Discretization Error Estimation and Exact Solution Generation Using the 2D Method of Nearby Problems

Matthew J. Kurzen

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Aerospace Engineering

Christopher J. Roy, Chair
Leigh S. McCue-Weil
Danesh K. Tafti

February 1, 2010
Blacksburg, VA

KEYWORDS: Method of Nearby Problems, MNP, Discretization Error, Method of Manufactured Solutions, Computational Fluid Dynamics

Abstract

Discretization Error Estimation and Exact Solution Generation Using the 2D Method of Nearby Problems

Matthew J. Kurzen

This work examines the Method of Nearby Problems as a way to generate analytical exact solutions to problems governed by partial differential equations (PDEs). The method involves generating a numerical solution to the original problem of interest, curve fitting the solution, and generating source terms by operating the governing PDEs upon the curve fit. Adding these source terms to the right-hand-side of the governing PDEs defines the nearby problem.

In addition to its use for generating exact solutions the MNP can be extended for use as an error estimator. The nearby problem can be solved numerically on the same grid as the original problem. The nearby problem discretization error is calculated as the difference between its numerical solution and exact solution (curve fit). This is an estimate of the discretization error in the original problem of interest.

The accuracy of the curve fits is quite important to this work. A method of curve fitting that takes local least squares fits and combines them together with weighting functions is used. This results in a piecewise fit with continuity at interface boundaries. A one-dimensional Burgers' equation case shows this to be a better approach then global curve fits.

Six two-dimensional cases are investigated including solutions to the time-varying Burgers' equation and to the 2D steady Euler equations. The results show that the Method of Nearby Problems can be used to create realistic, analytical exact solutions to problems governed by PDEs. The resulting discretization error estimates are also shown to be reasonable for several cases examined.

Acknowledgements

The author would like to acknowledge Dr. Roy, Dr. McCue, and Dr. Tafti for their role in serving on the thesis advisory committee. In particular the author would like to thank his primary advisor, Dr. Roy, for his guidance over the past several years and support for the Method of Nearby Problems research. He provided insightful advice to problems ranging from the conceptual approach to practical coding issues.

The author would also like to acknowledge several people who provided computer codes which were used in the Method of Nearby Problems research. Dr. Sinclair of Auburn University provided a 2D weighted least squares fitting code. Tyrone Phillips, a fellow graduate student at Virginia Tech, provided the Euler equation code and numerical solutions for several cases. He was also instrumental in working through input/output interface issues between the numerical solver and the MNP code. Santhip Kanholy, another fellow graduate student at Virginia Tech, provided the 1D Burgers' equation solver that was used early on in the research.

In addition, the author is thankful for the financial support provided through Dr. Roy and originating from Sandia National Laboratories. This work was supported by a grant from Sandia National Laboratories through a US Department of Energy Presidential Early Career Award for Scientists and Engineers with Dr. Ryan Bond serving as technical monitor. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract no. DE-AC04-94AL85000.

Table of Contents

Table of Figures

Table of Tables

1 Motivation

1.1 Introduction to Computational Fluid Dynamics

Computational fluid dynamics (CFD) is a procedure for solving fluid dynamics problems governed by the Euler or Navier-Stokes equations. The usual CFD approach begins with covering the physical domain of the problem with a grid of interconnected nodes, also referred to as a mesh. The governing PDEs are discretized by replacing the continuous expression (e.g. $\partial u/\partial x$) with discrete expressions (e.g. $(u_{i+1} - u_{i-1})/(2\Delta x)$, where the i subscripts denote the grid location index and Δx is the grid spacing). The discretized equations can now be applied to the grid, to update the solution values at the grid nodes. When applied iteratively and given certain stability criteria, the equations will converge to the solution.

1.2 Nature of Partial Differential Equations (PDEs)

Differential equations are equations which contain derivatives terms. When a dependent variable is a function of multiple independent variables, a partial derivative describes the change in the independent variable with respect to one of the independent variables. This is better explained with an example, such as the time-dependent Burgers' equation shown in Equation 1.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{1}$$

where u is the dependent velocity variable, while x and t are the independent space and time variables, respectively. The kinematic viscosity parameter ν is a constant. The velocity u depends on both x and t. The first term is the partial derivative of velocity with respect to time. The second term contains a partial derivative of velocity with respect to space and represents convection. Note that the second term is also nonlinear, as the partial derivative is multiplied by the velocity. The final term contains a second partial derivative and represents diffusion.

Generally there are a limited number of exact analytical solutions to PDEs. As Burgers' equation is relatively simple, there are several exact solutions, such as Equation 2, as given Benton and Platzman[1].

$$u'(x',t') = -\frac{2\sinh(x')}{\cosh(x') + e^{-t'}} \tag{2}$$

where the prime variables denote that they are nondimensional. This exact solution models the coalescence of two viscous shocks.

In order to solve Burgers' equation numerically through computational simulation, the equation needs to be discretized. A simple example of this is shown in Equation 3, where the continuous derivatives have been replaced with discrete finite difference representations

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}+u_i^n\frac{u_{i+1}^n-u_{i-1}^n}{2\Delta x}=\nu\frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{\Delta x^2} \tag{3}$$

where the i subscript refers to the discrete spatial location (grid point) and the n superscript refers to the discrete temporal location (time value). The time step and spatial step (grid spacing) are Δt and Δx, respectively. By rearranging the equation, we arrive at Equation 4, where the value on the left-hand-side is the velocity value for a given grid location, at a new time value. In this manner, the equation is solved by marching through the spatial points and progressing forward in time. Note that there are many different schemes for discretizing the partial derivatives and the procedure shown here is not exclusive.

$$u_i^{n+1}=u_i^n-(\Delta t)\left(u_i^n\frac{u_{i+1}^n-u_{i-1}^n}{2\Delta x}-\nu\frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{\Delta x^2}\right) \tag{4}$$

1.3 Numerical Error

Numerical error is inherent in CFD solutions and includes round-off error, iterative error, and discretization error (Oberkampf and Roy[2]). The presence of these errors is due to the fundamental nature of solving discretized equations on a computer system. Round-off error arises from how numerical values are stored in memory. This is due to the inability for a computer with finite binary space from perfectly representing nonfinite numbers (irrational or repeating) or finite numbers with more precision required than storage space allows. Equation 5 gives an example of round-off error in a single precision system. The exact value of $1/3$ cannot be stored, so the approximate value is stored instead.

$$\frac{1}{3}=0.3333333 \quad \Rightarrow \quad \frac{1}{3}*3=0.9999999 \tag{5}$$

Isolated round-off errors are usually not a problem, but the accumulation of round-off errors in a long or iterative calculation can be problematic.

Iterative error comes from a solution that is not completely iteratively converged. Iterative convergence is monitored through the iterative residuals, which result from operating the discretized equations upon the current iteration of the solution. If the solution exactly solves the discretized equations, then the residual will be zero. In practice, the solution is usually considered converged if the normalized residuals decrease considerably, such as falling 12 orders of magnitude. Figure 1 shows an example of an iterative convergence history. The iterative residuals of the mass, x-momentum, y-momentum, and energy equations are monitored for the Euler equations.

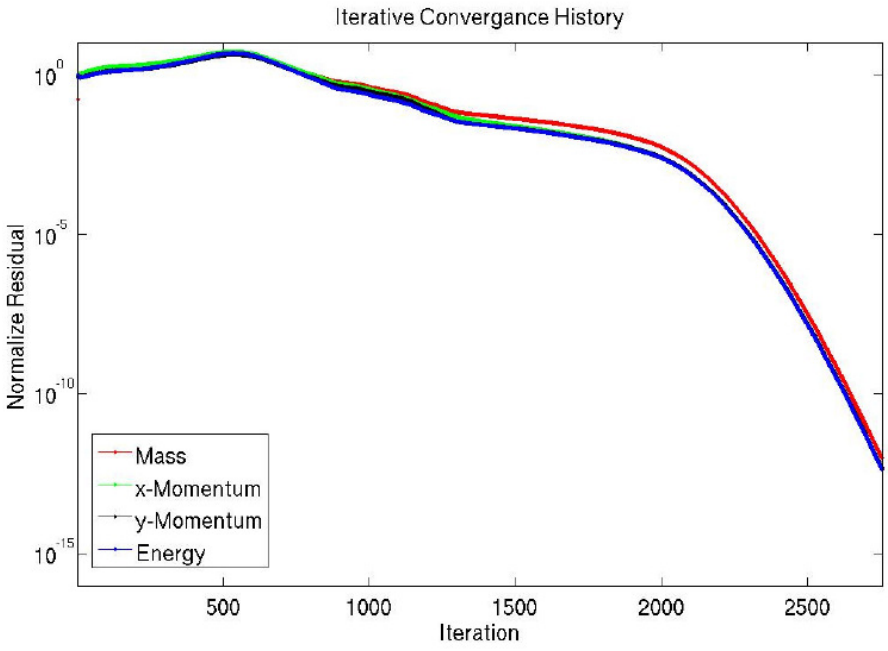
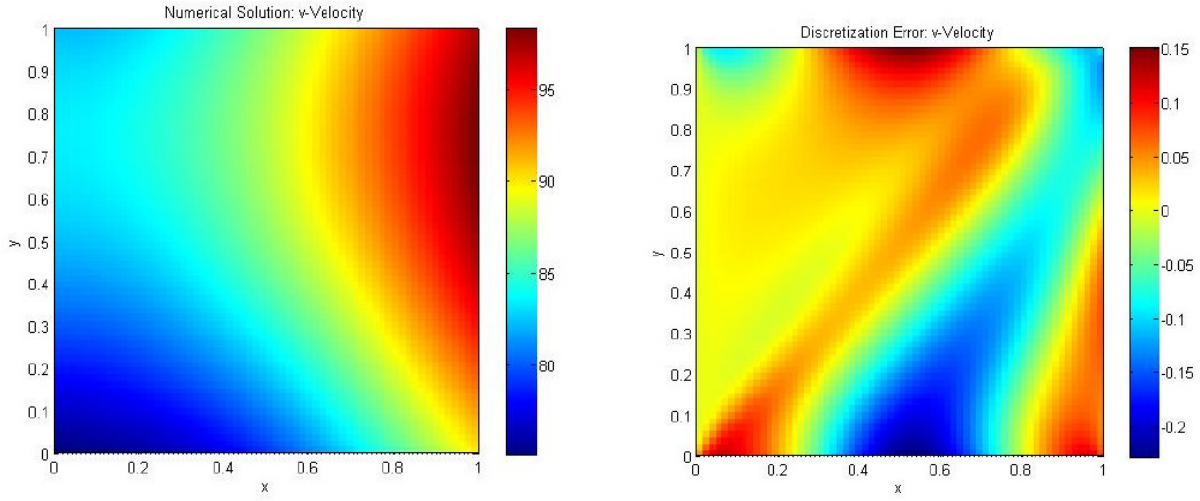


Figure 1: Iterative Convergence Example from Euler Equation Solver

Discretization error is the error due to discretizing the governing PDEs. It is defined as the difference between the exact solution to the discretized equations and the exact solution to the PDEs. This type of numerical error is often difficult to quantify, as the exact solution to the PDEs is not usually known for arbitrary problems. Figure 2 shows the v-velocity numerical solution to the Euler equations and the related discretization error. This particular case is for a manufactured solution which is a procedure described in Section 2.2. The left wall and bottom

wall are subsonic inlets, while the right wall and top wall are subsonic outlets. Therefore, the flow travels diagonally across the domain.



a) **b)**

Figure 2: Euler Equation v-Velocity: a) Numerical Solution, b) Discretization Error

1.4 Approaches for Estimating Discretization Error

Due to the previously mentioned difficulties with calculating discretization error, it is often necessary to estimate this error. Several approaches for discretization error estimation include: Richardson extrapolation, discretization error transport equations, and defect correction. In addition, recovery and residual methods exist for estimating discretization error in the finite element method (FEM).

Richardson extrapolation uses solutions on two grids, with one grid uniformly refined from the other grid. It can be used to estimate the discretization error in the fine grid or to provide a higher-order accurate estimate of the solution. The generalized Richardson extrapolation expression is given in Equation 6

$$\bar{f} = f_h + \frac{f_h - f_{rh}}{r^p - 1} \tag{6}$$

where $\bar{f}$ is the estimated exact solution, h is the fine grid spacing parameter, r is the grid refinement factor, and p is the formal order of accuracy. It follows that f_h is the fine grid solution and f_{rh} is the coarse grid solution. The formal order of accuracy is dependent on the

discretization expression chosen and is the order of the first term truncated from the Taylor series used to derive the discretization expression. If there are multiple partial derivatives and different order discretization expressions are used, the lowest order expression is the formal order of accuracy.

Richardson extrapolation can also give an estimate of the error in the fine grid solution through Equation 7

$$\bar{\varepsilon}_h = f_h - \bar{f} \tag{7}$$

where $\bar{\varepsilon}_h$ is the estimated discretization error.

The observed order of accuracy can be found if the numerical solution is calculated on three uniformly refined grids, through Equation 8

$$\hat{p} = \frac{\ln\left(\frac{f_3 - f_2}{f_2 - f_1}\right)}{\ln(r)} \tag{8}$$

where $\hat{p}$ is the observed order of accuracy, r is the refinement factor between two consecutive grids, and f_3, f_2, and f_1 are the solutions on the coarse, medium, and fine grids, respectively. If the observed order matches the formal order, then the grids are said to be in the asymptotic region, where the leading truncation term dominates the error. If the grids are not in the asymptotic range then Richardson extrapolation cannot be reliably used.

The error transport equations describe how the discretization error is transported throughout the problem domain. The equations are valid for linear problems. The continuous error transport equation is given in Equation 9

$$L(\varepsilon_h) = -TE_h(T_h) \tag{9}$$

where $L(\cdot)$ is the continuous PDE operator, ε_h is the discretization error, TE_h is the truncation error, and T_h is the discrete solution. The discrete error transport equation is given in Equation 10

$$L_h(\varepsilon_h) = -TE_h(\tilde{T}) \tag{10}$$

where $L_h(\cdot)$ is the discretized PDE operator and $\tilde{T}$ is the exact solution to the PDE. The continuous version of the equation is difficult to solve, as it contains the continuous PDE operator. The discrete version is more useful, but does require the exact solution to the PDE.

However, if we only wish to have an estimate of the discretization error, we can approximate the exact solution by curve fitting the discrete solution on a finer mesh. The equation can then be solved with the right-hand-side acting as a source term to the original discretized PDEs, to obtain an estimate of the discretization error.

There are several discretization error estimators that have been developed primarily by the finite element method (FEM) community, including recovery methods and residual methods. An overview of these is given by Oberkampf and Roy,[2] who make the following observations: Recovery methods involve curve fitting the solution values or slopes and ultimately finding an estimate of the global energy norms. The Zienkiewicz-Zhu (ZZ) error estimator is often used, with either superconvergent patch recovery (SPR) or polynomial preserving recovery (PPR) approaches. The residual method is another procedure for determining the global energy norm. In some cases these recovery and residual methods have been expanded outside of FEM (to finite difference methods (FDM) or finite volume methods (FVM)); however, they are most commonly used with FEM solvers. The adjoint method is related to the residual method but is expressly formulated to be independent of any particular discretization scheme (FEM, FDM, and FVM). The adjoint method is design to estimate error in global, integral quantities (e.g. lift, drag, heat flux through a wall).

Defect correction is another technique for both estimating error and improving the accuracy of the solution. The following framework for defect correction as set forth by Layton, et al.[3]

1. The original problem $L(u)=0$ has an exact solution $\tilde{u}$.
2. A related problem $G(u)=0$ has an exact solution u_0.
3. The defect (residual) is defined as $d_0 = L(u_0)$.
4. The correction (error) is defined as $e_0 = u_1 - u_0$ and can be found through:

5a. A linear update of $G'(u_0)e_0 = d_0$, or:

5b. Indirectly through a nonlinear update of $G(u_1) - G(u_0) = G(u_1) = d_0$.

This method can be applied iteratively by repeating parts 3-5.

1.5 The Method of Nearby Problems

The method of nearby problems (MNP) is a procedure for generating exact analytical solutions to equations that are nearby (slightly modified) to the original problem of interest. In addition to generating exact solutions, the MNP can be used to estimate the discretization error in the original problem. This process is described in detail in Chapter 3. The MNP is closely related to a single iteration of defect correction. In the MNP the related (nearby) problem $G(x)$ is a discretized version of the original, analytical problem $F(x)$. Therefore in order to operate $F(\cdot)$ on the discrete solution of $G(x)=0$, the discrete solution is curve fitted to produce a continuous expression.

The MNP is also related to the method of manufactured solutions (MMS). The MMS is used to generate exact solutions for the purpose of code verification. The discretization error for a MMS problem can be calculated directly (as opposed to being estimated), as discussed below. Code verification is directly tied to discretization error and how that error decreases with grid refinement.

Discretized equations are derived from Taylor series expansions. The formal order of accuracy for a discretized equation is the power to which the first truncated term is raised. The asymptotic range for a grid represents the grid refinement level needed such that the first truncated term is the dominant error term (contributes the most to the total truncation error).

In order to determine if a pair of grids are sufficiently fine to be within the asymptotic region the observed order of accuracy is calculated. When the exact solution to the PDE is known, this is defined in Equation 11

$$\hat{p} = \frac{\ln\left(\left\|\varepsilon_{rh}\right\| / \left\|\varepsilon_{h}\right\|\right)}{\ln(r)} \tag{11}$$

where $\hat{p}$ is the observed order of accuracy, r is the refinement factor between the grids, and $\left\|\varepsilon_{rh}\right\|$ and $\left\|\varepsilon_{h}\right\|$ are norms of the discretization errors on the coarse and fine grids, respectively. The steps to the MMS are:

1. Choose a manufactured solution, typically from sine and cosine terms, as they are infinitely differentiable.
2. Operate the governing PDEs upon the manufactured solution, generating analytical source terms.

3. Add the source terms to the right-hand-side of the original PDEs, to form the modified problem.
4. Generate the numerical solution to the modified problem.
5. Subtract the manufactured solution from the numerical solution, calculating the discretization error.
6. Repeat steps 4-5 for a uniformly refined grid.
7. Calculate the observed order of accuracy.

The MNP can be used to create exact solutions that are more physically realistic than manufactured solutions, as the MNP exact solution is a curve fit of the numerical solution.

2 Background (Literature Review)

2.1 Verification and Validation

Verification and validation are terms related to the process of building confidence in computational simulations. While these terms have separate meanings, the distinct definitions depend on who you ask. As discussed by Oberkampf and Roy,[2] a variety of standards have been published by professional organizations, including: the Society for Computer Simulation (SCS) in 1979, the Institute of Electrical and Electronics Engineers (IEEE) in 1984 and 1991, the Defense Modeling and Simulation Organization (DMSO) in 1994, the American Institute of Aeronautics and Astronautics (AIAA) in 1998, and the American Society of Mechanical Engineers (ASME) in 2006. The AIAA[4] guide presented the following definitions:

> *Verification*: The process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model.
>
> *Validation*: The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model.

Succinct definitions for verification and validation come from Blottner[5]: verification is "solving governing equations right" and validation is "solving right governing equations".

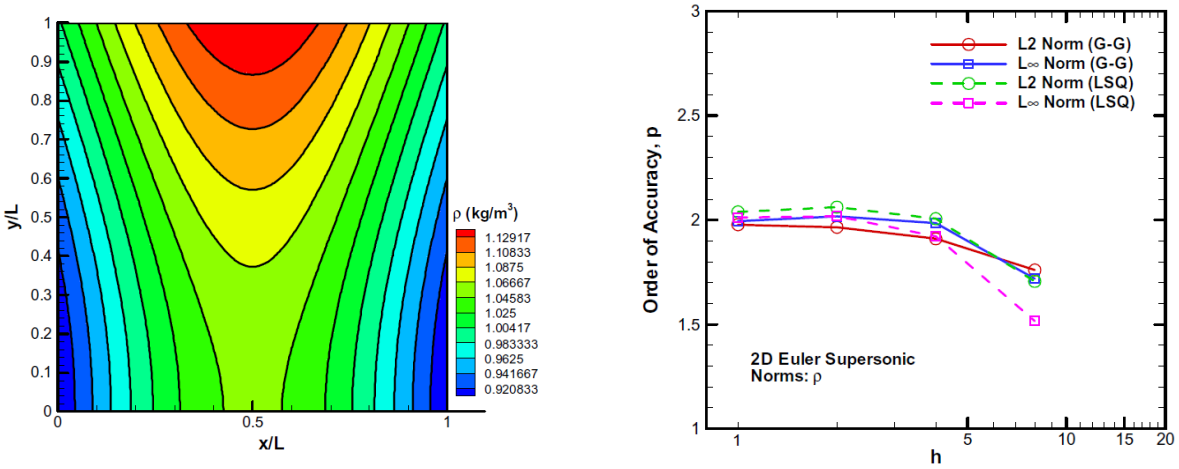
Roache[6] noted that there are two essential elements to verification: the code and the calculation. Code verification usually takes place through order verification, which is the process of determining that the observed order of accuracy approaches the formal order of accuracy as the grid for a computational simulation is refined. In short, the formal order of accuracy is the order of the leading term in the truncation error expression for a discretized equation. The observed order of accuracy is found by comparing norms of the discretization error on uniformly refined grids.

Solution verification (verification of the calculation, as per Roache) involves banding the numerical error in a particular solution. This can be done through the grid convergence index (GCI) as developed and discussed by Roache[6,7]. The GCI is built upon the Richardson extrapolation procedure which uses two grids to calculate an estimate of the error in the fine grid solution and/or improve the accuracy of the fine grid solution. Oberkampf and Roy[2] gave a

comprehensive presentation of verification and validation, above and beyond the relevant topics discussed in this section.

2.2 Method of Manufactured Solutions

The method of nearby problems (MNP) is in some regards an extension of the method of manufactured solutions (MMS). The MMS is used for code verification purposes; it provides an exact solution to a modified problem, such that discretization error and ultimately observed orders of accuracy can be calculated. An example of this is presented by Roy, et al.,[8] as they work towards verifying a compressible CFD code. Figure 3 shows the manufactured solution for ρ and the observed orders of accuracy with grid refinement.



a) **b)**

Figure 3: Euler, ρ Variable: a) Manufactured Solution, b) Observed Orders of Accuracy [Reproduced from Roy, et al.[8]]

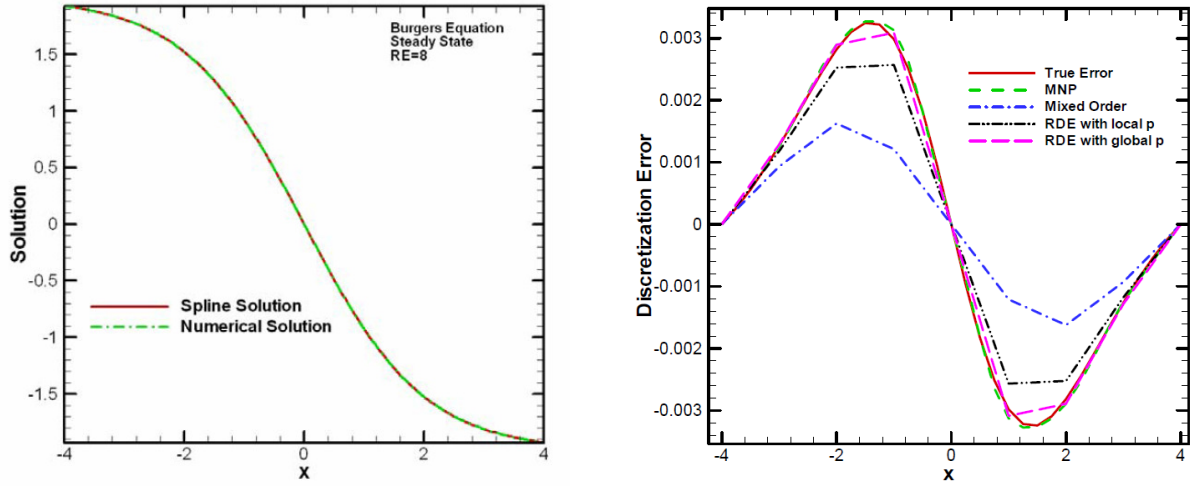
Another example of the use of the MMS is given by Eça and Hoekstra.[9] The authors applied manufactured solutions to a 2D RANS code, PARNASSOS, for code verification purposes. Several cases are examined: finite difference and finite volume methods, different discretization schemes for convective terms, calculating some flow quantities from the manufactured solutions of other quantities. One of the authors' conclusions is that the turbulence equations cannot be overlooked in the code verification process. As shown in their work, first-order discretization of the turbulence equations and second-order discretization of the other governing equations, can cause first-order observed order of accuracies.

One downside to the MMS is that the manufactured solutions are not necessarily physically realistic. An effort can be made to prevent physically impossible solutions (e.g. negative absolute pressures), through careful selection of the terms that form the manufactured solution. However, creating a solution that varies in a similar manner to many physical flow characteristics (boundary layers, sharp gradients, stagnation points, etc.) is much more difficult.

2.3 Method of Nearby Problems

The MNP was developed to produce a physically realistic exact solution to a modified problem, nearby to the original problem of interest. The details on how this is accomplished are presented in Chapter 3. The procedure can be used as an error estimator since the discretization error in the nearby problem can be directly calculated. This discretization error of the nearby problem can be used as an estimate of the discretization error in the original problem. Roy and Hopkins[10] first presented their work on MNP with applications to fully-developed flow through a channel and a lid-driven cavity.

Raju[11] provided a comprehensive look at several methods of curve fitting in one-dimension: standard polynomials, Legendre polynomials, cubic splines, and Hermite splines. In addition he looked at discretization error estimators for one-dimensional cases of Burgers' equation. A Hermite spline fit of the steady-state Burgers' equation and the use of the MNP as an error estimator are shown in Figure 4.



a) b)

Figure 4: Burgers' Equation, *Re* = 8: a) Hermite Spline Fit, b) Disc. Error Estimators [Reproduced from Raju[11]]

In further work, Roy, et al.[12] applied the MNP to the steady-state Burgers' equation. In particular, they examine the use of cubic splines and Hermite splines for curve fitting. A different curve fitting procedure, combining local least-squares fits with weight functions, is applied by Roy and Sinclair[13] to a heat conduction case and the lid-driven cavity problem. Figure 5 shows the u-velocity and streamlines for the driven cavity, as calculated numerically and resulting from the curve fit.

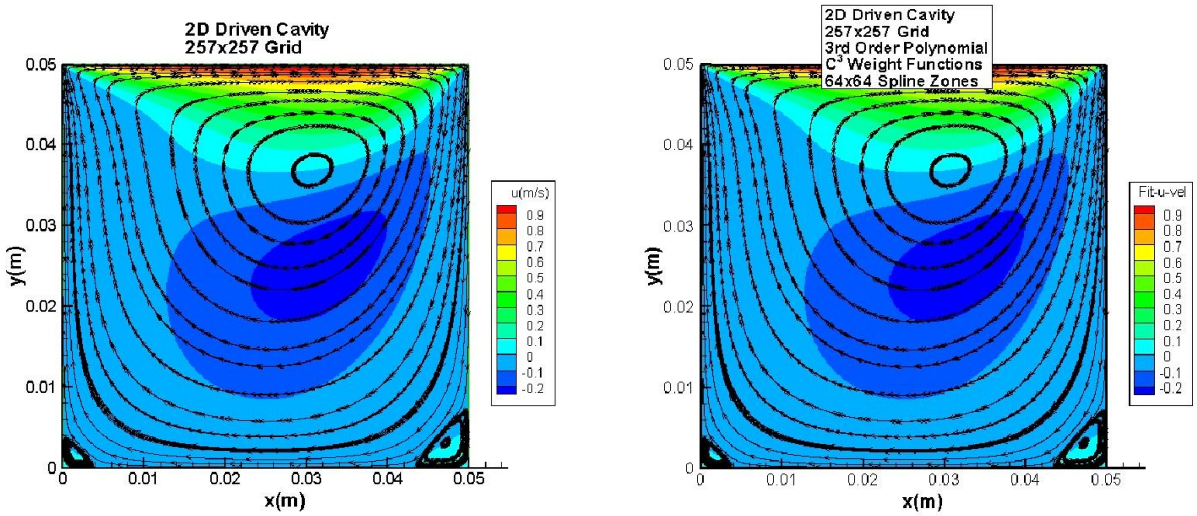


Figure 5: Lid-Driven Cavity, u-Velocity: a) Numerical Solution, b) Curve Fit [Reproduced from Roy and Sinclair[13]]

Kurzen, et al.[14] examined two cases of time-varying Burgers' equation and a specific case of the Euler equations known as Ringleb's flow. This work focused on curve fitting the two-dimensional solutions. The numerical solutions for the Euler equations were solved on non-Cartesian grids, but the curve fitting was done in the Cartesian computational space to allow the local fit and weight function method to be used.

2.4 Curve Fitting

The weight function approach to the curve fitting is important as it allows for local fits to be merged together to create a piecewise continuous fit of the entire solution. The advantage in using local fits is the decreased error in fitting smaller data sets, which will each typically have

less variation than the global data set (covering the entire solution domain). This weight function approach is described by Junkins, et al.,[15] as it applies to modeling topological surfaces from discrete data. A general approach (arbitrary dimension, arbitrary order of continuity enforced) to modeling data is presented in mathematical detail by Jancaitis and Junkins.[16]

2.5 Defect-Correction Methods

The MNP is related in form to the defect-correction method. As described by Layton, et al.[3]: "The *defect-correction method* is an iterative improvement technique which is used to increase the accuracy of a computed solution without introducing a grid refinement." The authors develop a defect-correction procedure based on a finite element method approach to solving the Navier-Stokes equations. The driven-cavity problem is used to demonstrate the effectiveness of the procedure.

A modified problem is defined by including an artificial viscosity term for numerical stability within the discretized governing equations. These modified equations are used to calculate a solution on a coarse grid. The defect is the residual resulting from operating the original governing equations upon the solution to the modified problem. The solution is updated with the correction, which is calculated from the defect and the linearized problem.

The authors' results are compared to a previously published numerical solution that was calculated on a highly refined grid. Figure 6 is reproduced from their paper, showing horizontal and vertical components of velocity, for their original solution, corrected solutions, and the highly refined solution. The star symbols indicate numerical solution data from a previous study referenced in the paper. The results show quite a good improvement after one correction and even better agreement after two corrections.

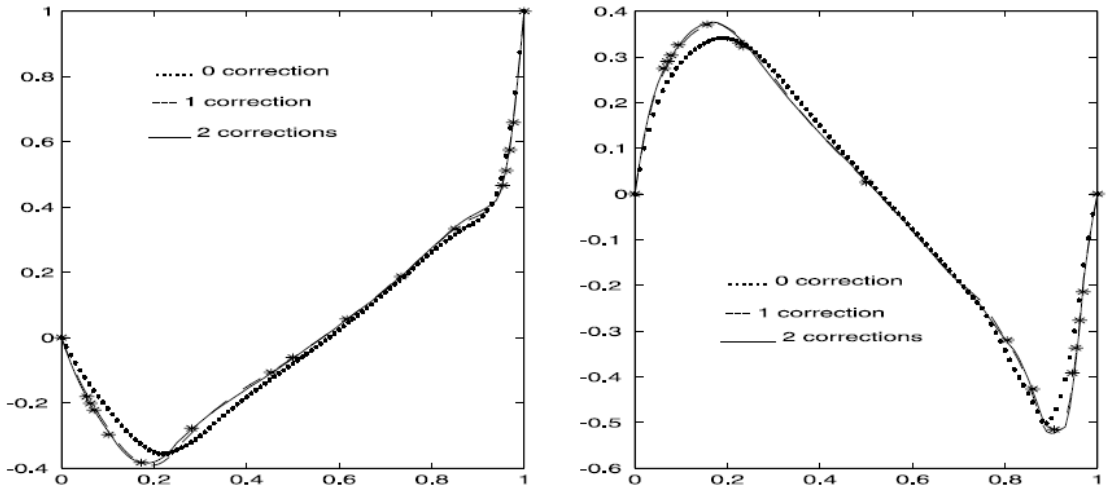


Figure 6: Horizontal and vertical velocity by defect-correction method for *Re*=1000, σ=0.01 Note: σ is artificial viscosity. [Reproduced from Layton, et al.[3]]

Two versions of the defect-correction procedure are presented in mathematical detail by Stetter.[17] He explains that the procedure can be used iteratively to either improve the previous solution or to estimate the error in the prior solution. It is also noted that defect-correction and variations are know as: differential correction, iterated deferred correction, and iterated difference correction.

More examples of the defect-correction approach as applied to CFD problems are given by Rogers and Pulliam.[18] The authors examine three problems solved on overset grids: transonic flow over an airfoil, Oseen vortex flow, and flow over a multi-element airfoil. Rather than use the typical overset grid approach of interpolating values at the boundaries of different grids, forcing functions are used to ensure that the solutions match on overlapping grid regions. The defect-correction procedure is used to improve solution accuracy.

3 Method of Nearby Problems

3.1 Conceptual Introduction / Steps to MNP

The core procedure of the method of nearby problems (MNP) generates an exact solution to a problem that is nearby to the original problem of interest. The steps and definitions in this procedure are:

1. *Original Problem* – Define the original problem of interest, through governing equations, discretization schemes, solution domain, boundary conditions, grid, etc.
2. *Original Numerical Solution* – Generate a numerical solution to the original problem based on the conditions and parameters chosen.
3. *Nearby Exact Solution* – Curve fit the original numerical solution, generating the exact solution to the as-of-now undefined nearby problem.
4. *Source Terms* – Generate source terms by operating the continuous governing equations on the curve fit (nearby exact solution).
5. *Nearby Problem* – Define the nearby problem by adding the source terms to the right-hand side of the equations governing the original problem.

At this point the core procedure is complete; the nearby problem and its exact solution have been generated. One specific use of the MNP is for error estimation, which additionally involves:

6. *Nearby Numerical Solution* – Generate a numerical solution to the nearby problem on the same grid as used for the original numerical solution.
7. *Nearby Discretization Error* – Calculate the nearby discretization error as the difference between the nearby numerical solution and the curve fit (nearby exact solution).

The nearby discretization error can be used as an estimate of the discretization error in the original problem. The accuracy of this estimate is clearly related to how nearby the nearby problem truly is to the original problem of interest. One quantitative measure of nearness is the magnitude of the source terms.

3.2 Discussion of Curve Fitting Principles

There are many types of curve fitting procedures, several of which will be outlined here to give some justification for the specific procedure chosen for later use. In general a curve fit is used to develop an analytical expression for a curve that represents a set of data points. Some

uses of curve fits are: for analytical representation of discrete points, to interpolate between points, or to extrapolate values beyond the data range. In the case of the MNP, it is important to have an analytical representation of the data.

A standard polynomial fit generates a polynomial that exactly passes through each of the data points. A polynomial of order $n-1$ is required to fit the general case of n points. While this is a simple fit to generate and exactly fits the data points, it has several drawbacks. First, it is not efficient for fitting large data sets, as each additional point in the set requires one more term in the polynomial. In addition, while the curve exactly fits the points, the behavior between the points may not be ideal. High order polynomials tend to exhibit high-frequency, oscillatory behavior, which is another reason the standard polynomial fit is not a good choice for large data sets. Furthermore, due to the oscillations, the derivatives at the data points may be wildly inaccurate.

A popular method of curve fitting is the least squares method. This method minimizes the sum of the squares of the differences between the polynomial fit and the data. This allows a polynomial of lower order to be used than with the standard polynomial fit. This can help to reduce oscillations, but comes at the cost of the curve not running exactly through the data points. Figure 7 shows two least squares fits and the standard polynomial fit. The data is sampled from the hyperbolic tangent function, which exhibits a steep slope in the middle of the domain. Note in particular that the ninth-order least squares fit does a good job of representing the values, but does not have the large oscillation at the endpoints like the standard polynomial does.

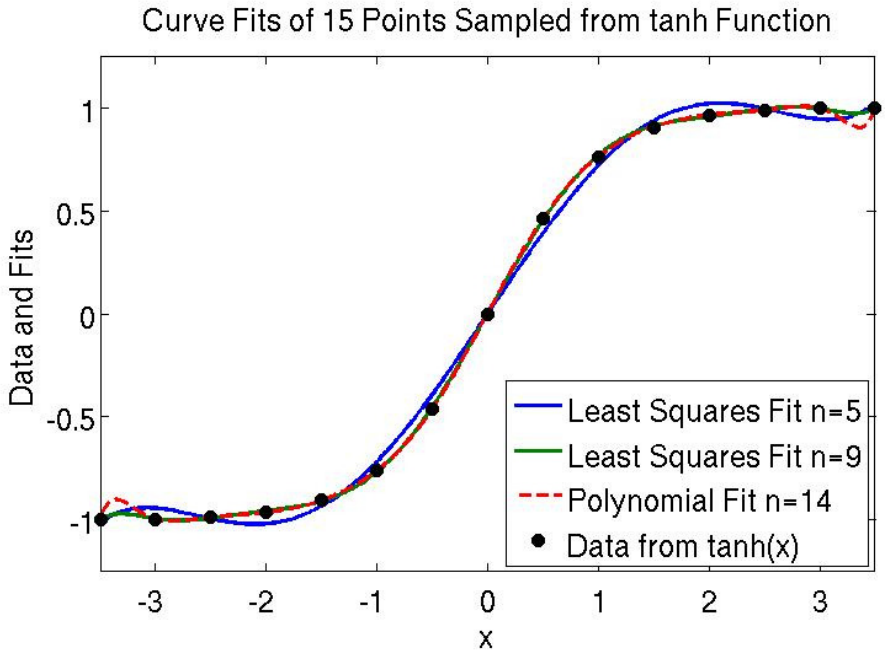


Figure 7: Comparison of Least Squares and Standard Polynomial Fits of $\tanh(x)$ Data

One way to get around the problem of fitting large sets of data is to break the set into subsets, which can each be fitted with a standard polynomial fit or a least squares fit. In this manner a piecewise fit of the global data can be achieved. However, a problem with this is that the interior boundaries of the local fits will not have any continuity in values or derivative enforced. In order to have a continuous, piecewise global fit in 1D, splines are often used. Splines split up the data into smaller zones, which are then fit locally, but with constraints to enforce continuity at the boundaries. If knowledge of the derivatives at the endpoints is known, this can be applied as well. However, standard splines are difficult to extend to multi-dimensional problems, while maintaining continuity at boundaries.

3.3 Weighting Function Approach

One approach to avoiding global curve fits is to split up the domain into local regions, which can each be fitted independently. However, unless certain steps are taken, there is no enforcement of continuity at the boundaries between adjacent regions. Weighting functions can be used to enforce arbitrary levels of continuity. The notation used to reference the level of continuity is C^k, where k is the highest derivative with enforced continuity. For example, C^3

enforces continuity at the boundaries in value, 1st, 2nd, and 3rd derivatives, while C^0 only guarantees that the values will match at the boundaries.

Jancaitis and Junkins[16] solve for the weight functions by formulating a boundary value problem. The resulting one-dimensional weight functions have several important properties:

- Sum to one throughout the domain
- Value is zero at one boundary and one at the other boundary
- Derivatives at boundaries are zero (up to kth derivative)

Figure 8 shows the right weight function, with varying orders of continuity and Figure 9 presents the pair of C^3 weight functions.

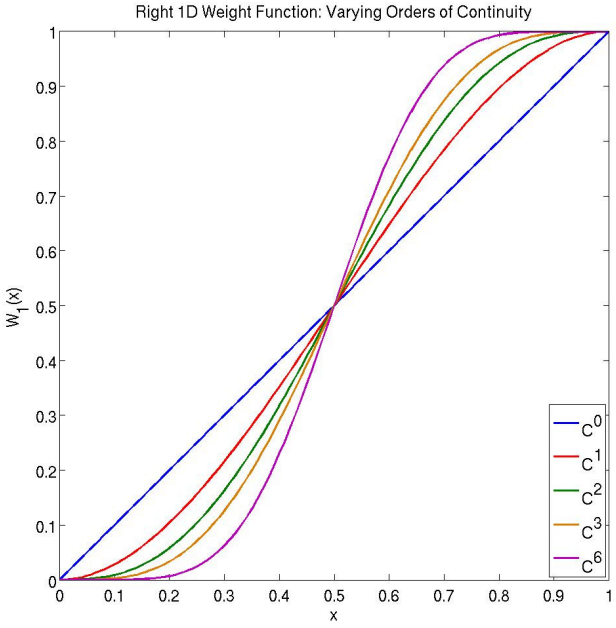


Figure 8: Right 1D Weight Functions: Varying Orders of Continuity

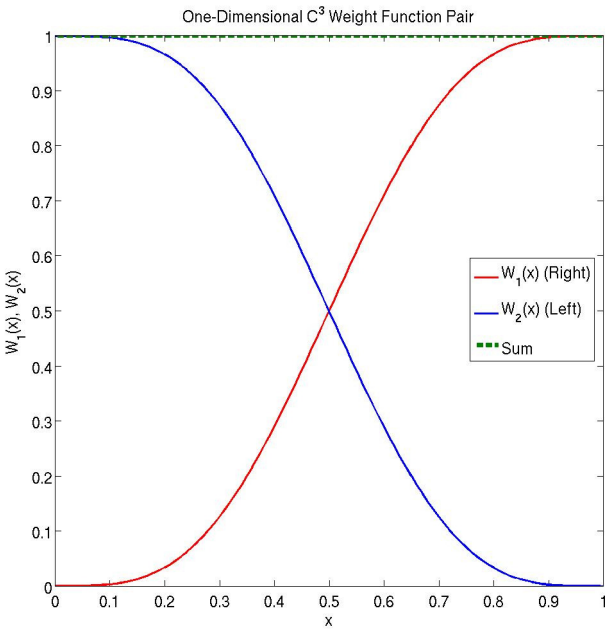


Figure 9: One-Dimensional C3 Weight Function Pair

The weight functions can be found by solving a system of equations with endpoint constraints, for a $k+2$ order polynomial. As an example, the right C^1 weight function is found through the following steps (also shown in Equation 12).

1. Value at left endpoint is zero.
2. First-derivative at left endpoint is zero.
3. Value at right endpoint is one.
4. First-derivative at right endpoint is zero.

Another condition that is not explicitly formulated into the matrix problem is the necessity of the weight functions to sum to 1 throughout the domain. This seems to be enforced by using the endpoint conditions and the smallest polynomial possible, such that the coefficients are uniquely determined. For the example shown, the coefficients of a cubic polynomial are to be solved for. As a cubic can have at most two points of zero slope and these are enforced at the endpoints the polynomial must be bounded by the values at the endpoints, which are selected as 0 and 1. As the other weight function in the set of two 1D weight function will be a reflection about $x = 1/2$, the pair will sum to 1 throughout the domain. This concept extends to the higher-order weight functions as well, due to the higher-order derivatives enforced at their boundaries.

$$\begin{matrix} 1.W(0)=0 \\ 2.W'(0)=0 \\ 3.W(1)=1 \\ 4.W'(1)=0 \end{matrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 3 & 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} -2 \\ 3 \\ 0 \\ 0 \end{bmatrix} \Rightarrow -2x^3 + 3x^2 = x^2(-2x+3) \quad \textbf{(12)}$$

Note that the left endpoint conditions will cause the first $k+1$ coefficients ($c_0 \rightarrow c_k$) to be zero. This allows the weight functions to be written in the form of Equation 13

$$x^{k+1}\left(c_k x^k + c_{k-1}x^{k-1} + \ldots + c_1 x + c_0\right). \quad \textbf{(13)}$$

The first seven right weight functions are given in Table 1.

Table 1: One-Dimensional Weight Functions

Continuity Order	Right Weight Function
0	$x^1(1)$
1	$x^2(-2x+3)$
2	$x^3(6x^2-15x+10)$
3	$x^4(-20x^3+70x^2-84x+35)$
4	$x^5(-70x^4+315x^3-540x^2+420x-126)$
5	$x^6(-252x^5+1386x^4-3080x^3+3465x^2-1980x+462)$
6	$x^7(924x^6-6006x^5+16380x^4-24024x^3+20020x^2-9009x+1716)$

A simple example of the one-dimensional weight function approach is reproduced from Roy and Sinclair[13] in Figure 10. The numerical data is sampled from a sine wave. The domain is broken up into four zones (Note that the terminology "zone" and "region" are reversed between Roy and Sinclair's paper and this thesis). Local fits are generated over five regions, using a quadratic least squares method. Each interior region consists of two adjacent zones. The local fit

regions at the ends of the domain consist of only one zone, as the other zone that would be included lies outside of the global domain. These five local fits are shown in Figure 10a.

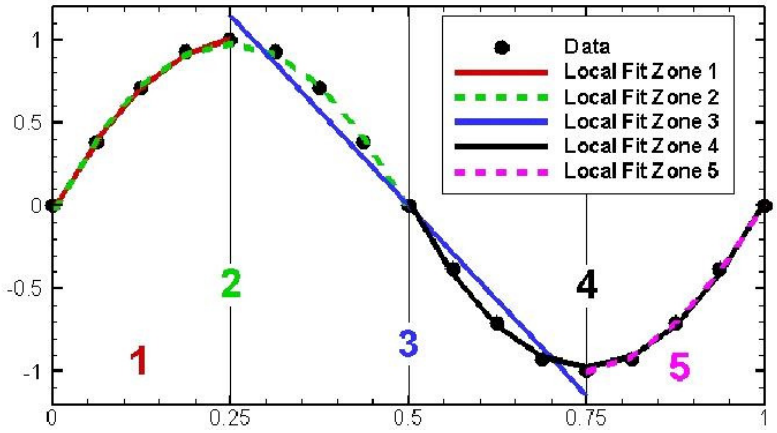
The weight functions are shown in Figure 10b. Each pair of weight functions covers one zone. In this case C^2 weight functions were chosen, to provide continuity in the value, first-derivative, and second-derivative at the boundaries between zones.

Each piece of the final fit is generated as in Equation 14

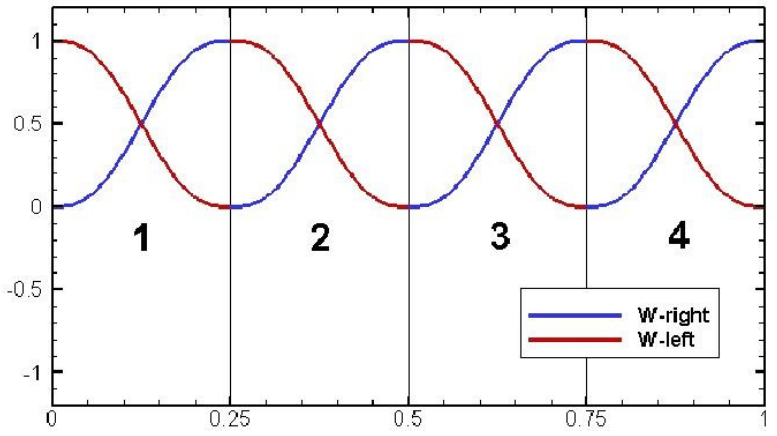
$$F_i = f_i * W_L + f_{i+1} * W_R \tag{14}$$

where F_i is one piece of the final fit, f_i is a local fit, and W is a weight function, with L and R representing a value of one at the left and right side of the domain respectively. The final piecewise fit is shown in Figure 10c.

a)



b)



c)

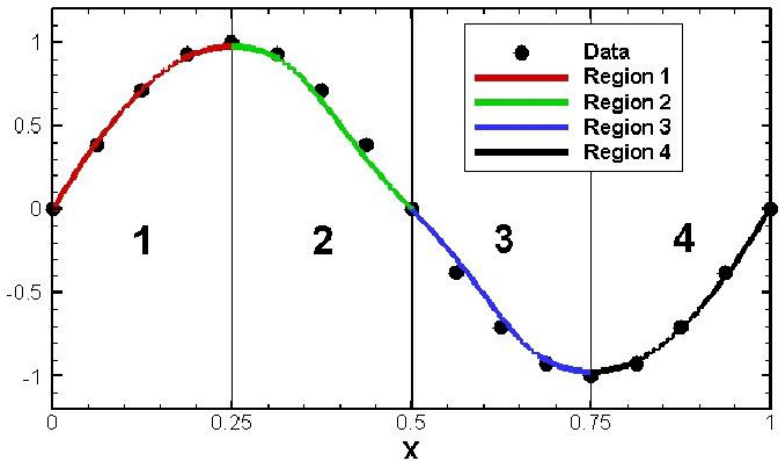


Figure 10: 1D Fit Example: a) Quadratic LS Fits, b) C2 WFs, c) Final Piecewise Fit [Reproduced from Roy and Sinclair[13]]

In two dimensions there is a set of four weight functions. As shown in Figure 11, each weight function has two adjacent boundaries with zero value and two adjacent boundaries that mimic the 1D weight functions, culminating at singular value of one at a corner. As with the 1D case, summing the set of weight functions gives a value of one throughout the domain. It is important to note that the 2D weight function can be constructed by multiplying two 1D weight functions together (where the 1D functions have different independent variables), such as shown in Equation 15.

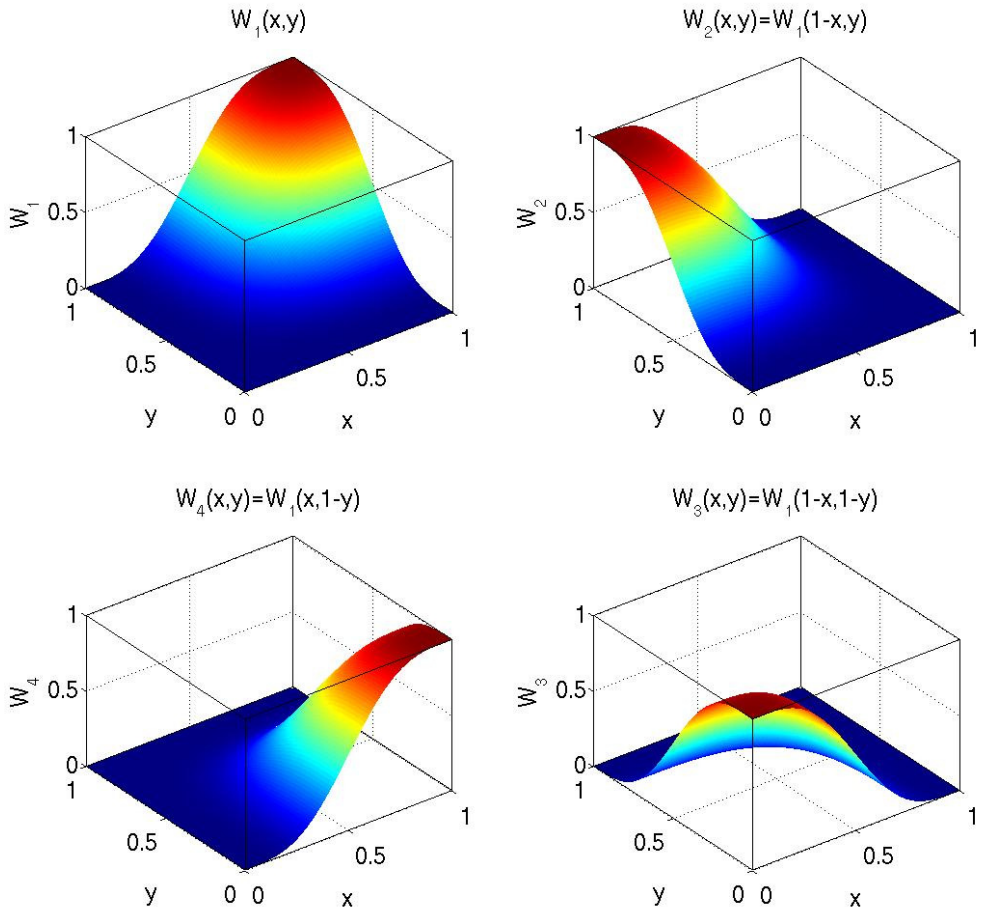


Figure 11: Set of Four Two-Dimensional Weight Functions

$$W_1(x,y)\big|_{2D} = W_1(x)\big|_{1D} * W_1(y)\big|_{1D}$$

$$\left\{x^3\left(6x^2-15x+10\right)\right\} * \left\{y^3\left(6y^2-15y+10\right)\right\} \qquad \textbf{(15)}$$

$$= x^3y^3\left(36x^2y^2-90x^2y+60x^2-90xy^2+225xy-150x+60y^2-150y+100\right)$$

Three-dimensional weight functions follow the same general form as the one- and two-dimensional weight function. Figure 12 shows six slices of one three-dimensional weight function. The slices are presented due to the difficulty of representing data that is dependent on three independent variables. Note that these weight functions could be used for a steady-state problem with three special dimensions or for a time-varying problem with two spatial and one time dimension. In the latter case the weight function could be visualized as an animation, with

the weight function growing from zero everywhere at $t = 0$ to the full two-dimensional weight function at $t = 1$.

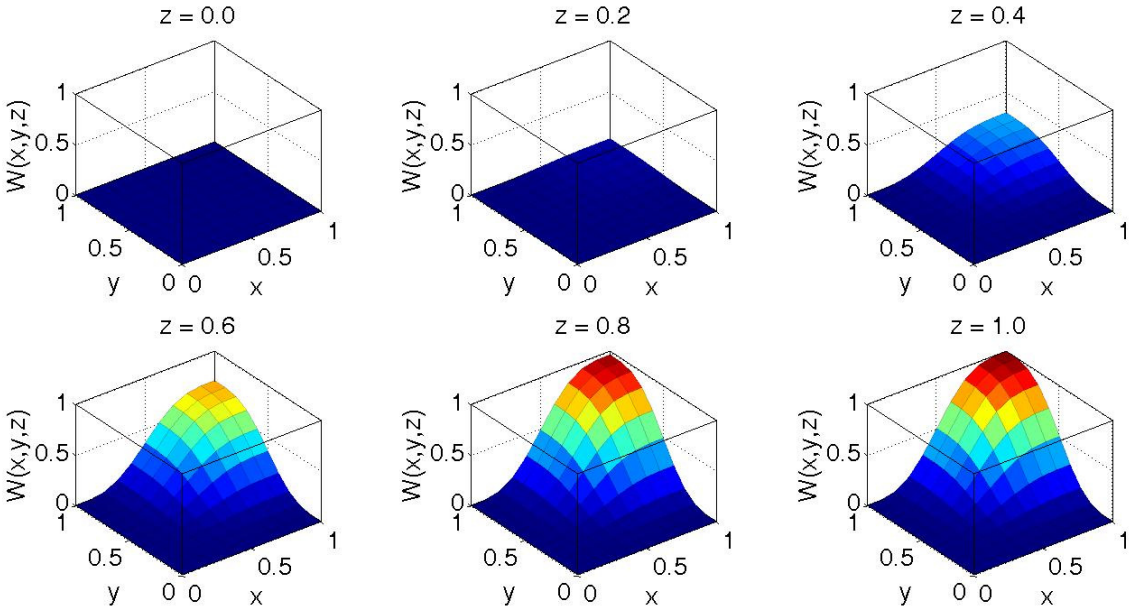


Figure 12: Six Slices of a Three-Dimensional Weight Function

A Matlab code attached in the Appendix A generates one- to four-dimensional weight functions of arbitrary (user defined) continuity. The user can input the order of continuity, the dimension of the weight function, and (optionally) a plotting option. The code returns the weight function in symbolic form.

3.4 Applications of MNP

The method of nearby problems can be used to generate analytical exact solutions to the nearby problem. It is important to note that as the nearby problem is similar to the problem of interest, the solution generated is a physically-realistic exact solution. This alone is a useful accomplishment. In addition, a numerical solution to the nearby problem can be used to calculate the discretization error for the nearby problem. This serves as an estimate of the discretization error in the original problem.

4 One-Dimensional Case Study

4.1 Burgers' Equation

Burgers' equation is a nonlinear PDE that can be used to represent various flow phenomena. The full (time-varying) equation is given in Equation 16 (previously Equation 1)

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{16}$$

where t is time, x is spatial location, u is velocity, and ν is kinematic viscosity. The steady-state equation removes the temporal derivative and reduces the equation to one dimension (Equation 17).

$$u\frac{du}{dx} = \nu\frac{d^2 u}{dx^2} \tag{17}$$

A solution to the steady-state equation is given by Benton and Platzman[1] in Equation 18. This solution represents a steady viscous shock. In this paper the solution has a domain reference length of $L = 8$ m and velocity reference $u_{ref} = 2$ m/s

$$u'(x', t') = -2\tanh(x') \tag{18}$$

where the primes denote dimensionless variables, calculated as $x' = x/L$, $t' = t\nu/L^2$, and $u' = uL/\nu$. A scaling factor α is also used to scale the solution appropriately within the domain. The relationships between the scaled and unscaled coordinates are $\bar{x} = x/\alpha$, $\bar{t} = t/\alpha^2$, and $\bar{u} = \alpha u$. A Reynolds number can be chosen by setting the scaling factor and kinematic viscosity as $\alpha = Re/2$ and $\nu = u_{ref}L/Re$. Choosing a higher Reynolds number will lead to a stronger shock (steeper slope).

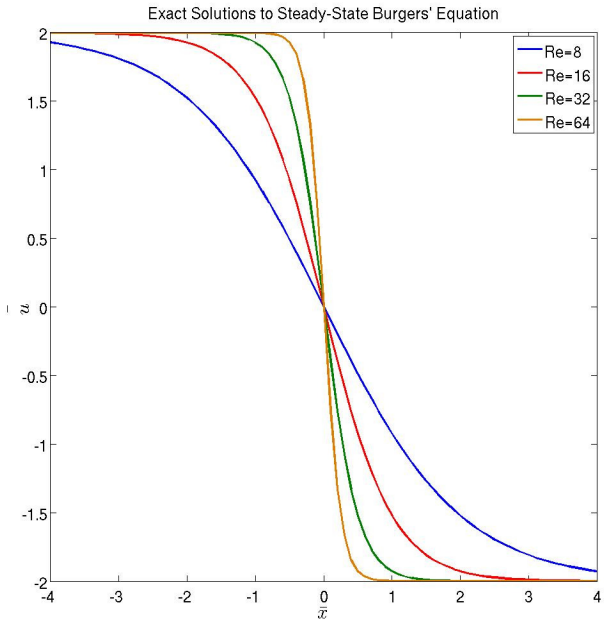
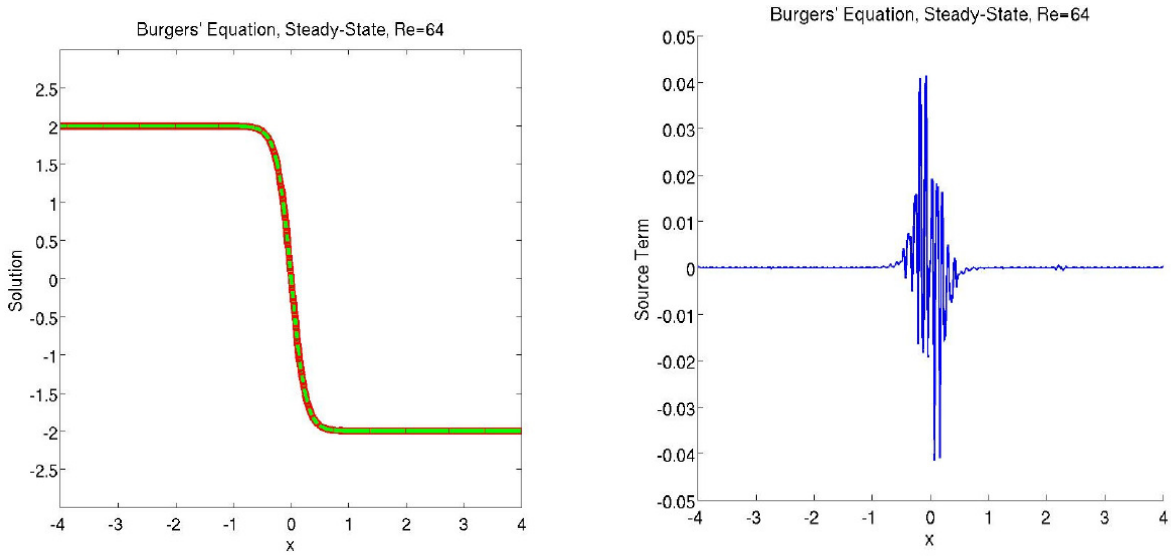


Figure 13: Exact Solution to Steady-State Burgers' Equation

4.2 Curve Fitting Results

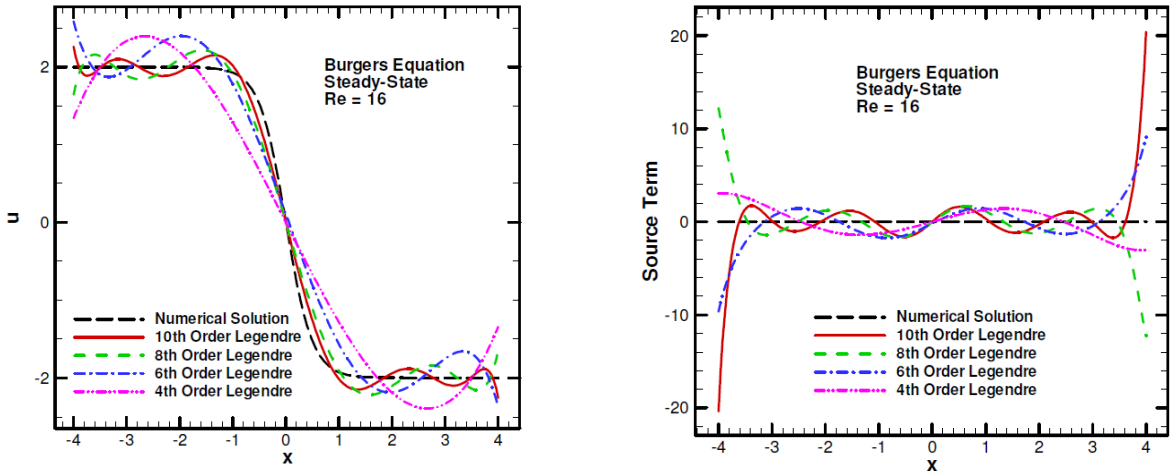
The final fits are determined as described previously; the local least squares fits are merged with the weight functions to generate the piecewise final fit. This is done in MATLAB with symbolic expressions. This case is for a Reynolds number of 64, with 513 points in the numerical solution. 64 fit zones are used with fifth order local least squares fits and C^3 weight functions. Figure 14 shows the numerical solution and curve fit, along with the source term.



a) b)

Figure 14: Burgers' Equation LS and Weight Function a) NS and CF and b) Source Term [Reproduced from Kurzen, et al.[14]]

For comparison purposes, an example of global curve fits is reproduced from Roy, et al.[12] in Figure 15. Note that this case has a Reynolds number of 16, which causes a more gradual transition at the shock. This makes the problem easier to fit, yet the curve fit is not as accurate as the previous example that uses local least squares fits and weight functions.

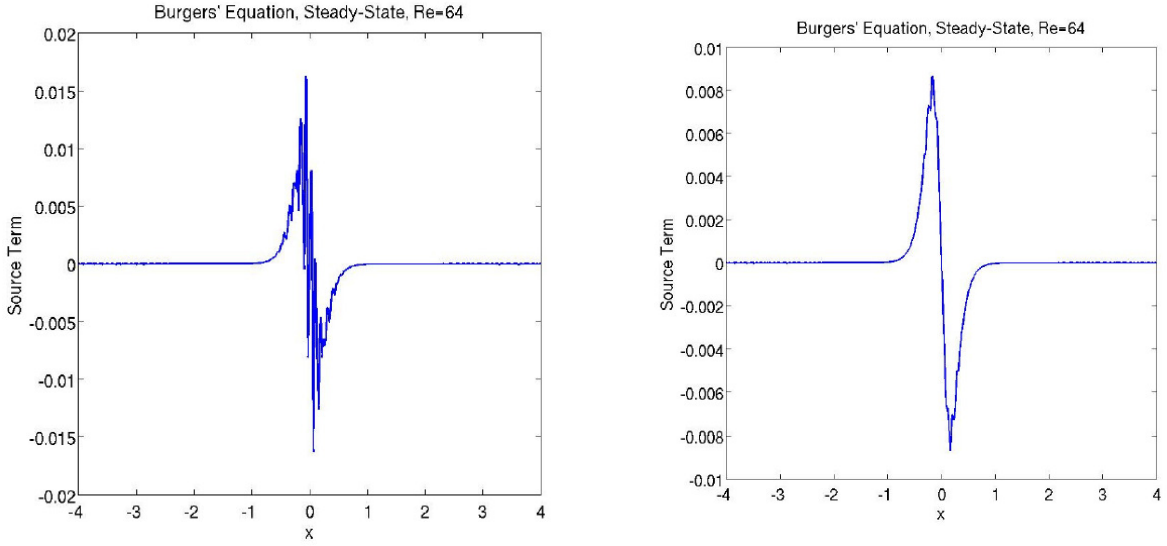


a) **b)**

Figure 15: Burgers' Equation with Global Curve Fits a) Fits and b) Source Terms [Reproduced from Roy and Sinclair[13] (L) and Roy, et al.[12] (R)]

4.3 High-Frequency Oscillations

The source terms often have a high frequency oscillation present. The magnitude of this oscillation seems to decrease when the order of the local fits is increased. As an example, Figure 16 shows the difference between the source terms resulting from sixth order local fits and seventh order local fits. The examples shown are for a Reynolds number of 64 and use 64 zones and C^3 weight functions.



a) b)

Figure 16: Effect of Local Fit Order on Source Terms a) 6th Order and b) 7th Order [Reproduced from Kurzen, et al.[14]]

4.4 Scaling of Least Squares Fits

The coefficients of the local least squares fits are found by solving a system of equations in matrix format as shown in Equation 19. This will produce a fit of y as a function of x. The example shown would result in a third order fit based on n points

$$\begin{bmatrix} \sum_{i=1}^{n} x_i^0 & \sum_{i=1}^{n} x_i^1 & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^1 & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^5 \\ \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 & \sum_{i=1}^{n} x_i^5 & \sum_{i=1}^{n} x_i^6 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \left(x_i^0 y_i \right) \\ \sum_{i=1}^{n} \left(x_i^1 y_i \right) \\ \sum_{i=1}^{n} \left(x_i^2 y_i \right) \\ \sum_{i=1}^{n} \left(x_i^3 y_i \right) \end{bmatrix} \tag{19}$$

where x is the independent variable, y is the dependent variable, i is an index, and c is a coefficient in the least squares fit, with the index on the coefficient relating to the power of the term it affects (e.g. c_2 is multiplied by x^2 in the polynomial). If care is not taken to shift and scale the data being fitted, the matrix can end up being ill-conditioned which can cause a loss of accuracy in the coefficients or even no solution at all. One approach to this problem is to scale the values such that when they are raised to an exponent, they do not grow to large. For instance, if the x values are shifted and scaled from 0 to 1, then the elements of the coefficient matrix will be bounded by 0 and n.

4.5 Discrete Values and Coefficients

One issue that arises with the curve fitting procedure is how to report the curve fit. One option is to report the coefficients of each piece of the piecewise final fit. This has the advantage of including all of the original fit information and representing a continuous solution across the domain. Another approach is to output the values of the final fit at each grid location, resulting in an array of values. This ties the fit to a particular grid however (as the continuous information between grid nodes is lost). In addition, the source terms also need to be generated and evaluated at the grid locations as well, and this information cannot be reconstructed from discrete fit values without introducing errors.

Both of these approaches have been tested in one-dimension and are shown to have equivalent values at grid nodes. For the subsequent two-dimensional cases, the later approach of reporting only the discrete values has been used. The 2D fitting code is designed to calculate the final fit and necessary derivatives, which are then evaluated at all grid locations. The curve fits and source terms are then outputted as arrays which makes subsequent calculation (such as running the nearby problem) easier to implement. There is no additional evaluation of coefficients needed.

The coefficient output approach is also with merit. If this procedure is implemented, there is a choice to make if grid transformations are used. In the transformed computational space each zone has a square domain. First, the coefficients could be reported in the transformed coordinate system, where they are applicable over the square zones. Alternatively, if fit coefficients are outputted for the physical domain the edges of each zone would need to be transformed and also reported. This adds to the complexity of the problem and requires storage of extra information but provides the fit in physical terms and space.

5 Two-Dimensional Case Studies

5.1 Burgers' Equation

The time-varying (unsteady) Burgers' equation is shown in Equation 20 (previously Equations 1 and 16)

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{20}$$

where t is time, x is spatial location, u is velocity, and ν is kinematic viscosity. One exact solution models viscous shock coalescence.[1] Two shocks travel from opposite ends of the x-axis and meet at the center. The two viscous shocks merge to form the steady-state shock as time increases. This solution is given in Equation 21 (previously Equation 2)

$$u' = -\frac{2\sinh(x')}{\cosh(x') + e^{-t'}} \tag{21}$$

where the prime coordinates represent non-dimensionalized coordinates. Recall that the equations are solved in a scaled, dimensionalized coordinate system. The relationship between these coordinate systems was discussed in Section 4.1. Another exact solution is a pulse decay.[1] Two equal but opposite pulses formed at the origin decay as time increases. This solution is given in Equation 22.

$$u' = -\frac{x'/t'}{1 + t'^{1/2} e^{x'^2/4t'}} \tag{22}$$

5.1.1 Shock Coalescence

The shock coalescence case is shown in Figure 17. The x-axis is the independent spatial axis varying from -4 m to 4 m and the t-axis represents time over a two second period. The u-axis measures the velocity, which varies from -2 m/s to 2 m/s. Note that two different grid levels are used in this section: a coarse grid with 65 points in each direction and a fine grid with 257 points in each direction.

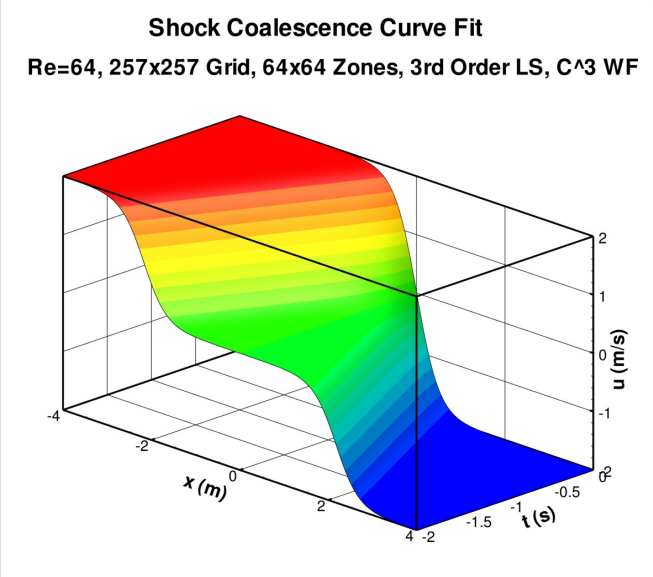


Figure 17: Shock Coalescence Curve Fit Surface Plot

Two-dimensional contour plots comparing the numerical solution and the curve fit on the fine grid are shown in Figure 18. The local fits for all of the two-dimensional results are done with a weighted least squares approach. The weights come from the same weighting functions that are used to patch the local fits together. Using the weighted least squares approach in this manner is intended to improve the local fits at the critical regions. A comparison of the curve fit error on the different grid levels is shown in Figure 19, while a comparison of the source term is shown in Figure 20. Both of these quantities have the largest values along the sharp gradients in the curve fit, where there are oscillations present. By refining the grid, these oscillations can be reduced in magnitude. Increasing the number of grid points that cover the sharp gradient helps to improve the resolution of the numerical solution and also provides for a better curve fit. Equation 23 is used to compute percent curve fit error, with the maximum shown for both grids in Table 2. The maximum percent curve fit error drops over two magnitudes when the grid is refined by a factor of four.

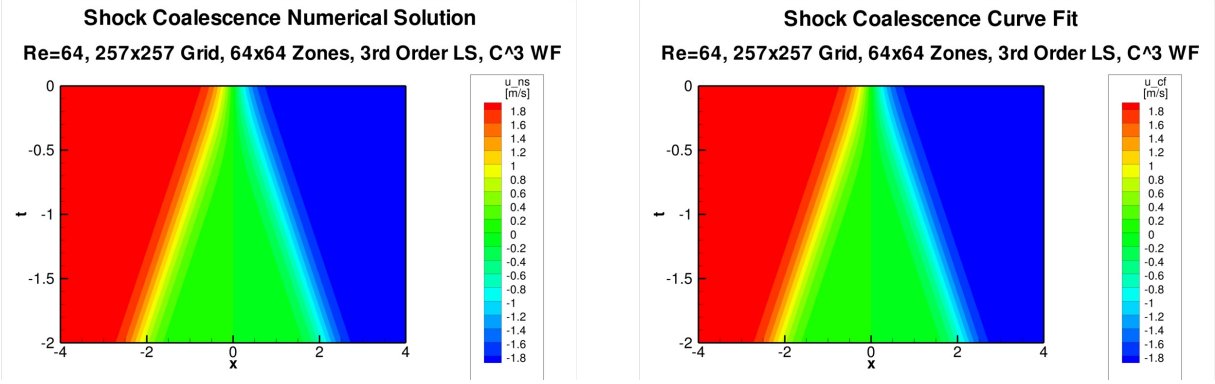


Figure 18: Shock Coalescence Numerical Solution and Curve Fit

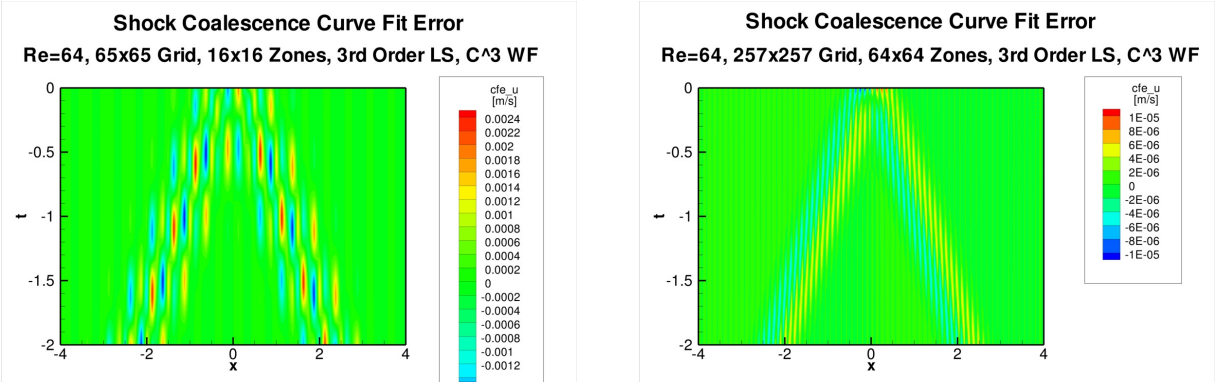


Figure 19: Shock Coalescence Curve Fit Error Comparison

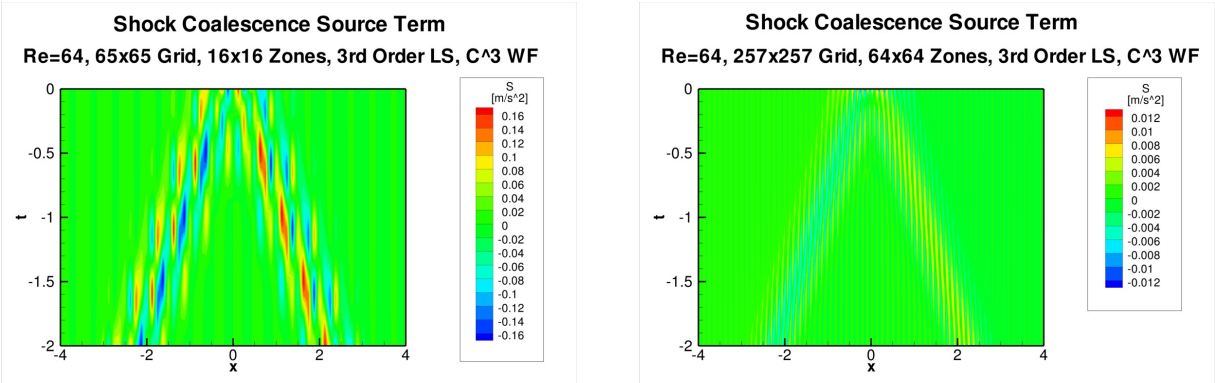


Figure 20: Shock Coalescence Source Terms Comparison

$$\varepsilon_{CF,\%} = \left| \frac{u_{CF} - u_{NS}}{u_{ref}} \right| * 100 \qquad \textbf{(23)}$$

In this equation $\varepsilon_{CF,\%}$ is percent curve fit error, u_{CF} is the curve fit, u_{NS} is the numerical solution, u_{ref} is the reference velocity of 2 m/s. The percentage curve fit error is defined this way with a reference velocity in the denominator for both Burgers' equation cases because the velocities pass through zero.

Table 2: Shock Coalescence Maximum Percentage Curve Fit Errors

% CF Error	u
Coarse Grid	0.13
Fine Grid	0.00058

A comparison of the original problem and nearby problem discretization errors is presented in Figure 21, for the coarse grid. The fine grid results are shown in Figure 22 and x-y plots showing a slice at $t = -1$ are given in Figure 23. The nearby problem discretization error matches the qualitative behavior of the original problem discretization error and the magnitudes are also close to one another. This holds true for both grid resolutions. However, the x-y plots show that the nearby problem discretization error exhibits noticeable oscillations near the shock region. These are minimized in the fine grid case.

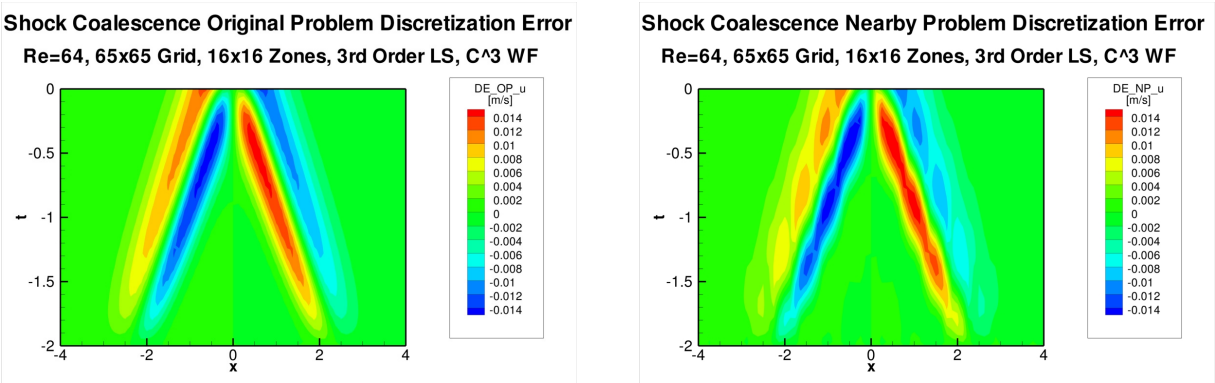


Figure 21: Shock Coalescence Discretization Error Comparison, Coarse Grid

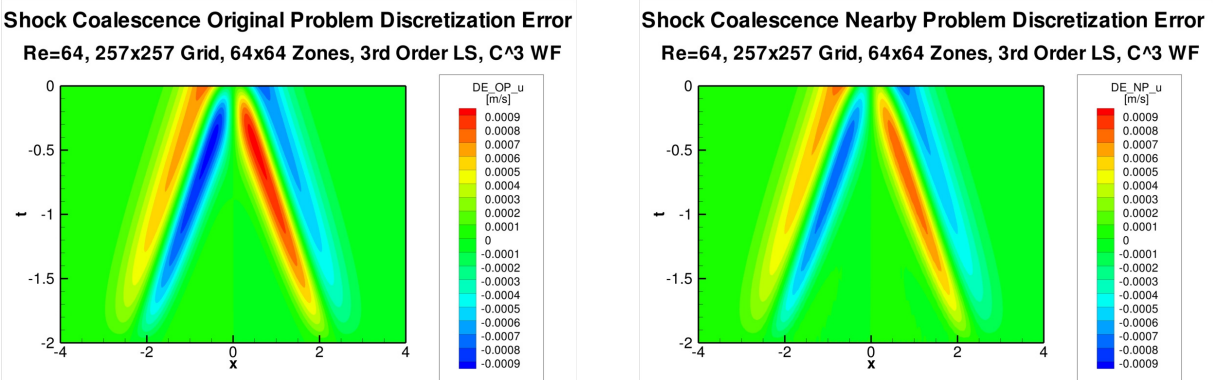


Figure 22: Shock Coalescence Discretization Error Comparison, Fine Grid

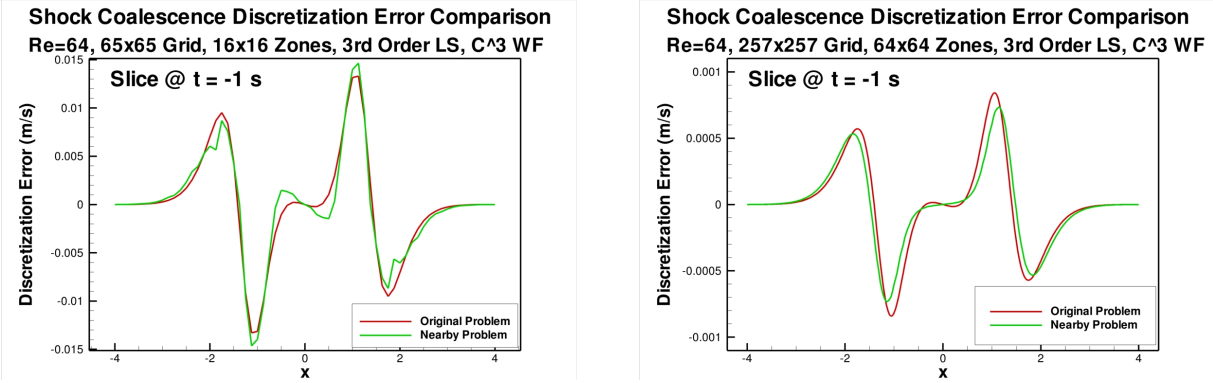


Figure 23: Shock Coalescence Discretization Error Comparisons, x-y Plots

5.1.2 Pulse Decay

The pulse decay curve fit is shown as a 3D surface in Figure 24. The axes are the same as for the previous case. The pulse decay is shown from 0.2 seconds to 1 second. The numerical solution and curve fit are shown in Figure 25. Curve fit error on the coarse and fine grids are shown in Figure 26, while the source term is shown in Figure 27. The curve fit of the pulse decay is more difficult than that of the previous shock coalescence case. The pulse decay has sharp solution gradients where the initial pulses are formed. This represents a difficult region to fit accurately. In addition the pulses are strongest right at a boundary. The magnitude of the oscillations in the curve fit error and source term are decreased by refining the grid. The maximum percent curve fit errors are shown in Table 3, where grid refinement decreases the error by two orders of magnitude.

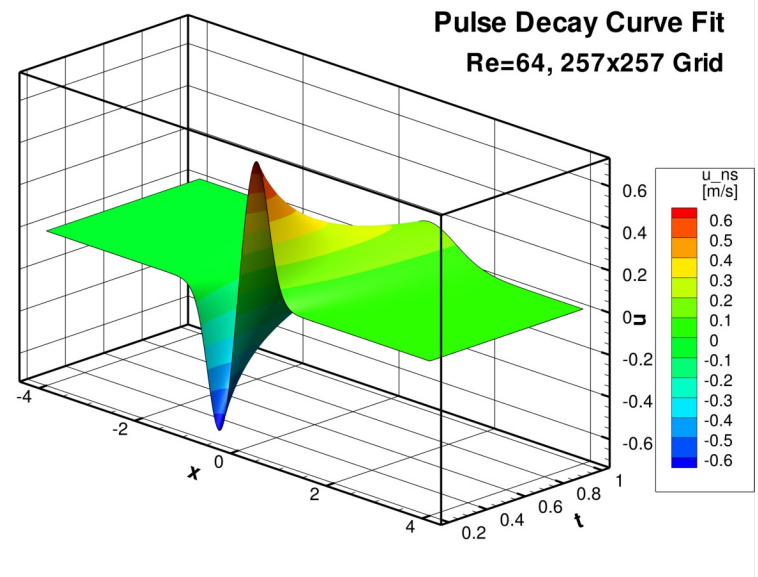


Figure 24: Pulse Decay Curve Fit Surface Plot

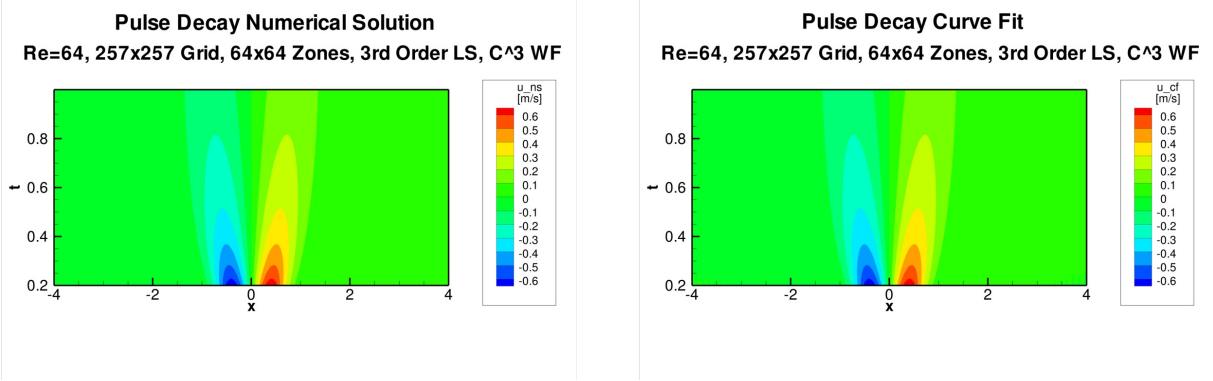


Figure 25: Pulse Decay Numerical Solution and Curve Fit

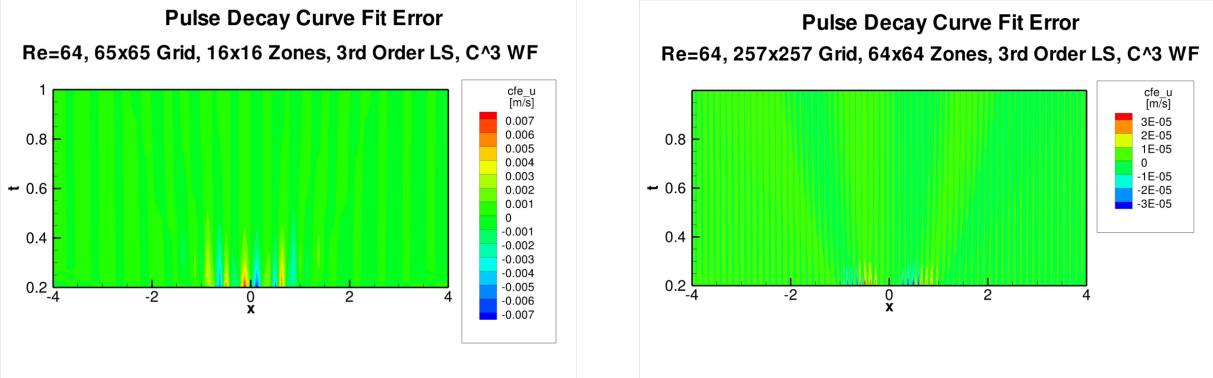


Figure 26: Pulse Decay Curve Fit Error Comparison

Table 3: Pulse Decay Maximum Percentage Curve Fit Errors

% CF Error	*u*
Coarse Grid	0.37
Fine Grid	0.0020

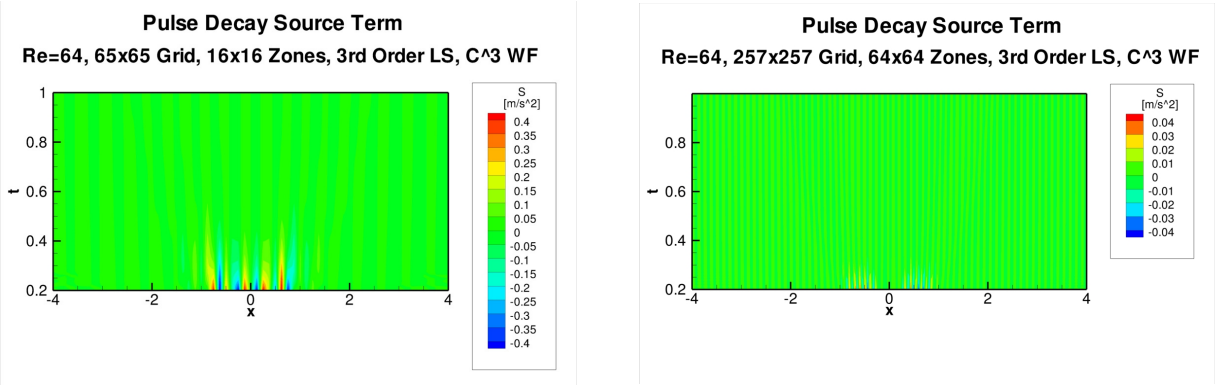


Figure 27: Pulse Decay Source Term Comparison

The discretization errors in the original and nearby problems are presented in Figure 28 for the coarse grid. The nearby problem discretization error matches the qualitative behavior of the original problem discretization error. The discretization errors on the fine grid are shown in Figure 29. This refined grid actually does a worse job at estimating the error in the original problem. The two complete oscillations seen in the original problem discretization error are represented by only one oscillation in the nearby problem discretization error. This behavior can also be seen in Figure 30 with x-y plots comparing the two discretization errors for both coarse and fine grids.

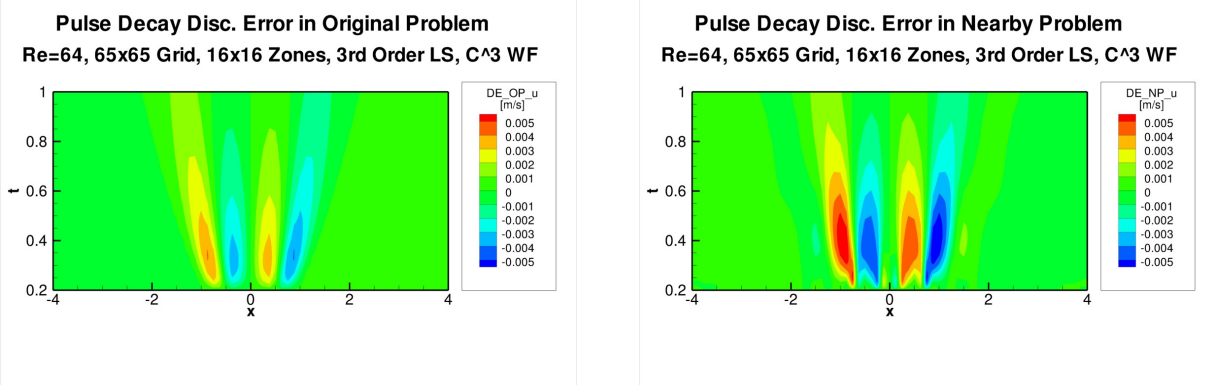


Figure 28: Pulse Decay Comparison of Discretization Error, Coarse Grid

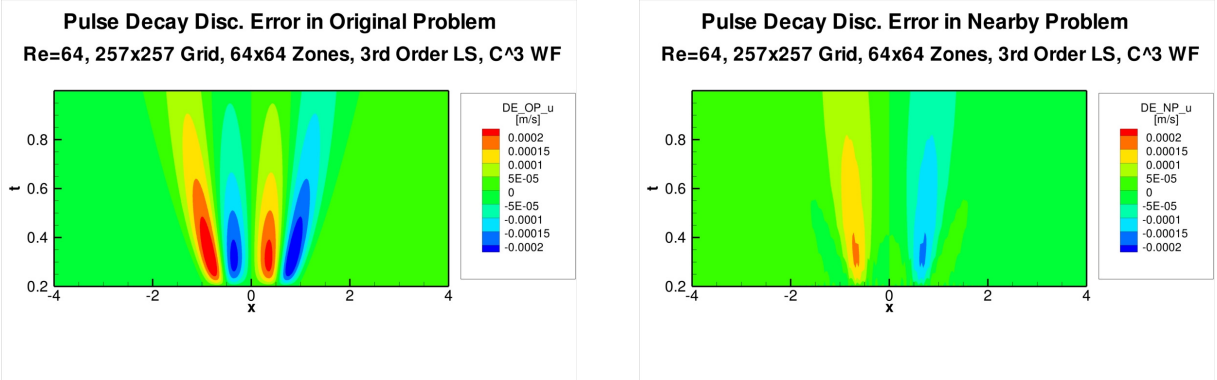


Figure 29: Pulse Decay Comparison of Discretization Error, Fine Grid

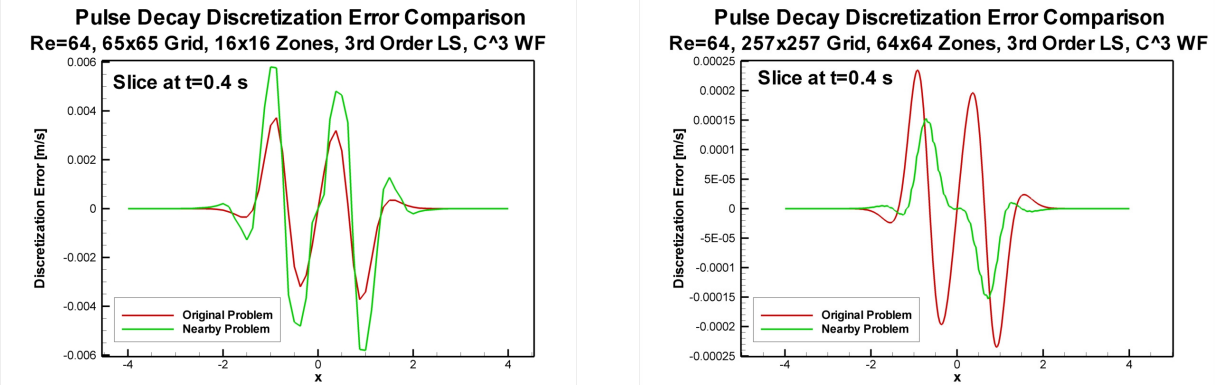


Figure 30: Pulse Decay Comparison of Discretization Error, x-y Plots

One change that affects the discretization error is the temporal discretization scheme used in the Burgers' equation solver. Previously the Crank-Nicolson scheme was used. By changing to a fully-implicit scheme the nearby problem discretization error matches the original problem discretization error much better.

Figure 31 shows the curve fit error and source terms for the fine grid, fully-implicit case. These are very similar to the fine grid, Crank-Nicolson case shown above, although the source term is slightly larger. Figure 32 compares the discretization error between the original and nearby problems. These discretization errors are a magnitude larger than for the Crank-Nicolson case, but more importantly the new discretization errors match each other much better. The features in the original problem discretization error are matched in the nearby problem discretization error. This is shown in more detail as an x-y plot in Figure 33. A theory for why this may be the case has to do with the boundary conditions and source terms. As this is a time-

dependent problem, the solver uses the exact solution as the initial condition. With the Crank-Nicolson scheme, the nearby problem uses some source term information from that initial condition. In other words, the second time step will include part of the initial time step source term and part of the current time step source term. With the fully-implicit scheme, the source term is only drawn from the current time step, which may be a more appropriate way to handle the nearby problem.

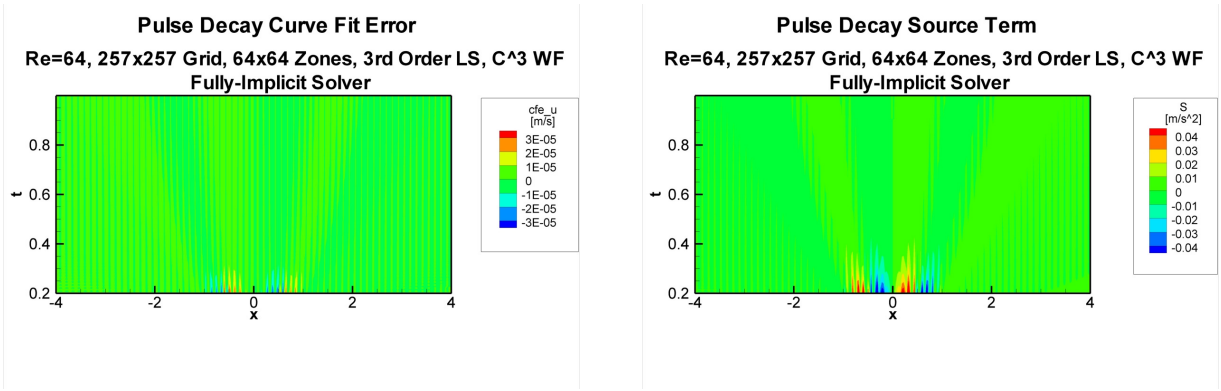


Figure 31: Pulse Decay Curve Fit Error and Source Term, Fine Grid, Fully-Implicit

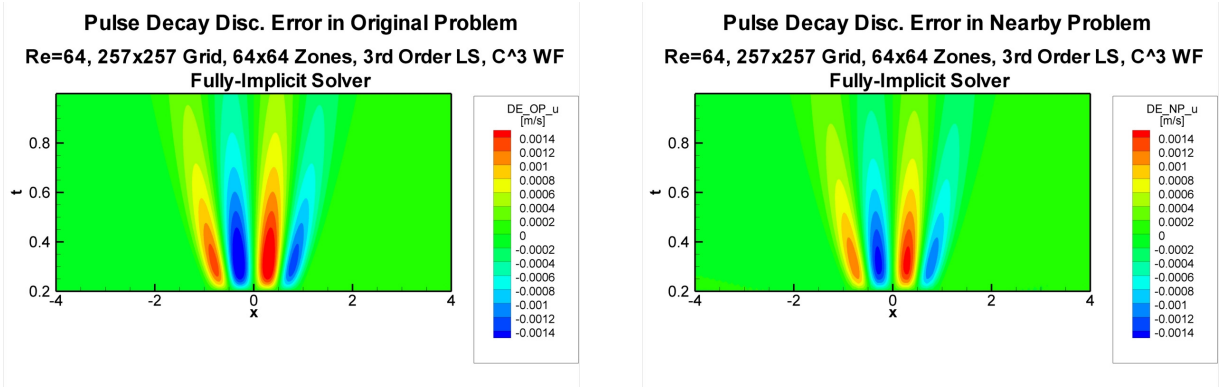


Figure 32: Pulse Decay Discretization Error Comparison, Fine Grid, Fully-Implicit

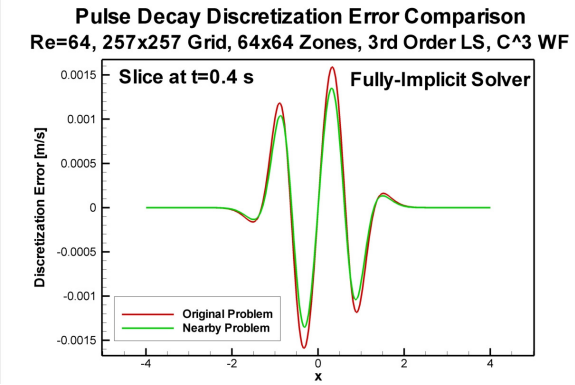


Figure 33: Pulse Decay Disc. Error Comparison, Fine Grid, Fully-Implicit, x-y Plots

5.2 Euler Equations

The Euler equations describe inviscid, compressible fluid flow. The set of four equations governing two-dimensional flow is given in conservation form in Equation 24

$$\begin{aligned}
&\frac{\partial(\rho)}{\partial t}+\frac{\partial(\rho u)}{\partial x}+\frac{\partial(\rho v)}{\partial y}=0\\
&\frac{\partial(\rho u)}{\partial t}+\frac{\partial\left(\rho u^2+p\right)}{\partial x}+\frac{\partial(\rho uv)}{\partial y}=0\\
&\frac{\partial(\rho v)}{\partial t}+\frac{\partial(\rho vu)}{\partial x}+\frac{\partial\left(\rho v^2+p\right)}{\partial y}=0\\
&\frac{\partial(\rho e_t)}{\partial t}+\frac{\partial(\rho u e_t+pu)}{\partial x}+\frac{\partial(\rho v e_t+pv)}{\partial y}=0
\end{aligned} \tag{24}$$

where ρ is density, u is velocity in the x-direction, v is velocity in the y-direction, and p is pressure. These four variables are referred to as the primitive variables. The e_t term represents the total energy and is expressed in primitive variables as shown in Equation 25

$$e_t=\frac{1}{\gamma-1}\frac{p}{\rho}+\frac{1}{2}\left(u^2+v^2\right) \tag{25}$$

where γ is the ratio of specific heats. The gas is assumed to be calorically perfect.

5.2.1 Curvilinear Manufactured Solution on Grid A

The first two-dimensional case examined is a manufactured solution to the Euler equations. During the process of debugging the curve fitting procedure it was helpful to have a

smooth, well-behaved numerical solution. The exact solution also allows the true discretization error to be directly calculated for the original problem, which is useful for evaluating the error estimate. Essentially an exact solution is manufactured, source terms are calculated, the MMS governing equations defined, and then the MMS problem is solved numerically. This numerical solution is then used as the basis for the MNP curve fit. A curvilinear grid was chosen with cells that included significant stretching, skewing and curvature, as shown in Figure 34.

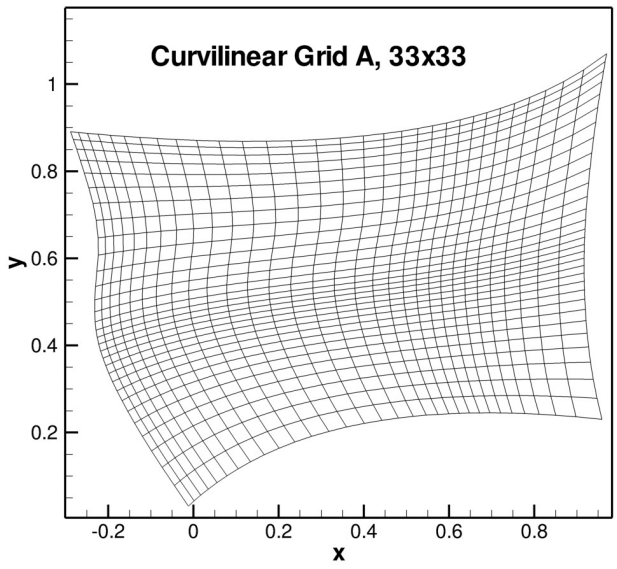


Figure 34: Two-Dimensional Curvilinear Grid A

The Euler equation flow solver uses the finite volume method for discretizing the governing equations. Specifically, Roe's method is used for calculating the interface fluxes. MUSCL extrapolation is used for a second-order accurate, fully upwinded scheme. Local time-stepping is also implemented. For the manufactured solution cases, Dirchlet boundary conditions are applied.

The Euler code is run to generate a cell-centered numerical solution on the curvilinear grid. The primitive variables are placed on a computational (Cartesian) grid and interpolated and extrapolated to grid nodes. This allows the curve fit to span the entire domain of the problem. Figure 35 shows a small section of a computational grid as a reference for explaining how the nodal values are calculated. The interpolation is done for interior grid nodes as given in Equation 26. Boundary nodes, excluding those at corners, are extrapolated using Equation 27, which first

extrapolates to the boundary face centers and then averages neighboring values. The corner nodes are extrapolated along diagonals as shown in Equation 28. These interpolation procedures are second-order accurate.

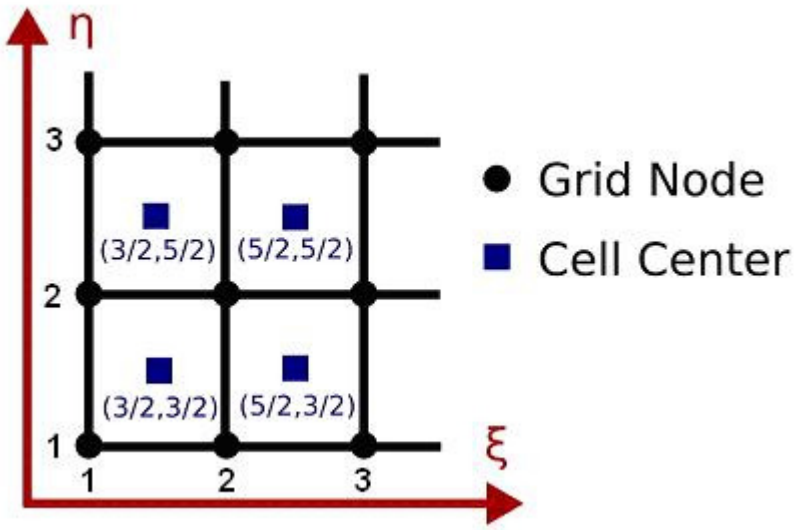


Figure 35: Grid Section for Referencing Node Indices

$$\phi_{2,2} = \frac{\phi_{3/2,3/2} + \phi_{3/2,5/2} + \phi_{5/2,5/2} + \phi_{5/2,3/2}}{4} \tag{26}$$

$$\phi_{1,2} = \frac{\frac{3\phi_{3/2,3/2} - \phi_{5/2,3/2}}{2} + \frac{3\phi_{3/2,5/2} - \phi_{5/2,5/2}}{2}}{2} \tag{27}$$

$$\phi_{1,1} = \frac{3\phi_{3/2,3/2} - \phi_{5/2,5/2}}{2} \tag{28}$$

A comparison of the numerical solution for pressure and the subsequent curve fit are shown in Figure 36. The curvilinear grid used has 129 nodes in both directions. The local least squares fits are 3rd order and merged with C^3 weight functions. There are 32 zones in each direction, for a total of $(32)^2$ pieces to the final fit. Note that all contour plots shown for the Euler equations are plotted on a cell-centered grid.

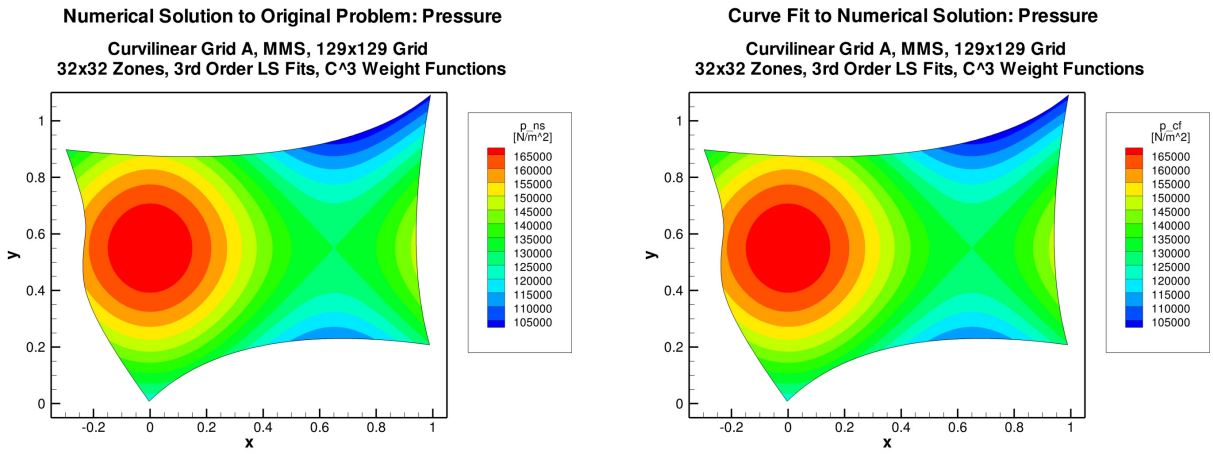
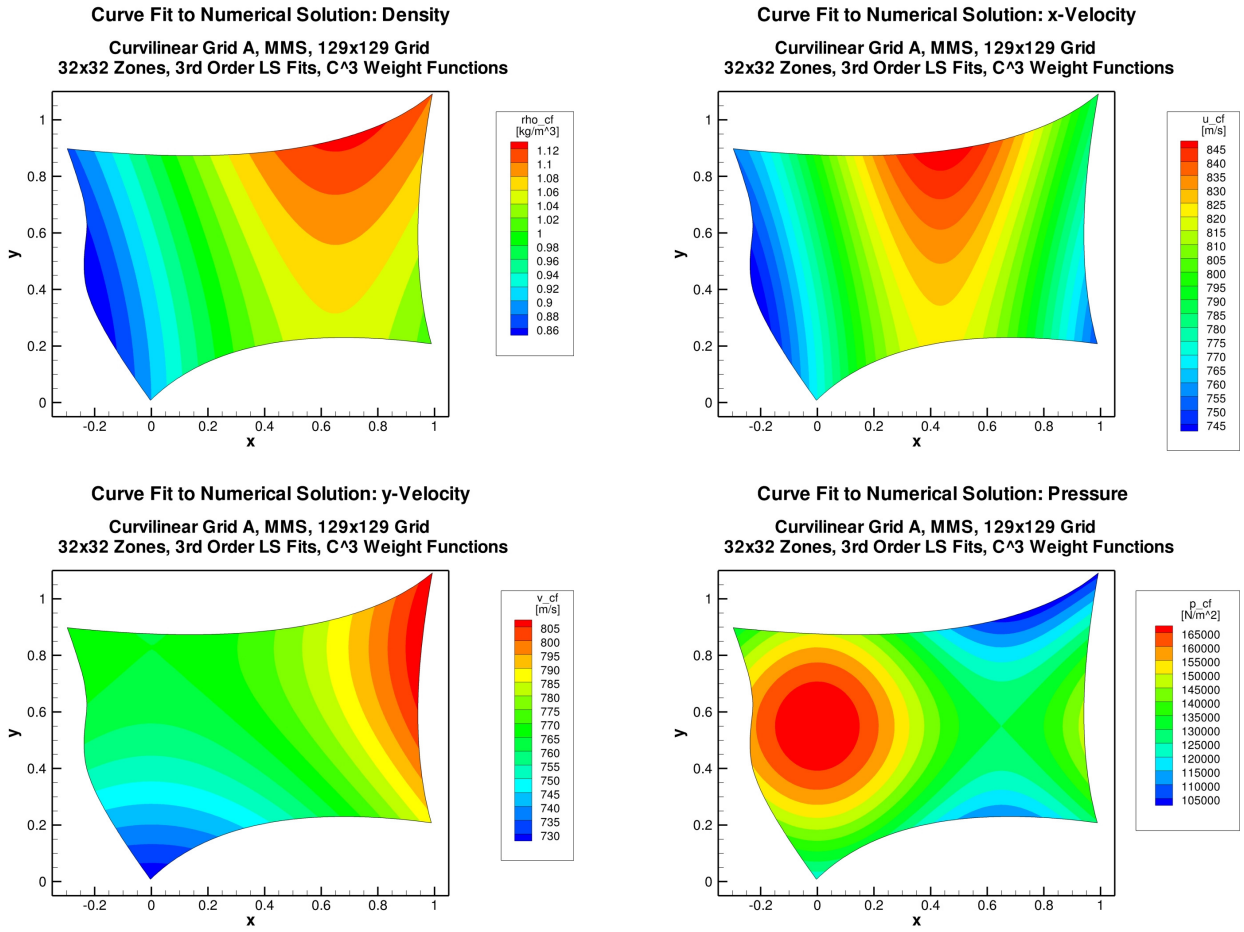


Figure 36: Curvilinear Grid A Numerical Solution and Curve Fit

Figure 37 shows the curve fits for all four primitive variables as well as the total energy quantity.



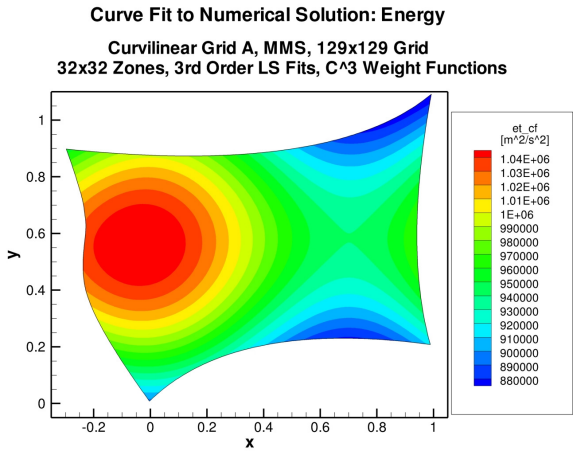
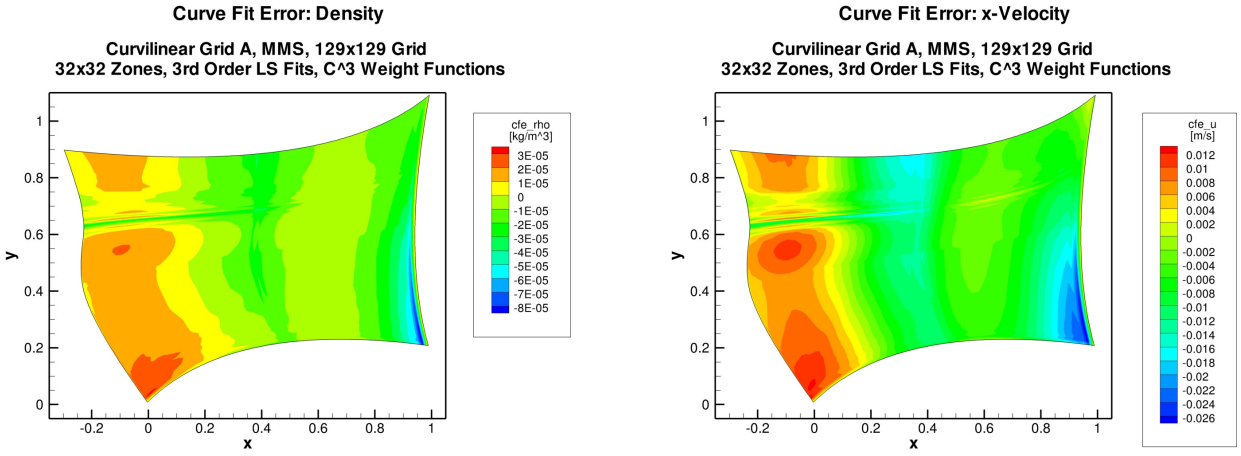


Figure 37: Curvilinear Grid A Curve Fits

The error in the curve fits is defined as the difference between the curve fit solution and the underlying numerical solution. Figure 38 shows the curve fit error for all four primitive variables, as well as the energy term. Naturally the magnitude of the error is proportional to the magnitude of the values fitted (e.g. the pressure curve fit is expected to have more absolute error than the density curve fit). The maximum percentage absolute curve fit error is shown in Table 4 for each primitive variable and the energy term. For this smooth manufactured solution, the curve fit errors are relatively small. The ridges in the curve fit errors can be caused by grid effects such as changes in cell aspect ratios.



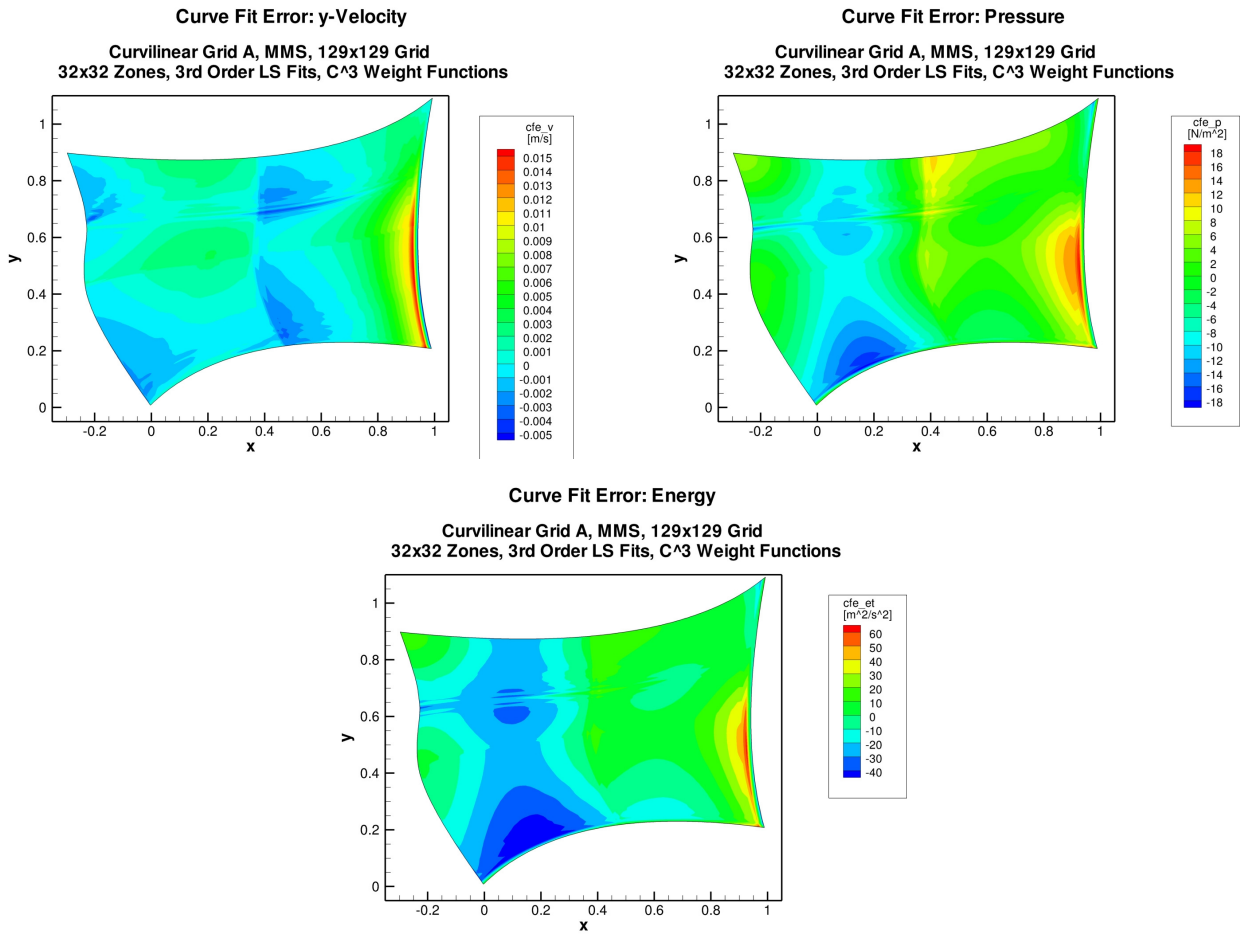


Figure 38: Curvilinear Grid A Curve Fit Error

Table 4: Curvilinear Grid A Maximum Percentage Curve Fit Errors

	ρ	u	v	p	e_t
% CF Error	0.00879	0.00374	0.00204	0.0165	0.00665

In order to introduce another perspective for analyzing the curve fitting procedure, several x-y line plots are presented. The lines were taken at a constant y value of 0.5 m. Figure 39 shows the curve fit of density and the curve fit error as well. The curve fit is smooth as the underlying manufactured solution is composed of sinusoidal terms. The curve fit error is small relative to the magnitude of the underlying solution. One feature of note is the spike in curve fit error at the boundaries of the slice. This is typical near boundaries.

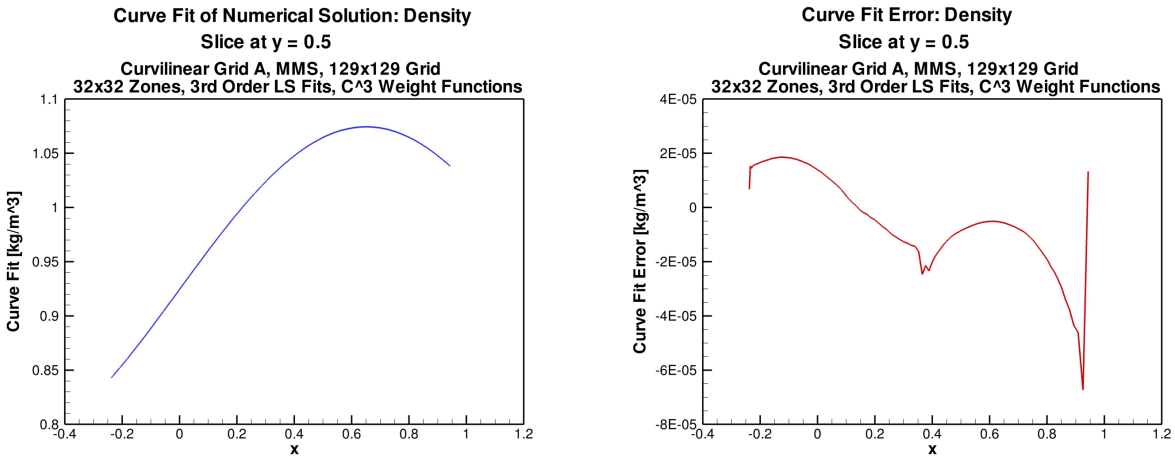


Figure 39: Curvilinear Grid A Slices of Curve Fit and Error

Source terms are generated by operating the continuous governing equations upon the analytical curve fit. As this particular problem already contains source terms, the following definitions are introduced:

- Original Problem Source Term, $S_{\phi,OP}$: the right-hand-side of the original governing equations. This source term will be present for a method of manufactured solutions problem and/or a general problem with a forcing source term.
- Method of Nearby Problems Source Term, $S_{\phi,MNP}$: the difference between the nearby problem source term and the original problem source term (if one exists).

The MNP source terms are defined in Equation 29.

$$\begin{aligned}
&\frac{\partial(\rho u)}{\partial x}+\frac{\partial(\rho v)|_{CF}}{\partial y}-S_{m,OP}=S_{m,MNP}\\
&\frac{\partial\left(\rho u^2+p\right)|_{CF}}{\partial x}+\frac{\partial(\rho uv)|_{CF}}{\partial y}-S_{x,OP}=S_{x,MNP}\\
&\frac{\partial(\rho vu)|_{CF}}{\partial x}+\frac{\partial\left(\rho v^2+p\right)|_{CF}}{\partial y}-S_{y,OP}=S_{y,MNP}\\
&\frac{\partial\left(\rho u e_t+pu\right)|_{CF}}{\partial x}+\frac{\partial\left(\rho v e_t+pv\right)|_{CF}}{\partial y}-S_{e,OP}=S_{e,MNP}
\end{aligned} \tag{29}$$

where the variables have been previously defined and the m, x, y, and e subscripts refer to mass, x-momentum, y-momentum, and energy. Contour plots of the four source terms are shown in Figure 40. The large source term magnitudes within the interior of the domain are often aligned with grid features such as change in cell stretching.

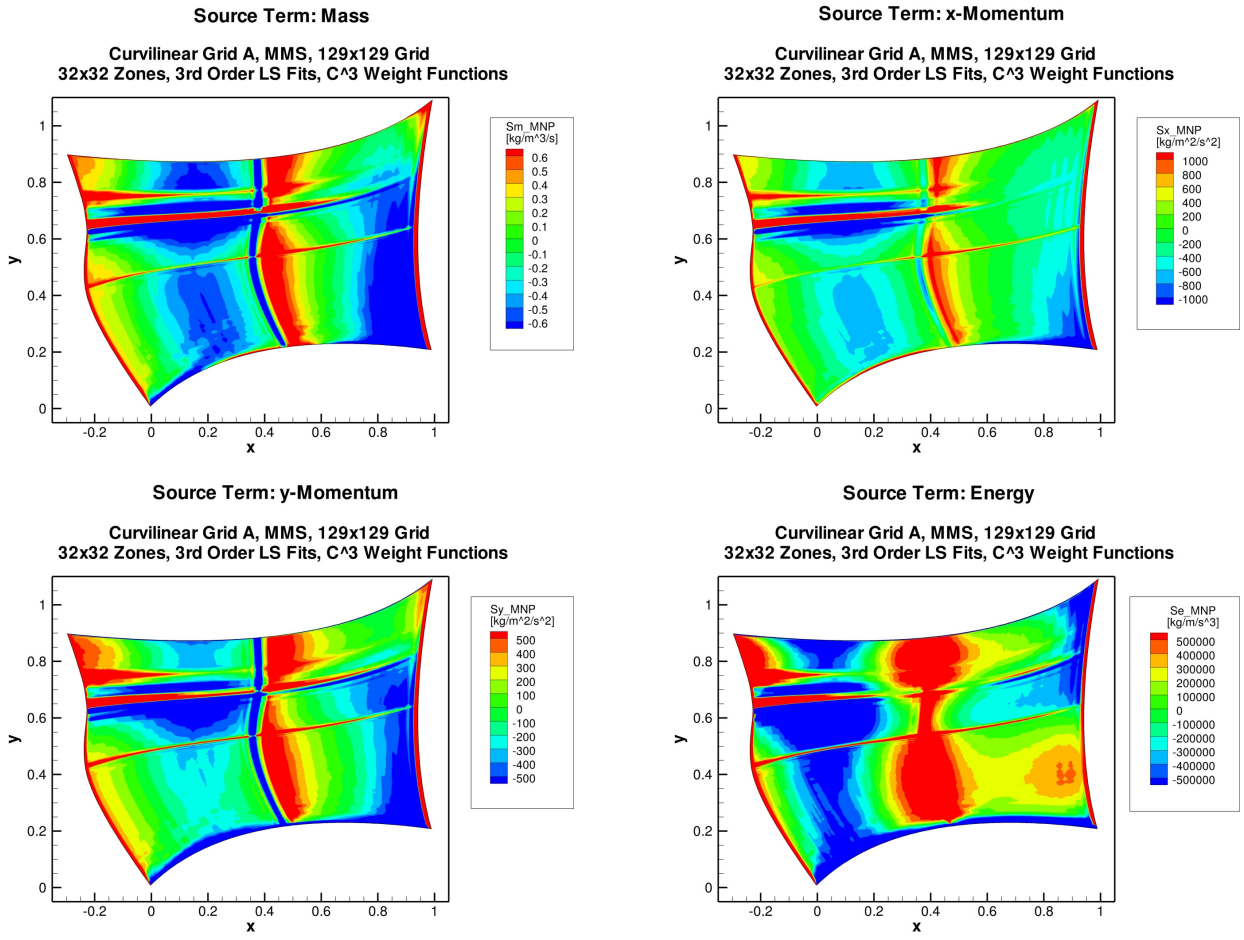


Figure 40: Curvilinear Grid A MNP Source Terms

An x-y plot of the mass source term is shown in Figure 41. The spikes at the boundaries can be present due to an accumulation of errors in the numerical solution and curve fits. The source terms are calculated from derivatives of the curve fit, so small changes in slope of the curve fit can cause this source term (derivative) behavior.

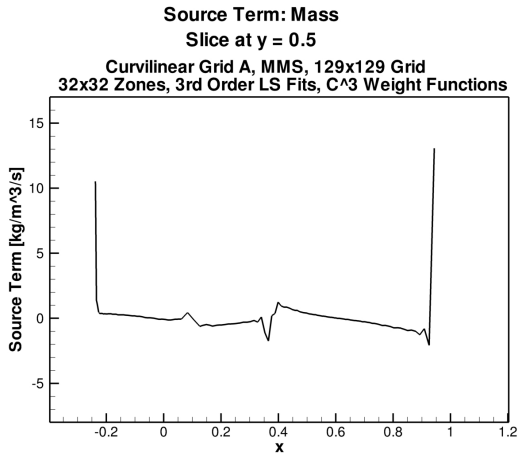
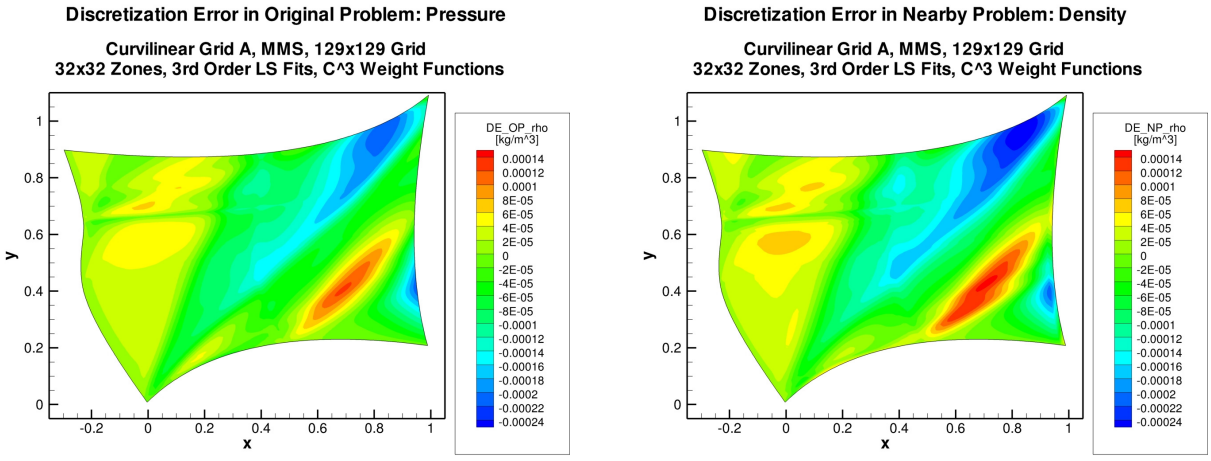


Figure 41: Curvilinear Grid A x-y Line Plot of Mass Source Term

The discretization error in the original problem is calculated as the difference between the numerical solution and the exact solution. The discretization error in the nearby problem is calculated as the difference between the numerical solution to the nearby problem and the curve fit (exact solution for the nearby problem). Plots of these errors are shown in Figure 42. Assuming the nearby problem is truly "nearby", the discretization error in the nearby problem can be used as an estimate of the discretization error in the original problem of interest. The plots are shown side-by-side for comparison and can be seen to match reasonably well. A direct comparison of the discretization errors for the pressure primitive variable can be seen in Figure 43. This is an x-y line plot for a constant y value of 0.5 m. The nearby problem discretization error has most of the features present in the original problem discretization error and tends to also have slightly higher error magnitudes.



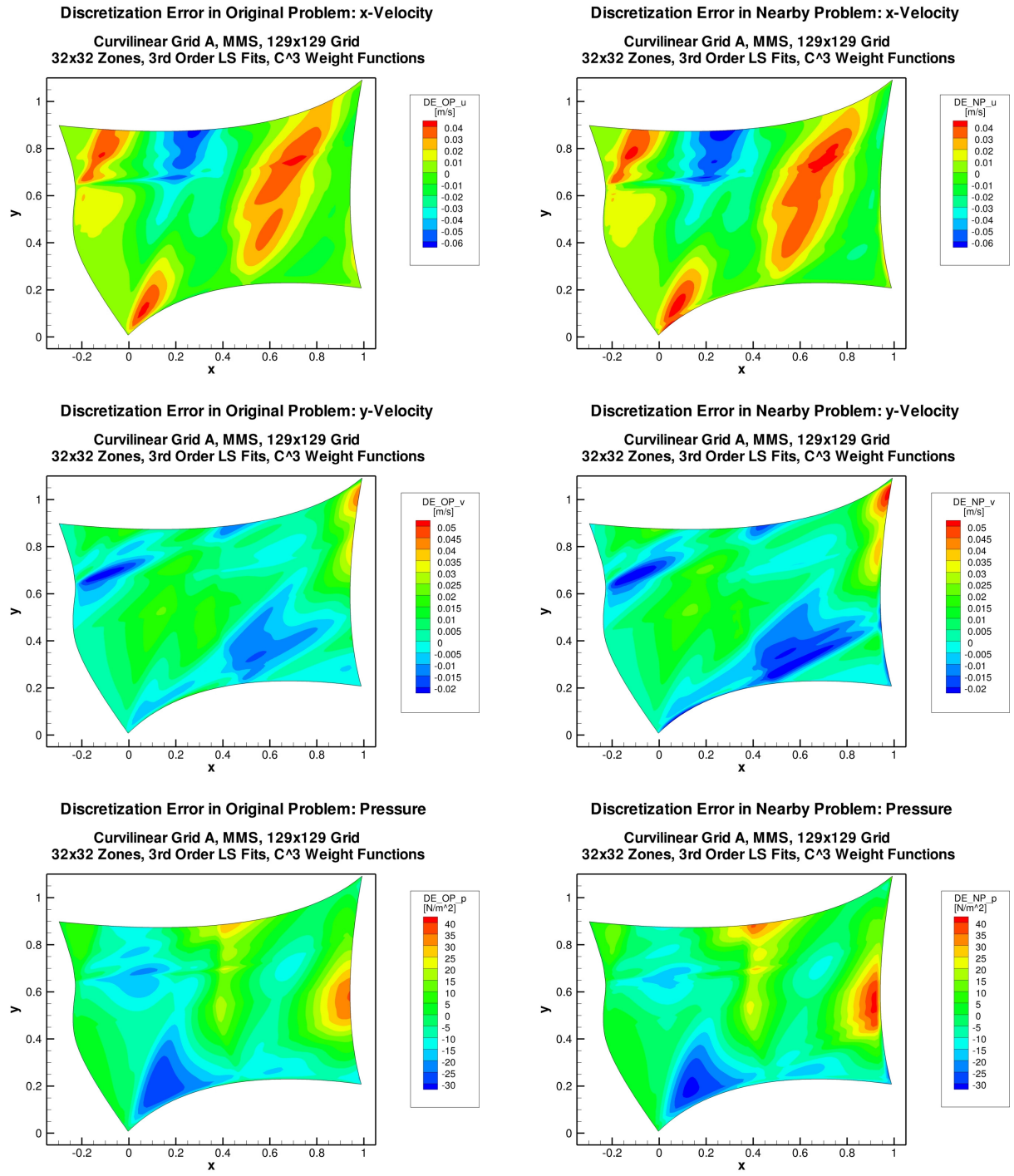


Figure 42: Curvilinear Grid A Comparison of Discretization Errors

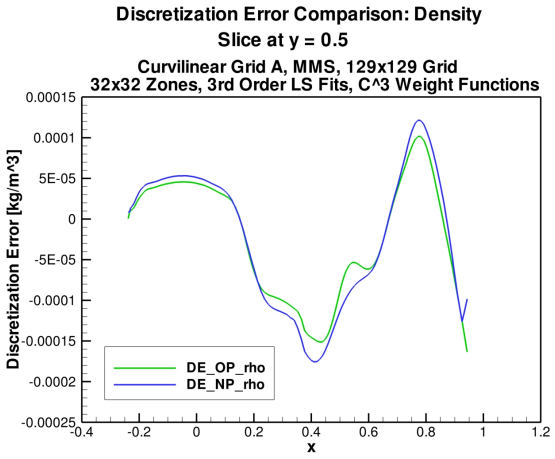


Figure 43: Curvilinear Grid A x-y Line Plot Discretization Error Comparison

The effects of changing the fit parameters have also been investigated. To begin with, a set of three uniformly refined grids were run. The number of zones was varied such that the ratio of grid points to zones was constant. The effect on the mass source term norms and pressure curve fit error norms are plotted in Figure 44. The trends show that by refining the points and zones, the norms drop at a second order rate when plotted on a log-log scale.

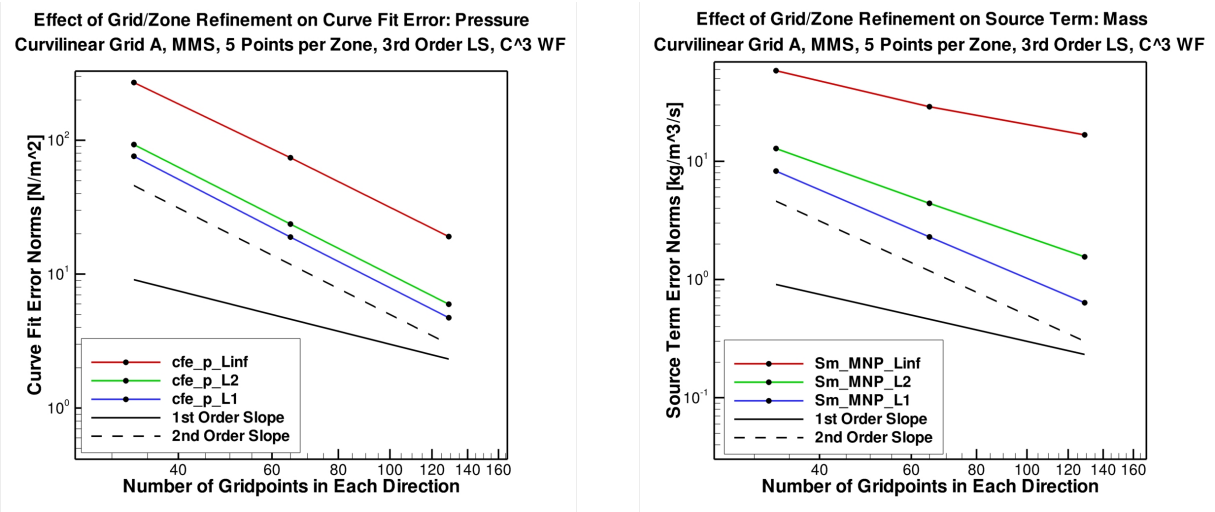


Figure 44: Curvilinear Grid A Effect of Grid/Zone Refinement

The next study involves running on the fine grid (129x129) while changing the number of zones used for the curve fit. These results are shown in Figure 45. It can be seen that by using more zones, the pressure curve fit error drops. Two mass source term norms decrease slightly

while the L_∞ norm actually increases. The trends are also plotted on a log-log scale, but do not show the same constant rate decrease as is the case when both grid points and zones are refined simultaneously. One thing to consider is that as the number of zones is increased, the zones cover less of the domain and are therefore less likely to have complex solution features contained within a single zone. This means that continuing to increase the number of zones used will eventually produce diminishing returns. This may explain some of the behavior seen here.

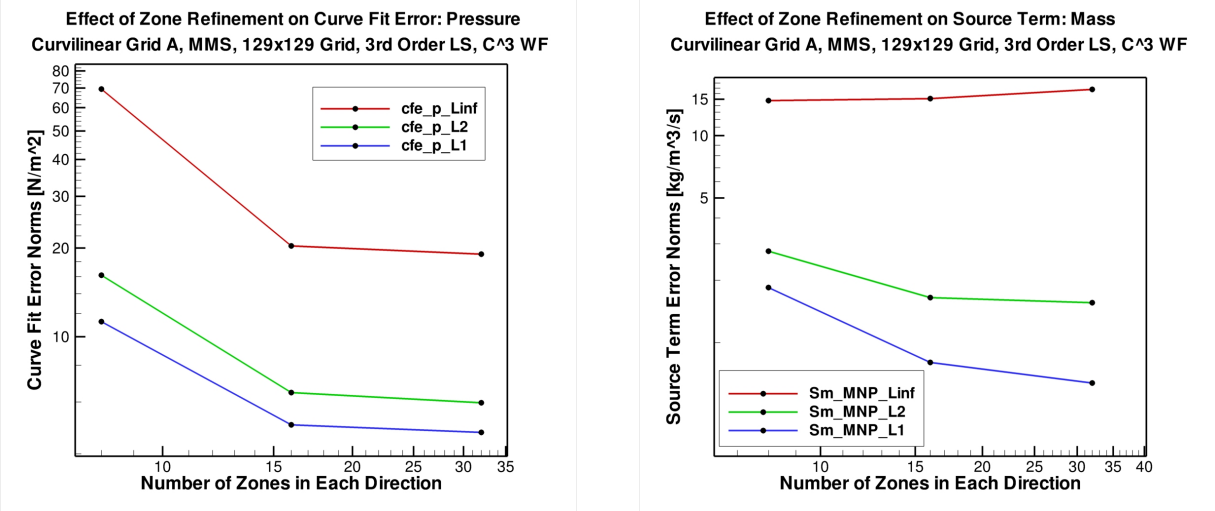


Figure 45: Curvilinear Grid A Effect of Zone Refinement

Another parameter that can be varied is the order of the least squares fits. These were varied from first to third order, in both x and y directions. As seen in Figure 46, increasing the least squares fit order decreases the curve fit error and source term norms. However, the largest decrease occurs when going from first order to second order. Again, for a fine grid there should be a limited number of zones with complex behavior, so increasing the order of the least squares fits will have diminishing returns.

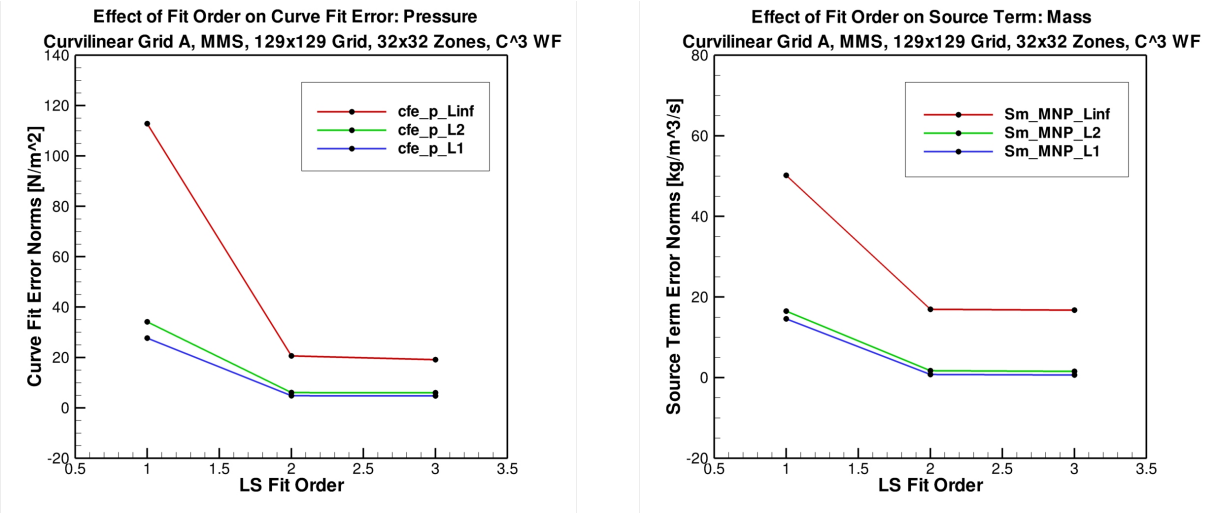


Figure 46: Curvilinear Grid A Effect of Changing LS Fit Order

5.2.2 Curvilinear Manufactured Solution on Grid B

A second curvilinear grid was created to investigate the effect of grid quality on the MNP problem. The second grid, Curvilinear Grid B, covers a similar domain in physical space, but has less stretching, skewing and curvature than Curvilinear Grid A. The new grid is pictured in Figure 47.

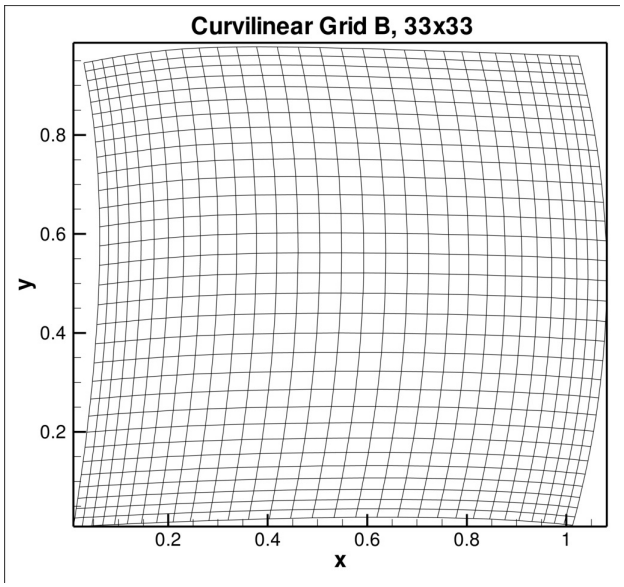


Figure 47: Curvilinear Grid B

Figure 48 shows a comparison of the numerical solution and curve fit for pressure. The curve fits for all primitive variables are shown in Figure 49. As for the Grid A case, the results are shown for a 129x129 grid, with 32 zones in each direction, 3rd order least squares fits, and C^3 weight functions. The curve fits do a good job of matching the underlying numerical solution. The curve fit errors are presented in Figure 50 with percent error shown in Table 5. The curve fit errors are smooth which is expected as this is the nature of the numerical solution and curve fits. The percent curve fit errors are less than those for Grid A, which is evidence that grid quality affects the curve fit accuracy.

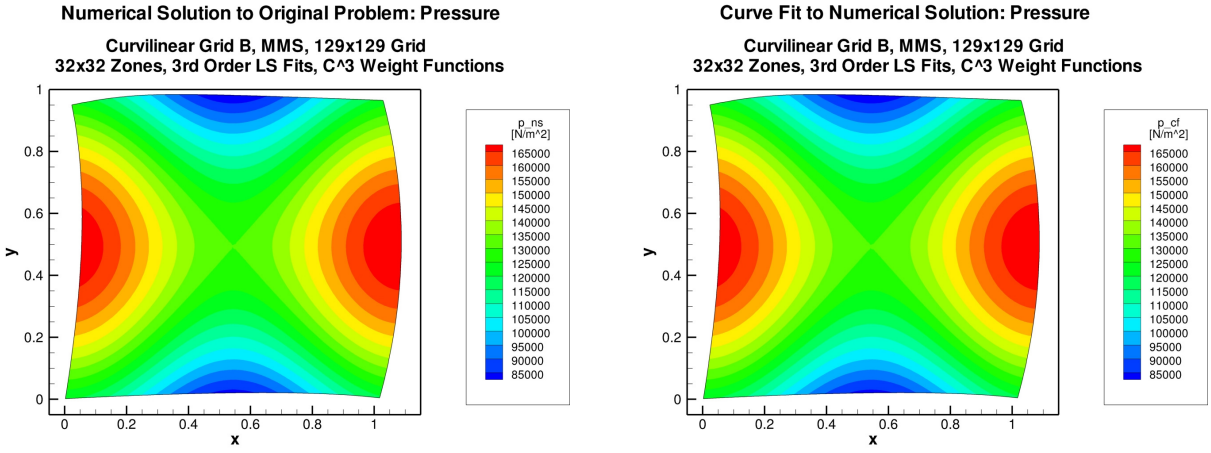
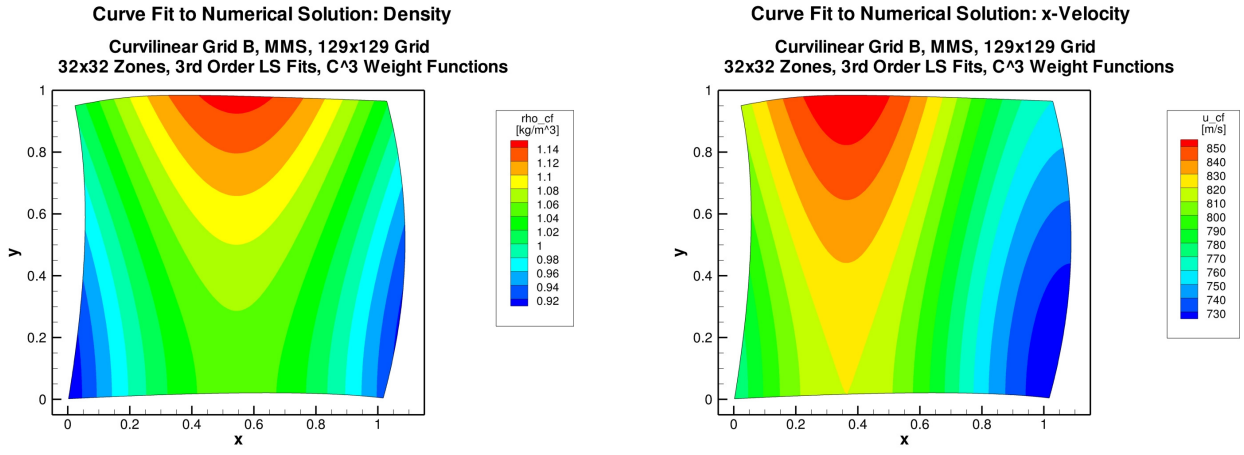


Figure 48: Curvilinear Grid B Numerical Solution and Curve Fit



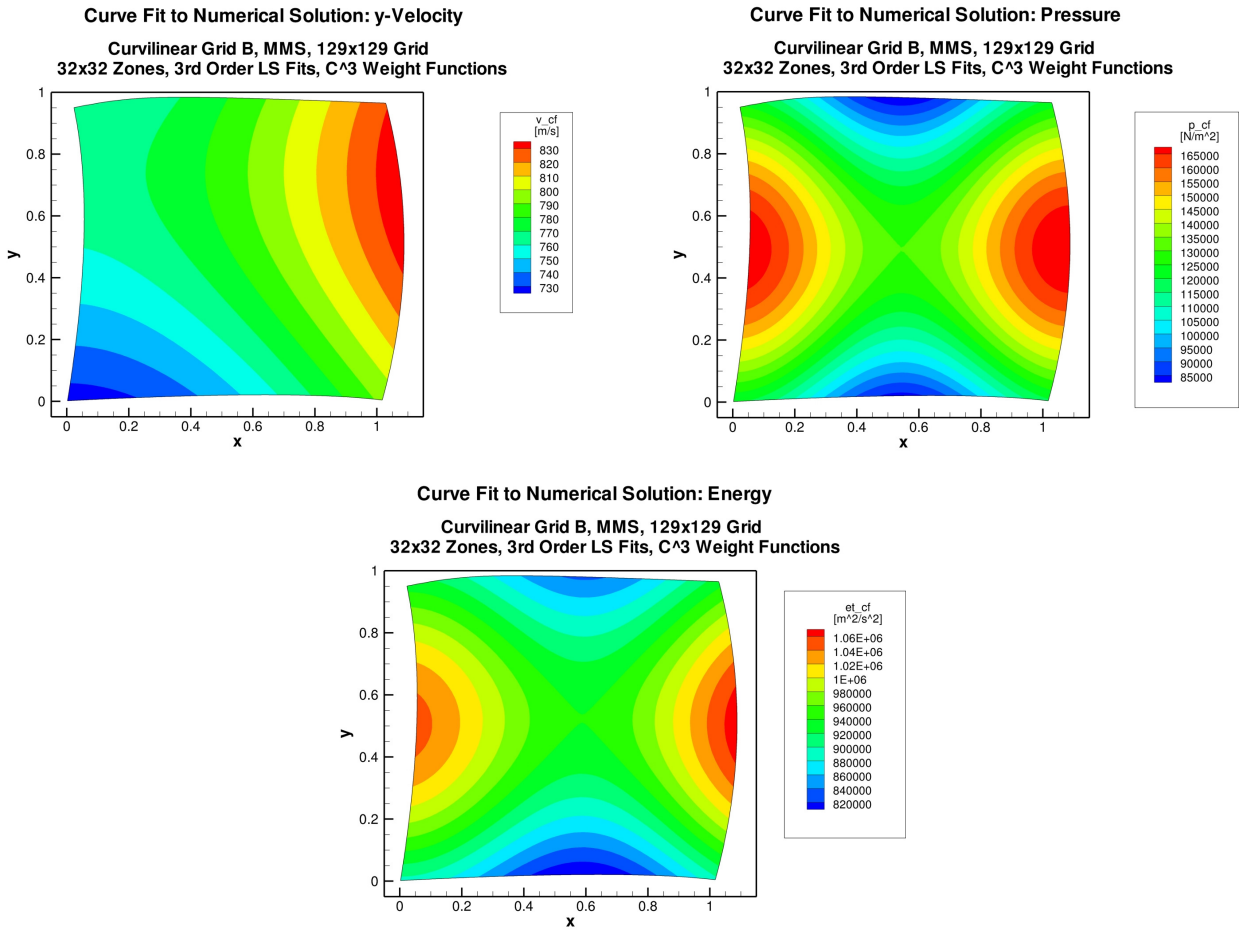
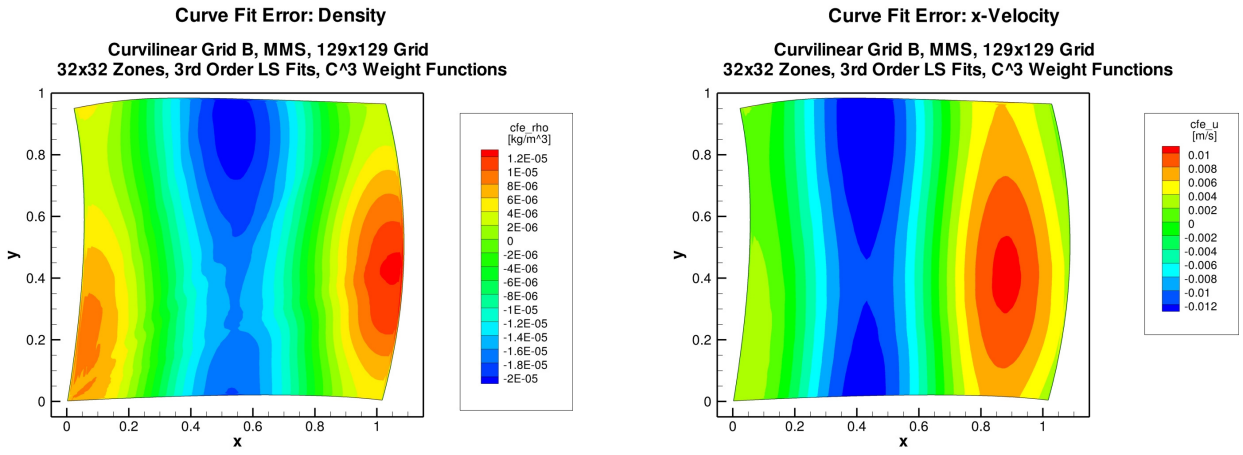


Figure 49: Curvilinear Grid B Curve Fits



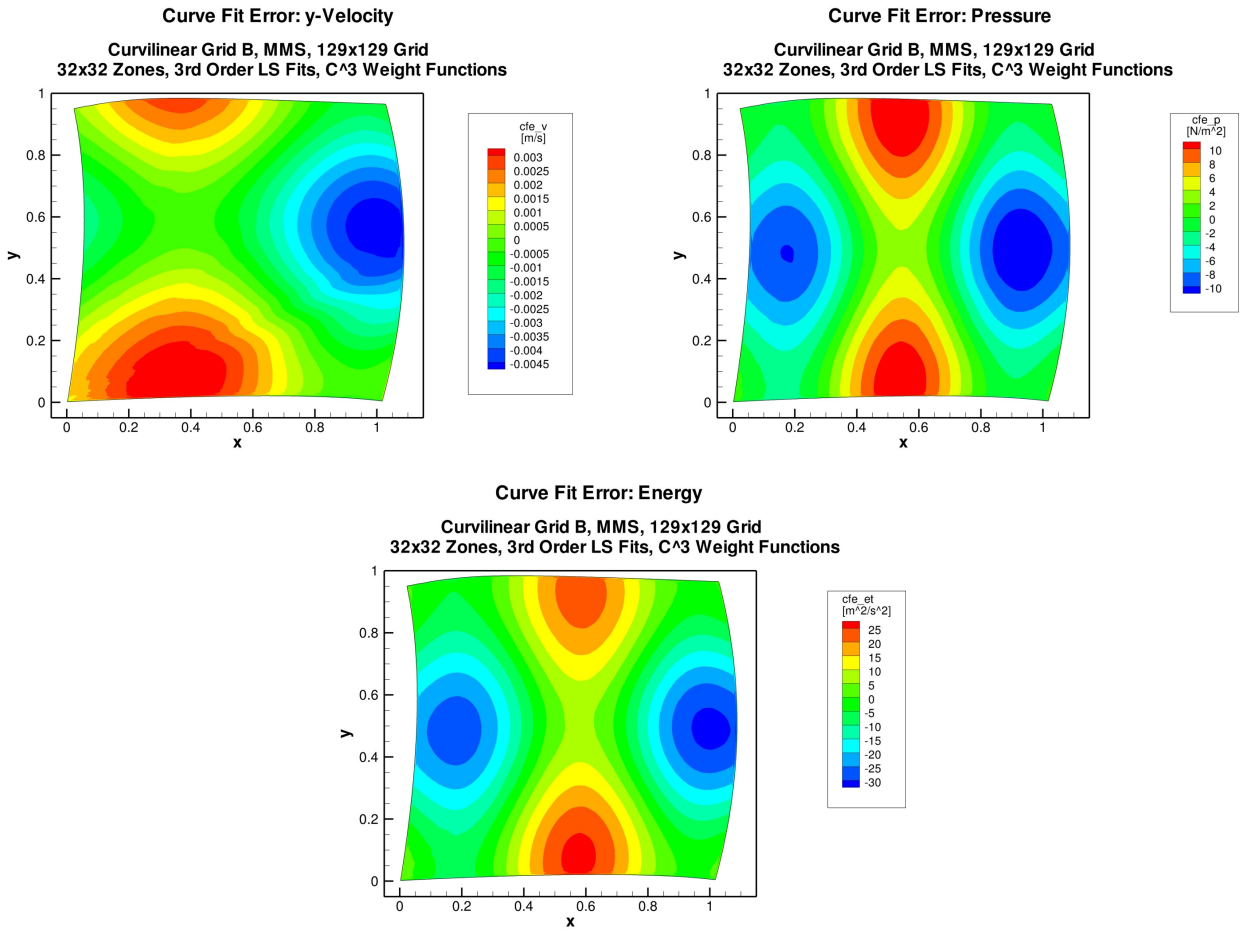


Figure 50: Curvilinear Grid B Curve Fit Error

Table 5: Curvilinear Grid B Maximum Percentage Curve Fit Errors

	ρ	u	v	p	e_t
% CF Error	0.00187	0.00167	0.000586	0.0152	0.00325

X-y plots of the curve fit and curve fit error of density are shown in Figure 51 for a slice at $y = 0.5$ m. There is a spike in the curve fit error at both boundaries, which indicates that even for a smooth solution, the boundaries of the problem are a more difficult region to fit.

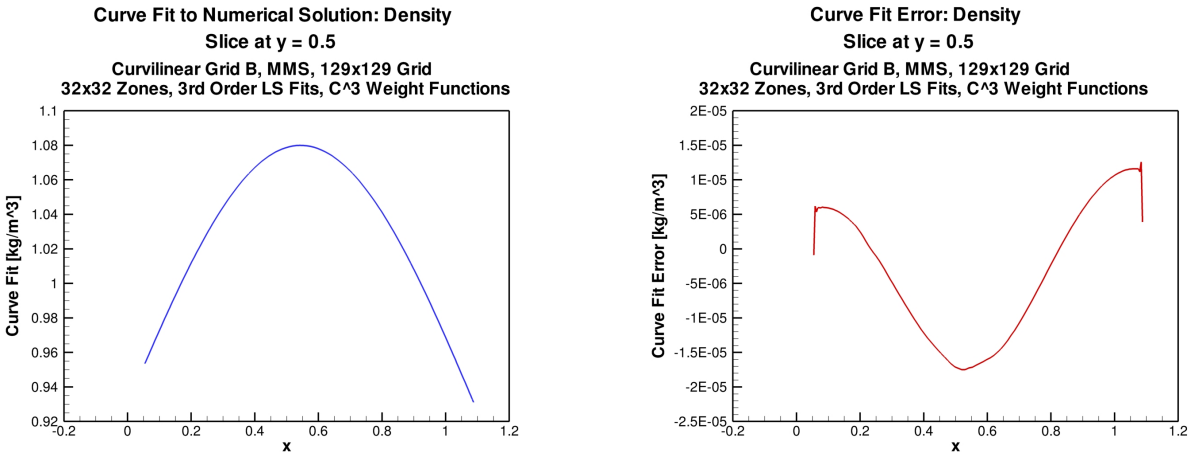
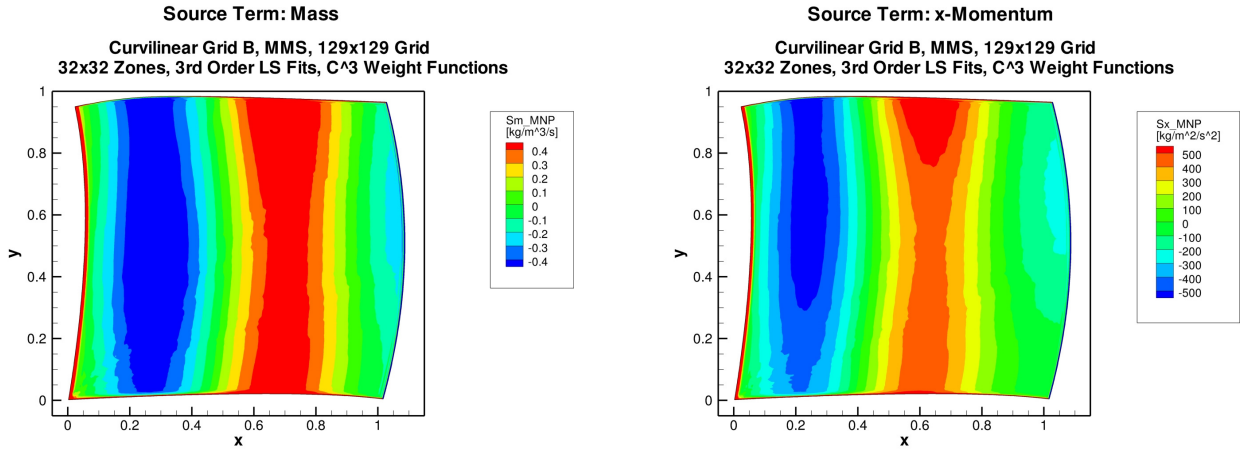


Figure 51: Curvilinear Grid B Slices of Curve Fit and Error

The source terms in Figure 52 exhibit the greatest magnitude near the boundaries. Figure 53 is an x-y plot of the mass source term, at a slice through the domain. The spike in source term magnitude at the boundaries can be seen clearly. This is related to the curve fit error at the boundaries.



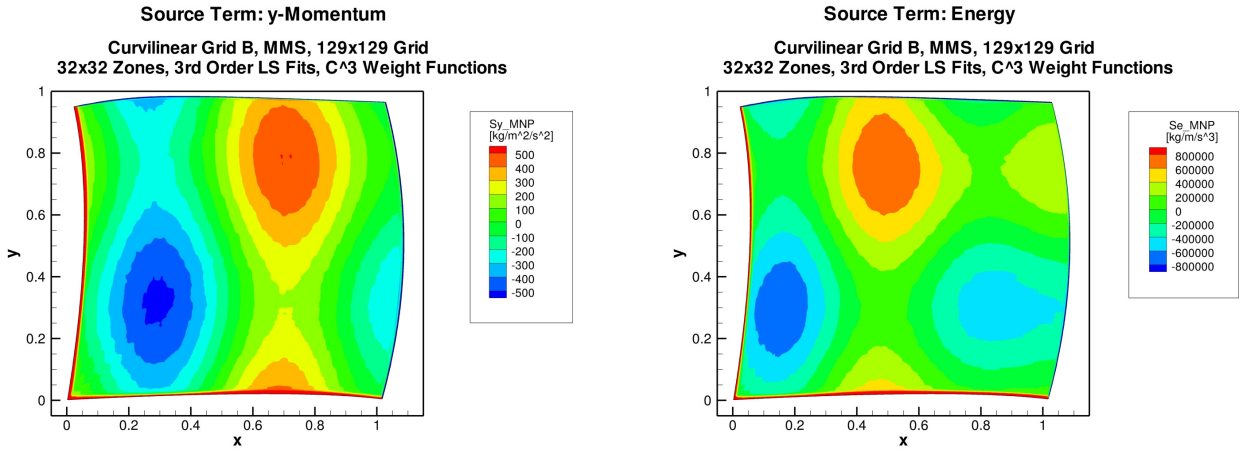


Figure 52: Curvilinear Grid B MNP Source Terms

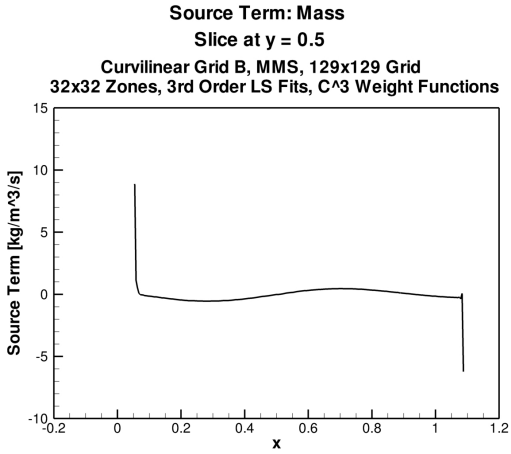
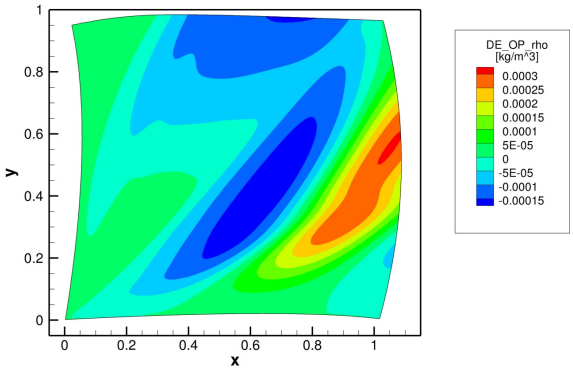


Figure 53: Curvilinear Grid B Slice of Mass Source Term

A comparison between original problem discretization error and nearby problem discretization error is given in Figure 54. The nearby problem discretization error provides a good estimate of the error in the original problem. A slice comparison of the discretization error is provided in Figure 55.

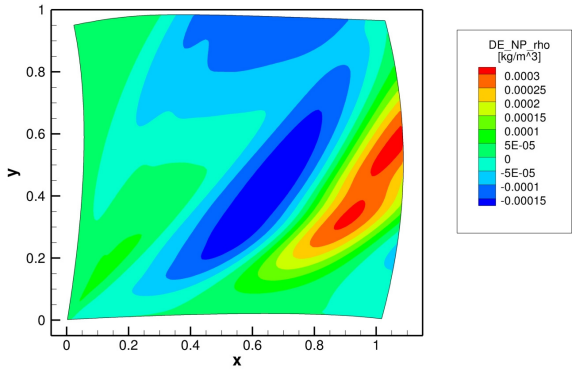
Discretization Error in Original Problem: Density

Curvilinear Grid B, MMS, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions



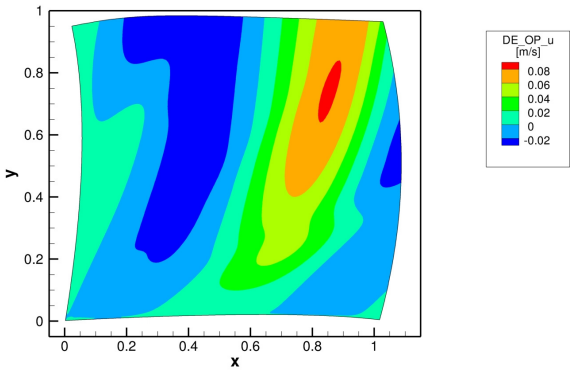
Discretization Error in Nearby Problem: Density

Curvilinear Grid B, MMS, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions



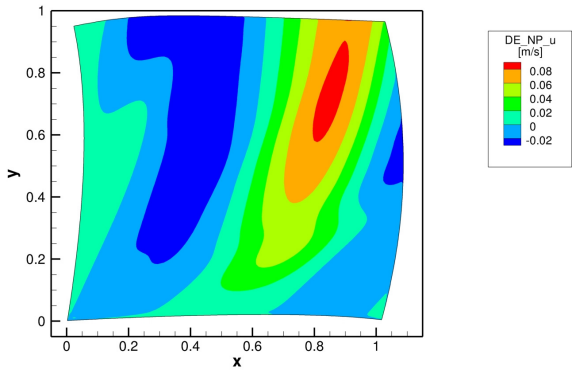
Discretization Error in Original Problem: x-Velocity

Curvilinear Grid B, MMS, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions



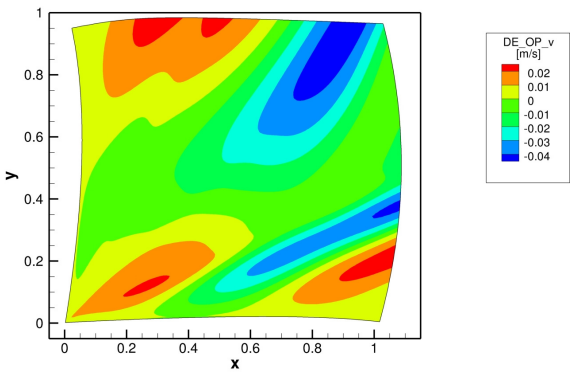
Discretization Error in Nearby Problem: x-Velocity

Curvilinear Grid B, MMS, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions



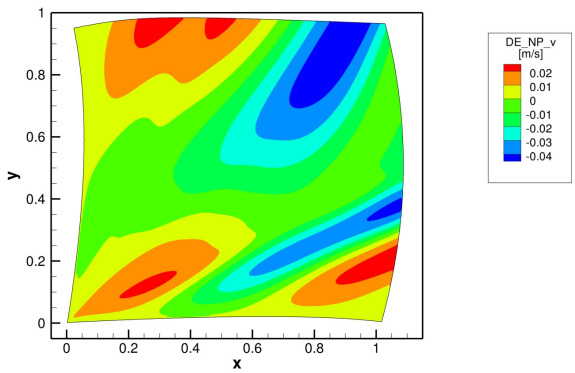
Discretization Error in Original Problem: y-Velocity

Curvilinear Grid B, MMS, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions



Discretization Error in Nearby Problem: y-Velocity

Curvilinear Grid B, MMS, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions



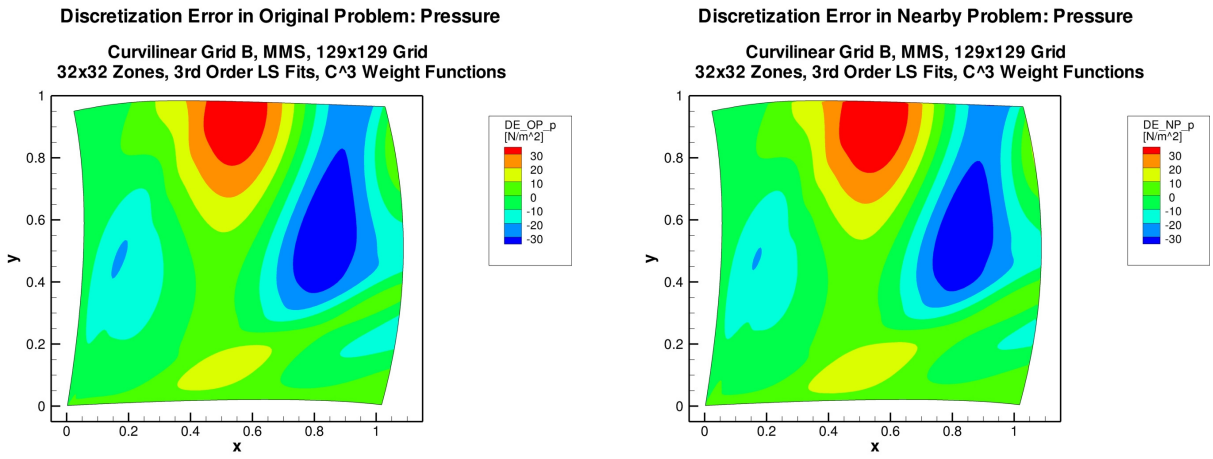


Figure 54: Curvilinear Grid B Discretization Error Comparison

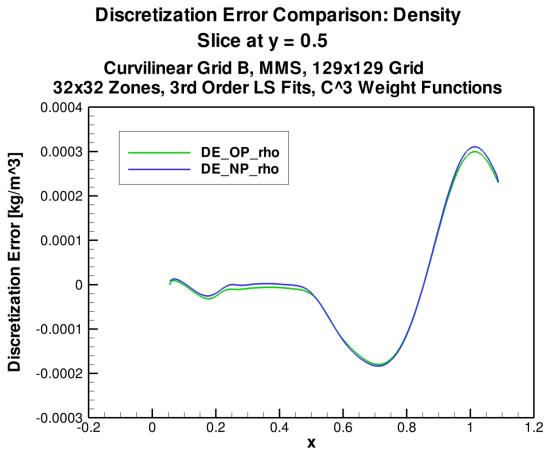


Figure 55: Curvilinear Grid B Slice Discretization Error Comparison

The same trends examined in the previous section are also presented here. Both the previous results for Grid A and the new results for Grid B are shown for comparison. The effect of refining the number of grid points and zones is shown in Figure 56. As both grid points and zones are refined, the curve fit error and source term both drop in magnitude at a constant rate. The effect of changing only the number of zones while holding grid points constant is given in Figure 57. The curve fit error drops as the number of zones is increased, but shows the biggest drop between 8 zones and 16 zones. The source term norms give a conflicting picture of the effect of zone refinement. It appears that increasing the number of zones does not significantly decrease the source term magnitude. Even with a small number of zones, the problem may be smooth enough that using more zones simply is not necessary. Figure 58 shows the effects of

changing the order of the local least squares fits. Increasing the order of the least squares fits causes the curve fit error norms and source term norms to drop, with the greatest reduction between first and second order fits. Again, with sufficiently small zones the numerical solution will approach a linear behavior over each zone, where a higher order least-squares fit may not improve the accuracy of the fit. One important observation to note is that for most of the trends observed, the norms for Grid B are smaller than for Grid A. This indicates that grid quality has an effect on the curve fits and source terms for the MNP.

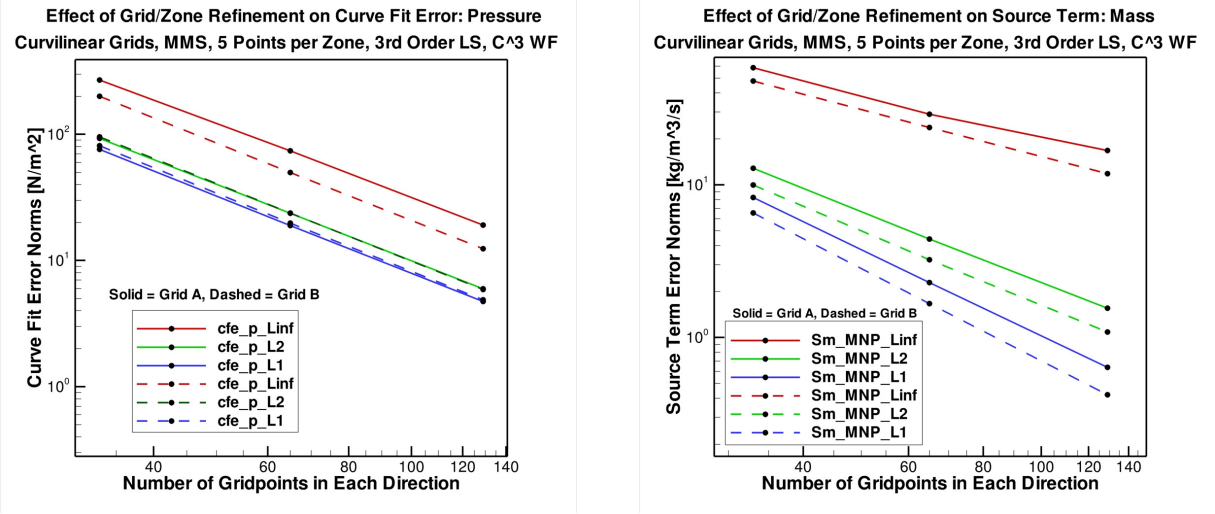


Figure 56: Curvilinear Grid B Effect of Grid/Zone Refinement

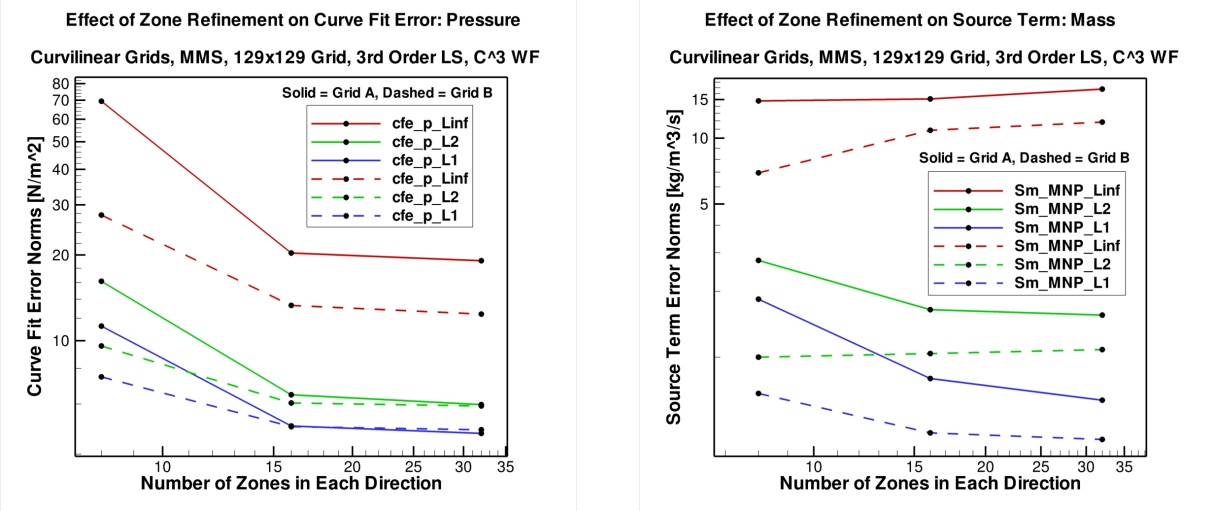


Figure 57: Curvilinear Grid B Effect of Zone Refinement Only

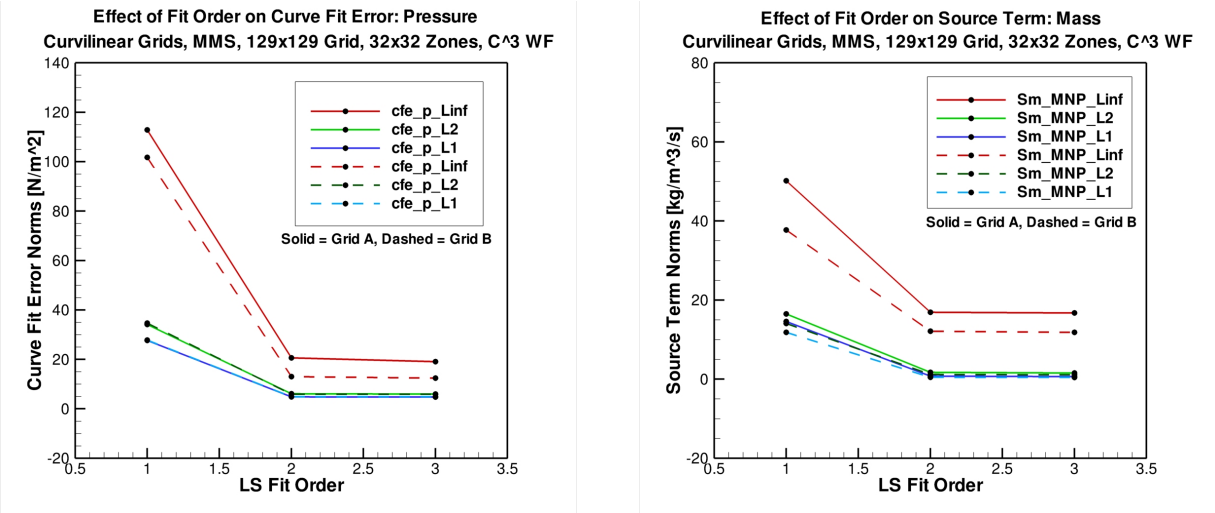


Figure 58: Curvilinear Grid B Effect of Least Squares Fit Order

5.2.3 Euler Equations: Ringleb's Flow

Ringleb's flow is an exact solution to the Euler equations. The solution represents a 180 degree turn in a freestream flow. The flow has both subsonic and supersonic regions as shown in Figure 59. As the flow makes the turn, the flow becomes supersonic near the inner radius. For this case, the numerical solution is solved for a region that is entirely supersonic. For more details on the exact solution to Ringleb's flow, please see Reference 14.

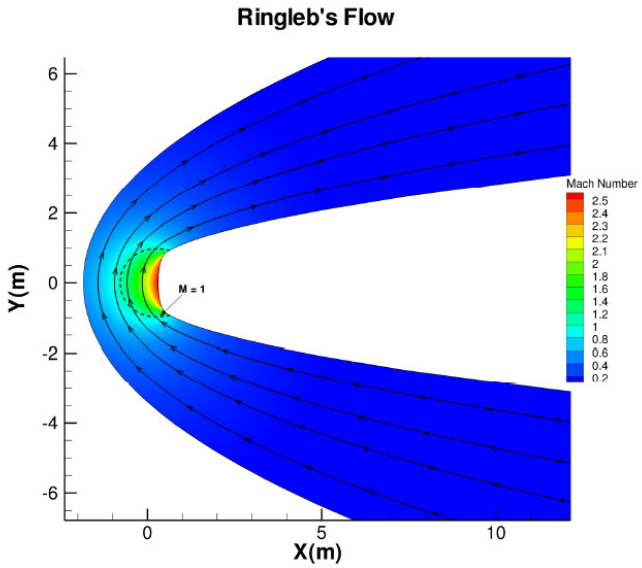


Figure 59: Ringleb's Flow Overview
[Reproduced from Kurzen, et al.[14]]

The grid used to solve the problem is shown in Figure 60. The grid covers a portion of the supersonic flow domain. The left and right boundaries are streamlines for the exact solution and are treated as slip walls. The lower boundary is a supersonic inlet and the top boundary is a supersonic outlet.

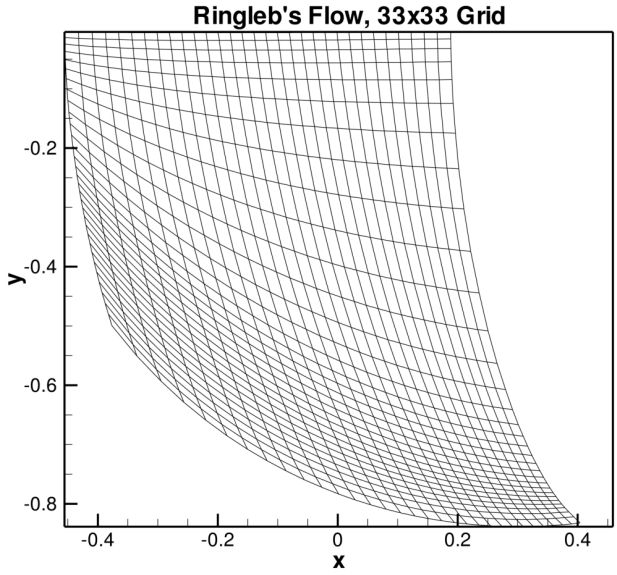


Figure 60: Ringleb's Flow Grid

A comparison of the numerical solution and curve fit for pressure is shown in Figure 61. These plots show good agreement. All of the curve fits are shown in Figure 62. The lower left corner of the domain has some strong gradients due to the flow accelerating around the corner, which makes it a difficult area to fit.

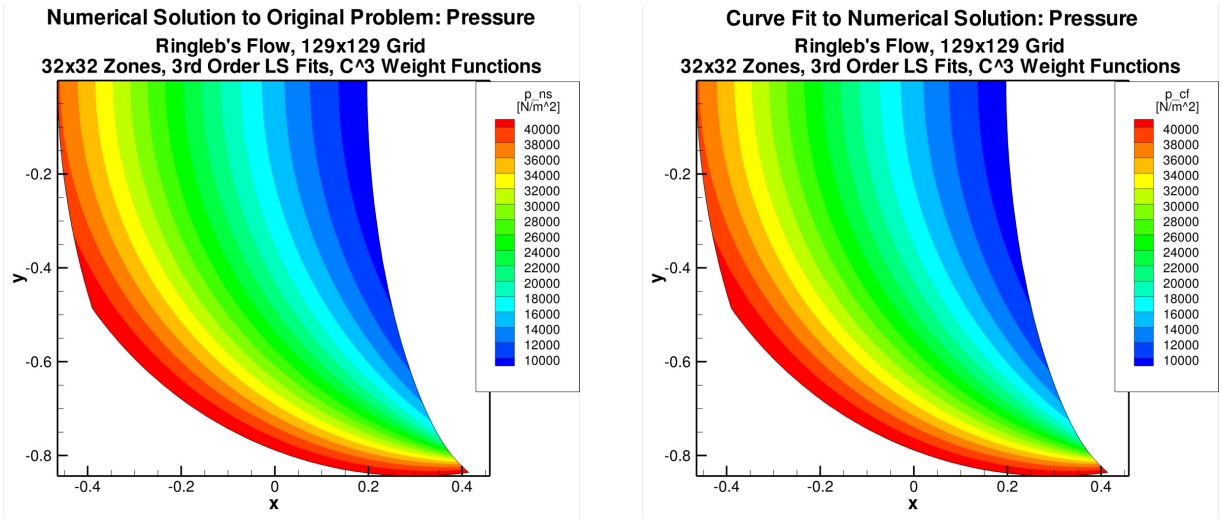
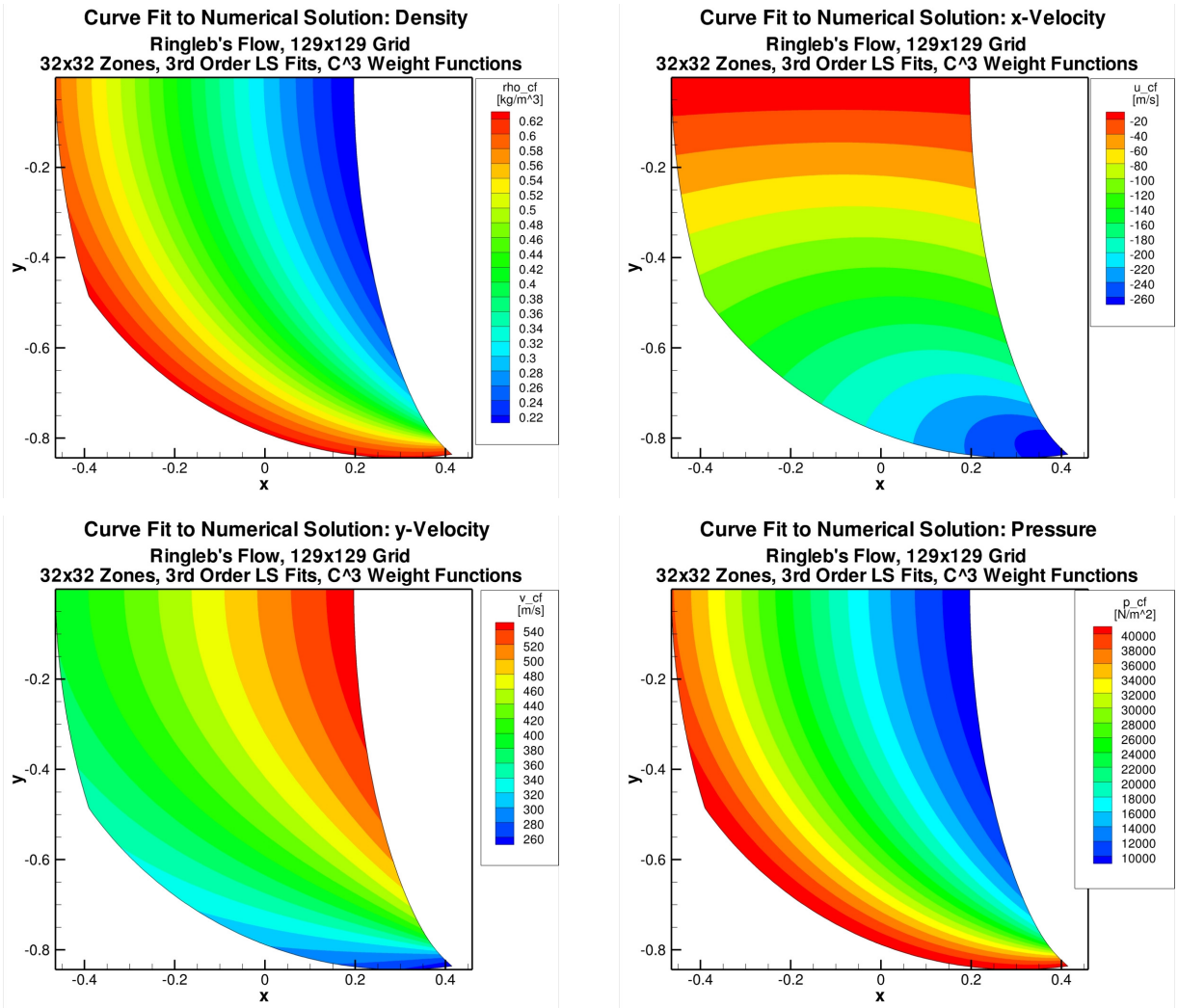


Figure 61: Ringleb's Flow Numerical Solution and Curve Fit



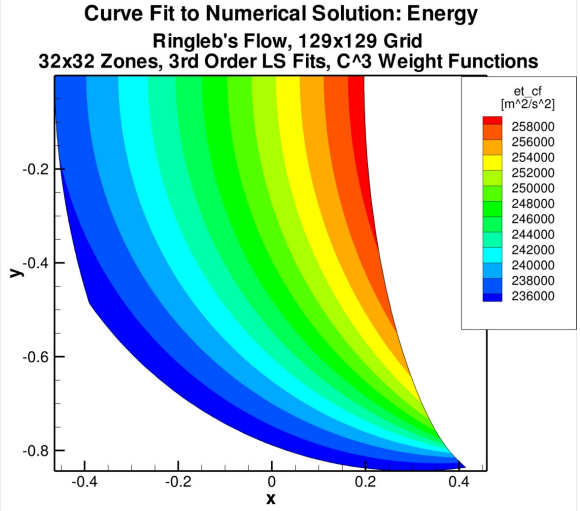
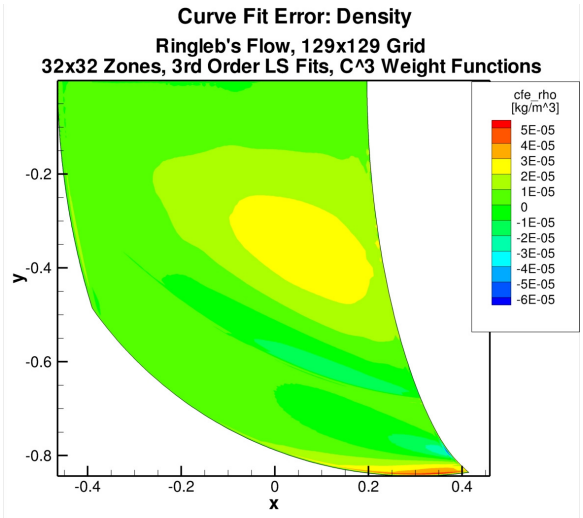
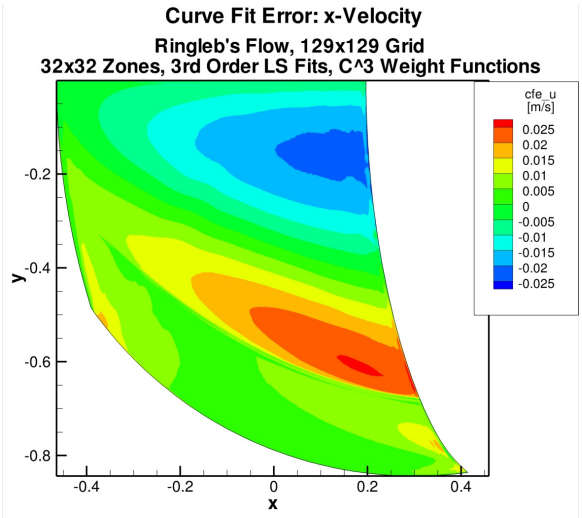


Figure 62: Ringleb's Flow Curve Fits

The curve fit errors are given in Figure 63. Some of the error is clustered near the inlet and also along the right-hand wall where the flow is rapidly changing. The errors are relatively small compared to the quantities being fitted, as shown through the percent curve fit errors in Table 6. The x-velocity (u) curve fit has a much larger percent error than the other variables because there is a location where the x-velocity passes through zero.





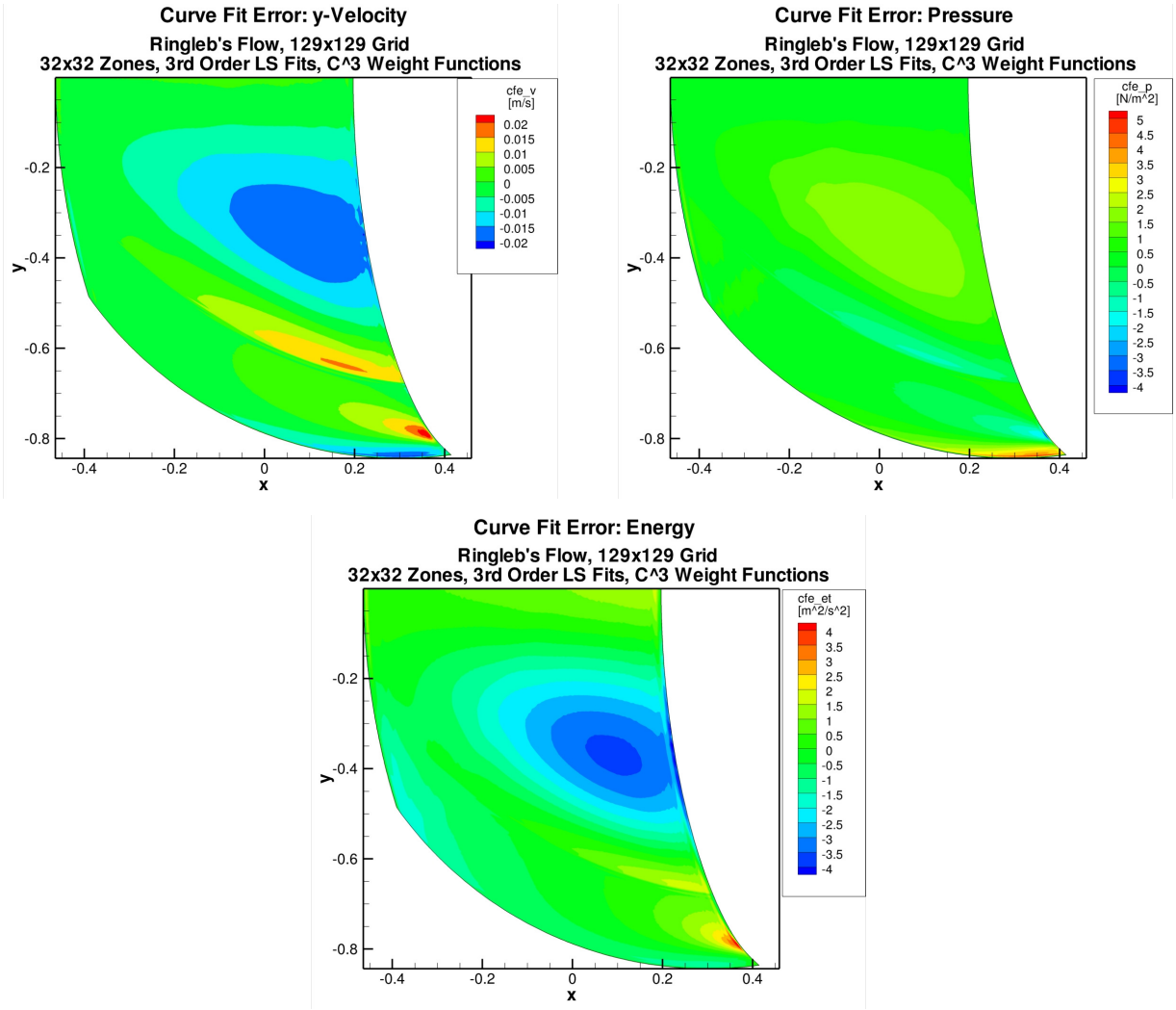


Figure 63: Ringleb's Flow Curve Fit Error

Table 6: Ringleb's Flow Maximum Percentage Curve Fit Errors

	ρ	u	v	p	e_t
% CF Error	0.0138	1.57	0.00835	0.0177	0.00184

The source terms for Ringleb's flow are shown in Figure 64. The source terms are largest at the boundaries and also along one curved path in the interior of the domain. This line corresponds to a change is the grid spacing, which affects the curve fit and error. This was seen before in the curvilinear MMS case with Grid A.

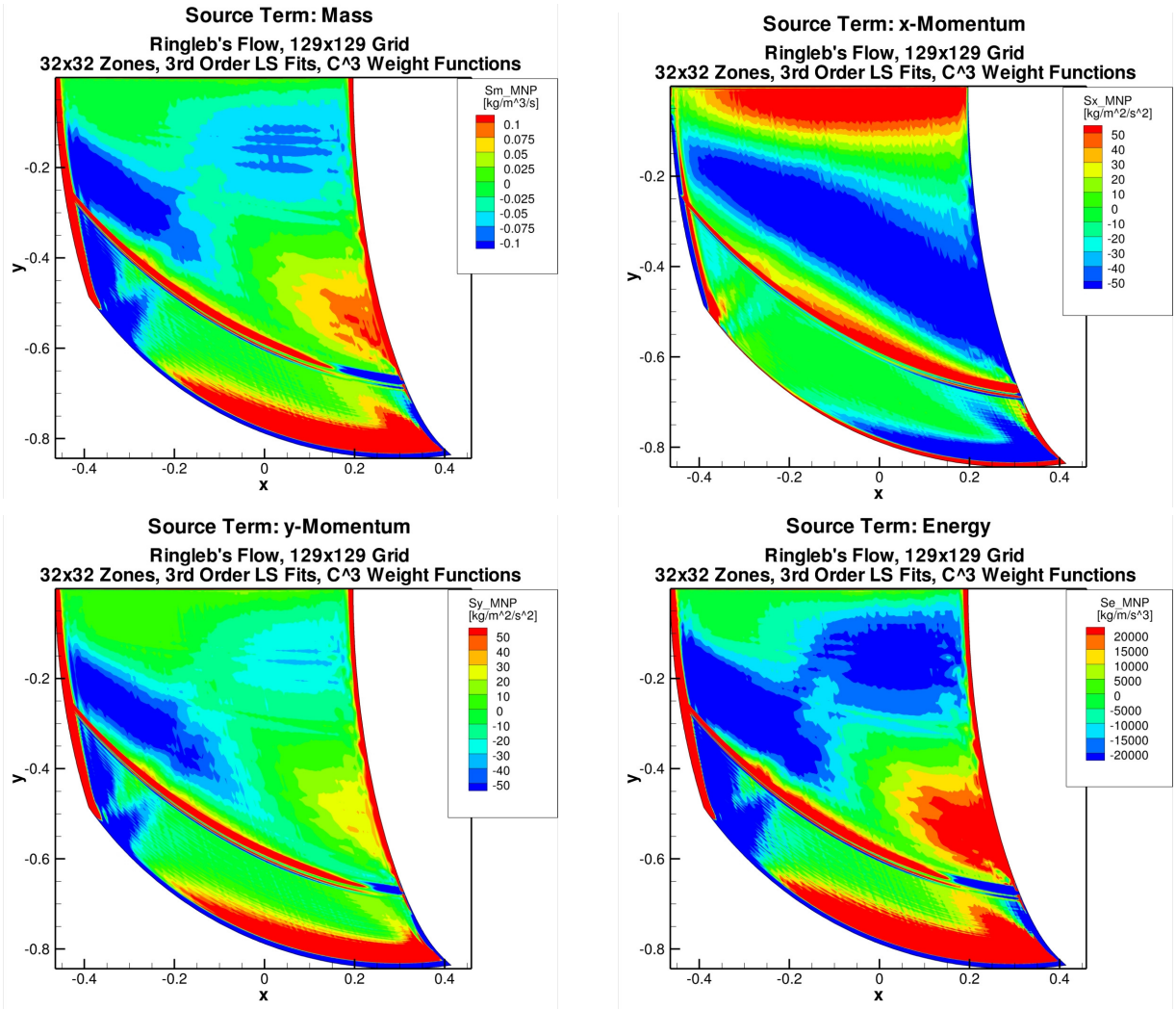
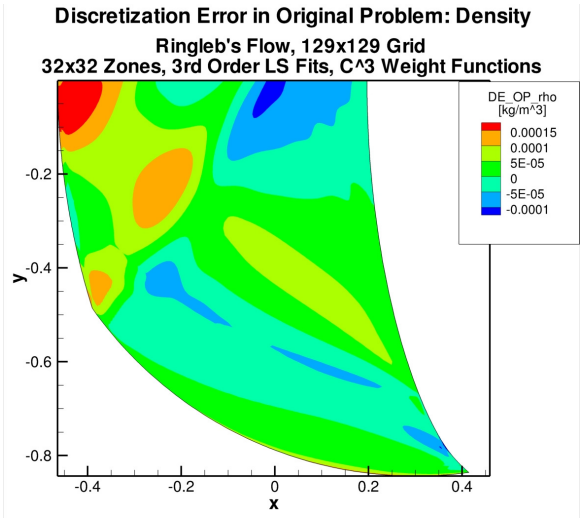
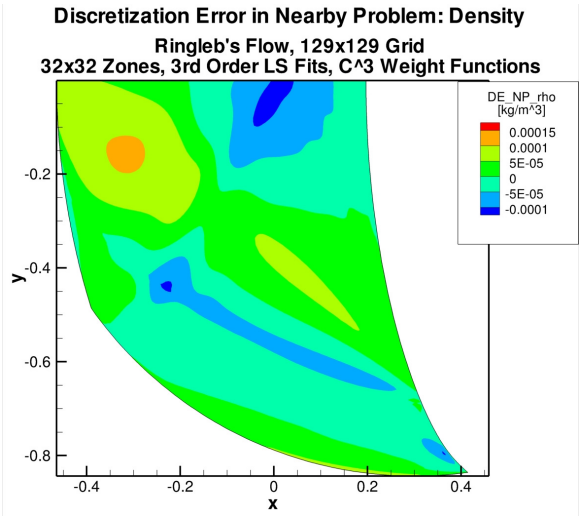
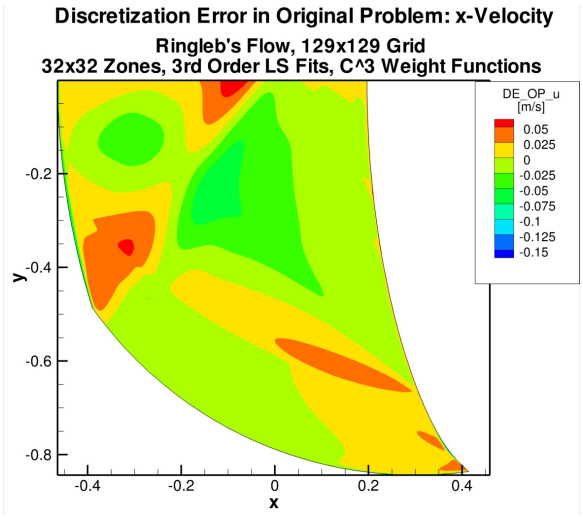


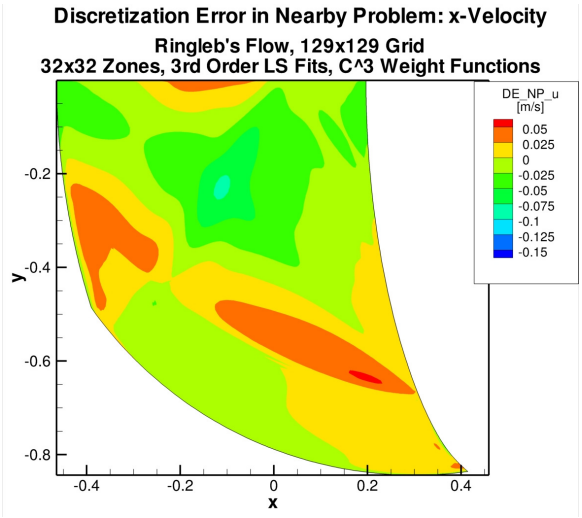
Figure 64: Ringleb's Flow MNP Source Terms

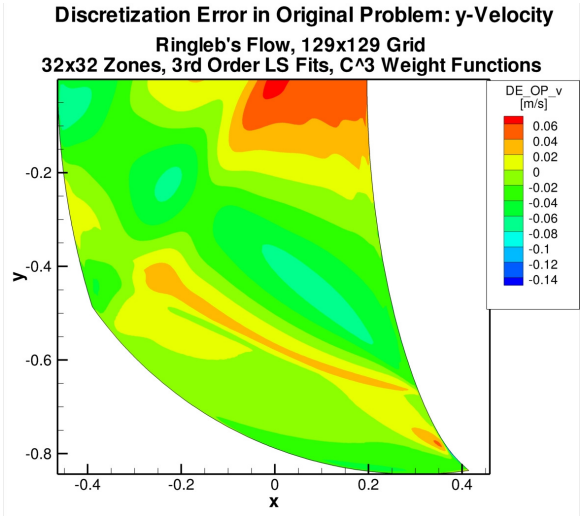
A comparison of the discretization error in the original problem and the nearby problem is presented in Figure 65. In general the two discretization errors are similar, but there are some qualitative differences, especially in the upper left hand corner. It should be noted that the nearby problem was run with nearby boundary conditions. Dirchlet boundary conditions were used to set the outside state at all boundaries. This differs from the boundary conditions used for the original problem, where supersonic inlet/outlet conditions were used with slip walls.

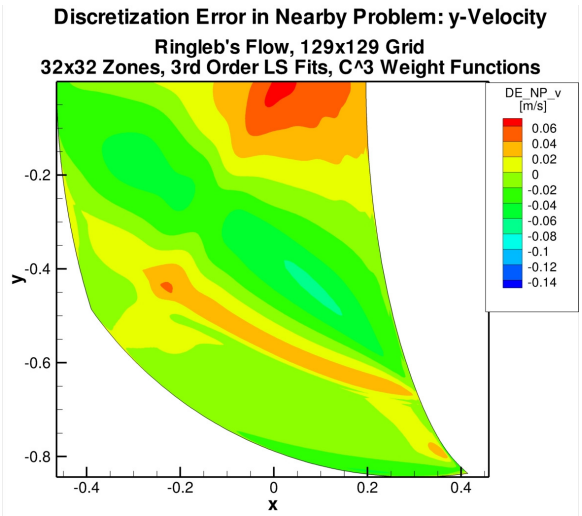

Discretization Error in Original Problem: Density
Ringleb's Flow, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions
DE_OP_rho
[kg/m^3]
0.00015
0.0001
5E-05
0
-5E-05
-0.0001
y
x
-0.2
-0.4
-0.6
-0.8
-0.4
-0.2
0
0.2
0.4


Discretization Error in Nearby Problem: Density
Ringleb's Flow, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions
DE_NP_rho
[kg/m^3]
0.00015
0.0001
5E-05
0
-5E-05
-0.0001
y
x
-0.2
-0.4
-0.6
-0.8
-0.4
-0.2
0
0.2
0.4


Discretization Error in Original Problem: x-Velocity
Ringleb's Flow, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions
DE_OP_u
[m/s]
0.05
0.025
0
-0.025
-0.05
-0.075
-0.1
-0.125
-0.15
y
x
-0.2
-0.4
-0.6
-0.8
-0.4
-0.2
0
0.2
0.4


Discretization Error in Nearby Problem: x-Velocity
Ringleb's Flow, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions
DE_NP_u
[m/s]
0.05
0.025
0
-0.025
-0.05
-0.075
-0.1
-0.125
-0.15
y
x
-0.2
-0.4
-0.6
-0.8
-0.4
-0.2
0
0.2
0.4


Discretization Error in Original Problem: y-Velocity
Ringleb's Flow, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions
DE_OP_v
[m/s]
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12
-0.14
y
x
-0.2
-0.4
-0.6
-0.8
-0.4
-0.2
0
0.2
0.4


Discretization Error in Nearby Problem: y-Velocity
Ringleb's Flow, 129x129 Grid
32x32 Zones, 3rd Order LS Fits, C^3 Weight Functions
DE_NP_v
[m/s]
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12
-0.14
y
x
-0.2
-0.4
-0.6
-0.8
-0.4
-0.2
0
0.2
0.4

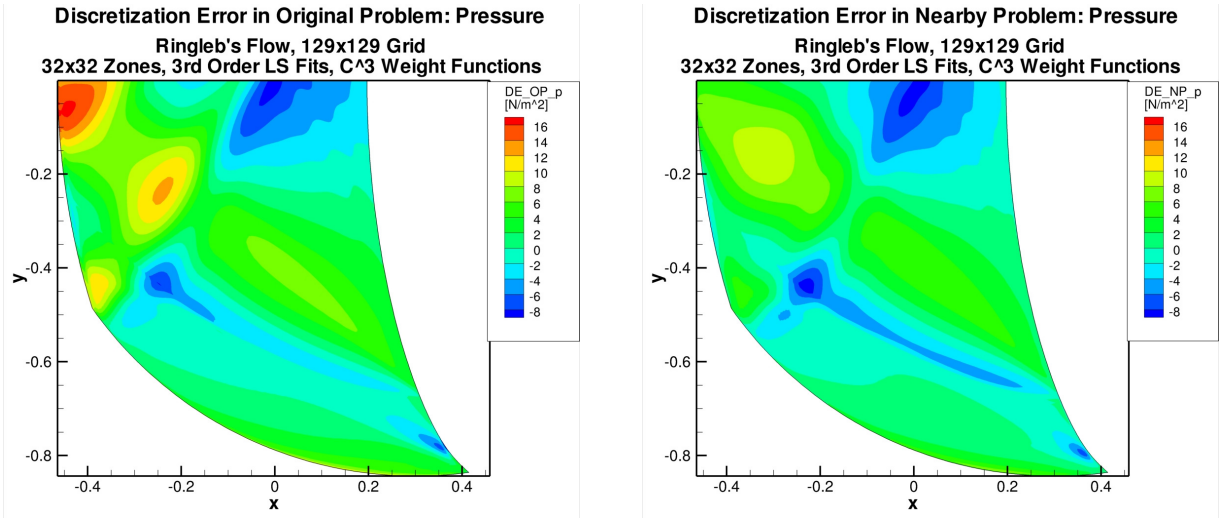


Figure 65: Ringleb's Flow Discretization Error Comparison

Figure 66 shows an x-y plot of the discretization error as a slice through the domain at a constant $x = -0.22$ m. The nearby problem discretization error has features similar to that of the original problem discretization error, but under predicts one peak and over predicts another.

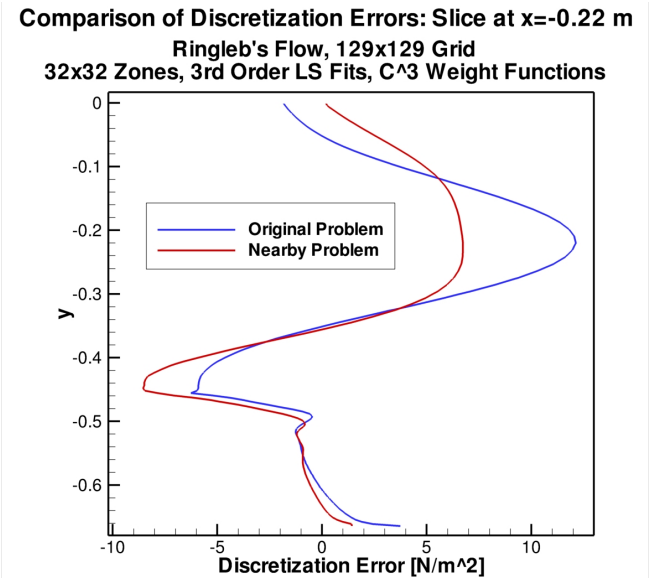


Figure 66: Comparison of Discretization Errors, x-y Plot

5.2.4 Subsonic Airfoil

A subsonic airfoil case is presented next. The Euler equations are used to solve the inviscid problem which has no analytical exact solution. The airfoil is run at sea level atmospheric conditions with a Mach number of 0.5 and 4 degrees angle of attack. The airfoil is gridded using a "C" shaped grid, which wraps around the airfoil. The fine grid is 129x257 and shown in Figure 67.

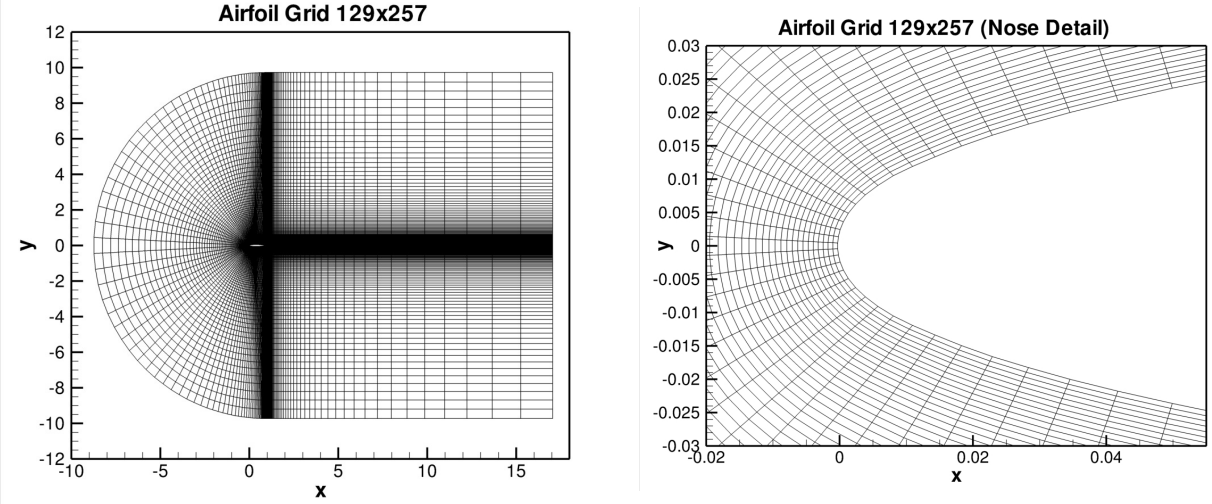
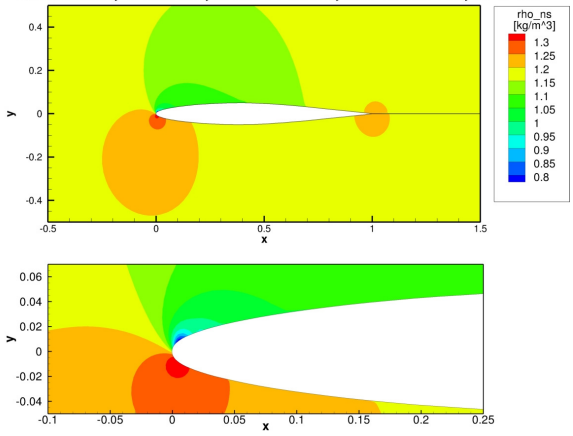


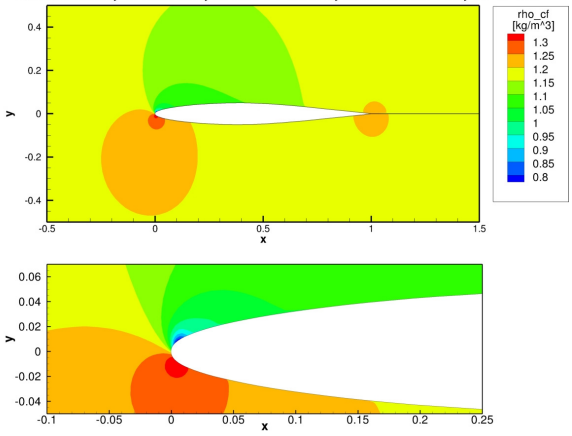
Figure 67: Airfoil Grid

The curve fits of the primitive variables and the energy term are presented in Figure 68. The curve fit errors are presented in Figure 69. These plots show the domain around the airfoil as well as a close-up view of the nose region. The curve fit error is largest near the top of the nose. The curve fit error plots use different contour levels between the overview and close-up to show the behavior throughout the domain. Maximum absolute percent curve fit errors are given in Table 7 for density, pressure, and energy. The velocity percent errors use the approximate inlet velocity of 175 m/s in the denominator, as both velocities pass through zero.

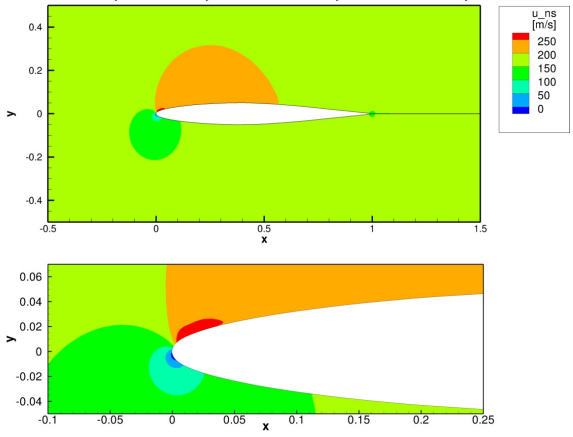
Numerical Solution to Original Problem: Density
Airfoil Grid, 129x257, 32x64 Zones, 3rd Order LS, C^3 WF



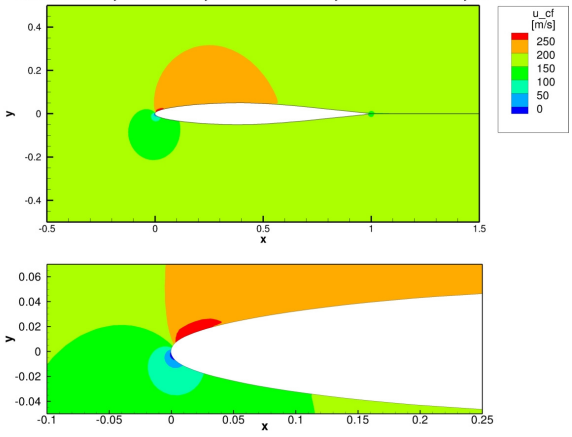
Curve Fit to Numerical Solution: Density
Airfoil Grid, 129x257, 32x64 Zones, 3rd Order LS, C^3 WF



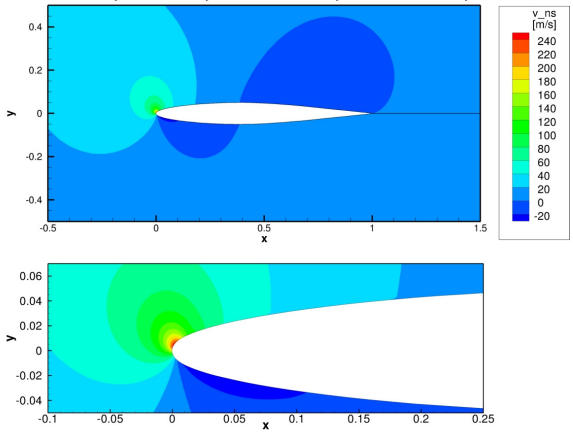
Numerical Solution to Original Problem: x-Velocity
Airfoil Grid, 129x257, 32x64 Zones, 3rd Order LS, C^3 WF



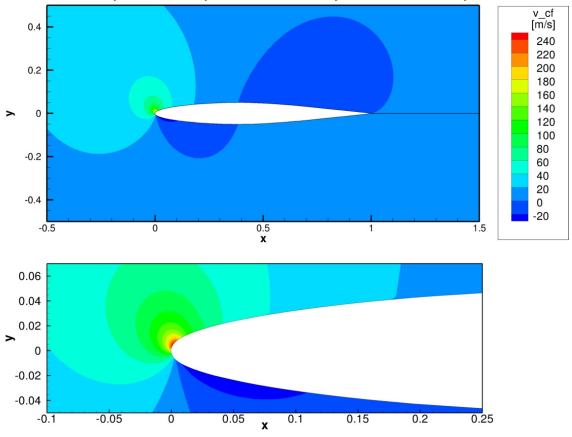
Curve Fit to Numerical Solution: x-Velocity
Airfoil Grid, 129x257, 32x64 Zones, 3rd Order LS, C^3 WF



Numerical Solution to Original Problem: y-Velocity
Airfoil Grid, 129x257, 32x64 Zones, 3rd Order LS, C^3 WF



Curve Fit to Numerical Solution: y-Velocity
Airfoil Grid, 129x257, 32x64 Zones, 3rd Order LS, C^3 WF



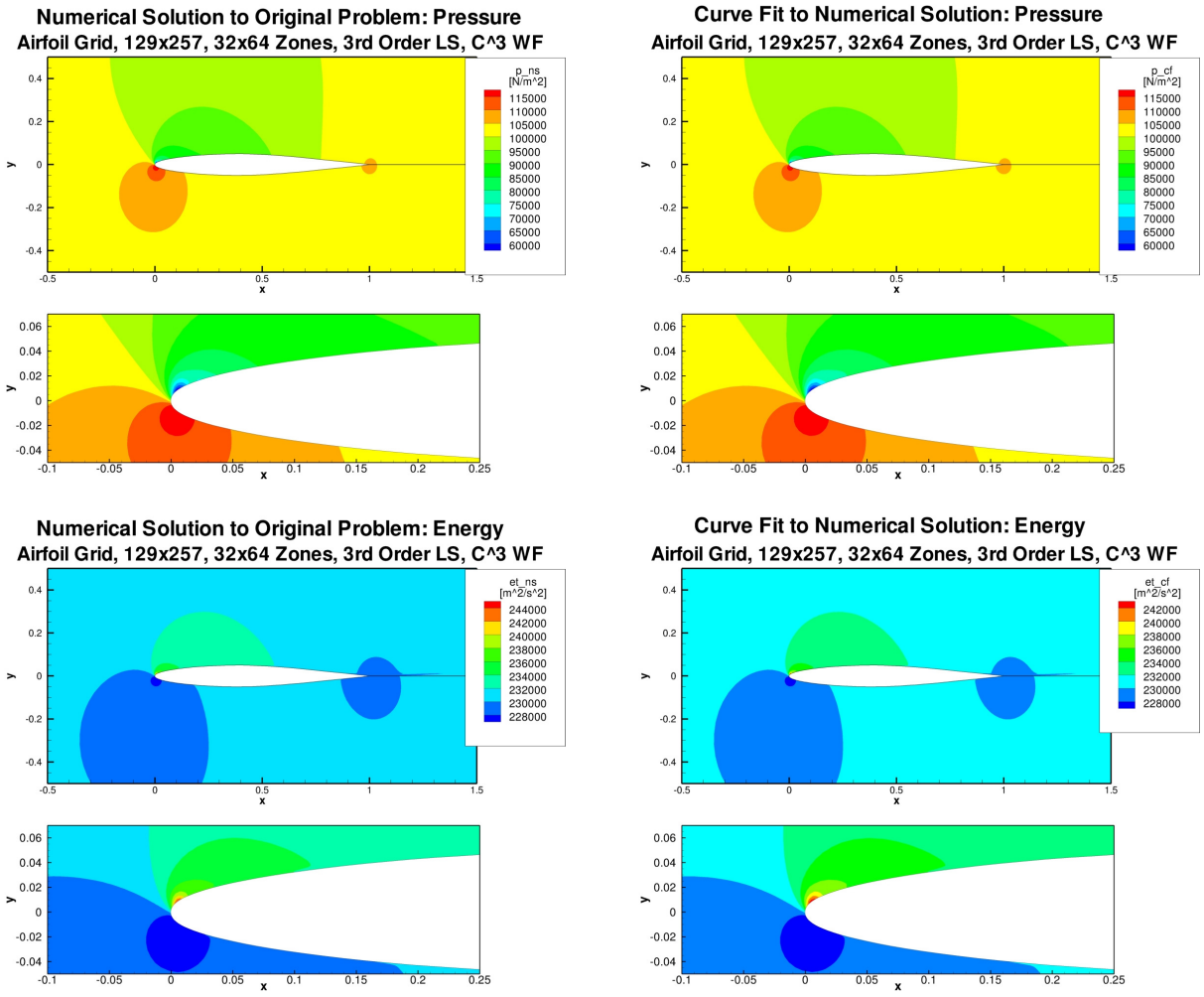
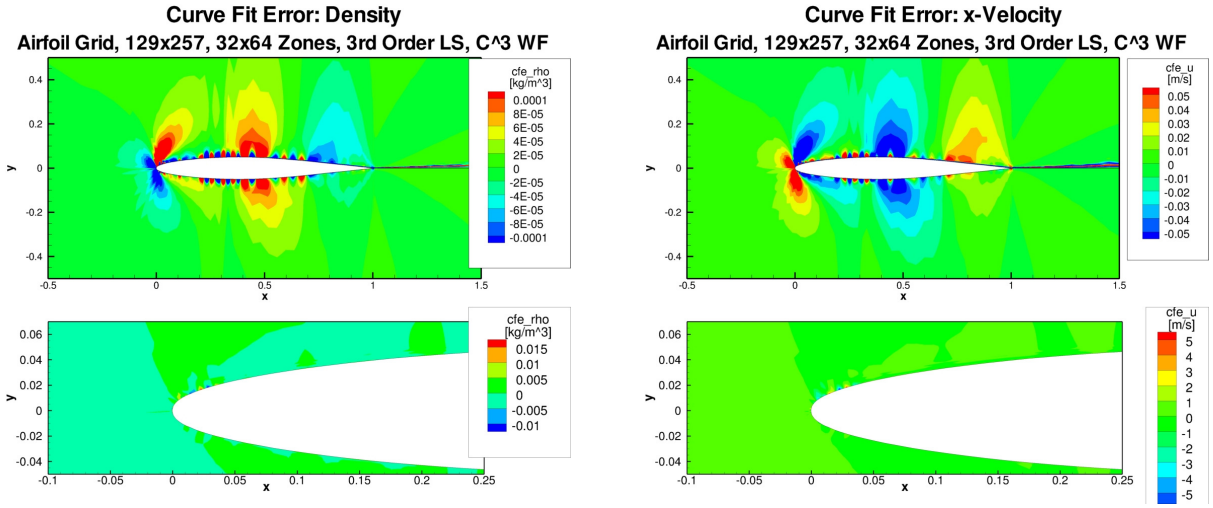


Figure 68: Airfoil Numerical Solution and Curve Fits



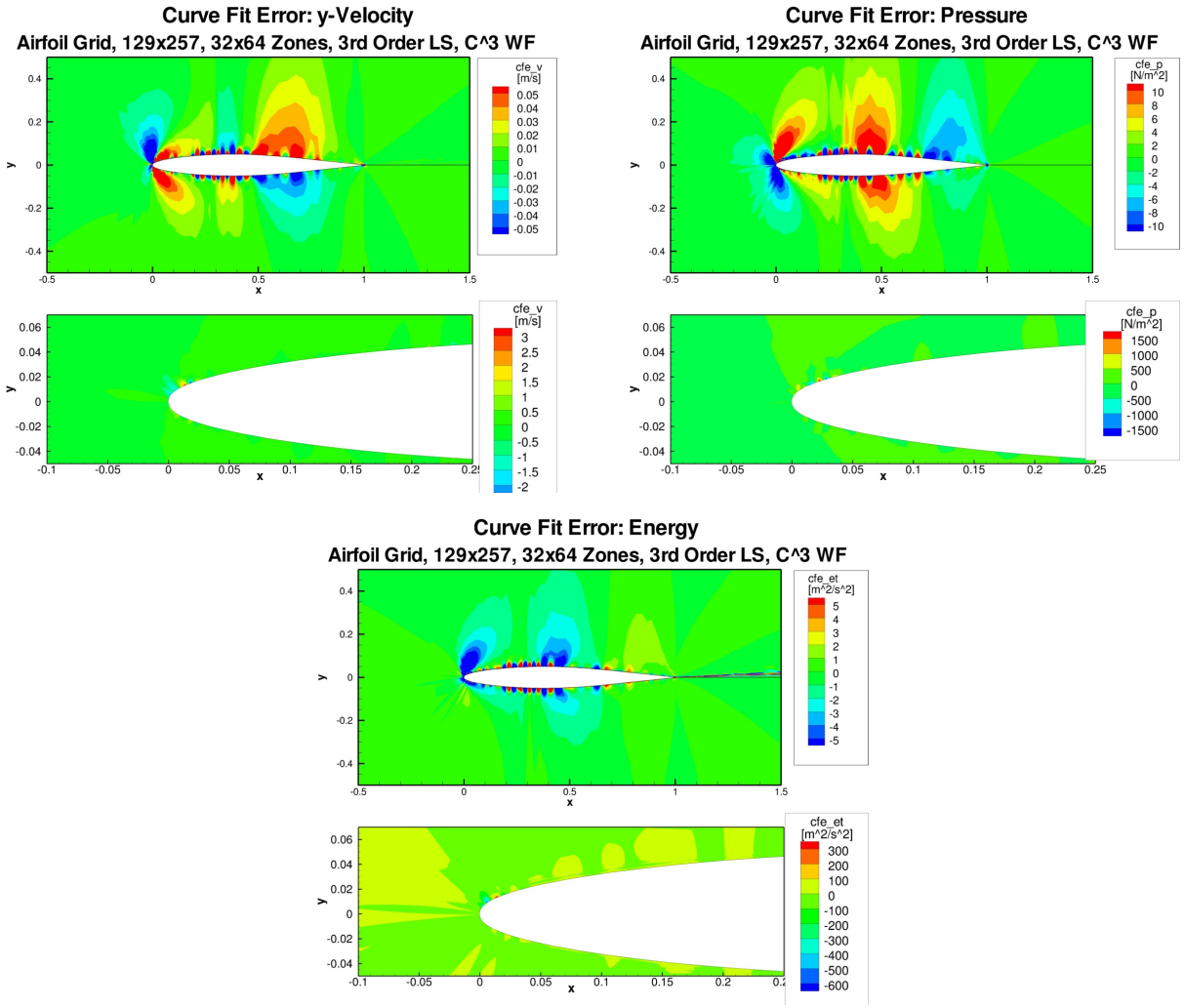


Figure 69: Airfoil Curve Fit Errors

Table 7: Subsonic Airfoil Maximum Percentage Curve Fit Errors

	ρ	u	v	p	e_t
% CF Error	2.02	3.81	2.02	2.82	0.287

The source terms are shown in Figure 70. The source terms are concentrated at the surface of the airfoil, which is expected as this is where the solution changes most rapidly. This causes some curve fitting errors which translate into source terms. Some cells near the nose of the airfoil have significantly higher values than anywhere else within the domain.

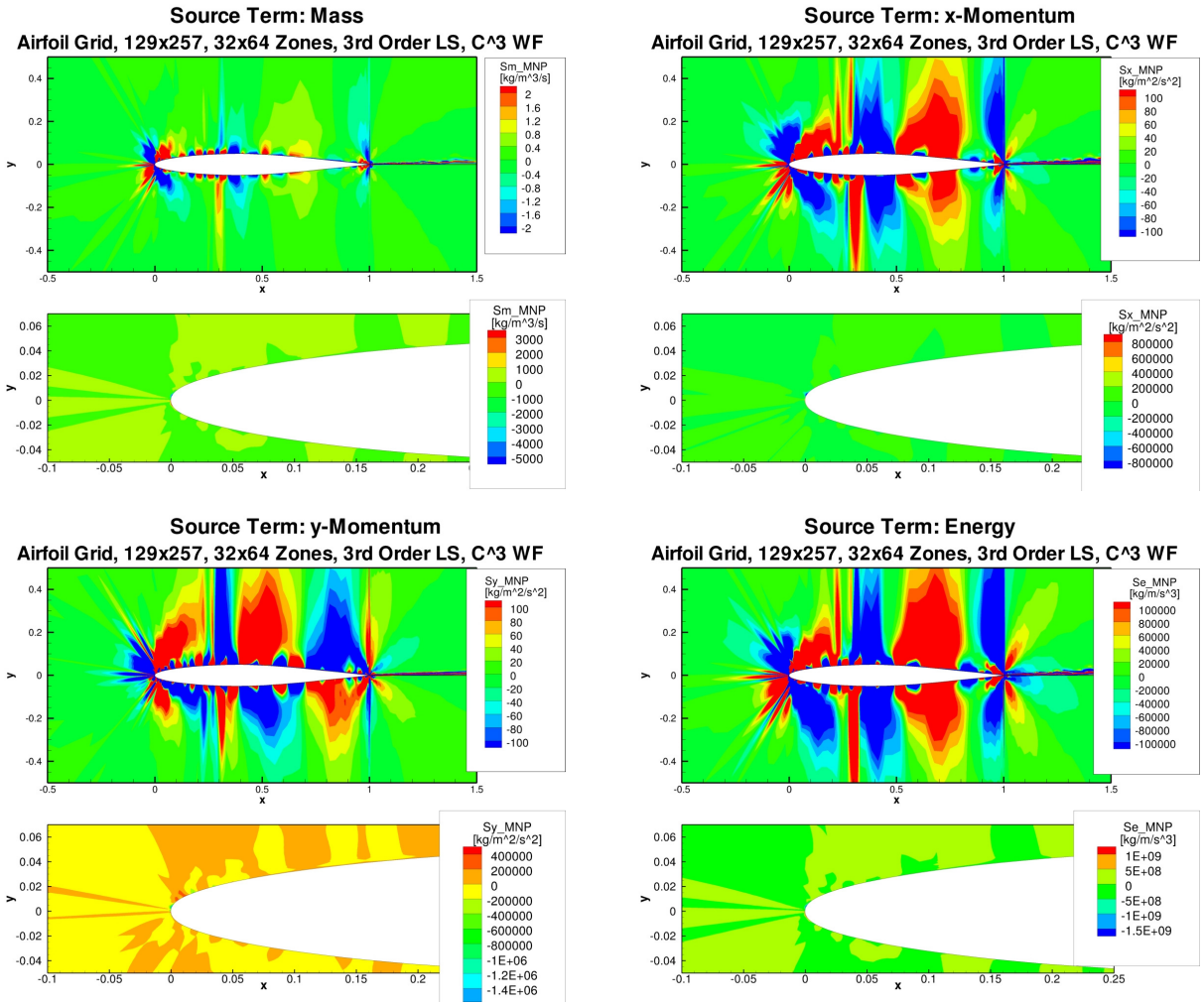


Figure 70: Airfoil MNP Source Terms

The exact discretization error in the original problem cannot be calculated, as the exact solution to the original problem in not known. The discretization error in the nearby problem is shown in Figure 71.

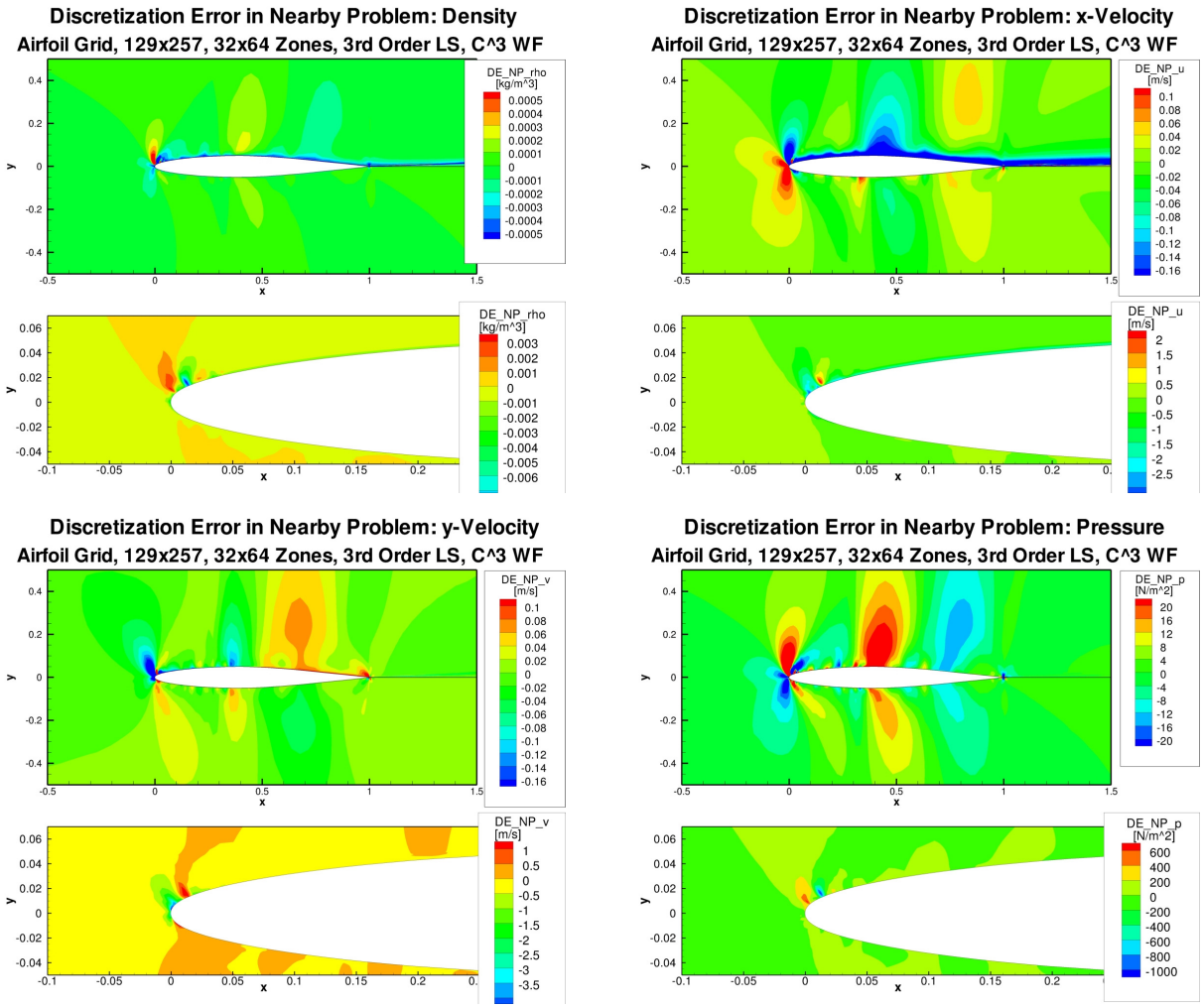


Figure 71: Airfoil Nearby Problem Discretization Errors

Several x-y line plots are shown in Figure 72, showing the pressure discretization error for a constant $y = 0$. One is an enlargement of the interesting behavior, while the other shows an overview. There is a large oscillation in the discretization error just before the flow reaches the nose of the airfoil. This is not unexpected as there is a sharp pressure gradient here, with the stagnation point close by. The region around the stagnation point is a difficult region to solve for numerically. There is no data shown behind the airfoil, due to the nature of the grid. The values plotted are cell-centered. As the grid wraps around the airfoil, it meets up with itself behind the trailing edge of the airfoil. However, as the cell-centered, ordered data is read into Tecplot there is a seam in the grid, through which the $y = 0$ line falls. Another set of x-y line plots are shown

in Figure 73. This time the pressure discretization error is shown for a constant $x/c = 0.05$. There are some oscillations as near the surface of the airfoil, as expected.

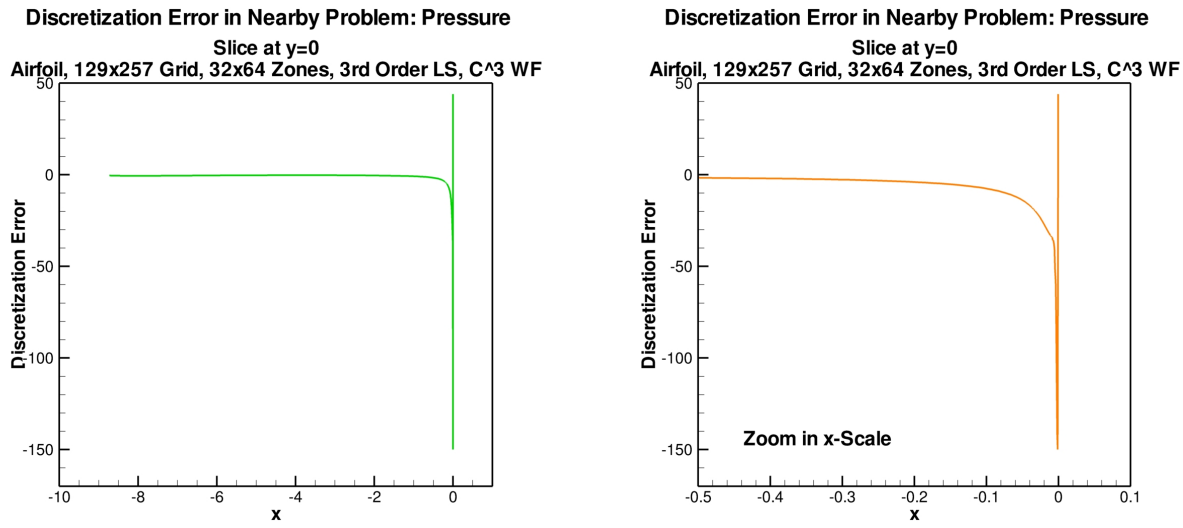


Figure 72: Airfoil x-y Plot of Pressure Discretization Error, Slice at y=0

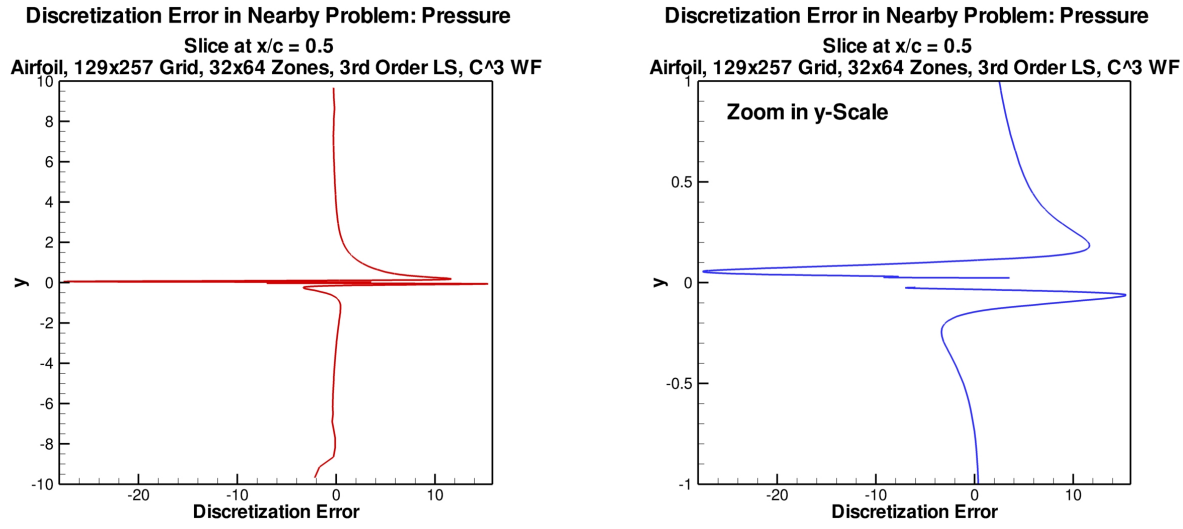


Figure 73: Airfoil x-y Plot of Pressure Discretization Error, Slice at x/c=0.05

Richardson extrapolation was also performed on the airfoil problem. This calculation uses the fine grid (129x257) and coarse grid (65x129), for a refinement factor of two. The formal order of accuracy of two was also used. Equation 30 shows the expression for the Richardson extrapolation error estimate in the fine solution

$$f_h - \bar{f} = -\frac{f_h - f_{rh}}{r^p - 1} \tag{30}$$

where $\bar{f}$ is the estimated exact solution, h is the fine grid spacing parameter, r is the grid refinement factor, and p is the formal order of accuracy. It follows that f_h is the fine grid solution and f_{rh} is the coarse grid solution. The Richardson extrapolation error estimates are shown in Figure 74. The Richardson extrapolation error estimates seem to be on the same order of magnitude as the MNP nearby problem discretization errors. Richard extrapolation also suggests that the location of maximum discretization error tends to be on the upper surface of the airfoil's nose.

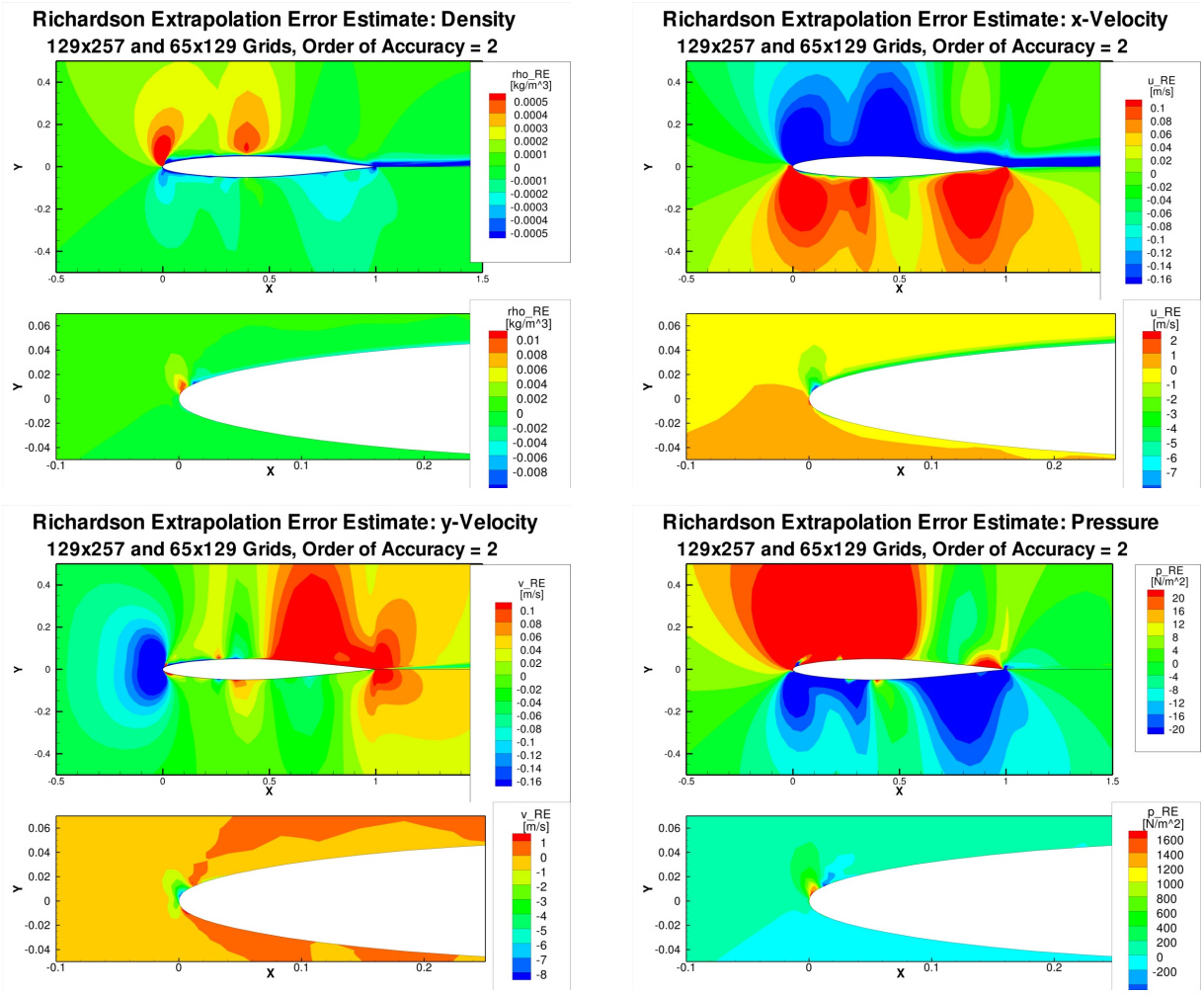


Figure 74: Airfoil Richardson Extrapolation Error Estimate

In order to improve the airfoil curve fit and decrease the discretization errors, a finer grid with 129 by 385 points was used. In addition to using more points, the ratio of points around the airfoil surface compared to the rest of the domain was increased. The old grid used previously is shown alongside the new grid in Figure 75. The new grid clearly has more cells defining the leading edge geometry.

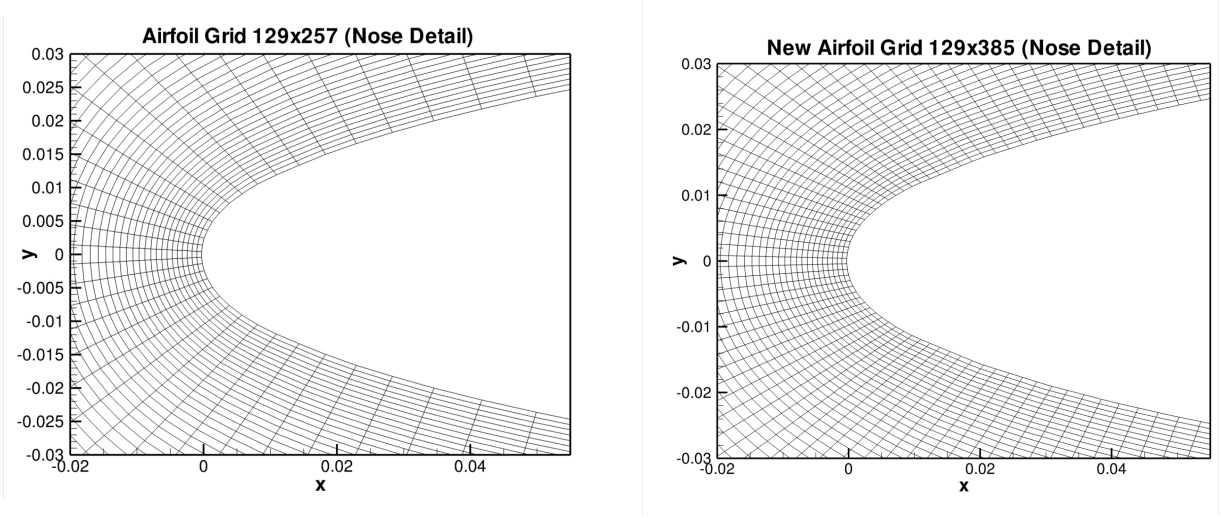


Figure 75: Comparison of Grid Leading Edges

The curve fit errors are presented in Figure 76. These are similar to the curve fit errors for the old grid that were shown earlier. However, the magnitude of the curve fit errors has decreased. This can be seen by examining the contour levels and also through Table 8 which compares the maximum percent curve fit error for the previous and current cases. For example, the maximum percent curve fit for pressure decreased by 21% between the old and new cases. The source terms are shown in Figure 77. These exhibit similar behavior to the curve fit errors, where there are oscillations along the surface of the airfoil.

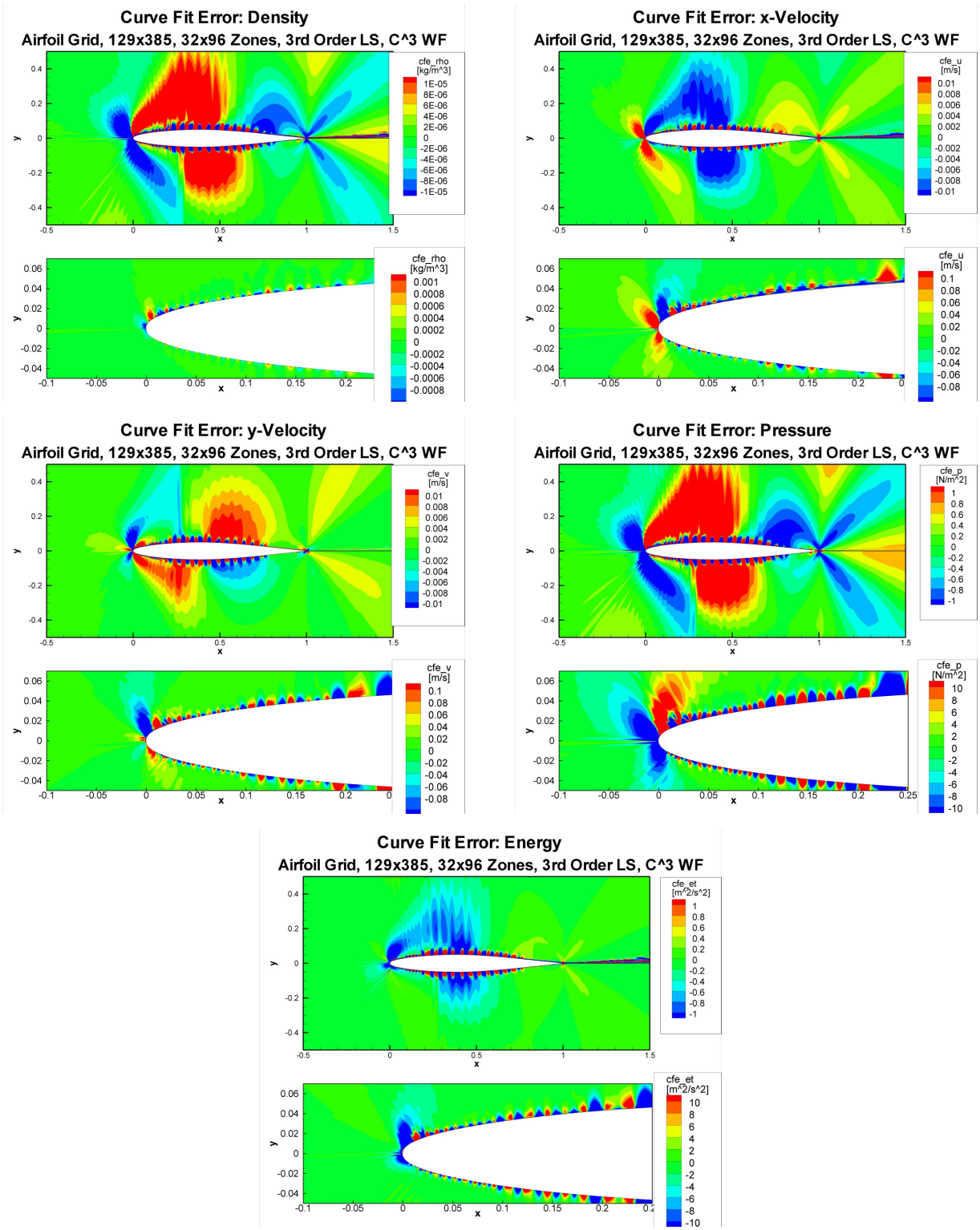


Figure 76: Airfoil Curve Fit Errors, New Grid

Table 8: Subsonic Airfoil Maximum Percentage Curve Fit Errors, Grid Comparison

% CF Error	ρ	u	v	p	e_t
129x257	2.02	3.81	2.02	2.82	0.287
129x385	1.61	2.74	2.02	2.24	0.272

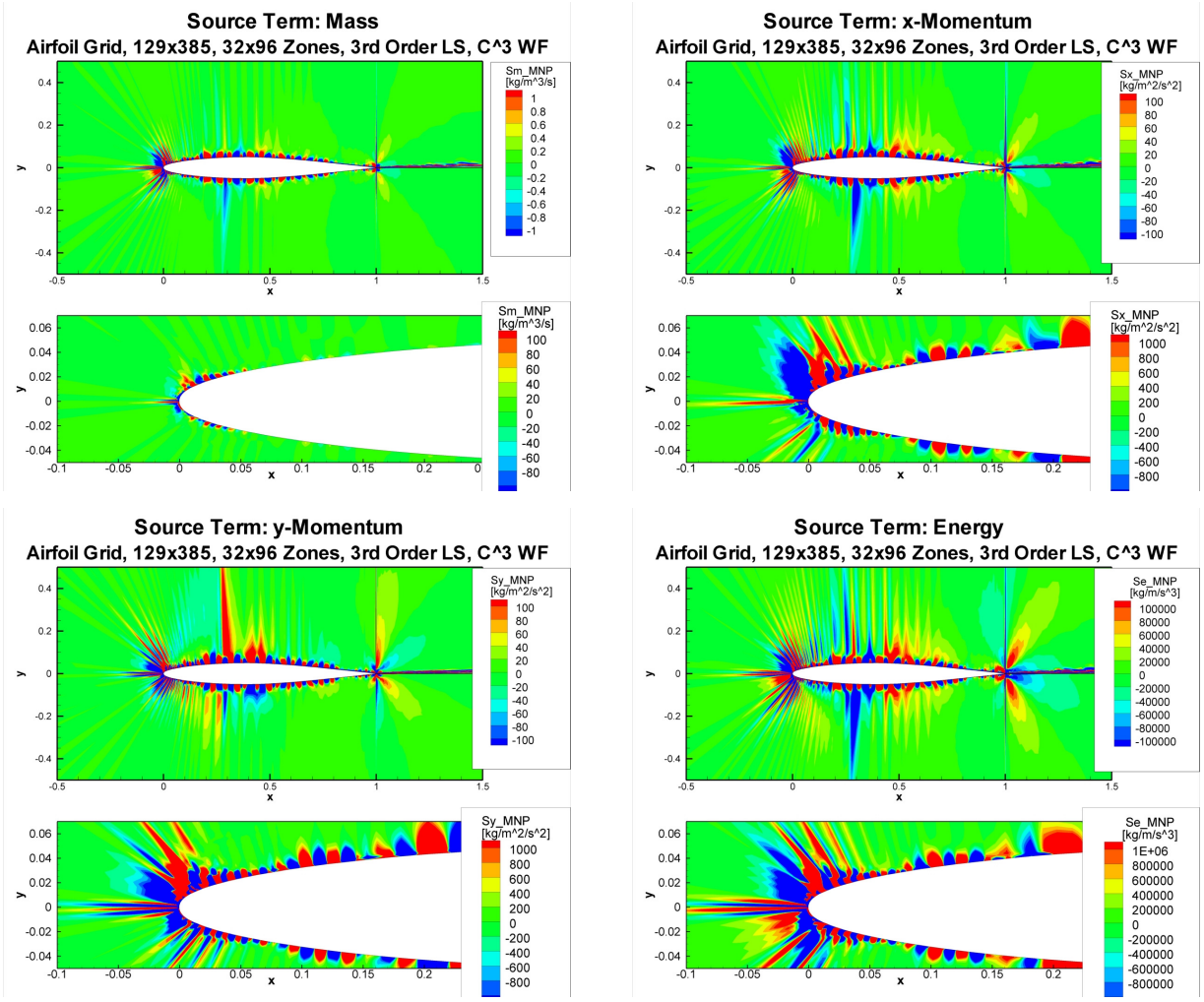


Figure 77: Airfoil MNP Source Terms, New Grid

The discretization error for the nearby problem is shown in Figure 78. The magnitude of the discretization error has decreased due to the use of the finer grid. The Richard extrapolation discretization error estimate was calculated with the 129x385 grid and a 65x193 grid. These error estimates are shown in Figure 79. The Richardson extrapolation error estimates are larger in magnitude than the MNP discretization error estimates (nearby problem discretization error).

There is some qualitative matching of error features, such as for the x-velocity where both procedures estimate negative error along the top surface of the airfoil. As with the previous case, Richardson extrapolation also predicts large discretization errors on top of the airfoil nose.

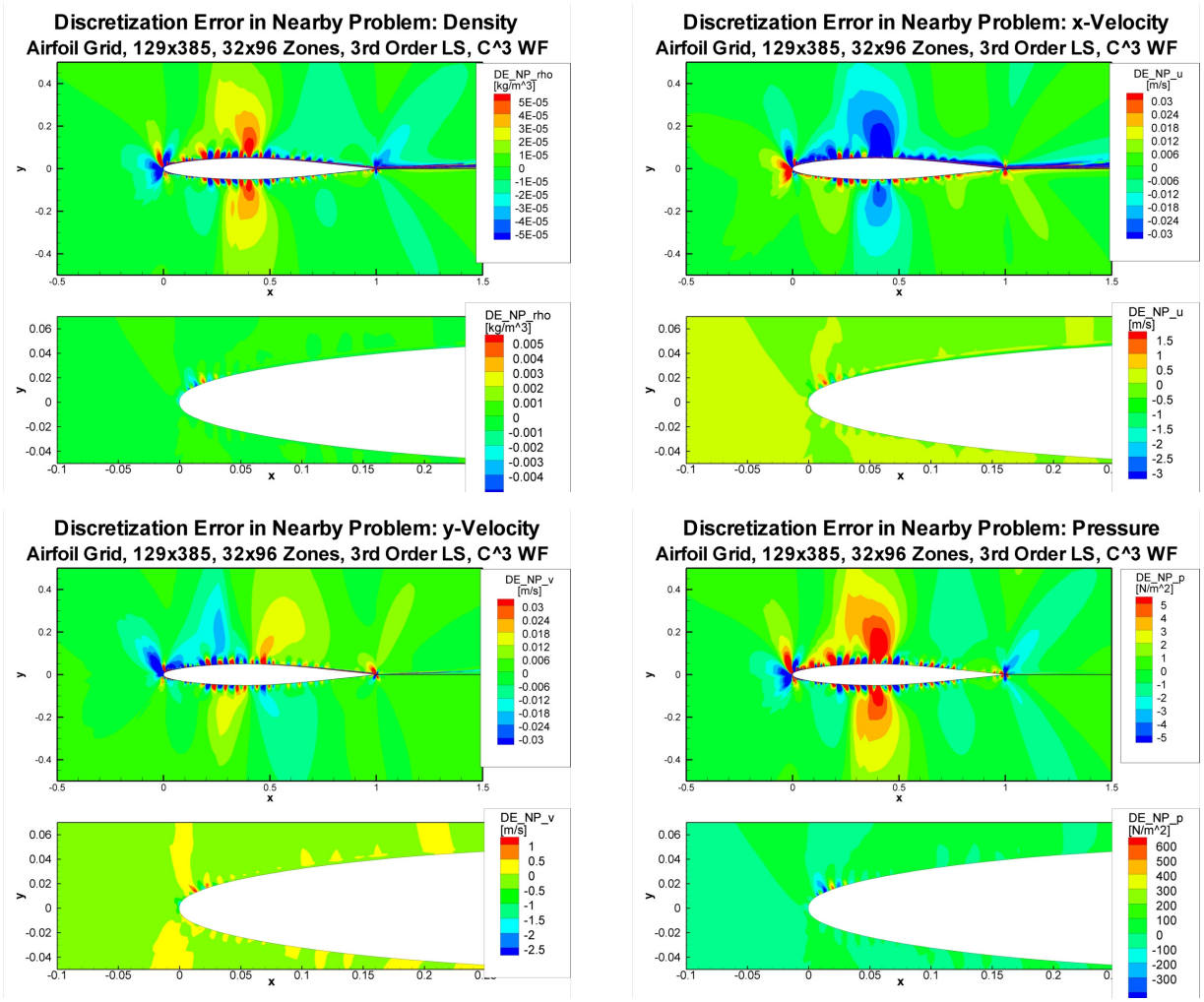


Figure 78: Airfoil Nearby Problem Discretization Errors, New Grid

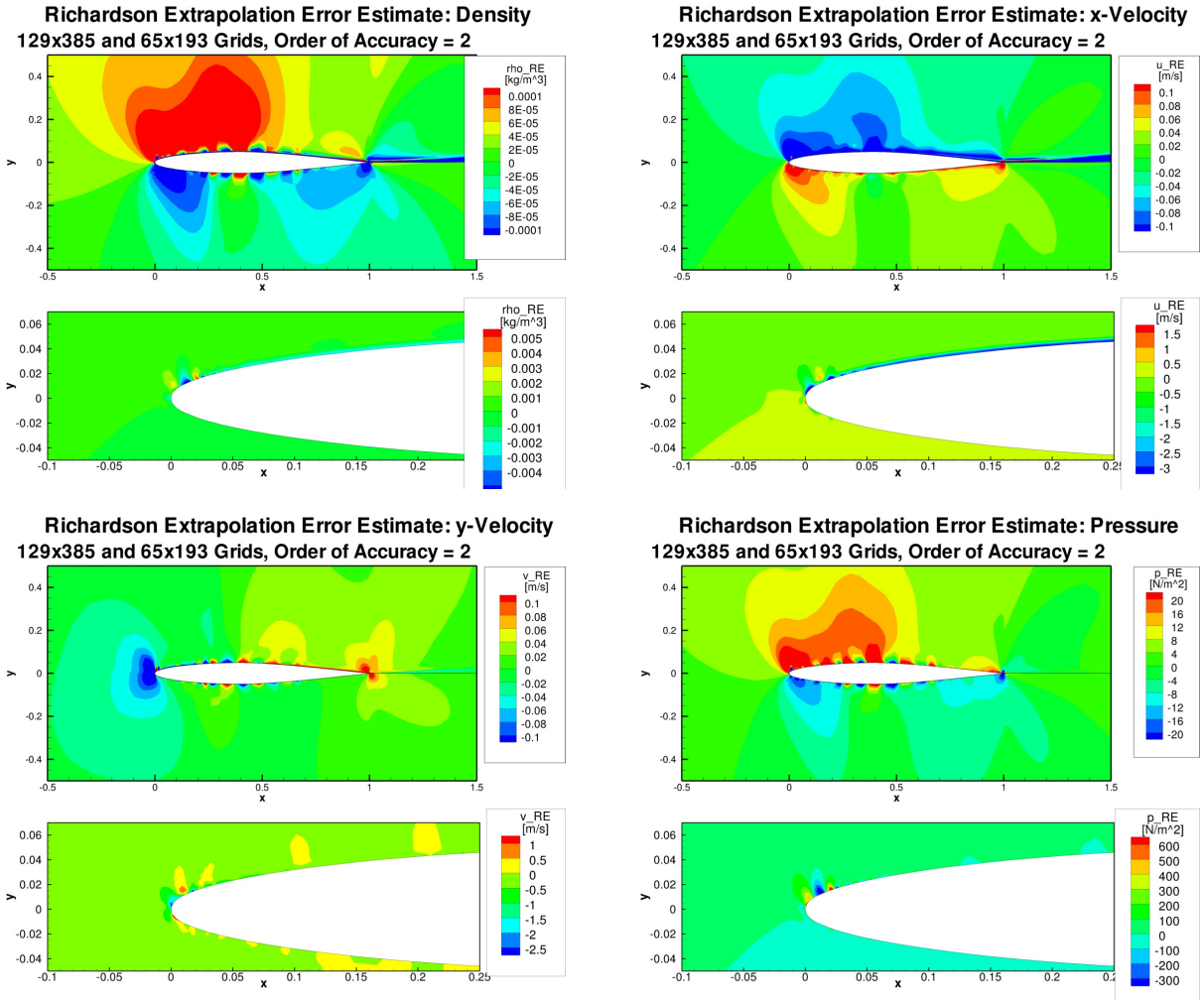


Figure 79: Airfoil Richardson Extrapolation Error Estimate, New Grid

6 Conclusions

6.1 Summary of Results

This work covers the Method of Nearby Problems (MNP) as a method for producing analytical exact solutions to problems governed by partial differential equations (PDEs). Many problems governed by PDEs do not have analytical exact solution, so the ability to define a nearby problem, with an exact solution is important. The ability to use this nearby problem to estimate the discretization error in the original problem of interest with out additional grid levels is also discussed. Having a realistic exact solution is useful for evaluating numerical schemes, discretization error estimators, and mesh adaptation strategies.

The topic of curve fitting is covered as it plays an important role in the process. Previous literature has shown that global curve fits are often ineffective at fitting complex solutions. To avoid this, the solution is broken into regions which are fitted independently with a least squares approach. These local fits are merged together with weight functions that are specifically chosen to enforce a particular level of continuity between the pieces of the final piecewise fit.

A one-dimensional case of the steady state Burgers' equation is shown to compare the least squares and weighting function approach to previous examples of MNP applied with global curve fits. The weighting function approach gives a more accurate curve fit, while reducing the magnitude of the resulting source term.

Six two-dimensional cases are explored, as follows:

1. Burgers' Equation: Shock Coalescence
2. Burgers' Equation: Pulse Decay
3. Euler Equations: Manufactured Solution on Curvilinear Grid A
4. Euler Equations: Manufactured Solution on Curvilinear Grid B
5. Euler Equations: Ringleb's Flow
6. Euler Equations: Airfoil

The two time-varying Burgers' equation cases were chosen as further examples of the ability of the MNP in producing exact solution. One case represents a shock coalescence, which as time approaches infinity becomes the steady state shock discussed as a one-dimensional case. The other time-varying case is the pulse decay. The shock coalescence provided better results than the pulse decay, as it was an easier case to fit, at least for the

particular parameters chosen. For the Reynolds number and time period run, the pulse decay has very sharp flow features at one boundary. The curve fit had difficulty in fully capturing this behavior, which caused large source terms.

The manufactured solution cases were chosen for their smooth solutions. The two curvilinear grids differ by the severity of the stretching, skewing, and curvature. This allows for a comparison of the two cases in terms of the effect of the grid upon the curve fit and source terms. Refining the grid and simultaneously increasing the number of fit zones used was shown to decrease the curve fit errors and source term magnitudes. Refining only the number of fit zones used decreased some of the curve fitting error, but had less of an effect on the source term examined. Increasing the order of the least squares fit decreased the curve fit error and source term, although the increase from 2^{nd} to 3^{rd} order was not as noticeable as from 1^{st} to 2^{nd} order. Having an exact solution to the nearby problem was useful as well, as the true discretization error in the original problem could be directly calculated. The effectiveness of using the nearby problem discretization error as an estimate of the original problem discretization error could be examined. The MNP provided a good estimate of the error. The form and magnitude of the original discretization error are captured in the nearby discretization error.

Ringleb's flow was chosen as another Euler equation case as it has an analytical exact solution. The curve fit error and source terms have some noticeable effects from the grid present. The abrupt change in cell aspect ratio from top to bottom cause a noticeable feature in the contour plots. The boundaries of the problem also show relatively high curve fit error and source terms. This is probably due to the fact this problem is more difficult to solve numerically than a manufactured solution problem with Dirchlet boundary conditions. The nearby problem discretization error was on the same order of magnitude as the original problem discretization error, but did not capture all of the features accurately.

The airfoil case was run as an example of a more realistic engineering problem. A subsonic flow with moderate angle of attack was chosen in order to produce a relatively smooth solution with no shock. The airfoil nearby problem shows results similar to the previous cases. The nose of the airfoil was the most difficult region of the flow domain to fit. This was expected as the flow experiences a rapid expansion at this point. The nearby problem discretization errors seem reasonable. Discretization error estimates from Richardson

extrapolation are also examined. In general Richardson extrapolation shows discretization error on the same order of magnitude as the MNP, but the form of the discretization errors is not identical. There is also a large amount of error on the top of the nose, similar to the MNP discretization error estimate. In addition, using a finer grid was shown to decrease curve fit error and discretization error.

These cases provide more evidence as to the effectiveness of the Method of Nearby Problems. The process can create realistic, analytical exact solutions to problems of interest. The method can also be used to give a reasonable estimate of the discretization error in the nearby problem. While this error estimate requires the problem be run twice, it requires only the original grids; no new grids need to be generated.

6.2 Direction of Future Work

The next logical extension of the MNP is to three and then four-dimensional problems. There is no inherent limit on the dimensionality of the problems that the MNP is applied to. The local fit and weighting function approach can be applied to an arbitrary number of dimensions. The higher dimension problems will of course be more complex and require more computing time. Eventually, time-varying 3D MNP cases should allow for the study of the interaction between numerical methods and subgrid turbulence models in turbulence simulations.

7 References

[1] Benton, E. R. and Platzman, G. W., "A Table of solutions of the One-Dimensional Burger's Equation," *Quarterly of Applied Math*, Vol. 30, p. 195-212, 1972.

[2] Oberkampf, W. L. and Roy, C. J., Verification and Validation in Scientific Computing, Cambridge University Press, New York, to appear 2010.

[3] Layton, W., Lee, H.K and Peterson, J., "A Defect-correction Method for the Incompressible Navier-Stokes Equations," *Applied Mathematics and Computation,* Vol. 129, p. 1-19, 2002.

[4] "Guide for the Verification and Validation of Computational Fluid Dynamics Simulations," *American Association of Aeronautics and Astronautics*, G-077-1998, Reston, VA, 1998.

[5] Blottner, F. G., "Accurate Navier-Stokes Results for the Hypersonic Flow Over a Spherical Nosetip," *Journal of Spacecraft and Rockets*, Vol. 27, No. 2, p. 113-122, 1990.

[6] Roache, P. J., "Verification of Codes and Calculations," *AIAA Journal*, Vol. 36, No. 5, May 1998.

[7] Roache, P. J., "Error Bars for CFD," *41st Aerospace Sciences Meeting and Exhibit,* Jan. 6-9, 2003, Reno, NV.

[8] Roy, C. J., Smith, T. M., and Ober, C. C., "Verification of a Compressible CFD Code using the Method of Manufactured Solutions," *AIAA Paper 2002-3110*.

[9] Eça, L. and Hoekstra, M., "Verification of Turbulence Models with as Manufactured Solution," *European Conference on Computational Fluid Dynamics*, ECCOMAS CFD 2006.

[10] Roy, C. J. and Hopkins M. M., "Discretization Error Estimates Using Exact Solutions to Nearby Problems," *41st Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, Jan. 6-9, 2003.

[11] Raju, A., "Discretization Error Estimation Using the Method of Nearby Problems: One-Dimensional Cases," Master's Thesis, Auburn University, Auburn, AL, Aug. 8, 2005.

[12] Roy, C. J., Raju, A., and Hopkins, M. M., "Estimation of Discretization Errors Using the Method of Nearby Problems," *AIAA Journal*, Vol. 45, No. 6, June 2007.

[13] Roy, C. J. and Sinclair, A. J., "On the Generation of Exact Solutions for Evaluating Numerical Schemes and Estimating Discretization Error," *Journal of Computational Physics*, Vol. 228, p. 1790-1802, Nov. 2009.

[14] Kurzen, M. J., Phillips, T. S., Roy, C. J., and Sinclair, A. J., "Method of Nearby Problems for Generating Exact Solutions to 1D Unsteady and 2D Steady Problems," *19th AIAA Computational Fluid Dynamics Conference*, San Antonio, TX, June 2009.

[15] Junkins, J. L., Miller, G. W., and Jancaitis, J. R., "A Weighting Function Approach to Modeling of Irregular Surfaces," *Journal of Geophysical Research,* Vol. 78 No. 11, April 10, 1973.

[16] Jancaitis, J. R. and Junkins, J. L., "Modeling in n Dimensions Using a Weighting Function Approach," Journal of Geophysical Research. Vol. 79 No. 23, Aug. 10, 1974.

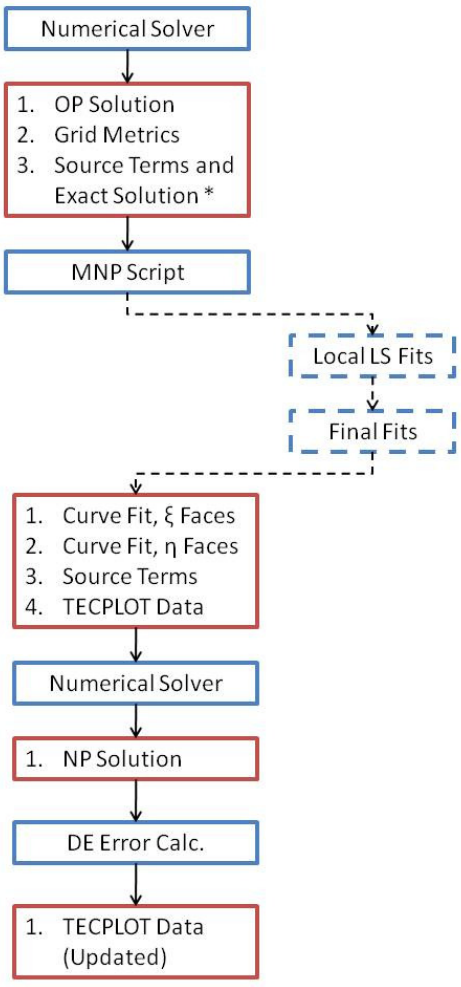
[17] Stetter, H. J., "The Defect Correction Principle and Discretization Methods," *Numerische Mathematik*, Vol. 29, p. 425-443, 1978.

[18] Rogers, S. E. and Pulliam, T. H., "Accuracy Enhancements For Overset Grids Using A Defect Correction Approach," *32nd Aerospace Sciences Meeting & Exhibit*, Jan. 10-13, 1994, Reno, NV.

Appendix A: Code Flow Overview

Code Flow Overview: The following figure shows the overall flow of information and codes used to run the Method of Nearby Problems.

The blue boxes represent codes while the red boxes list relevant input and output to the codes. The dashed arrows and boxes are for information and codes that are called automatically by another code.



1. The numerical solver is run.
2. This generates several output files including a file containing the numerical solution to the original problem and another file containing the grid and grid metrics. If applicable to the particular problem, a file detailing the source terms and/or exact solution is also produced.
3. A script file is run which then generates the local least squares fits and then merges them with the weighting function to create the final fits. The source terms are also generated in this step.
4. The output files include the curve fit evaluated at ξ and η faces (boundary conditions for the nearby problem) and the nearby problem source terms. A file containing data for plotting in TECPLOT is also generated.
5. The numerical solver is run again.
6. The numerical solution to the nearby problem is created.
7. The discretization error in the nearby problem is then calculated.
8. The TECPLOT data file is updated to include this error.

Appendix B: Weight Function Generation Code

Weight Function Generation Code: This code generates the symbolic expression for the requested weight function. Optional outputs include an array of weight function values evaluated over a uniform grid and the associated plot.

Contents


```
function [p,val] = wfgen(m,dim,h)
tic
```

Weighting Function Generator

```
VERSION: 6
WRITTEN BY: Matthew Kurzen
            Graduate Research Assistant to Dr. Christopher Roy
            Aerospace Engineering Department at Virginia Tech
This code generates a polynomial weighting function of arbitrary order
continuity and of 1-4 dimensions, depending on user input.
Form and conditions for the weight functions is taken in part from:
Jancaitis, J. R. and Junkins, J. L., "Modeling in n Dimensions Using a
Weighting Function Approach," Journal of Geophysical Research, Vol. 79,
No. 23, 1974.
NOTE: There may be numerical issues with selecting extremely high orders
of continuity. The matrix used to solve for the coefficinents becomes
ill-conditioned leading to incorrect decimal places. As all of the
coefficients are so far known to be integers, the code rounds the
double-precision coefficients to the nearest integer value.
INPUT:
m = order of continuity (must be non-negative integer)
dim = (must be integer values of 1, 2, 3, or 4)
h = grid step size [OPTIONAL INPUT] - if included this will be used for
    evaluating the weight function at certain grid points within the
    domain and for plotting, where applicable
OUTPUT:
p = weighting function polynomial
val = array of weighting function values over a grid of coordinates
      [OPTIONAL OUTPUT] - this is only outputted if h is inputted
* Plots are also generated for 1-3 dimensional weighing functions, if
```

```
  the parameter h is specified
```

Generate Form of Weighting Function

```
syms x y z t real

% Order of 1D polynomial weighting function
n=(m+1)+m;

% Generate array of symbolic terms
for i=1:n+1
    w(1,i)=x^(n+1-i);
end
```

Generate Partial Matrix and Vector, from Left Endpoint Conditions

```
% Value Condition
A(1,:)=subs(w,x,0);
b(1,1)=0;

% Derivative Conditions (if applicable)
k=0;
while m>k
    j=0;
    wder=w;
    while j<=k
        wder=diff(wder,x);
        j=j+1;
    end
    A(k+2,:)=subs(wder,x,0);
    b(k+2,1)=0;
    k=k+1;
end
```

Generate Partial Matrix and Vector, from Right Endpoint Conditions

```
% Value Condition
A(m+2,:)=subs(w,x,1);
b(m+2,1)=1;

% Derivative Conditions (if applicable)
k=0;
while m>k
    j=0;
    wder=w;
    while j<=k
        wder=diff(wder,x);
        j=j+1;
    end
    A(m+2+k+1,:)=subs(wder,x,1);
    b(m+2+k+1,1)=0;
    k=k+1;
end
```

Solve for Coefficients

```
% Solve
c=A\b;

% Convert coefficients to integers
cint=round(c);
```

Generate Polynomial (Advanced Form)

```
% ONE-DIMENSIONAL
if dim==1
    p=0;
    for i=m+2:n+1
        p=p+cint(i-m-1)*w(1,i);
    end
    p=x^(m+1)*p;

% TWO-DIMENSIONAL
elseif dim==2
    wy=subs(w,x,y);
    p=0;
    for i=m+2:n+1
        for j=m+2:n+1
            p=p+cint(i-m-1)*w(1,i)*cint(j-m-1)*wy(1,j);
        end
    end
    p=x^(m+1)*y^(m+1)*p;

% THREE-DIMENSIONAL
elseif dim==3
    wy=subs(w,x,y);
    wz=subs(w,x,z);
    p=0;
    for i=m+2:n+1
        for j=m+2:n+1
            for k=m+2:n+1
            p=p+cint(i-m-1)*w(1,i)*cint(j-m-1)*wy(1,j)*cint(k-m-1)*wz(1,k);
            end
        end
    end
    p=x^(m+1)*y^(m+1)*z^(m+1)*p;

% FOUR-DIMENSIONAL
elseif dim==4
    wy=subs(w,x,y);
    wz=subs(w,x,z);
    wt=subs(w,x,t);
    p=0;
    for i=m+2:n+1
        for j=m+2:n+1
            for k=m+2:n+1
                for l=m+2:n+1
                    p=p+cint(i-m-1)*w(1,i)*cint(j-m-1)*wy(1,j)*cint(k-m-
1)*wz(1,k)*cint(l-m-1)*wt(1,l);
                end
            end
        end
    end
    p=x^(m+1)*y^(m+1)*z^(m+1)*t^(m+1)*p;
end

fprintf('\n\n---------- C^%d %dD Weighing Function ----------\n\n', m, dim)
%p
```

---------- C^2 2D Weighing Function ----------

GRID GENERATION, WEIGHT FUNCTION EVALUATION and PLOTS

```
if nargin==3
```

```
    grd=0:h:1;

    % ONE-DIMENSIONAL Coordinate Values of Weighting Function
    if dim==1
        val=zeros(length(grd),1);
        for i=1:length(grd)
            val(i)=subs(p,x,grd(i));
        end

        % Call Plotting Subfunction
        comp1dplot(grd,val)

    end

    % TWO-DIMENSIONAL Coordinate Values of Weighting Function
    if dim==2
        val=zeros(length(grd),length(grd));
        for i=1:length(grd)
            for j=1:length(grd)
                val(i,j)=subs(subs(p,x,grd(i)),y,grd(j));
            end
        end

        % Call Plotting Subfunction
        comp2dplot(h,val)

    end

    % THREE-DIMENSIONAL Coordinate Values of Weighting Function
    if dim==3
        val=zeros(length(grd),length(grd),length(grd));
        for i=1:length(grd)
            for j=1:length(grd)
                for k=1:length(grd)
                    val(i,j,k)=subs(subs(subs(p,x,grd(i)),y,grd(j)),z,grd(k));
                end
            end
        end

        % Call Plotting Subfunction
        % NOTE: The slice plots will only be generated if h=0.1
        if length(grd)==11
            sliceplot(h,val)
        end

    end

    % FOUR-DIMENSIONAL Coordinate Values of Weighting Function
    if dim==4
        val=zeros(length(grd),length(grd),length(grd),length(grd));
        for i=1:length(grd)
            for j=1:length(grd)
                for k=1:length(grd)
                    for l=1:length(grd)

val(i,j,k,l)=subs(subs(subs(subs(p,x,grd(i)),y,grd(j)),z,grd(k)),t,grd(l));
                    end
                end
            end
        end
    end

end
```

END OF MAIN FUNCTION

```
toc
```

```
Elapsed time is 0.319191 seconds.
```

```
end
```

```
ans =
```

```
x^3*y^3*(36*x^2*y^2 - 90*x^2*y + 60*x^2 - 90*x*y^2 + 225*x*y - 150*x + 60*y^2 - 150*y
+ 100)
```

PLOTTING SUB-FUNCTIONS

```
function [] = comp1dplot(grd,val)

    figure
    set(gca,'FontSize',20)
    plot(grd,val,'ob-')
    hold on
    plot(flipdim(grd,2),val,'or-')
    axis([0 1 0 1])
    xlabel('x')
    ylabel('W_1(x), W_2(x)')
    title('One-Dimensional Weighting Functions')

end

function [] = comp2dplot(h,wv1)

    % Complimentary 2D Weighting Functions

    wv2=flipdim(wv1,2);
    wv3=flipdim(wv1,1);
    wv4=flipdim(flipdim(wv1,2),1);

    [xv,yv]=meshgrid(0:h:1);

    figure
    subplot(2,2,1,'FontSize',20);
    surfc(xv,yv,wv1)
    axis([0 1 0 1 0 1])
    title('W_1(x,y)')
    xlabel('x')
    ylabel('y')
    zlabel('z')

    subplot(2,2,2,'FontSize',20);
    surfc(xv,yv,wv2)
    axis([0 1 0 1 0 1])
    title('W_2(x,y)=W_1(1-x,y)')
    xlabel('x')
    ylabel('y')
    zlabel('z')

    subplot(2,2,3,'FontSize',20);
    surfc(xv,yv,wv3)
    axis([0 1 0 1 0 1])
    title('W_3(x,y)=W_1(x,1-y)')
    xlabel('x')
    ylabel('y')
    zlabel('z')
```

```
    subplot(2,2,4,'FontSize',20);
    surfc(xv,yv,wv4)
    axis([0 1 0 1 0 1])
    title('W_4(x,y)=W_1(1-x,1-y)=W_2(x,1-y)')
    xlabel('x')
    ylabel('y')
    zlabel('z')

end


function [] = sliceplot(h,val)

    % Slices of 3D Weighting Function

    [xv,yv]=meshgrid(0:h:1);

    figure
    subplot(2,3,1,'FontSize',20);
    surfc(xv,yv,val(:,:,1))
    axis([0 1 0 1 0 1])
    title('z = 0.0')
    caxis([0 1])

    subplot(2,3,2,'FontSize',20);
    surfc(xv,yv,val(:,:,3))
    axis([0 1 0 1 0 1])
    title('z = 0.2')
    caxis([0 1])

    subplot(2,3,3,'FontSize',20);
    surfc(xv,yv,val(:,:,5))
    axis([0 1 0 1 0 1])
    title('z = 0.4')
    caxis([0 1])

    subplot(2,3,4,'FontSize',20);
    surfc(xv,yv,val(:,:,7))
    axis([0 1 0 1 0 1])
    title('z = 0.6')
    caxis([0 1])

    subplot(2,3,5,'FontSize',20);
    surfc(xv,yv,val(:,:,9))
    axis([0 1 0 1 0 1])
    title('z = 0.8')
    caxis([0 1])

    subplot(2,3,6,'FontSize',20);
    surfc(xv,yv,val(:,:,11))
    axis([0 1 0 1 0 1])
    title('z = 1.0')
    caxis([0 1])

end
```

Appendix C: 1D Unsteady Burgers' Solver

1D Unsteady Burgers' Solver: This MATLAB code solves the time-varying Burgers' equation. The solver can be run in explicit, implicit, or semi-implicit (Crank-Nicolson) modes. A flag is included for switching between shock coalescence and pulse decay cases. Source terms can be included for running the nearby problem.

Contents


```
% UNSTEADY BURGERS' EQUATION SOLVER
% WRITTEN BY: Matt Kurzen @ Virginia Tech
% CURRENT VERSION: 1
% PREVIOUS VERSIONS: 0 [June 2009]

close all
clear all
```

INPUT PARAMETERS

```
% discretization scheme flag (0=explicit, 1/2=Crank-Nicolson, 1=fully implicit)
theta=1/2;

% flag for unsteady case (1=Shock Coalescence, 2=Pulse Decay)
caseflag=2;

% Reynolds Number
Re=64;

% bar initial time
tbarinit=0.2;

% bar final time
tbarfinal=1;

% number of grid points
```

```
imax=257;

% number of time steps +1
tmax=257;

% output file name
fid='test.txt';

% file path
filepath= '2DMNP/burgersdata/';

% file id beginning  (string for future naming)
%fileid= 'burgers_coal_n65_z16_o3_c3';
fileid= 'burgers_pulse_n257_z64_o3_c3';

% MNP toggle
MNPflag=0;
```

CALCULATED PARAMETERS

```
% scaling factor
alpha=Re/2;

% reference length
L=8;

% reference velocity
uref=2;

% kinematic viscosity
nu=uref*L/Re;

% bar spatial-step
dxbar=L/(imax-1);

% bar spatial-grid
for i=1:imax
    xbar(i,1)=-L/2+(i-1)/(imax-1)*L;
end

% prime spatial-grid
xp=xbar*alpha/L;

% bar time-step
dtbar=(tbarfinal-tbarinit)/(tmax-1);

% bar time-grid
for j=1:tmax
    tbar(j,1)=tbarinit+(j-1)/(tmax-1)*(tbarfinal-tbarinit);
end

% prime time-grid
tp=tbar*alpha^2*nu/L^2;
```

METRICS

```
xsix=1/dxbar*ones(imax,tmax);
etat=1/dtbar*ones(imax,tmax);

xsit=zeros(imax,tmax);
etax=zeros(imax,tmax);
```

SOURCE TERM

```
if MNPflag==0;
    S=zeros(imax,tmax);
elseif MNPflag==1;
    load(strcat(filepath,fileid,'_','src.mat'))
end
```

EXACT MNP SOLUTION

```
if MNPflag==1
    load(strcat(filepath,fileid,'_','MNPex.mat'))
end
```

PRE-ALLOCATE

```
% pre-allocate vectors and matrices
ubar=zeros(imax,1);
ubarnewk=zeros(imax,1);
ubarnewknew=zeros(imax,1);

tdm=zeros(imax,imax);
rhs=zeros(imax,1);
```

INITIAL CONDITIONS

```
for i=1:imax
    if MNPflag==0
        if caseflag==1
            ubar(i,1)= alpha * (nu/L) * ( (-2*sinh(xp(i))) / (cosh(xp(i)) + exp(-
tp(1))) );
        elseif caseflag==2
            ubar(i,1)= alpha * (nu/L) * ( (xp(i)/tp(1)) /
(1+tp(1)^(1/2)*exp(xp(i)^2/(4*tp(1)))) );
        end
    elseif MNPflag==1
        ubar(i,1)= u_MNPex(i,1);
    end
end


%
ubarnewk=ubar;
ubarnewknew=ubar;


% write initial conditions to file
dlmwrite(fid,ubar','delimiter', ' ','precision', '%.8e')
hist(:,1)=ubar;

% calculate discretization error
errorhist(:,1)=ubar-ubar;

% NEW NEW NEW
ubarhist(:,1)=ubar;
ubarexacthist(:,1)=ubar;
```

MAIN LOOP

```
% set inner iteration counter
itercount=0;

% outer iteration
for j=2:tmax
```

BOUNDARY CONDITIONS

```
    % value at boundaries
    if MNPflag==0
        if caseflag==1
            ubarbnd(1,1)=   alpha * (nu/L) * (-2*sinh(xp(1)   )/(cosh(xp(1)   )+exp(-
tp(j))));
            ubarbnd(imax,1)=alpha * (nu/L) * (-2*sinh(xp(imax))/(cosh(xp(imax))+exp(-
tp(j))));
        elseif caseflag==2
            ubarbnd(1,1)=   alpha * (nu/L) * ( (xp(1   )/tp(j)) /
(1+tp(j)^(1/2)*exp(xp(1   )^2/(4*tp(j))) );
            ubarbnd(imax,1)=alpha * (nu/L) * ( (xp(imax)/tp(j)) /
(1+tp(j)^(1/2)*exp(xp(imax)^2/(4*tp(j))) );
        end
    elseif MNPflag==1
        ubarbnd(1,1)=u_MNPex(1,j);
        ubarbnd(imax,1)=u_MNPex(imax,j);
    end

    % right-hand-side at boundaries
    rhs(1   ,1)=ubarbnd(1,1);
    rhs(imax   ,1)=ubarbnd(imax,1);
    %u(1   ,1)=ubnd(1,1);
    %u(imax   ,1)=ubnd(imax,1);
    %%%
```

INNER ITERATION

```
         tol=10^-8;
        normres=tol+1;

        while abs(normres)>tol

            % update unewk
            ubarnewk=ubarnewknew;

            % generate diagonal vectors
            for i=2:imax-1
                % lower diagonal
                A(i,1)=theta*( (-ubarnewk(i)/(2*dxbar)) - (nu/(dxbar^2)) );
                % middle diagonal
                B(i,1)=theta*(2*nu)/(dxbar^2) + 1/dtbar;
                % upper diagonal
                C(i,1)=theta*( ( ubarnewk(i)/(2*dxbar)) - (nu/(dxbar^2)) );
            end

            % set tdm boundary conditions
            A(1)=0; B(1)=1; C(1)=0;
            A(imax)=0; B(imax)=1; C(imax)=0;

            % create tri-diagonal matrix
            tdm=diag(A(2:imax),-1) + diag(B(1:imax)) + diag(C(1:imax-1),1);

            % create right-hand-side vector
            for i=2:imax-1
                %rhs(i,1)= ubar(i)/dtbar - (1-theta)*(ubar(i)*(ubar(i+1)-ubar(i-
1))/(2*dxbar)-nu*(ubar(i+1)-2*ubar(i)+ubar(i-1))/(dxbar^2)) + S(i,j);
                rhs(i,1)= ubar(i)/dtbar - (1-theta)*(ubar(i)*(ubar(i+1)-ubar(i-
1))/(2*dxbar)-nu*(ubar(i+1)-2*ubar(i)+ubar(i-1))/(dxbar^2)) + (1-theta)*S(i,j-1) +
(theta)*S(i,j);
            end

            % solve for unewknew
```

```
            ubarnewknew=tdm\rhs;

            % residual
            for i=2:imax-1
                %res(i,1)=(ubarnewknew(i)-ubar(i))/dtbar + (1-
theta)*(ubar(i)*(ubar(i+1)-ubar(i-1))/(2*dxbar)-nu*(ubar(i+1)-2*ubar(i)+ubar(i-
1))/(dxbar^2)) + (theta)*(ubarnewknew(i)*(ubarnewknew(i+1)-ubarnewknew(i-
1))/(2*dxbar)-nu*(ubarnewknew(i+1)-2*ubarnewknew(i)+ubarnewknew(i-1))/(dxbar^2)) -
S(i,j);
                res(i,1)=(ubarnewknew(i)-ubar(i))/dtbar + (1-
theta)*(ubar(i)*(ubar(i+1)-ubar(i-1))/(2*dxbar)-nu*(ubar(i+1)-2*ubar(i)+ubar(i-
1))/(dxbar^2)) + (theta)*(ubarnewknew(i)*(ubarnewknew(i+1)-ubarnewknew(i-
1))/(2*dxbar)-nu*(ubarnewknew(i+1)-2*ubarnewknew(i)+ubarnewknew(i-1))/(dxbar^2)) - (1-
theta)*S(i,j-1) - (theta)*S(i,j);
            end

            % L2 norm of residual
            normres=sqrt(sum(res(2:imax-1).^2)/(imax-2));

            % update iteration counter
            itercount=itercount+1;
        end

    % update
    for i=2:imax-1
        ubarnewknew(i,1)=ubar(i)-dtbar*( (1-theta)*(ubar(i)*(ubar(i+1)-ubar(i-
1))/(2*dxbar)-nu*(ubar(i+1)-2*ubar(i)+ubar(i-1))/(dxbar^2)) +
(theta)*(ubarnewknew(i)*(ubarnewknew(i+1)-ubarnewknew(i-1))/(2*dxbar)-
nu*(ubarnewknew(i+1)-2*ubarnewknew(i)+ubarnewknew(i-1))/(dxbar^2)) - (1-theta)*S(i,j-
1) - (theta)*S(i,j) );
    end
    ubar=ubarnewknew;
    %u(1   ,1)=ubnd(1,1);
    %u(imax   ,1)=ubnd(imax,1);
    %%%

    % write solution to file at every time-step
    dlmwrite(fid,ubar','delimiter', ' ', '-append','precision', '%.8e')
    ubarhist(:,j)=ubar;

    if MNPflag==0
        % generate exact solution at current timestep
        for i=1:imax
            if caseflag==1
                ubarexact(i,1)=alpha * (nu/L) * (-2*sinh(xp(i))/(cosh(xp(i))+exp(-
tp(j))));
            elseif caseflag==2
                ubarexact(i,1)=alpha * (nu/L) * (xp(i)/tp(j)) / ( 1 + tp(j)^(1/2) *
exp( xp(i)^2 / (4*tp(j)) ) );
            end
        end
        ubarexacthist(:,j)=ubarexact;
    end
end
```

EXACT SOLUTION

```
if MNPflag==0
    % calculate discretization error
    finalerror=ubarhist(:,tmax)-ubarexacthist(:,tmax);

    errorhist=ubarexacthist-ubarhist;
end
```

```
% new save
if MNPflag==0
    output=[ repmat(xbar,tmax,1) repmat(tbar,imax,1)
reshape(ubarexacthist,imax*tmax,1) reshape(ubarhist,imax*tmax,1)...
             reshape(errorhist,imax*tmax,1) reshape(xsix,imax*tmax,1)
reshape(etat,imax*tmax,1)];

     if caseflag==1
        % write output

fid=fopen(strcat('2DMNP/burgersdata/','burgers_coal.',num2str(imax),'x',num2str(tmax),
'.OPoutput.dat'),'w');
        fprintf(fid,'testoutput\n\n');
        fclose(fid);

dlmwrite(strcat('2DMNP/burgersdata/','burgers_coal.',num2str(imax),'x',num2str(tmax),'
.OPoutput.dat'), output ,'delimiter', ' ', 'precision', '%22.15e', '-append');

        % and with no header

dlmwrite(strcat('2DMNP/burgersdata/','burgers_coal.',num2str(imax),'x',num2str(tmax),'
.OPoutput.noheader.dat'), output ,'delimiter', ' ', 'precision', '%22.15e');
     elseif caseflag==2

        % write output

fid=fopen(strcat('2DMNP/burgersdata/','burgers_pulse.',num2str(imax),'x',num2str(tmax)
,'.OPoutput.dat'),'w');
        fprintf(fid,'testoutput\n\n');
        fclose(fid);

dlmwrite(strcat('2DMNP/burgersdata/','burgers_pulse.',num2str(imax),'x',num2str(tmax),
'.OPoutput.dat'), output ,'delimiter', ' ', 'precision', '%22.15e', '-append');

        % and with no header

dlmwrite(strcat('2DMNP/burgersdata/','burgers_pulse.',num2str(imax),'x',num2str(tmax),
'.OPoutput.noheader.dat'), output ,'delimiter', ' ', 'precision', '%22.15e');
     end

% allDE
elseif MNPflag==1

    % nearby problem discretization error
    DE_NP_u=ubarhist-u_MNPex;

    % append nearby problem discretization error to ALLDE tecplot file
    % note that header data will be correct after appending
    dlmwrite(strcat(filepath,fileid,'_TECPLOTcc_ALLDE.dat'),...
        [reshape(DE_NP_u  ,(imax)*(tmax),1) ],...
        'delimiter', ' ', 'precision', '%22.15e', '-append');
end
```

Appendix D: MNP Script

MNP Script: This is a MATLAB script file which takes input from a 2D Euler numerical solution and generates the curve fits and source terms. It does so by calling a least squares fitting code as well as another code which evaluates the curve fits.

Contents


```
% Master Script File: 2D MNP

% POSTFIX NAMING CONVENTIONS:
% _ex = exact     solution to original problem
% _ns = numerical solution to original problem
%     = exact     solution to nearby   problem (curve fit)
% _np = numerical solution to nearby   problem

clear all
close all

tic
```

INPUTS

```
% converts *.dat to *.mat as needed for LWCF2D input

% file path
filepath= '2DMNP/data/';


%{
% NEW CURVILINEAR MS

% file id beginning (string for future naming)
fileidfront= 'curv2d_new_mms';
% numerical solution
```

```
inputfilename= 'curv2d.129.solution.cntr.noheader';
% grid metrics
metricfilename= 'curv2d.129.metrics.noheader';
% flag for MMS (0=no, 1=yes-->read in MMS src and subtract from NP src)
MMSflag=1;
% MMS src term file
srcfilename='curv2d.129.MNP.MMS.noheader';
% esol
esol=1;

%}


%{
% OLD CURVILINEAR MS

% file id beginning (string for future naming)
fileidfront= 'curv2d_old_mms';
% numerical solution
inputfilename= 'curv2d.old.129.solution.cntr.noheader';
% grid metrics
metricfilename= 'curv2d.old.129.metrics.noheader';
% flag for MMS (0=no, 1=yes-->read in MMS src and subtract from NP src)
MMSflag=1;
% MMS src term file
srcfilename='curv2d.old.129.MNP.MMS.noheader';
% esol
esol=1;

%}


%{
% RINGLEB'S FLOW

% file id beginning  (string for future naming)
fileidfront= 'ringleb25';
% numerical solution
inputfilename= 'R2.5.129.solution.cntr.noheader';
% grid metrics
metricfilename= 'R2.5.129.metrics.noheader';
% flag for MMS (0=no, 1=yes-->read in MMS src and subtract from NP src)
MMSflag=0;
% esol
esol=1;
% exact solution
exactfilename='R2.5.129.exact.cntr.noheader';

%}


%
% AIRFOIL FLOW

% file id beginning  (string for future naming)
fileidfront= 'airfoil';
% numerical solution
inputfilename= 'airfoil.129x257.solution.cntr.noheader';
% grid metrics
metricfilename= 'airfoil.129x257.metrics.noheader';
% flag for MMS (0=no, 1=yes-->read in MMS src and subtract from NP src)
MMSflag=0;
% esol
```

```
esol=0;
% exact solution
%exactfilename='R2.5.129.cntr.exact.noheader'; %!!!!!!!!!!!!!!!!!!!!!!!!!!!!

%


% number of grid points
numptsx=257;
numptsy=129;

% number of zones/regions (# of local LS fits -1)
numregionsx=64;
numregionsy=32;

% order of local LS fits (cannot be greater than 5, due to code limitation)
fitorderx=3;
fitordery=3;

% "order" of continuity, C^k (currently the only valid value is 3)
continuityorder=3;


%%%%%%%%%%% CHECK LINE 78ish for var loc%%%%%%%%%%%%%%%%%%%%
```

READ IN NUMERICAL SOLUTION DATA

```
fileid=strcat(fileidfront,'_n',num2str(numptsx),'_z',num2str(numregionsx),'_o',num2str
(fitorderx),'_c',num2str(continuityorder))

% cell-centered grid data
griddatacc=dlmread(strcat(filepath,metricfilename,'.dat'),'');
xcc=reshape(griddatacc(:,1),numptsx-1,numptsy-1);
ycc=reshape(griddatacc(:,2),numptsx-1,numptsy-1);

% nodal computational coordinates
[eta,xsi]=meshgrid(1:numptsy,1:numptsx);
xsi=reshape(xsi,numptsx*numptsy,1);
eta=reshape(eta,numptsx*numptsy,1);

% cell-centered computational coordinates
[etacc,xsicc]=meshgrid(1.5:numptsy-0.5,1.5:numptsx-0.5);

% read in data to temporary array
data=dlmread(strcat(filepath,inputfilename,'.dat'),'');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% read in numerical solution to nearby problem
% |COL1 |COL2 |COL3 |COL4 |COL3 |COL4 |
% |i    |j    |rho  |u    |v    |p    |
rho_ns=data(:,3);
u_ns=  data(:,4);
v_ns=  data(:,5);
p_ns=  data(:,6);

% interpolate from cell centers to nodes
[rho_ns_nodal]=intp(rho_ns,numptsx,numptsy);
[u_ns_nodal]=intp(u_ns,numptsx,numptsy);
[v_ns_nodal]=intp(v_ns,numptsx,numptsy);
[p_ns_nodal]=intp(p_ns,numptsx,numptsy);

% reshape column arrays into 2D arrays
rho_ns=reshape(rho_ns,numptsx-1,numptsy-1);
```

```
u_ns=reshape(u_ns,numptsx-1,numptsy-1);
v_ns=reshape(v_ns,numptsx-1,numptsy-1);
p_ns=reshape(p_ns,numptsx-1,numptsy-1);
et_ns=1/(1.4-1)*p_ns./rho_ns+1/2*(u_ns.^2+v_ns.^2);

% reshape 2D arrays into column arrays
rho_ns_nodal=reshape(rho_ns_nodal,numptsx*numptsy,1);
u_ns_nodal=reshape(u_ns_nodal,numptsx*numptsy,1);
v_ns_nodal=reshape(v_ns_nodal,numptsx*numptsy,1);
p_ns_nodal=reshape(p_ns_nodal,numptsx*numptsy,1);

% TECPLOT nodal output

fid=fopen(strcat(filepath,fileid,'_NODALNUMSOL.dat'),'w');
    fprintf(fid,'TITLE = "NODAL Numerical Solution (Interpolated)"\n');
    fprintf(fid,'FILETYPE = FULL\n');
    fprintf(fid,'VARIABLES = "x", "y",');
    fprintf(fid,' "rho_ns_nodal\\\\n[kg/m^3]", "u_ns_nodal\\\\n[m/s]",
"v_ns_nodal\\\\n[m/s]", "p_ns_nodal\\\\n[N/m^2]"\n');
    fprintf(fid,'ZONE I = %f, J = %f, K = 1\n', numptsx, numptsy);
    fprintf(fid,'ZONETYPE = Ordered, DATAPACKING = BLOCK\n');
    fprintf(fid,'DT = (DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE)\n');
    fclose(fid);
    dlmwrite(strcat(filepath,fileid,'_NODALNUMSOL.dat'),...
        [xsi ; eta ;...
         reshape(rho_ns_nodal ,(numptsx)*(numptsy),1) ; reshape(u_ns_nodal
,(numptsx)*(numptsy),1) ;...
         reshape(v_ns_nodal   ,(numptsx)*(numptsy),1) ; reshape(p_ns_nodal
,(numptsx)*(numptsy),1) ],...
        'delimiter', ' ', 'precision', '%22.15e', '-append');


% save numerical solution as input to LS fitting code
%{
% COL1 |COL2 |COL3 |COL4 |COL5 |COL6 |COL7 |COL8 |COL9 |COL10|COL11|COL12|
% x    |y    |z    |rho  |u    |v    |p    |Mach |U1   |U2   |U3   |U4   |
%}

% rho data
data=[xsi eta rho_ns_nodal rho_ns_nodal rho_ns_nodal];
save(strcat(filepath,fileid,'_rho_ns_nodal'),'data');
% u data
data=[xsi eta u_ns_nodal u_ns_nodal u_ns_nodal];
save(strcat(filepath,fileid,'_u_ns_nodal'),'data');
% v data
data=[xsi eta v_ns_nodal v_ns_nodal v_ns_nodal];
save(strcat(filepath,fileid,'_v_ns_nodal'),'data');
% p data
data=[xsi eta p_ns_nodal p_ns_nodal p_ns_nodal];
save(strcat(filepath,fileid,'_p_ns_nodal'),'data');
% et calculation and data
et_ns_nodal=1/(1.4-1)*p_ns_nodal./rho_ns_nodal+1/2*(u_ns_nodal.^2+v_ns_nodal.^2);
data=[xsi eta et_ns_nodal et_ns_nodal et_ns_nodal];
save(strcat(filepath,fileid,'_et_ns_nodal'),'data');
fileid =

airfoil_n257_z64_o3_c3
```

READ IN METRICS

```
% read in data to temporary array
metricdata=dlmread(strcat(filepath,metricfilename,'.dat'),'');
```

```
% COL1 |COL2 |COL3 |COL4 |COL5 |COL6 |COL7 |COL8 |COL9 |COL10|COL11|COL12|
% x    |y    |i    |j    |xxsi |yxsi |xeta |yeta |etax |etay |xsix |xsiy |

etax=  reshape(metricdata(:, 9),numptsx-1,numptsy-1);
etay=  reshape(metricdata(:,10),numptsx-1,numptsy-1);
xsix=  reshape(metricdata(:,11),numptsx-1,numptsy-1);
xsiy=  reshape(metricdata(:,12),numptsx-1,numptsy-1);
J   =  reshape(metricdata(:,13),numptsx-1,numptsy-1);
```

READ IN EXACT SOLUTION AND MMS SOURCE TERM (if applicable)

```
if MMSflag==1
    % read in data to temporary array
    srcdata=dlmread(strcat(filepath,srcfilename,'.dat'),'');

    rho_ex=reshape(srcdata(:,5),numptsx-1,numptsy-1);
    u_ex=reshape(srcdata(:,6),numptsx-1,numptsy-1);
    v_ex=reshape(srcdata(:,7),numptsx-1,numptsy-1);
    p_ex=reshape(srcdata(:,8),numptsx-1,numptsy-1);

    Sm_OP=  reshape(srcdata(:, 13),numptsx-1,numptsy-1);
    Sx_OP=  reshape(srcdata(:, 14),numptsx-1,numptsy-1);
    Sy_OP=  reshape(srcdata(:, 15),numptsx-1,numptsy-1);
    Se_OP=  reshape(srcdata(:, 16),numptsx-1,numptsy-1);
elseif MMSflag==0 && esol==1
    % read in exact solution
    exdata=dlmread(strcat(filepath,exactfilename,'.dat'),'');

    rho_ex=reshape(exdata(:,3),numptsx-1,numptsy-1);
    u_ex=reshape(exdata(:,4),numptsx-1,numptsy-1);
    v_ex=reshape(exdata(:,5),numptsx-1,numptsy-1);
    p_ex=reshape(exdata(:,6),numptsx-1,numptsy-1);
elseif MMSflag==0 && esol==0
    % no exact solution
end
```

LOCAL LS FITS

```
% variable names
filevar1='rho';
filevar2='u';
filevar3='v';
filevar4='p';
filevar5='et';

% input filenames
inputfile1=strcat(filepath,fileid,'_',filevar1,'_ns_nodal');
inputfile2=strcat(filepath,fileid,'_',filevar2,'_ns_nodal');
inputfile3=strcat(filepath,fileid,'_',filevar3,'_ns_nodal');
inputfile4=strcat(filepath,fileid,'_',filevar4,'_ns_nodal');
inputfile5=strcat(filepath,fileid,'_',filevar5,'_ns_nodal');

% run Sinclair fitting code to generate local LS fit coefficients
outputfile1=sinclairmodallinone(inputfile1,filepath,fileid,filevar1,numregionsx,numreg
ionsy,fitorderx,fitordery,continuityorder);
outputfile2=sinclairmodallinone(inputfile2,filepath,fileid,filevar2,numregionsx,numreg
ionsy,fitorderx,fitordery,continuityorder);
outputfile3=sinclairmodallinone(inputfile3,filepath,fileid,filevar3,numregionsx,numreg
ionsy,fitorderx,fitordery,continuityorder);
outputfile4=sinclairmodallinone(inputfile4,filepath,fileid,filevar4,numregionsx,numreg
ionsy,fitorderx,fitordery,continuityorder);
```

```
outputfile5=sinclairmodallinone(inputfile5,filepath,fileid,filevar5,numregionsx,numreg
ionsy,fitorderx,fitordery,continuityorder);

% run Kurzen fitting code to generate final weighted fit coefficients cell centers
[rho,drhodxsi,drhodeta,rho_xsiface,rho_etaface]=finalfitall(strcat(filepath,outputfile
1),numptsx,numptsy);
[u,dudxsi,dudeta,u_xsiface,u_etaface]=
finalfitall(strcat(filepath,outputfile2),numptsx,numptsy);
[v,dvdxsi,dvdeta,v_xsiface,v_etaface]=
finalfitall(strcat(filepath,outputfile3),numptsx,numptsy);
[p,dpdxsi,dpdeta,p_xsiface,p_etaface]=
finalfitall(strcat(filepath,outputfile4),numptsx,numptsy);
[et,detdxsi,detdeta,et_xsiface,et_etaface]=
finalfitall(strcat(filepath,outputfile5),numptsx,numptsy);

% save fit and fit derivative arrays
save( strcat(filepath,fileid,'_','allfits.mat'), 'rho', 'drhodxsi',...
    'drhodeta', 'u', 'dudxsi', 'dudeta', 'v', 'dvdxsi', 'dvdeta', 'p',...
    'dpdxsi', 'dpdeta', 'et', 'detdxsi', 'detdeta' )
```

GENERATE SOURCE TERMS

```
% INPUT: | rho, u, v, p
%        | drhodxsi, dudxsi, dvdxsi, dpdxsi
%        | drhodeta, dudeta, dvdeta, dpdeta
%        | et, detdxsi, detdeta
%        | xsix, xsiy, etax, etay
%        | J

% physical source terms
Sm_NP=   xsix .* (drhodxsi.*u + rho.*dudxsi)...
    + etax .* (drhodeta.*u + rho.*dudeta)...
    + xsiy .* (drhodxsi.*v + rho.*dvdxsi)...
    + etay .* (drhodeta.*v + rho.*dvdeta) ;

Sx_NP=   xsix .* ( drhodxsi.*u.*u + rho.*( dudxsi.*u + u.*dudxsi ) + dpdxsi )...
    + etax .* ( drhodeta.*u.*u + rho.*( dudeta.*u + u.*dudeta ) + dpdeta )...
    + xsiy .* ( drhodxsi.*u.*v + rho.*( dudxsi.*v + u.*dvdxsi ) )...
    + etay .* ( drhodeta.*u.*v + rho.*( dudeta.*v + u.*dvdeta ) ) ;

Sy_NP=   xsix .* ( drhodxsi.*u.*v + rho.*( dudxsi.*v + u.*dvdxsi ) )...
    + etax .* ( drhodeta.*u.*v + rho.*( dudeta.*v + u.*dvdeta ) )...
    + xsiy .* ( drhodxsi.*v.*v + rho.*( dvdxsi.*v + v.*dvdxsi ) + dpdxsi )...
    + etay .* ( drhodeta.*v.*v + rho.*( dvdeta.*v + v.*dvdeta ) + dpdeta ) ;

Se_NP=   xsix .* ( drhodxsi.*u.*et + rho.*( dudxsi.*et + u.*detdxsi ) + dpdxsi.*u +
p.*dudxsi )...
    + etax .* ( drhodeta.*u.*et + rho.*( dudeta.*et + u.*detdeta ) + dpdeta.*u +
p.*dudeta )...
    + xsiy .* ( drhodxsi.*v.*et + rho.*( dvdxsi.*et + v.*detdxsi ) + dpdxsi.*v +
p.*dvdxsi )...
    + etay .* ( drhodeta.*v.*et + rho.*( dvdeta.*et + v.*detdeta ) + dpdeta.*v +
p.*dvdeta ) ;

% computational source terms
Sm1_NP=Sm_NP./J;
Sx1_NP=Sx_NP./J;
Sy1_NP=Sy_NP./J;
Se1_NP=Se_NP./J;

% save computational source terms
save( strcat(filepath,fileid,'_','metricsandsrc.mat'), 'xsix', 'xsiy', 'etax', 'etay',
'J', 'Sm1_NP', 'Sx1_NP', 'Sy1_NP', 'Se1_NP' )
```

CURVE FIT ERRORS (cellcentered)

```
% calculate errors
cfe_rho=rho-rho_ns;
cfe_u=u-u_ns;
cfe_v=v-v_ns;
cfe_p=p-p_ns;
cfe_et=et-et_ns;

% calculate norms
[cfe_rho_Linf,cfe_rho_L2,cfe_rho_L1] = norms(cfe_rho,numptsx-1,numptsy-1);
[cfe_u_Linf,cfe_u_L2,cfe_u_L1] = norms(cfe_u,numptsx-1,numptsy-1);
[cfe_v_Linf,cfe_v_L2,cfe_v_L1] = norms(cfe_v,numptsx-1,numptsy-1);
[cfe_p_Linf,cfe_p_L2,cfe_p_L1] = norms(cfe_p,numptsx-1,numptsy-1);
[cfe_et_Linf,cfe_et_L2,cfe_et_L1] = norms(cfe_et,numptsx-1,numptsy-1);
```

SOURCE TERMS AND ERRORS

```
% subtract MMS source term if neccesary
if MMSflag==1
    Sm_MNP=Sm_NP-Sm_OP;
    Sx_MNP=Sx_NP-Sx_OP;
    Sy_MNP=Sy_NP-Sy_OP;
    Se_MNP=Se_NP-Se_OP;
elseif MMSflag==0
    Sm_MNP=Sm_NP;
    Sx_MNP=Sx_NP;
    Sy_MNP=Sy_NP;
    Se_MNP=Se_NP;
end

% calculate norms
[Sm_MNP_Linf,Sm_MNP_L2,Sm_MNP_L1] = norms(Sm_MNP,numptsx-1,numptsy-1);
[Sx_MNP_Linf,Sx_MNP_L2,Sx_MNP_L1] = norms(Sx_MNP,numptsx-1,numptsy-1);
[Sy_MNP_Linf,Sy_MNP_L2,Sy_MNP_L1] = norms(Sy_MNP,numptsx-1,numptsy-1);
[Se_MNP_Linf,Se_MNP_L2,Se_MNP_L1] = norms(Se_MNP,numptsx-1,numptsy-1);


% write data to file
fid=fopen(strcat(filepath,fileid,'_ALLnorms.dat'),'w');
fprintf(fid,'Norms of Curve Fit Error and Source Terms\n\n');

fprintf(fid,'cfe_rho_Linf = %.4e\n', cfe_rho_Linf);
fprintf(fid,'cfe_rho_L2 = %.4e\n', cfe_rho_L2);
fprintf(fid,'cfe_rho_L1 = %.4e\n\n', cfe_rho_L1);

fprintf(fid,'cfe_u_Linf = %.4e\n', cfe_u_Linf);
fprintf(fid,'cfe_u_L2 = %.4e\n', cfe_u_L2);
fprintf(fid,'cfe_u_L1 = %.4e\n\n', cfe_u_L1);

fprintf(fid,'cfe_v_Linf = %.4e\n', cfe_v_Linf);
fprintf(fid,'cfe_v_L2 = %.4e\n', cfe_v_L2);
fprintf(fid,'cfe_v_L1 = %.4e\n\n', cfe_v_L1);

fprintf(fid,'cfe_p_Linf = %.4e\n', cfe_p_Linf);
fprintf(fid,'cfe_p_L2 = %.4e\n', cfe_p_L2);
fprintf(fid,'cfe_p_L1 = %.4e\n\n', cfe_p_L1);

fprintf(fid,'cfe_et_Linf = %.4e\n', cfe_et_Linf);
fprintf(fid,'cfe_et_L2 = %.4e\n', cfe_et_L2);
fprintf(fid,'cfe_et_L1 = %.4e\n\n', cfe_et_L1);

fprintf(fid,'Sm_MNP_Linf = %.4e\n', Sm_MNP_Linf);
fprintf(fid,'Sm_MNP_L2 = %.4e\n', Sm_MNP_L2);
```

```
fprintf(fid,'Sm_MNP_L1 = %.4e\n\n', Sm_MNP_L1);

fprintf(fid,'Sx_MNP_Linf = %.4e\n', Sx_MNP_Linf);
fprintf(fid,'Sx_MNP_L2 = %.4e\n', Sx_MNP_L2);
fprintf(fid,'Sx_MNP_L1 = %.4e\n\n', Sx_MNP_L1);

fprintf(fid,'Sy_MNP_Linf = %.4e\n', Sy_MNP_Linf);
fprintf(fid,'Sy_MNP_L2 = %.4e\n', Sy_MNP_L2);
fprintf(fid,'Sy_MNP_L1 = %.4e\n\n', Sy_MNP_L1);

fprintf(fid,'Se_MNP_Linf = %.4e\n', Se_MNP_Linf);
fprintf(fid,'Se_MNP_L2 = %.4e\n', Se_MNP_L2);
fprintf(fid,'Se_MNP_L1 = %.4e\n\n', Se_MNP_L1);


fprintf(fid,'\n#ptsx, #ptsy, orderx, ordery, #zonesx, #zonesy, #ptsperzonex,
#ptsperzoney, cfe_rho_Linf, cfe_rho_L2, ..., cfe_et_L1, Sm_MNP_Linf, Sm_MNP_L2, ...,
Se_MNP_L1\n');
fprintf(fid,'%3.0f %3.0f %3.0f %3.0f %3.0f %3.0f %3.0f %3.0f %.4e %.4e %.4e %.4e %.4e
%.4e %.4e %.4e %.4e %.4e %.4e %.4e %.4e %.4e %.4e %.4e %.4e %.4e %.4e %.4e %.4e
%.4e %.4e %.4e %.4e %.4e', numptsx, numptsy, fitorderx, fitordery, numregionsx,
numregionsy, (numptsx-1)/numregionsx+1, (numptsy-1)/numregionsy+1, cfe_rho_Linf,
cfe_rho_L2, cfe_rho_L1,  cfe_u_Linf, cfe_u_L2, cfe_u_L1, cfe_v_Linf, cfe_v_L2,
cfe_v_L1, cfe_p_Linf, cfe_p_L2, cfe_p_L1, cfe_et_Linf, cfe_et_L2, cfe_et_L1,
Sm_MNP_Linf, Sm_MNP_L2, Sm_MNP_L1, Sx_MNP_Linf, Sx_MNP_L2, Sx_MNP_L1, Sy_MNP_Linf,
Sy_MNP_L2, Sy_MNP_L1, Se_MNP_Linf, Se_MNP_L2, Se_MNP_L1);

fclose(fid);
```

DISCRETIZATION ERROR

```
if esol==1;
% calculate discretization error in original problem
DE_OP_rho=rho_ns-rho_ex;
DE_OP_u=u_ns-u_ex;
DE_OP_v=v_ns-v_ex;
DE_OP_p=p_ns-p_ex;
end

% norms...
```

WRITE OUT *.dat FILES

```
% MNP source terms
fid=fopen(strcat(filepath,fileid,'_srcterms.dat'),'w');
fprintf(fid,'| COL1 | COL2 | COL3 | COL4 | COL5 | COL6 |\n| xsi  | eta  | Sm1_NP  |
Sx1_NP  | Sy1_NP  | Se1_NP  |\n');
fclose(fid);
dlmwrite(strcat(filepath,fileid,'_srcterms.dat'),[reshape(xsicc,(numptsx-1)*(numptsy-
1),1) reshape(etacc,(numptsx-1)*(numptsy-1),1) reshape(Sm1_NP,(numptsx-1)*(numptsy-
1),1) reshape(Sx1_NP,(numptsx-1)*(numptsy-1),1) reshape(Sy1_NP,(numptsx-1)*(numptsy-
1),1) reshape(Se1_NP,(numptsx-1)*(numptsy-1),1) ], 'delimiter', ' ', 'precision',
'%22.15e', '-append');

% calculate face centered coordinate arrays
[eta,xsi]=meshgrid(1:numptsx);
[etacc,xsicc]=meshgrid(1.5:numptsy-0.5,1.5:numptsx-0.5);
xsi_xsiface=xsi(1:numptsx,1:numptsy-1);
eta_xsiface=[etacc;etacc(1,:)];
xsi_etaface=[xsicc,xsicc(:,1)];
eta_etaface=eta(1:numptsx-1,1:numptsy);

% xsi-face centers (xsi = integer)
```

```
fid=fopen(strcat(filepath,fileid,'_xsibndconds.dat'),'w');
fprintf(fid,'| COL1 | COL2 | COL3 | COL4 | COL5 | COL6 |\n| xsi  | eta  | rho  | u
| v    | p    |\n');
fclose(fid);
dlmwrite(strcat(filepath,fileid,'_xsibndconds.dat'),[
reshape(xsi_xsiface,(numptsx)*(numptsy-1),1) reshape(eta_xsiface,(numptsx)*(numptsy-
1),1)...

reshape(rho_xsiface,(numptsx)*(numptsy-1),1) reshape(u_xsiface,(numptsx)*(numptsy-
1),1)...

reshape(v_xsiface,(numptsx)*(numptsy-1),1) reshape(p_xsiface,(numptsx)*(numptsy-
1),1)],...
                                                    'delimiter', ' ', 'precision',
'%22.15e', '-append');

% eta-face centers (eta = integer)
fid=fopen(strcat(filepath,fileid,'_etabndconds.dat'),'w');
fprintf(fid,'| COL1 | COL2 | COL3 | COL4 | COL5 | COL6 |\n| xsi  | eta  | rho  | u
| v    | p    |\n');
fclose(fid);
dlmwrite(strcat(filepath,fileid,'_etabndconds.dat'),[ reshape(xsi_etaface,(numptsx-
1)*(numptsy),1) reshape(eta_etaface,(numptsx-1)*(numptsy),1)...
                                                    reshape(rho_etaface,(numptsx-
1)*(numptsy),1) reshape(u_etaface,(numptsx-1)*(numptsy),1)...
                                                    reshape(v_etaface,(numptsx-
1)*(numptsy),1) reshape(p_etaface,(numptsx-1)*(numptsy),1)],...
                                                    'delimiter', ' ', 'precision',
'%22.15e', '-append');



if esol==1
    % write tecplot output file for contour plots (WITH OP DE)
    fid=fopen(strcat(filepath,fileid,'_TECPLOTcc.dat'),'w');
    fprintf(fid,'TITLE = "Sample Tecplot File"\n');
    fprintf(fid,'FILETYPE = FULL\n');
    fprintf(fid,'VARIABLES = "x", "y",');
    fprintf(fid,' "rho_ns\\\\n[kg/m^3]", "u_ns\\\\n[m/s]", "v_ns\\\\n[m/s]",
"p_ns\\\\n[N/m^2]", "et_ns\\\\n[m^2/s^2]",');
    fprintf(fid,' "rho_cf\\\\n[kg/m^3]", "u_cf\\\\n[m/s]", "v_cf\\\\n[m/s]",
"p_cf\\\\n[N/m^2]", "et_cf\\\\n[m^2/s^2]",');
    fprintf(fid,' "cfe_rho\\\\n[kg/m^3]", "cfe_u\\\\n[m/s]", "cfe_v\\\\n[m/s]",
"cfe_p\\\\n[N/m^2]", "cfe_et\\\\n[m^2/s^2]",');
    fprintf(fid,' "Sm_MNP\\\\n[kg/m^3/s]", "Sx_MNP\\\\n[kg/m^2/s^2]",
"Sy_MNP\\\\n[kg/m^2/s^2]", "Se_MNP\\\\n[kg/m/s^3]",');
    fprintf(fid,' "DE_OP_rho\\\\n[kg/m^3]", "DE_OP_u\\\\n[m/s]", "DE_OP_v\\\\n[m/s]",
"DE_OP_p\\\\n[N/m^2]"\n');
    fprintf(fid,'ZONE I = %f, J = %f, K = 1\n', numptsx-1, numptsy-1);
    fprintf(fid,'ZONETYPE = Ordered, DATAPACKING = BLOCK\n');
    fprintf(fid,'DT = (DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE)\n');
    fclose(fid);
    dlmwrite(strcat(filepath,fileid,'_TECPLOTcc.dat'),...
        [reshape(xcc    ,(numptsx-1)*(numptsy-1),1) ; reshape(ycc   ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(rho_ns ,(numptsx-1)*(numptsy-1),1) ; reshape(u_ns  ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(v_ns   ,(numptsx-1)*(numptsy-1),1) ; reshape(p_ns  ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(et_ns  ,(numptsx-1)*(numptsy-1),1) ; ...
```

```
        reshape(rho    ,(numptsx-1)*(numptsy-1),1) ; reshape(u     ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(v      ,(numptsx-1)*(numptsy-1),1) ; reshape(p     ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(et     ,(numptsx-1)*(numptsy-1),1) ;...
        reshape(cfe_rho,(numptsx-1)*(numptsy-1),1) ; reshape(cfe_u ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(cfe_v  ,(numptsx-1)*(numptsy-1),1) ; reshape(cfe_p ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(cfe_et ,(numptsx-1)*(numptsy-1),1) ;...
        reshape(Sm_MNP    ,(numptsx-1)*(numptsy-1),1) ; reshape(Sx_MNP   ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(Sy_MNP    ,(numptsx-1)*(numptsy-1),1) ; reshape(Se_MNP   ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(DE_OP_rho ,(numptsx-1)*(numptsy-1),1) ; reshape(DE_OP_u  ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(DE_OP_v   ,(numptsx-1)*(numptsy-1),1) ; reshape(DE_OP_p  ,(numptsx-
1)*(numptsy-1),1) ],...
       'delimiter', ' ', 'precision', '%22.15e', '-append');

    % write tecplot output file for contour plots (WITH OP DE AND NP DE)
    fid=fopen(strcat(filepath,fileid,'_TECPLOTccALLDE.dat'),'w');
    fprintf(fid,'TITLE = "Sample Tecplot File"\n');
    fprintf(fid,'FILETYPE = FULL\n');
    fprintf(fid,'VARIABLES = "x", "y",');
    fprintf(fid,' "rho_ns\\\\n[kg/m^3]", "u_ns\\\\n[m/s]", "v_ns\\\\n[m/s]",
"p_ns\\\\n[N/m^2]", "et_ns\\\\n[m^2/s^2]",');
    fprintf(fid,' "rho_cf\\\\n[kg/m^3]", "u_cf\\\\n[m/s]", "v_cf\\\\n[m/s]",
"p_cf\\\\n[N/m^2]", "et_cf\\\\n[m^2/s^2]",');
    fprintf(fid,' "cfe_rho\\\\n[kg/m^3]", "cfe_u\\\\n[m/s]", "cfe_v\\\\n[m/s]",
"cfe_p\\\\n[N/m^2]", "cfe_et\\\\n[m^2/s^2]",');
    fprintf(fid,' "Sm_MNP\\\\n[kg/m^3/s]", "Sx_MNP\\\\n[kg/m^2/s^2]",
"Sy_MNP\\\\n[kg/m^2/s^2]", "Se_MNP\\\\n[kg/m/s^3]",');
    fprintf(fid,' "DE_OP_rho\\\\n[kg/m^3]", "DE_OP_u\\\\n[m/s]", "DE_OP_v\\\\n[m/s]",
"DE_OP_p\\\\n[N/m^2]",');
    fprintf(fid,' "DE_NP_rho\\\\n[kg/m^3]", "DE_NP_u\\\\n[m/s]", "DE_NP_v\\\\n[m/s]",
"DE_NP_p\\\\n[N/m^2]"\n');
    fprintf(fid,'ZONE I = %f, J = %f, K = 1\n', numptsx-1, numptsy-1);
    fprintf(fid,'ZONETYPE = Ordered, DATAPACKING = BLOCK\n');
    fprintf(fid,'DT = (DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE)\n');
    fclose(fid);
    dlmwrite(strcat(filepath,fileid,'_TECPLOTccALLDE.dat'),...
        [reshape(xcc    ,(numptsx-1)*(numptsy-1),1) ; reshape(ycc   ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(rho_ns ,(numptsx-1)*(numptsy-1),1) ; reshape(u_ns  ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(v_ns   ,(numptsx-1)*(numptsy-1),1) ; reshape(p_ns  ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(et_ns  ,(numptsx-1)*(numptsy-1),1) ; ...
        reshape(rho    ,(numptsx-1)*(numptsy-1),1) ; reshape(u     ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(v      ,(numptsx-1)*(numptsy-1),1) ; reshape(p     ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(et     ,(numptsx-1)*(numptsy-1),1) ;...
        reshape(cfe_rho,(numptsx-1)*(numptsy-1),1) ; reshape(cfe_u ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(cfe_v  ,(numptsx-1)*(numptsy-1),1) ; reshape(cfe_p ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(cfe_et ,(numptsx-1)*(numptsy-1),1) ;...
        reshape(Sm_MNP    ,(numptsx-1)*(numptsy-1),1) ; reshape(Sx_MNP   ,(numptsx-
1)*(numptsy-1),1) ;...
```

```
        reshape(Sy_MNP    ,(numptsx-1)*(numptsy-1),1) ; reshape(Se_MNP   ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(DE_OP_rho ,(numptsx-1)*(numptsy-1),1) ; reshape(DE_OP_u  ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(DE_OP_v   ,(numptsx-1)*(numptsy-1),1) ; reshape(DE_OP_p  ,(numptsx-
1)*(numptsy-1),1) ],...
       'delimiter', ' ', 'precision', '%22.15e', '-append');

elseif esol==0
    % write tecplot output file for contour plots (WITH OP DE)
    fid=fopen(strcat(filepath,fileid,'_TECPLOTcc.dat'),'w');
    fprintf(fid,'TITLE = "Sample Tecplot File"\n');
    fprintf(fid,'FILETYPE = FULL\n');
    fprintf(fid,'VARIABLES = "x", "y",');
    fprintf(fid,' "rho_ns\\\\n[kg/m^3]", "u_ns\\\\n[m/s]", "v_ns\\\\n[m/s]",
"p_ns\\\\n[N/m^2]", "et_ns\\\\n[m^2/s^2]",');
    fprintf(fid,' "rho_cf\\\\n[kg/m^3]", "u_cf\\\\n[m/s]", "v_cf\\\\n[m/s]",
"p_cf\\\\n[N/m^2]", "et_cf\\\\n[m^2/s^2]",');
    fprintf(fid,' "cfe_rho\\\\n[kg/m^3]", "cfe_u\\\\n[m/s]", "cfe_v\\\\n[m/s]",
"cfe_p\\\\n[N/m^2]", "cfe_et\\\\n[m^2/s^2]",');
    fprintf(fid,' "Sm_MNP\\\\n[kg/m^3/s]", "Sx_MNP\\\\n[kg/m^2/s^2]",
"Sy_MNP\\\\n[kg/m^2/s^2]", "Se_MNP\\\\n[kg/m/s^3]"\n');
    fprintf(fid,'ZONE I = %f, J = %f, K = 1\n', numptsx-1, numptsy-1);
    fprintf(fid,'ZONETYPE = Ordered, DATAPACKING = BLOCK\n');
    fprintf(fid,'DT = (DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE)\n');
    fclose(fid);
    dlmwrite(strcat(filepath,fileid,'_TECPLOTcc.dat'),...
        [reshape(xcc    ,(numptsx-1)*(numptsy-1),1) ; reshape(ycc   ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(rho_ns ,(numptsx-1)*(numptsy-1),1) ; reshape(u_ns  ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(v_ns   ,(numptsx-1)*(numptsy-1),1) ; reshape(p_ns  ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(et_ns  ,(numptsx-1)*(numptsy-1),1) ; ...
        reshape(rho    ,(numptsx-1)*(numptsy-1),1) ; reshape(u     ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(v      ,(numptsx-1)*(numptsy-1),1) ; reshape(p     ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(et     ,(numptsx-1)*(numptsy-1),1) ;...
        reshape(cfe_rho,(numptsx-1)*(numptsy-1),1) ; reshape(cfe_u ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(cfe_v  ,(numptsx-1)*(numptsy-1),1) ; reshape(cfe_p ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(cfe_et ,(numptsx-1)*(numptsy-1),1) ;...
        reshape(Sm_MNP    ,(numptsx-1)*(numptsy-1),1) ; reshape(Sx_MNP   ,(numptsx-
1)*(numptsy-1),1) ;...
        reshape(Sy_MNP    ,(numptsx-1)*(numptsy-1),1) ; reshape(Se_MNP   ,(numptsx-
1)*(numptsy-1),1)],...
       'delimiter', ' ', 'precision', '%22.15e', '-append');

    % write tecplot output file for contour plots (WITH OP DE AND NP DE)
    fid=fopen(strcat(filepath,fileid,'_TECPLOTccALLDE.dat'),'w');
    fprintf(fid,'TITLE = "Sample Tecplot File"\n');
    fprintf(fid,'FILETYPE = FULL\n');
    fprintf(fid,'VARIABLES = "x", "y",');
    fprintf(fid,' "rho_ns\\\\n[kg/m^3]", "u_ns\\\\n[m/s]", "v_ns\\\\n[m/s]",
"p_ns\\\\n[N/m^2]", "et_ns\\\\n[m^2/s^2]",');
    fprintf(fid,' "rho_cf\\\\n[kg/m^3]", "u_cf\\\\n[m/s]", "v_cf\\\\n[m/s]",
"p_cf\\\\n[N/m^2]", "et_cf\\\\n[m^2/s^2]",');
    fprintf(fid,' "cfe_rho\\\\n[kg/m^3]", "cfe_u\\\\n[m/s]", "cfe_v\\\\n[m/s]",
"cfe_p\\\\n[N/m^2]", "cfe_et\\\\n[m^2/s^2]",');
```

```
    fprintf(fid,' "Sm_MNP\\\\n[kg/m^3/s]", "Sx_MNP\\\\n[kg/m^2/s^2]",
"Sy_MNP\\\\n[kg/m^2/s^2]", "Se_MNP\\\\n[kg/m/s^3]",');
    fprintf(fid,' "DE_NP_rho\\\\n[kg/m^3]", "DE_NP_u\\\\n[m/s]", "DE_NP_v\\\\n[m/s]",
"DE_NP_p\\\\n[N/m^2]"\n');
    fprintf(fid,'ZONE I = %f, J = %f, K = 1\n', numptsx-1, numptsy-1);
    fprintf(fid,'ZONETYPE = Ordered, DATAPACKING = BLOCK\n');
    fprintf(fid,'DT = (DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE)\n');
    fclose(fid);
    dlmwrite(strcat(filepath,fileid,'_TECPLOTccALLDE.dat'),...
        [reshape(xcc    ,(numptsx-1)*(numptsy-1),1) ; reshape(ycc   ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(rho_ns ,(numptsx-1)*(numptsy-1),1) ; reshape(u_ns  ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(v_ns   ,(numptsx-1)*(numptsy-1),1) ; reshape(p_ns  ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(et_ns  ,(numptsx-1)*(numptsy-1),1) ; ...
         reshape(rho    ,(numptsx-1)*(numptsy-1),1) ; reshape(u     ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(v      ,(numptsx-1)*(numptsy-1),1) ; reshape(p     ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(et     ,(numptsx-1)*(numptsy-1),1) ;...
         reshape(cfe_rho,(numptsx-1)*(numptsy-1),1) ; reshape(cfe_u ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(cfe_v  ,(numptsx-1)*(numptsy-1),1) ; reshape(cfe_p ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(cfe_et ,(numptsx-1)*(numptsy-1),1) ;...
         reshape(Sm_MNP    ,(numptsx-1)*(numptsy-1),1) ; reshape(Sx_MNP   ,(numptsx-
1)*(numptsy-1),1) ;...
         reshape(Sy_MNP    ,(numptsx-1)*(numptsy-1),1) ; reshape(Se_MNP   ,(numptsx-
1)*(numptsy-1),1)],...
        'delimiter', ' ', 'precision', '%22.15e', '-append');
end
```

END

```
toc
```

Elapsed time is 704.310606 seconds.

Appendix E: Interpolation

Interpolation: The MATLAB code interpolates cell-centered data to nodes, using second-order accurate methods.

```
function [varintp] = intp(vardata,numptsx,numptsy)

% interpolation of cell centered data to nodal data

varcc=reshape(vardata,numptsx-1,numptsy-1);

varintp=zeros(numptsx,numptsy);

% 4 point averaging for interior nodes
for i=2:numptsx-1
    for j=2:numptsy-1
        varintp(i,j)=( varcc(i-1,j-1) + varcc(i-1,j) + varcc(i,j) + varcc(i,j-1) ) /
4;
    end
end

% xsi faces extrapolation
xsi_min_fc(1,:)=(3*varcc(1        ,:)-varcc(2        ,:))/2;
xsi_max_fc(1,:)=(3*varcc(numptsx-1,:)-varcc(numptsx-2,:))/2;

% eta faces extrapolation
eta_min_fc(:,1)=(3*varcc(:        ,1)-varcc(:        ,2))/2;
eta_max_fc(:,1)=(3*varcc(:,numptsy-1)-varcc(:,numptsy-2))/2;

% xsi faces averaging
xsi_min_n(1,:)=(xsi_min_fc(1,1:numptsy-2)+xsi_min_fc(1,2:numptsy-1))/2;
xsi_max_n(1,:)=(xsi_max_fc(1,1:numptsy-2)+xsi_max_fc(1,2:numptsy-1))/2;

% eta faces averaging
eta_min_n(:,1)=(eta_min_fc(1:numptsx-2,1)+eta_min_fc(2:numptsx-1,1))/2;
eta_max_n(:,1)=(eta_max_fc(1:numptsx-2,1)+eta_max_fc(2:numptsx-1,1))/2;

% add non corner boundary node data
varintp(1      ,2:numptsy-1)=xsi_min_n;
varintp(numptsx,2:numptsy-1)=xsi_max_n;
varintp(2:numptsx-1,1      )=eta_min_n;
varintp(2:numptsx-1,numptsy)=eta_max_n;

% corners (diag.)
varintp(1,1)=(3*varcc(1,1)-varcc(2,2))/2;
varintp(1,numptsy)=(3*varcc(1,numptsy-1)-varcc(2,numptsy-2))/2;
varintp(numptsx,numptsy)=(3*varcc(numptsx-1,numptsy-1)-varcc(numptsx-2,numptsy-2))/2;
varintp(numptsx,1)=(3*varcc(numptsx-1,1)-varcc(numptsx-2,2))/2;

end
```

Appendix F: Norms

Norms: The MATLAB code calculates norms.

```
function [cfe_var_Linf,cfe_var_L2,cfe_var_L1] =
norms(cfe_var,numptsx,numptsy)

% norms

cfe_var_Linf=max(max(abs(cfe_var)));
cfe_var_L2=sqrt( sum(sum(abs(cfe_var).^2)) / ((numptsx-1)*(numptsy-1)) );
cfe_var_L1=sum(sum(abs(cfe_var))) / ((numptsx-1)*(numptsy-1));
```

Appendix G: 2D Final Fit and Evaluation

2D Final Fit and Evaluation: The MATLAB code generates the final fits from the local least squares fits. It also evaluates the fits and derivatives.

Contents


```
function [F_eval, dFdx_eval, dFdy_eval,F_eval_xsibnd, F_eval_etabnd] =
finalfitall(filename,ptsx,ptsy)
```

read in data (local LS fit coefficients)

```
% read in parameters
fid=fopen(filename);
tempdata_1=textscan(fid, '%*s %*s %f %*s %*s %*s %f', 2, 'HeaderLines', 2);
fclose(fid);
tempdata_2=[tempdata_1{1} tempdata_1{2}];

% zone=unit of final fit, region=unit of LS fit (4x4 zones)
numzonesx=tempdata_2(1,1);
numzonesy=tempdata_2(1,2);

fitorderx=tempdata_2(2,1);
fitordery=tempdata_2(2,2);

% data must be manually entered
%ptsx=129;
%ptsy=129;

domainx=ptsx-1; % assumes computational grid
domainy=ptsy-1; %

% calculated values
numregionsx=numzonesx+1;
numregionsy=numzonesy+1;

ptsperzonex=(ptsx-1)/numzonesx+1;
ptsperzoney=(ptsy-1)/numzonesy+1;

cellsperzonex=ptsperzonex-1;
cellsperzoney=ptsperzoney-1;

dxf=domainx/(ptsx-1);
dyf=domainy/(ptsy-1);

zonewidthx=domainx/numzonesx;
```

```
zonewidthy=domainy/numzonesy;

xfshift=zonewidthx;
yfshift=zonewidthy;

dxw=1/cellsperzonex;
dyw=1/cellsperzoney;

% read in region numbering (COL1=x, COL2=y) and coefficients (rest of COLs)
tempdata_3=dlmread(filename,'',10,0);

% region numbering (COL1=x, COL2=y)
%regnum=int32(tempdata_3(:,1:2));
regnum=tempdata_3(:,1:2);

% coefficient data
localLScoeff=tempdata_3(:,3:size(tempdata_3,2));

% 4D array of coefficients (i=xregnum, j=yregnum, m=xpower-1, n=ypower-1)

% pad with zeros
LScoeffs=zeros([numregionsx numregionsy 5+1 5+1]);

%LScoeffs=zeros([numregionsx numregionsy fitorderx+1 fitordery+1]);
for j=1:numregionsy
    for i=1:numregionsx

LScoeffs(i,j,1:fitorderx+1,1:fitordery+1)=reshape(localLScoeff((numregionsx)*(j-
1)+i,:),fitorderx+1,fitordery+1);
    end
end
```

FUNCTIONS

```
% anonymous functions of the final fit and derivatives
% 3rd-order LS fits (x and y), C^3 WF

F=
@(x,y,xscale,yscale,xshifta,yshifta,xshiftb,yshiftb,xshiftc,yshiftc,xshiftd,yshiftd,LS
a,LSb,LSc,LSd) x^4*y^4*(20*x^3 - 70*x^2 + 84*x - 35)*(20*y^3 - 70*y^2 + 84*y -
35)*(LSa(2, 1)*(xshifta + x/xscale) + LSa(1, 2)*(yshifta + y/yscale) + LSa(1, 1) +
LSa(3, 1)*(xshifta + x/xscale)^2 + LSa(4, 1)*(xshifta + x/xscale)^3 + LSa(5,
1)*(xshifta + x/xscale)^4 + LSa(6, 1)*(xshifta + x/xscale)^5 + LSa(1, 3)*(yshifta +
y/yscale)^2 + LSa(1, 4)*(yshifta + y/yscale)^3 + LSa(1, 5)*(yshifta + y/yscale)^4 +
LSa(1, 6)*(yshifta + y/yscale)^5 + LSa(2, 3)*(xshifta + x/xscale)*(yshifta +
y/yscale)^2 + LSa(3, 2)*(xshifta + x/xscale)^2*(yshifta + y/yscale) + LSa(2,
4)*(xshifta + x/xscale)*(yshifta + y/yscale)^3 + LSa(4, 2)*(xshifta +
x/xscale)^3*(yshifta + y/yscale) + LSa(2, 5)*(xshifta + x/xscale)*(yshifta +
y/yscale)^4 + LSa(5, 2)*(xshifta + x/xscale)^4*(yshifta + y/yscale) + LSa(2,
6)*(xshifta + x/xscale)*(yshifta + y/yscale)^5 + LSa(6, 2)*(xshifta +
x/xscale)^5*(yshifta + y/yscale) + LSa(3, 3)*(xshifta + x/xscale)^2*(yshifta +
y/yscale)^2 + LSa(3, 4)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^3 + LSa(4,
3)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^2 + LSa(3, 5)*(xshifta +
x/xscale)^2*(yshifta + y/yscale)^4 + LSa(4, 4)*(xshifta + x/xscale)^3*(yshifta +
y/yscale)^3 + LSa(5, 3)*(xshifta + x/xscale)^4*(yshifta + y/yscale)^2 + LSa(3,
6)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^5 + LSa(4, 5)*(xshifta +
x/xscale)^3*(yshifta + y/yscale)^4 + LSa(5, 4)*(xshifta + x/xscale)^4*(yshifta +
y/yscale)^3 + LSa(6, 3)*(xshifta + x/xscale)^5*(yshifta + y/yscale)^2 + LSa(4,
6)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^5 + LSa(5, 5)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^4 + LSa(6, 4)*(xshifta + x/xscale)^5*(yshifta +
y/yscale)^3 + LSa(5, 6)*(xshifta + x/xscale)^4*(yshifta + y/yscale)^5 + LSa(6,
5)*(xshifta + x/xscale)^5*(yshifta + y/yscale)^4 + LSa(6, 6)*(xshifta +
x/xscale)^5*(yshifta + y/yscale)^5 + LSa(2, 2)*(xshifta + x/xscale)*(yshifta +
```

```
y/yscale)) - y^4*(x - 1)^4*(84*x + 70*(x - 1)^2 + 20*(x - 1)^3 - 49)*(20*y^3 - 70*y^2
+ 84*y - 35)*(LSb(2, 1)*(xshiftb + x/xscale) + LSb(1, 2)*(yshiftb + y/yscale) + LSb(1,
1) + LSb(3, 1)*(xshiftb + x/xscale)^2 + LSb(4, 1)*(xshiftb + x/xscale)^3 + LSb(5,
1)*(xshiftb + x/xscale)^4 + LSb(6, 1)*(xshiftb + x/xscale)^5 + LSb(1, 3)*(yshiftb +
y/yscale)^2 + LSb(1, 4)*(yshiftb + y/yscale)^3 + LSb(1, 5)*(yshiftb + y/yscale)^4 +
LSb(1, 6)*(yshiftb + y/yscale)^5 + LSb(2, 3)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)^2 + LSb(3, 2)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale) + LSb(2,
4)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^3 + LSb(4, 2)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale) + LSb(2, 5)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)^4 + LSb(5, 2)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale) + LSb(2,
6)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^5 + LSb(6, 2)*(xshiftb +
x/xscale)^5*(yshiftb + y/yscale) + LSb(3, 3)*(xshiftb + x/xscale)^2*(yshiftb +
y/yscale)^2 + LSb(3, 4)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^3 + LSb(4,
3)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^2 + LSb(3, 5)*(xshiftb +
x/xscale)^2*(yshiftb + y/yscale)^4 + LSb(4, 4)*(xshiftb + x/xscale)^3*(yshiftb +
y/yscale)^3 + LSb(5, 3)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale)^2 + LSb(3,
6)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^5 + LSb(4, 5)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale)^4 + LSb(5, 4)*(xshiftb + x/xscale)^4*(yshiftb +
y/yscale)^3 + LSb(6, 3)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)^2 + LSb(4,
6)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^5 + LSb(5, 5)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^4 + LSb(6, 4)*(xshiftb + x/xscale)^5*(yshiftb +
y/yscale)^3 + LSb(5, 6)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale)^5 + LSb(6,
5)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)^4 + LSb(6, 6)*(xshiftb +
x/xscale)^5*(yshiftb + y/yscale)^5 + LSb(2, 2)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)) - x^4*(y - 1)^4*(84*y + 70*(y - 1)^2 + 20*(y - 1)^3 - 49)*(20*x^3 - 70*x^2
+ 84*x - 35)*(LSd(2, 1)*(xshiftd + x/xscale) + LSd(1, 2)*(yshiftd + y/yscale) + LSd(1,
1) + LSd(3, 1)*(xshiftd + x/xscale)^2 + LSd(4, 1)*(xshiftd + x/xscale)^3 + LSd(5,
1)*(xshiftd + x/xscale)^4 + LSd(6, 1)*(xshiftd + x/xscale)^5 + LSd(1, 3)*(yshiftd +
y/yscale)^2 + LSd(1, 4)*(yshiftd + y/yscale)^3 + LSd(1, 5)*(yshiftd + y/yscale)^4 +
LSd(1, 6)*(yshiftd + y/yscale)^5 + LSd(2, 3)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^2 + LSd(3, 2)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale) + LSd(2,
4)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^3 + LSd(4, 2)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale) + LSd(2, 5)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^4 + LSd(5, 2)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale) + LSd(2,
6)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^5 + LSd(6, 2)*(xshiftd +
x/xscale)^5*(yshiftd + y/yscale) + LSd(3, 3)*(xshiftd + x/xscale)^2*(yshiftd +
y/yscale)^2 + LSd(3, 4)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^3 + LSd(4,
3)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale)^2 + LSd(3, 5)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale)^4 + LSd(4, 4)*(xshiftd + x/xscale)^3*(yshiftd +
y/yscale)^3 + LSd(5, 3)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale)^2 + LSd(3,
6)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^5 + LSd(4, 5)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale)^4 + LSd(5, 4)*(xshiftd + x/xscale)^4*(yshiftd +
y/yscale)^3 + LSd(6, 3)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)^2 + LSd(4,
6)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale)^5 + LSd(5, 5)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^4 + LSd(6, 4)*(xshiftd + x/xscale)^5*(yshiftd +
y/yscale)^3 + LSd(5, 6)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale)^5 + LSd(6,
5)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)^4 + LSd(6, 6)*(xshiftd +
x/xscale)^5*(yshiftd + y/yscale)^5 + LSd(2, 2)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)) + (x - 1)^4*(y - 1)^4*(84*x + 70*(x - 1)^2 + 20*(x - 1)^3 - 49)*(84*y +
70*(y - 1)^2 + 20*(y - 1)^3 - 49)*(LSc(2, 1)*(xshiftc + x/xscale) + LSc(1, 2)*(yshiftc
+ y/yscale) + LSc(1, 1) + LSc(3, 1)*(xshiftc + x/xscale)^2 + LSc(4, 1)*(xshiftc +
x/xscale)^3 + LSc(5, 1)*(xshiftc + x/xscale)^4 + LSc(6, 1)*(xshiftc + x/xscale)^5 +
LSc(1, 3)*(yshiftc + y/yscale)^2 + LSc(1, 4)*(yshiftc + y/yscale)^3 + LSc(1,
5)*(yshiftc + y/yscale)^4 + LSc(1, 6)*(yshiftc + y/yscale)^5 + LSc(2, 3)*(xshiftc +
x/xscale)*(yshiftc + y/yscale)^2 + LSc(3, 2)*(xshiftc + x/xscale)^2*(yshiftc +
y/yscale) + LSc(2, 4)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^3 + LSc(4, 2)*(xshiftc
+ x/xscale)^3*(yshiftc + y/yscale) + LSc(2, 5)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)^4 + LSc(5, 2)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale) + LSc(2,
6)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^5 + LSc(6, 2)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale) + LSc(3, 3)*(xshiftc + x/xscale)^2*(yshiftc +
y/yscale)^2 + LSc(3, 4)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^3 + LSc(4,
3)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^2 + LSc(3, 5)*(xshiftc +
x/xscale)^2*(yshiftc + y/yscale)^4 + LSc(4, 4)*(xshiftc + x/xscale)^3*(yshiftc +
```

```
y/yscale)^3 + LSc(5, 3)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale)^2 + LSc(3,
6)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^5 + LSc(4, 5)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^4 + LSc(5, 4)*(xshiftc + x/xscale)^4*(yshiftc +
y/yscale)^3 + LSc(6, 3)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)^2 + LSc(4,
6)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^5 + LSc(5, 5)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^4 + LSc(6, 4)*(xshiftc + x/xscale)^5*(yshiftc +
y/yscale)^3 + LSc(5, 6)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale)^5 + LSc(6,
5)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)^4 + LSc(6, 6)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale)^5 + LSc(2, 2)*(xshiftc + x/xscale)*(yshiftc +
y/yscale));

dFdx=
@(x,y,xscale,yscale,xshifta,yshifta,xshiftb,yshiftb,xshiftc,yshiftc,xshiftd,yshiftd,LS
a,LSb,LSc,LSd) (x - 1)^4*(y - 1)^4*(140*x + 60*(x - 1)^2 - 56)*(84*y + 70*(y - 1)^2 +
20*(y - 1)^3 - 49)*(LSc(2, 1)*(xshiftc + x/xscale) + LSc(1, 2)*(yshiftc + y/yscale) +
LSc(1, 1) + LSc(3, 1)*(xshiftc + x/xscale)^2 + LSc(4, 1)*(xshiftc + x/xscale)^3 +
LSc(5, 1)*(xshiftc + x/xscale)^4 + LSc(6, 1)*(xshiftc + x/xscale)^5 + LSc(1,
3)*(yshiftc + y/yscale)^2 + LSc(1, 4)*(yshiftc + y/yscale)^3 + LSc(1, 5)*(yshiftc +
y/yscale)^4 + LSc(1, 6)*(yshiftc + y/yscale)^5 + LSc(2, 3)*(xshiftc +
x/xscale)*(yshiftc + y/yscale)^2 + LSc(3, 2)*(xshiftc + x/xscale)^2*(yshiftc +
y/yscale) + LSc(2, 4)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^3 + LSc(4, 2)*(xshiftc
+ x/xscale)^3*(yshiftc + y/yscale) + LSc(2, 5)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)^4 + LSc(5, 2)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale) + LSc(2,
6)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^5 + LSc(6, 2)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale) + LSc(3, 3)*(xshiftc + x/xscale)^2*(yshiftc +
y/yscale)^2 + LSc(3, 4)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^3 + LSc(4,
3)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^2 + LSc(3, 5)*(xshiftc +
x/xscale)^2*(yshiftc + y/yscale)^4 + LSc(4, 4)*(xshiftc + x/xscale)^3*(yshiftc +
y/yscale)^3 + LSc(5, 3)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale)^2 + LSc(3,
6)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^5 + LSc(4, 5)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^4 + LSc(5, 4)*(xshiftc + x/xscale)^4*(yshiftc +
y/yscale)^3 + LSc(6, 3)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)^2 + LSc(4,
6)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^5 + LSc(5, 5)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^4 + LSc(6, 4)*(xshiftc + x/xscale)^5*(yshiftc +
y/yscale)^3 + LSc(5, 6)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale)^5 + LSc(6,
5)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)^4 + LSc(6, 6)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale)^5 + LSc(2, 2)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)) + x^4*y^4*(60*x^2 - 140*x + 84)*(20*y^3 - 70*y^2 + 84*y - 35)*(LSa(2,
1)*(xshifta + x/xscale) + LSa(1, 2)*(yshifta + y/yscale) + LSa(1, 1) + LSa(3,
1)*(xshifta + x/xscale)^2 + LSa(4, 1)*(xshifta + x/xscale)^3 + LSa(5, 1)*(xshifta +
x/xscale)^4 + LSa(6, 1)*(xshifta + x/xscale)^5 + LSa(1, 3)*(yshifta + y/yscale)^2 +
LSa(1, 4)*(yshifta + y/yscale)^3 + LSa(1, 5)*(yshifta + y/yscale)^4 + LSa(1,
6)*(yshifta + y/yscale)^5 + LSa(2, 3)*(xshifta + x/xscale)*(yshifta + y/yscale)^2 +
LSa(3, 2)*(xshifta + x/xscale)^2*(yshifta + y/yscale) + LSa(2, 4)*(xshifta +
x/xscale)*(yshifta + y/yscale)^3 + LSa(4, 2)*(xshifta + x/xscale)^3*(yshifta +
y/yscale) + LSa(2, 5)*(xshifta + x/xscale)*(yshifta + y/yscale)^4 + LSa(5, 2)*(xshifta
+ x/xscale)^4*(yshifta + y/yscale) + LSa(2, 6)*(xshifta + x/xscale)*(yshifta +
y/yscale)^5 + LSa(6, 2)*(xshifta + x/xscale)^5*(yshifta + y/yscale) + LSa(3,
3)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^2 + LSa(3, 4)*(xshifta +
x/xscale)^2*(yshifta + y/yscale)^3 + LSa(4, 3)*(xshifta + x/xscale)^3*(yshifta +
y/yscale)^2 + LSa(3, 5)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^4 + LSa(4,
4)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^3 + LSa(5, 3)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^2 + LSa(3, 6)*(xshifta + x/xscale)^2*(yshifta +
y/yscale)^5 + LSa(4, 5)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^4 + LSa(5,
4)*(xshifta + x/xscale)^4*(yshifta + y/yscale)^3 + LSa(6, 3)*(xshifta +
x/xscale)^5*(yshifta + y/yscale)^2 + LSa(4, 6)*(xshifta + x/xscale)^3*(yshifta +
y/yscale)^5 + LSa(5, 5)*(xshifta + x/xscale)^4*(yshifta + y/yscale)^4 + LSa(6,
4)*(xshifta + x/xscale)^5*(yshifta + y/yscale)^3 + LSa(5, 6)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^5 + LSa(6, 5)*(xshifta + x/xscale)^5*(yshifta +
y/yscale)^4 + LSa(6, 6)*(xshifta + x/xscale)^5*(yshifta + y/yscale)^5 + LSa(2,
2)*(xshifta + x/xscale)*(yshifta + y/yscale)) - x^4*(y - 1)^4*(60*x^2 - 140*x +
84)*(84*y + 70*(y - 1)^2 + 20*(y - 1)^3 - 49)*(LSd(2, 1)*(xshiftd + x/xscale) + LSd(1,
2)*(yshiftd + y/yscale) + LSd(1, 1) + LSd(3, 1)*(xshiftd + x/xscale)^2 + LSd(4,
```

```
1)*(xshiftd + x/xscale)^3 + LSd(5, 1)*(xshiftd + x/xscale)^4 + LSd(6, 1)*(xshiftd +
x/xscale)^5 + LSd(1, 3)*(yshiftd + y/yscale)^2 + LSd(1, 4)*(yshiftd + y/yscale)^3 +
LSd(1, 5)*(yshiftd + y/yscale)^4 + LSd(1, 6)*(yshiftd + y/yscale)^5 + LSd(2,
3)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^2 + LSd(3, 2)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale) + LSd(2, 4)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^3 + LSd(4, 2)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale) + LSd(2,
5)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^4 + LSd(5, 2)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale) + LSd(2, 6)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^5 + LSd(6, 2)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale) + LSd(3,
3)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^2 + LSd(3, 4)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale)^3 + LSd(4, 3)*(xshiftd + x/xscale)^3*(yshiftd +
y/yscale)^2 + LSd(3, 5)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^4 + LSd(4,
4)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale)^3 + LSd(5, 3)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^2 + LSd(3, 6)*(xshiftd + x/xscale)^2*(yshiftd +
y/yscale)^5 + LSd(4, 5)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale)^4 + LSd(5,
4)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale)^3 + LSd(6, 3)*(xshiftd +
x/xscale)^5*(yshiftd + y/yscale)^2 + LSd(4, 6)*(xshiftd + x/xscale)^3*(yshiftd +
y/yscale)^5 + LSd(5, 5)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale)^4 + LSd(6,
4)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)^3 + LSd(5, 6)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^5 + LSd(6, 5)*(xshiftd + x/xscale)^5*(yshiftd +
y/yscale)^4 + LSd(6, 6)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)^5 + LSd(2,
2)*(xshiftd + x/xscale)*(yshiftd + y/yscale)) + 4*x^3*y^4*(20*x^3 - 70*x^2 + 84*x -
35)*(20*y^3 - 70*y^2 + 84*y - 35)*(LSa(2, 1)*(xshifta + x/xscale) + LSa(1, 2)*(yshifta
+ y/yscale) + LSa(1, 1) + LSa(3, 1)*(xshifta + x/xscale)^2 + LSa(4, 1)*(xshifta +
x/xscale)^3 + LSa(5, 1)*(xshifta + x/xscale)^4 + LSa(6, 1)*(xshifta + x/xscale)^5 +
LSa(1, 3)*(yshifta + y/yscale)^2 + LSa(1, 4)*(yshifta + y/yscale)^3 + LSa(1,
5)*(yshifta + y/yscale)^4 + LSa(1, 6)*(yshifta + y/yscale)^5 + LSa(2, 3)*(xshifta +
x/xscale)*(yshifta + y/yscale)^2 + LSa(3, 2)*(xshifta + x/xscale)^2*(yshifta +
y/yscale) + LSa(2, 4)*(xshifta + x/xscale)*(yshifta + y/yscale)^3 + LSa(4, 2)*(xshifta
+ x/xscale)^3*(yshifta + y/yscale) + LSa(2, 5)*(xshifta + x/xscale)*(yshifta +
y/yscale)^4 + LSa(5, 2)*(xshifta + x/xscale)^4*(yshifta + y/yscale) + LSa(2,
6)*(xshifta + x/xscale)*(yshifta + y/yscale)^5 + LSa(6, 2)*(xshifta +
x/xscale)^5*(yshifta + y/yscale) + LSa(3, 3)*(xshifta + x/xscale)^2*(yshifta +
y/yscale)^2 + LSa(3, 4)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^3 + LSa(4,
3)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^2 + LSa(3, 5)*(xshifta +
x/xscale)^2*(yshifta + y/yscale)^4 + LSa(4, 4)*(xshifta + x/xscale)^3*(yshifta +
y/yscale)^3 + LSa(5, 3)*(xshifta + x/xscale)^4*(yshifta + y/yscale)^2 + LSa(3,
6)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^5 + LSa(4, 5)*(xshifta +
x/xscale)^3*(yshifta + y/yscale)^4 + LSa(5, 4)*(xshifta + x/xscale)^4*(yshifta +
y/yscale)^3 + LSa(6, 3)*(xshifta + x/xscale)^5*(yshifta + y/yscale)^2 + LSa(4,
6)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^5 + LSa(5, 5)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^4 + LSa(6, 4)*(xshifta + x/xscale)^5*(yshifta +
y/yscale)^3 + LSa(5, 6)*(xshifta + x/xscale)^4*(yshifta + y/yscale)^5 + LSa(6,
5)*(xshifta + x/xscale)^5*(yshifta + y/yscale)^4 + LSa(6, 6)*(xshifta +
x/xscale)^5*(yshifta + y/yscale)^5 + LSa(2, 2)*(xshifta + x/xscale)*(yshifta +
y/yscale)) + x^4*y^4*(20*x^3 - 70*x^2 + 84*x - 35)*(20*y^3 - 70*y^2 + 84*y -
35)*(1/xscale*LSa(2, 1) + 2/xscale*LSa(3, 1)*(xshifta + x/xscale) + 1/xscale*LSa(2,
2)*(yshifta + y/yscale) + 3/xscale*LSa(4, 1)*(xshifta + x/xscale)^2 + 4/xscale*LSa(5,
1)*(xshifta + x/xscale)^3 + 5/xscale*LSa(6, 1)*(xshifta + x/xscale)^4 +
1/xscale*LSa(2, 3)*(yshifta + y/yscale)^2 + 1/xscale*LSa(2, 4)*(yshifta + y/yscale)^3
+ 1/xscale*LSa(2, 5)*(yshifta + y/yscale)^4 + 1/xscale*LSa(2, 6)*(yshifta +
y/yscale)^5 + 3/xscale*LSa(4, 3)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^2 +
3/xscale*LSa(4, 4)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^3 + 4/xscale*LSa(5,
3)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^2 + 3/xscale*LSa(4, 5)*(xshifta +
x/xscale)^2*(yshifta + y/yscale)^4 + 4/xscale*LSa(5, 4)*(xshifta +
x/xscale)^3*(yshifta + y/yscale)^3 + 5/xscale*LSa(6, 3)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^2 + 3/xscale*LSa(4, 6)*(xshifta +
x/xscale)^2*(yshifta + y/yscale)^5 + 4/xscale*LSa(5, 5)*(xshifta +
x/xscale)^3*(yshifta + y/yscale)^4 + 5/xscale*LSa(6, 4)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^3 + 4/xscale*LSa(5, 6)*(xshifta +
x/xscale)^3*(yshifta + y/yscale)^5 + 5/xscale*LSa(6, 5)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^4 + 5/xscale*LSa(6, 6)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^5 + 2/xscale*LSa(3, 2)*(xshifta + x/xscale)*(yshifta
```

```
+ y/yscale) + 2/xscale*LSa(3, 3)*(xshifta + x/xscale)*(yshifta + y/yscale)^2 +
3/xscale*LSa(4, 2)*(xshifta + x/xscale)^2*(yshifta + y/yscale) + 2/xscale*LSa(3,
4)*(xshifta + x/xscale)*(yshifta + y/yscale)^3 + 4/xscale*LSa(5, 2)*(xshifta +
x/xscale)^3*(yshifta + y/yscale) + 2/xscale*LSa(3, 5)*(xshifta + x/xscale)*(yshifta +
y/yscale)^4 + 5/xscale*LSa(6, 2)*(xshifta + x/xscale)^4*(yshifta + y/yscale) +
2/xscale*LSa(3, 6)*(xshifta + x/xscale)*(yshifta + y/yscale)^5) - 4*y^4*(x -
1)^3*(84*x + 70*(x - 1)^2 + 20*(x - 1)^3 - 49)*(20*y^3 - 70*y^2 + 84*y - 35)*(LSb(2,
1)*(xshiftb + x/xscale) + LSb(1, 2)*(yshiftb + y/yscale) + LSb(1, 1) + LSb(3,
1)*(xshiftb + x/xscale)^2 + LSb(4, 1)*(xshiftb + x/xscale)^3 + LSb(5, 1)*(xshiftb +
x/xscale)^4 + LSb(6, 1)*(xshiftb + x/xscale)^5 + LSb(1, 3)*(yshiftb + y/yscale)^2 +
LSb(1, 4)*(yshiftb + y/yscale)^3 + LSb(1, 5)*(yshiftb + y/yscale)^4 + LSb(1,
6)*(yshiftb + y/yscale)^5 + LSb(2, 3)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^2 +
LSb(3, 2)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale) + LSb(2, 4)*(xshiftb +
x/xscale)*(yshiftb + y/yscale)^3 + LSb(4, 2)*(xshiftb + x/xscale)^3*(yshiftb +
y/yscale) + LSb(2, 5)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^4 + LSb(5, 2)*(xshiftb
+ x/xscale)^4*(yshiftb + y/yscale) + LSb(2, 6)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)^5 + LSb(6, 2)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale) + LSb(3,
3)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^2 + LSb(3, 4)*(xshiftb +
x/xscale)^2*(yshiftb + y/yscale)^3 + LSb(4, 3)*(xshiftb + x/xscale)^3*(yshiftb +
y/yscale)^2 + LSb(3, 5)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^4 + LSb(4,
4)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^3 + LSb(5, 3)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^2 + LSb(3, 6)*(xshiftb + x/xscale)^2*(yshiftb +
y/yscale)^5 + LSb(4, 5)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^4 + LSb(5,
4)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale)^3 + LSb(6, 3)*(xshiftb +
x/xscale)^5*(yshiftb + y/yscale)^2 + LSb(4, 6)*(xshiftb + x/xscale)^3*(yshiftb +
y/yscale)^5 + LSb(5, 5)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale)^4 + LSb(6,
4)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)^3 + LSb(5, 6)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^5 + LSb(6, 5)*(xshiftb + x/xscale)^5*(yshiftb +
y/yscale)^4 + LSb(6, 6)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)^5 + LSb(2,
2)*(xshiftb + x/xscale)*(yshiftb + y/yscale)) - 4*x^3*(y - 1)^4*(84*y + 70*(y - 1)^2 +
20*(y - 1)^3 - 49)*(20*x^3 - 70*x^2 + 84*x - 35)*(LSd(2, 1)*(xshiftd + x/xscale) +
LSd(1, 2)*(yshiftd + y/yscale) + LSd(1, 1) + LSd(3, 1)*(xshiftd + x/xscale)^2 + LSd(4,
1)*(xshiftd + x/xscale)^3 + LSd(5, 1)*(xshiftd + x/xscale)^4 + LSd(6, 1)*(xshiftd +
x/xscale)^5 + LSd(1, 3)*(yshiftd + y/yscale)^2 + LSd(1, 4)*(yshiftd + y/yscale)^3 +
LSd(1, 5)*(yshiftd + y/yscale)^4 + LSd(1, 6)*(yshiftd + y/yscale)^5 + LSd(2,
3)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^2 + LSd(3, 2)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale) + LSd(2, 4)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^3 + LSd(4, 2)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale) + LSd(2,
5)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^4 + LSd(5, 2)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale) + LSd(2, 6)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^5 + LSd(6, 2)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale) + LSd(3,
3)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^2 + LSd(3, 4)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale)^3 + LSd(4, 3)*(xshiftd + x/xscale)^3*(yshiftd +
y/yscale)^2 + LSd(3, 5)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^4 + LSd(4,
4)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale)^3 + LSd(5, 3)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^2 + LSd(3, 6)*(xshiftd + x/xscale)^2*(yshiftd +
y/yscale)^5 + LSd(4, 5)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale)^4 + LSd(5,
4)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale)^3 + LSd(6, 3)*(xshiftd +
x/xscale)^5*(yshiftd + y/yscale)^2 + LSd(4, 6)*(xshiftd + x/xscale)^3*(yshiftd +
y/yscale)^5 + LSd(5, 5)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale)^4 + LSd(6,
4)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)^3 + LSd(5, 6)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^5 + LSd(6, 5)*(xshiftd + x/xscale)^5*(yshiftd +
y/yscale)^4 + LSd(6, 6)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)^5 + LSd(2,
2)*(xshiftd + x/xscale)*(yshiftd + y/yscale)) - y^4*(x - 1)^4*(84*x + 70*(x - 1)^2 +
20*(x - 1)^3 - 49)*(20*y^3 - 70*y^2 + 84*y - 35)*(1/xscale*LSb(2, 1) + 2/xscale*LSb(3,
1)*(xshiftb + x/xscale) + 1/xscale*LSb(2, 2)*(yshiftb + y/yscale) + 3/xscale*LSb(4,
1)*(xshiftb + x/xscale)^2 + 4/xscale*LSb(5, 1)*(xshiftb + x/xscale)^3 +
5/xscale*LSb(6, 1)*(xshiftb + x/xscale)^4 + 1/xscale*LSb(2, 3)*(yshiftb + y/yscale)^2
+ 1/xscale*LSb(2, 4)*(yshiftb + y/yscale)^3 + 1/xscale*LSb(2, 5)*(yshiftb +
y/yscale)^4 + 1/xscale*LSb(2, 6)*(yshiftb + y/yscale)^5 + 3/xscale*LSb(4, 3)*(xshiftb
+ x/xscale)^2*(yshiftb + y/yscale)^2 + 3/xscale*LSb(4, 4)*(xshiftb +
x/xscale)^2*(yshiftb + y/yscale)^3 + 4/xscale*LSb(5, 3)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale)^2 + 3/xscale*LSb(4, 5)*(xshiftb +
```

```
x/xscale)^2*(yshiftb + y/yscale)^4 + 4/xscale*LSb(5, 4)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale)^3 + 5/xscale*LSb(6, 3)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^2 + 3/xscale*LSb(4, 6)*(xshiftb +
x/xscale)^2*(yshiftb + y/yscale)^5 + 4/xscale*LSb(5, 5)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale)^4 + 5/xscale*LSb(6, 4)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^3 + 4/xscale*LSb(5, 6)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale)^5 + 5/xscale*LSb(6, 5)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^4 + 5/xscale*LSb(6, 6)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^5 + 2/xscale*LSb(3, 2)*(xshiftb + x/xscale)*(yshiftb
+ y/yscale) + 2/xscale*LSb(3, 3)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^2 +
3/xscale*LSb(4, 2)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale) + 2/xscale*LSb(3,
4)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^3 + 4/xscale*LSb(5, 2)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale) + 2/xscale*LSb(3, 5)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)^4 + 5/xscale*LSb(6, 2)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale) +
2/xscale*LSb(3, 6)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^5) - x^4*(y - 1)^4*(84*y
+ 70*(y - 1)^2 + 20*(y - 1)^3 - 49)*(20*x^3 - 70*x^2 + 84*x - 35)*(1/xscale*LSd(2, 1)
+ 2/xscale*LSd(3, 1)*(xshiftd + x/xscale) + 1/xscale*LSd(2, 2)*(yshiftd + y/yscale) +
3/xscale*LSd(4, 1)*(xshiftd + x/xscale)^2 + 4/xscale*LSd(5, 1)*(xshiftd + x/xscale)^3
+ 5/xscale*LSd(6, 1)*(xshiftd + x/xscale)^4 + 1/xscale*LSd(2, 3)*(yshiftd +
y/yscale)^2 + 1/xscale*LSd(2, 4)*(yshiftd + y/yscale)^3 + 1/xscale*LSd(2, 5)*(yshiftd
+ y/yscale)^4 + 1/xscale*LSd(2, 6)*(yshiftd + y/yscale)^5 + 3/xscale*LSd(4,
3)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^2 + 3/xscale*LSd(4, 4)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale)^3 + 4/xscale*LSd(5, 3)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale)^2 + 3/xscale*LSd(4, 5)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale)^4 + 4/xscale*LSd(5, 4)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale)^3 + 5/xscale*LSd(6, 3)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^2 + 3/xscale*LSd(4, 6)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale)^5 + 4/xscale*LSd(5, 5)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale)^4 + 5/xscale*LSd(6, 4)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^3 + 4/xscale*LSd(5, 6)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale)^5 + 5/xscale*LSd(6, 5)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^4 + 5/xscale*LSd(6, 6)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^5 + 2/xscale*LSd(3, 2)*(xshiftd + x/xscale)*(yshiftd
+ y/yscale) + 2/xscale*LSd(3, 3)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^2 +
3/xscale*LSd(4, 2)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale) + 2/xscale*LSd(3,
4)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^3 + 4/xscale*LSd(5, 2)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale) + 2/xscale*LSd(3, 5)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^4 + 5/xscale*LSd(6, 2)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale) +
2/xscale*LSd(3, 6)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^5) + 4*(x - 1)^3*(y -
1)^4*(84*x + 70*(x - 1)^2 + 20*(x - 1)^3 - 49)*(84*y + 70*(y - 1)^2 + 20*(y - 1)^3 -
49)*(LSc(2, 1)*(xshiftc + x/xscale) + LSc(1, 2)*(yshiftc + y/yscale) + LSc(1, 1) +
LSc(3, 1)*(xshiftc + x/xscale)^2 + LSc(4, 1)*(xshiftc + x/xscale)^3 + LSc(5,
1)*(xshiftc + x/xscale)^4 + LSc(6, 1)*(xshiftc + x/xscale)^5 + LSc(1, 3)*(yshiftc +
y/yscale)^2 + LSc(1, 4)*(yshiftc + y/yscale)^3 + LSc(1, 5)*(yshiftc + y/yscale)^4 +
LSc(1, 6)*(yshiftc + y/yscale)^5 + LSc(2, 3)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)^2 + LSc(3, 2)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale) + LSc(2,
4)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^3 + LSc(4, 2)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale) + LSc(2, 5)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)^4 + LSc(5, 2)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale) + LSc(2,
6)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^5 + LSc(6, 2)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale) + LSc(3, 3)*(xshiftc + x/xscale)^2*(yshiftc +
y/yscale)^2 + LSc(3, 4)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^3 + LSc(4,
3)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^2 + LSc(3, 5)*(xshiftc +
x/xscale)^2*(yshiftc + y/yscale)^4 + LSc(4, 4)*(xshiftc + x/xscale)^3*(yshiftc +
y/yscale)^3 + LSc(5, 3)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale)^2 + LSc(3,
6)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^5 + LSc(4, 5)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^4 + LSc(5, 4)*(xshiftc + x/xscale)^4*(yshiftc +
y/yscale)^3 + LSc(6, 3)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)^2 + LSc(4,
6)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^5 + LSc(5, 5)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^4 + LSc(6, 4)*(xshiftc + x/xscale)^5*(yshiftc +
y/yscale)^3 + LSc(5, 6)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale)^5 + LSc(6,
5)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)^4 + LSc(6, 6)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale)^5 + LSc(2, 2)*(xshiftc + x/xscale)*(yshiftc +
```

```
y/yscale)) + (x - 1)^4*(y - 1)^4*(84*x + 70*(x - 1)^2 + 20*(x - 1)^3 - 49)*(84*y +
70*(y - 1)^2 + 20*(y - 1)^3 - 49)*(1/xscale*LSc(2, 1) + 2/xscale*LSc(3, 1)*(xshiftc +
x/xscale) + 1/xscale*LSc(2, 2)*(yshiftc + y/yscale) + 3/xscale*LSc(4, 1)*(xshiftc +
x/xscale)^2 + 4/xscale*LSc(5, 1)*(xshiftc + x/xscale)^3 + 5/xscale*LSc(6, 1)*(xshiftc
+ x/xscale)^4 + 1/xscale*LSc(2, 3)*(yshiftc + y/yscale)^2 + 1/xscale*LSc(2,
4)*(yshiftc + y/yscale)^3 + 1/xscale*LSc(2, 5)*(yshiftc + y/yscale)^4 +
1/xscale*LSc(2, 6)*(yshiftc + y/yscale)^5 + 3/xscale*LSc(4, 3)*(xshiftc +
x/xscale)^2*(yshiftc + y/yscale)^2 + 3/xscale*LSc(4, 4)*(xshiftc +
x/xscale)^2*(yshiftc + y/yscale)^3 + 4/xscale*LSc(5, 3)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^2 + 3/xscale*LSc(4, 5)*(xshiftc +
x/xscale)^2*(yshiftc + y/yscale)^4 + 4/xscale*LSc(5, 4)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^3 + 5/xscale*LSc(6, 3)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^2 + 3/xscale*LSc(4, 6)*(xshiftc +
x/xscale)^2*(yshiftc + y/yscale)^5 + 4/xscale*LSc(5, 5)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^4 + 5/xscale*LSc(6, 4)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^3 + 4/xscale*LSc(5, 6)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^5 + 5/xscale*LSc(6, 5)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^4 + 5/xscale*LSc(6, 6)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^5 + 2/xscale*LSc(3, 2)*(xshiftc + x/xscale)*(yshiftc
+ y/yscale) + 2/xscale*LSc(3, 3)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^2 +
3/xscale*LSc(4, 2)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale) + 2/xscale*LSc(3,
4)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^3 + 4/xscale*LSc(5, 2)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale) + 2/xscale*LSc(3, 5)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)^4 + 5/xscale*LSc(6, 2)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale) +
2/xscale*LSc(3, 6)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^5) - y^4*(x - 1)^4*(140*x
+ 60*(x - 1)^2 - 56)*(20*y^3 - 70*y^2 + 84*y - 35)*(LSb(2, 1)*(xshiftb + x/xscale) +
LSb(1, 2)*(yshiftb + y/yscale) + LSb(1, 1) + LSb(3, 1)*(xshiftb + x/xscale)^2 + LSb(4,
1)*(xshiftb + x/xscale)^3 + LSb(5, 1)*(xshiftb + x/xscale)^4 + LSb(6, 1)*(xshiftb +
x/xscale)^5 + LSb(1, 3)*(yshiftb + y/yscale)^2 + LSb(1, 4)*(yshiftb + y/yscale)^3 +
LSb(1, 5)*(yshiftb + y/yscale)^4 + LSb(1, 6)*(yshiftb + y/yscale)^5 + LSb(2,
3)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^2 + LSb(3, 2)*(xshiftb +
x/xscale)^2*(yshiftb + y/yscale) + LSb(2, 4)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)^3 + LSb(4, 2)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale) + LSb(2,
5)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^4 + LSb(5, 2)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale) + LSb(2, 6)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)^5 + LSb(6, 2)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale) + LSb(3,
3)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^2 + LSb(3, 4)*(xshiftb +
x/xscale)^2*(yshiftb + y/yscale)^3 + LSb(4, 3)*(xshiftb + x/xscale)^3*(yshiftb +
y/yscale)^2 + LSb(3, 5)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^4 + LSb(4,
4)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^3 + LSb(5, 3)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^2 + LSb(3, 6)*(xshiftb + x/xscale)^2*(yshiftb +
y/yscale)^5 + LSb(4, 5)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^4 + LSb(5,
4)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale)^3 + LSb(6, 3)*(xshiftb +
x/xscale)^5*(yshiftb + y/yscale)^2 + LSb(4, 6)*(xshiftb + x/xscale)^3*(yshiftb +
y/yscale)^5 + LSb(5, 5)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale)^4 + LSb(6,
4)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)^3 + LSb(5, 6)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^5 + LSb(6, 5)*(xshiftb + x/xscale)^5*(yshiftb +
y/yscale)^4 + LSb(6, 6)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)^5 + LSb(2,
2)*(xshiftb + x/xscale)*(yshiftb + y/yscale));

dFdy=
@(x,y,xscale,yscale,xshifta,yshifta,xshiftb,yshiftb,xshiftc,yshiftc,xshiftd,yshiftd,LS
a,LSb,LSc,LSd) (x - 1)^4*(y - 1)^4*(140*y + 60*(y - 1)^2 - 56)*(84*x + 70*(x - 1)^2 +
20*(x - 1)^3 - 49)*(LSc(2, 1)*(xshiftc + x/xscale) + LSc(1, 2)*(yshiftc + y/yscale) +
LSc(1, 1) + LSc(3, 1)*(xshiftc + x/xscale)^2 + LSc(4, 1)*(xshiftc + x/xscale)^3 +
LSc(5, 1)*(xshiftc + x/xscale)^4 + LSc(6, 1)*(xshiftc + x/xscale)^5 + LSc(1,
3)*(yshiftc + y/yscale)^2 + LSc(1, 4)*(yshiftc + y/yscale)^3 + LSc(1, 5)*(yshiftc +
y/yscale)^4 + LSc(1, 6)*(yshiftc + y/yscale)^5 + LSc(2, 3)*(xshiftc +
x/xscale)*(yshiftc + y/yscale)^2 + LSc(3, 2)*(xshiftc + x/xscale)^2*(yshiftc +
y/yscale) + LSc(2, 4)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^3 + LSc(4, 2)*(xshiftc
+ x/xscale)^3*(yshiftc + y/yscale) + LSc(2, 5)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)^4 + LSc(5, 2)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale) + LSc(2,
6)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^5 + LSc(6, 2)*(xshiftc +
```

```
x/xscale)^5*(yshiftc + y/yscale) + LSc(3, 3)*(xshiftc + x/xscale)^2*(yshiftc +
y/yscale)^2 + LSc(3, 4)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^3 + LSc(4,
3)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^2 + LSc(3, 5)*(xshiftc +
x/xscale)^2*(yshiftc + y/yscale)^4 + LSc(4, 4)*(xshiftc + x/xscale)^3*(yshiftc +
y/yscale)^3 + LSc(5, 3)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale)^2 + LSc(3,
6)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^5 + LSc(4, 5)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^4 + LSc(5, 4)*(xshiftc + x/xscale)^4*(yshiftc +
y/yscale)^3 + LSc(6, 3)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)^2 + LSc(4,
6)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^5 + LSc(5, 5)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^4 + LSc(6, 4)*(xshiftc + x/xscale)^5*(yshiftc +
y/yscale)^3 + LSc(5, 6)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale)^5 + LSc(6,
5)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)^4 + LSc(6, 6)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale)^5 + LSc(2, 2)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)) + x^4*y^4*(60*y^2 - 140*y + 84)*(20*x^3 - 70*x^2 + 84*x - 35)*(LSa(2,
1)*(xshifta + x/xscale) + LSa(1, 2)*(yshifta + y/yscale) + LSa(1, 1) + LSa(3,
1)*(xshifta + x/xscale)^2 + LSa(4, 1)*(xshifta + x/xscale)^3 + LSa(5, 1)*(xshifta +
x/xscale)^4 + LSa(6, 1)*(xshifta + x/xscale)^5 + LSa(1, 3)*(yshifta + y/yscale)^2 +
LSa(1, 4)*(yshifta + y/yscale)^3 + LSa(1, 5)*(yshifta + y/yscale)^4 + LSa(1,
6)*(yshifta + y/yscale)^5 + LSa(2, 3)*(xshifta + x/xscale)*(yshifta + y/yscale)^2 +
LSa(3, 2)*(xshifta + x/xscale)^2*(yshifta + y/yscale) + LSa(2, 4)*(xshifta +
x/xscale)*(yshifta + y/yscale)^3 + LSa(4, 2)*(xshifta + x/xscale)^3*(yshifta +
y/yscale) + LSa(2, 5)*(xshifta + x/xscale)*(yshifta + y/yscale)^4 + LSa(5, 2)*(xshifta
+ x/xscale)^4*(yshifta + y/yscale) + LSa(2, 6)*(xshifta + x/xscale)*(yshifta +
y/yscale)^5 + LSa(6, 2)*(xshifta + x/xscale)^5*(yshifta + y/yscale) + LSa(3,
3)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^2 + LSa(3, 4)*(xshifta +
x/xscale)^2*(yshifta + y/yscale)^3 + LSa(4, 3)*(xshifta + x/xscale)^3*(yshifta +
y/yscale)^2 + LSa(3, 5)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^4 + LSa(4,
4)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^3 + LSa(5, 3)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^2 + LSa(3, 6)*(xshifta + x/xscale)^2*(yshifta +
y/yscale)^5 + LSa(4, 5)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^4 + LSa(5,
4)*(xshifta + x/xscale)^4*(yshifta + y/yscale)^3 + LSa(6, 3)*(xshifta +
x/xscale)^5*(yshifta + y/yscale)^2 + LSa(4, 6)*(xshifta + x/xscale)^3*(yshifta +
y/yscale)^5 + LSa(5, 5)*(xshifta + x/xscale)^4*(yshifta + y/yscale)^4 + LSa(6,
4)*(xshifta + x/xscale)^5*(yshifta + y/yscale)^3 + LSa(5, 6)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^5 + LSa(6, 5)*(xshifta + x/xscale)^5*(yshifta +
y/yscale)^4 + LSa(6, 6)*(xshifta + x/xscale)^5*(yshifta + y/yscale)^5 + LSa(2,
2)*(xshifta + x/xscale)*(yshifta + y/yscale)) - y^4*(x - 1)^4*(60*y^2 - 140*y +
84)*(84*x + 70*(x - 1)^2 + 20*(x - 1)^3 - 49)*(LSb(2, 1)*(xshiftb + x/xscale) + LSb(1,
2)*(yshiftb + y/yscale) + LSb(1, 1) + LSb(3, 1)*(xshiftb + x/xscale)^2 + LSb(4,
1)*(xshiftb + x/xscale)^3 + LSb(5, 1)*(xshiftb + x/xscale)^4 + LSb(6, 1)*(xshiftb +
x/xscale)^5 + LSb(1, 3)*(yshiftb + y/yscale)^2 + LSb(1, 4)*(yshiftb + y/yscale)^3 +
LSb(1, 5)*(yshiftb + y/yscale)^4 + LSb(1, 6)*(yshiftb + y/yscale)^5 + LSb(2,
3)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^2 + LSb(3, 2)*(xshiftb +
x/xscale)^2*(yshiftb + y/yscale) + LSb(2, 4)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)^3 + LSb(4, 2)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale) + LSb(2,
5)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^4 + LSb(5, 2)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale) + LSb(2, 6)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)^5 + LSb(6, 2)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale) + LSb(3,
3)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^2 + LSb(3, 4)*(xshiftb +
x/xscale)^2*(yshiftb + y/yscale)^3 + LSb(4, 3)*(xshiftb + x/xscale)^3*(yshiftb +
y/yscale)^2 + LSb(3, 5)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^4 + LSb(4,
4)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^3 + LSb(5, 3)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^2 + LSb(3, 6)*(xshiftb + x/xscale)^2*(yshiftb +
y/yscale)^5 + LSb(4, 5)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^4 + LSb(5,
4)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale)^3 + LSb(6, 3)*(xshiftb +
x/xscale)^5*(yshiftb + y/yscale)^2 + LSb(4, 6)*(xshiftb + x/xscale)^3*(yshiftb +
y/yscale)^5 + LSb(5, 5)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale)^4 + LSb(6,
4)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)^3 + LSb(5, 6)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^5 + LSb(6, 5)*(xshiftb + x/xscale)^5*(yshiftb +
y/yscale)^4 + LSb(6, 6)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)^5 + LSb(2,
2)*(xshiftb + x/xscale)*(yshiftb + y/yscale)) + 4*x^4*y^3*(20*x^3 - 70*x^2 + 84*x -
35)*(20*y^3 - 70*y^2 + 84*y - 35)*(LSa(2, 1)*(xshifta + x/xscale) + LSa(1, 2)*(yshifta
+ y/yscale) + LSa(1, 1) + LSa(3, 1)*(xshifta + x/xscale)^2 + LSa(4, 1)*(xshifta +
```

```
x/xscale)^3 + LSa(5, 1)*(xshifta + x/xscale)^4 + LSa(6, 1)*(xshifta + x/xscale)^5 +
LSa(1, 3)*(yshifta + y/yscale)^2 + LSa(1, 4)*(yshifta + y/yscale)^3 + LSa(1,
5)*(yshifta + y/yscale)^4 + LSa(1, 6)*(yshifta + y/yscale)^5 + LSa(2, 3)*(xshifta +
x/xscale)*(yshifta + y/yscale)^2 + LSa(3, 2)*(xshifta + x/xscale)^2*(yshifta +
y/yscale) + LSa(2, 4)*(xshifta + x/xscale)*(yshifta + y/yscale)^3 + LSa(4, 2)*(xshifta
+ x/xscale)^3*(yshifta + y/yscale) + LSa(2, 5)*(xshifta + x/xscale)*(yshifta +
y/yscale)^4 + LSa(5, 2)*(xshifta + x/xscale)^4*(yshifta + y/yscale) + LSa(2,
6)*(xshifta + x/xscale)*(yshifta + y/yscale)^5 + LSa(6, 2)*(xshifta +
x/xscale)^5*(yshifta + y/yscale) + LSa(3, 3)*(xshifta + x/xscale)^2*(yshifta +
y/yscale)^2 + LSa(3, 4)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^3 + LSa(4,
3)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^2 + LSa(3, 5)*(xshifta +
x/xscale)^2*(yshifta + y/yscale)^4 + LSa(4, 4)*(xshifta + x/xscale)^3*(yshifta +
y/yscale)^3 + LSa(5, 3)*(xshifta + x/xscale)^4*(yshifta + y/yscale)^2 + LSa(3,
6)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^5 + LSa(4, 5)*(xshifta +
x/xscale)^3*(yshifta + y/yscale)^4 + LSa(5, 4)*(xshifta + x/xscale)^4*(yshifta +
y/yscale)^3 + LSa(6, 3)*(xshifta + x/xscale)^5*(yshifta + y/yscale)^2 + LSa(4,
6)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^5 + LSa(5, 5)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^4 + LSa(6, 4)*(xshifta + x/xscale)^5*(yshifta +
y/yscale)^3 + LSa(5, 6)*(xshifta + x/xscale)^4*(yshifta + y/yscale)^5 + LSa(6,
5)*(xshifta + x/xscale)^5*(yshifta + y/yscale)^4 + LSa(6, 6)*(xshifta +
x/xscale)^5*(yshifta + y/yscale)^5 + LSa(2, 2)*(xshifta + x/xscale)*(yshifta +
y/yscale)) - 4*y^3*(x - 1)^4*(84*x + 70*(x - 1)^2 + 20*(x - 1)^3 - 49)*(20*y^3 -
70*y^2 + 84*y - 35)*(LSb(2, 1)*(xshiftb + x/xscale) + LSb(1, 2)*(yshiftb + y/yscale) +
LSb(1, 1) + LSb(3, 1)*(xshiftb + x/xscale)^2 + LSb(4, 1)*(xshiftb + x/xscale)^3 +
LSb(5, 1)*(xshiftb + x/xscale)^4 + LSb(6, 1)*(xshiftb + x/xscale)^5 + LSb(1,
3)*(yshiftb + y/yscale)^2 + LSb(1, 4)*(yshiftb + y/yscale)^3 + LSb(1, 5)*(yshiftb +
y/yscale)^4 + LSb(1, 6)*(yshiftb + y/yscale)^5 + LSb(2, 3)*(xshiftb +
x/xscale)*(yshiftb + y/yscale)^2 + LSb(3, 2)*(xshiftb + x/xscale)^2*(yshiftb +
y/yscale) + LSb(2, 4)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^3 + LSb(4, 2)*(xshiftb
+ x/xscale)^3*(yshiftb + y/yscale) + LSb(2, 5)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)^4 + LSb(5, 2)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale) + LSb(2,
6)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^5 + LSb(6, 2)*(xshiftb +
x/xscale)^5*(yshiftb + y/yscale) + LSb(3, 3)*(xshiftb + x/xscale)^2*(yshiftb +
y/yscale)^2 + LSb(3, 4)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^3 + LSb(4,
3)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^2 + LSb(3, 5)*(xshiftb +
x/xscale)^2*(yshiftb + y/yscale)^4 + LSb(4, 4)*(xshiftb + x/xscale)^3*(yshiftb +
y/yscale)^3 + LSb(5, 3)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale)^2 + LSb(3,
6)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^5 + LSb(4, 5)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale)^4 + LSb(5, 4)*(xshiftb + x/xscale)^4*(yshiftb +
y/yscale)^3 + LSb(6, 3)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)^2 + LSb(4,
6)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^5 + LSb(5, 5)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^4 + LSb(6, 4)*(xshiftb + x/xscale)^5*(yshiftb +
y/yscale)^3 + LSb(5, 6)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale)^5 + LSb(6,
5)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)^4 + LSb(6, 6)*(xshiftb +
x/xscale)^5*(yshiftb + y/yscale)^5 + LSb(2, 2)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)) - 4*x^4*(y - 1)^3*(84*y + 70*(y - 1)^2 + 20*(y - 1)^3 - 49)*(20*x^3 -
70*x^2 + 84*x - 35)*(LSd(2, 1)*(xshiftd + x/xscale) + LSd(1, 2)*(yshiftd + y/yscale) +
LSd(1, 1) + LSd(3, 1)*(xshiftd + x/xscale)^2 + LSd(4, 1)*(xshiftd + x/xscale)^3 +
LSd(5, 1)*(xshiftd + x/xscale)^4 + LSd(6, 1)*(xshiftd + x/xscale)^5 + LSd(1,
3)*(yshiftd + y/yscale)^2 + LSd(1, 4)*(yshiftd + y/yscale)^3 + LSd(1, 5)*(yshiftd +
y/yscale)^4 + LSd(1, 6)*(yshiftd + y/yscale)^5 + LSd(2, 3)*(xshiftd +
x/xscale)*(yshiftd + y/yscale)^2 + LSd(3, 2)*(xshiftd + x/xscale)^2*(yshiftd +
y/yscale) + LSd(2, 4)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^3 + LSd(4, 2)*(xshiftd
+ x/xscale)^3*(yshiftd + y/yscale) + LSd(2, 5)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^4 + LSd(5, 2)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale) + LSd(2,
6)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^5 + LSd(6, 2)*(xshiftd +
x/xscale)^5*(yshiftd + y/yscale) + LSd(3, 3)*(xshiftd + x/xscale)^2*(yshiftd +
y/yscale)^2 + LSd(3, 4)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^3 + LSd(4,
3)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale)^2 + LSd(3, 5)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale)^4 + LSd(4, 4)*(xshiftd + x/xscale)^3*(yshiftd +
y/yscale)^3 + LSd(5, 3)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale)^2 + LSd(3,
6)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^5 + LSd(4, 5)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale)^4 + LSd(5, 4)*(xshiftd + x/xscale)^4*(yshiftd +
```

```
y/yscale)^3 + LSd(6, 3)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)^2 + LSd(4,
6)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale)^5 + LSd(5, 5)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^4 + LSd(6, 4)*(xshiftd + x/xscale)^5*(yshiftd +
y/yscale)^3 + LSd(5, 6)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale)^5 + LSd(6,
5)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)^4 + LSd(6, 6)*(xshiftd +
x/xscale)^5*(yshiftd + y/yscale)^5 + LSd(2, 2)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)) + x^4*y^4*(20*x^3 - 70*x^2 + 84*x - 35)*(20*y^3 - 70*y^2 + 84*y -
35)*(1/yscale*LSa(1, 2) + 1/yscale*LSa(2, 2)*(xshifta + x/xscale) + 2/yscale*LSa(1,
3)*(yshifta + y/yscale) + 1/yscale*LSa(3, 2)*(xshifta + x/xscale)^2 + 1/yscale*LSa(4,
2)*(xshifta + x/xscale)^3 + 1/yscale*LSa(5, 2)*(xshifta + x/xscale)^4 +
1/yscale*LSa(6, 2)*(xshifta + x/xscale)^5 + 3/yscale*LSa(1, 4)*(yshifta + y/yscale)^2
+ 4/yscale*LSa(1, 5)*(yshifta + y/yscale)^3 + 5/yscale*LSa(1, 6)*(yshifta +
y/yscale)^4 + 3/yscale*LSa(3, 4)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^2 +
4/yscale*LSa(3, 5)*(xshifta + x/xscale)^2*(yshifta + y/yscale)^3 + 3/yscale*LSa(4,
4)*(xshifta + x/xscale)^3*(yshifta + y/yscale)^2 + 5/yscale*LSa(3, 6)*(xshifta +
x/xscale)^2*(yshifta + y/yscale)^4 + 4/yscale*LSa(4, 5)*(xshifta +
x/xscale)^3*(yshifta + y/yscale)^3 + 3/yscale*LSa(5, 4)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^2 + 5/yscale*LSa(4, 6)*(xshifta +
x/xscale)^3*(yshifta + y/yscale)^4 + 4/yscale*LSa(5, 5)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^3 + 3/yscale*LSa(6, 4)*(xshifta +
x/xscale)^5*(yshifta + y/yscale)^2 + 5/yscale*LSa(5, 6)*(xshifta +
x/xscale)^4*(yshifta + y/yscale)^4 + 4/yscale*LSa(6, 5)*(xshifta +
x/xscale)^5*(yshifta + y/yscale)^3 + 5/yscale*LSa(6, 6)*(xshifta +
x/xscale)^5*(yshifta + y/yscale)^4 + 2/yscale*LSa(2, 3)*(xshifta + x/xscale)*(yshifta
+ y/yscale) + 3/yscale*LSa(2, 4)*(xshifta + x/xscale)*(yshifta + y/yscale)^2 +
2/yscale*LSa(3, 3)*(xshifta + x/xscale)^2*(yshifta + y/yscale) + 4/yscale*LSa(2,
5)*(xshifta + x/xscale)*(yshifta + y/yscale)^3 + 2/yscale*LSa(4, 3)*(xshifta +
x/xscale)^3*(yshifta + y/yscale) + 5/yscale*LSa(2, 6)*(xshifta + x/xscale)*(yshifta +
y/yscale)^4 + 2/yscale*LSa(5, 3)*(xshifta + x/xscale)^4*(yshifta + y/yscale) +
2/yscale*LSa(6, 3)*(xshifta + x/xscale)^5*(yshifta + y/yscale)) + 4*(x - 1)^4*(y -
1)^3*(84*x + 70*(x - 1)^2 + 20*(x - 1)^3 - 49)*(84*y + 70*(y - 1)^2 + 20*(y - 1)^3 -
49)*(LSc(2, 1)*(xshiftc + x/xscale) + LSc(1, 2)*(yshiftc + y/yscale) + LSc(1, 1) +
LSc(3, 1)*(xshiftc + x/xscale)^2 + LSc(4, 1)*(xshiftc + x/xscale)^3 + LSc(5,
1)*(xshiftc + x/xscale)^4 + LSc(6, 1)*(xshiftc + x/xscale)^5 + LSc(1, 3)*(yshiftc +
y/yscale)^2 + LSc(1, 4)*(yshiftc + y/yscale)^3 + LSc(1, 5)*(yshiftc + y/yscale)^4 +
LSc(1, 6)*(yshiftc + y/yscale)^5 + LSc(2, 3)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)^2 + LSc(3, 2)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale) + LSc(2,
4)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^3 + LSc(4, 2)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale) + LSc(2, 5)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)^4 + LSc(5, 2)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale) + LSc(2,
6)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^5 + LSc(6, 2)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale) + LSc(3, 3)*(xshiftc + x/xscale)^2*(yshiftc +
y/yscale)^2 + LSc(3, 4)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^3 + LSc(4,
3)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^2 + LSc(3, 5)*(xshiftc +
x/xscale)^2*(yshiftc + y/yscale)^4 + LSc(4, 4)*(xshiftc + x/xscale)^3*(yshiftc +
y/yscale)^3 + LSc(5, 3)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale)^2 + LSc(3,
6)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^5 + LSc(4, 5)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^4 + LSc(5, 4)*(xshiftc + x/xscale)^4*(yshiftc +
y/yscale)^3 + LSc(6, 3)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)^2 + LSc(4,
6)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^5 + LSc(5, 5)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^4 + LSc(6, 4)*(xshiftc + x/xscale)^5*(yshiftc +
y/yscale)^3 + LSc(5, 6)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale)^5 + LSc(6,
5)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)^4 + LSc(6, 6)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale)^5 + LSc(2, 2)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)) - y^4*(x - 1)^4*(84*x + 70*(x - 1)^2 + 20*(x - 1)^3 - 49)*(20*y^3 - 70*y^2
+ 84*y - 35)*(1/yscale*LSb(1, 2) + 1/yscale*LSb(2, 2)*(xshiftb + x/xscale) +
2/yscale*LSb(1, 3)*(yshiftb + y/yscale) + 1/yscale*LSb(3, 2)*(xshiftb + x/xscale)^2 +
1/yscale*LSb(4, 2)*(xshiftb + x/xscale)^3 + 1/yscale*LSb(5, 2)*(xshiftb + x/xscale)^4
+ 1/yscale*LSb(6, 2)*(xshiftb + x/xscale)^5 + 3/yscale*LSb(1, 4)*(yshiftb +
y/yscale)^2 + 4/yscale*LSb(1, 5)*(yshiftb + y/yscale)^3 + 5/yscale*LSb(1, 6)*(yshiftb
+ y/yscale)^4 + 3/yscale*LSb(3, 4)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^2 +
4/yscale*LSb(3, 5)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale)^3 + 3/yscale*LSb(4,
4)*(xshiftb + x/xscale)^3*(yshiftb + y/yscale)^2 + 5/yscale*LSb(3, 6)*(xshiftb +
```

```
x/xscale)^2*(yshiftb + y/yscale)^4 + 4/yscale*LSb(4, 5)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale)^3 + 3/yscale*LSb(5, 4)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^2 + 5/yscale*LSb(4, 6)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale)^4 + 4/yscale*LSb(5, 5)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^3 + 3/yscale*LSb(6, 4)*(xshiftb +
x/xscale)^5*(yshiftb + y/yscale)^2 + 5/yscale*LSb(5, 6)*(xshiftb +
x/xscale)^4*(yshiftb + y/yscale)^4 + 4/yscale*LSb(6, 5)*(xshiftb +
x/xscale)^5*(yshiftb + y/yscale)^3 + 5/yscale*LSb(6, 6)*(xshiftb +
x/xscale)^5*(yshiftb + y/yscale)^4 + 2/yscale*LSb(2, 3)*(xshiftb + x/xscale)*(yshiftb
+ y/yscale) + 3/yscale*LSb(2, 4)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^2 +
2/yscale*LSb(3, 3)*(xshiftb + x/xscale)^2*(yshiftb + y/yscale) + 4/yscale*LSb(2,
5)*(xshiftb + x/xscale)*(yshiftb + y/yscale)^3 + 2/yscale*LSb(4, 3)*(xshiftb +
x/xscale)^3*(yshiftb + y/yscale) + 5/yscale*LSb(2, 6)*(xshiftb + x/xscale)*(yshiftb +
y/yscale)^4 + 2/yscale*LSb(5, 3)*(xshiftb + x/xscale)^4*(yshiftb + y/yscale) +
2/yscale*LSb(6, 3)*(xshiftb + x/xscale)^5*(yshiftb + y/yscale)) - x^4*(y - 1)^4*(84*y
+ 70*(y - 1)^2 + 20*(y - 1)^3 - 49)*(20*x^3 - 70*x^2 + 84*x - 35)*(1/yscale*LSd(1, 2)
+ 1/yscale*LSd(2, 2)*(xshiftd + x/xscale) + 2/yscale*LSd(1, 3)*(yshiftd + y/yscale) +
1/yscale*LSd(3, 2)*(xshiftd + x/xscale)^2 + 1/yscale*LSd(4, 2)*(xshiftd + x/xscale)^3
+ 1/yscale*LSd(5, 2)*(xshiftd + x/xscale)^4 + 1/yscale*LSd(6, 2)*(xshiftd +
x/xscale)^5 + 3/yscale*LSd(1, 4)*(yshiftd + y/yscale)^2 + 4/yscale*LSd(1, 5)*(yshiftd
+ y/yscale)^3 + 5/yscale*LSd(1, 6)*(yshiftd + y/yscale)^4 + 3/yscale*LSd(3,
4)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^2 + 4/yscale*LSd(3, 5)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale)^3 + 3/yscale*LSd(4, 4)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale)^2 + 5/yscale*LSd(3, 6)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale)^4 + 4/yscale*LSd(4, 5)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale)^3 + 3/yscale*LSd(5, 4)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^2 + 5/yscale*LSd(4, 6)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale)^4 + 4/yscale*LSd(5, 5)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^3 + 3/yscale*LSd(6, 4)*(xshiftd +
x/xscale)^5*(yshiftd + y/yscale)^2 + 5/yscale*LSd(5, 6)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^4 + 4/yscale*LSd(6, 5)*(xshiftd +
x/xscale)^5*(yshiftd + y/yscale)^3 + 5/yscale*LSd(6, 6)*(xshiftd +
x/xscale)^5*(yshiftd + y/yscale)^4 + 2/yscale*LSd(2, 3)*(xshiftd + x/xscale)*(yshiftd
+ y/yscale) + 3/yscale*LSd(2, 4)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^2 +
2/yscale*LSd(3, 3)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale) + 4/yscale*LSd(2,
5)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^3 + 2/yscale*LSd(4, 3)*(xshiftd +
x/xscale)^3*(yshiftd + y/yscale) + 5/yscale*LSd(2, 6)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^4 + 2/yscale*LSd(5, 3)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale) +
2/yscale*LSd(6, 3)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)) + (x - 1)^4*(y -
1)^4*(84*x + 70*(x - 1)^2 + 20*(x - 1)^3 - 49)*(84*y + 70*(y - 1)^2 + 20*(y - 1)^3 -
49)*(1/yscale*LSc(1, 2) + 1/yscale*LSc(2, 2)*(xshiftc + x/xscale) + 2/yscale*LSc(1,
3)*(yshiftc + y/yscale) + 1/yscale*LSc(3, 2)*(xshiftc + x/xscale)^2 + 1/yscale*LSc(4,
2)*(xshiftc + x/xscale)^3 + 1/yscale*LSc(5, 2)*(xshiftc + x/xscale)^4 +
1/yscale*LSc(6, 2)*(xshiftc + x/xscale)^5 + 3/yscale*LSc(1, 4)*(yshiftc + y/yscale)^2
+ 4/yscale*LSc(1, 5)*(yshiftc + y/yscale)^3 + 5/yscale*LSc(1, 6)*(yshiftc +
y/yscale)^4 + 3/yscale*LSc(3, 4)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^2 +
4/yscale*LSc(3, 5)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale)^3 + 3/yscale*LSc(4,
4)*(xshiftc + x/xscale)^3*(yshiftc + y/yscale)^2 + 5/yscale*LSc(3, 6)*(xshiftc +
x/xscale)^2*(yshiftc + y/yscale)^4 + 4/yscale*LSc(4, 5)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^3 + 3/yscale*LSc(5, 4)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^2 + 5/yscale*LSc(4, 6)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale)^4 + 4/yscale*LSc(5, 5)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^3 + 3/yscale*LSc(6, 4)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale)^2 + 5/yscale*LSc(5, 6)*(xshiftc +
x/xscale)^4*(yshiftc + y/yscale)^4 + 4/yscale*LSc(6, 5)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale)^3 + 5/yscale*LSc(6, 6)*(xshiftc +
x/xscale)^5*(yshiftc + y/yscale)^4 + 2/yscale*LSc(2, 3)*(xshiftc + x/xscale)*(yshiftc
+ y/yscale) + 3/yscale*LSc(2, 4)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^2 +
2/yscale*LSc(3, 3)*(xshiftc + x/xscale)^2*(yshiftc + y/yscale) + 4/yscale*LSc(2,
5)*(xshiftc + x/xscale)*(yshiftc + y/yscale)^3 + 2/yscale*LSc(4, 3)*(xshiftc +
x/xscale)^3*(yshiftc + y/yscale) + 5/yscale*LSc(2, 6)*(xshiftc + x/xscale)*(yshiftc +
y/yscale)^4 + 2/yscale*LSc(5, 3)*(xshiftc + x/xscale)^4*(yshiftc + y/yscale) +
2/yscale*LSc(6, 3)*(xshiftc + x/xscale)^5*(yshiftc + y/yscale)) - x^4*(y - 1)^4*(140*y
```

```
+ 60*(y - 1)^2 - 56)*(20*x^3 - 70*x^2 + 84*x - 35)*(LSd(2, 1)*(xshiftd + x/xscale) +
LSd(1, 2)*(yshiftd + y/yscale) + LSd(1, 1) + LSd(3, 1)*(xshiftd + x/xscale)^2 + LSd(4,
1)*(xshiftd + x/xscale)^3 + LSd(5, 1)*(xshiftd + x/xscale)^4 + LSd(6, 1)*(xshiftd +
x/xscale)^5 + LSd(1, 3)*(yshiftd + y/yscale)^2 + LSd(1, 4)*(yshiftd + y/yscale)^3 +
LSd(1, 5)*(yshiftd + y/yscale)^4 + LSd(1, 6)*(yshiftd + y/yscale)^5 + LSd(2,
3)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^2 + LSd(3, 2)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale) + LSd(2, 4)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^3 + LSd(4, 2)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale) + LSd(2,
5)*(xshiftd + x/xscale)*(yshiftd + y/yscale)^4 + LSd(5, 2)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale) + LSd(2, 6)*(xshiftd + x/xscale)*(yshiftd +
y/yscale)^5 + LSd(6, 2)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale) + LSd(3,
3)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^2 + LSd(3, 4)*(xshiftd +
x/xscale)^2*(yshiftd + y/yscale)^3 + LSd(4, 3)*(xshiftd + x/xscale)^3*(yshiftd +
y/yscale)^2 + LSd(3, 5)*(xshiftd + x/xscale)^2*(yshiftd + y/yscale)^4 + LSd(4,
4)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale)^3 + LSd(5, 3)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^2 + LSd(3, 6)*(xshiftd + x/xscale)^2*(yshiftd +
y/yscale)^5 + LSd(4, 5)*(xshiftd + x/xscale)^3*(yshiftd + y/yscale)^4 + LSd(5,
4)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale)^3 + LSd(6, 3)*(xshiftd +
x/xscale)^5*(yshiftd + y/yscale)^2 + LSd(4, 6)*(xshiftd + x/xscale)^3*(yshiftd +
y/yscale)^5 + LSd(5, 5)*(xshiftd + x/xscale)^4*(yshiftd + y/yscale)^4 + LSd(6,
4)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)^3 + LSd(5, 6)*(xshiftd +
x/xscale)^4*(yshiftd + y/yscale)^5 + LSd(6, 5)*(xshiftd + x/xscale)^5*(yshiftd +
y/yscale)^4 + LSd(6, 6)*(xshiftd + x/xscale)^5*(yshiftd + y/yscale)^5 + LSd(2,
2)*(xshiftd + x/xscale)*(yshiftd + y/yscale));
```

EVALUATIONS

*note that scale factor has been included (possible re-write needed?)

```
% returns an array of the function evaluations

F_eval=      evalfunc(F
,numzonesx,numzonesy,ptsperzonex,ptsperzoney,dxf,dyf,dxw,dyw,xfshift,yfshift,LScoeffs)
;

dFdx_eval=    (dxw/dxf) * evalfunc(dFdx
,numzonesx,numzonesy,ptsperzonex,ptsperzoney,dxf,dyf,dxw,dyw,xfshift,yfshift,LScoeffs)
;

dFdy_eval=    (dyw/dyf) * evalfunc(dFdy
,numzonesx,numzonesy,ptsperzonex,ptsperzoney,dxf,dyf,dxw,dyw,xfshift,yfshift,LScoeffs)
;

F_eval_xsibnd = evalfuncxsibnd(F
,numzonesx,numzonesy,ptsperzonex,ptsperzoney,dxf,dyf,dxw,dyw,xfshift,yfshift,LScoeffs)
;

F_eval_etabnd = evalfuncetabnd(F
,numzonesx,numzonesy,ptsperzonex,ptsperzoney,dxf,dyf,dxw,dyw,xfshift,yfshift,LScoeffs)
;
end
```

SUBFUNCTIONS

```
function [eval] =
evalfunc(func,numzonesx,numzonesy,ptsperzonex,ptsperzoney,dxf,dyf,dxw,dyw,xfshift,yfsh
ift,LScoeffs)

    % interior/right wall/right-upper corner/upper wall
    for i=2:numzonesx
        for j=2:numzonesy
            for m=1:ptsperzonex-1
```

```
                for n=1:ptsperzoney-1
                    eval((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)= func( (m-
1)*dxw+1/2*dxw , (n-1)*dyw+1/2*dyw , dxw/dxf , dyw/dyf , 0,0 , xfshift,0 ,
xfshift,yfshift , 0,yfshift ,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i
,j+1,:,:)),squeeze(LScoeffs(i  ,j  ,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

    % upper-left corner/left wall
    for i=1
        for j=2:numzonesy
            for m=1:ptsperzonex-1
                for n=1:ptsperzoney-1
                    eval((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)= func( (m-
1)*dxw+1/2*dxw , (n-1)*dyw+1/2*dyw , dxw/dxf , dyw/dyf , 0,0 , 0,0 , 0,yfshift ,
0,yfshift ,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i
,j+1,:,:)),squeeze(LScoeffs(i  ,j  ,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

    % left-lower corner
    for i=1
        for j=1
            for m=1:ptsperzonex-1
                for n=1:ptsperzoney-1
                    eval((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)= func( (m-
1)*dxw+1/2*dxw , (n-1)*dyw+1/2*dyw , dxw/dxf , dyw/dyf , 0,0 , 0,0 , 0,0 , 0,0
,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i  ,j+1,:,:)),squeeze(LScoeffs(i  ,j
,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

    % lower wall/lower-right corner
    for i=2:numzonesx
        for j=1
            for m=1:ptsperzonex-1
                for n=1:ptsperzoney-1
                    eval((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)= func( (m-
1)*dxw+1/2*dxw , (n-1)*dyw+1/2*dyw , dxw/dxf , dyw/dyf , 0,0 , xfshift,0 , xfshift,0 ,
0,0 ,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i  ,j+1,:,:)),squeeze(LScoeffs(i
,j  ,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

end
function [eval_xsibnd] =
evalfuncxsibnd(func,numzonesx,numzonesy,ptsperzonex,ptsperzoney,dxf,dyf,dxw,dyw,xfshif
t,yfshift,LScoeffs)

    % interior/right wall/right-upper corner/upper wall
    for i=2:numzonesx
        for j=2:numzonesy
            for m=1:ptsperzonex
                for n=1:ptsperzoney-1
```

```
                    eval_xsibnd((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)=
func( (m-1)*dxw , (n-1)*dyw+1/2*dyw , dxw/dxf , dyw/dyf , 0,0 , xfshift,0 ,
xfshift,yfshift , 0,yfshift ,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i
,j+1,:,:)),squeeze(LScoeffs(i  ,j  ,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

    % upper-left corner/left wall
    for i=1
        for j=2:numzonesy
            for m=1:ptsperzonex
                for n=1:ptsperzoney-1
                    eval_xsibnd((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)=
func( (m-1)*dxw , (n-1)*dyw+1/2*dyw , dxw/dxf , dyw/dyf , 0,0 , 0,0 , 0,yfshift ,
0,yfshift ,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i
,j+1,:,:)),squeeze(LScoeffs(i  ,j  ,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

    % left-lower corner
    for i=1
        for j=1
            for m=1:ptsperzonex
                for n=1:ptsperzoney-1
                    eval_xsibnd((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)=
func( (m-1)*dxw , (n-1)*dyw+1/2*dyw , dxw/dxf , dyw/dyf , 0,0 , 0,0 , 0,0 , 0,0
,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i  ,j+1,:,:)),squeeze(LScoeffs(i  ,j
,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

    % lower wall/lower-right corner
    for i=2:numzonesx
        for j=1
            for m=1:ptsperzonex
                for n=1:ptsperzoney-1
                    eval_xsibnd((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)=
func( (m-1)*dxw , (n-1)*dyw+1/2*dyw , dxw/dxf , dyw/dyf , 0,0 , xfshift,0 , xfshift,0
, 0,0 ,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i
,j+1,:,:)),squeeze(LScoeffs(i  ,j  ,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

end
function [eval_etabnd] =
evalfuncetabnd(func,numzonesx,numzonesy,ptsperzonex,ptsperzoney,dxf,dyf,dxw,dyw,xfshif
t,yfshift,LScoeffs)

    % interior/right wall/right-upper corner/upper wall
    for i=2:numzonesx
        for j=2:numzonesy
            for m=1:ptsperzonex-1
                for n=1:ptsperzoney
                    eval_etabnd((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)=
func( (m-1)*dxw+1/2*dxw , (n-1)*dyw , dxw/dxf , dyw/dyf , 0,0 , xfshift,0 ,
```

```
xfshift,yfshift , 0,yfshift ,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i
,j+1,:,:)),squeeze(LScoeffs(i  ,j  ,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

    % upper-left corner/left wall
    for i=1
        for j=2:numzonesy
            for m=1:ptsperzonex-1
                for n=1:ptsperzoney
                    eval_etabnd((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)=
func( (m-1)*dxw+1/2*dxw , (n-1)*dyw , dxw/dxf , dyw/dyf , 0,0 , 0,0 , 0,yfshift ,
0,yfshift ,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i
,j+1,:,:)),squeeze(LScoeffs(i  ,j  ,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

    % left-lower corner
    for i=1
        for j=1
            for m=1:ptsperzonex-1
                for n=1:ptsperzoney
                    eval_etabnd((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)=
func( (m-1)*dxw+1/2*dxw , (n-1)*dyw , dxw/dxf , dyw/dyf , 0,0 , 0,0 , 0,0 , 0,0
,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i  ,j+1,:,:)),squeeze(LScoeffs(i  ,j
,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

    % lower wall/lower-right corner
    for i=2:numzonesx
        for j=1
            for m=1:ptsperzonex-1
                for n=1:ptsperzoney
                    eval_etabnd((i-1)*(ptsperzonex-1)+m,(j-1)*(ptsperzoney-1)+n)=
func( (m-1)*dxw+1/2*dxw , (n-1)*dyw , dxw/dxf , dyw/dyf , 0,0 , xfshift,0 , xfshift,0
, 0,0 ,squeeze(LScoeffs(i+1,j+1,:,:)),squeeze(LScoeffs(i
,j+1,:,:)),squeeze(LScoeffs(i  ,j  ,:,:)),squeeze(LScoeffs(i+1,j  ,:,:)));
                end
            end
        end
    end

end
```

Appendix H: Nearby Problem DE

Nearby Problem DE: This MATLAB script calculates the nearby problem discretization error and appends it to a TECPLOT data file.

```
% to be run after nearby problem is run and data file is in directory
% workspace must still include variables from masterscriptEuler.m file

% numerical solution to nearby problem

%npfilename='R2.5.129.MNP.solution.cntr.noheader';
%npfilename='R2.5.MMS.129.MNP.solution.cntr.noheader';
%npfilename='cart.129.MNP.solution.cntr.noheader';
%npfilename='curv2d.129.MNP.solution.cntr.noheader';
npfilename='airfoil.129x257.MNP.solution.cntr.noheader';
npdata=dlmread(strcat(filepath,npfilename,'.dat'),'');

rho_np=reshape(npdata(:,3),numptsx-1,numptsy-1); %3
u_np=reshape(npdata(:,4),numptsx-1,numptsy-1);
v_np=reshape(npdata(:,5),numptsx-1,numptsy-1);
p_np=reshape(npdata(:,6),numptsx-1,numptsy-1);


% nearby problem discretization error
DE_NP_rho=rho_np-rho;
DE_NP_u=u_np-u;
DE_NP_v=v_np-v;
DE_NP_p=p_np-p;

% append nearby problem discretization error to ALLDE tecplot file
% note that header data will be correct after appending
dlmwrite(strcat(filepath,fileid,'_TECPLOTccALLDE.dat'),...
    [reshape(DE_NP_rho ,(numptsx-1)*(numptsy-1),1) ; reshape(DE_NP_u  ,(numptsx-
1)*(numptsy-1),1) ;...
     reshape(DE_NP_v   ,(numptsx-1)*(numptsy-1),1) ; reshape(DE_NP_p  ,(numptsx-
1)*(numptsy-1),1) ],...
    'delimiter', ' ', 'precision', '%22.15e', '-append');
```