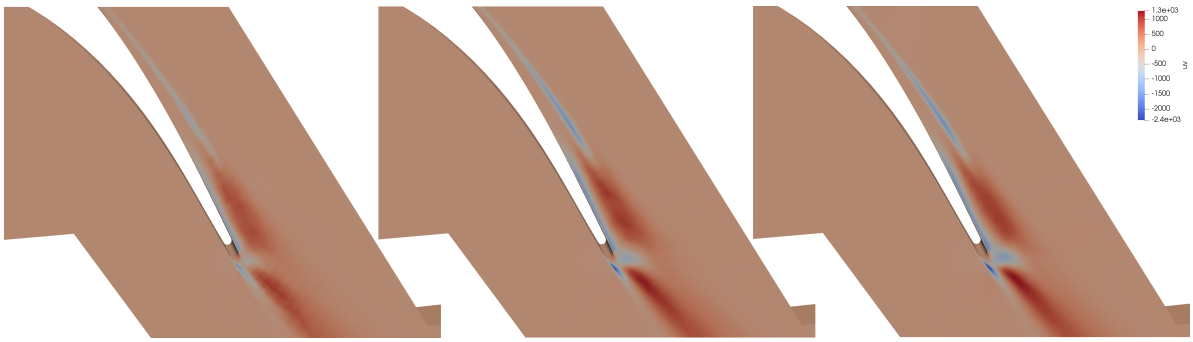
Figures 4.15 through 4.17 plot the power spectrum of the pressure fluctuation. Figures 4.18 through 4.20 plot the power spectrum of the turbulent kinetic energy. In these graphs, the dashed line denotes the theoretical slopes of the energy cascade, and the black line marks the spectrum of the DNS flow field. Each set of plots show the progression from base mesh (Figures 4.15 & 4.18 ) to the final mesh after two rounds of refinement (Figures 4.17 & 4.20 ).

At each refinement level, there is clear improvement in the spectral resolution coming from every error indicator. Universally, the higher frequency parts of the spectra shift upward toward the DNS line, while the peaks at the lower frequencies come into sharper focus. There is a visibly clear difference between the error indicators in their rate of improvement. Indicator ${}^{\mathbf{smth}}\mathcal{E}$ is the slowest to converge to DNS. Indicators ${}^{\mathbf{T.L.err}}\mathcal{E}$ and ${}^{\mathbf{unStd}}\mathcal{E}$ converge almost equally fast, with

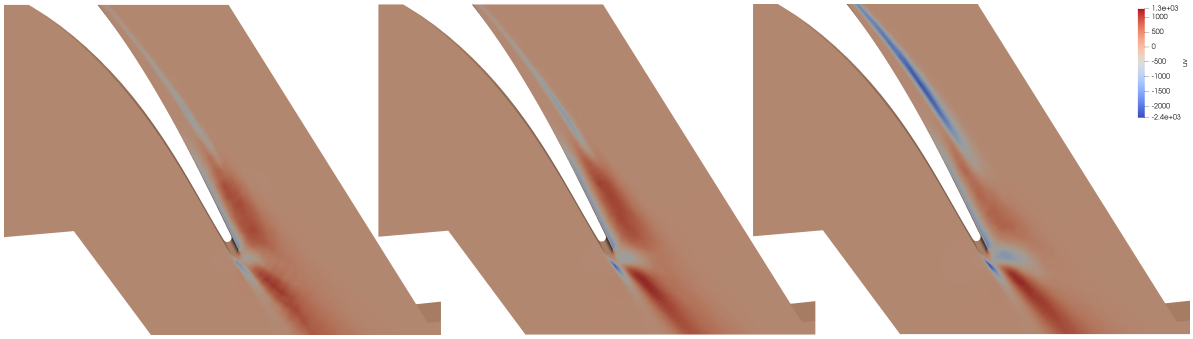
$^{\mathbf{T.L.err}}\mathcal{E}$ slightly outpacing $^{\mathbf{unStd}}\mathcal{E}$.

### 4.1.5 Assessment

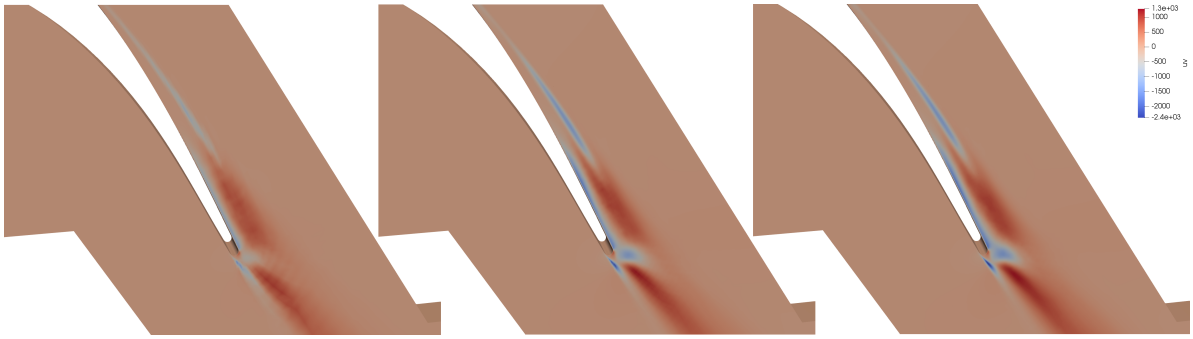
At this point, it can be said that all of the estimators have done a tolerable job. All three identified the important parts of the flow field, boosting resolution where the flow over the blade both detached & reattached. They also boosted resolution in the bubble of turbulence that formed at the tail of the blade and spun up into a vortex train. The estimators exhibited a difference in preference for refining the small cells in the boundary layer versus the large cell along the path of vortex flow. But despite the difference in the targeted refinement locations, all error estimators were able to improve the resolution of the simulation. That improvement was noticeable: 1) in the visible rendering of the vortices in the turbulent wake, 2) through minor improvements to the force coefficients, 3) upon plots of the Reynolds stress, and 4) within the energy spectra of flow. There were several indications that $^{\mathbf{smth}}\mathcal{E}$ was the weakest of the three error estimators.



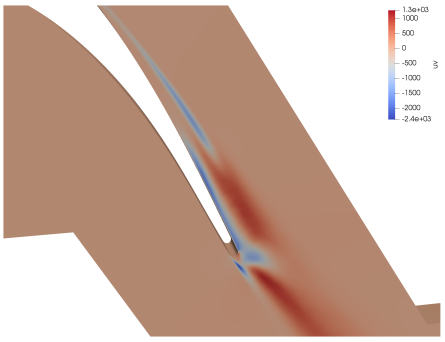

(a) from error indicator $^{\mathbf{unStd}}\mathcal{E}$




(b) from error indicator $^{\mathbf{smth}}\mathcal{E}$

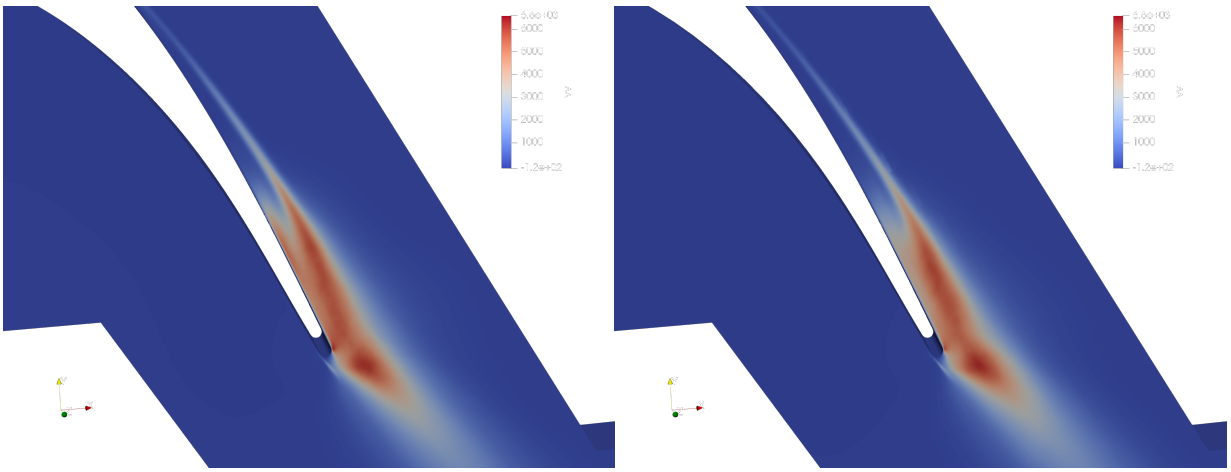



(c) from error indicator $^{\mathbf{T.L.err}}\mathcal{E}$




(d) DNS truth comparison

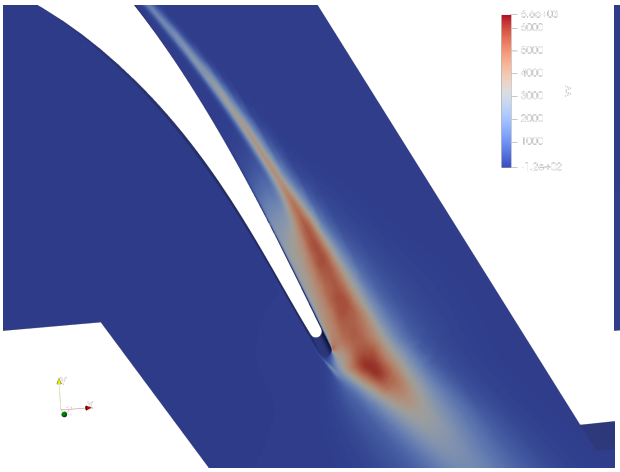
Figure 4.12: Progression of the horizontal cross term of the Reynolds stress tensor, from the coarse mesh on the left to the fine mesh on the right.



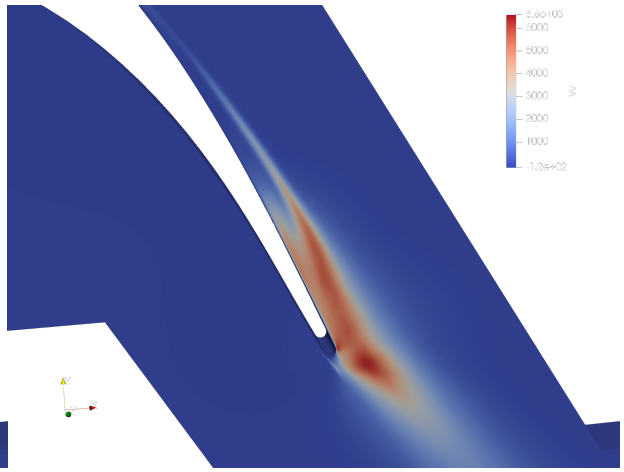
(a) DNS



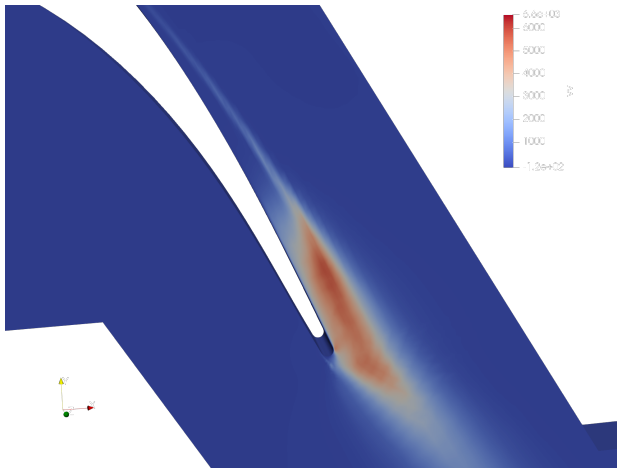
(b) $^{\mathbf{unStd}}\mathcal{E}$



(c) $^{\mathbf{smth}}\mathcal{E}$

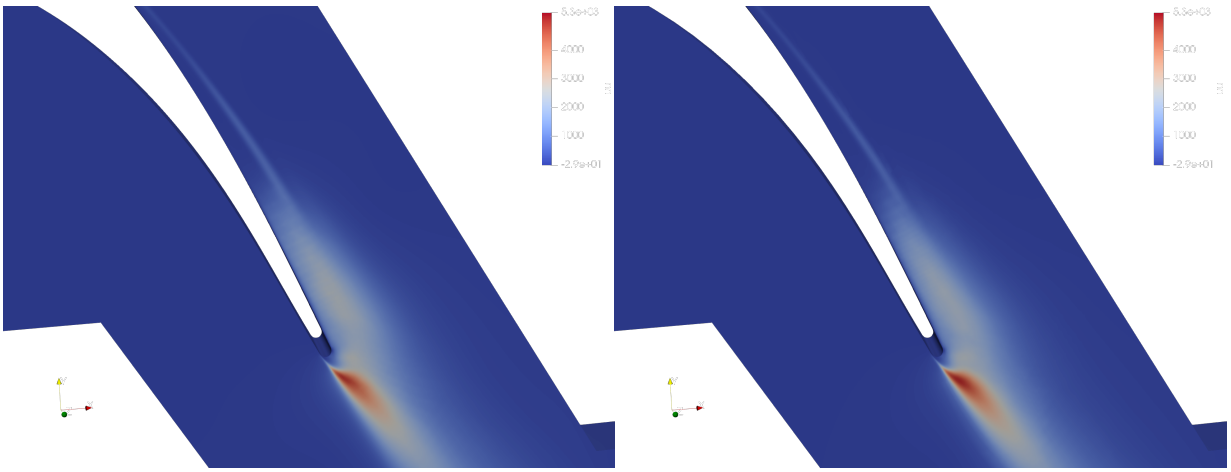


(d) $^{\mathbf{T.L.err}}\mathcal{E}$



(e) base mesh

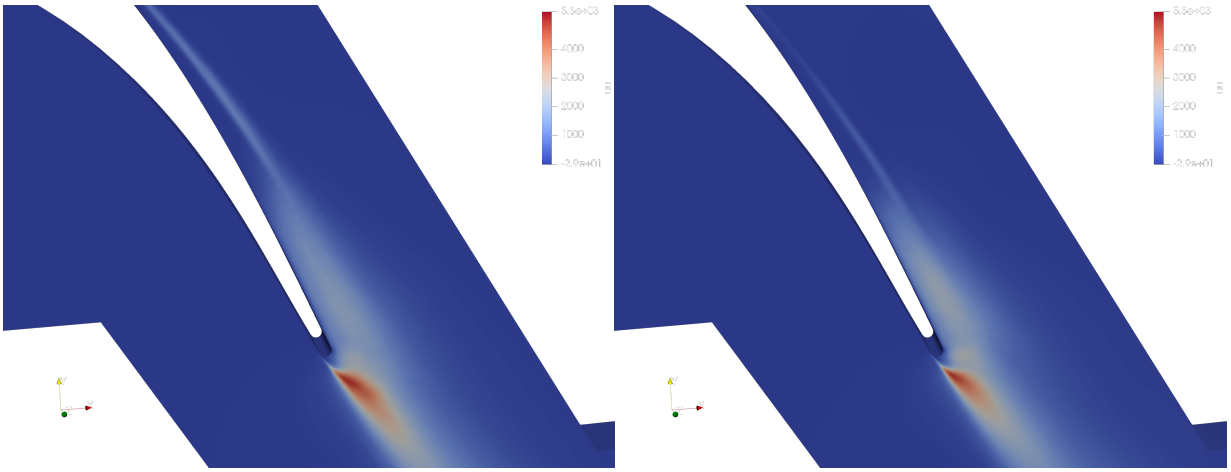
Figure 4.13: Reynolds stress VV component for the refinement #2.



(a) DNS



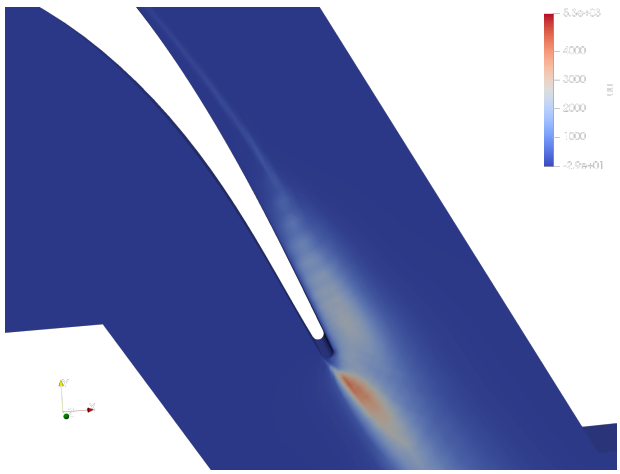
(b) $^{\mathbf{unStd}}\mathcal{E}$



(c) $^{\mathbf{smth}}\mathcal{E}$



(d) $^{\mathbf{T.L.err}}\mathcal{E}$



(e) base mesh

Figure 4.14: Reynolds stress UU component for the refinement #2.

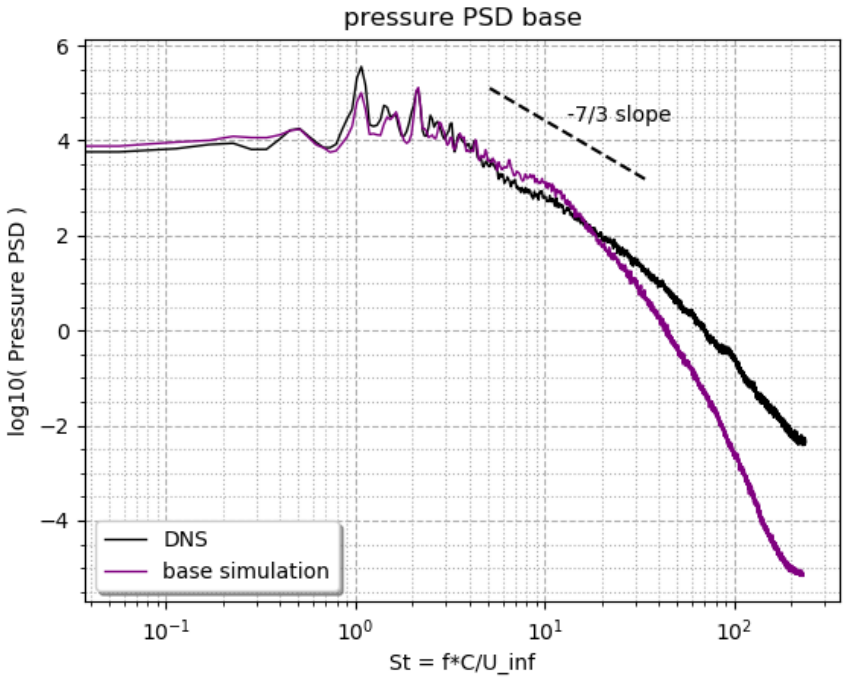



Figure 4.15: Pressure PSD comparison between the DNS truth (black) and the unrefined coarse mesh (purple).

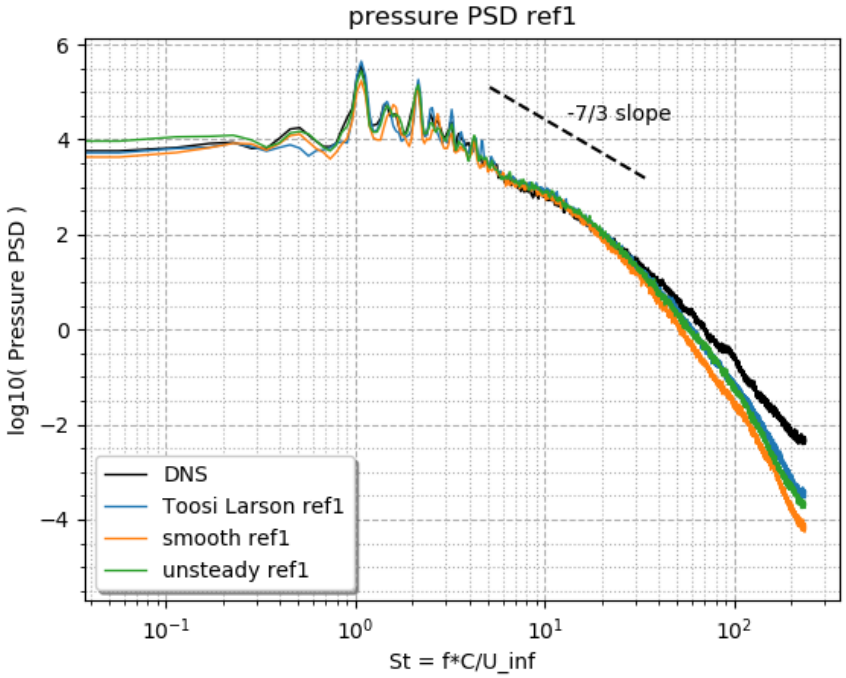



Figure 4.16: Pressure PSD comparison after the 1st refinement. DNS truth is plotted in black. ${}^{\mathbf{unStd}}\mathcal{E}$ is plotted in green. ${}^{\mathbf{smth}}\mathcal{E}$ is plotted in orange. ${}^{\mathbf{T.L.err}}\mathcal{E}$ is plotted in blue.

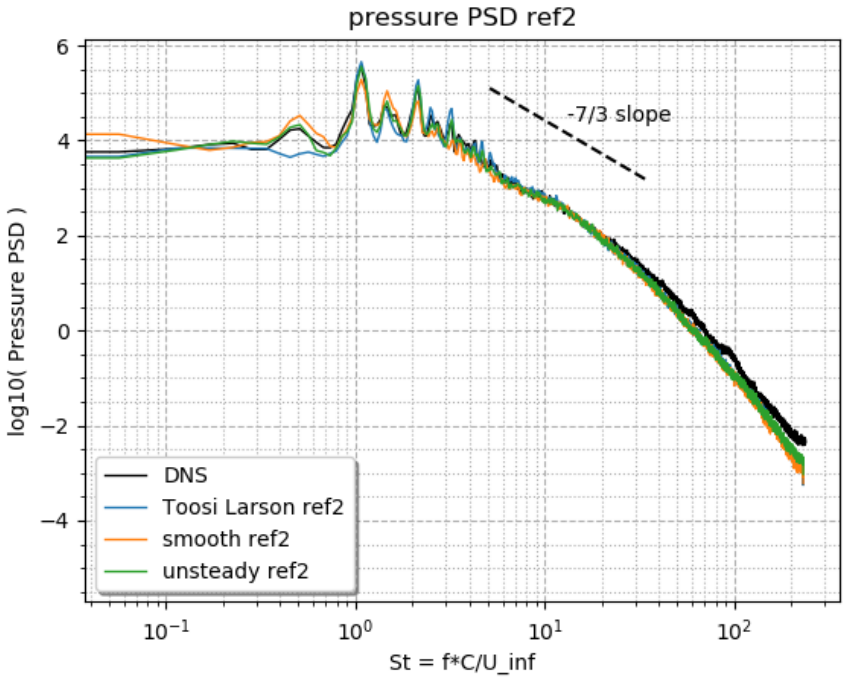



Figure 4.17: Pressure PSD comparison after the 2nd refinement. DNS truth is plotted in black. ${}^{\mathbf{unStd}}\mathcal{E}$ is plotted in green. ${}^{\mathbf{smth}}\mathcal{E}$ is plotted in orange. ${}^{\mathbf{T.L.err}}\mathcal{E}$ is plotted in blue.

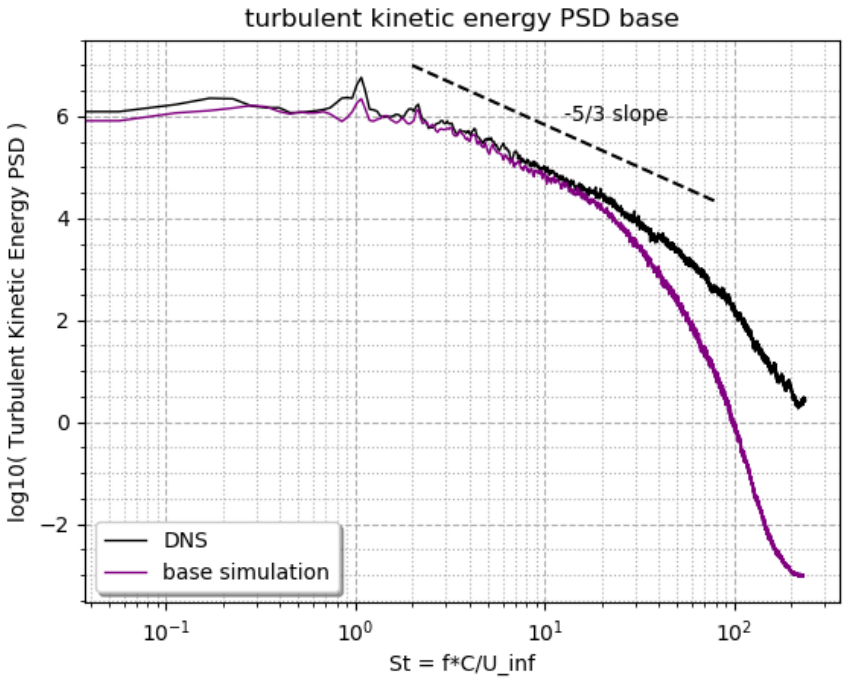



Figure 4.18: Turbulent kinetic energy PSD comparison between the DNS truth (black) and the unrefined coarse mesh (purple).

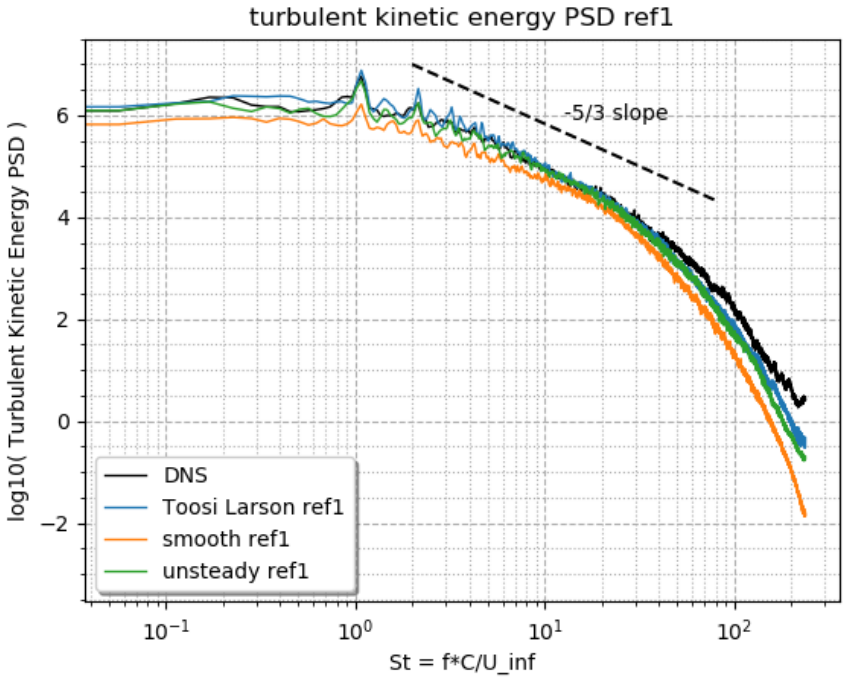



Figure 4.19: Turbulent kinetic energy PSD comparison after the 1st mesh refinement. DNS truth is plotted in black. ${}^{\mathbf{unStd}}\mathcal{E}$ is plotted in green. ${}^{\mathbf{smth}}\mathcal{E}$ is plotted in orange. ${}^{\mathbf{T.L.err}}\mathcal{E}$ is plotted in blue.

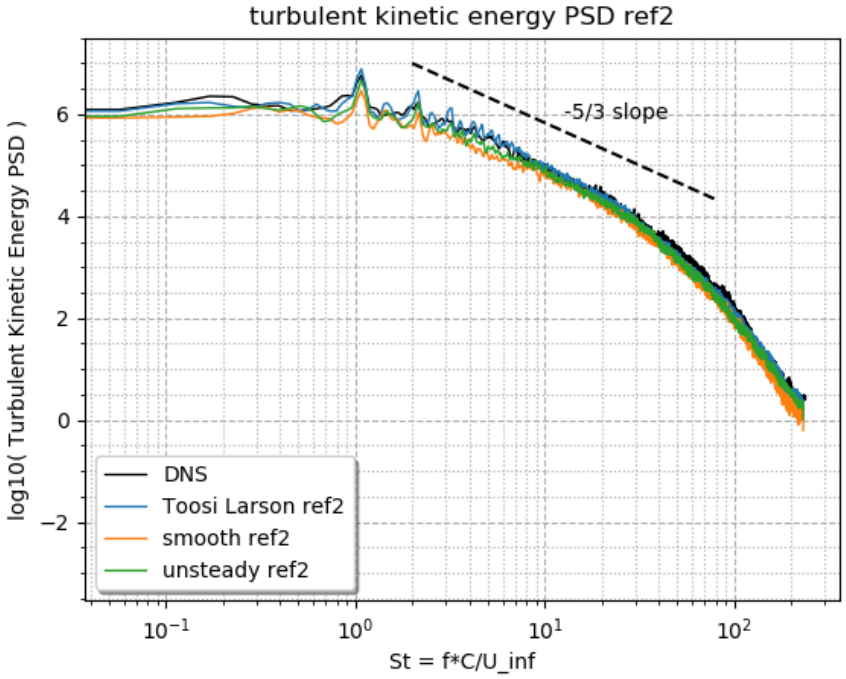



Figure 4.20: Turbulent kinetic energy PSD comparison after the 2nd mesh refinement. DNS truth is plotted in black. ${}^{\mathbf{unStd}}\mathcal{E}$ is plotted in green. ${}^{\mathbf{smth}}\mathcal{E}$ is plotted in orange. ${}^{\mathbf{T.L.err}}\mathcal{E}$ is plotted in blue.

## 4.2 Modified T106a

The second test case is derived from the T106a simulation, another well known LES benchmark[2], similar in design to the previous one. Figure 4.21 illustrates the geometry, drawing the mesh in red. The domain is again an infinite stack of T106 turbine blades, duplicated above and below the primary blade by way of cyclic boundary conditions. The vertical spacing between the blades (the pitch-to-chord ratio) is $0.798\,C$, with $C$ being the chord length of the primary blade. The span-to-chord ratio (depth into the page) is $0.1C$. The flow state is defined by the ambient conditions upon the back exit. Outgoing air leaves with isentropic Mach number $M_{is} = 0.4$ at isentropic Reynolds number of $Re_{is} = 60,000$ with scale $C = 1$ m. Incoming flow is angled upward by $46.1°$.

A few differences exist between this T106a simulation and the version described by Hillewaert & JS. (2016, 2018). First, the incoming angle differs slightly. Second, *symmetric* boundary conditions are applied above and below in the z-direction (in-to and out-of the page). The classic T106a applies *cyclic* boundaries in the z-direction, making the blade infinitely wide. In this dissertation, the blade is of finite width, sandwiched between reflective walls. [3]

A final point of difference must also be mentioned: There is a geometry flaw upon the blade's tip and its tail. The coarse mesh has a sharp angle at the juncture between splines, and the highly curved parts of the blade surface have a seashell-like bobble. It was not the intent to have a sharp corner or a seashell bobble. The reason for highlighting them is to point out a particular dynamic of the flow field that might be the consequence of these features: It is not until the resolution of the simulation is heightened that the primary vortex shedding frequency becomes visible. The suspected but unconfirmed cause is that the sharp corner and seashell pattern inject miniature eddies into the turbulence, changing its character. The T106a test case was designed to challenge the error estimators at resolving a flow field from a very coarse and poorly designed mesh. The unintended geometry quirk is in keeping with that challenge, increasing the difficulty level.

[2]For more on the classic T106a simulation, consult the websites for the 4th and the 5th International Workshops on High-Order CFD Methods: (Hillewaert & JS., 2016, 2018).

[3]Why the change to *symmetric* boundaries? Answer: Pragmatism. Originally, hpMusic, the lab's FR/CPR solver, could not tolerate a hanging-node mismatch between cyclic boundaries. The missing feature has now been implemented, but the T106a setup file is preserved.

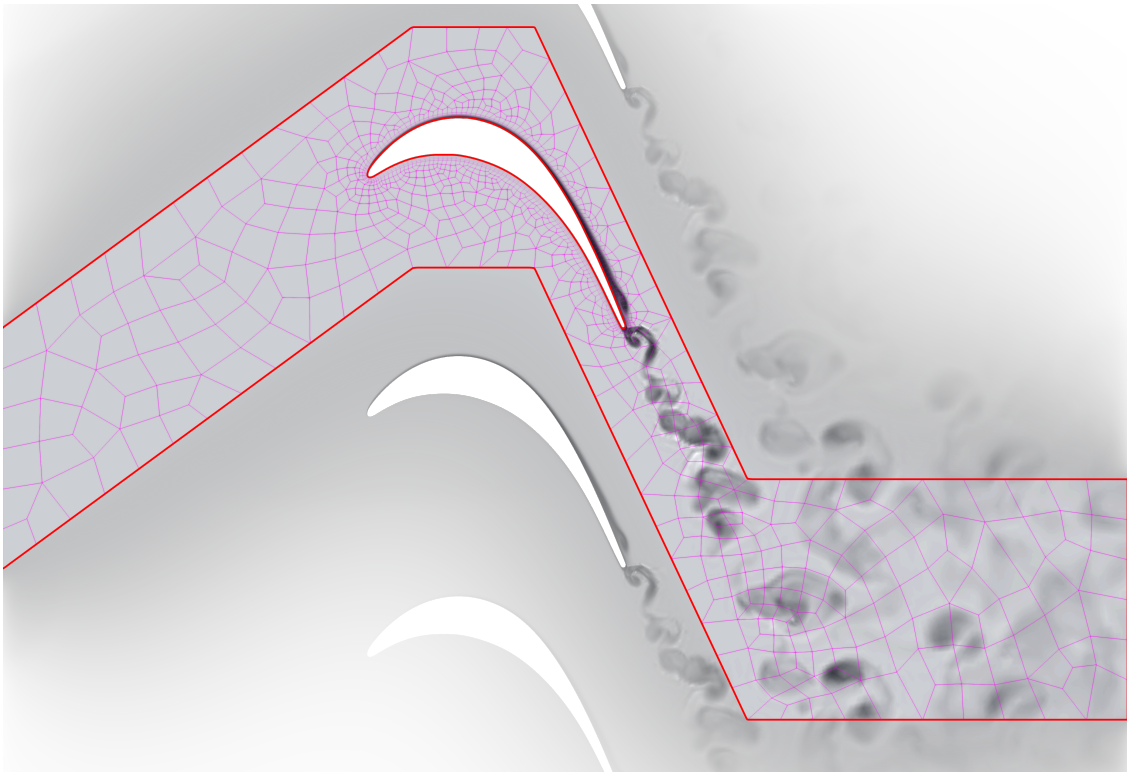


Figure 4.21: The simulation domain for the T106a turbine blade.

The test procedure is similar to that for T106c. Two rounds of H-adaptation are applied, starting from a coarse initial mesh, incapable of supporting the final flow field dynamics. The simulation's polynomial order is again $\mathcal{P}_2$. Unlike the previous test case, there is no readily available DNS simulation to act as a definitive truth source. Comparison will be made against a higher resolution $\mathcal{P}_4$ simulation with 1-level of uniform H-refinement to the mesh. Also featured in the comparison will be a sequence of increasingly fine uniformly HP-adapted simulations leading to the $\mathcal{P}_4$ simulation. Those additional simulations will be used to place the actions of the error-estimators in a broader context to illustrate exactly how they are responding to the flow field. Unlike the previous test case, there will not be a definitive winner from this test because none of the error estimators will have managed to completely reproduce all aspects of the flow field by the third round of refinement, with the 20% cell splitting budget. However, it will be shown that both ${}^{\mathbf{T.L.err}}\mathcal{E}$ and ${}^{\mathbf{unStd}}\mathcal{E}$ land closer to the $\mathcal{P}_4$ "truth" source than ${}^{\mathbf{smth}}\mathcal{E}$.

### 4.2.1 The $\mathcal{P}_4$ simulation

Before discussing the error estimators, it is necessary to open with a few words about the $\mathcal{P}_4$ simulation, justifying its claim to be a sufficient truth source for the analysis. Figure 4.22 begins the justification by plotting the distribution of the pressure coefficient ($C_p$) across the T106a blade as a function of x-axis distance. Five simulations are superimposed in the graph, drawn with increasing levels of blackness to signify their increasing resolution. Figure 4.23 is a zoomed view of the rightward side, showing the line clustering. The lightest curve, "P1_uref0," is from a $\mathcal{P}_1$ simulation upon the base mesh (uniform refinement level "0"). "P2_uref0" is a $\mathcal{P}_2$ simulation upon the same mesh. "P3_uref1" is a $\mathcal{P}_3$ simulation upon a once uniformly refined mesh. Finally, "P3_uref2," visibly coincident with "P4_uref1," is from a $\mathcal{P}_3$ simulation upon a twice uniformly refined mesh, while "P4_uref1" is the $\mathcal{P}_4$ simulation upon the once uniformly refined mesh.

Figure 4.24 plots the error in the lift and drag coefficients ($C_L$ and $C_D$) from the sequence, illustrating that error is shrinking exponentially and has approached floating point roundoff by the $\mathcal{P}_4$ simulation. For these plots, grid size is proxied by $(\sqrt[3]{n_{\text{DOF}}})^{-1}$, while error is proxied via the subtraction of the force coefficients from "P3_uref2."

If it were necessary to establish DNS accuracy, justification would be required that a further increase to resolution would not alter the flow field. However, DNS accuracy will not be needed for the forthcoming analysis. DNS resolution will never be reached by any of the error estimator simulations. At best, the error estimators will be able to duplicate small pockets of resolution from "P4_uref1" and "P3_uref2." The following analysis will make use of the $\mathcal{P}_4$ simulation as a comparative aid to illustrate the progress of the error estimators.

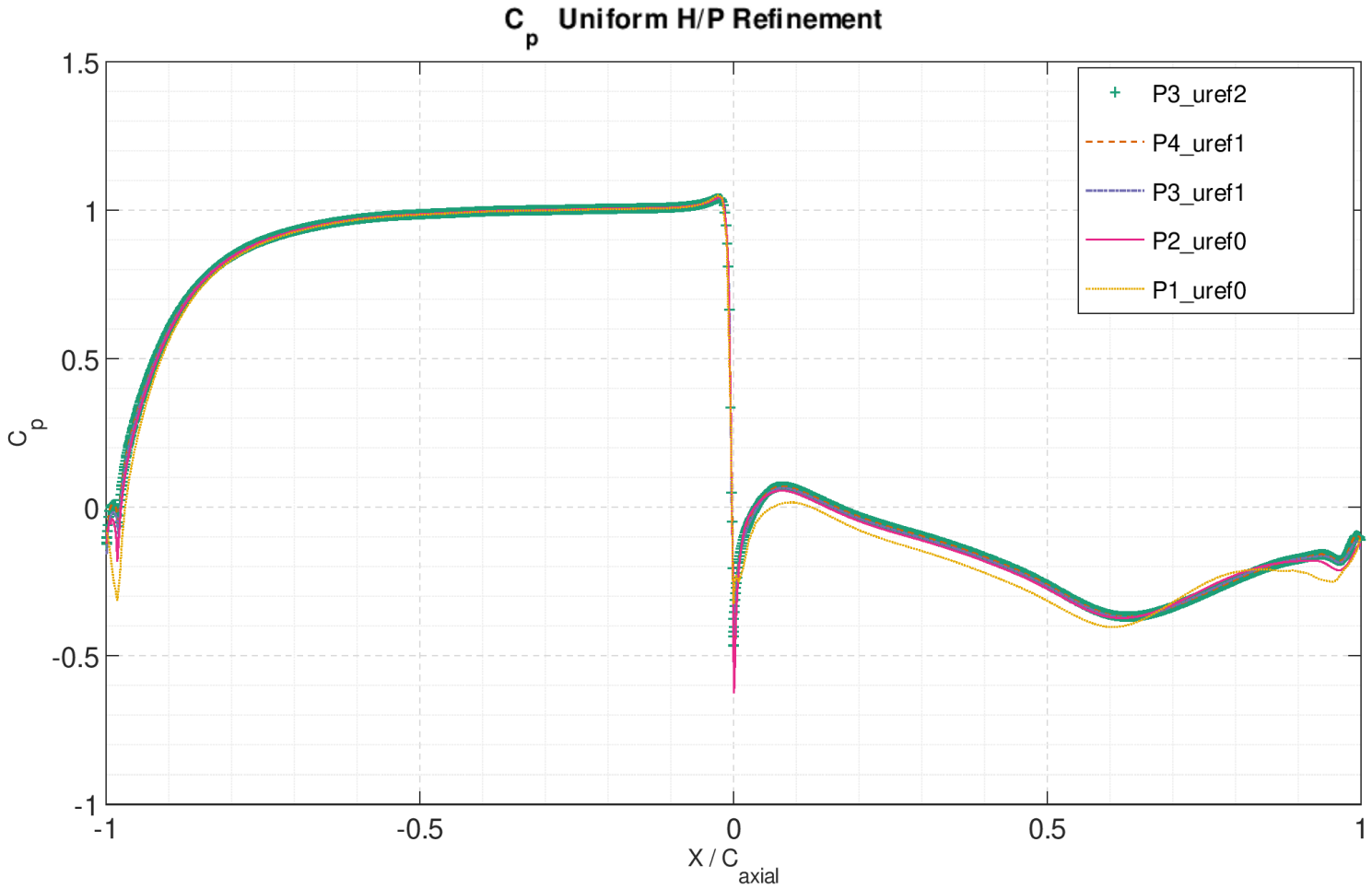



Figure 4.22: Pressure coefficient distribution across the surface of the T106a turbine blade, simulated at increasing resolution via uniform HP-refinement. As resolution increases, the simulations become increasingly close, until the final two simulations land on top of each other: simulation $\mathcal{P}_4$ upon a once uniformly refined mesh and simulation $\mathcal{P}_3$ upon a twice uniformly refined mesh.

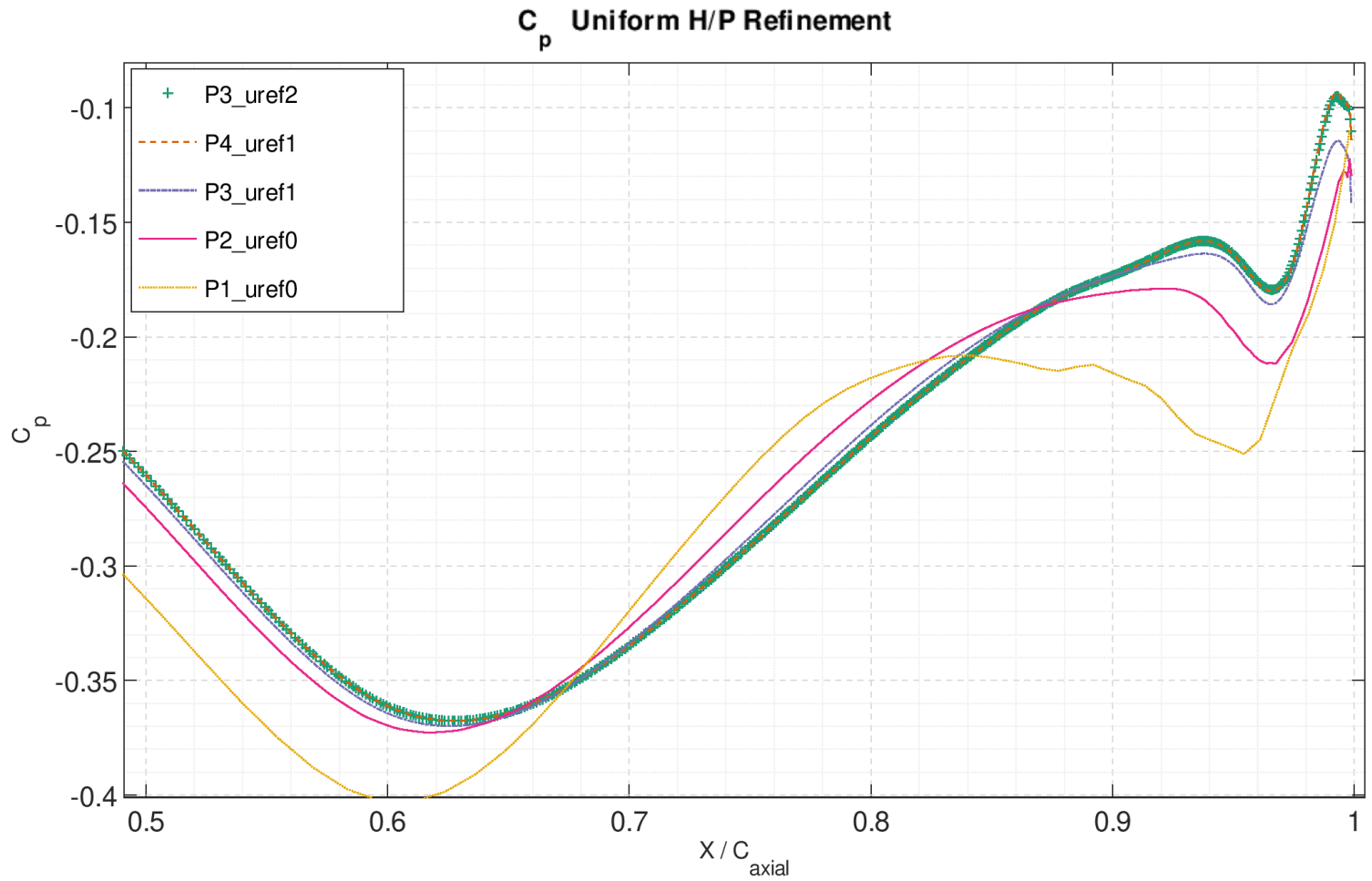



Figure 4.23: Zoomed view of the pressure coefficient upon the upper surface of the turbine blade, illustrating the match between "P3_uref2" "P4_uref1."

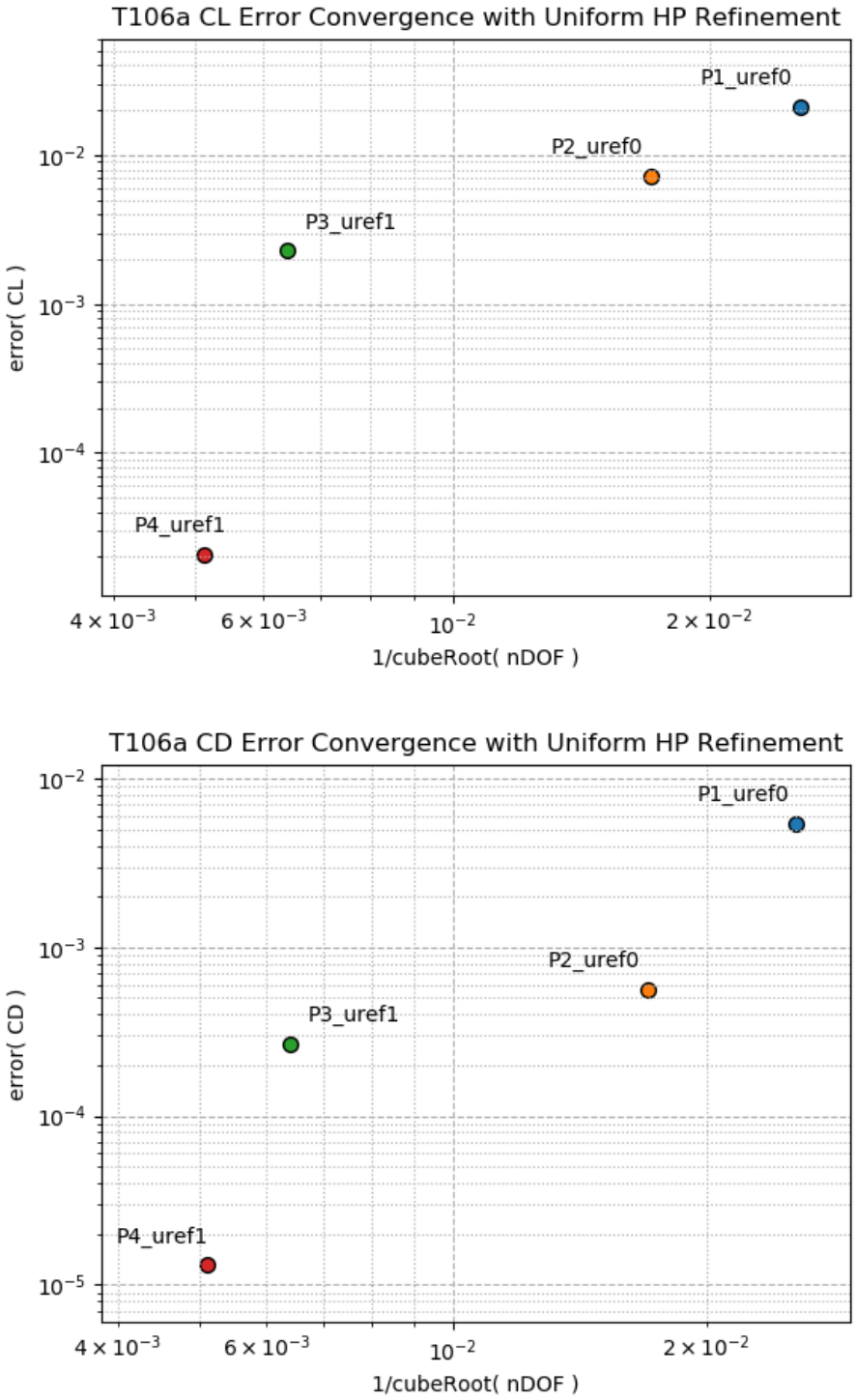



Figure 4.24: Log-Log plots of the error in the lift and drag coefficients from the high-order simulations used as a "truth."

### 4.2.2 The Refinement Zones

Following the same topic ordering as Section 4.1, Figure 4.25 begins the analysis with a visual comparison of the turbulent wake after H-adaptation #2. Views of the mesh have been superimposed in the flow field, with hanging node boundaries outlined in green. It is clear that all of the error estimators have dramatically improved the level of detail in the the flow field, encircling the turbulent eddies refinement zones. Even with this first qualitative comparison of the error estimators, it is clear that the refinement pattern witnessed in Section 4.1 is starting to repeat here. That is, ${}^{\mathbf{smth}}\mathcal{E}$ is sticking close to the surface, venturing only a little into the turbulent wake, while both ${}^{\mathbf{unStd}}\mathcal{E}$ and ${}^{\mathbf{T.L.err}}\mathcal{E}$ are working within the wake.

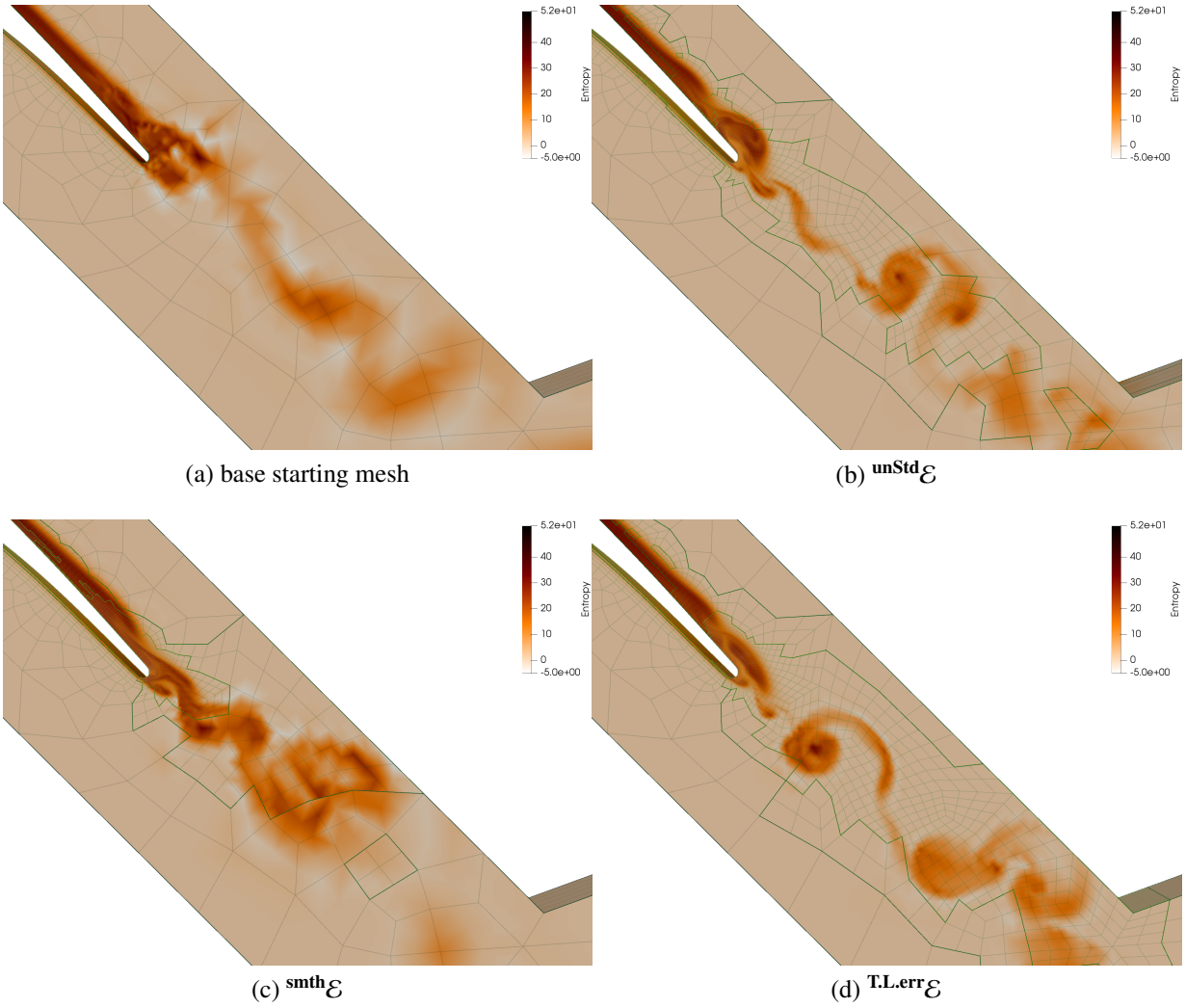
Figure 4.26 directly contrasts the applied H-adaptation, color coding the flagged cells. Zoomed views are provided by Figures 4.26, 4.27, 4.28. It can be seen that ${}^{\mathbf{T.L.err}}\mathcal{E}$ and ${}^{\mathbf{unStd}}\mathcal{E}$ are refining in remarkably similar places, but ${}^{\mathbf{T.L.err}}\mathcal{E}$ is covering a wider area. As will be shown by the next sequence of plots, the wider coverage of ${}^{\mathbf{T.L.err}}\mathcal{E}$ is because ${}^{\mathbf{unStd}}\mathcal{E}$ chose to spend more of its refinement budget on the cells in the boundary layer.

Figures 4.29, 4.30, and 4.31 plot the $y^+$ resolution. These charts show that ${}^{\mathbf{smth}}\mathcal{E}$ has refined along both the pressure and suctions sides of blade, while ${}^{\mathbf{unStd}}\mathcal{E}$ has refined far more upon the suction side than the pressure side. ${}^{\mathbf{T.L.err}}\mathcal{E}$ has only refined the blade surface in targeted areas: the tail and certain locations upon the suction side.

### 4.2.3 Force Analysis: lift, drag, and pressure

The time-averaged lift and drag coefficients are reported by Tables 4.2a and 4.2b, while Figures 4.32a and 4.32b plot the numbers. Note that both tables and plots contain an extra round of refinement to help illustrate the trend lines. While the lift coefficient appears to be converging toward the "P4_uref1" truth reference, the drag coefficient appears to diverging. In a reversal of the usual trend, it is ${}^{\mathbf{T.L.err}}\mathcal{E}$ that is most in disagreement with the $C_{\mathrm{D}}$ truth reference, and it is ${}^{\mathbf{smth}}\mathcal{E}$ that is the least in disagreement. ${}^{\mathbf{unStd}}\mathcal{E}$, as always, is in the middle.

Figure 4.33 plots the pressure coefficient, one error indicator per row. The center plots cover




(a) base starting mesh

(b) $^{\mathbf{unStd}}\mathcal{E}$

(c) $^{\mathbf{smth}}\mathcal{E}$

(d) $^{\mathbf{T.L.err}}\mathcal{E}$

Figure 4.25: Visual quality comparison of the T106a turbulent wake after refinement #2. The vortices have been made visible by rendering the instantaneous entropy increase relative to the far field.

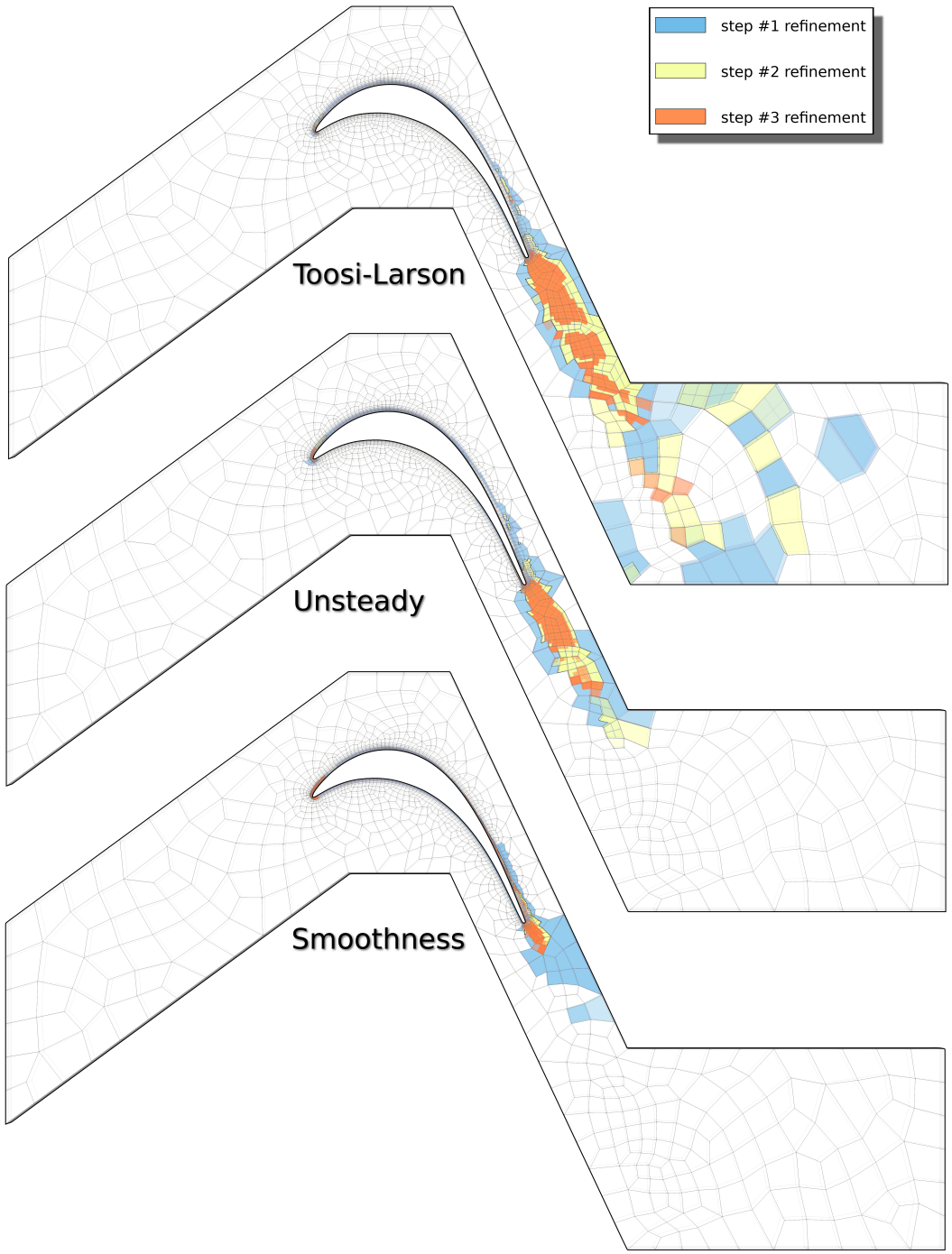



Figure 4.26: Comparison of the mesh refinement requested for simulation T106a. With each error indicator, the simulation was executed three times in sequence, starting with the same base mesh. Blue marks cells that were flagged for refinement at round #1. Yellow marks cells that were flagged for refinement at round #2. Orange marks cells that were flagged at round #3. All error indicators were granted the same refinement budget: Split the top 20% most erroneous cells, plus as many additional cells as it takes to meet the two smoothness requirements.

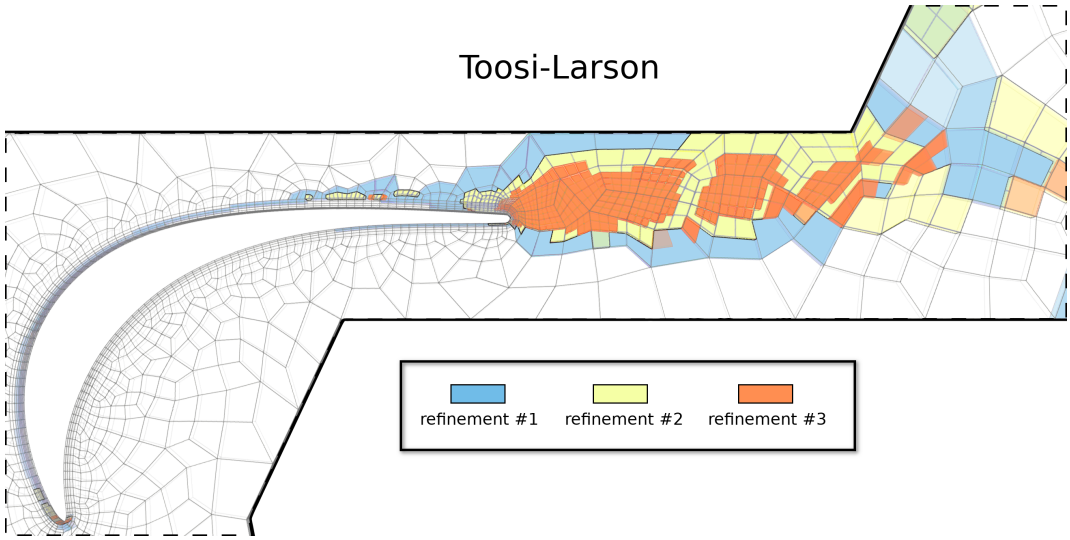



Figure 4.26: Zoomed view of the cells flagged from simulation T106a by $^{\mathbf{T.L.err}}\mathcal{E}$.

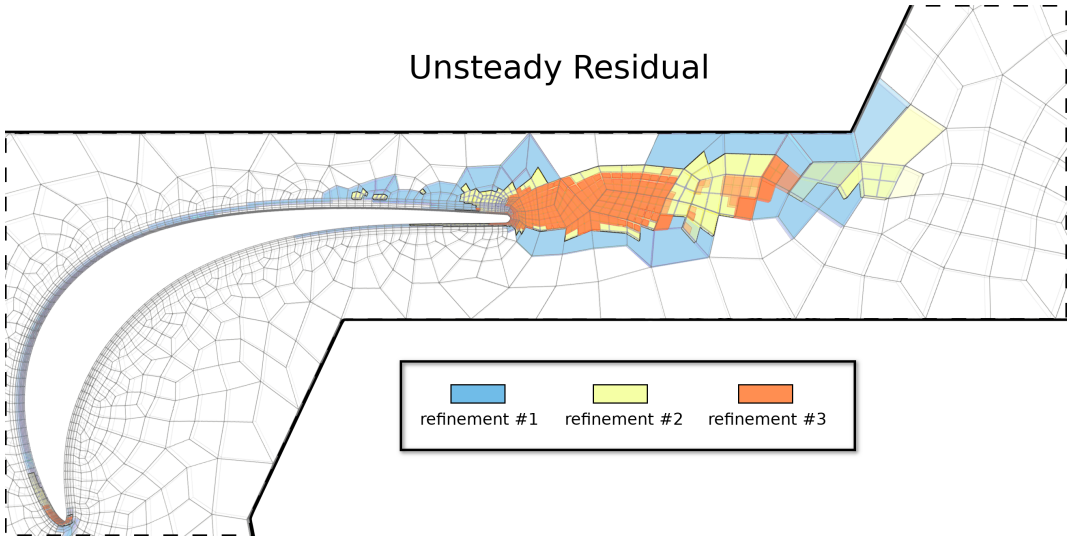



Figure 4.27: Zoomed view of the cells flagged from simulation T106a by $^{\mathbf{unStd}}\mathcal{E}$.

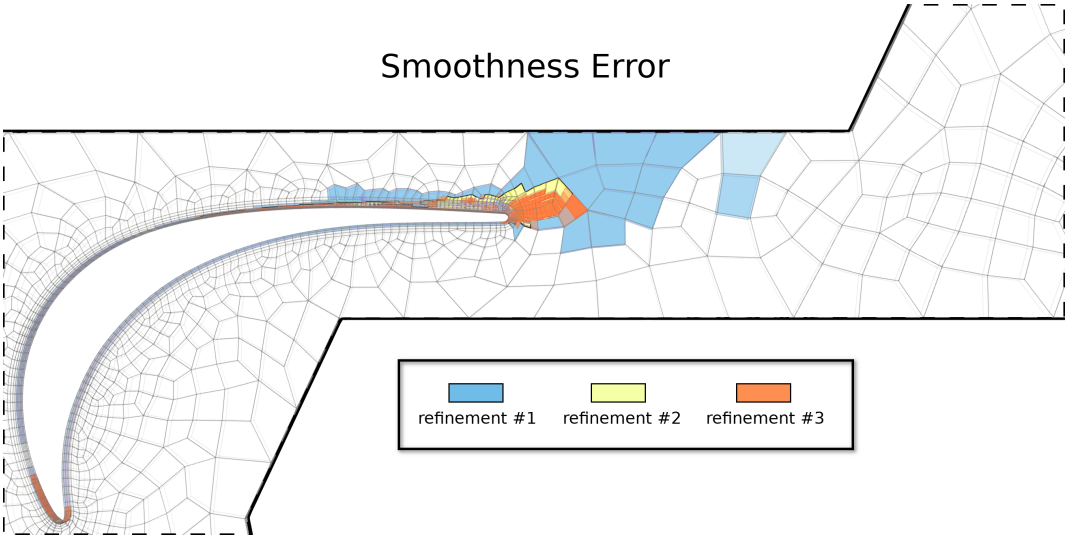



Figure 4.28: Zoomed view of the cells flagged from simulation T106a by ${}^{\mathbf{smth}}\mathcal{E}$.

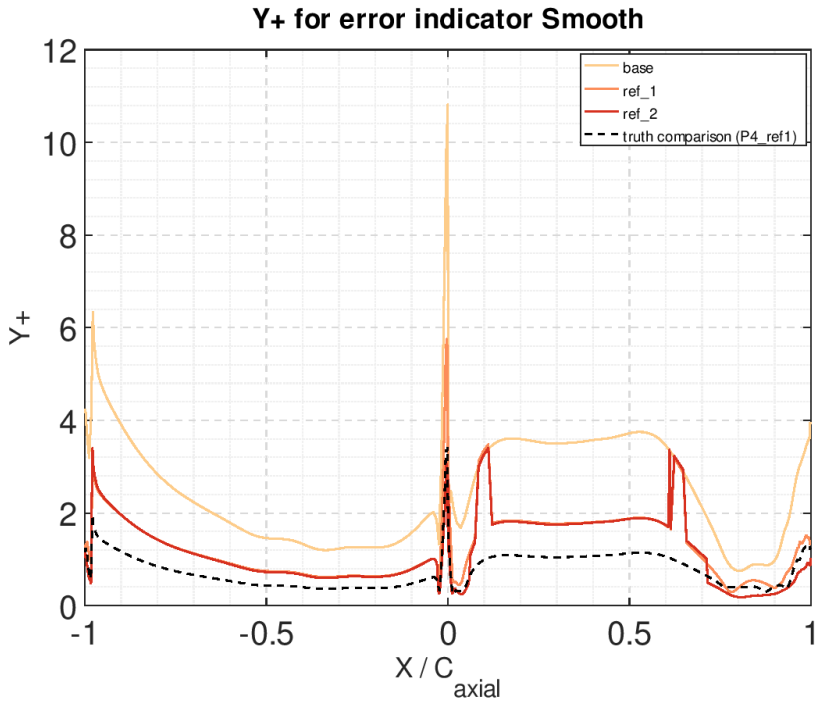



Figure 4.29: Progression of $y^+$ for ${}^{\mathbf{smth}}\mathcal{E}$

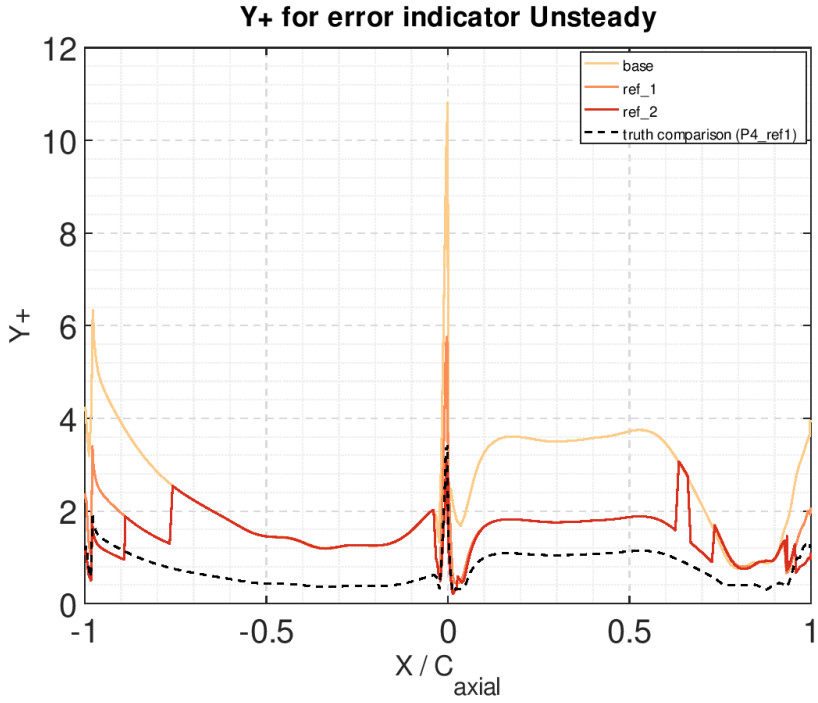



Figure 4.30: Progression of $y^+$ for ${}^{\mathbf{unStd}}\mathcal{E}$

| | $C_L$ common | $C_L$ for ${}^{\mathbf{smth}}\mathcal{E}$ | $C_L$ for ${}^{\mathbf{unStd}}\mathcal{E}$ | $C_L$ for ${}^{\mathbf{T.L.err}}\mathcal{E}$ |
|---|---|---|---|---|
| base mesh | 0.2675 ± 0.005 | | | |
| refinement #1 | | 0.2656 ± 0.004 | 0.2652 ± 0.003 | 0.2660 ± 0.003 |
| refinement #2 | | 0.2652 ± 0.003 | 0.2645 ± 0.002 | 0.2656 ± 0.003 |
| (extra) ref. #3 | | 0.2645 ± 0.002 | 0.2648 ± 0.003 | 0.2632 ± 0.003 |
| truth ref. | 0.2603 ± 0.006 | | | |

(a) T106a lift coefficient

| | $C_D$ common | $C_D$ for ${}^{\mathbf{smth}}\mathcal{E}$ | $C_D$ for ${}^{\mathbf{unStd}}\mathcal{E}$ | $C_D$ for ${}^{\mathbf{T.L.err}}\mathcal{E}$ |
|---|---|---|---|---|
| base mesh | 1.0031 ± 0.011 | | | |
| refinement #1 | | 1.0038 ± 0.011 | 1.0044 ± 0.007 | 1.0043 ± 0.008 |
| refinement #2 | | 1.0046 ± 0.010 | 1.0058 ± 0.007 | 1.0064 ± 0.005 |
| (extra) ref. #3 | | 1.0046 ± 0.012 | 1.0058 ± 0.007 | 1.0066 ± 0.007 |
| truth ref. | 1.0036 ± 0.002 | | | |

(b) T106a drag coefficient

Table 4.2: progression of the lift coefficient (table 4.2a) and the drag coefficient (table 4.2b) for simulation T106a

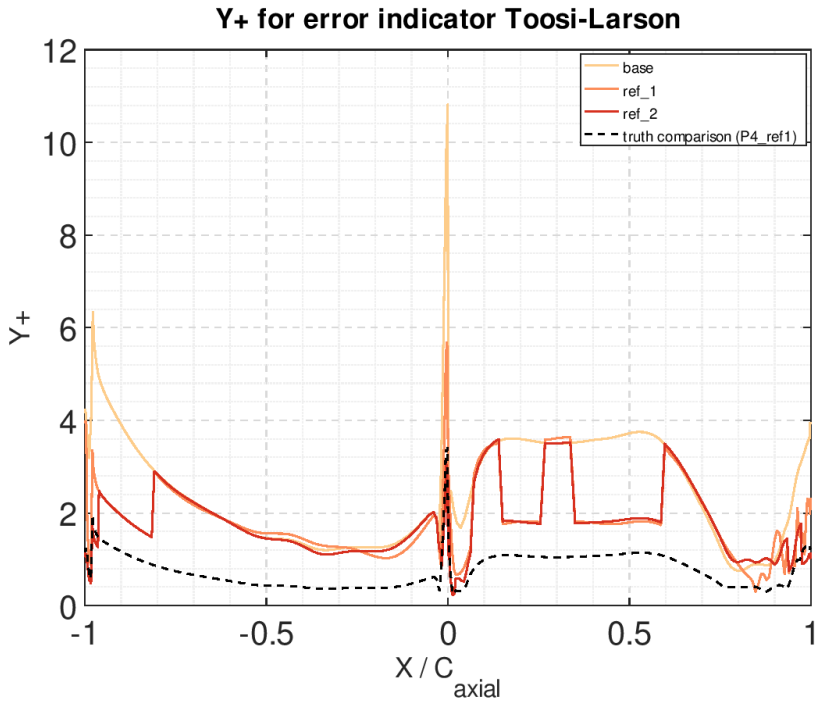
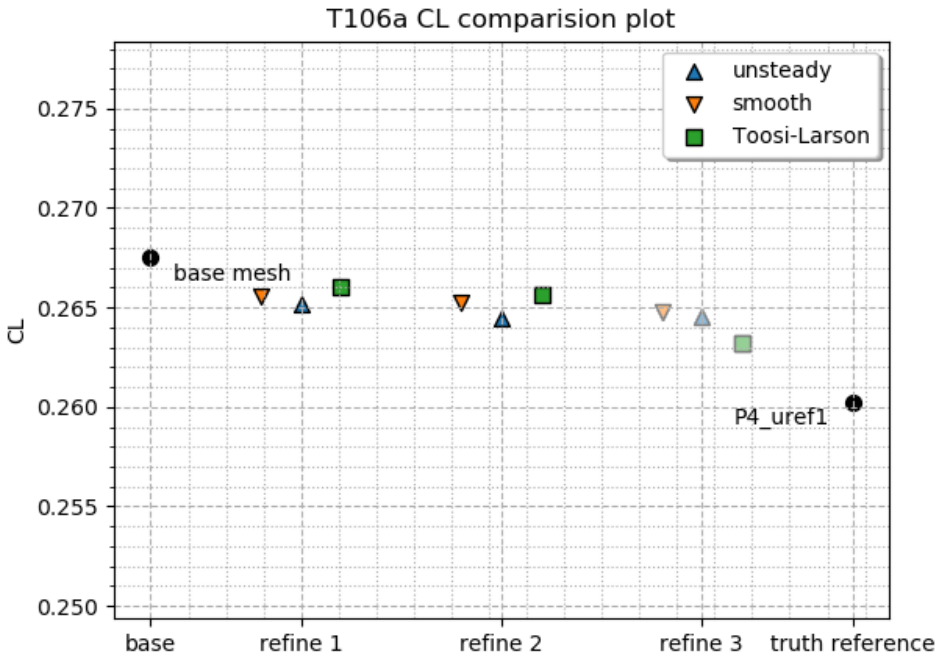



Figure 4.31: Progression of $y^+$ for $^{\mathbf{T.L.err}}\mathcal{E}$

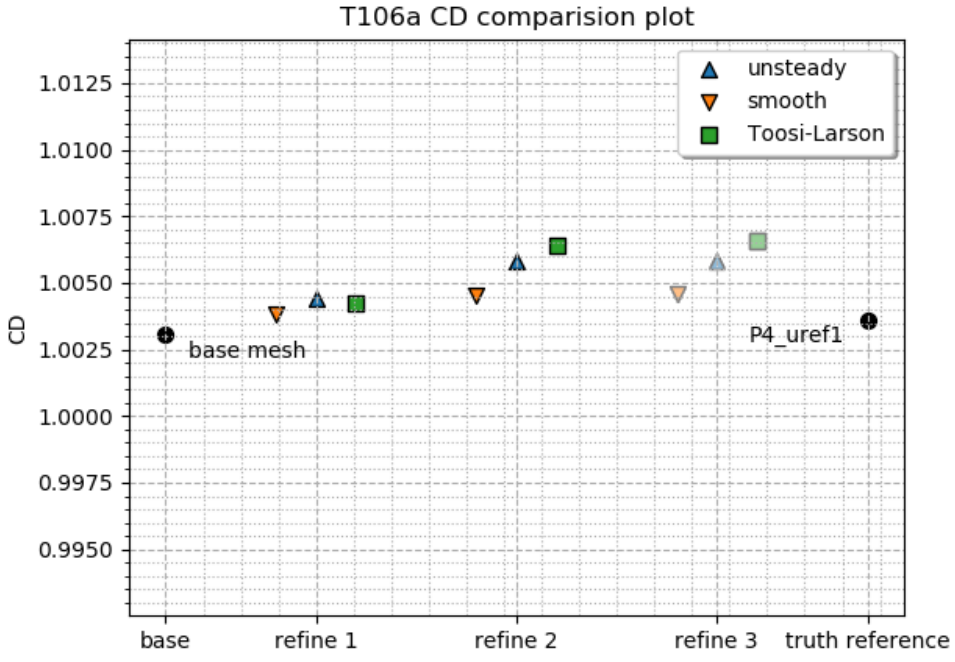
the whole blade, while the the left and right columns are zoomed views. The yellow curve is from the base mesh; the orange curve is from the once H-adapted mesh, and the red curve is from the twice H-adapted mesh. The dashed line is the "P4_uref1" simulation.

Over most of the blade's surface, the red and orange lines align with the dashed lines, indicating good agreement between the truth simulation and the error-adapted ones. However, at the tail of the blade, it is clear the none of the adapted mesh has reproduced the data from "P4_uref1." In fact, for the tail portion of $C_p$, it looks as if $^{\mathbf{unStd}}\mathcal{E}$ and $^{\mathbf{smth}}\mathcal{E}$ have actually become stuck at an incorrect pressure distribution. $^{\mathbf{T.L.err}}\mathcal{E}$ is not stuck as evidenced by the shift in its "ref_2" curve upon the tail.

On their own, these plots are mysterious, but when they are put in context of the uniformly HP-adapted simulations from 4.23, they are easier to interpret. Figures 4.34 through 4.36 overlay the HP-adaptation sequence with error-guided H-adaptive sequence, revealing that the error estimators have "converged" to a point midway between "P2_uref0," the base simulation, and "P3_uref1." Close examination of Figure 4.34 shows that upstream of the tail, $^{\mathbf{T.L.err}}\mathcal{E}$ was strategically shifting




(a) Lift coefficients for T106a




(b) Drag coefficients for T106a

Figure 4.32: Progression of the lift and drag coefficients for simulation T106a

between the accuracy of "P2_uref0" and "P3_uref1." $^{\mathbf{unStd}}\mathcal{E}$ was doing similarly but with less stark transitions. $^{\mathbf{smth}}\mathcal{E}$ didn't progress much beyond "P2_uref0," compared to the other error estimators. $^{\mathbf{smth}}\mathcal{E}$ tended to do better at locations of high flow compression, such as the stagnation point (not shown).

### 4.2.4 Turbulence Analysis: Reynolds stress and energy spectra

Figure 4.37 shows the progression in the UV component of the Reynolds stress tensor for refinements #0 (i.e. the base mesh) through #2. Figure 4.38 shows a zoomed view of the final refinement. Meanwhile Figures 4.39 and 4.40 complement the picture by adding views of the VV and UU stress terms, also from the final round of refinement. There is something of interest to note. Although all of the error-adapted meshes have greatly improved upon the quality of the starting simulation by bringing the Reynolds stress into focus, they have not been altogether successful at producing the correct shapes. For example, the UU hotspot from "P4_uref1" is rounder than the corresponding hotspots from the error-adapted meshes.

The final set of plots to examine are the power spectra. The monitor point position is unchanged from the previous test case, illustrated by Figure 4.11. The pressure and turbulent kinetic energy spectra are presented by Figures 4.41 and 4.42, respectively. Two items are worth comment. First, all of the error estimators are clearly improving the resolution of the turbulent energy cascade. That improvement is evidenced by the high-frequency part of the spectrum, where the curves exhibit increasingly good alignment with the truth reference. Second, the error estimators are *not* clearly improving the position of the energy spikes in the low frequency part of the spectrum. The failure to resolve the spikes is highlighted by Figure 4.43, which provides a zoomed view from refinement #2. For contrast, Figure 4.44 provides the same view from the base simulation. From these two plots, it looks like $^{\mathbf{unStd}}\mathcal{E}$ alone has made progress toward mimicking the first two spikes. The implication is that the error estimators have not fully captured the flow field dynamics, which means that the resolution is lacking somewhere in the domain.

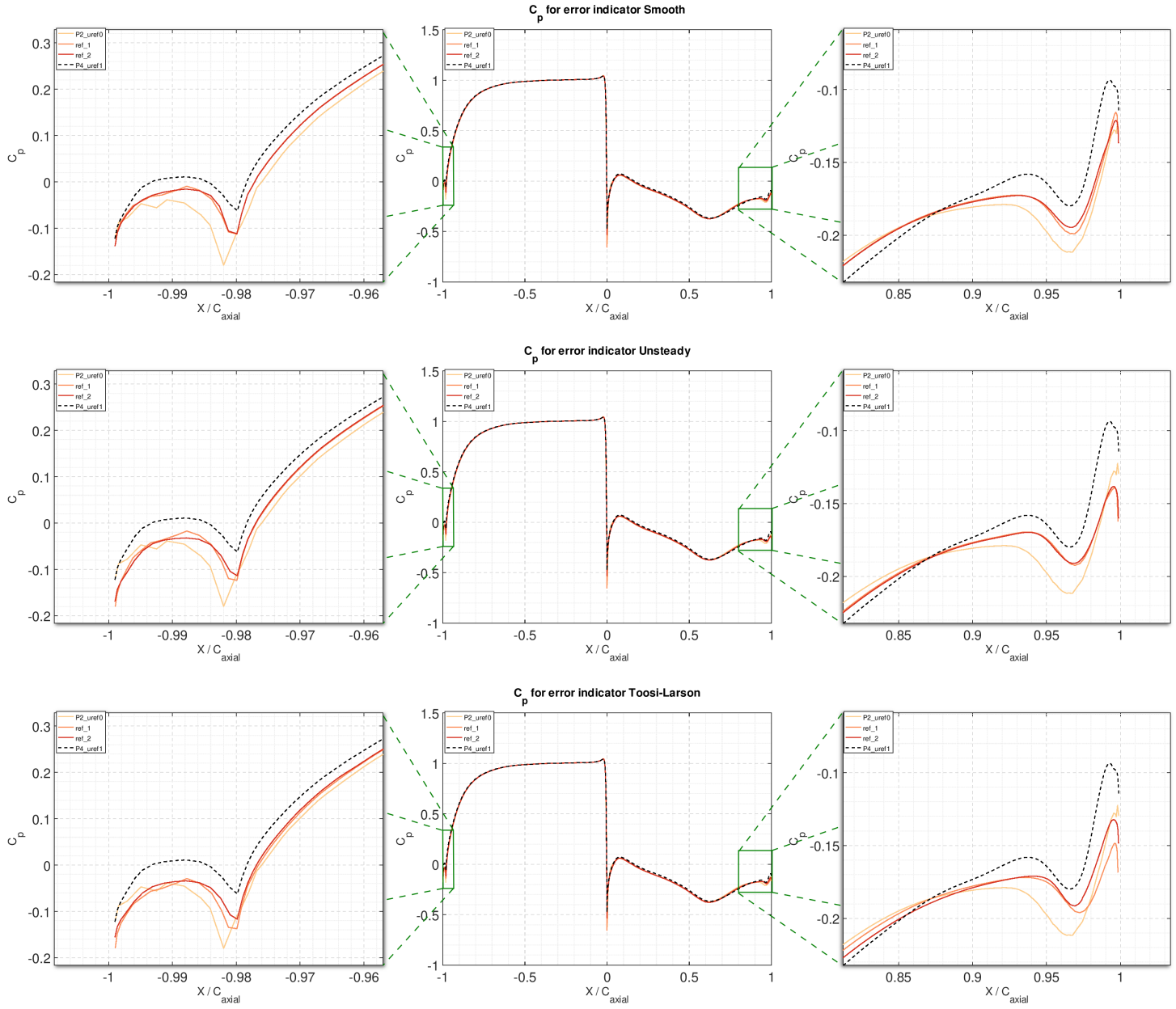



Figure 4.33: Progression of the pressure coefficients. $^{\mathbf{smth}}\mathcal{E}$ is on top. $^{\mathbf{unStd}}\mathcal{E}$ is in the middle. $^{\mathbf{T.L.err}}\mathcal{E}$ is at bottom. The refinement sequence #0, #1, #2 is colored yellow, orange, & red, respectively.

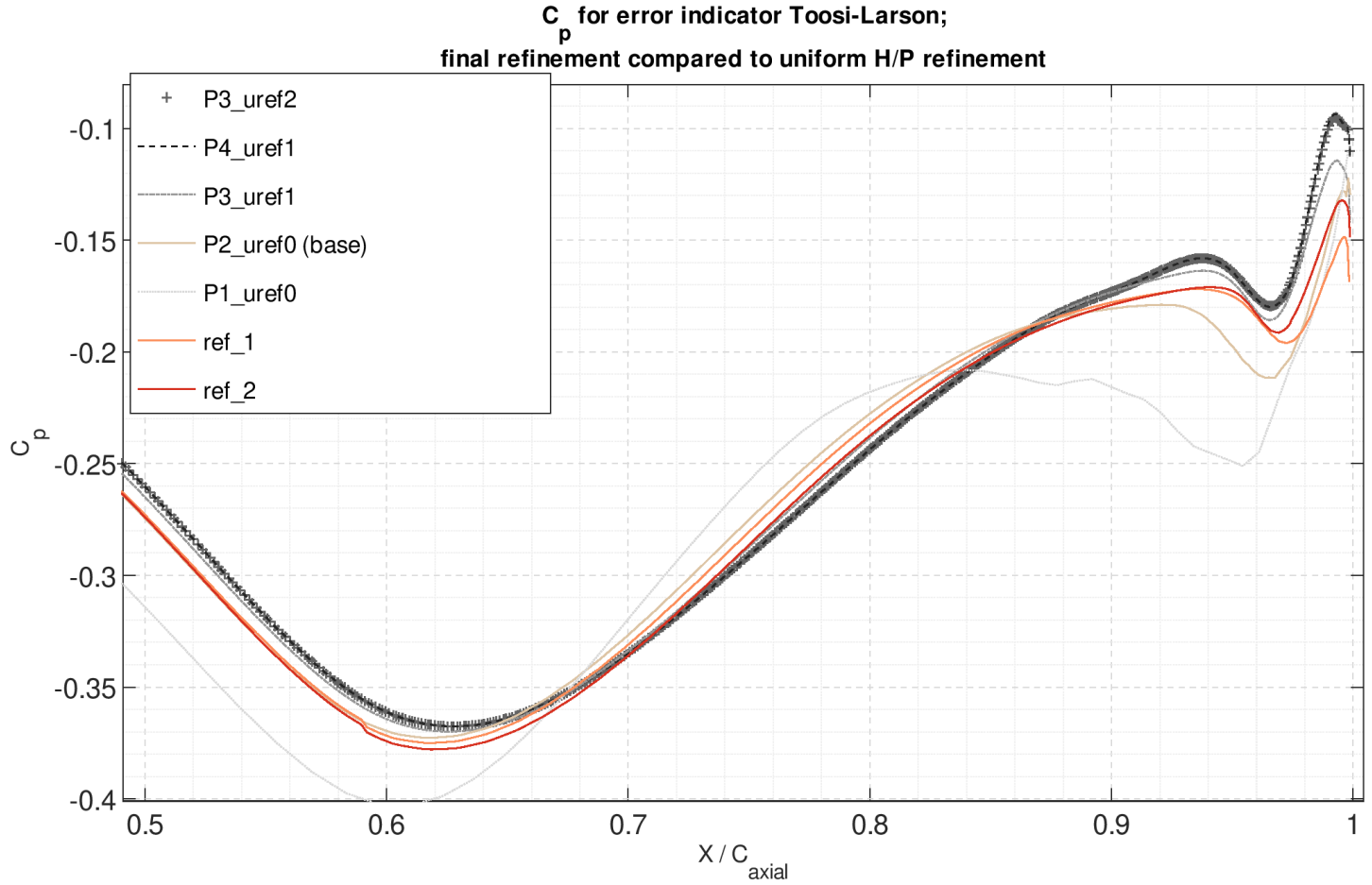



Figure 4.34: $C_p$ distribution from ${}^{\text{T.L.err}}\mathcal{E}$ along the upper surface of the turbine blade. The refinement sequence #0, #1, #2 is colored yellow, orange, & red, respectively.

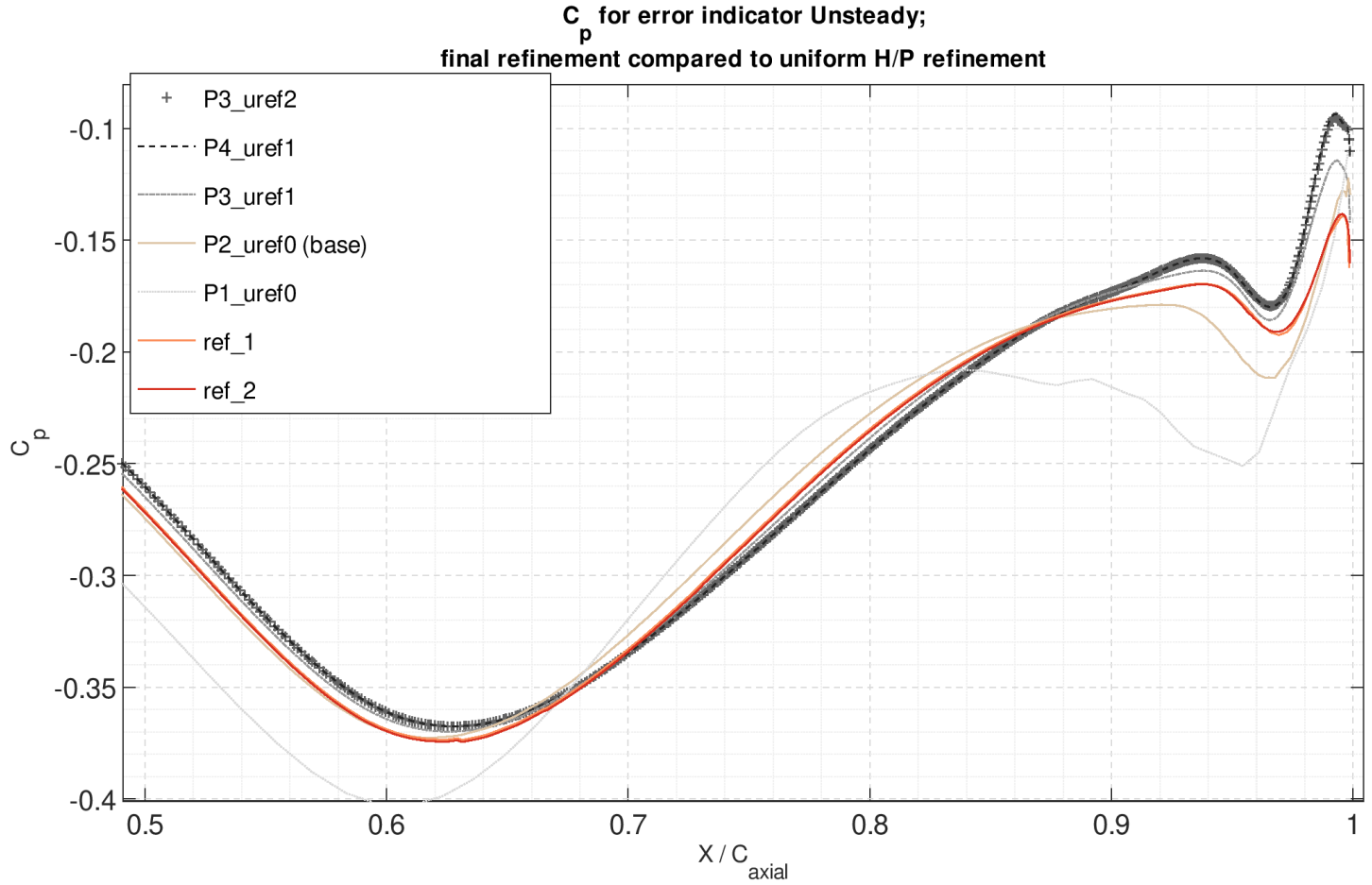



Figure 4.35: $C_p$ distribution from ${}^{\mathbf{unStd}}\mathcal{E}$ along the upper surface of the turbine blade. The refinement sequence #0, #1, #2 is colored yellow, orange, & red, respectively.

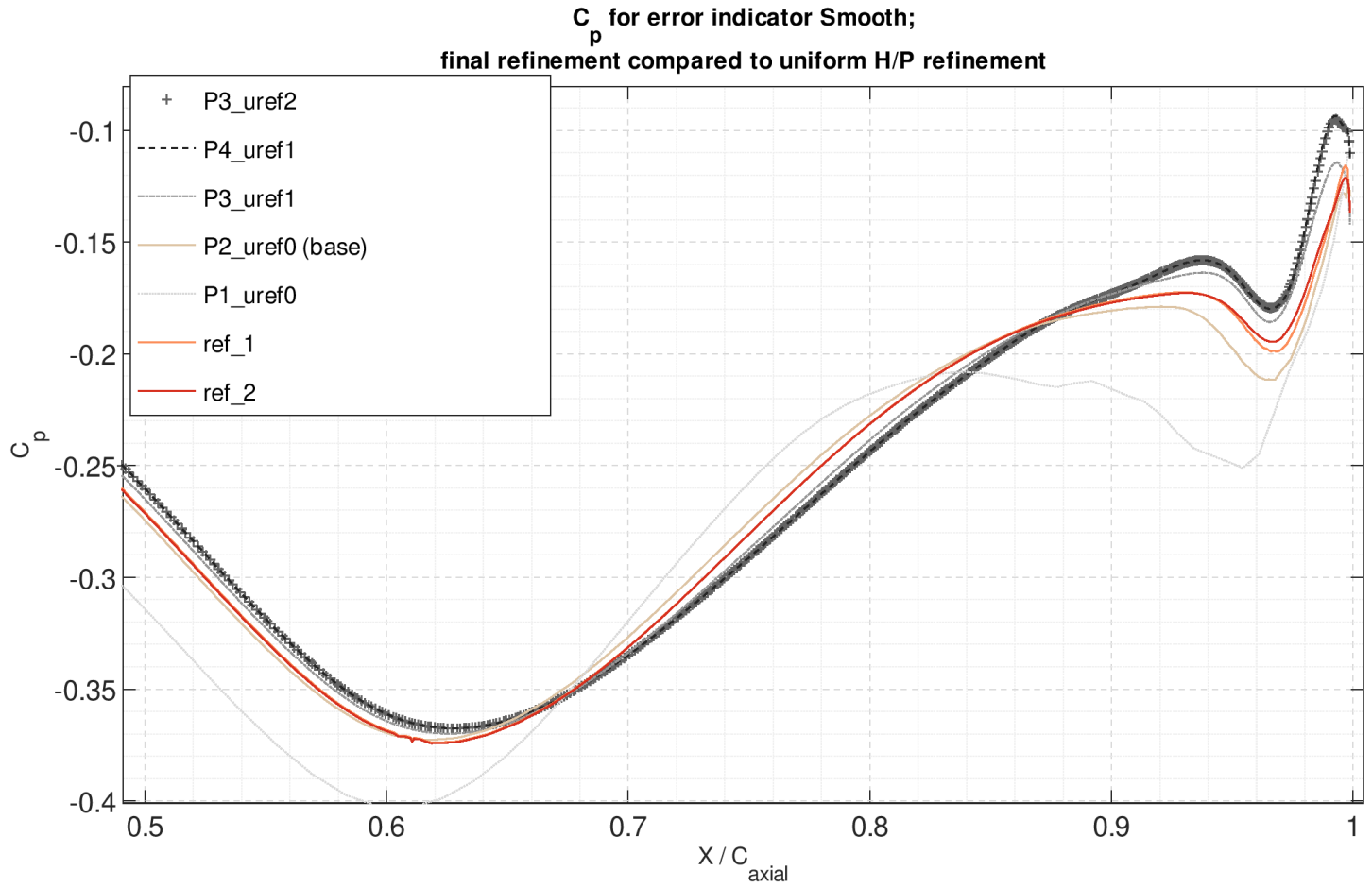
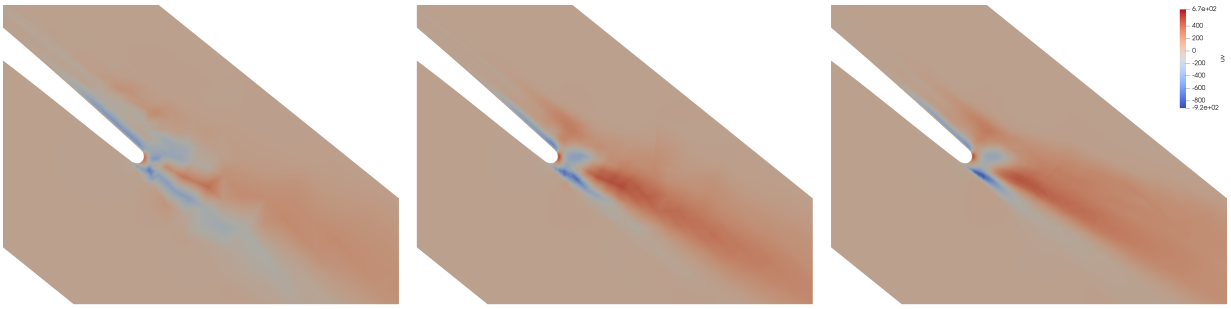
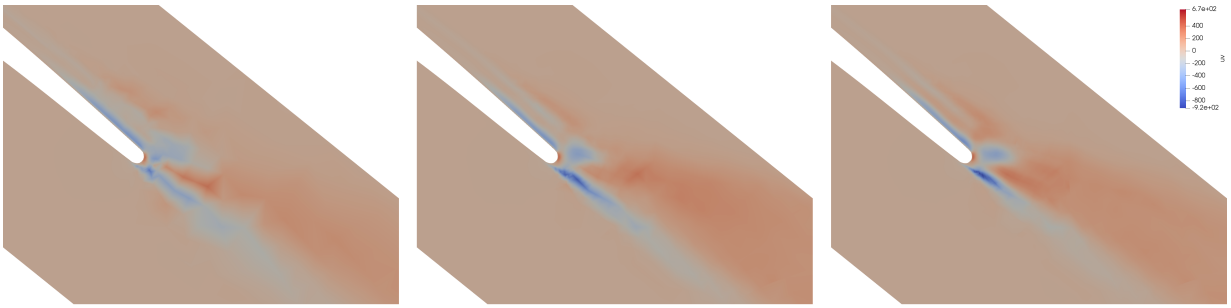


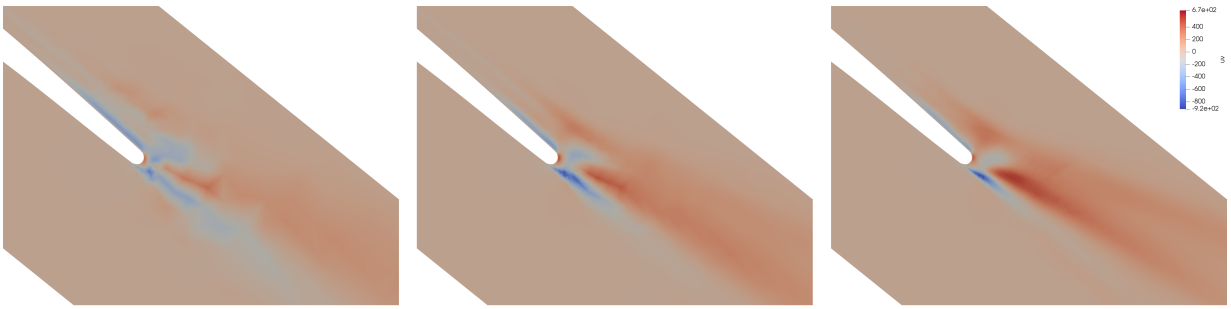

Figure 4.36: $C_p$ distribution from ${}^{\text{smth}}\mathcal{E}$ along the upper surface of the turbine blade. The refinement sequence #0, #1, #2 is colored yellow, orange, & red, respectively.



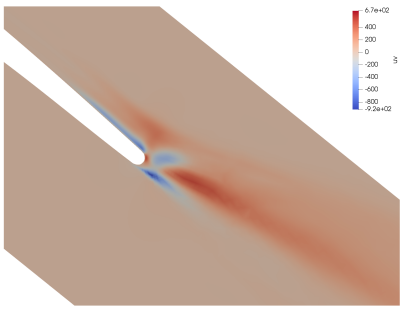
(a) from error indicator $^{\mathbf{unStd}}\mathcal{E}$



(b) from error indicator $^{\mathbf{smth}}\mathcal{E}$

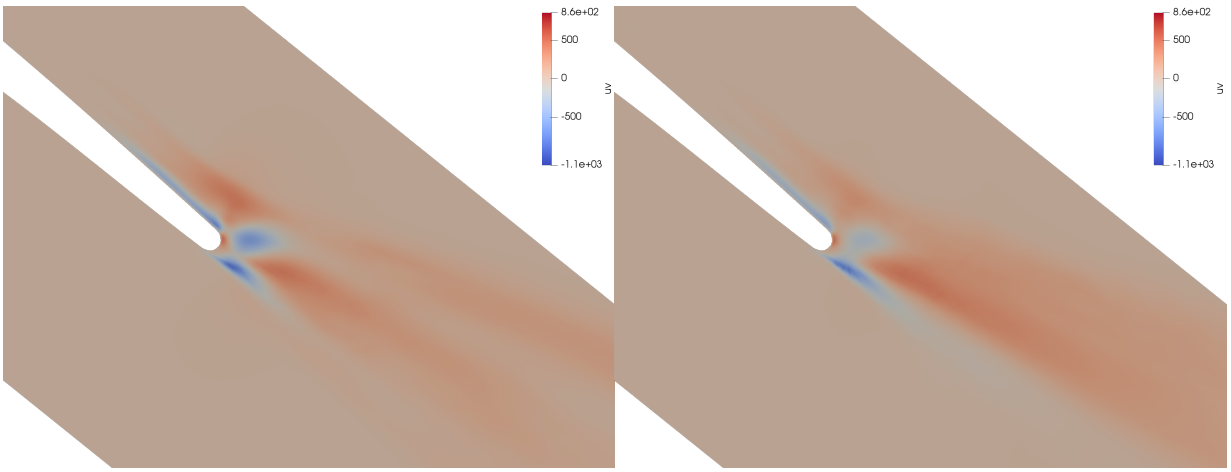


(c) from error indicator $^{\mathbf{T.L.err}}\mathcal{E}$



(d) truth comparison

Figure 4.37: Progression of the horizontal cross term of the Reynolds stress tensor, from the coarse mesh on the left to the fine mesh on the right




(a) “P4_uref1”

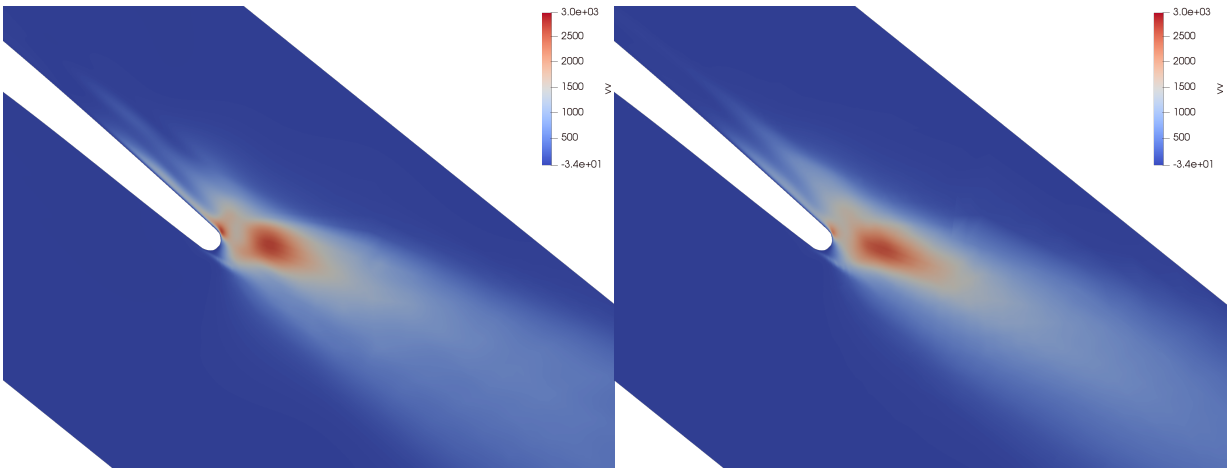
(b) $^{\mathbf{unStd}}\mathcal{E}$

(c) $^{\mathbf{smth}}\mathcal{E}$

(d) $^{\mathbf{T.L.err}}\mathcal{E}$

(e) base mesh

Figure 4.38: Reynolds stress UV component for refinement #2.




(a) “P4_uref1”

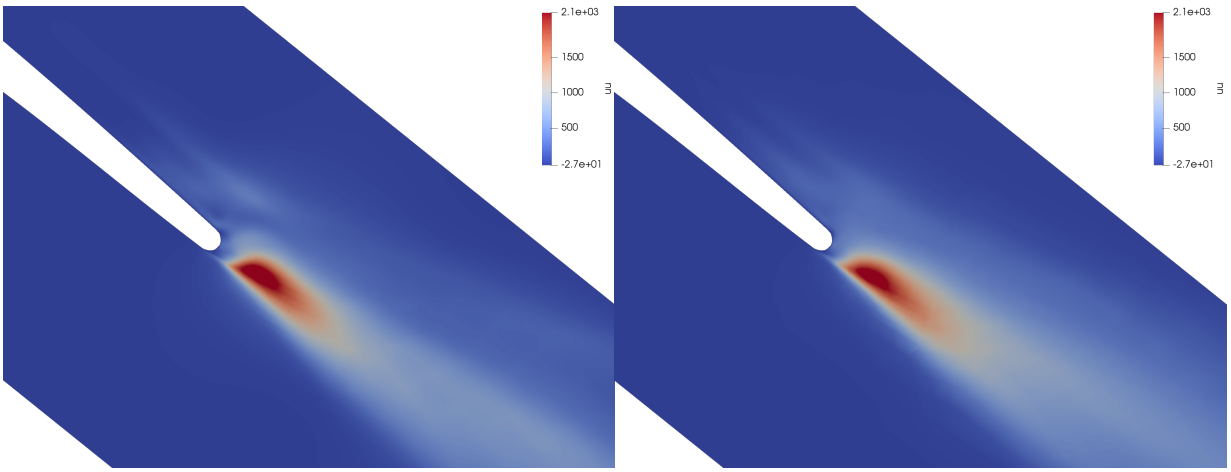
(b) $^{\mathbf{unStd}}\mathcal{E}$

(c) $^{\mathbf{smth}}\mathcal{E}$

(d) $^{\mathbf{T.L.err}}\mathcal{E}$

(e) base mesh

Figure 4.39: Reynolds stress VV component for refinement #2.




(a) “P4_uref1”

(b) $^{\mathbf{unStd}}\mathcal{E}$

(c) $^{\mathbf{smth}}\mathcal{E}$

(d) $^{\mathbf{T.L.err}}\mathcal{E}$

(e) base mesh

Figure 4.40: Reynolds stress UU component for refinement #2.

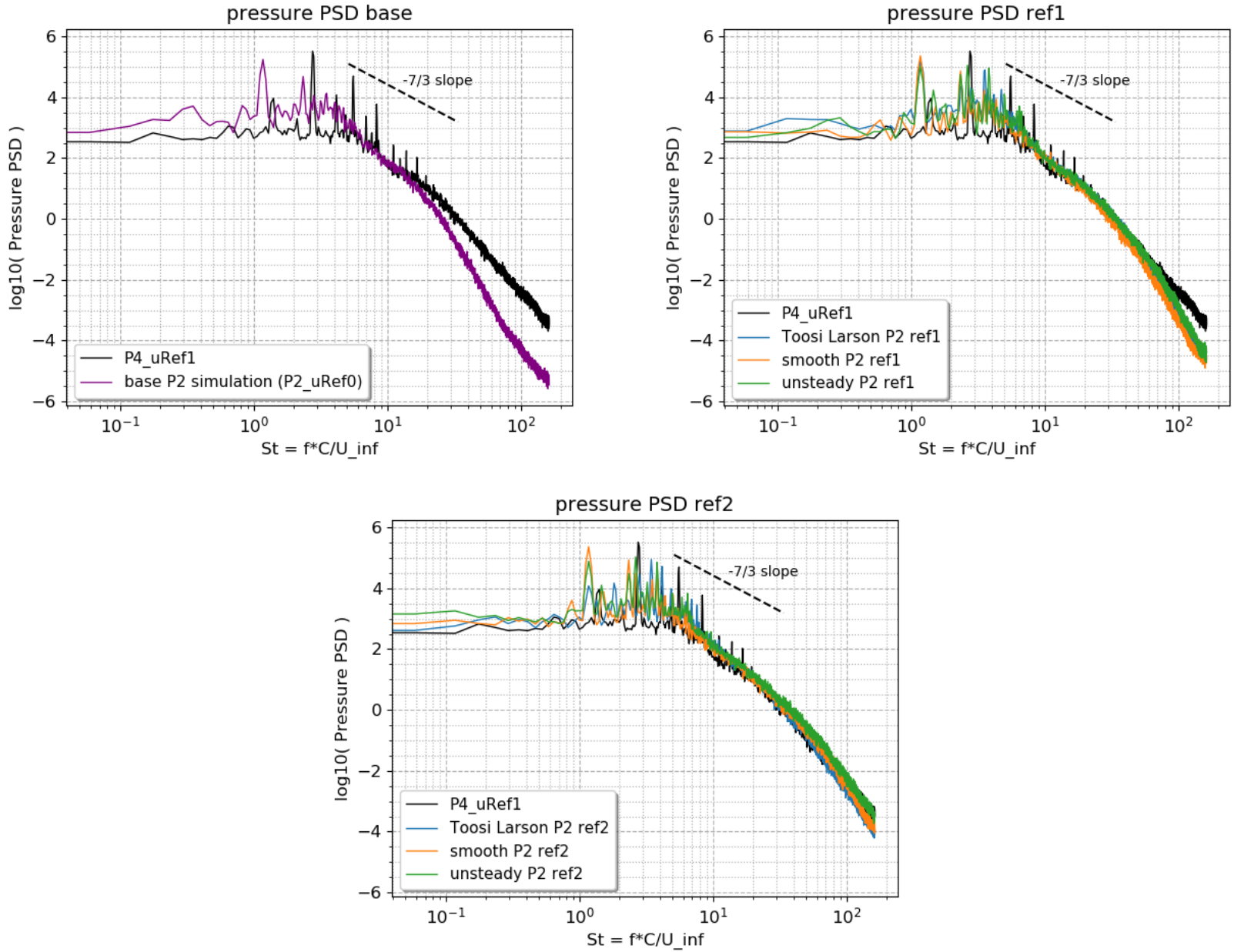



Figure 4.41: Power spectral density plots of the pressure fluctuations within the turbulent wake, sampled at the monitor point shown in Figure 4.11.

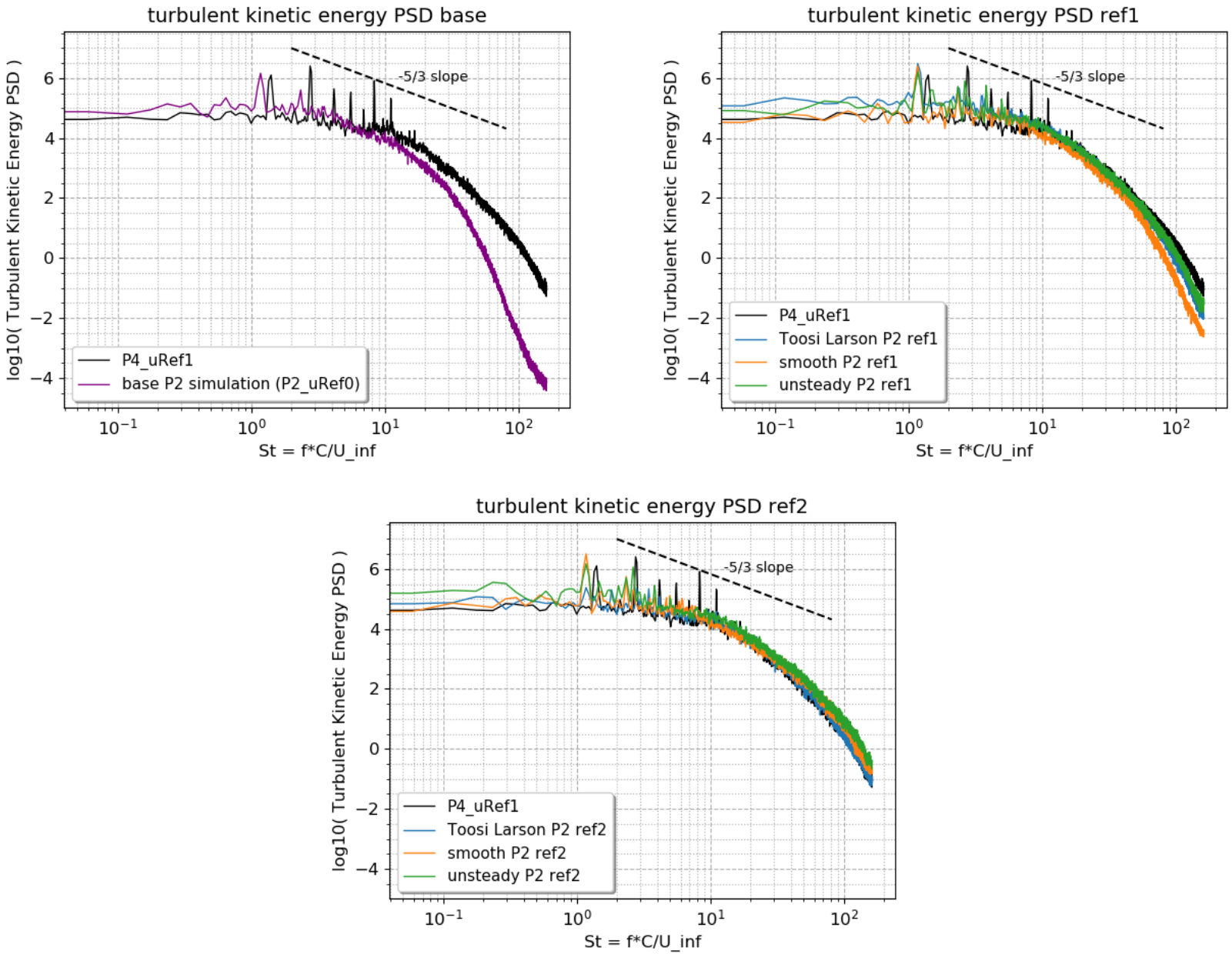



Figure 4.42: Power spectral density plots of the turbulent kinetic energy fluctuations within the turbulent wake, sampled at the monitor point shown in Figure 4.11.

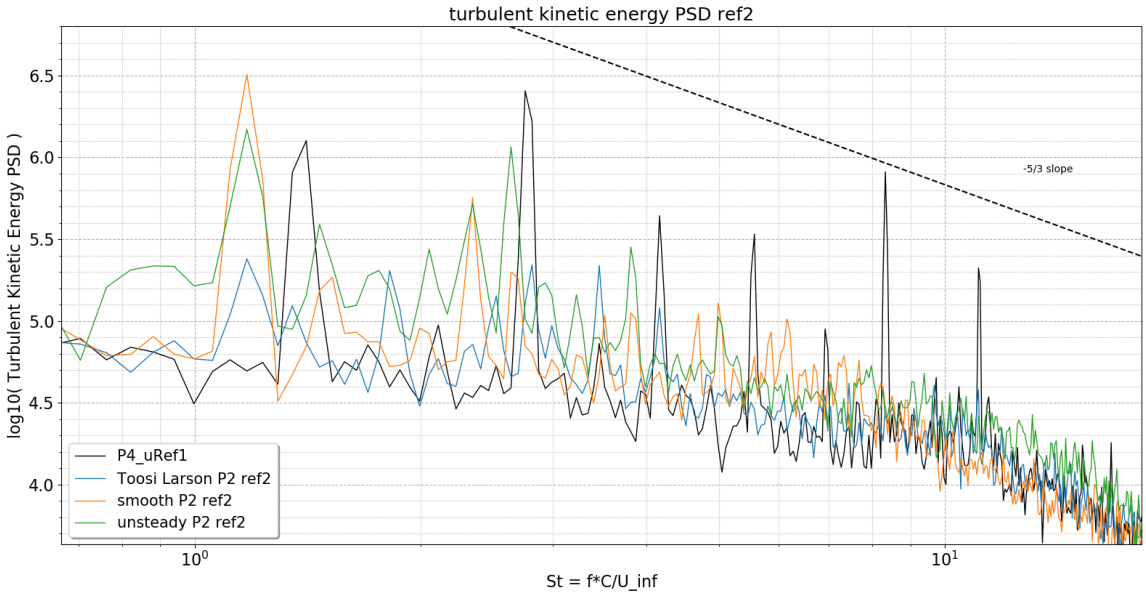



Figure 4.43: Zoomed view of the spikes in the turbulent kinetic energy spectra from refinement #2. The 'P4_uref1" truth reference is drawn in black. $^{\mathbf{T.L.err}}\mathcal{E}$, $^{\mathbf{smth}}\mathcal{E}$, and $^{\mathbf{unStd}}\mathcal{E}$ are drawn in blue, orange, and green, respectively.

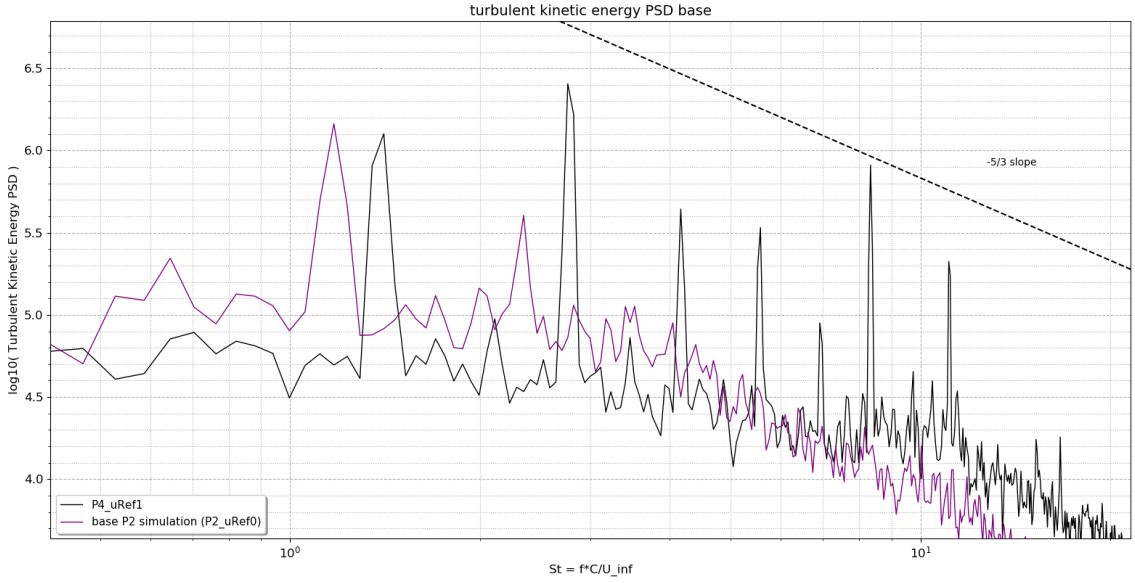



Figure 4.44: Zoomed view of the spikes in the turbulent kinetic energy spectra from refinement #0. The 'P4_uref1" truth reference is drawn in black. The base simulation is drawn in purple.

### 4.2.5 Assessment

All three error estimators improved the flow field, as evidenced by 1) the entropy images, 2) the pressure distribution, 3) the $C_{\mathrm{L}}$ value, 4) the Reynolds stresses, and 5) the high-frequency part of the energy spectra. Improvement was either *not* visible or *negative* in 1) the $C_{\mathrm{D}}$ value and 2) the low-frequency part of the energy spectra. The metrics that did not improve are indicative of under-resolved flow field dynamics. In other words, there is an area of the domain that requires further H-adaptation for the fluid's self-interaction to manifest. It is not possible to say where the under-resolved area is. The under-resolved area may be the region around the sharp corner geometry quirk previously mentioned. The under-resolved area might also be a section of the detached flow's path, impinging on the turbulent wake. As a future study, it would be interesting to find the key location and to see if the error estimators would eventually find it.

Without the flow field dynamics fully resolved, it would be hasty to designate a definitive winner, but a relative judgment is possible, founded upon how far the error estimators progressed by the end of the experiment. The result is consistent with the previous test case. Namely, the most capable error estimators appears to be ${}^{\mathbf{T.L.err}}\mathcal{E}$ and ${}^{\mathbf{unStd}}\mathcal{E}$. ${}^{\mathbf{T.L.err}}\mathcal{E}$ is tentatively deemed to be in the lead because it alone was continuing to make adjustments to the flow dynamics at the end of the experiment. The lest capable error estimator is clearly ${}^{\mathbf{smth}}\mathcal{E}$.

## 4.3 Discussion

For both the T106c and the T106a test cases, error indicators ${}^{\mathbf{T.L.err}}\mathcal{E}$, ${}^{\mathbf{unStd}}\mathcal{E}$, and ${}^{\mathbf{smth}}\mathcal{E}$ all produced workable meshes that boosted resolution of the turbulence. However, the indicators were not equally quick in converging the simulations to truth. Indicator ${}^{\mathbf{smth}}\mathcal{E}$ was consistently the slowest while ${}^{\mathbf{T.L.err}}\mathcal{E}$ and ${}^{\mathbf{unStd}}\mathcal{E}$ were in close competition for fastest. Upon examination, it is found that these performance dissimilarities are consistent with the underlying mathematical definitions of the error indicators.

For each indicator, it is possible to identify a built-in sensitivity to a particular mathematical

characteristic of the flow field, summarized thus:

- Error indicator ${}^{\mathbf{smth}}\mathcal{E}$, founded upon a shock-capturing gradient algorithm, picks out unsmooth regions of the flow, specifically areas with a high degree of spacial variability.

- Error indicator ${}^{\mathbf{T.L.err}}\mathcal{E}$, founded upon an estimate of small-scale turbulent kinetic energy, picks out energy dense regions, specifically areas rich in under-resolved vortex motion.

- Error indicator ${}^{\mathbf{unStd}}\mathcal{E}$, founded upon a subtractive difference of the unsteady flow residual, picks out "active" regions of the flow, specifically areas that are temporally very dynamic.

It is possible to trace how these built-in sensitivities lead naturally to the unique mesh adaptations produced by the error indicators. In particular:

- Indicator ${}^{\mathbf{smth}}\mathcal{E}$ h-refines the areas of high gradient flow, e.g. in boundary layers and where front lines forcefully collide.

- Indicator ${}^{\mathbf{T.L.err}}\mathcal{E}$ h-refines the areas of vortex decay, e.g. along vortex cascades and within turbulent wakes.

- Indicator ${}^{\mathbf{unStd}}\mathcal{E}$ h-refines the areas of heaviest "action," e.g. a blend of the vortex dense areas and the high-gradient regions.

As for the differences in speed of convergence, two explanations are offered. The first explanation stems from the immediate observation that the faster error indicators are also the more vortex sensitive. Vortex sensitivity may assist convergence by driving the adaptation process to quickly resolve the major churning motions in the flow, helping the adaptation process correctly prioritize between large scale features and small scale detail. Supporting this view is the observation that ${}^{\mathbf{T.L.err}}\mathcal{E}$, the most vortex sensitive indicator of the group, was also the indicator producing the most visually complete-looking eddy patterns. ${}^{\mathbf{unStd}}\mathcal{E}$, the second most vortex sensitive indicator, was also the second most visually complete. In contrast, ${}^{\mathbf{smth}}\mathcal{E}$, the least vortex sensitive indicator, produced noticeably lopsided-looking pictures with very high resolution near walls but low resolution

at active areas away from walls. Thus, it would appear that ${}^{\mathbf{T.L.err}}\mathcal{E}$, and to a slightly lesser extent ${}^{\mathbf{unStd}}\mathcal{E}$, both optimized resolution across the entire flow field while ${}^{\mathbf{smth}}\mathcal{E}$ incorrectly focused on some spots to the detriment of others.

The second explanation stems from the math observation that both ${}^{\mathbf{unStd}}\mathcal{E}$ and ${}^{\mathbf{T.L.err}}\mathcal{E}$ may be interpreted as actual measures of a CFD error-like quantity, defined as *the inconsistency between the actualized flow field and the underlying physics equations*. In fact, ${}^{\mathbf{unStd}}\mathcal{E}$ purports to actually be a direct measure (really an estimate) of this very quantity. It is literally the instantaneous flaw in the simulated flow field—i.e. a spatial map of where the Navier-Stokes equations (2.8) are not being followed. Similarly, ${}^{\mathbf{T.L.err}}\mathcal{E}$ is a spatial map of the turbulent motion too intricate for the simulation to resolve, and where the simulation cannot resolve the motion, it also cannot control the virtual fluid enough to enforce the virtual laws of physics. Per this view, the slowness of ${}^{\mathbf{smth}}\mathcal{E}$ is attributable to the weak relationship between the indicator's measured quantity (sharp gradients) and the simulation's physical realism. Although it is undeniably true that: 1) *A fine mesh is needed to resolve a sharp gradient*, and 2) *Turbulence will produce sharp gradients*, the presence of a sharp gradient is really only a warning that poor physics may be occurring in the vicinity.

# Chapter 5

# Extention to Machine Learning

As previously stated, CFD mesh design requires the hand of an experienced engineer. In point of fact, mesh design requires a *brain*. That observation raises the question: Might mesh adaptation be doable by artificial neural network (ANN)?

There are at least two reasons to recommend the use of ANNs for mesh adaptation. First, it is a feature of ANNs that a properly designed network can mimic any function (Pinkus, 1999). This property makes ANNs appropriate for mimicking semi-mysterious patterns which have no analytical formula. Examples are self-driving cars, image recognition, language translation, and the biological cycle prediction. CFD mesh adaptation would seem to be a similar type of puzzle. Second, the computational cost of a neural network is low compared to most operations in CFD, with the computational cost concentrated in the setup. Once an ANN has been designed and trained, it is cheap to run because the runtime calculation is merely matrix multiplication. So for mesh adaptation, a potential benefit of a neural network is that an expensive calculation can be replaced by a cheap one. Furthermore, because of the high commercial demand for ANNs, it is possible that future computer chips will incorporate dedicated circuitry for them, offering a further speed boost which the CFD community may wish to leverage.

The young discipline of *machine learning* has bloomed within the last decade, with the invention of Deep Learning, and is only now beginning to affect the CFD research community. The specific idea to apply machine learning to H-adaptation is so recent that publications on the topic are to be found only in the last couple of years and are numbered in the single digits. At KU, this topic of research was opened by a student MS thesis (Phommachanh, 2021), training an ANN to do AI-powered mesh adaptation. That work leaned heavily upon the knowledge and tools from

this present work, which was under development at the time, forging a link between that research and the research here. The topic of ML-guided mesh adaptation is both a natural extension to this dissertation and a suitable topic to close upon, so this final chapter will expand upon the work of Phommachanh (2021) by reporting the CFD mesh adaptation performance of the trained ANN.

## 5.1 Related Work

So far, no two researchers have used a neural network in the same way. The three most relevant papers to H-adaptation will be reviewed here. The first paper by Tingfan et al. (2022) uses a neural network to assist R-adaptation by interpolating a CFD flow field. The second paper by Yang et al. (2021) trains a *hypernetworks*, a linked series of ANNs, to drive H-adaptation for advection problems. The third paper by Fidkowski & Chen (2021) trains an ANN to replace a sub-algorithm in a complex H-adaptation procedure.

### 5.1.1 R-Adaptation with the Assistance of an Artificial Neural Network

Tingfan et al. (2022) deployed an ANN in an attempt to speed R-adaptation by lowering the computational burden of the mesh update iteration. The mesh update iteration normally requires either intermittent execution of the CFD solver to adjust the flow field to the new mesh — which is time consuming — or it requires execution of an interpolation algorithm — which is inaccurate. The proposal was to update the flow field by way of a neural network, achieving accuracy at low cost. It is worth noting that the feasibility of this proposal is debatable because the maneuver requires that the ANN be trained on the initial flow field, a lengthy process that could negate any speed benefit. Nevertheless, Tingfan et al. reported success with the type of flow fields tested, which were extremely simple: 2 dimensional, static, and non-turbulent. The authors also found that a seven layer NN outperformed the accuracy of inverse distance weighted interpolation.

The paper leaves open the question of generalizability. In particular, it is not clear that the proposed network's size and topology will be sufficient for complex flow fields. That means the

network might require re-crafting on a case-by-case basis. In fact, the paper's data suggests the likelihood of this flaw is high. It reports that when the ANN was transitioned between artificial flow fields of increasing complexity, the ANN required expansion. Another weakness is that the training data is limited by design to the informational content of the flow field, casting doubt on the degree of accuracy compared to fancy interpolation methods. That said, neither flaw detracts from the demonstrated potential of an ANN to memorize a complex image and extrapolate embedded patterns. Furthermore, both the problem of generalizability and accuracy might be solvable in the figure with a physics-informed neural network, trained on the actual flow physics of Navier-Stokes (Raissi et al., 2019).

### 5.1.2 H-Adaptation by Hypernetworks and Graph Neural Networks

Yang et al. (2021), sought to apply *hypernetworks*[1] and *graph neural networks*[2] to the task of H-adaptation, producing meshes for time-dependent advection. The goal of the research was to produce a generalizable ANN that could adapt meshes for simulations dissimilar from the training data.

The primary contribution of the research was a new way to conceptualize the underlying math problem, which made it possible to use neural networks in the desired way. The adaptation procedure needed to produce a sequence of meshes, tuned for the unknown advection field at each timestep. The proposed approach was to envision the task of finding the ideal mesh sequence as a probabilistic Markov decision process, where the infinite variety of possible meshes constitutes an abstract space of probabilities. This creative idea re-cast the job of the ANN from mesh-adapter to pilot. The ANN needed to calculate an abstract path through the space of probabilities, yielding the best possible sequence of meshes.

After presenting the re-conceptualization of the math problem, the researchers leveraged prior

[1] A *hypernetwork* is an extended version of the normal ANN, featuring a sub-network that controls the between-neuron weights of another ANN (Ha et al., 2016).

[2] A *graph neural network* is a type of ANN specialized to process data in graph form (Scarselli et al., 2008). "Graph" is referring to a linked network of nodes, not an image.

work in hypernetworks and graph neural networks to craft several ANNs capable of handling the needed sequential decision-making. As a final test of those ANNs, the researchers set up a simple training exercise with an advection simulation.

They researchers crafted a group of simple truth fields for which exact $L_2$ errors could be calculated. They used very simple geometric shapes: bumps, circles, a flat plane, and stair-steps. The ANNs were presented with advection simulations of the fields on auto-repeat, allowing the ANNs to explore the effects of their refinement decisions. The result was reportedly several ANNs that could drive the H-adaptation process. However, no attempt was made to try out the final ANNs on any complex problems.

### 5.1.3 ANN assisted Adjoint-Adaptation for CFD

Fidkowski & Chen (2021) proposed a four-part adaptation procedure, consisting of: 1) an adjoint-based error estimator, 2) an interpolation-based error estimator, 3) an ANN, and 4) the 2D mesher BAMG.[3]. With a dizzyingly complex array of algorithm interconnections, the ANN was configured to take input from: 1) the adjoint estimator, and 2) the interpolation-based estimator, while funneling its output data to BAMG.[4] The entire complex was a copy of an earlier super-algorithm into which the ANN had been dropped, replacing a sub-component algorithm called MOESS.[5]

The training was successful, resulting in a cross-linked, multi-part adaptation procedure. In final analysis, Fidkowski & Chen (2021) claim that the ANN outperformed the original MOESS, exhibiting greater restraint when challenged by outlier inputs. Fidkowski & Chen also report that the ANN learned to calculate in one step what MOESS required many interactions to produce.

The complexity of the (Fidkowski & Chen, 2021) setup is daunting. With the additional 2D limitation of BAMG and the already high cost of the adjoint calculation, it is safe to conclude that the proposed machine is not close to the final solution for H-adaptation. However, the experiment

[3]Bi-dimensional Anisotropic Mesh Generator: (Hecht, 1998)

[4]BAMG itself was configured with dual input data streams, one from the ANN and the other direct from the adjoint estimator.

[5]Mesh Optimization via Error Sampling and Synthesis: (Carson et al., 2020)

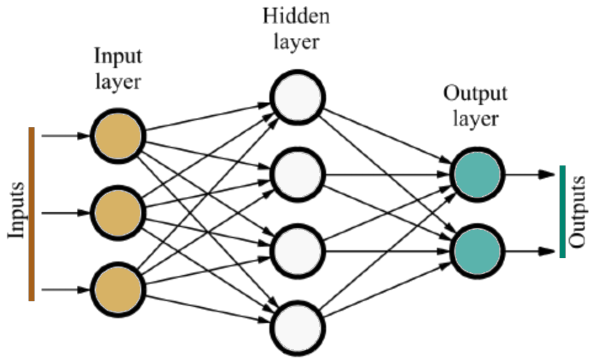



Figure 5.1: Illustration of a Feedforward Neural Network (FFNN). When there are two or more hidden layers, the network is to as a Multilayer Perceptron (MLP). (Reprinted, with edits from Phommachanh, 2021)

appears close to the optimal utilage of an ANN for the problem of H-adaptation. To summarize:

- Drop the ANN in place of an already existing algorithm for optimized meshing.
- Feed the ANN all potentially relevant data, spanning both geometry and flow field.
- Train the ANN to mimic the optimized meshes of the algorithm it is intended to replace.

## 5.2 Mesh Adaptation by Machine Learning

The experiment underlying Phommachanh's MS thesis is similar in style to the work of Fidkowski & Chen (2021), but the type of algorithm replaced by the ANN is different, the mesh type is different, the supporting adaptation framework is different, and so is the targeted CFD problem. The replaced algorithm was an error estimator from this dissertation, specifically $\overline{\mathcal{E}}^{\mathbf{T.L.err}}$, described in Chapter 2. The mesh type was 3D unstructured hexahedral. The supporting adaptation framework was the one described in Chapter 3. The targeted CFD problem was turbulence. What is reported next is the description of the complete experiment, beginning with a summary of the ANN and its training (Phommachanh, 2021) and then extending into the CFD testing of the trained ANN.

### 5.2.1 The Neural Network

The neural network was a fully connected Multilayer Perceptron (MLP), a type of Feed Forward Neural Network (FFNN) with more than one hidden layer. Figure 5.1 diagrams the basic structure. Data flows through the network left to right, per the direction of the arrows, passing from one layer of *neurons* to the next. At each neuron $n$, the incoming datums are uniquely weighted and summed. The group is then biased and passed to an *activation function*, producing a single output datum, per neuron, for the next layer of the network to operate upon. Activation functions come in many varieties. The important characteristic that they share, at least for the work here, is that of non-linearity, for it is the non-linearity of the activation function, in combination with the hidden layers, that imbibe the ANN with its capacity as a universal function approximator (Pinkus, 1999). The activation function used here is the LeakyReLU activation function, equation (5.2). The per-node datum weighting and biasing is mathematically represented by equation (5.1). In these formulas, bold symbols denote column matrices, and subscript $n$ refers to neuron ID number.

$$y_n = \phi_{\text{LeakyReLU}}\left( b_n + \mathbf{w}_n^{\mathrm{T}}\mathbf{x} \right) \tag{5.1}$$

$$\phi_{\text{LeakyReLU}}(\mathbf{x}) \equiv \begin{cases} \mathbf{x} & \text{for } \mathbf{x} \geq 0 \\ \alpha\mathbf{x} & \text{for } \mathbf{x} < 0, \quad \text{with } 0 < \alpha < 1 \end{cases} \tag{5.2}$$

In this work, $\alpha \equiv 0.03$.

As is the case for biological neural networks, an ANN requires *training* before use. During training, example data is passed into the ANN and the random errors that emerge are backpropogated through the neuron layers, tuning the per-node weights and biases, $\mathbf{w}_n$ and $b_n$. The training is akin to least-squares regression but instead of twisting a function of known form to fit laboratory data, the aim is to solve for the function itself. That function will form in the "mind" of the ANN, and it will never be written in equation form. Thus, the great power and weakness of ANNs is that they may be trained to mimic patterns which the trainer does not understand.

| Layer | Neuron Quantity |
|---|---|
| First | 256 |
| Second | 128 |
| Third | 64 |
| Fourth | 32 |

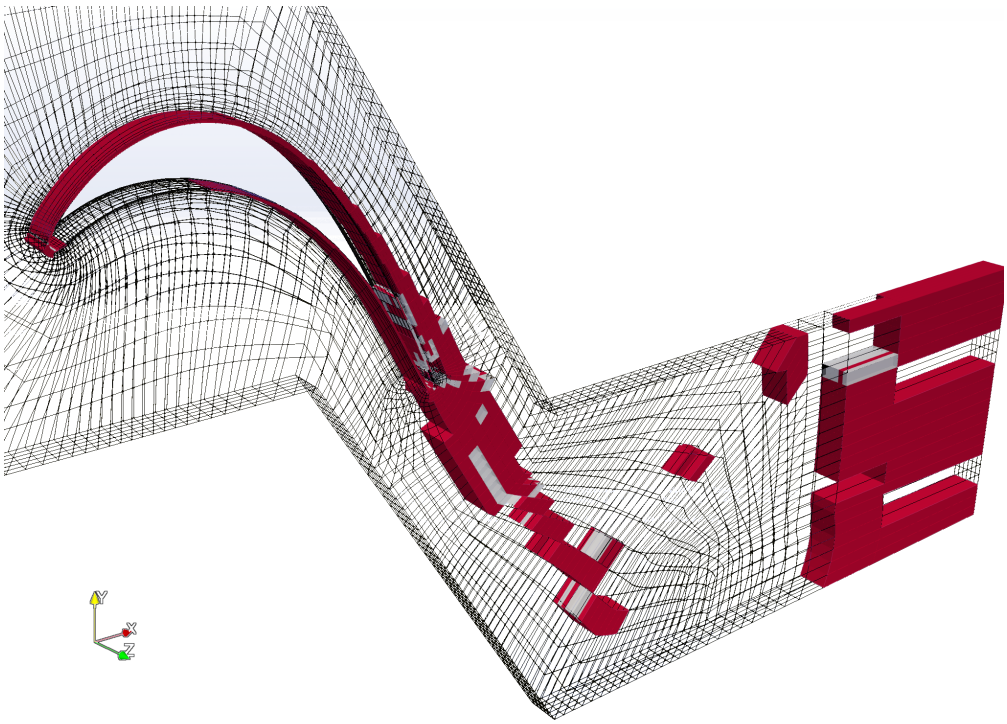
Table 5.1: Size of the neural network. (Phommachanh, 2021)

A danger with any regression problem, ANN training included, is that the math model will either overfit or underfit the data. That is, the math may have so much inherent flexibility that it will exactly mimic the training data, errors & all, failing to infer general patterns. Conversely, the math may be so rigidly inflexible that it also fails to extract the patterns. Although there is no guaranteed way to avoid the dangers, mitigation is possible. The common training technique is to split the training data into parts, reserving a portion purely for testing and a portion purely for validation.

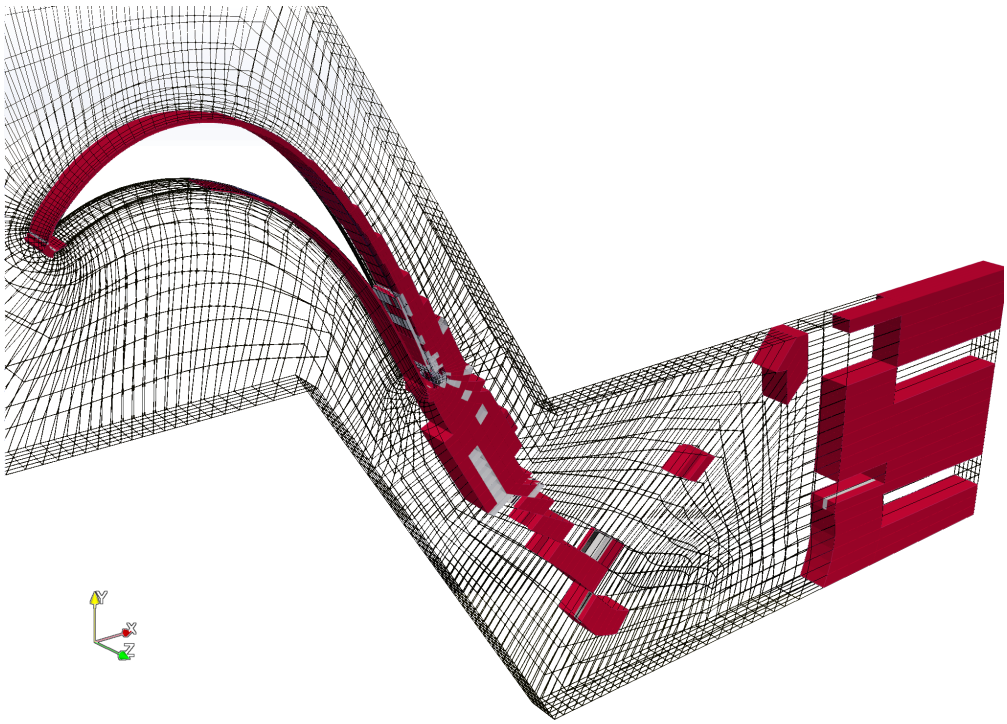
For the KU experiment, the goal was to train an ANN with size as reported by Table 5.1 to mimic the error estimators $\overline{^{\mathbf{T.L.err}}\mathcal{E}}$. The network was provided with per-cell data consisting of: 1) the cross cell distance and 2) the average velocity field.

Training data for the ANN was sourced from two meshes from the T106c simulation: the base mesh and the once H-adapted mesh from $^{\mathbf{T.L.err}}\mathcal{E}$. The list of 63,498 cells was split randomly, partitioned with 70% for training, 20% for testing, and 10% for validation. The ANN was trained to reproduce the raw error values assigned to each cell by error indicator $\overline{^{\mathbf{T.L.err}}\mathcal{E}}$, a task requiring 2000 training epochs (iterations). Batch size was 10, and the learning rate was $1.0 \times 10^{-5}$.

Figure 5.2 is an illustration of how well the final ANN was able to mimic the $\overline{^{\mathbf{T.L.err}}\mathcal{E}}$ error estimator upon one of the training meshes. The figure shows a wire frame view from mesh with solid red cubes marking cells that have been flagged for subdivision. The few solid white cubes mark cells that have been flagged indirectly, through the action of the smoothing algorithm 1, discussed in Chapter 3. Subfigure 5.2a shows the work of the formula version of $\overline{^{\mathbf{T.L.err}}\mathcal{E}}$ while Subfigure 5.2b shows the work of the ANN.



(a) flagged cells from the formula version of $\overline{\mathcal{E}}^{\mathbf{T.L.err}}$



(b) flagged cells by the ANN

Figure 5.2: Demonstration of the successful training of the ANN. Subfigure 5.2a draws the flagged cells upon the base mesh of T106c with the formula version of $\overline{\mathcal{E}}^{\mathbf{T.L.err}}$. Subfigure 5.2b draws the cells flagged by the trained ANN.

### 5.2.2 CFD Tests

The trained ANN-based error estimator was tested on both of the prior two CFD simulations: T106c and T106a. Presented next is an analysis of those simulations.

#### 5.2.2.1 T106c

Figure 5.3 contrasts the cell adaptations between the ANN-trained version of $\overline{\text{T.L.err}}\mathcal{E}$ and its predecessor formula $^{\text{T.L.err}}\mathcal{E}$. Significant differences between the refinement pattern are visible. While $^{\text{T.L.err}}\mathcal{E}$ refined in the unstable wake, $\overline{\text{T.L.err}}\mathcal{E}$ refined along the path of detached flow. The difference in the refinement pattern can mainly be attributed to the difference in input data type to the error estimators. $^{\text{T.L.err}}\mathcal{E}$ operates upon the instantaneous flow field velocity, while $\overline{\text{T.L.err}}\mathcal{E}$ operates upon the average flow field velocity. Because the average velocity field is *averaged*, it lacks the unstable turbulent wake. Consequently, the only detail left for $\overline{\text{T.L.err}}\mathcal{E}$ to act upon are the areas of rapid spatial change, which in this case is the path of the detached jet.

Figure 5.4 represents the lift and drag coefficient plots from Chapter 4 Section 4.1, adding the data from the ANN. The $C_{\text{L}}$ sequence from the ANN is low quality, but not worse than $^{\text{smth}}\mathcal{E}$. For $C_{\text{D}}$, the ANN error estimator is on par with the other error estimators.

Figure 5.5 plots the energy spectra from the ANN-based error estimator, with the pressure spectra at top and the turbulent kinetic energy spectra at the bottom. Both graphs show that the first round of refinement from the ANN boosted the turbulent resolution, but the later rounds of refinement did not. Referring back to Figure 5.3, it is easy to explain why the estimator seemed to stall. It was only at the step #1 refinement that the error estimator adapted in the vicinity of the turbulent wake. So although the error estimator was undoubtedly improving the flow field elsewhere for steps #2 and #3, it was not boosting resolution of the visible turbulence.

Finally, Figure 5.6 renders a sequence of the Reynolds stresses, the vv component. The bottom two subimages are of the DNS field (left) and the $^{\text{T.L.err}}\mathcal{E}$ field (right). The ANN error estimator clearly did well here, comparable to $^{\text{T.L.err}}\mathcal{E}$, in terms of matching DNS.

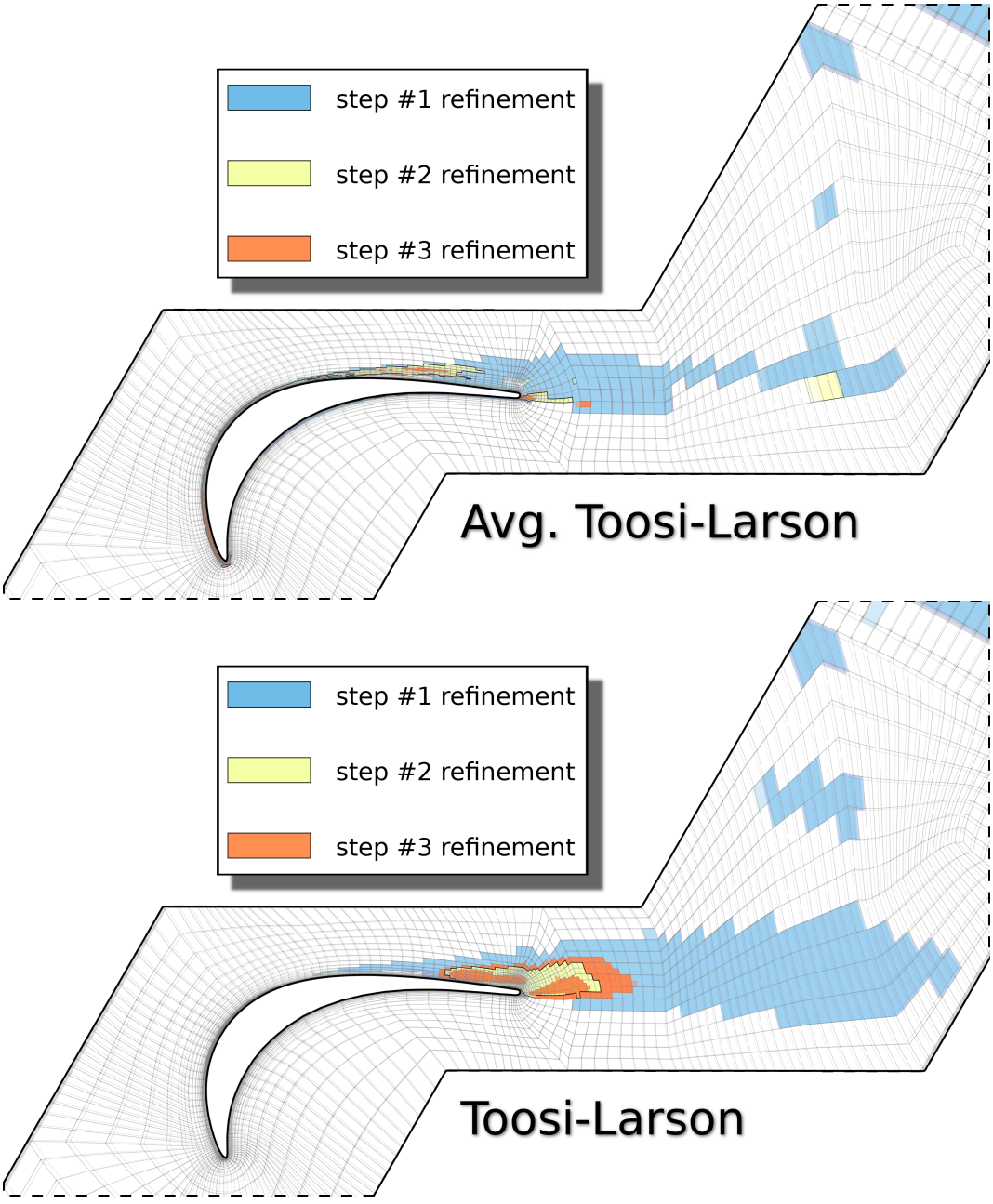
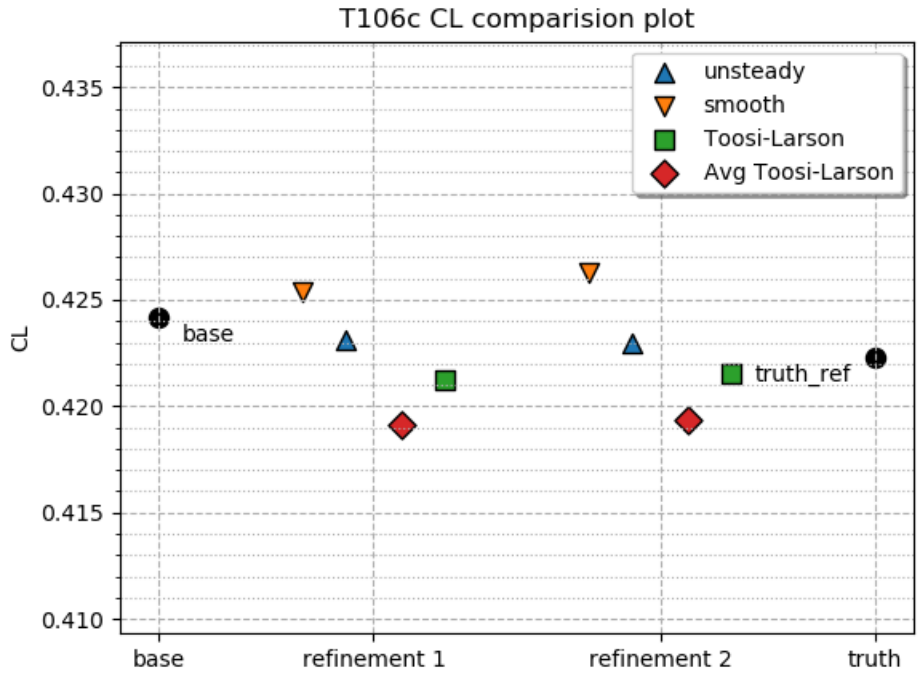
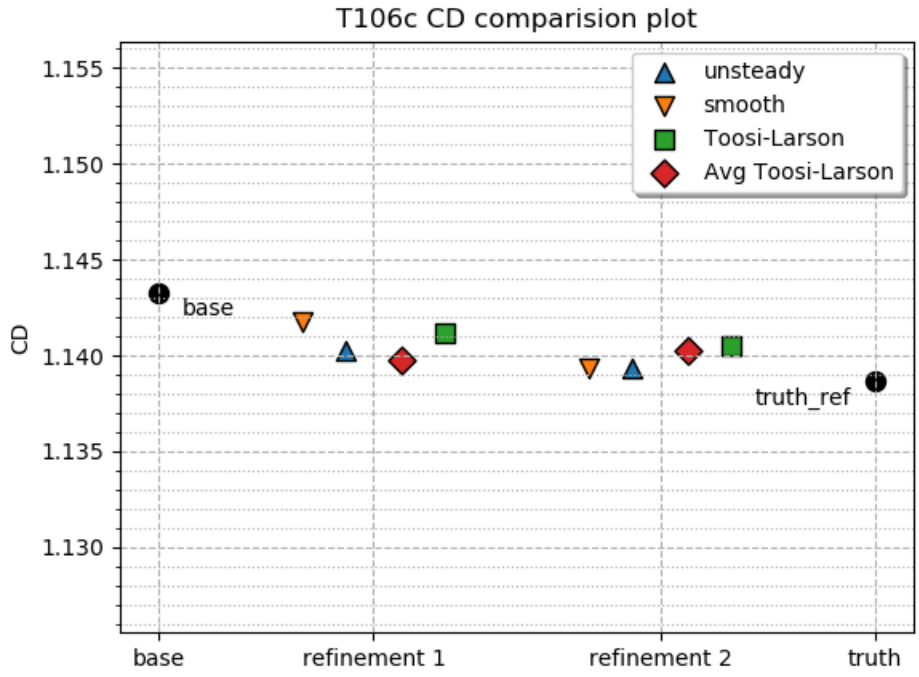



Figure 5.3: Zoomed view of the cells flagged from simulation T106c.




(a) Lift coefficients for T106c




(b) Drag coefficients for T106c

Figure 5.4: progression of the lift and drag coefficients for simulation T106c

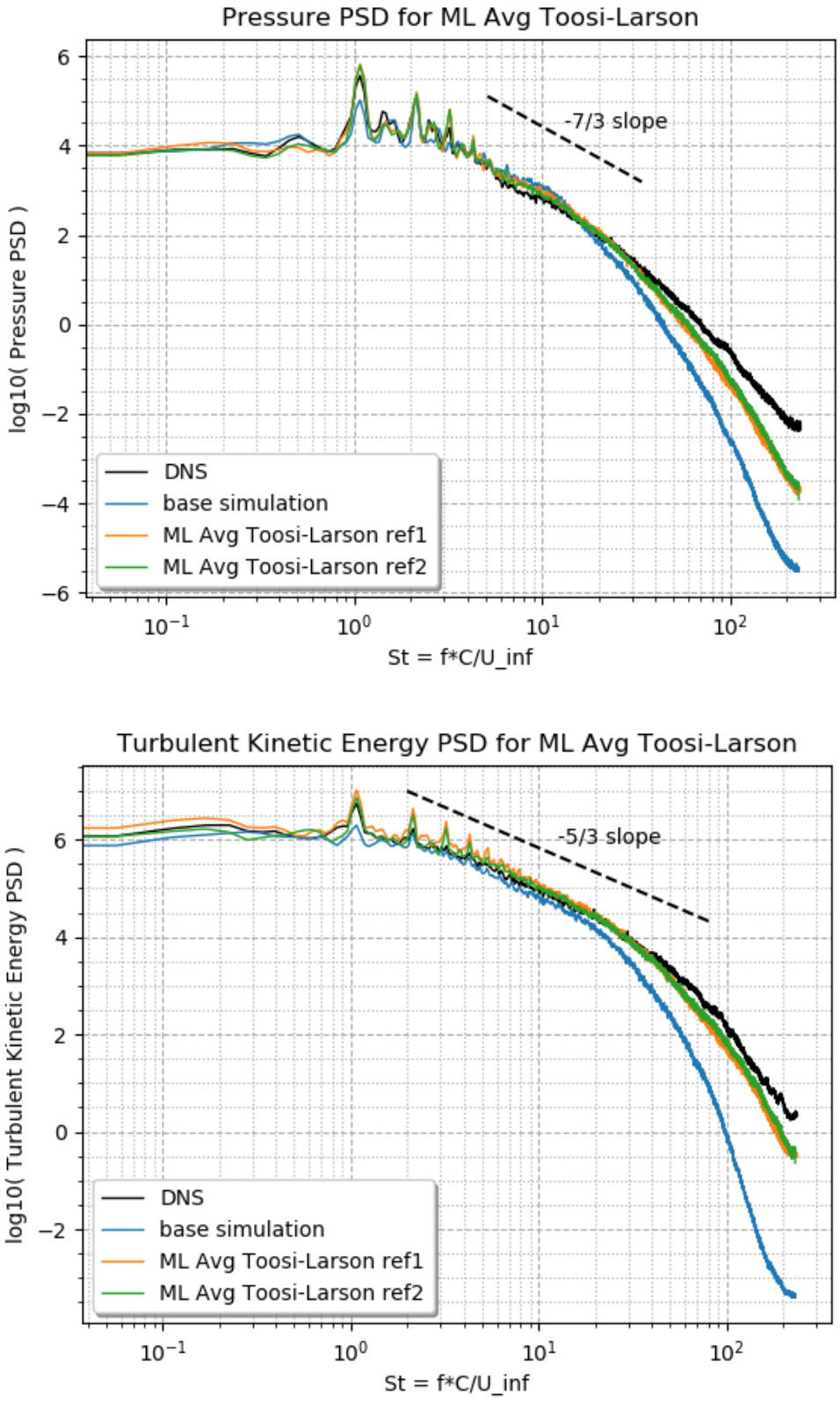
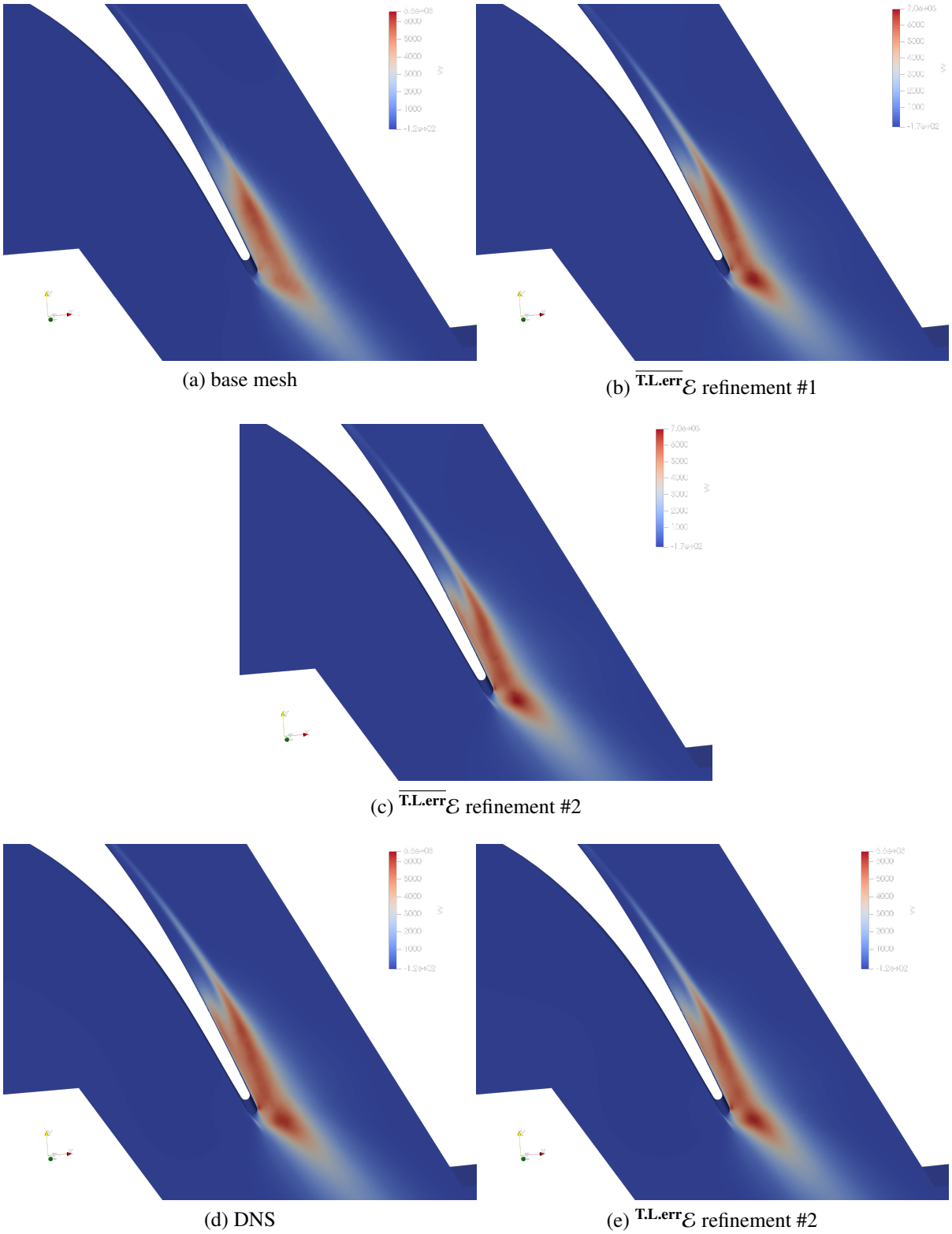



Figure 5.5: Progression of the energy spectra for T106c from the ANN.



(a) base mesh

(b) $\overline{\text{T.L.err}}\mathcal{E}$ refinement #1

(c) $\overline{\text{T.L.err}}\mathcal{E}$ refinement #2

(d) DNS

(e) $\overline{\text{T.L.err}}\mathcal{E}$ refinement #2

Figure 5.6: The first three sub-images show the progression of the VV component of Reynolds stress from error indicator $\overline{\text{T.L.err}}\mathcal{E}$ computed by the ANN upon simulation T106c. The two sub-images at the bottom, for comparison, are from the DNS simulation (left) and refinement #2 from $\overline{\text{T.L.err}}\mathcal{E}$.

#### 5.2.2.2 T106a

Figure 5.7 shows which cells were flagged by the ANN upon the T106a test case, contrasting with ${}^{\text{T.L.err}}\mathcal{E}$. It should be noted that the ANN was never trained upon any T106a, so the fact that it was able to operate at all upon T106a is already a measure of success. Comparing the flagged cells between the two error estimators, it is apparent that the ANN gave relatively little priority to the turbulent wake. While ${}^{\text{T.L.err}}\mathcal{E}$ refined three times over the entire wake, the ANN only performed similarly in the area immediately behind the blade. Instead, it targeted the path of the detached flow and also — inexplicably — intermittent groups of collinear cells along the blade boundary. The ANN also refined irregularly scattered patches above and below the blade. The haphazard patchwork gives the impression that at least some of the ANN's choices are due to randomness.

Figure 5.8 illustrates the effect wrought by the eclectic refinement pattern upon the lift and drag coefficients. Note that the plots contain the extra refinement round to illustrate the trend lines. The ANN's refinement choices have clearly put the simulation on a different trajectory than the other three error estimators. Whereas the other three error estimators move straight downward toward the truth reference on the $C_{\text{L}}$ plot, the ANN moves upward in what might be the beginning of an arc. On the $C_{\text{D}}$ plot, the other three error estimators move upward, overshooting the target, but the ANN moves downward and then angles up toward the truth reference. The ANN's trajectory puts it in last place for $C_{\text{L}}$ but at second place for $C_{\text{D}}$.

The ANN's success at $C_{\text{D}}$ suggests that it might be more successful than the other error estimators at uncovering the hidden dynamics of the T106a flow field. To find out, Figure 5.9 plots the energy spectra. In these plots, colors designate the different rounds of H-adaptation. Blue is the base simulation. Orange is the first found of H-adaptation. Green is the second round of H-adaptation. Black is the truth reference.

Unfortunately, the plots do not provide evidence that the ANN has discerned the elusive dynamics of the T106a flow. In fact, the low frequency peaks do not appear to improve *at all* as the ANN's H-adaptation progresses. However, in the high-frequency part of spectra, the ANN is clearly doing well at driving the spectral curves toward the truth line.

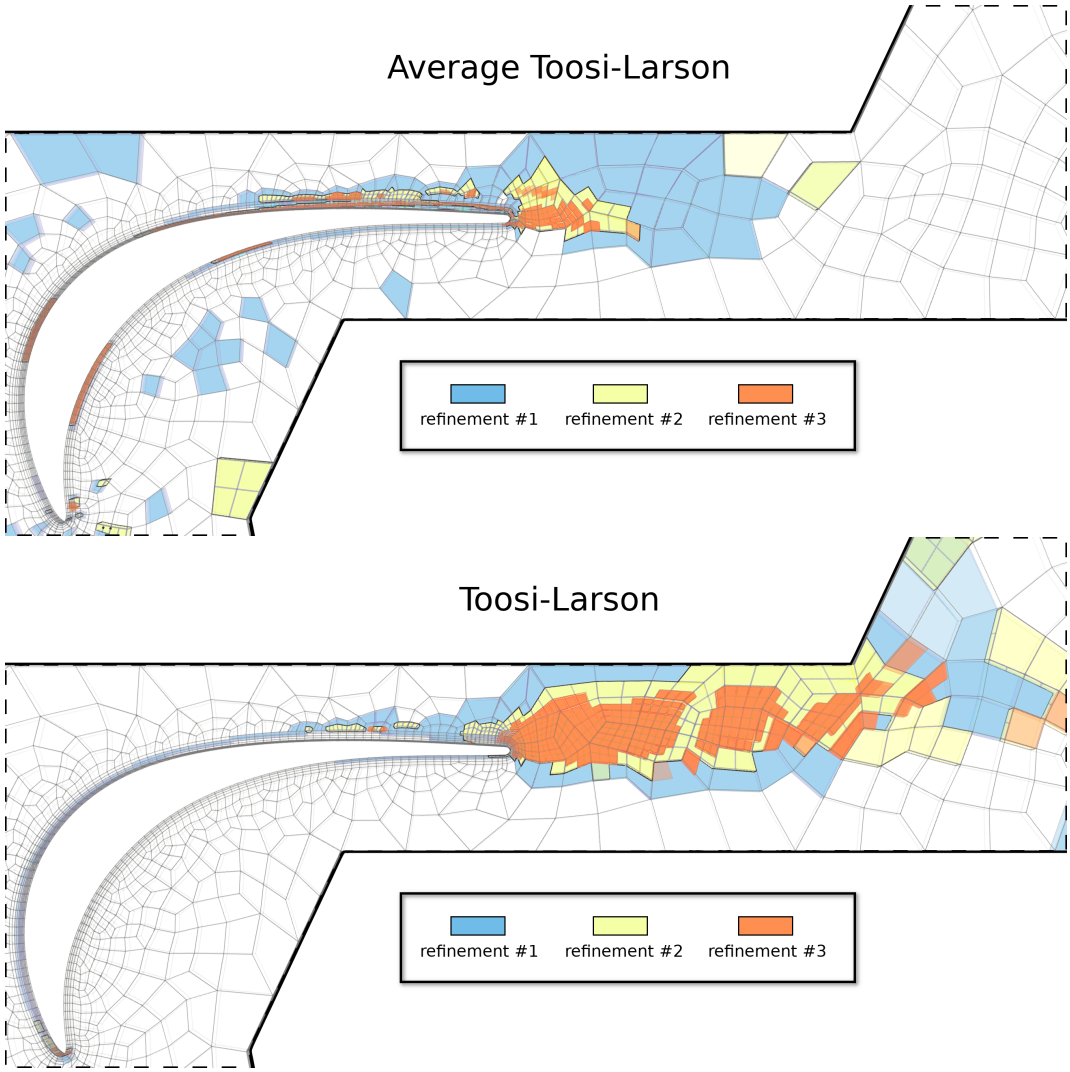
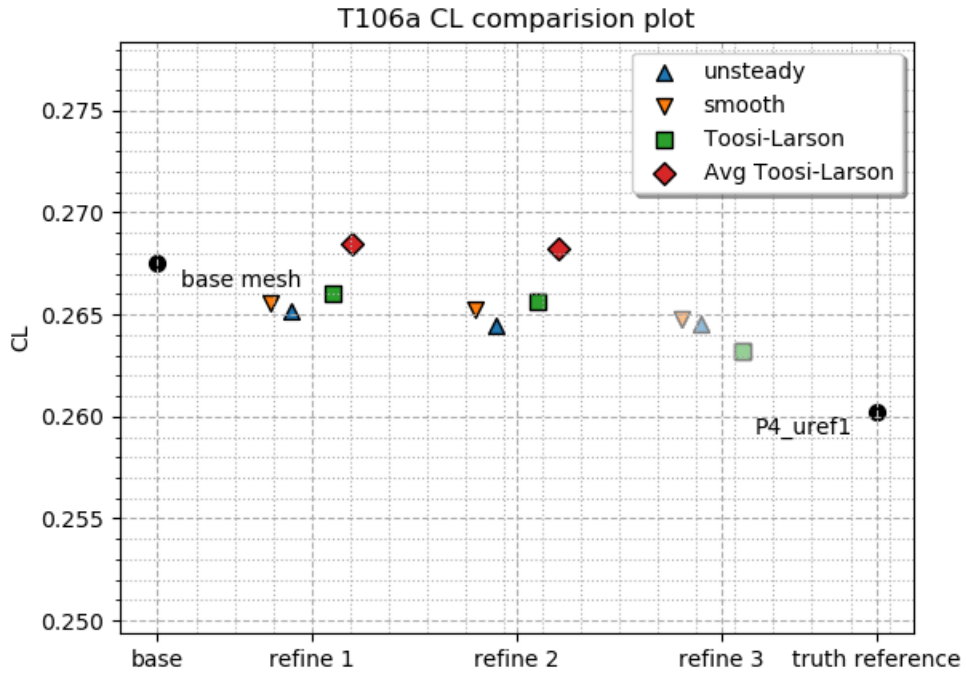
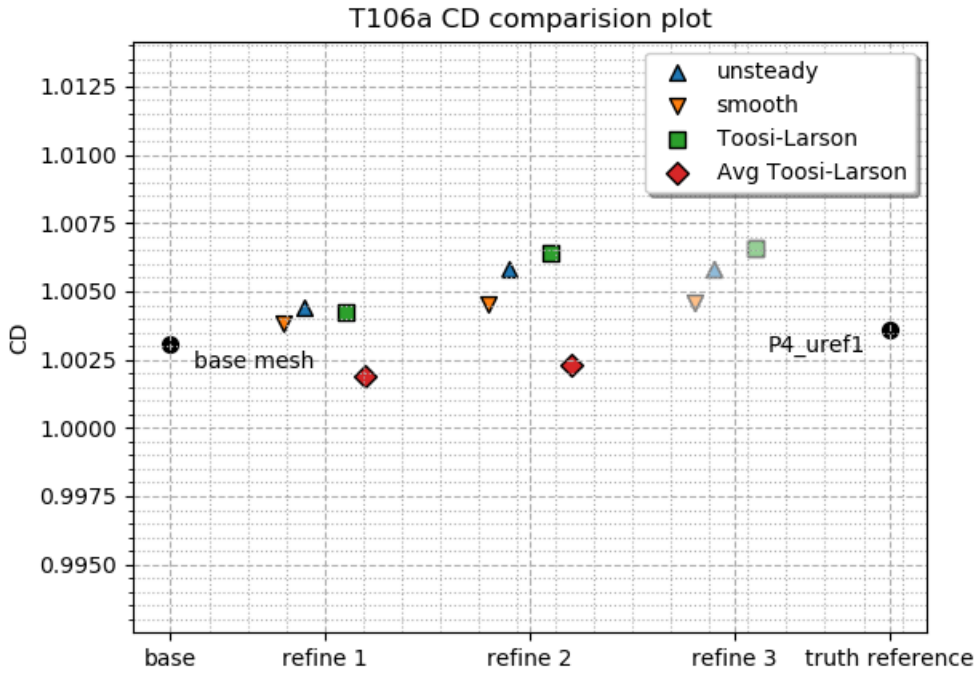



Figure 5.7: Zoomed view of the cells flagged from simulation T106a.




(a) Lift coefficients for T106a




(b) Drag coefficients for T106a

Figure 5.8: progression of the lift and drag coefficients for simulation T106a

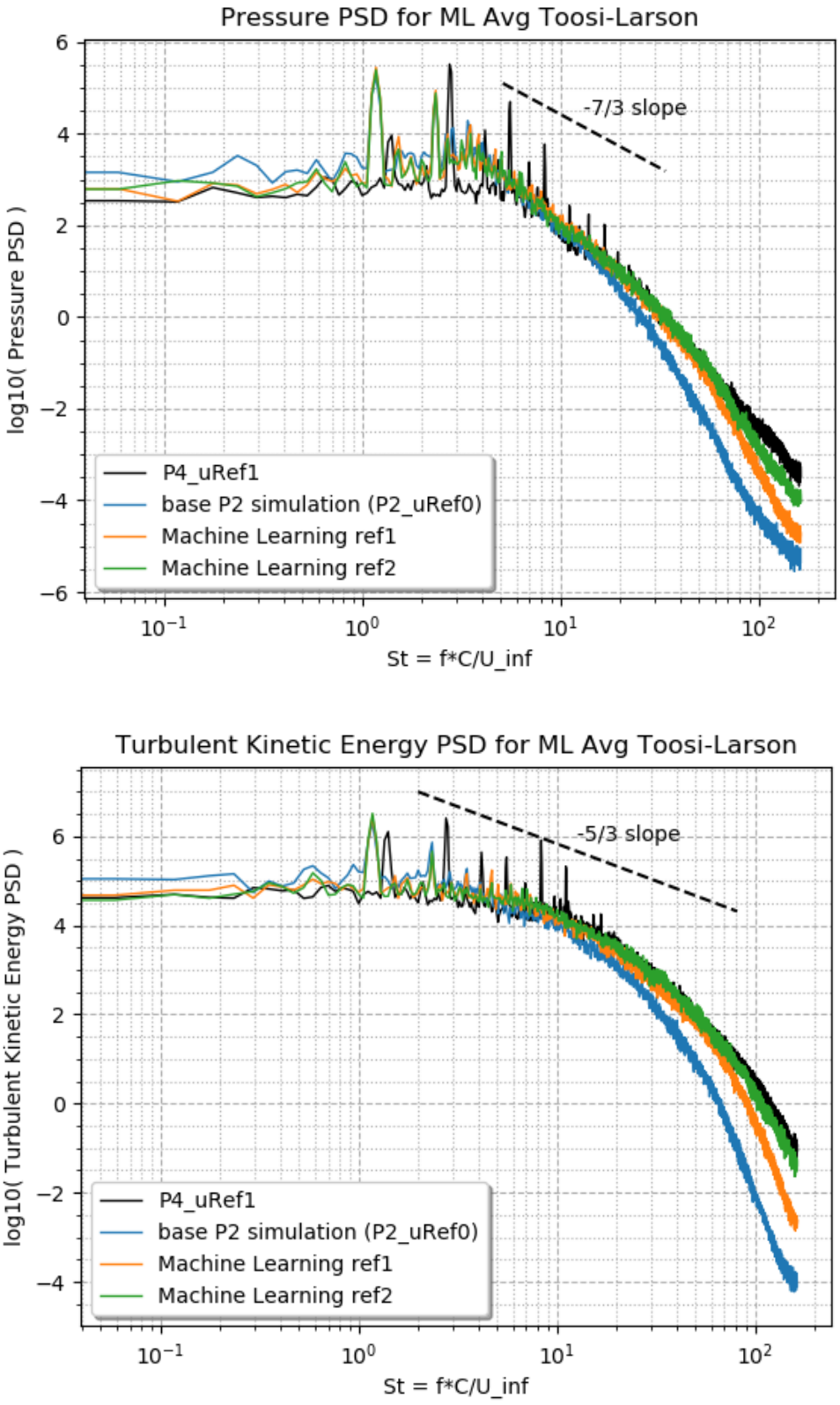



Figure 5.9: Progression of the energy spectra for T106a from the ANN.

Figure 5.10 is the final plot sequence, showing the VV component of the Reynolds stress tensor. The first images are the refinement sequence from the ANN. The bottom two are the "P4_uref1" truth reference (left) and the refinement #2 solution from ${}^{\text{T.L.err}}\mathcal{E}$. These plots show that the ANN has majorly sharpened the Reynolds stress image, but it has not quite gotten the correct distribution shape. ${}^{\text{T.L.err}}\mathcal{E}$ doesn't have the correct shape either, but it is closer to the truth reference than the ANN.

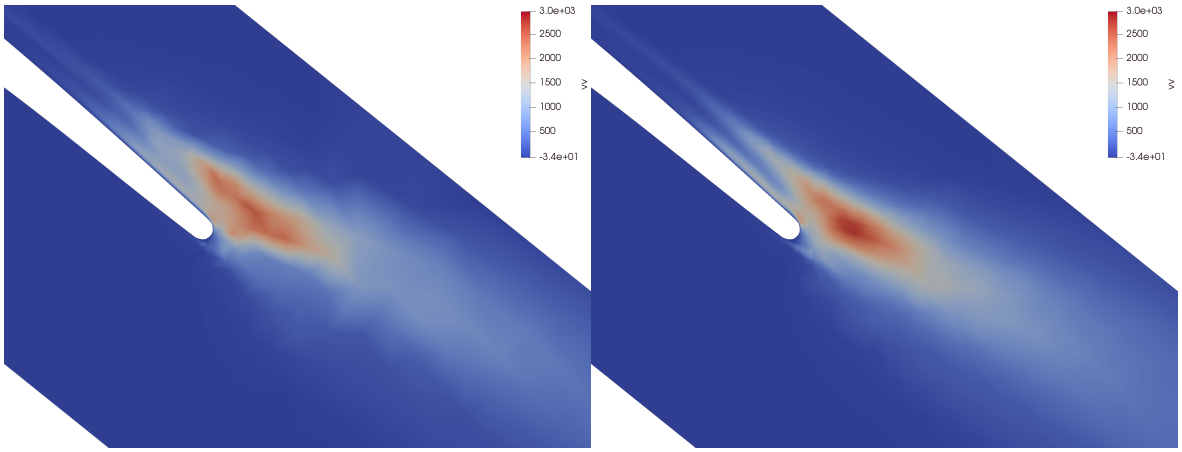
### 5.2.3 Assessment

As an error estimator, the ANN is effective. It is not the best error estimator out of those tested. The ANN was created and trained as an experiment in machine learning to determine the viability of training an ANN as an error estimator.

### 5.2.4 Recommendations for Future Machine Learning Study

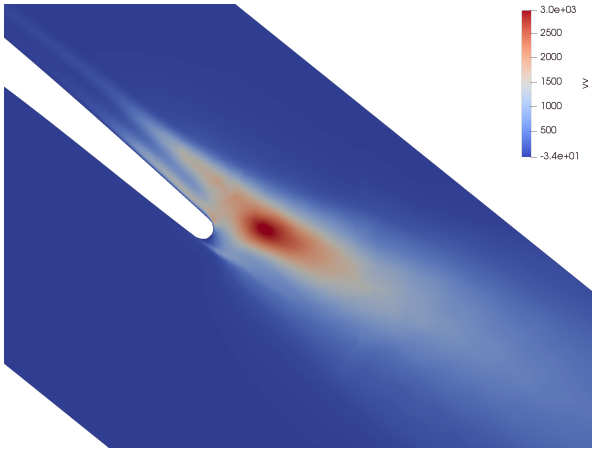
There are several ways to build upon and improve the results. The first is to train the network to mimic a more capable error estimator formula. The runtime flow field data provided to the ANN could be broadened. The second way to boost performance is to increase the amount and variety of the training data. Generating that training data will be a slow process if CFD simulations are used to produce it, so a better source for training data may be the error estimator formula directly, via Monte Carlo.

One of the shortcomings of the current ANN is that it is limited to polynomial order $\mathcal{P}_2$. That limitation exists because the network was built to consume per-cell flow field data of a specific size and sequence. Unfortunately, the amount and the grouping of the per-cell data changes with the polynomial order, shattering the ANN. So, one of the longer term items for a future investigator to tackle is to design an ANN that can operate independently of the polynomial order.

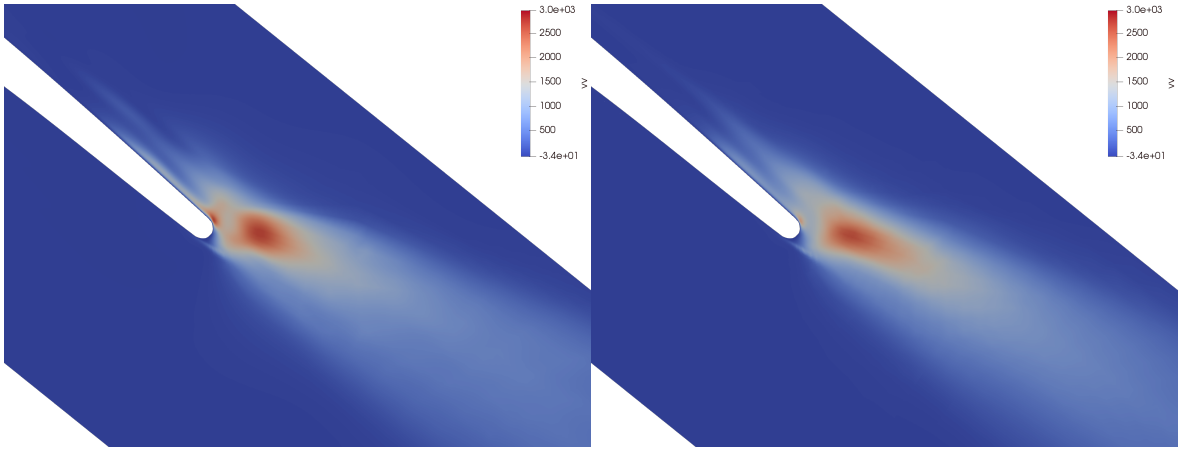



(a) base mesh

(b) $\overline{\text{T.L.err}}\mathcal{E}$ refinement #1




(c) $\overline{\text{T.L.err}}\mathcal{E}$ refinement #2




(d) "P4_uref1"

(e) $\overline{\text{T.L.err}}\mathcal{E}$ refinement #2

Figure 5.10: The first three sub-images show the progression of the VV component of Reynolds stress from error indicator $\overline{\text{T.L.err}}\mathcal{E}$ (computed by the ANN) upon simulation T106a. The two sub-images at the bottom, for comparison, are from the "P4_uref1" simulation (left) and refinement #2 from $\overline{\text{T.L.err}}\mathcal{E}$.

# Chapter 6

# Conclusions and Future Work

## 6.1 Summary

CFD error estimation methods were judged on driving mesh adaptation. Four specific error estimators were introduced, formulated especially for use with the discontinuous high-order FR/CPR numerical scheme, targeted for simulation of unsteady turbulence via LES. Those error estimators are:

- $^{\mathbf{smth}}\mathcal{E}$, founded upon a shock-capturing gradient algorithm
- $^{\mathbf{T.L.err}}\mathcal{E}$, founded upon an estimate of small-scale turbulent kinetic energy
- $^{\mathbf{unStd}}\mathcal{E}$, founded upon a subtractive difference of the unsteady flow residual
- $\overline{^{\mathbf{T.L.err}}\mathcal{E}}$, a less computationally burdensome variant of $^{\mathbf{T.L.err}}\mathcal{E}$.

Formulas were derived for the error estimators, and their characteristic behaviors elucidated by both theoretical derivation and numerical testing. The numerical testing was supported by developed software that performs error-guided H-adaptation of 3D unstructured hexahedral meshes. The software has two parts: 1) a stand alone mesh adapter and 2) a flow field error estimation module. The error estimation module implements the error estimators, incorporating their calculations into a highly parallelized FR/CPR flow solver program. The error estimation module & flow solver run simultaneously.

The performance of both the software and the error estimators were proven upon two classic benchmark LES challenge problems, featuring transitional turbulence over the T106 low pressure

turbine blade. Upon the T106c and T106a simulations, the error estimators all demonstrated an ability to locate the turbulent parts of the flow field and boost the resolution of the turbulence. Under their influence, unresolved vortex structure became visible; force coefficients moved toward truth, Reynolds stresses sharpened, and the turbulent energy spectra came into view. The error indicators were not equally quick in converging the simulations to truth. Indicator ${}^{\mathbf{smth}}\mathcal{E}$ was consistently the slowest while ${}^{\mathbf{T.L.err}}\mathcal{E}$ and ${}^{\mathbf{unStd}}\mathcal{E}$ were in close competition for fastest.

It was also found that each of error indicators is sensitive to an independent characteristic of the flow field, so depending on which characteristic the CFD analyst wishes to emphasize, he or she should choose the indicators according to the following list:

- Error indicator ${}^{\mathbf{smth}}\mathcal{E}$ picks out un-smooth regions of the flow, specifically areas with a high degree of spacial variability. When controlling a mesh adapter, it will primarily enhance the resolution of the boundary layer

- Error indicator ${}^{\mathbf{T.L.err}}\mathcal{E}$ picks out energy dense regions, specifically areas rich in under-resolved vortex motion. When controlling a mesh adapter, it will primarily enhance vortex dominated regions, especially wakes.

- Error indicator ${}^{\mathbf{unStd}}\mathcal{E}$ picks out "active" regions of the flow, specifically areas that are temporally very dynamic. When controlling a mesh adapter, it will distribute resolution both in vortex dominated areas and within surface boundary layers, refining in similar areas as ${}^{\mathbf{T.L.err}}\mathcal{E}$ but not as far from the surface, not as far downstream, and not as wide an area of coverage.

If the CFD analyst wishes to boost simulation spatial resolution as fast as possible, he is advised to choose either ${}^{\mathbf{T.L.err}}\mathcal{E}$ or ${}^{\mathbf{unStd}}\mathcal{E}$. ${}^{\mathbf{unStd}}\mathcal{E}$ is easier to implement in the context of FR/CPR, but ${}^{\mathbf{T.L.err}}\mathcal{E}$ is slightly more performant as a driver of H-adaptation.

## 6.2 Future Research

This research involved the development of a toolset that has opened a considerably wide array of potential follow-up investigations. The first spin-off project, a machine learning endeavor to train an artificial neural network as an error estimator, followed quickly the creation of the adaptation system and became a joint project, covered in this dissertation. Several specific suggestions for future research are listed below, organized by their relation to the error estimators or the adaptation machinery.

### 6.2.1 Future Research Regarding the Error Estimators

1. **Subcomponent Tuning for ${}^{\mathbf{unStd}}\mathcal{E}$ and ${}^{\mathbf{smth}}\mathcal{E}$**

   The formula for ${}^{\mathbf{unStd}}\mathcal{E}$, Equation (2.12), includes a weighted average of subcomponent variables, one per flow field variable. It was observed in testing that these subcomponents can possess an order of magnitude size difference, indicating that the simple average might not be the best way to combine them. It is suggested that a future research project explore alternatives. For instance, it may be beneficial to apply exponential scaling or dissimilar weighting to the average.

   For error estimator ${}^{\mathbf{smth}}\mathcal{E}$, it might be beneficial to alter the time-averaging applied at the final step of its formula, Equation (2.15), replacing it with a maximum operator. In other words, the cell flagging could be based on the maximum error value witnessed over the simulation's time history. The delivered code already has the ability to perform the adaptation in this way, but the effect is not completely tested. Early results suggest that this change to the formula may impart to ${}^{\mathbf{smth}}\mathcal{E}$ the ability to find some flow field features that ${}^{\mathbf{unStd}}\mathcal{E}$ and ${}^{\mathbf{T.L.err}}\mathcal{E}$ cannot.

2. **A Multi-Faceted Error Estimator**

   Since each of the error estimators is sensitive to an independent dynamic of the flow field and

all of them are related to the flow field's resolution, it is worth exploring to see if the error estimators are combinable to yield a super-estimator. The super-estimator might gain benefit by running the sub-indicators cyclically or by operating them as a voting ensemble. The super-estimator could also be augmented with useful information about the flow field. For example, a physics & geometry constraint could be applied to near-wall cell size, ensuring the boundary layer cells will have enough isotropy to support near-wall vortices.

3. **Add a Stopping Criteria**

   To prevent an LES simulation from turning into a DNS simulation, it is necessary to halt the mesh refinement before spatial resolution becomes too high. A useful future research project would be to add a refinement-stopping criteria to the error estimation system. A good starting place might be the physics-based LES quality indicator from (Celik et al., 2005), covered in the literature review.

### 6.2.2 Future Research Regarding the Adaptation Machinery

1. **Add support for more cell shapes.**

   The CFD flow solver supports all four of the cell shapes that can appear in unstructured meshes: hexahedra, tetrahedra, pyramid, and triangular prism. However, the mesh adapter fully supports only one cell shape: hexahedra. The mesh adapter could be expanded to provide support for all of the cell shapes so that error-adapted meshes can be created for all simulations. The challenge of adding these cells is that the researcher has to devise a way to isotropically split pyramids and triangular prisms into equal-sized, equal-shaped parts. Tetrahedral splitting could be accomplished by placing a node at its centroid and running a line to each vertex.

2. **Add hanging node support to all operating modes of the FR/CPR flow solver**

   The FR/CPR flow solver has the ability to run upon a GPU cluster, but the GPU executable cannot handle hanging node meshes. Hanging node support could be implemented, yielding

a combined speedup from both the GPU execution and the adaptive meshing.

3. **Add decimation ability.**

   The developed adaptation software can only add resolution. It cannot optimize resolution if there is already too much resolution. A future research project could expand the software to dynamically combine cells, subtracting resolution that is not providing benefit to the simulation.

# References


Abbà, A., Recanati, A., Tugnoli, M., & Bonaventura, L. (2020). Dynamical p-adaptivity for LES of compressible flows in a high order DG framework. *Journal of Computational Physics*, 420, 109720.

Ait-Ali-Yahia, D., Baruzzi, G., Habashi, W. G., Fortin, M., Dompierre, J., & Vallet, M.-G. (2002). Anisotropic mesh adaptation: Towards user-independent, mesh-independent and solver-independent CFD. part II: Structured grids. *International Journal for Numerical Methods in Fluids*, 39(8), 657–673.

Alauzet, F. & Loseille, A. (2016). A decade of progress on anisotropic mesh adaptation for computational fluid dynamics. *Computer-Aided Design*, 72, 13–39.

Alhawwary, M. & Wang, Z. J. (2019). On the mesh resolution of industrial LES based on the DNS of flow over the T106C turbine. *Advances in Aerodynamics*, 1(1), 1–18.

Antepara, O., Lehmkuhl, O., Borrell, R., Chiva, J., & A.Oliva (2015). Parallel adaptive mesh refinement for large-eddy simulations of turbulent flows. *Computer & Fluids*, 110, 48–61.

Bassi, F., Crivellini, A., Di Pietro, D. A., & Rebay, S. (2006). A high-order discontinuous Galerkin solver for 3D aerodynamic turbulent flows. In *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics* Delft, The Netherlands: Delft University of Technology.

Bassi, F. & Rebay, S. (2000). A high order discontinuous Galerkin method for compressible turbulent flows. In *Discontinuous Galerkin Methods* (pp. 77–88). Springer.

Benard, P., Balarac, G., Moureau, V., Dobrzynski, C., Lartigue, G., & d'Angelo, Y. (2016). Mesh

adaptation for large-eddy simulations in complex geometries. *International Journal for Numerical Methods in Fluids*, 81(12), 719–740.

Bhaskaran, R., Jia, F., Laskowski, G. M., Wang, Z., & Paliath, U. (2017). Towards high-order large eddy simulation of aero-thermal flows for turbomachinery applications. In *Proceedings of the ASME Turbo Expo 2017: Turbine Technical Conference and Exposition*, volume 2B: American Society of Mechanical Engineers.

Blonigan, P. J., Wang, Q., Nielsen, E. J., & Diskin, B. (2018). Least-squares shadowing sensitivity analysis of chaotic flow around a two-dimensional airfoil. *AIAA Journal*, 56(2), 658–672.

Carson, H. A., Huang, A. C., Galbraith, M. C., Allmaras, S. R., & Darmofal, D. L. (2020). Mesh optimization via error sampling and synthesis: An update. In *AIAA SciTech 2020 Forum* (pp. 0087).

Celik, I., Cehreli, Z., & Yavuz, I. (2005). Index of resolution quality for large eddy simulations. *Journal of Fluids Engineering*, 127(5), 949–958.

Diaz, M., Hecht, F., Mohammadi, B., & Pironneau, O. (1997). Anisotropic unstructured mesh adaptation for flows simulations. *International Journal for Numerical Methods in Fluids*, 25, 475–491.

Dompierre, J., Vallet, M.-G., Bourgault, Y., Fortin, M., & Habashi, W. G. (2002). Anisotropic mesh adaptation: Towards user-independent, mesh-independent and solver-independent CFD. part III: Unstructured meshes. *International Journal for Numerical Methods in Fluids*, 39(8), 675–702.

Donath, M. (2020). Standard error of the (estimated) mean for time-series data. `https://ljmartin.github.io/technical-notes/stats/estimators-autocorrelated/`. [Online; accessed 16-June-2022].

Fidkowski, K. (2017). Output-based space-time mesh optimization for unsteady flows using continuous-in-time adjoints. *Journal of Computational Physics*, 341(15), 258–277.

Fidkowski, K. & Darmofal, D. (2011). Output-based error estimation and mesh adaptation in computational fluid dynamics: Overview and recent results. *AIAA Journal*, 49(4), 673–694.

Fidkowski, K. J. & Chen, G. (2021). Metric-based, goal-oriented mesh adaptation using machine learning. *Journal of Computational Physics*, 426, 109957.

Foti, D., Giorno, S., & Duraisamy, K. (2020). An adaptive mesh refinement approach based on optimal sparse sensing. *Theoretical and Computational Fluid Dynamics*, 34(4), 457–482.

Frey, P. & Alauzet, F. (2005). Anisotropic mesh adaptation for CFD computations. *Comput. Methods Appl. Mech. Engrg.*, 194, 5068–5082.

Gao, H. & Wang, Z. J. (2011). A residual-based procedure for hp-adaptation on 2D hybrid meshes. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition* (pp. 492).

Ha, D., Dai, A., & Le, Q. V. (2016). Hypernetworks. *arXiv preprint arXiv:1609.09106*.

Habashi, W., Dompierre, J., Bourgault, Y., Ait-Ali, D., Fortin, M., & Vallet, M.-G. (2000). Anisotropic mesh adaptation: Towards user-independent, mesh-independent and solver-independent CFD. part I: General principles. *International Journal for Numerical Methods in Fluids*, 32, 725 – 744.

Harlow, F. H., Evans, M., & Richtmyer, R. D. (1955). *A machine calculation method for hydrodynamic problems*. Los Alamos Scientific Laboratory of the University of California.

Hartmann, R., Held, J., & Leicht, T. (2011). Adjoint-based error estimation and adaptive mesh refinement for the RANS and k–$\omega$ turbulence model equations. *Journal of Computational Physics*, 230(11), 4268–4284.

Hecht, F. (1998). BAMG: bidimensional anisotropic mesh generator. *User Guide. INRIA, Rocquencourt*, 17.

Hillewaert, K. & JS., C. (2016). As2 - spanwise periodic DNS/LES of transitional flow in T106 LP turbine cascades — HiOCFD4, 4th international workshop on high-order CFD methods. https://how4.cenaero.be/content/as2-spanwise-periodic-dnsles-transitional-turbine-cascades. [Online; accessed 14-April-2022].

Hillewaert, K. & JS., C. (2018). As2 - spanwise periodic DNS/LES of transitional flow in T106 LP turbine cascades — HiOCFD5, 5th international workshop on high-order CFD methods. https://how5.cenaero.be/content/cs2-t106-lpt-cascades. [Online; accessed 1-May-2022].

Hirsch, C. (2007). *Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics*, chapter 1-2, (pp. 35,70–71). Elsevier.

Huynh, H., Wang, Z. J., & Vincent, P. E. (2014). High-order methods for computational fluid dynamics: A brief review of compact differential formulations on unstructured grids. *Computers & Fluids*, 98, 209–220.

Huynh, H. T. (2007). A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods. In *18th AIAA Computational Fluid Dynamics Conference* (pp. 4079).

Ims, J., Duan, Z., & Wang, Z. J. (2015). meshCurve: An automated low-order to high-order mesh generator. In *22nd AIAA Computational Fluid Dynamics Conference* (pp. 2293).

Ims, J. & Wang, Z. (2019). Automated low-order to high-order mesh conversion. *Engineering with Computers*, 35(1), 323–335.

Ims, J. & Wang, Z. J. (2022). A comparison of three error indicators for adaptive high-order large eddy simulation. In *AIAA SciTech 2022 Forum* (pp. 1201).

Ims, J. & Wang, Z. J. (under review). A comparison of three error indicators for adaptive high-order large eddy simulation. *(submitted to) Journal of Computational Physics*.

Jameson, A., Schmidt, W., & Turkel, E. (1981). Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes. In *14th Fluid and Plasma Dynamics Conference* (pp. 1259).

Jia, F., Ims, J., Wang, Z., Kopriva, J., & Laskowski, G. M. (2019). Evaluation of second-and high-order solvers in wall-resolved large-eddy simulation. *AIAA Journal*, 57(4), 1636–1648.

Jia, F., Ims, J., Wang, Z. J., Kopriva, J., & Laskowski, G. M. (2018). An evaluation of a commercial and a high order FR/CPR flow solvers for industrial large eddy simulation. In *2018 AIAA Aerospace Sciences Meeting* (pp. 0827).

Kopriva, D. A. (1996). A conservative staggered-grid Chebyshev multidomain method for compressible flows. II. a semi-structured method. *Journal of Computational Physics*, 128(2), 475–488.

Li, R., Tang, T., & Zhang, P. (2001). Moving mesh methods in multiple dimensions based on harmonic maps. *Journal of Computational Physics*, 170(2), 562–588.

Li, Y. & Wang, Z. J. (2017). A convergent and accuracy preserving limiter for the FR/CPR method. In *55th AIAA Aerospace Sciences Meeting* (pp. 0756).

Lin, J. & Dominik, C. (1995). Optimization of an advanced design three-element airfoil at high Reynolds numbers. In *13th Applied Aerodynamics Conference* (pp. 1858).

Morton, K. W. & Mayers, D. F. (2005). *Numerical Solution of Partial Differential Equations: An Introduction*. Cambridge University Press, second edition.

Naddei, F., de la Llave Plata, M., Couaillier, V., & Coquel, F. (2019). A comparison of refinement indicators for p-adaptive simulations of steady and unsteady flows using discontinuous Galerkin methods. *Journal of Computational Physics*, 376, 508–533.

OpenMP Architecture Review Board (2015). *OpenMP Application Program Interface Version 4.5*.

Park, M. A. (2004). Adjoint-based, three-dimensional error prediction and grid adaptation. *AIAA Journal*, 42(9), 1854–1862.

Persson, P.-O. & Peraire, J. (2006). Sub-cell shock capturing for discontinuous Galerkin methods. In *44th AIAA Aerospace Sciences Meeting and Exhibit* (pp. 112).

Phommachanh, J. (2021). Mesh adaptation using machine learning. Master's thesis, University of Kansas.

Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta numerica*, 8, 143–195.

Qin, Y., Shih, T., Keller, P., Sun, R., Hernandez, E., Perng, C., Trigui, N., Han, Z., Shen, F., & Shieh, T. (2004). Estimating grid-induced errors in CFD by discrete-error-transport equations. In *42nd AIAA Aerospace Sciences Meeting and Exhibit* (pp. 656).

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 686–707.

Richardson, L. F. (1922). *Weather prediction by numerical process*. University Press.

Roe, P. L. (1981). Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2), 357–372.

Roy, C. J. (2009). Strategies for driving mesh adaptation in CFD. In *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1), 61–80.

Shampine, L. F., Allen, R. C., & Pruess, S. (1997). *Fundamentals of numerical computing*, volume 1. Wiley New York.

Shi, L., Zhou, C., & Wang, Z. J. (2016). Adaptive RANS solution with the high-order correction procedure via reconstruction method. In *54th AIAA Aerospace Sciences Meeting* (pp. 1826).

Shu, C.-W. (1988). Total-variation-diminishing time discretizations. *SIAM Journal on Scientific and Statistical Computing*, 9(6), 1073–1084.

Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., & Mavriplis, D. (2014). CFD vision 2030 study: A path to revolutionary computational aerosciences.

Tange, O. (2011). GNU Parallel - the command-line power tool. *The USENIX Magazine*, 36(1), 42–47.

Tingfan, W., Xuejun, L., Wei, A., Huang, Z., & Hongqiang, L. (2022). A mesh optimization method using machine learning technique and variational mesh adaptation. *Chinese Journal of Aeronautics*, 35(3), 27–41.

Toosi, S. & Larsson, J. (2017a). Anisotropic grid-adaptation in large eddy simulations. *Computers & Fluids*, 156, 146–161.

Toosi, S. & Larsson, J. (2017b). Anisotropic grid-adaptation in large eddy simulations of wall-bounded and free shear flows. In *55th AIAA Aerospace Sciences Meeting* (pp. 0978).

Vatsa, V., Carpenter, M., & Lockard, D. (2010). Re-evaluation of an optimized second order backward difference (BDF2OPT) scheme for unsteady flow applications. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition* (pp. 122).

Venditti, D. A. & Darmofal, D. L. (2002). Grid adaptation for functional outputs: Application to two-dimensional inviscid flows. *Journal of Computational Physics*, 176(1), 40–69.

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

Wang, L., Gobbert, M., & Yu, M. (2020). A dynamically load-balanced parallel p-adaptive implicit high-order flux reconstruction method for under-resolved turbulence simulation. *Journal of Computational Physics*, 417, 109581.

Wang, L. & Mavriplis, D. (2009). Adjoint-based hp-adaptive discontinuous Galerkin methods for the 2D compressible Euler equations. *Journal of Computational Physics*, 228(20), 7643–7661.

Wang, Q., Hu, R., & Blonigan, P. (2014). Least squares shadowing sensitivity analysis of chaotic limit cycle oscillations. *Journal of Computational Physics*, 267, 210–224.

Wang, Z. J. & Huynh, H. (2016). A review of flux reconstruction or correction procedure via reconstruction method for the Navier-Stokes equations. *Mechanical Engineering Reviews*, 3(1), 15–00475.

Wang, Z. J., Li, Y., Jia, F., Laskowski, G., Kopriva, J., Paliath, U., & Bhaskaran, R. (2017). Towards industrial large eddy simulation using the FR/CPR method. *Computers & Fluids*, 156, 579–589.

Yang, J., Dzanic, T., Petersen, B., Kudo, J., Mittal, K., Tomov, V., Camier, J.-S., Zhao, T., Zha, H., Kolev, T., et al. (2021). Reinforcement learning for adaptive mesh refinement. *arXiv e-prints*, (pp. arXiv–2103).

Yano, M. & Darmofal, D. L. (2012). An optimization-based framework for anisotropic simplex mesh adaptation. *Journal of Computational Physics*, 231(22), 7626–7649.

Zhou, C., Shi, L., & Wang, Z. J. (2017). Adaptive high-order discretization of the Reynolds-averaged Navier-Stokes equations. *Computers & Fluids*, 159, 137–155.

# Appendix A

# Publications

Ims, J., Duan, Z., & Wang, Z. J. (2015). meshCurve: An automated low-order to high-order mesh generator. In *22nd AIAA Computational Fluid Dynamics Conference* (pp. 2293).

Ims, J. & Wang, Z. (2019). Automated low-order to high-order mesh conversion. *Engineering with Computers*, 35(1), 323–335.

Ims, J. & Wang, Z. J. (2022). A comparison of three error indicators for adaptive high-order large eddy simulation. In *AIAA SciTech 2022 Forum* (pp. 1201).

Ims, J. & Wang, Z. J. (Under Review). A comparison of three error indicators for adaptive high-order large eddy simulation. *(submitted to) Journal of Computational Physics*.

Jia, F., Ims, J., Wang, Z., Kopriva, J., & Laskowski, G. M. (2019). Evaluation of second-and high-order solvers in wall-resolved large-eddy simulation. *AIAA Journal*, 57(4), 1636–1648.

Jia, F., Ims, J., Wang, Z. J., Kopriva, J., & Laskowski, G. M. (2018). An evaluation of a commercial and a high order FR/CPR flow solvers for industrial large eddy simulation. In *2018 AIAA Aerospace Sciences Meeting* (pp. 0827).