Spring 2016

Modeling Shock Waves Using Exponential Interpolation Functions with the Least-Squares Finite Element Method

Bradford Scott Smith Jr.
Old Dominion University

MODELING SHOCK WAVES USING EXPONENTIAL INTERPOLATION FUNCTIONS WITH THE LEAST-SQUARES FINITE ELEMENT METHOD

by

Bradford Scott Smith Jr.
B.S. May 2007, Virginia Polytechnic Institute and State University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

AEROSPACE ENGINEERING

OLD DOMINION UNIVERSITY
May 2016

Approved by:

Gene J. Hou (Director)

Miltiadis D. Kotinis (Member)

Duc T. Nguyen (Member)

ABSTRACT

MODELING SHOCK WAVES USING EXPONENTIAL INTERPOLATION FUNCTIONS WITH THE LEAST-SQUARES FINITE ELEMENT METHOD

Bradford Scott Smith Jr.
Old Dominion University, 2016
Director: Dr. Gene J. Hou

The hypothesis of this research is that exponential interpolation functions will approximate fluid properties at shock waves with less error than polynomial interpolation functions. Exponential interpolation functions are derived for the purpose of modeling sharp gradients. General equations for conservation of mass, momentum, and energy for an inviscid flow of a perfect gas are converted to finite element equations using the least-squares method. Boundary conditions and a mesh adaptation scheme are also presented. An oblique shock reflection problem is used as a benchmark to determine whether or not exponential interpolation provides any advantages over Lagrange polynomial interpolation. Using exponential interpolation in elements downstream of a shock and having edges coincident with the shock showed a slight reduction in the solution error. However there was very little qualitative difference between solutions using polynomial and exponential interpolation. Regardless of the type of interpolation used, the shocks were smeared and oscillations were present both upstream and downstream of the shock waves. When a mesh adaptation scheme was implemented, exponential elements adjacent to the shock waves became much smaller and the numerical solution diverged. Changing the exponential elements to polynomial elements yielded a convergent solution. There appears to be no significant advantage to using exponential interpolation in comparison to Lagrange polynomial interpolation.

NOMENCLATURE

$\bar{\mathcal{H}}$	Component of the Hessian reconstructed with positive eigenvalues
$\bar{h}$	Specific enthalpy
β	Acute angle between a shock wave and the upstream velocity vector
Δt	Time step
Δx	Step in the x-direction
Δy	Step in the y-direction
ℓ	Lagrange polynomial
η	Local element coordinate in one dimension
η_k	Location of node k in one-dimensional local coordinates
γ	Ratio of specific heat capacities
Γ_{wall}	Solid wall boundary
λ	Eigenvalue
$[J]$	Jacobian
$[K]$	Finite element coefficient matrix
$[Q]$	Finite element coefficient matrix for a solid wall boundary
$\{\kappa\}$	Eigenvector
$\{\mathscr{R}\}$	Vector of nonlinear residuals
$\{f\}$	Finite element residual vector
$\mathcal{H}$	Component of the Hessian
$\mathcal{P}$	Spring potential
$\mathcal{R}$	Gas constant
$\mathscr{R}$	Nonlinear residual
Ω	Domain of the finite element problem
ω	Relaxation parameter
Ω_e	Domain of an element
ϕ	Potential field
ψ	Two-dimensional interpolation function in global coordinates
ρ	Density
ρ_∞	Free stream density
σ	Dummy variable that can represent density, velocity, or pressure
θ	Angle of flow deflection downstream of an oblique shock
ε	Exponential interpolation function in one dimension
ϑ	Penalty weight
$\vec{n}$	Outward pointing unit normal vector of an element

$\vec{v}$	Velocity vector
$\widehat{\phi}$	Two-dimensional shape function
$\widehat{\psi}$	Two-dimensional interpolation function in the local coordinate system
ξ, η	Local coordinate directions
$\{U\}$	Vector of dependent variables
a	Speed of sound
C_v	Specific heat capacity at constant volume
E	Error
e	Specific internal energy
f_x	Body force in the x-direction
f_y	Body force in the y-direction
f_z	Body force in the z-direction
g	Interpolation coefficients
h	Exponential parameter
I	Functional of residuals
i, j, k, s	Index variables
I_0	Integral calculated using the `quadgk` function in MATLAB
I_e	Functional of residuals on an element
I_{GL}	Integral calculated using Gauss-Legendre quadrature
k	Spring stiffness
L	Length
M	Mach number
M_n	Mach number of the velocity component normal to a shock wave
N_{GL}	Number of Gauss-Legendre quadrature points
P	Pressure
q	Heat transfer
R	Linear residual
T	Temperature
t	Time
u	Dummy dependent variable
V	Magnitude of velocity
v_∞	Free stream velocity magnitude
v_x	Component of velocity in the x-direction
v_y	Component of velocity in the y-direction
v_z	Component of velocity in the z-direction
x, y, z	Global coordinate directions

H Heaviside function

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Fluid properties in flows containing shock waves experience a sudden jump at the shock waves, similar to a Heaviside step function. Typical finite element models use polynomials as smooth approximations for the fluid properties which tend to either have a smearing effect on shock waves or create oscillations in the region of shock waves. Improvements to the finite element method for modeling shock waves fall into one or a mix of three general categories; mesh manipulation, discontinuous methods, and interpolation function refinement.

Mesh manipulation typically involves subdividing elements into smaller elements, increasing the degrees of freedom available to approximate large gradients but also requiring more computational resources to generate the coefficient matrix. Another mesh manipulation approach is to move nodes closer to large gradients while keeping the total global degrees of freedom constant. Taghaddosi et al.[1] developed a mesh adaptation method that also aligns the edges of elements with shock waves. The method estimates the error in the approximation of the dependent variables and moves nodes to distribute the error evenly over the mesh.

Subdividing one element into smaller elements usually also requires the neighboring elements to be subdivided to maintain continuity. Discontinuous finite element methods circumvent that disadvantage by breaking the continuity between adjacent elements and imposing a constraint on the inter-element fluxes. Since inter-element continuity is not required, discontinuous Galerkin methods allow easy implementation of mesh refinement schemes.[2] The degree of interpolation polynomials can also be increased or decreased without affecting neighboring elements.[2] A discontinuous least-sqares method was published by Potanza and Reddy[3] but they did not address the subject of shock capturing.

Another approach to shock capturing is to increase the degree of the interpolation polynomial. Like mesh refinement, polynomial refinement provides more degrees of freedom to approximate discontinuities and also requires refinement in neighboring elements unless discontinuous methods are used. If nodal interpolation functions are used, polynomial refinement requires adding more nodes to the elements. An alternative that does not require additional nodes to increase the polynomial degree is modal interpolation functions. When using modal functions, the finite element method solves for coefficients of the hierarchical modes of the functions.[4]

The approach to shock modeling proposed here is to do away with polynomial interpolation functions in elements adjacent to shock waves in favor of exponential

interpolation functions. Since Heaviside functions can be approximated by continuous exponential functions, the hypothesis of this research is that exponential interpolation functions will approximate fluid properties at shock waves with less error than polynomial interpolation functions. The motivation to investigate the suitability of exponential functions is that the potential reduction of interpolation error near shock waves may also reduce or eliminate the need for mesh refinement in those regions.

There are only a few published works in which Heaviside functions are used for finite element modeling. Meiring and Rosinger[5] used basis functions composed of products of Heaviside and continuous functions to solve nonlinear partial differential equations. Their results did not show an overall increase or decrease in error and they conclude that their basis functions lead to a "system that is too loosely connected" and that "the possibility exists of improving the method by choosing basis functions which are not completely disjointed." Ichimura, Hori, and Wijerathne[6] applied Heaviside basis functions to finite element simulations of earthquake ground displacement. However, their objective was to obtain a diagonalized mass matrix without lumping, which can increase numerical error. The use of continuous approximations of Heaviside functions to model sharp gradients in the finite element method appears to have received very little attention.

In Chapter 2, the exponential interpolation functions are derived and some properties of the functions are demonstrated. General expressions for conservation of mass, conservation of momentum, and conservation of energy are converted to the dimensionless Euler equations in Chapter 3. In Chapter 4, the Euler equations are linearized using Newton's method, the least-squares finite element method is used to generate the element equations, and the boundary conditions are explained. A shock reflection example is presented in Chapter 5. Numerical solutions of the shock reflection example are calculated using polynomial and exponential interpolation. The analytical solution is used to calculate the absolute error in the numerical solution and the performance of the exponential and polynomial interpolation functions is discussed.

CHAPTER 2
EXPONENTIAL INTERPOLATION FUNCTIONS

This chapter introduces the exponential interpolation functions. Section 2.1 presents a detailed derivation of one-dimensional exponential interpolation functions along with a more general approach using Cramer's rule. In Section 2.2, exponential parameters are selected for interpolation of large gradients. Limits are presented to demonstrate the behavior of the exponential interpolation functions and several examples are used to show potential mistakes that could lead to large interpolation errors. An empirical equation is developed in Section 2.3 to determine the necessary number of quadrature points to integrate the exponential interpolation functions accurately. In Section 2.4, two-dimensional interpolation functions are derived using products of exponential interpolation functions and Lagrange polynomials.

2.1 Derivation

The procedure to derive the exponential interpolation functions is demonstrated for a one-dimensional element with two nodes and C^0 continuity. The nodes are located at $\eta = \pm 1$. Start with a general approximation for some dependent variable u.

$$u(\eta) \approx g_1 + g_2 e^{h_1 \eta} \tag{1}$$

The variables g_1 and g_2 do not have any physical meaning. They are only used in the procedure to derive the interpolation equations. They are found by imposing the condition that $u(\eta_k) = u_k$, where u_k is the value of the dependent variable u at node k and η_k is the location of node k, and solving the resulting set of equations.

$$u(-1) = u_1 = g_1 + g_2 e^{-h_1} \tag{2}$$

$$u(1) = u_2 = g_1 + g_2 e^{h_1} \tag{3}$$

Rearrange Eq. (3) and substitute it into Eq. (2).

$$g_1 = u_2 - g_2 e^{h_1} \tag{4}$$

$$u_1 = u_2 - g_2 e^{h_1} + g_2 e^{-h_1} \tag{5}$$

Solve Eq. (5) for g_2.

$$g_2 = \frac{u_1}{e^{-h_1} - e^{h_1}} - \frac{u_2}{e^{-h_1} - e^{h_1}} \tag{6}$$

Equation (6) is substituted into Eq. (4), which is solved for g_1.

$$g_1 = u_2 - \frac{u_1 e^{h_1}}{e^{-h_1} - e^{h_1}} + \frac{u_2 e^{h_1}}{e^{-h_1} - e^{h_1}} \tag{7}$$

Equations (6) and (7) are substituted into Eq. (1) and the result is rearranged.

$$\begin{aligned} u(\eta) &\approx u_2 - \frac{u_1 e^{h_1}}{e^{-h_1} - e^{h_1}} + \frac{u_2 e^{h_1}}{e^{-h_1} - e^{h_1}} + \left(\frac{u_1}{e^{-h_1} - e^{h_1}} - \frac{u_2}{e^{-h_1} - e^{h_1}} \right) e^{h_1 \eta} \\ &= \left(\frac{e^{h_1 \eta} - e^{h_1}}{e^{-h_1} - e^{h_1}} \right) u_1 + \left(1 + \frac{e^{h_1} - e^{h_1 \eta}}{e^{-h_1} - e^{h_1}} \right) u_2 \\ &= \left(\frac{e^{h_1} - e^{h_1 \eta}}{2 \sinh h_1} \right) u_1 + \left(\frac{e^{h_1 \eta} - e^{-h_1}}{2 \sinh h_1} \right) u_2 \end{aligned} \tag{8}$$

Equation (8) is rewritten to resemble the typical representation of approximation equations used in the finite element method.

$$u(\eta) \approx \varepsilon_1 u_1 + \varepsilon_2 u_2 = \sum_{k=1}^{2} \varepsilon_k u_k \tag{9}$$

$$\varepsilon_1 = \frac{e^{h_1} - e^{h_1 \eta}}{2 \sinh h_1} \tag{10}$$

$$\varepsilon_2 = \frac{e^{h_1 \eta} - e^{-h_1}}{2 \sinh h_1} \tag{11}$$

Hereafter ε_k is used to represent the exponential interpolation functions. For higher degree functions, the preceding steps become tedious. A simpler way to generate high degree exponential interpolation functions is to use Cramer's rule. The first step is to write a general approximation function.

$$u(\eta) \approx g_1 + g_2 e^{h_1 \eta} + g_3 e^{h_2 \eta^2} + g_4 e^{h_3 \eta^3} \tag{12}$$

Again, the condition that $u(\eta_k) = u_k$ is imposed. The resulting set of equations is arranged in matrix form.

$$\begin{bmatrix} 1 & e^{h_1\eta_1} & e^{h_2\eta_1^2} & e^{h_3\eta_1^3} \\ 1 & e^{h_1\eta_2} & e^{h_2\eta_2^2} & e^{h_3\eta_2^3} \\ 1 & e^{h_1\eta_3} & e^{h_2\eta_3^2} & e^{h_3\eta_3^3} \\ 1 & e^{h_1\eta_4} & e^{h_2\eta_4^2} & e^{h_3\eta_4^3} \end{bmatrix} \begin{Bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{Bmatrix} = \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{Bmatrix} \tag{13}$$

The next step in Cramer's rule is to compute the determinant of the 4×4 matrix in Eq. (13).

$$D = \begin{vmatrix} 1 & e^{h_1\eta_1} & e^{h_2\eta_1^2} & e^{h_3\eta_1^3} \\ 1 & e^{h_1\eta_2} & e^{h_2\eta_2^2} & e^{h_3\eta_2^3} \\ 1 & e^{h_1\eta_3} & e^{h_2\eta_3^2} & e^{h_3\eta_3^3} \\ 1 & e^{h_1\eta_4} & e^{h_2\eta_4^2} & e^{h_3\eta_4^3} \end{vmatrix} \tag{14}$$

The first column of the 4×4 matrix in Eq. (13) is replaced with $\{u_1\, u_2\, u_3\, u_4\}^T$ and the determinant is calculated.

$$D_1 = \begin{vmatrix} u_1 & e^{h_1\eta_1} & e^{h_2\eta_1^2} & e^{h_3\eta_1^3} \\ u_2 & e^{h_1\eta_2} & e^{h_2\eta_2^2} & e^{h_3\eta_2^3} \\ u_3 & e^{h_1\eta_3} & e^{h_2\eta_3^2} & e^{h_3\eta_3^3} \\ u_4 & e^{h_1\eta_4} & e^{h_2\eta_4^2} & e^{h_3\eta_4^3} \end{vmatrix} \tag{15}$$

The previous step is repeated, replacing each column with $\{u_1\, u_2\, u_3\, u_4\}^T$ and calculating the determinant.

$$D_2 = \begin{vmatrix} 1 & u_1 & e^{h_2\eta_1^2} & e^{h_3\eta_1^3} \\ 1 & u_2 & e^{h_2\eta_2^2} & e^{h_3\eta_2^3} \\ 1 & u_3 & e^{h_2\eta_3^2} & e^{h_3\eta_3^3} \\ 1 & u_4 & e^{h_2\eta_4^2} & e^{h_3\eta_4^3} \end{vmatrix} \tag{16}$$

$$D_3 = \begin{vmatrix} 1 & e^{h_1\eta_1} & u_1 & e^{h_3\eta_1^3} \\ 1 & e^{h_1\eta_2} & u_2 & e^{h_3\eta_2^3} \\ 1 & e^{h_1\eta_3} & u_3 & e^{h_3\eta_3^3} \\ 1 & e^{h_1\eta_4} & u_4 & e^{h_3\eta_4^3} \end{vmatrix} \tag{17}$$

$$D_4 = \begin{vmatrix} 1 & e^{h_1\eta_1} & e^{h_2\eta_1^2} & u_1 \\ 1 & e^{h_1\eta_2} & e^{h_2\eta_2^2} & u_2 \\ 1 & e^{h_1\eta_3} & e^{h_2\eta_3^2} & u_3 \\ 1 & e^{h_1\eta_4} & e^{h_2\eta_4^2} & u_4 \end{vmatrix} \tag{18}$$

The determinants D_1, D_2, D_3, and D_4 can be computed such that the result is a sum of coefficients multiplied by u_1, u_2, u_3, and u_4. The determinant D is constant for a given set of exponential parameters, h_i, and node locations, η_k. The coefficients of the interpolation equation are calculated using Eq. (19).

$$g_i = \frac{D_i}{D} \tag{19}$$

The coefficients g_i are then substituted into Eq. (12) and the result is rearranged to obtain Eq. (20). Cramer's rule can be implemented in software to generate interpolation functions of any degree.

$$u\left(\eta\right) \approx \sum_{k=1}^{4} \varepsilon_k u_k \tag{20}$$

One-dimensional exponential interpolation functions for a five node element with nodes located at $\eta = \pm 1$, $\pm\frac{1}{2}$, and 0 are derived starting with Eq. (21).

$$u(\eta) \approx g_1 + g_2 e^{h_1\eta} + g_3 e^{h_2\eta^2} + g_4 e^{h_3\eta^3} + g_5 e^{h_4\eta^4} \tag{21}$$

The resulting interpolation equations are shown without the remaining derivation steps. They will be used to demonstrate the effect of various choices of the exponential parameters.

$$
\begin{aligned}
&\varepsilon_1(\eta) = \\
&\left(e^{\eta h_1}-1\right)\frac{\sinh\frac{h_3}{8}}{2\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)} \\
&+\left(e^{\eta^2 h_2}-1\right) \\
&\times\left[\frac{\sinh\frac{h_1}{2}\left(\left(e^{h_3}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{-\frac{h_3}{8}}-1\right)\left(e^{h_4}-1\right)\right)+\sinh\frac{h_3}{8}\left(\left(e^{-\frac{h_1}{2}}-1\right)\left(e^{h_4}-1\right)-\left(e^{h_1}-1\right)\left(e^{\frac{h_4}{16}}-1\right)\right)}{2\left(\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\right)\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)}\right] \\
&+\left(e^{\eta^3 h_3}-1\right)\frac{-\sinh\frac{h_1}{2}}{2\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)} \\
&+\left(e^{\eta^4 h_4}-1\right) \\
&\times\left[\frac{\sinh\frac{h_1}{2}\left(\left(e^{h_2}-1\right)\left(e^{-\frac{h_3}{8}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_3}-1\right)\right)+\sinh\frac{h_3}{8}\left(\left(e^{h_1}-1\right)\left(e^{\frac{h_2}{4}}-1\right)-\left(e^{-\frac{h_1}{2}}-1\right)\left(e^{h_2}-1\right)\right)}{2\left(\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\right)\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)}\right]
\end{aligned}
\tag{22}
$$

$$
\begin{aligned}
&\varepsilon_2(\eta) = \\
&\left(e^{\eta h_1}-1\right)\frac{-\sinh h_3}{2\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)} \\
&+\left(e^{\eta^2 h_2}-1\right) \\
&\times\left[\frac{\sinh h_1\left(\left(e^{\frac{h_3}{8}}-1\right)\left(e^{h_4}-1\right)-\left(e^{-h_3}-1\right)\left(e^{\frac{h_4}{16}}-1\right)\right)+\sinh h_3\left(\left(e^{-h_1}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_1}{2}}-1\right)\left(e_4^h-1\right)\right)}{2\left(\left(e_2^h-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\right)\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)}\right] \\
&+\left(e^{\eta^3 h_3}-1\right)\frac{\sinh h_1}{2\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)} \\
&+\left(e^{\eta^4 h_4}-1\right) \\
&\times\left[\frac{\sinh h_1\left(\left(e^{\frac{h_2}{4}}-1\right)\left(e^{-h_3}-1\right)-\left(e^{h_2}-1\right)\left(e^{\frac{h_3}{8}}-1\right)\right)+\sinh h_3\left(\left(e^{\frac{h_1}{2}}-1\right)\left(e^{h_2}-1\right)-\left(e^{-h_1}-1\right)\left(e^{\frac{h_2}{4}}-1\right)\right)}{2\left(\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\right)\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)}\right]
\end{aligned}
\tag{23}
$$

$$
\begin{aligned}
&\varepsilon_3(\eta) = \\
&\left(e^{\eta^2 h_2}-1\right)\frac{\left(e^{h_4}-1\right)-\left(e^{\frac{h_4}{16}}-1\right)}{\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)} \\
&+\left(e^{\eta^4 h_4}-1\right)\frac{\left(e^{\frac{h_2}{4}}-1\right)-\left(e^{h_2}-1\right)}{\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)}+1
\end{aligned}
\tag{24}
$$

$$
\begin{aligned}
&\varepsilon_4(\eta) = \\
&\left(e^{\eta h_1}-1\right)\frac{\sinh h_3}{2\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)} \\
&+\left(e^{\eta^2 h_2}-1\right) \\
&\times\left[\frac{\sinh h_1\left(\left(e^{-h_3}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{-\frac{h_3}{8}}-1\right)\left(e_4^{h}-1\right)\right)+\sinh h_3\left(\left(e^{-\frac{h_1}{2}}-1\right)\left(e^{h_4}-1\right)-\left(e^{-h_1}-1\right)\left(e^{\frac{h_4}{16}}-1\right)\right)}{2\left(\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\right)\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)}\right] \\
&+\left(e^{\eta^3 h_3}-1\right)\frac{-\sinh h_1}{2\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)} \\
&+\left(e^{\eta^4 h_4}-1\right) \\
&\times\left[\frac{\sinh h_1\left(\left(e^{h_2}-1\right)\left(e^{-\frac{h_3}{8}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{-h_3}-1\right)\right)+\sinh h_3\left(\left(e^{-h_1}-1\right)\left(e^{\frac{h_2}{4}}-1\right)-\left(e^{-\frac{h_1}{2}}-1\right)\left(e^{h_2}-1\right)\right)}{2\left(\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\right)\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)}\right]
\end{aligned}
\tag{25}
$$

$$
\begin{aligned}
&\varepsilon_5(\eta) = \\
&\left(e^{\eta h_1}-1\right)\frac{-\sinh\frac{h_3}{8}}{2\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)} \\
&+\left(e^{\eta^2 h_2}-1\right) \\
&\times\left[\frac{\sinh\frac{h_1}{2}\left(\left(e^{-\frac{h_3}{8}}-1\right)\left(e^{h_4}-1\right)-\left(e^{-h_3}-1\right)\left(e^{\frac{h_4}{16}}-1\right)\right)+\sinh\frac{h_3}{8}\left(\left(e^{-h_1}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{-\frac{h_1}{2}}-1\right)\left(e^{h_4}-1\right)\right)}{2\left(\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\right)\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)}\right] \\
&+\left(e^{\eta^3 h_3}-1\right)\frac{\sinh\frac{h_1}{2}}{2\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)} \\
&+\left(e^{\eta^4 h_4}-1\right) \\
&\times\left[\frac{\sinh\frac{h_1}{2}\left(\left(e^{\frac{h_2}{4}}-1\right)\left(e^{-h_3}-1\right)-\left(e^{h_2}-1\right)\left(e^{-\frac{h_3}{8}}-1\right)\right)+\sinh\frac{h_3}{8}\left(\left(e^{-\frac{h_1}{2}}-1\right)\left(e^{h_2}-1\right)-\left(e^{-h_1}-\right)\left(e^{\frac{h_2}{4}}-1\right)\right)}{2\left(\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\right)\left(\sinh\frac{h_1}{2}\sinh h_3-\sinh h_1\sinh\frac{h_3}{8}\right)}\right]
\end{aligned}
\tag{26}
$$

2.2 Selection of the Exponent Parameters

In order to completely define the exponential interpolation functions, the exponential parameters, h_i, must be selected to suit the problem at hand. For inviscid compressible flows, the fluid properties experience a sudden jump at a shock wave, similar to a Heaviside step function. The dependent variable u defined in Eq. (27) is approximated by Eq. (28). The nodal values, u_k, in Eq. (28) are the exact values of u at the nodes located at $\eta = \pm 1$, $\pm\frac{1}{2}$, and 0.

$$u(\eta) = \mathrm{H}(\eta - \eta_0) \tag{27}$$

$$u(\eta) \approx \sum_{k=1}^{5} \varepsilon_k u_k \tag{28}$$

The `fmincon` function in MATLAB was used to minimize the norm of the difference between Eqs. (27) and (28) over $-1 \leq \eta \leq 1$ by changing the exponential parameters. The optimization algorithm was constrained to search for values of h_i between ± 700 to prevent MATLAB from evaluating ε_k as $\pm\infty$. There is more than one combination of exponential parameters that yield satisfactory results and the exponential parameters returned by `fmincon` are sensitive to the initial guess. A simple set of parameters was found using trial and error to vary the inputs to the `fmincon` function. For $\eta_0 = 1$, $h = \{0, 0, 700, 0\}$. For $\eta_0 = -1$, $h = \{0, 0, -700, 0\}$. More generally, $h_j = 700$ is used to approximate a discontinuity at the right edge of an element and $h_j = -700$ is used to approximate a discontinuity at the left edge of an element where j is the highest odd numbered index of the exponential parameters. Suitable sets of parameters were not found for any case where $\eta_0 \neq \pm 1$.

A brief examination of Eqs. (22) to (26) shows that setting any exponential parameter equal to zero will result in division by zero. To circumvent that problem and to better illustrate the behavior of the exponential interpolation functions, the limits as h_1, h_2, and h_4 approach zero and h_3 approaches infinity were calculated using L'Hopital's rule. The calculation steps are only shown for Eq. (22). The steps to calculate the limits of Eqs. (23) to (26) are the same.

First, the derivatives of the numerators are calculated.

$$\begin{aligned}
&\frac{\partial^3}{\partial h_1 \partial h_2 \partial h_4}\left\{\left(e^{\eta h_1}-1\right)\left(\sinh\frac{h_3}{8}\right)\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\eta h_1}-1\right)\left(\sinh\frac{h_3}{8}\right)\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\right\} \\
&\quad=\left(\eta e^{\eta h_1}\right)\left(\sinh\frac{h_3}{8}\right)\left(e^{h_4}\right)\left(\frac{1}{16}e^{\frac{h_4}{16}}\right)-\left(\eta e^{\eta h_1}\right)\left(\sinh\frac{h_3}{8}\right)\left(\frac{1}{4}e^{\frac{h_2}{4}}\right)\left(e^{h_4}\right)
\end{aligned} \tag{29}$$

$$\begin{aligned}
&\frac{\partial^3}{\partial h_1 \partial h_2 \partial h_4}\left\{\left(e^{\eta^2 h_2}-1\right)\left(\sinh\frac{h_1}{2}\right)\left(e^{h_3}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\eta^2 h_2}-1\right)\left(\sinh\frac{h_1}{2}\right)\left(e^{-\frac{h_3}{8}}-1\right)\left(e^{h_4}-1\right)\right. \\
&\quad\left.+\left(e^{\eta^2 h_2}-1\right)\left(\sinh\frac{h_3}{8}\right)\left(e^{-\frac{h_1}{2}}-1\right)\left(e^{h_4}-1\right)-\left(e^{\eta^2 h_2}-1\right)\left(\sinh\frac{h_3}{8}\right)\left(e^{h_1}-1\right)\left(e^{\frac{h_4}{16}}-1\right)\right\} \\
&\quad=\left(\eta^2 e^{\eta^2 h_2}\right)\left(\frac{1}{2}\cosh\frac{h_1}{2}\right)\left(e^{h_3}-1\right)\left(\frac{1}{16}e^{\frac{h_4}{16}}\right)-\left(\eta^2 e^{\eta^2 h_2}\right)\left(\frac{1}{2}\cosh\frac{h_1}{2}\right)\left(e^{-\frac{h_3}{8}}-1\right)\left(e^{h_4}\right) \\
&\quad+\left(\eta^2 e^{\eta^2 h_2}\right)\left(\sinh\frac{h_3}{8}\right)\left(-\frac{1}{2}e^{-\frac{h_1}{2}}\right)\left(e^{h_4}\right)-\left(\eta^2 e^{\eta^2 h_2}\right)\left(\sinh\frac{h_3}{8}\right)\left(e^{h_1}\right)\left(\frac{1}{16}e^{\frac{h_4}{16}}\right)
\end{aligned} \tag{30}$$

$$\begin{aligned}
&\frac{\partial^3}{\partial h_1 \partial h_2 \partial h_4}\left\{\left(e^{\eta^3 h_3}-1\right)\left(-\sinh\frac{h_1}{2}\right)\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)-\left(e^{\eta^3 h_3}-1\right)\left(-\sinh\frac{h_1}{2}\right)\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\right\} \\
&\quad=\left(e^{\eta^3 h_3}-1\right)\left(-\frac{1}{2}\cosh\frac{h_1}{2}\right)\left(e^{h_2}\right)\left(\frac{1}{16}e^{\frac{h_4}{16}}\right)\left(e^{\eta^3 h_3}-1\right)\left(-\frac{1}{2}\cosh\frac{h_1}{2}\right)\left(\frac{1}{4}e^{\frac{h_2}{4}}\right)\left(e^{h_4}\right)
\end{aligned} \tag{31}$$

$$\begin{aligned}
&\frac{\partial^3}{\partial h_1 \partial h_2 \partial h_4}\left\{\left(e^{\eta^4 h_4}-1\right)\left(\sinh\frac{h_1}{2}\right)\left(e^{h_2}-1\right)\left(e^{-\frac{h_3}{8}}-1\right)-\left(e^{\eta^4 h_4}-1\right)\left(\sinh\frac{h_1}{2}\right)\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_3}-1\right)\right. \\
&\quad\left.+\left(e^{\eta^4 h_4}-1\right)\left(\sinh\frac{h_3}{8}\right)\left(e^{h_1}-1\right)\left(e^{\frac{h_2}{4}}-1\right)-\left(e^{\eta^4 h_4}-1\right)\left(\sinh\frac{h_3}{8}\right)\left(e^{-\frac{h_1}{2}}-1\right)\left(e^{h_2}-1\right)\right\} \\
&\quad=\left(\eta^4 e^{\eta^4 h_4}\right)\left(\frac{1}{2}\cosh\frac{h_1}{2}\right)\left(e^{h_2}\right)\left(e^{-\frac{h_3}{8}}-1\right)-\left(\eta^4 e^{\eta^4 h_4}\right)\left(\frac{1}{2}\cosh\frac{h_1}{2}\right)\left(\frac{1}{4}e^{\frac{h_2}{4}}\right)\left(e^{h_3}-1\right) \\
&\quad\left(\eta^4 e^{\eta^4 h_4}\right)\left(\sinh\frac{h_3}{8}\right)\left(e^{h_1}\right)\left(\frac{1}{4}e^{\frac{h_2}{4}}\right)-\left(\eta^4 e^{\eta^4 h_4}\right)\left(\sinh\frac{h_3}{8}\right)\left(-\frac{1}{2}e^{-\frac{h_1}{2}}\right)\left(e^{h_2}\right)
\end{aligned} \tag{32}$$

Then the derivative of the denominator is calculated.

$$
\begin{aligned}
&\frac{\partial^3}{\partial h_1 \partial h_2 \partial h_4} \left\{ 2\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)\left(\sinh\frac{h_1}{2}\right)\left(\sinh h_3\right) - 2\left(e^{h_2}-1\right)\left(e^{\frac{h_4}{16}}-1\right)\left(\sinh h_1\right)\left(\sinh\frac{h_3}{8}\right) \right. \\
&\quad \left. - 2\left(e^{\frac{h_2}{4}}-1\right)\left(e^{h_4}-1\right)\left(\sinh\frac{h_1}{2}\right)\left(\sinh h_3\right) + 2\left(e^{h_2}4-1\right)\left(e^{h_4}-1\right)\left(\sinh h_1\right)\left(\sinh\frac{h_3}{8}\right) \right\} \\
&\quad = 2\left(e^{h_2}\right)\left(\frac{1}{16}e^{\frac{h_4}{16}}\right)\left(\frac{1}{2}\cosh\frac{h_1}{2}\right)\left(\sinh h_3\right) - 2\left(e^{h_2}\right)\left(\frac{1}{16}e^{\frac{h_4}{16}}\right)\left(\cosh h_1\right)\left(\sinh\frac{h_3}{8}\right) \\
&\quad - 2\left(\frac{1}{4}e^{\frac{h_2}{4}}\right)\left(e^{h_4}\right)\left(\frac{1}{2}\cosh\frac{h_1}{2}\right)\left(\sinh h_3\right) + 2\left(\frac{1}{4}e^{\frac{h_2}{4}}\right)\left(e^{h_4}\right)\left(\cosh h_1\right)\left(\sinh\frac{h_3}{8}\right)
\end{aligned}
\tag{33}
$$

Next, $h_1 = h_2 = h_4 = 0$ is subtituted into the numerators and the denominator.

$$\begin{aligned}\lim_{h_1,h_2,h_4\to 0}\varepsilon_1 =&\frac{1}{2\sinh\frac{h_3}{8}-\sinh h_3}\left[\frac{1}{2}\left(e^{\eta^3 h_3}-1\right)+\left(-\eta-3\eta^2+4\eta^4\right)\left(\sinh\frac{h_3}{8}\right)\right.\\&\left.\frac{8}{3}\left(\eta^4-\eta^2\right)\left(e^{-\frac{h_3}{8}}-1\right)+\frac{1}{6}\left(\eta^2-4\eta^4\right)\left(e^{h_3}-1\right)\right]\end{aligned} \tag{34}$$

Equation (34) is rearranged into a more convenient form.

$$\begin{aligned}\lim_{h_1,h_2,h_4\to 0}\varepsilon_1 =&\frac{1}{2\left(e^{\frac{h_3}{8}}-e^{-\frac{h_3}{8}}\right)-\left(e^{h_3}-e^{-h_3}\right)}\left(\frac{1}{3}\right)\\&\left[3\left(e^{\eta^3 h_3}-1\right)+3\left(4\eta^4-3\eta^2-\eta\right)\left(e^{\frac{h_3}{8}}-1\right)+\right.\\&\left.\left(4\eta^4-7\eta^2+3\eta\right)\left(e^{-\frac{h_3}{8}}-1\right)+\left(\eta^2-4\eta^4\right)\left(e^{h_3}-1\right)\right]\end{aligned} \tag{35}$$

To find the limit as h_3 approaches infinity, a change of variables is used.

$$h_3 = 8\ln H_3 \tag{36}$$

As H_3 approaches infinity, h_3 also approaches infinity.

$$\lim_{h_3\to\infty}\left[\frac{e^{\frac{h_3}{8}}-1}{2\left(e^{\frac{h_3}{8}}-e^{-\frac{h_3}{8}}\right)-\left(e^{h_3}-e^{-h_3}\right)}\right]=\lim_{H_3\to\infty}\left[\frac{H_3^9-H_3^8}{-H_3^{16}+2H_3^9-2H_3^7+1}\right]=0 \tag{37}$$

$$\lim_{h_3\to\infty}\left[\frac{e^{-\frac{h_3}{8}}-1}{2\left(e^{\frac{h_3}{8}}-e^{-\frac{h_3}{8}}\right)-\left(e^{h_3}-e^{-h_3}\right)}\right]=\lim_{H_3\to\infty}\left[\frac{-H_3^8+H_3^7}{-H_3^{16}+2H_3^9-2H_3^7+1}\right]=0 \tag{38}$$

$$\lim_{h_3\to\infty}\left[\frac{e^{h_3}-1}{2\left(e^{\frac{h_3}{8}}-e^{-\frac{h_3}{8}}\right)-\left(e^{h_3}-e^{-h_3}\right)}\right]=\lim_{H_3\to\infty}\left[\frac{H_3^{16}-H_3^8}{-H_3^{16}+2H_3^9-2H_3^7+1}\right]=-1 \tag{39}$$

$$\lim_{h_3\to\infty}\left[\frac{e^{\eta^3 h_3}-1}{2\left(e^{\frac{h_3}{8}}-e^{-\frac{h_3}{8}}\right)-\left(e^{h_3}-e^{-h_3}\right)}\right]=$$
$$\lim_{H_3\to\infty}\left[\frac{H_3^{16\eta^3}-H_3^8}{-H_3^{16}+2H_3^9-2H_3^7+1}\right]=\begin{cases}0 & \eta\in[-1,1)\\ -1 & \eta=1\end{cases} \tag{40}$$

Equation (40) behaves like a Heaviside step function, which is represented by $\mathrm{H}(\eta)$. Throughout this thesis, the convention that $\mathrm{H}(0)=\frac{1}{2}$ is used. Equations (37) to (40) are substituted into (35).

$$\lim_{h_3\to\infty}\left(\lim_{h_1,h_2,h_4\to 0}\varepsilon_1\right)=2\mathrm{H}(-\eta+1)-2-\frac{1}{3}\eta^2+\frac{4}{3}\eta^4 \tag{41}$$

The limits of the other one-dimensional exponential interpolation functions were calculated using the same procedure. Plots of the functions for $h=\{0,0,700,0\}$, which is a continuous approximation for $h=\{0,0,\infty,0\}$, are shown in Figure 1.

$$\lim_{h_3\to\infty}\left(\lim_{h_1,h_2,h_4\to 0}\varepsilon_2\right)=-4\mathrm{H}(-\eta+1)+4-\eta+3\eta^2-4\eta^4 \tag{42}$$

$$\lim_{h_2,h_4\to 0}\varepsilon_3=1-5\eta^2+4\eta^4 \tag{43}$$

$$\lim_{h_3\to\infty}\left(\lim_{h_1,h_2,h_4\to 0}\varepsilon_4\right)=4\mathrm{H}(-\eta+1)-4+\eta+\frac{7}{3}\eta^2-\frac{4}{3}\eta^4 \tag{44}$$

$$\lim_{h_3\to\infty}\left(\lim_{h_1,h_2,h_4\to 0}\varepsilon_5\right)=-2\mathrm{H}(-\eta+1)+2 \tag{45}$$

Again, using a similar procedure the limits as h_3 approaches $-\infty$ are found. Plots of the functions for $h=\{0,0,-700,0\}$, which is a continuous approximation for $h=\{0,0,-\infty,0\}$, are shown in Figure 2.

$$\lim_{h_3\to-\infty}\left(\lim_{h_1,h_2,h_4\to 0}\varepsilon_1\right)=-2\mathrm{H}(\eta+1)+2 \tag{46}$$

$$\lim_{h_3\to-\infty}\left(\lim_{h_1,h_2,h_4\to 0}\varepsilon_2\right)=4\mathrm{H}(\eta+1)-4-\eta+\frac{7}{3}\eta^2-\frac{4}{3}\eta^4 \tag{47}$$

$$\lim_{h_2,h_4\to 0}\varepsilon_3=1-5\eta^2+4\eta^4 \tag{48}$$

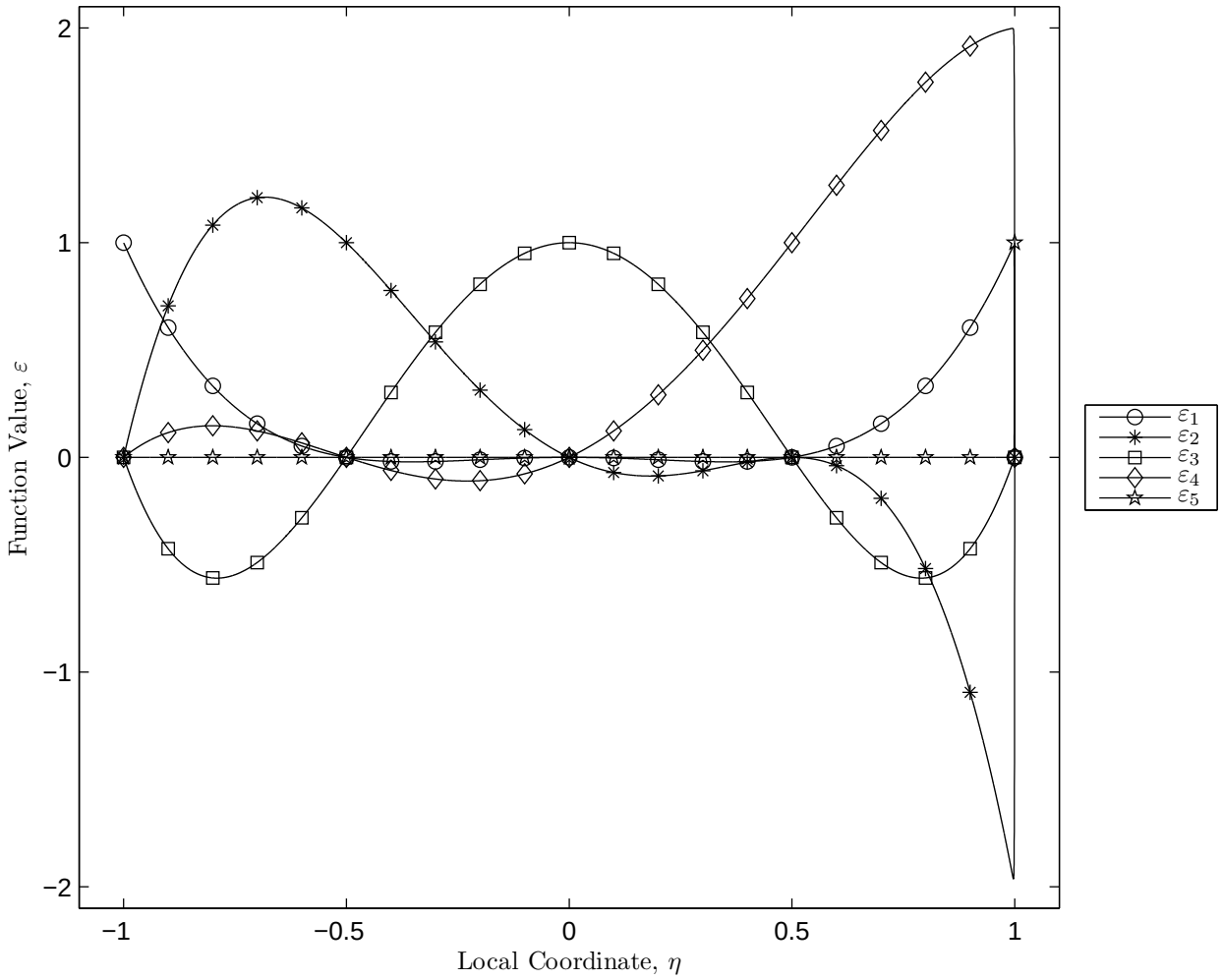

Figure 1. One-dimensional exponential interpolation functions for $h = \{0, 0, \infty, 0\}$.

$$\lim_{h_3 \to -\infty} \left(\lim_{h_1, h_2, h_4 \to 0} \varepsilon_4 \right) = -4\mathrm{H}\left(\eta + 1\right) + 4 + \eta + 3\eta^2 - 4\eta^4 \tag{49}$$

$$\lim_{h_3 \to -\infty} \left(\lim_{h_1, h_2, h_4 \to 0} \varepsilon_5 \right) = 2\mathrm{H}\left(\eta + 1\right) - 2 - \frac{1}{3}\eta^2 + \frac{4}{3}\eta^4 \tag{50}$$

The functions shown here can be generated without taking a limit by starting with Eq. (51) or Eq. (52) for the cases where h approaches $\{0, 0, \infty, 0\}$ or $\{0, 0, -\infty, 0\}$, respectively.

$$u\left(\eta\right) \approx g_1 + g_2\eta + g_3\eta^2 + g_4 2\mathrm{H}\left(\eta - 1\right) + g_5\eta^4 \tag{51}$$

$$u\left(\eta\right) \approx g_1 + g_2\eta + g_3\eta^2 + g_4 2\mathrm{H}\left(-\eta - 1\right) + g_5\eta^4 \tag{52}$$

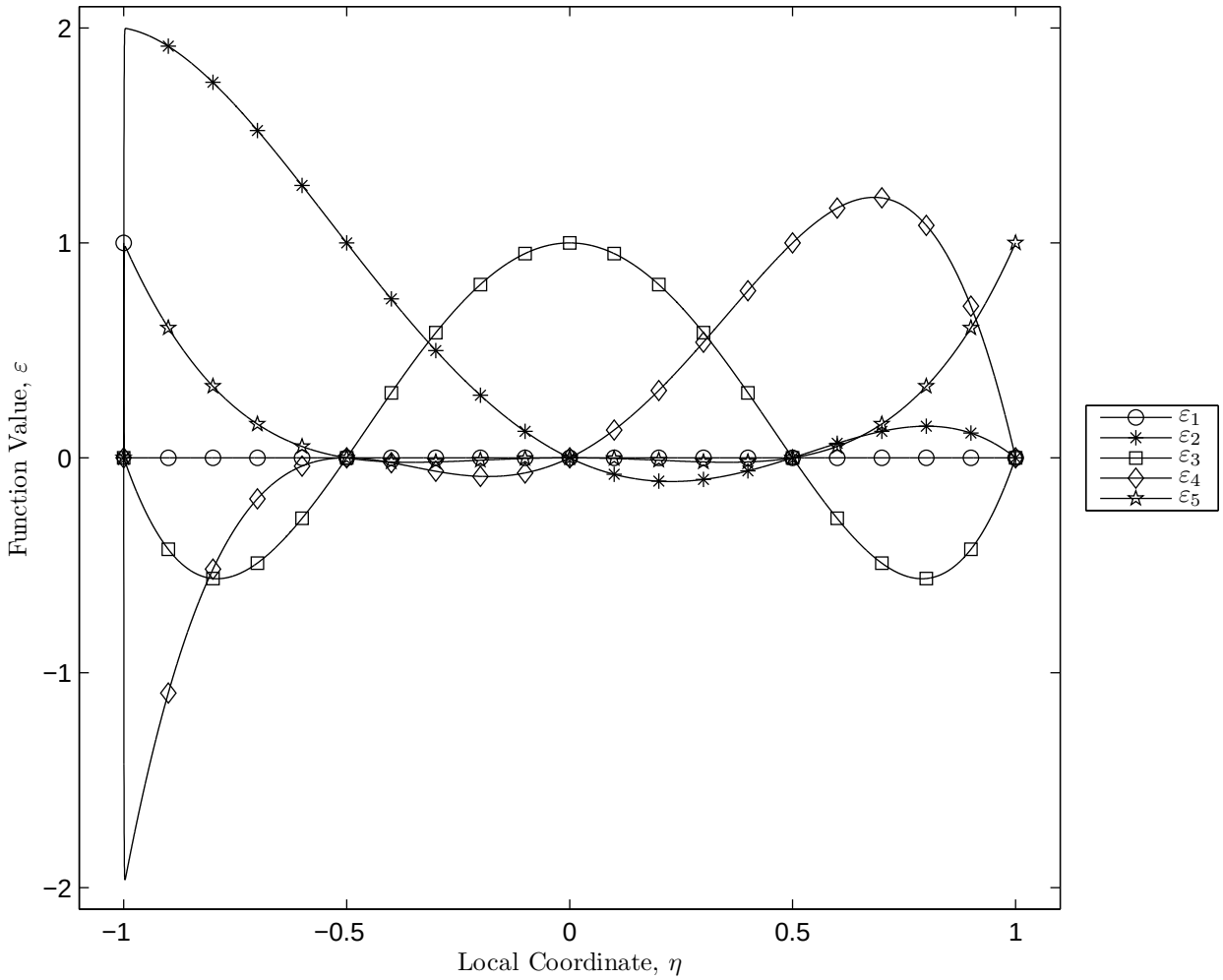

Figure 2. One-dimensional exponential interpolation functions for $h = \{0, 0, -\infty, 0\}$**.**

The condition that $u(\eta_k) = u_k$ is imposed. The resulting set of equations is arranged in matrix form as shown in Eqs. (53) and (54), Cramer's rule is used to find g_k, and the remaining steps in Section 2.1 are applied.

$$\begin{bmatrix} 1 & \eta_1 & \eta_1^2 & 0 & \eta_1^4 \\ 1 & \eta_2 & \eta_2^2 & 0 & \eta_2^4 \\ 1 & \eta_3 & \eta_3^2 & 0 & \eta_3^4 \\ 1 & \eta_4 & \eta_4^2 & 0 & \eta_4^4 \\ 1 & \eta_5 & \eta_5^2 & 1 & \eta_5^4 \end{bmatrix} \begin{Bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \end{Bmatrix} = \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} \tag{53}$$

$$\begin{bmatrix} 1 & \eta_1 & \eta_1^2 & 1 & \eta_1^4 \\ 1 & \eta_2 & \eta_2^2 & 0 & \eta_2^4 \\ 1 & \eta_3 & \eta_3^2 & 0 & \eta_3^4 \\ 1 & \eta_4 & \eta_4^2 & 0 & \eta_4^4 \\ 1 & \eta_5 & \eta_5^2 & 0 & \eta_5^4 \end{bmatrix} \begin{Bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \end{Bmatrix} = \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} \tag{54}$$

One other noteworthy limit is that in which all of the exponential parameters approach zero. The exponential interpolation functions become Lagrange polynomials. Plots of the functions are shown in Figure 3.

$$\lim_{h_1,h_2,h_3,h_4\to 0} \varepsilon_1 = \frac{1}{6}\eta - \frac{1}{6}\eta^2 - \frac{2}{3}\eta^3 + \frac{2}{3}\eta^4 \tag{55}$$

$$\lim_{h_1,h_2,h_3,h_4\to 0} \varepsilon_2 = -\frac{4}{3}\eta + \frac{8}{3}\eta^2 + \frac{4}{3}\eta^3 - \frac{8}{3}\eta^4 \tag{56}$$

$$\lim_{h_2,h_4\to 0} \varepsilon_3 = 1 - 5\eta^2 + 4\eta^4 \tag{57}$$

$$\lim_{h_1,h_2,h_3,h_4\to 0} \varepsilon_4 = \frac{4}{3}\eta + \frac{8}{3}\eta^2 - \frac{4}{3}\eta^3 - \frac{8}{3}\eta^4 \tag{58}$$

$$\lim_{h_1,h_2,h_3,h_4\to 0} \varepsilon_5 = -\frac{1}{6}\eta - \frac{1}{6}\eta^2 + \frac{2}{3}\eta^3 + \frac{2}{3}\eta^4 \tag{59}$$

Equations (41) to (50) show that the third degree of the interpolation functions is traded for a step at either the left or right edge of an element and that the resulting interpolation functions are not complete polynomials. To ensure that the polynomials are complete, elements should contain an even number of nodes. In that case, the highest polynomial degree is traded for a step at either $\eta = 1$ or $\eta = -1$ and the interpolation functions still contain all of the lower polynomial degrees. Figures 4 and 5 each show a function and an approximation to the function using exponential interpolation with $h = \{0, 0, 700, 0\}$. The approximations in each figure use incomplete polynomials, which may work well in some cases, as shown in Figure 5 but they can lead to large approximation errors as shown in Figure 4.

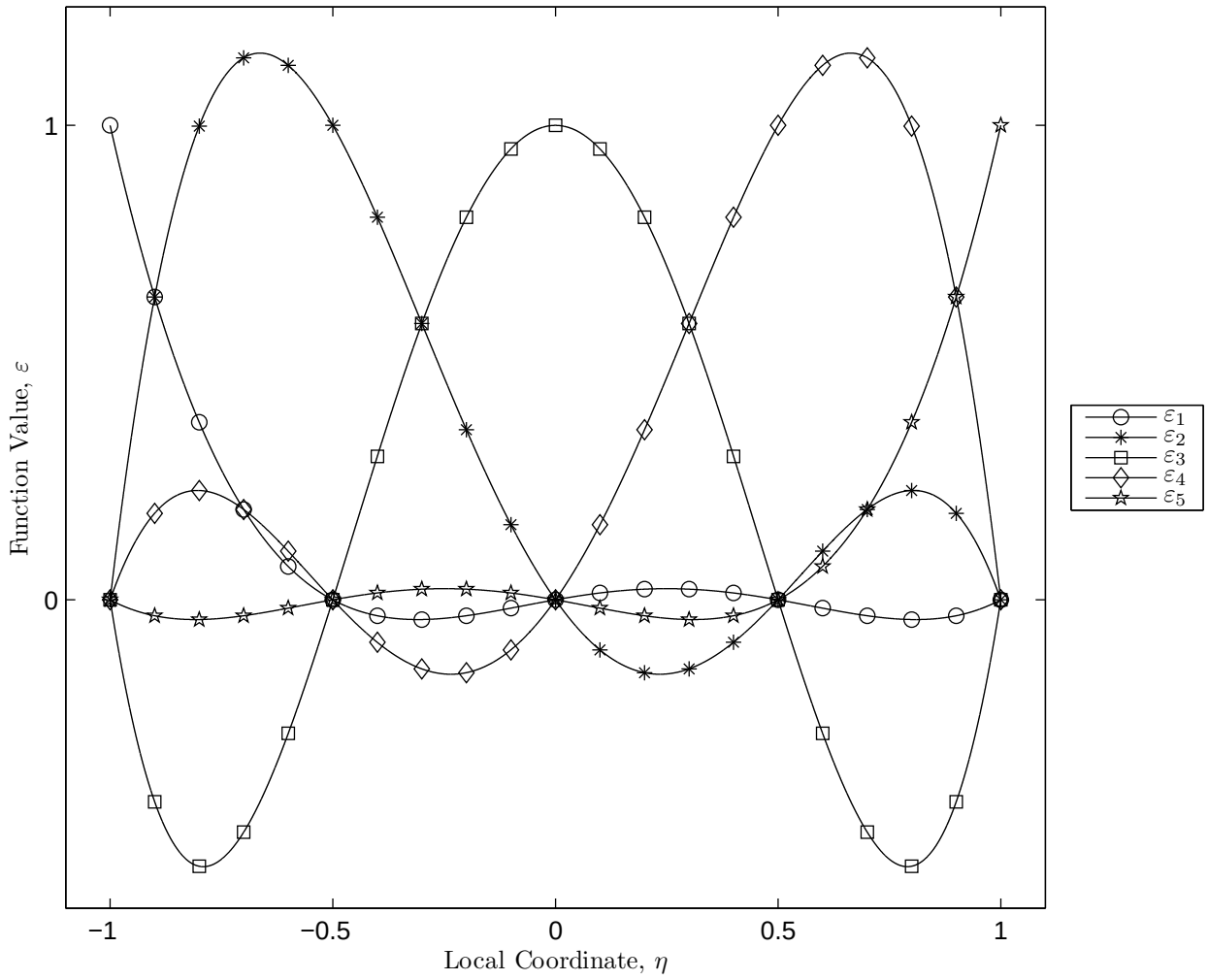

Figure 3. One-dimensional exponential interpolation functions for $h = \{0, 0, 0, 0\}$.

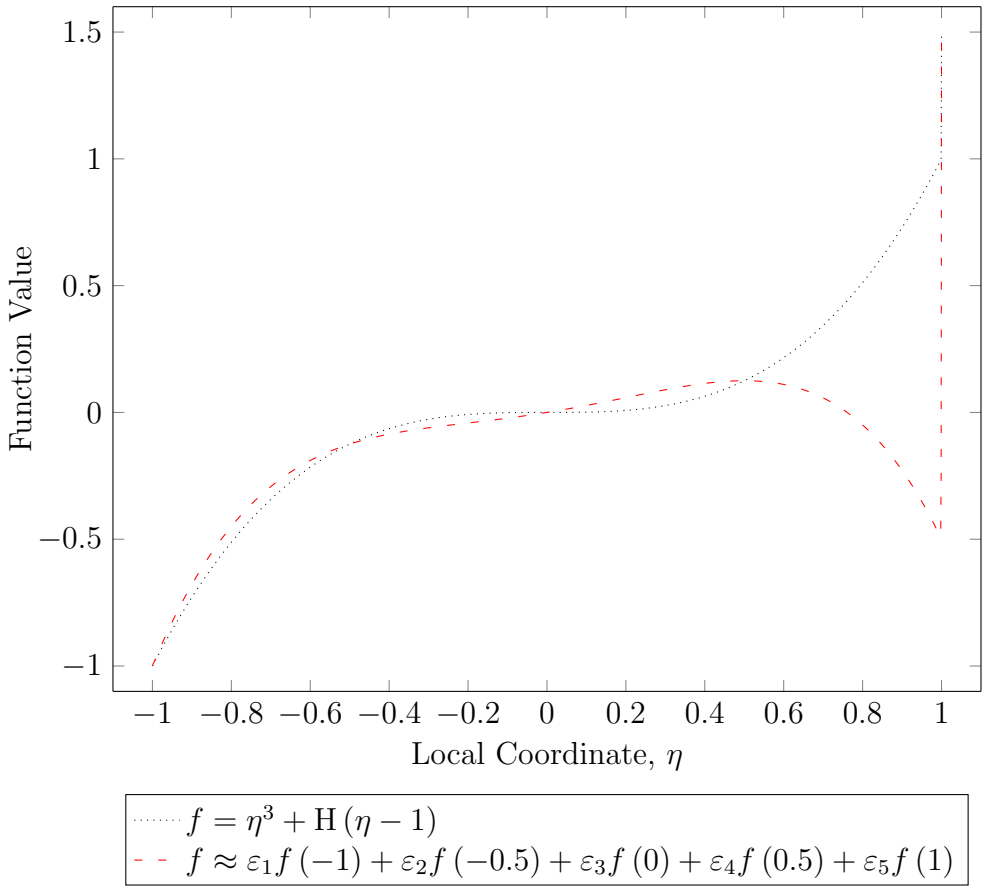

Figure 4. Example of a function and a poor approximation to the function using exponential interpolation and $h = \{0, 0, 700, 0\}$.

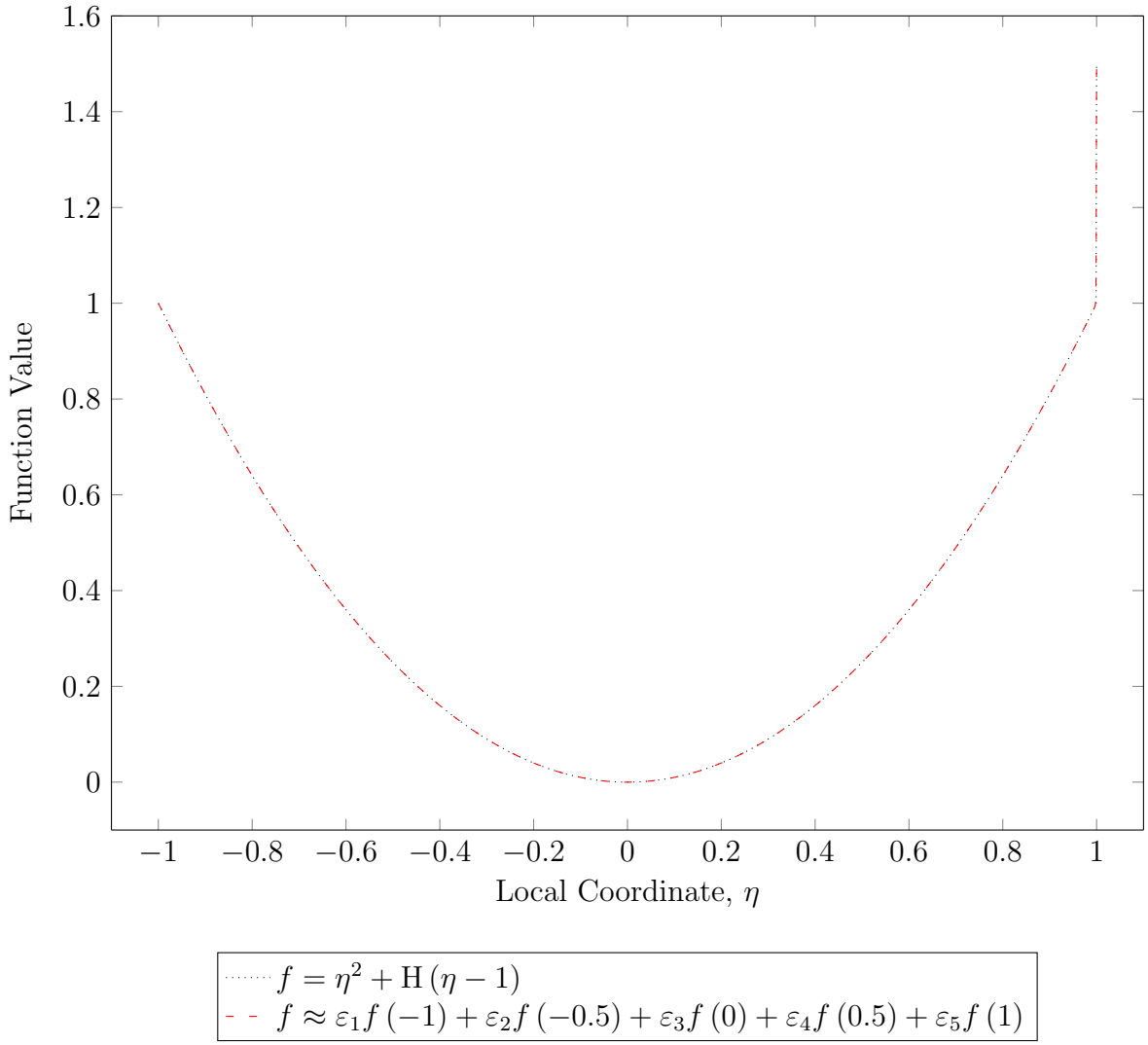

Figure 5. Example of a function and a good approximation to the function using exponential interpolation and $h = \{0, 0, 700, 0\}$.

2.3 Gauss-Legendre Quadrature

For the choice of exponential parameters described in the previous section, the resulting interpolation functions are the sum of a polynomial and a smooth approximation of a Heaviside function. The number of quadrature points required to integrate the exponential interpolation functions is driven by the term with a nonzero exponential parameter, which is the term that approximates the Heaviside function. A general expression for the term that approximates a step function is shown in Eq. (60). If i is even and $h \gg 0$, there is a step at $\eta = \pm 1$. If i is odd and $h \gg 0$, there is a step at $\eta = 1$. If i is odd and $h \ll 0$, there is a step at $\eta = -1$.

$$\mathrm{H} \approx \frac{e^{h_i \eta^i}}{e^{h_i}} \tag{60}$$

Gauss-Legendre quadrature is used for all of the numerical integration for the examples presented in Chapter 5. To determine the necessary number of quadrature points, h_i and i in Eq. (60) were varied and the resulting functions were integrated with increasing numbers of quadrature points until the integration error, Eq. (61), fell below 1×10^{-13}. Then least squares regression was used to fit a simple function to the data. For most choices of i, the exponential interpolation functions cannot be integrated analytically. The `quadgk` function in MATLAB was used to find a close approximation to the exact integral. The integral calculated using Gauss-Legendre quadrature is I_{GL} and the integral calculated using `quadgk` is I_0.

$$E = \frac{|I_{GL} - I_0|}{I_0} \tag{61}$$

Equation (62) is a least squares regression fit for the quadrature data. To ensure that enough quadrature points are used in every case, the slope of the regression equation was increased. The result is Eq. (63). Figure 6 is a plot of Eq. (62), Eq. (63), and the numerical integration data points. The one data point that lies above the lines in Figure 6 is an outlier. It corresponds to $|h_9| = 3000$. The integration error for this case is plotted in Figure 7. The first point where $E \leq 1 \times 10^{-13}$ is $N_{GL} = 756$, which is much greater than the number of quadrature points needed to obtain an accurate integral. The numbers of points predicted by Eqs. (62) and (63) are also plotted in Figure 7.

$$N_{GL} = 4.0478\sqrt{|h_i| i} \tag{62}$$

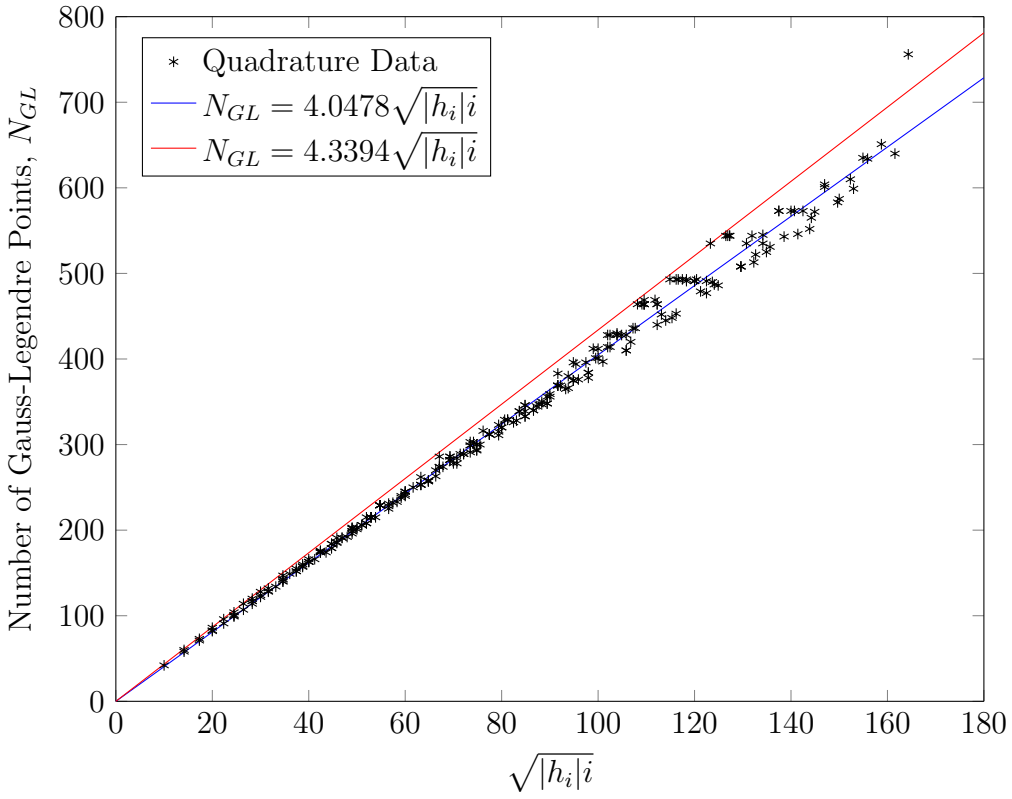

Figure 6. Number of Gauss-Legendre quadrature points needed to obtain $E \leq 1 \times 10^{-13}$.

$$N_{GL} = 4.3394\sqrt{|h_i|i} \tag{63}$$

In general terms, the larger $|h_i|$ and i become, the sharper the gradient at $\eta = \pm 1$ becomes and therefore more quadrature points are needed for integration. To integrate an exponential interpolation function, the number of quadrature points predicted by Eq. (63) is added to the number of points that would normally be used to integrate a polynomial. For example, 3 quadrature points are required to integrate a fourth degree polynomial using Gauss-Legendre quadrature. Numerical integration of Eq. (22) with $h = \{0, 0, 700, 0\}$ requires $3 + 199 = 202$ Gauss-Legendre quadrature points. The number of quadrature points needed to integrate the derivative of an exponential interpolation function is the same as the number needed for the original function, which is apparent if the derivative of Eq. (60) is calculated. No matter how many times the exponential interpolation functions are differentiated, $h_i\eta^i$ is always in the exponent. In the finite element equations that will follow, most of the matrix terms will involve products of four interpolation functions. Raising the Heaviside approximation to the fourth power, as shown in Eq. (64), has the

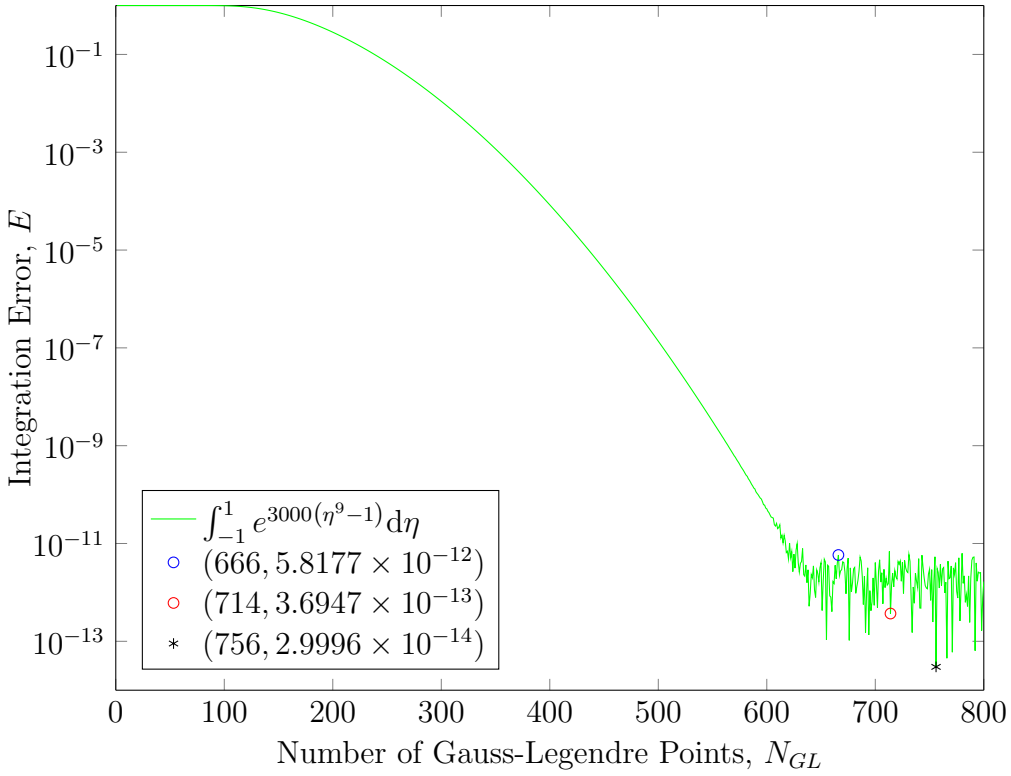

Figure 7. Integration error for $|h_9| = 3000$.

same effect as multiplying the exponential parameter by four.

$$\mathrm{H}^4 \approx \left(\frac{e^{h_i\eta^i}}{e^{h_i}}\right)^4 = \frac{e^{4h_i\eta^i}}{e^{4h_i}} \tag{64}$$

Products of the exponential interpolation functions require more quadrature points than a single exponential interpolation function. Stated another way, numerically integrating ε^4 defined using $h = \{0, 0, 100\}$ requires the same number of quadrature points as the numerical integral of ε defined using $h = \{0, 0, 400\}$.

2.4 Extension to Higher Dimensions

In the formulas that follow ξ and η are used to represent orthogonal coordinate directions in the local coordinates of a master element. Lagrange polynomials in one dimension are represented by ℓ_i and two-dimensional interpolation functions in the local coordinate system are represented by $\widehat{\psi}_i$. The typical method of defining interpolation functions in higher dimensions is to use products of one-dimensional interpolation functions. Equation (65) shows the formulation of interpolation functions for a two-dimensional element with three nodes along each coordinate direction and C^0 continuity.

$$\begin{bmatrix} \widehat{\psi}_1 & \widehat{\psi}_2 & \widehat{\psi}_3 \\ \widehat{\psi}_4 & \widehat{\psi}_5 & \widehat{\psi}_6 \\ \widehat{\psi}_7 & \widehat{\psi}_8 & \widehat{\psi}_9 \end{bmatrix} = \begin{Bmatrix} \ell_1(\eta) \\ \ell_2(\eta) \\ \ell_3(\eta) \end{Bmatrix} \begin{Bmatrix} \ell_1(\xi) & \ell_2(\xi) & \ell_3(\xi) \end{Bmatrix} \tag{65}$$

The one-dimensional exponential interpolation functions are incorporated into the formulation of two-dimensional functions by replacing ℓ_i with ε_i in one or both of the vectors on the right side of Eq. (65). A superscript is added to $\widehat{\psi}_i$ to distinguish the different types of interpolation functions. A superscript 0 is used for two-dimensional C^0 interpolation functions composed of one-dimensional Lagrange polynomials in each coordinate direction. The nine two-dimensional interpolation functions defined in Eq. (66) are plotted in Figure 8.

$$\begin{bmatrix} \widehat{\psi}_1^0 & \widehat{\psi}_2^0 & \widehat{\psi}_3^0 \\ \widehat{\psi}_4^0 & \widehat{\psi}_5^0 & \widehat{\psi}_6^0 \\ \widehat{\psi}_7^0 & \widehat{\psi}_8^0 & \widehat{\psi}_9^0 \end{bmatrix} = \begin{Bmatrix} \ell_1(\eta) \\ \ell_2(\eta) \\ \ell_3(\eta) \end{Bmatrix} \begin{Bmatrix} \ell_1(\xi) & \ell_2(\xi) & \ell_3(\xi) \end{Bmatrix} \tag{66}$$

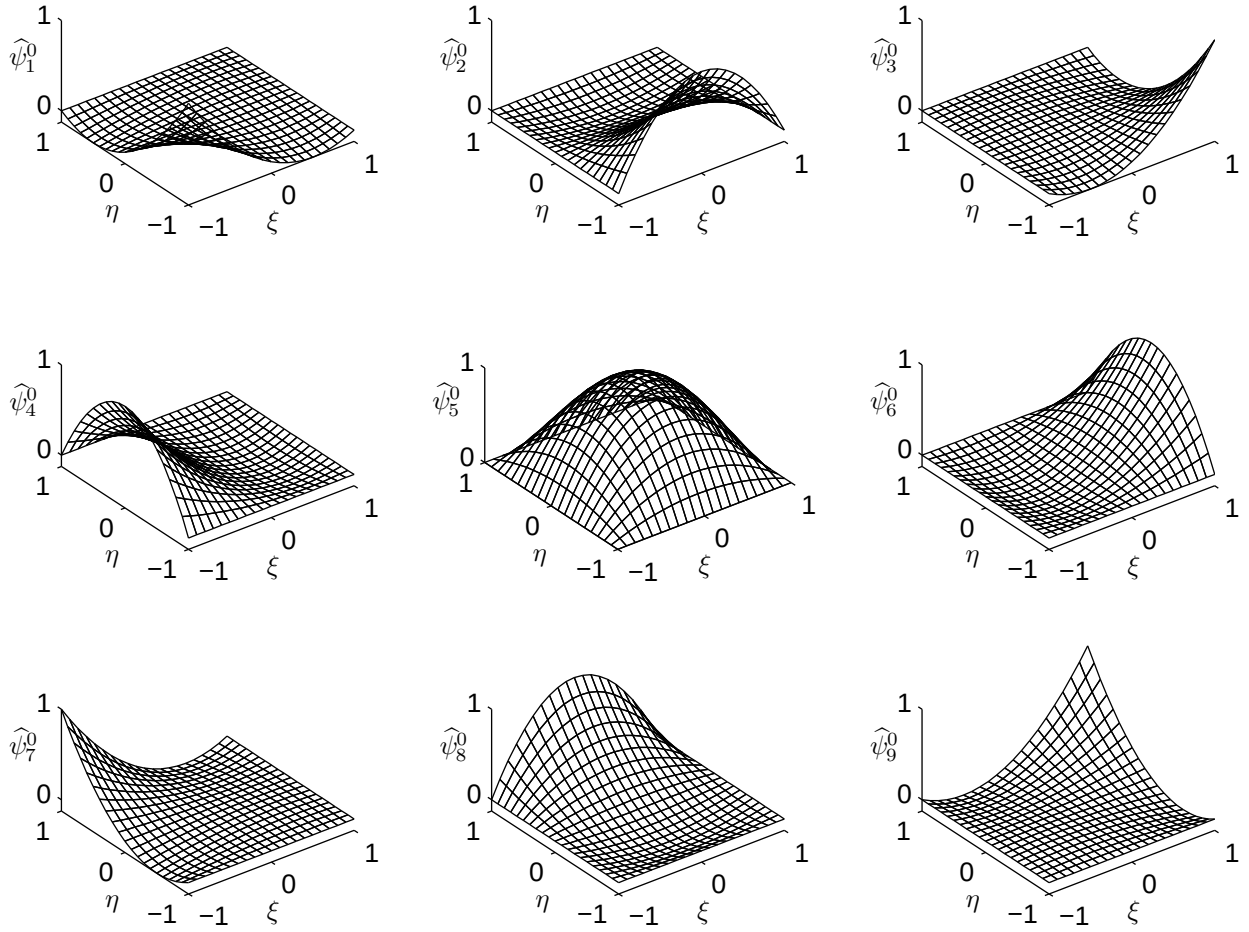

Figure 8. Interpolation functions defined in Eq. (66)**.**

A superscript 1 is used for two-dimensional C^0 interpolation functions composed of one-dimensional exponential interpolation functions in the ξ-direction and Lagrange polynomials in the η-direction. The nine two-dimensional interpolation functions defined in Eq. (67) are plotted in Figure 9.

$$\begin{bmatrix} \widehat{\psi}_1^1 & \widehat{\psi}_2^1 & \widehat{\psi}_3^1 \\ \widehat{\psi}_4^1 & \widehat{\psi}_5^1 & \widehat{\psi}_6^1 \\ \widehat{\psi}_7^1 & \widehat{\psi}_8^1 & \widehat{\psi}_9^1 \end{bmatrix} = \begin{Bmatrix} \ell_1(\eta) \\ \ell_2(\eta) \\ \ell_3(\eta) \end{Bmatrix} \begin{Bmatrix} \varepsilon_1(\xi) & \varepsilon_2(\xi) & \varepsilon_3(\xi) \end{Bmatrix} \tag{67}$$

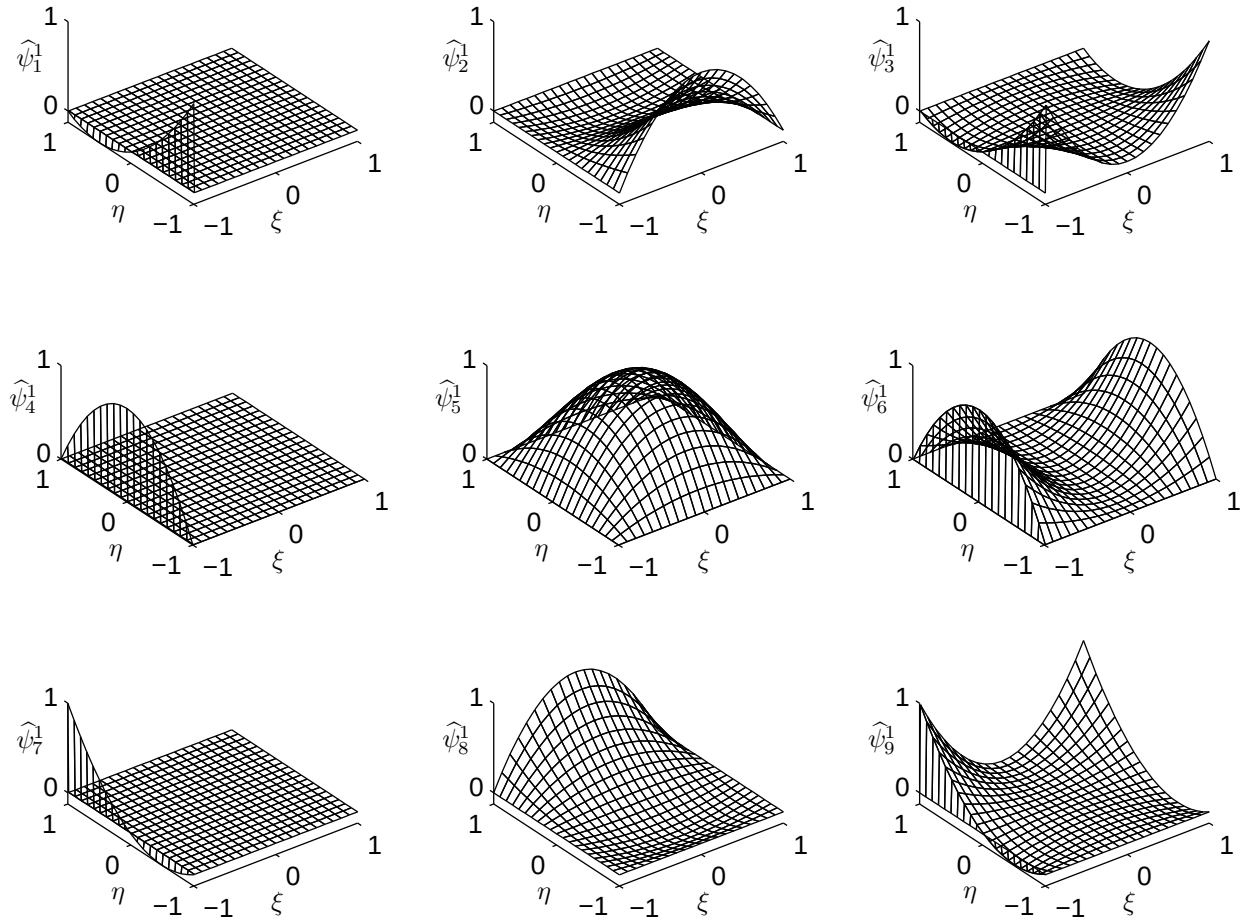

Figure 9. Interpolation functions defined in Eq. (67) **using the exponential parameters** $h = \{-700, 0\}$**.**

A superscript 2 is used for two-dimensional C^0 interpolation functions composed of one-dimensional Lagrange polynomials in the ξ-direction and exponential interpolation functions in the η-direction. The nine two-dimensional interpolation functions defined in Eq. (68) are plotted in Figure 10.

$$\begin{bmatrix} \widehat{\psi}_1^2 & \widehat{\psi}_2^2 & \widehat{\psi}_3^2 \\ \widehat{\psi}_4^2 & \widehat{\psi}_5^2 & \widehat{\psi}_6^2 \\ \widehat{\psi}_7^2 & \widehat{\psi}_8^2 & \widehat{\psi}_9^2 \end{bmatrix} = \begin{Bmatrix} \varepsilon_1(\eta) \\ \varepsilon_2(\eta) \\ \varepsilon_3(\eta) \end{Bmatrix} \begin{Bmatrix} \ell_1(\xi) & \ell_2(\xi) & \ell_3(\xi) \end{Bmatrix} \tag{68}$$

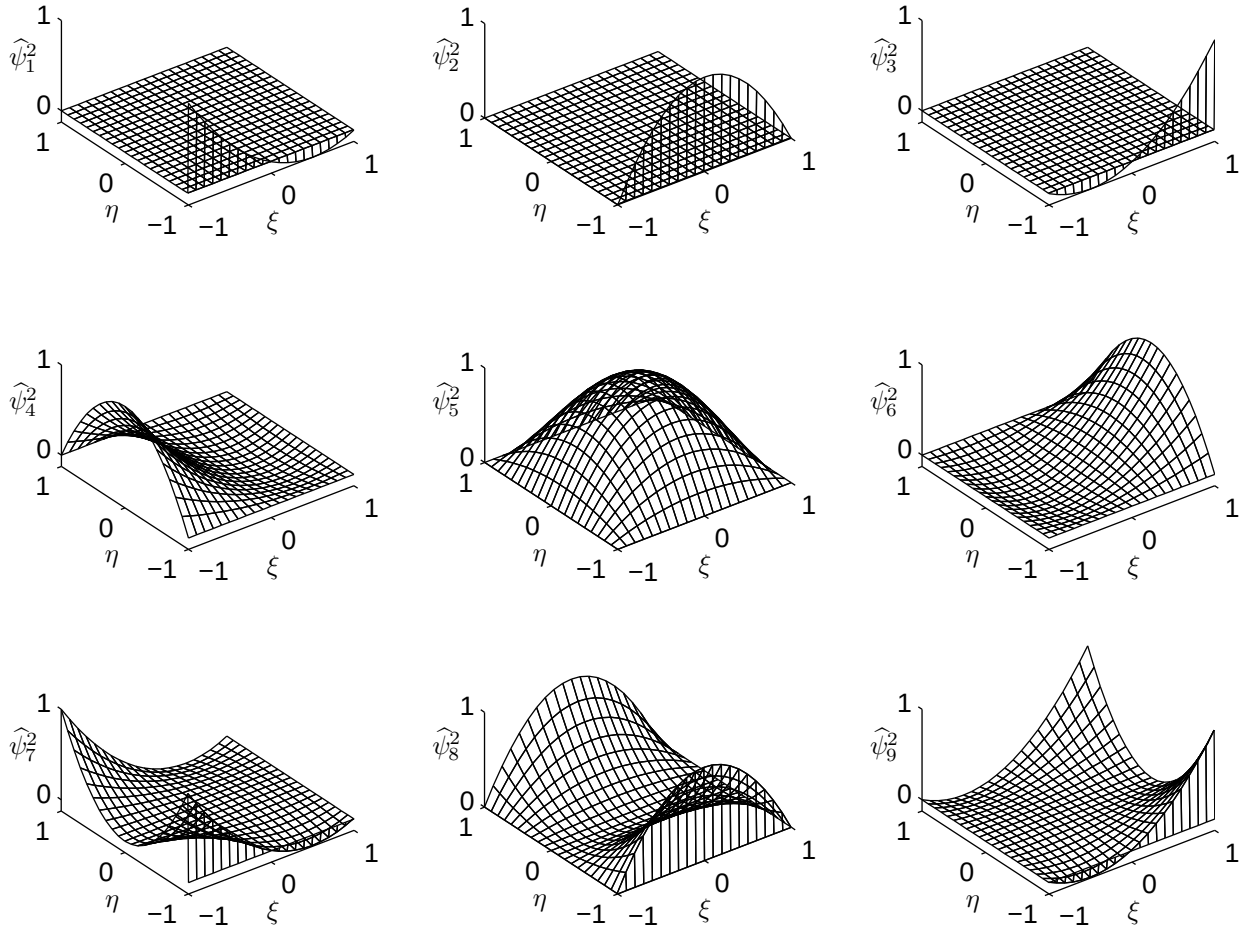

Figure 10. Interpolation functions defined in Eq. (68) **using the exponential parameters** $h = \{-700, 0\}$**.**

CHAPTER 3
GOVERNING DIFFERENTIAL EQUATIONS

General expressions for conservation of mass, momentum, and energy are converted to the form of the Euler equations used in Reference 4. For convenience, the equations are converted to dimensionless form. The fluid is assumed to be a perfect gas and body forces and heat transfer are assumed to be negligible. Viscosity is assumed to be zero.

3.1 Conservation of Mass

Equation (69) is a general form of conservation of mass applied to a stationary point in a fluid.

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho v_x) + \frac{\partial}{\partial y}(\rho v_y) + \frac{\partial}{\partial z}(\rho v_z) = 0 \tag{69}$$

The chain rule of differentiation is applied.

$$\frac{\partial \rho}{\partial t} + \rho\frac{\partial v_x}{\partial x} + v_x\frac{\partial \rho}{\partial x} + \rho\frac{\partial v_y}{\partial y} + v_y\frac{\partial \rho}{\partial y} + \rho\frac{\partial v_z}{\partial z} + v_z\frac{\partial \rho}{\partial z} = 0 \tag{70}$$

Free stream quantities are used to convert the properties of the flow to dimensionless form. A subscript ∞ indicates a free stream quantity and bold font indicates a dimensionless quantity.

$$\boldsymbol{v_x} = \frac{v_x}{v_\infty} \qquad \boldsymbol{v_y} = \frac{v_y}{v_\infty} \qquad \boldsymbol{v_z} = \frac{v_z}{v_\infty} \tag{71}$$

$$v_x = \boldsymbol{v_x} v_\infty \qquad v_y = \boldsymbol{v_y} v_\infty \qquad v_z = \boldsymbol{v_z} v_\infty \tag{72}$$

$$\boldsymbol{\rho} = \frac{\rho}{\rho_\infty} \tag{73}$$

$$\rho = \boldsymbol{\rho}\rho_\infty \tag{74}$$

To convert gradients to dimensionless form a constant reference length, L is used.

$$\boldsymbol{x} = \frac{x}{L} \qquad \boldsymbol{y} = \frac{y}{L} \qquad \boldsymbol{z} = \frac{z}{L} \tag{75}$$

$$x = \boldsymbol{x}L \qquad y = \boldsymbol{y}L \qquad z = \boldsymbol{z}L \tag{76}$$

$$\frac{\partial}{\partial \boldsymbol{x}} = \frac{\partial}{\partial \left(\frac{x}{L}\right)} \qquad \frac{\partial}{\partial \boldsymbol{y}} = \frac{\partial}{\partial \left(\frac{y}{L}\right)} \qquad \frac{\partial}{\partial \boldsymbol{z}} = \frac{\partial}{\partial \left(\frac{z}{L}\right)} \tag{77}$$

Since L is a constant, it can be moved outside of the partial derivatives in Eq. (77).

$$\frac{\partial}{\partial \boldsymbol{x}} = L\frac{\partial}{\partial x} \qquad \frac{\partial}{\partial \boldsymbol{y}} = L\frac{\partial}{\partial y} \qquad \frac{\partial}{\partial \boldsymbol{z}} = L\frac{\partial}{\partial z} \tag{78}$$

$$\frac{\partial}{\partial x} = \frac{1}{L}\frac{\partial}{\partial \boldsymbol{x}} \qquad \frac{\partial}{\partial y} = \frac{1}{L}\frac{\partial}{\partial \boldsymbol{y}} \qquad \frac{\partial}{\partial z} = \frac{1}{L}\frac{\partial}{\partial \boldsymbol{z}} \tag{79}$$

Time and derivatives with respect to time are converted to dimensionless form using the free stream velocity, v_∞ and the reference length, L.

$$\boldsymbol{t} = \frac{tv_\infty}{L} \tag{80}$$

$$t = \frac{\boldsymbol{t}L}{v_\infty} \tag{81}$$

$$\frac{\partial}{\partial \boldsymbol{t}} = \frac{\partial}{\partial \left(\frac{tv_\infty}{L}\right)} \tag{82}$$

Since v_∞ and L are constants, they can be moved outside of the partal derivative.

$$\frac{\partial}{\partial \boldsymbol{t}} = \frac{L}{v_\infty}\frac{\partial}{\partial t} \tag{83}$$

$$\frac{\partial}{\partial t} = \frac{v_\infty}{L}\frac{\partial}{\partial \boldsymbol{t}} \tag{84}$$

Equations (72), (74), (79), and (84) are substituted into Eq. (70) and the constant quantites are factored out of each term.

$$\frac{\rho_\infty v_\infty}{L}\left(\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{t}} + \boldsymbol{\rho}\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{x}} + \boldsymbol{v_x}\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{x}} + \boldsymbol{\rho}\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{y}} + \boldsymbol{v_y}\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{y}} + \boldsymbol{\rho}\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{z}} + \boldsymbol{v_z}\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{z}}\right) = 0 \tag{85}$$

The term outside of the parentheses in Eq. (85) is never zero unless the flow is static. Therefore the terms inside the parentheses must add to zero. Equation (86) is the dimensionless form of conservation of mass under the assumptions previously stated.

$$\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{t}} + \boldsymbol{\rho}\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{x}} + \boldsymbol{v_x}\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{x}} + \boldsymbol{\rho}\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{y}} + \boldsymbol{v_y}\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{y}} + \boldsymbol{\rho}\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{z}} + \boldsymbol{v_z}\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{z}} = 0 \tag{86}$$

3.2 Conservation of Momentum

Equation (87) is a general form of conservation of momentum applied to a stationary point in a fluid with no viscosity. Momentum is conserved in each coordinate direction. Cartesian coordinates are used here.

$$\begin{aligned}
x\text{-directon:} \quad & \frac{\partial}{\partial t}(\rho v_x) + \frac{\partial}{\partial x}(\rho v_x v_x) + \frac{\partial}{\partial y}(\rho v_x v_y) + \frac{\partial}{\partial z}(\rho v_x v_z) = -\frac{\partial P}{\partial x} + \rho f_x \\
y\text{-directon:} \quad & \frac{\partial}{\partial t}(\rho v_y) + \frac{\partial}{\partial x}(\rho v_y v_x) + \frac{\partial}{\partial y}(\rho v_y v_y) + \frac{\partial}{\partial z}(\rho v_y v_z) = -\frac{\partial P}{\partial y} + \rho f_y \\
z\text{-directon:} \quad & \frac{\partial}{\partial t}(\rho v_z) + \frac{\partial}{\partial x}(\rho v_z v_x) + \frac{\partial}{\partial y}(\rho v_z v_y) + \frac{\partial}{\partial z}(\rho v_z v_z) = -\frac{\partial P}{\partial z} + \rho f_z
\end{aligned} \tag{87}$$

Body forces are assumed to be negligible.

$$\begin{aligned}
x\text{-directon:} \quad & \frac{\partial}{\partial t}(\rho v_x) + \frac{\partial}{\partial x}(\rho v_x v_x) + \frac{\partial}{\partial y}(\rho v_x v_y) + \frac{\partial}{\partial z}(\rho v_x v_z) = -\frac{\partial P}{\partial x} \\
y\text{-directon:} \quad & \frac{\partial}{\partial t}(\rho v_y) + \frac{\partial}{\partial x}(\rho v_y v_x) + \frac{\partial}{\partial y}(\rho v_y v_y) + \frac{\partial}{\partial z}(\rho v_y v_z) = -\frac{\partial P}{\partial y} \\
z\text{-directon:} \quad & \frac{\partial}{\partial t}(\rho v_z) + \frac{\partial}{\partial x}(\rho v_z v_x) + \frac{\partial}{\partial y}(\rho v_z v_y) + \frac{\partial}{\partial z}(\rho v_z v_z) = -\frac{\partial P}{\partial z}
\end{aligned} \tag{88}$$

The chain rule of differentiation is applied to the terms to the left of the equal sign.

x-directon:

$$\rho\frac{\partial v_x}{\partial t}+v_x\frac{\partial \rho}{\partial t}+\rho v_x\frac{\partial v_x}{\partial x}+\rho v_x\frac{\partial v_x}{\partial x}+v_xv_x\frac{\partial \rho}{\partial x}+\rho v_x\frac{\partial v_y}{\partial y}+\rho v_y\frac{\partial v_x}{\partial y}+v_xv_y\frac{\partial \rho}{\partial y}+\rho v_x\frac{\partial v_z}{\partial z}+\rho v_z\frac{\partial v_x}{\partial z}+v_xv_z\frac{\partial \rho}{\partial z}=-\frac{\partial P}{\partial x}$$

y-directon:

$$\rho\frac{\partial v_y}{\partial t}+v_y\frac{\partial \rho}{\partial t}+\rho v_y\frac{\partial v_x}{\partial x}+\rho v_x\frac{\partial v_y}{\partial x}+v_yv_x\frac{\partial \rho}{\partial x}+\rho v_y\frac{\partial v_y}{\partial y}+\rho v_y\frac{\partial v_y}{\partial y}+v_yv_y\frac{\partial \rho}{\partial y}+\rho v_y\frac{\partial v_z}{\partial z}+\rho v_z\frac{\partial v_y}{\partial z}+v_yv_z\frac{\partial \rho}{\partial z}=-\frac{\partial P}{\partial y} \tag{89}$$

z-directon:

$$\rho\frac{\partial v_z}{\partial t}+v_z\frac{\partial \rho}{\partial t}+\rho v_z\frac{\partial v_x}{\partial x}+\rho v_x\frac{\partial v_z}{\partial x}+v_zv_x\frac{\partial \rho}{\partial x}+\rho v_z\frac{\partial v_y}{\partial y}+\rho v_y\frac{\partial v_z}{\partial y}+v_zv_y\frac{\partial \rho}{\partial y}+\rho v_z\frac{\partial v_z}{\partial z}+\rho v_z\frac{\partial v_z}{\partial z}+v_zv_z\frac{\partial \rho}{\partial z}=-\frac{\partial P}{\partial z}$$

The terms to the left of the equal sign are rearranged.

$$\begin{aligned}
x\text{-directon:}\quad & v_x\left[\frac{\partial \rho}{\partial t}+v_x\frac{\partial \rho}{\partial x}+\rho\frac{\partial v_x}{\partial x}+v_y\frac{\partial \rho}{\partial y}+\rho\frac{\partial v_y}{\partial y}+v_z\frac{\partial \rho}{\partial z}+\rho\frac{\partial v_z}{\partial z}\right]+\rho\frac{\partial v_x}{\partial t}+\rho v_x\frac{\partial v_x}{\partial x}+\rho v_y\frac{\partial v_x}{\partial y}+\rho v_z\frac{\partial v_x}{\partial z}=-\frac{\partial P}{\partial x}\\
y\text{-directon:}\quad & v_y\left[\frac{\partial \rho}{\partial t}+v_x\frac{\partial \rho}{\partial x}+\rho\frac{\partial v_x}{\partial x}+v_y\frac{\partial \rho}{\partial y}+\rho\frac{\partial v_y}{\partial y}+v_z\frac{\partial \rho}{\partial z}+\rho\frac{\partial v_z}{\partial z}\right]+\rho\frac{\partial v_y}{\partial t}+\rho v_x\frac{\partial v_y}{\partial x}+\rho v_y\frac{\partial v_y}{\partial y}+\rho v_z\frac{\partial v_y}{\partial z}=-\frac{\partial P}{\partial y}\\
z\text{-directon:}\quad & v_z\left[\frac{\partial \rho}{\partial t}+v_x\frac{\partial \rho}{\partial x}+\rho\frac{\partial v_x}{\partial x}+v_y\frac{\partial \rho}{\partial y}+\rho\frac{\partial v_y}{\partial y}+v_z\frac{\partial \rho}{\partial z}+\rho\frac{\partial v_z}{\partial z}\right]+\rho\frac{\partial v_z}{\partial t}+\rho v_x\frac{\partial v_z}{\partial x}+\rho v_y\frac{\partial v_z}{\partial y}+\rho v_z\frac{\partial v_z}{\partial z}=-\frac{\partial P}{\partial z}
\end{aligned} \tag{90}$$

Due to the conservation mass, Eq. (70), the terms in brackets add to zero. The remaining terms are rearranged.

$$\begin{aligned}
x\text{-directon:} \quad & \frac{\partial v_x}{\partial t} + v_x\frac{\partial v_x}{\partial x} + v_y\frac{\partial v_x}{\partial y} + v_z\frac{\partial v_x}{\partial z} + \frac{1}{\rho}\frac{\partial P}{\partial x} = 0 \\
y\text{-directon:} \quad & \frac{\partial v_y}{\partial t} + v_x\frac{\partial v_y}{\partial x} + v_y\frac{\partial v_y}{\partial y} + v_z\frac{\partial v_y}{\partial z} + \frac{1}{\rho}\frac{\partial P}{\partial y} = 0 \\
z\text{-directon:} \quad & \frac{\partial v_z}{\partial t} + v_x\frac{\partial v_z}{\partial x} + v_y\frac{\partial v_z}{\partial y} + v_z\frac{\partial v_z}{\partial z} + \frac{1}{\rho}\frac{\partial P}{\partial z} = 0
\end{aligned} \tag{91}$$

Free stream quantities are used to convert the properties of the flow to dimensionless form.

$$\boldsymbol{P} = \frac{P}{\rho_\infty v_\infty^2} \tag{92}$$

$$P = \boldsymbol{P}\rho_\infty v_\infty^2 \tag{93}$$

Equations (72), (74), (79), (84), and (93) are substituted into Eq. (91) and the constant quantites are factored out of each term.

$$\begin{aligned}
x\text{-directon:} \quad & \frac{v_\infty^2}{L}\left[\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{t}} + \boldsymbol{v_x}\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{x}} + \boldsymbol{v_y}\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{y}} + \boldsymbol{v_z}\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{z}} + \frac{1}{\boldsymbol{\rho}}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{x}}\right] = 0 \\
y\text{-directon:} \quad & \frac{v_\infty^2}{L}\left[\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{t}} + \boldsymbol{v_x}\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{x}} + \boldsymbol{v_y}\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{y}} + \boldsymbol{v_z}\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{z}} + \frac{1}{\boldsymbol{\rho}}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{y}}\right] = 0 \\
z\text{-directon:} \quad & \frac{v_\infty^2}{L}\left[\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{t}} + \boldsymbol{v_x}\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{x}} + \boldsymbol{v_y}\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{y}} + \boldsymbol{v_z}\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{z}} + \frac{1}{\boldsymbol{\rho}}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{z}}\right] = 0
\end{aligned} \tag{94}$$

The terms outside of the brackets in Eq. (94) are never zero unless the flow is static. Therefore, the terms inside the brackets must add to zero. Equation (95) is the dimensionless form of conservation of momenutm under the assumptions previously stated.

$$\begin{aligned}
x\text{-directon:} \quad & \frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{t}} + \boldsymbol{v_x}\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{x}} + \boldsymbol{v_y}\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{y}} + \boldsymbol{v_z}\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{z}} + \frac{1}{\boldsymbol{\rho}}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{x}} = 0 \\
y\text{-directon:} \quad & \frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{t}} + \boldsymbol{v_x}\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{x}} + \boldsymbol{v_y}\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{y}} + \boldsymbol{v_z}\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{z}} + \frac{1}{\boldsymbol{\rho}}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{y}} = 0 \\
z\text{-directon:} \quad & \frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{t}} + \boldsymbol{v_x}\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{x}} + \boldsymbol{v_y}\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{y}} + \boldsymbol{v_z}\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{z}} + \frac{1}{\boldsymbol{\rho}}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{z}} = 0
\end{aligned} \tag{95}$$

3.3 Conservation of Energy

Equation (96) is a general form of conservation of energy applied to a stationary point in a fluid with no viscosity.

$$\begin{aligned}\frac{\partial}{\partial t}\left[\rho\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]+\frac{\partial}{\partial x}\left[\rho v_x\left(\bar{h}+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]+\\ \frac{\partial}{\partial y}\left[\rho v_y\left(\bar{h}+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]+\frac{\partial}{\partial z}\left[\rho v_z\left(\bar{h}+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]\\ =-\frac{\partial q_x}{\partial x}-\frac{\partial q_y}{\partial y}-\frac{\partial q_z}{\partial z}\end{aligned} \tag{96}$$

It is assumed that there is no heat transfer between the fluid and its surroundings.

$$\begin{aligned}\frac{\partial}{\partial t}\left[\rho\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]+\frac{\partial}{\partial x}\left[\rho v_x\left(\bar{h}+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]+\\ \frac{\partial}{\partial y}\left[\rho v_y\left(\bar{h}+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]+\frac{\partial}{\partial z}\left[\rho v_z\left(\bar{h}+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]=0\end{aligned} \tag{97}$$

The equation for specific enthalpy, Eq. (98), is substituted into Eq. (97).

$$\bar{h}=e+\frac{P}{\rho} \tag{98}$$

$$\begin{aligned}&\frac{\partial}{\partial t}\left[\rho\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]+\\ &\frac{\partial}{\partial x}\left[\rho v_x\left(e+\frac{P}{\rho}+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]+\\ &\frac{\partial}{\partial y}\left[\rho v_y\left(e+\frac{P}{\rho}+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]+\\ &\frac{\partial}{\partial z}\left[\rho v_z\left(e+\frac{P}{\rho}+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]=0\end{aligned} \tag{99}$$

The terms inside the brackets are rearranged.

$$\begin{aligned}&\frac{\partial}{\partial t}\left[\rho\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\right]\\ &+\frac{\partial}{\partial x}\left[\rho v_x\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+Pv_x\right]\\ &+\frac{\partial}{\partial y}\left[\rho v_y\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+Pv_y\right]\\ &+\frac{\partial}{\partial z}\left[\rho v_z\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+Pv_z\right]=0\end{aligned} \tag{100}$$

The chain rule of differentiation is applied.

$$\begin{aligned}
&\frac{\partial \rho}{\partial t}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\rho\frac{\partial}{\partial t}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\\
&\rho v_x\frac{\partial}{\partial x}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\frac{\partial}{\partial x}\left(\rho v_x\right)\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\\
&\rho v_y\frac{\partial}{\partial y}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\frac{\partial}{\partial y}\left(\rho v_y\right)\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\\
&\rho v_z\frac{\partial}{\partial z}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\frac{\partial}{\partial z}\left(\rho v_z\right)\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\\
&\frac{\partial}{\partial x}\left(Pv_x\right)+\frac{\partial}{\partial y}\left(Pv_y\right)+\frac{\partial}{\partial z}\left(Pv_z\right)=0
\end{aligned} \tag{101}$$

The equation is rearranged.

$$\begin{aligned}
&\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)\left[\frac{\partial \rho}{\partial t}+\frac{\partial}{\partial x}\left(\rho v_x\right)+\frac{\partial}{\partial y}\left(\rho v_y\right)+\frac{\partial}{\partial z}\left(\rho v_z\right)\right]+\\
&\rho\frac{\partial}{\partial t}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\\
&\rho v_x\frac{\partial}{\partial x}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\frac{\partial}{\partial x}\left(Pv_x\right)+\\
&\rho v_y\frac{\partial}{\partial y}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\frac{\partial}{\partial y}\left(Pv_y\right)+\\
&\rho v_z\frac{\partial}{\partial z}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\frac{\partial}{\partial z}\left(Pv_z\right)=0
\end{aligned} \tag{102}$$

Due to conservation of mass, the terms in brackets add to zero.

$$\begin{aligned}
&\rho\frac{\partial}{\partial t}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\\
&\rho v_x\frac{\partial}{\partial x}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\frac{\partial}{\partial x}\left(Pv_x\right)+\\
&\rho v_y\frac{\partial}{\partial y}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\frac{\partial}{\partial y}\left(Pv_y\right)+\\
&\rho v_z\frac{\partial}{\partial z}\left(e+\frac{1}{2}v_x^2+\frac{1}{2}v_y^2+\frac{1}{2}v_z^2+\phi\right)+\frac{\partial}{\partial z}\left(Pv_z\right)=0
\end{aligned} \tag{103}$$

The partial derivatives are distributed to the terms in parentheses.

$$\begin{aligned}
&\rho\left(\frac{\partial e}{\partial t}+v_x\frac{\partial v_x}{\partial t}+v_y\frac{\partial v_y}{\partial t}+v_z\frac{\partial v_z}{\partial t}+\frac{\partial \phi}{\partial t}\right)+\\
&\rho v_x\left(\frac{\partial e}{\partial x}+v_x\frac{\partial v_x}{\partial x}+v_y\frac{\partial v_y}{\partial x}+v_z\frac{\partial v_z}{\partial x}+\frac{\partial \phi}{\partial x}\right)+\frac{\partial}{\partial x}\left(Pv_x\right)+\\
&\rho v_y\left(\frac{\partial e}{\partial y}+v_x\frac{\partial v_x}{\partial y}+v_y\frac{\partial v_y}{\partial y}+v_z\frac{\partial v_z}{\partial y}+\frac{\partial \phi}{\partial y}\right)+\frac{\partial}{\partial y}\left(Pv_y\right)+\\
&\rho v_z\left(\frac{\partial e}{\partial z}+v_x\frac{\partial v_x}{\partial z}+v_y\frac{\partial v_y}{\partial z}+v_z\frac{\partial v_z}{\partial z}+\frac{\partial \phi}{\partial z}\right)+\frac{\partial}{\partial z}\left(Pv_z\right)=0
\end{aligned} \tag{104}$$

The variable ϕ represents a potential field. The negative gradient of a potential field $(-\nabla\phi)$ is a body force.

$$\frac{\partial \phi}{\partial t}=-f_t \qquad \frac{\partial \phi}{\partial x}=-f_x \qquad \frac{\partial \phi}{\partial y}=-f_y \qquad \frac{\partial \phi}{\partial z}=-f_z \tag{105}$$

Body forces are assumed to be negligible.

$$\begin{aligned}
&\rho\left(\frac{\partial e}{\partial t}+v_x\frac{\partial v_x}{\partial t}+v_y\frac{\partial v_y}{\partial t}+v_z\frac{\partial v_z}{\partial t}\right)+\\
&\rho v_x\left(\frac{\partial e}{\partial x}+v_x\frac{\partial v_x}{\partial x}+v_y\frac{\partial v_y}{\partial x}+v_z\frac{\partial v_z}{\partial x}\right)+\frac{\partial}{\partial x}\left(Pv_x\right)+\\
&\rho v_y\left(\frac{\partial e}{\partial y}+v_x\frac{\partial v_x}{\partial y}+v_y\frac{\partial v_y}{\partial y}+v_z\frac{\partial v_z}{\partial y}\right)+\frac{\partial}{\partial y}\left(Pv_y\right)+\\
&\rho v_z\left(\frac{\partial e}{\partial z}+v_x\frac{\partial v_x}{\partial z}+v_y\frac{\partial v_y}{\partial z}+v_z\frac{\partial v_z}{\partial z}\right)+\frac{\partial}{\partial z}\left(Pv_z\right)=0
\end{aligned} \tag{106}$$

The products of density and velocity are distributed to the terms in parentheses and the chain rule of differentiation is applied to the products of pressure and velocity.

$$\begin{aligned}
&\rho\frac{\partial e}{\partial t}+\rho v_x\frac{\partial v_x}{\partial t}+\rho v_y\frac{\partial v_y}{\partial t}+\rho v_z\frac{\partial v_z}{\partial t}+\\
&\rho v_x\frac{\partial e}{\partial x}+\rho v_x v_x\frac{\partial v_x}{\partial x}+\rho v_x v_y\frac{\partial v_y}{\partial x}+\rho v_x v_z\frac{\partial v_z}{\partial x}+v_x\frac{\partial P}{\partial x}+P\frac{\partial v_x}{\partial x}+\\
&\rho v_y\frac{\partial e}{\partial y}+\rho v_y v_x\frac{\partial v_x}{\partial y}+\rho v_y v_y\frac{\partial v_y}{\partial y}+\rho v_y v_z\frac{\partial v_z}{\partial y}+v_y\frac{\partial P}{\partial y}+P\frac{\partial v_y}{\partial y}+\\
&\rho v_z\frac{\partial e}{\partial z}+\rho v_z v_x\frac{\partial v_x}{\partial z}+\rho v_z v_y\frac{\partial v_y}{\partial z}+\rho v_z v_z\frac{\partial v_z}{\partial z}+v_z\frac{\partial P}{\partial z}+P\frac{\partial v_z}{\partial z}=0
\end{aligned} \tag{107}$$

The terms are rearranged.

$$\begin{aligned}
&\rho\frac{\partial e}{\partial t}+\rho v_x\frac{\partial e}{\partial x}+\rho v_y\frac{\partial e}{\partial y}+\rho v_z\frac{\partial e}{\partial z}+P\frac{\partial v_x}{\partial x}+P\frac{\partial v_y}{\partial y}+P\frac{\partial v_z}{\partial z}+\\
&\rho v_x\left(\frac{\partial v_x}{\partial t}+v_x\frac{\partial v_x}{\partial x}+v_y\frac{\partial v_x}{\partial y}+v_z\frac{\partial v_x}{\partial z}+\frac{1}{\rho}\frac{\partial P}{\partial x}\right)+\\
&\rho v_y\left(\frac{\partial v_y}{\partial t}+v_x\frac{\partial v_y}{\partial x}+v_y\frac{\partial v_y}{\partial y}+v_z\frac{\partial v_y}{\partial z}+\frac{1}{\rho}\frac{\partial P}{\partial y}\right)+\\
&\rho v_z\left(\frac{\partial v_z}{\partial t}+v_x\frac{\partial v_z}{\partial x}+v_y\frac{\partial v_z}{\partial y}+v_z\frac{\partial v_z}{\partial z}+\frac{1}{\rho}\frac{\partial P}{\partial z}\right)=0
\end{aligned} \tag{108}$$

Due to conservation of momentum for a flow with no viscosity and negligible body force acting on it, the terms in parentheses add to zero.

$$\rho\frac{\partial e}{\partial t}+\rho v_x\frac{\partial e}{\partial x}+\rho v_y\frac{\partial e}{\partial y}+\rho v_z\frac{\partial e}{\partial z}+P\frac{\partial v_x}{\partial x}+P\frac{\partial v_y}{\partial y}+P\frac{\partial v_z}{\partial z}=0 \tag{109}$$

For a calorically perfect gas, $e=C_vT$, C_v is constant, and $C_v=\mathcal{R}/\left(\gamma-1\right)$.

$$\begin{aligned}
&\rho\frac{\mathcal{R}}{\gamma-1}\frac{\partial T}{\partial t}+\rho v_x\frac{\mathcal{R}}{\gamma-1}\frac{\partial T}{\partial x}+\rho v_y\frac{\mathcal{R}}{\gamma-1}\frac{\partial T}{\partial y}+\rho v_z\frac{\mathcal{R}}{\gamma-1}\frac{\partial T}{\partial z}+\\
&\qquad P\frac{\partial v_x}{\partial x}+P\frac{\partial v_y}{\partial y}+P\frac{\partial v_z}{\partial z}=0
\end{aligned} \tag{110}$$

For a perfect gas, $T=P/\left(\rho\mathcal{R}\right)$.

$$\begin{aligned}
&\rho\frac{1}{\gamma-1}\frac{\partial}{\partial t}\left(\frac{P}{\rho}\right)+\rho v_x\frac{1}{\gamma-1}\frac{\partial}{\partial x}\left(\frac{P}{\rho}\right)+\rho v_y\frac{1}{\gamma-1}\frac{\partial}{\partial y}\left(\frac{P}{\rho}\right)+\rho v_z\frac{1}{\gamma-1}\frac{\partial}{\partial z}\left(\frac{P}{\rho}\right)+\\
&\qquad P\frac{\partial v_x}{\partial x}+P\frac{\partial v_y}{\partial y}+P\frac{\partial v_z}{\partial z}=0
\end{aligned} \tag{111}$$

$$\begin{aligned}
&\rho\frac{1}{\gamma-1}\left(\frac{1}{\rho}\frac{\partial P}{\partial t}-\frac{P}{\rho^2}\frac{\partial \rho}{\partial t}\right)+\rho v_x\frac{1}{\gamma-1}\left(\frac{1}{\rho}\frac{\partial P}{\partial x}-\frac{P}{\rho^2}\frac{\partial \rho}{\partial x}\right)+\\
&\rho v_y\frac{1}{\gamma-1}\left(\frac{1}{\rho}\frac{\partial P}{\partial y}-\frac{P}{\rho^2}\frac{\partial \rho}{\partial y}\right)+\rho v_z\frac{1}{\gamma-1}\left(\frac{1}{\rho}\frac{\partial P}{\partial z}-\frac{P}{\rho^2}\frac{\partial \rho}{\partial z}\right)+\\
&\qquad P\frac{\partial v_x}{\partial x}+P\frac{\partial v_y}{\partial y}+P\frac{\partial v_z}{\partial z}=0
\end{aligned} \tag{112}$$

Density is canceled in the numerators and denominators and the entire equation is multiplied by $(\gamma-1)$.

$$\left(\frac{\partial P}{\partial t}-\frac{P}{\rho}\frac{\partial \rho}{\partial t}\right)+v_x\left(\frac{\partial P}{\partial x}-\frac{P}{\rho}\frac{\partial \rho}{\partial x}\right)+v_y\left(\frac{\partial P}{\partial y}-\frac{P}{\rho}\frac{\partial \rho}{\partial y}\right)+v_z\left(\frac{\partial P}{\partial z}-\frac{P}{\rho}\frac{\partial \rho}{\partial z}\right)+ (\gamma-1)\,P\frac{\partial v_x}{\partial x}+(\gamma-1)\,P\frac{\partial v_y}{\partial y}+(\gamma-1)\,P\frac{\partial v_z}{\partial z}=0 \tag{113}$$

The equation is expanded and all terms are multiplied by density.

$$\rho\frac{\partial P}{\partial t}-P\frac{\partial \rho}{\partial t}+\rho v_x\frac{\partial P}{\partial x}-v_xP\frac{\partial \rho}{\partial x}+\rho v_y\frac{\partial P}{\partial y}-v_yP\frac{\partial \rho}{\partial y}+\rho v_z\frac{\partial P}{\partial z}-v_zP\frac{\partial \rho}{\partial z}+ \gamma\rho P\frac{\partial v_x}{\partial x}-\rho P\frac{\partial v_x}{\partial x}+\gamma\rho P\frac{\partial v_y}{\partial y}-\rho P\frac{\partial v_y}{\partial y}+\gamma\rho P\frac{\partial v_z}{\partial z}-\rho P\frac{\partial v_z}{\partial z}=0 \tag{114}$$

The equation is rearranged.

$$\rho\frac{\partial P}{\partial t}+\rho v_x\frac{\partial P}{\partial x}+\rho v_y\frac{\partial P}{\partial y}+\rho v_z\frac{\partial P}{\partial z}+\gamma\rho P\frac{\partial v_x}{\partial x}+\gamma\rho P\frac{\partial v_y}{\partial y}+\gamma\rho P\frac{\partial v_z}{\partial z}- P\left(\frac{\partial \rho}{\partial t}+\rho\frac{\partial v_x}{\partial x}+v_x\frac{\partial \rho}{\partial x}+\rho\frac{\partial v_y}{\partial y}+v_y\frac{\partial \rho}{\partial y}+\rho\frac{\partial v_z}{\partial z}+v_z\frac{\partial \rho}{\partial z}\right)=0 \tag{115}$$

Due to conservation of mass, the terms in parentheses add to zero. Density is factored out of the remaining terms.

$$\rho\left(\frac{\partial P}{\partial t}+v_x\frac{\partial P}{\partial x}+v_y\frac{\partial P}{\partial y}+v_z\frac{\partial P}{\partial z}+\gamma P\frac{\partial v_x}{\partial x}+\gamma P\frac{\partial v_y}{\partial y}+\gamma P\frac{\partial v_z}{\partial z}\right)=0 \tag{116}$$

The density of a fluid is always a positive number, therefore the terms in parentheses must add to zero.

$$v_x\frac{\partial P}{\partial x}+v_y\frac{\partial P}{\partial y}+v_z\frac{\partial P}{\partial z}+\gamma P\frac{\partial v_x}{\partial x}+\gamma P\frac{\partial v_y}{\partial y}+\gamma P\frac{\partial v_z}{\partial z}=0 \tag{117}$$

Equation (117) represents conservation of energy for an adiabatic flow of a perfect gas with no viscosity and no body force acting on it. It is converted to dimensionless form by substituting Eqs. (72), (74), (79), (84), and (93) and factoring out constant terms.

$$\frac{\rho_\infty v_\infty^3}{L}\left(\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{t}}+\boldsymbol{v_x}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{x}}+\boldsymbol{v_y}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{y}}+\boldsymbol{v_z}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{z}}+\gamma\boldsymbol{P}\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{x}}+\gamma\boldsymbol{P}\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{y}}+\gamma\boldsymbol{P}\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{z}}\right)=0 \tag{118}$$

The terms outside the parentheses in Eq. (118) are never zero unless the flow is static. The terms inside the parentheses must add to zero. Equation (119) is the dimensionless form of Eq. (117).

$$\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{t}} + \boldsymbol{v_x}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{x}} + \boldsymbol{v_y}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{y}} + \boldsymbol{v_z}\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{z}} + \gamma \boldsymbol{P}\frac{\partial \boldsymbol{v_x}}{\partial \boldsymbol{x}} + \gamma \boldsymbol{P}\frac{\partial \boldsymbol{v_y}}{\partial \boldsymbol{y}} + \gamma \boldsymbol{P}\frac{\partial \boldsymbol{v_z}}{\partial \boldsymbol{z}} = 0 \qquad (119)$$

CHAPTER 4
FINITE ELEMENT IMPLEMENTATION

The two-dimensional forms of the governing equations are used in the subsequent sections. Therefore the component of velocity in the z-direction is zero and all partial derivatives with respect to z are zero. The bold font used to indicate dimensionless variables is not carried through the remaining sections because all of the variables are in dimensionless form. The governing equations are repeated here in two dimensions.

Conservation of Mass:

$$\frac{\partial \rho}{\partial t} + \rho \frac{\partial v_x}{\partial x} + v_x \frac{\partial \rho}{\partial x} + \rho \frac{\partial v_y}{\partial y} + v_y \frac{\partial \rho}{\partial y} = 0 \tag{120}$$

Conservation of Momentum, x-direction:

$$\frac{\partial v_x}{\partial t} + v_x \frac{\partial v_x}{\partial x} + v_y \frac{\partial v_x}{\partial y} + \frac{1}{\rho} \frac{\partial P}{\partial x} = 0 \tag{121}$$

Conservation of Momentum, y-direction:

$$\frac{\partial v_y}{\partial t} + v_x \frac{\partial v_y}{\partial x} + v_y \frac{\partial v_y}{\partial y} + \frac{1}{\rho} \frac{\partial P}{\partial y} = 0 \tag{122}$$

Conservation of Energy:

$$\frac{\partial P}{\partial t} + v_x \frac{\partial P}{\partial x} + v_y \frac{\partial P}{\partial y} + \gamma P \frac{\partial v_x}{\partial x} + \gamma P \frac{\partial v_y}{\partial y} = 0 \tag{123}$$

In Sections 4.1 and 4.2, the governing equations are converted to linear forms using Newton's method and then they are converted to finite element equations using the least squares method. Boundary conditions and their finite element implementations are presented in Section 4.3. Section 4.4 explains the conversion from global to local coordinates that is used to facilitate numerical integration. A mesh adaptation scheme is described in Section 4.5.

4.1 Linearization

The dependent variables of the governing equations are ρ, v_x, v_y, and P. The governing equations are nonlinear because they contain products of the dependent variables and their

derivatives. Newton's method is used to linearize the governing equations before they are converted to finite element equations, which will yield a symmetric positive definite matrix. Newton's method can be applied after the governing equations are converted to finite element equations but the resulting matrix will not be symmetric positive definite. First, Eqs. (120) to (123) are written as residuals.

$$\mathscr{R}_1 = \frac{\partial \rho}{\partial t} + \rho \frac{\partial v_x}{\partial x} + v_x \frac{\partial \rho}{\partial x} + \rho \frac{\partial v_y}{\partial y} + v_y \frac{\partial \rho}{\partial y} \tag{124}$$

$$\mathscr{R}_2 = \frac{\partial v_x}{\partial t} + v_x \frac{\partial v_x}{\partial x} + v_y \frac{\partial v_x}{\partial y} + \frac{1}{\rho} \frac{\partial P}{\partial x} \tag{125}$$

$$\mathscr{R}_3 = \frac{\partial v_y}{\partial t} + v_x \frac{\partial v_y}{\partial x} + v_y \frac{\partial v_y}{\partial y} + \frac{1}{\rho} \frac{\partial P}{\partial y} \tag{126}$$

$$\mathscr{R}_4 = \frac{\partial P}{\partial t} + v_x \frac{\partial P}{\partial x} + v_y \frac{\partial P}{\partial y} + \gamma P \frac{\partial v_x}{\partial x} + \gamma P \frac{\partial v_y}{\partial y} \tag{127}$$

The dependent variables and the residuals are assembled into vectors.

$$\{U\} = \left\{ \rho \quad \frac{\partial \rho}{\partial t} \quad \frac{\partial \rho}{\partial x} \quad \frac{\partial \rho}{\partial y} \quad v_x \quad \frac{\partial v_x}{\partial t} \quad \frac{\partial v_x}{\partial x} \quad \frac{\partial v_x}{\partial y} \quad v_y \quad \frac{\partial v_y}{\partial t} \quad \frac{\partial v_y}{\partial x} \quad \frac{\partial v_y}{\partial y} \quad P \quad \frac{\partial P}{\partial t} \quad \frac{\partial P}{\partial x} \quad \frac{\partial P}{\partial y} \right\}^T \tag{128}$$

$$\{\mathscr{R}\} = \left\{ \mathscr{R}_1 \quad \mathscr{R}_2 \quad \mathscr{R}_3 \quad \mathscr{R}_4 \right\}^T \tag{129}$$

The Jacobian of $\{\mathscr{R}\}$ with respect to $\{U\}$ is calculated.

$$\frac{\partial \{\mathscr{R}\}}{\partial \{U\}} = \begin{bmatrix} \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} & 1 & v_x & v_y & \frac{\partial \rho}{\partial x} & 0 & \rho & 0 & \frac{\partial \rho}{\partial y} & 0 & 0 & \rho & 0 & 0 & 0 & 0 \\ -\frac{1}{\rho^2}\frac{\partial P}{\partial x} & 0 & 0 & 0 & \frac{\partial v_x}{\partial x} & 1 & v_x & v_y & \frac{\partial v_x}{\partial y} & 0 & 0 & 0 & 0 & 0 & \frac{1}{\rho} & 0 \\ -\frac{1}{\rho^2}\frac{\partial P}{\partial y} & 0 & 0 & 0 & \frac{\partial v_y}{\partial x} & 0 & 0 & 0 & \frac{\partial v_y}{\partial y} & 1 & v_x & v_y & 0 & 0 & 0 & \frac{1}{\rho} \\ 0 & 0 & 0 & 0 & \frac{\partial P}{\partial x} & 0 & \gamma P & 0 & \frac{\partial P}{\partial y} & 0 & 0 & \gamma P & \gamma\frac{\partial v_x}{\partial x} + \gamma\frac{\partial v_y}{\partial y} & 1 & v_x & v_y \end{bmatrix} \tag{130}$$

The Jacobian, $\{\mathscr{R}\}$, and $\{U\}$ are substituted into Eq. (131). The superscript s is used as an index variable to indicate different iteration steps in Newton's method.

$$\frac{\partial \{\mathscr{R}\}^s}{\partial \{U\}^s} \{\Delta U\}^{s+1} = -\{\mathscr{R}\}^s \tag{131}$$

$$\{\Delta U\}^{s+1} = \{U\}^{s+1} - \{U\}^s \tag{132}$$

Matrix multiplication is carried out to obtain four equations.

$$\frac{\partial}{\partial t}\left(\rho^{s+1}-\rho^{s}\right)+\frac{\partial v_x^s}{\partial x}\left(\rho^{s+1}-\rho^{s}\right)+v_x^s\frac{\partial}{\partial x}\left(\rho^{s+1}-\rho^{s}\right)+\frac{\partial v_y^s}{\partial y}\left(\rho^{s+1}-\rho^{s}\right)+v_y^s\frac{\partial}{\partial y}\left(\rho^{s+1}-\rho^{s}\right)+\frac{\partial \rho^s}{\partial x}\left(v_x^{s+1}-v_x^{s}\right)+$$
$$\rho^s\frac{\partial}{\partial x}\left(v_x^{s+1}-v_x^{s}\right)+\frac{\partial \rho^s}{\partial y}\left(v_y^{s+1}-v_y^{s}\right)+\rho^s\frac{\partial}{\partial y}\left(v_y^{s+1}-v_y^{s}\right)=-\frac{\partial \rho^s}{\partial t}-\rho^s\frac{\partial v_x^s}{\partial x}-v_x^s\frac{\partial \rho^s}{\partial x}-\rho^s\frac{\partial v_y^s}{\partial y}-v_y^s\frac{\partial \rho^s}{\partial y} \tag{133}$$

$$-\frac{1}{(\rho^2)^s}\frac{\partial P^s}{\partial x}\left(\rho^{s+1}-\rho^{s}\right)+\frac{\partial}{\partial t}\left(v_x^{s+1}-v_x^{s}\right)+\frac{\partial v_x^s}{\partial x}\left(v_x^{s+1}-v_x^{s}\right)+v_x^s\frac{\partial}{\partial x}\left(v_x^{s+1}-v_x^{s}\right)+v_y^s\frac{\partial}{\partial y}\left(v_x^{s+1}-v_x^{s}\right)+\frac{\partial v_x^s}{\partial y}\left(v_y^{s+1}-v_y^{s}\right)+$$
$$\frac{1}{\rho^s}\frac{\partial}{\partial x}\left(P^{s+1}-P^{s}\right)=-\frac{\partial v_x^s}{\partial t}-v_x^s\frac{\partial v_x^s}{\partial x}-v_y^s\frac{\partial v_x^s}{\partial y}-\frac{1}{\rho^s}\frac{\partial P}{\partial x} \tag{134}$$

$$-\frac{1}{(\rho^2)^s}\frac{\partial P^s}{\partial y}\left(\rho^{s+1}-\rho^{s}\right)+\frac{\partial v_y^s}{\partial x}\left(v_x^{s+1}-v_x^{s}\right)+\frac{\partial}{\partial t}\left(v_y^{s+1}-v_y^{s}\right)+v_x^s\frac{\partial}{\partial x}\left(v_y^{s+1}-v_y^{s}\right)+\frac{\partial v_y^s}{\partial y}\left(v_y^{s+1}-v_y^{s}\right)+v_y^s\frac{\partial}{\partial y}\left(v_y^{s+1}-v_y^{s}\right)+$$
$$\frac{1}{\rho^s}\frac{\partial}{\partial y}\left(P^{s+1}-P^{s}\right)=-\frac{\partial v_y^s}{\partial t}-v_x^s\frac{\partial v_y^s}{\partial x}-v_y^s\frac{\partial v_y^s}{\partial y}-\frac{1}{\rho^s}\frac{\partial P}{\partial y} \tag{135}$$

$$\frac{\partial P^s}{\partial x}\left(v_x^{s+1}-v_x^{s}\right)+\gamma P^s\frac{\partial}{\partial x}\left(v_x^{s+1}-v_x^{s}\right)+\frac{\partial P}{\partial y}\left(v_y^{s+1}-v_y^{s}\right)+\gamma P^s\frac{\partial}{\partial y}\left(v_y^{s+1}-v_y^{s}\right)+\frac{\partial}{\partial t}\left(P^{s+1}-P^{s}\right)+\gamma\frac{\partial v_x^s}{\partial x}\left(P^{s+1}-P^{s}\right)+$$
$$v_x^s\frac{\partial}{\partial x}\left(P^{s+1}-P^{s}\right)+\gamma\frac{\partial v_y^s}{\partial y}\left(P^{s+1}-P^{s}\right)+v_y^r\frac{\partial}{\partial y}\left(P^{s+1}-P^{s}\right)=-\frac{\partial P^s}{\partial t}-v_x^s\frac{\partial P^s}{\partial x}-v_y^s\frac{\partial P^s}{\partial y}-\gamma P^s\frac{\partial v_x^s}{\partial x}-\gamma P^s\frac{\partial v_y^s}{\partial y} \tag{136}$$

Terms are distributed through the parentheses and all terms are moved to the same side of the equal sign.

$$\begin{aligned}\frac{\partial \rho^{s+1}}{\partial t} + \rho^{s+1}\frac{\partial v_x^s}{\partial x} + v_x^s\frac{\partial \rho^{s+1}}{\partial x} + \rho^{s+1}\frac{\partial v_y^s}{\partial y} + v_y^s\frac{\partial \rho^{s+1}}{\partial y} + v_x^{s+1}\frac{\partial \rho^s}{\partial x} - v_x^s\frac{\partial \rho^s}{\partial x} + \\ \rho^s\frac{\partial v_x^{s+1}}{\partial x} - \rho^s\frac{\partial v_x^s}{\partial x} + v_y^{s+1}\frac{\partial \rho^s}{\partial y} - v_y^s\frac{\partial \rho^s}{\partial y} + \rho^s\frac{\partial v_y^{s+1}}{\partial y} - \rho^s\frac{\partial v_y^s}{\partial y} = 0\end{aligned} \tag{137}$$

$$\begin{aligned}-\frac{1}{(\rho^2)^s}\frac{\partial P^s}{\partial x}\rho^{s+1} + \frac{1}{\rho^s}\frac{\partial P^s}{\partial x} + \frac{\partial v_x^{s+1}}{\partial t} + v_x^{s+1}\frac{\partial v_x^s}{\partial x} + v_x^s\frac{\partial v_x^{s+1}}{\partial x} + \\ v_x^s\frac{\partial v_x^{s+1}}{\partial x} - v_x^s\frac{\partial v_x^s}{\partial x} + v_y^s\frac{\partial v_x^{s+1}}{\partial y} + v_y^{s+1}\frac{\partial v_x^s}{\partial y} - v_y^s\frac{\partial v_x^s}{\partial y} + \frac{1}{\rho^s}\frac{\partial P^{s+1}}{\partial x} = 0\end{aligned} \tag{138}$$

$$\begin{aligned}-\frac{1}{(\rho^2)^s}\frac{\partial P^s}{\partial y}\rho^{s+1} + \frac{1}{\rho^s}\frac{\partial P^s}{\partial y} + v_x^{s+1}\frac{\partial v_y^s}{\partial x} - v_x^s\frac{\partial v_y^s}{\partial x} + \frac{\partial v_y^{s+1}}{\partial t} + \\ v_x^s\frac{\partial v_y^{s+1}}{\partial x} + v_y^{s+1}\frac{\partial v_y^s}{\partial y} + v_y^s\frac{\partial v_y^{s+1}}{\partial y} - v_y^s\frac{\partial v_y^s}{\partial y} + \frac{1}{\rho^s}\frac{\partial P^{s+1}}{\partial y} = 0\end{aligned} \tag{139}$$

$$\begin{aligned}v_x^{s+1}\frac{\partial P^s}{\partial x} + \gamma P^s\frac{\partial v_x^{s+1}}{\partial x} - \gamma P^s\frac{\partial v_x^s}{\partial x} + v_y^{s+1}\frac{\partial P^s}{\partial y} + \gamma P^s\frac{\partial v_y^{s+1}}{\partial y} - \gamma P^s\frac{\partial v_y^s}{\partial y} + \frac{\partial P^{s+1}}{\partial t} + \\ \gamma P^{s+1}\frac{\partial v_x^s}{\partial x} + v_x^s\frac{\partial P^{s+1}}{\partial x} - v_x^s\frac{\partial P^s}{\partial x} + \gamma P^{s+1}\frac{\partial v_y^s}{\partial y} + v_y^s\frac{\partial P^{s+1}}{\partial y} - v_y^s\frac{\partial P^s}{\partial y} = 0\end{aligned} \tag{140}$$

The partial derivatives with respect to time are approximated with a backward finite difference and R_1 to R_4 are used to represent the resulting linearized residuals.

$$\begin{aligned}R_1 = &\frac{\rho^{s+1}}{\Delta t} - \frac{\rho^s}{\Delta t} + \rho^{s+1}\frac{\partial v_x^s}{\partial x} + v_x^s\frac{\partial \rho^{s+1}}{\partial x} + \rho^{s+1}\frac{\partial v_y^s}{\partial y} + v_y^s\frac{\partial \rho^{s+1}}{\partial y} + v_x^{s+1}\frac{\partial \rho^s}{\partial x} - \\ &v_x^s\frac{\partial \rho^s}{\partial x} + \rho^s\frac{\partial v_x^{s+1}}{\partial x} - \rho^s\frac{\partial v_x^s}{\partial x} + v_y^{s+1}\frac{\partial \rho^s}{\partial y} - v_y^s\frac{\partial \rho^s}{\partial y} + \rho^s\frac{\partial v_y^{s+1}}{\partial y} - \rho^s\frac{\partial v_y^s}{\partial y}\end{aligned} \tag{141}$$

$$\begin{aligned}R_2 = &-\frac{1}{(\rho^2)^s}\frac{\partial P^s}{\partial x}\rho^{s+1} + \frac{1}{\rho^s}\frac{\partial P^s}{\partial x} + \frac{v_x^{s+1}}{\Delta t} - \frac{v_x^s}{\Delta t} + v_x^{s+1}\frac{\partial v_x^s}{\partial x} + v_x^s\frac{\partial v_x^{s+1}}{\partial x} + \\ &v_x^s\frac{\partial v_x^{s+1}}{\partial x} - v_x^s\frac{\partial v_x^s}{\partial x} + v_y^s\frac{\partial v_x^{s+1}}{\partial y} + v_y^{s+1}\frac{\partial v_x^s}{\partial y} - v_y^s\frac{\partial v_x^s}{\partial y} + \frac{1}{\rho^s}\frac{\partial P^{s+1}}{\partial x}\end{aligned} \tag{142}$$

$$R_3 = -\frac{1}{(\rho^2)^s}\frac{\partial P^s}{\partial y}\rho^{s+1} + \frac{1}{\rho^s}\frac{\partial P^s}{\partial y} + v_x^{s+1}\frac{\partial v_y^s}{\partial x} - v_x^s\frac{\partial v_y^s}{\partial x} + \frac{v_y^{s+1}}{\Delta t} - \frac{v_y^s}{\Delta t} + v_x^s\frac{\partial v_y^{s+1}}{\partial x} + v_y^{s+1}\frac{\partial v_y^s}{\partial y} + v_y^s\frac{\partial v_y^{s+1}}{\partial y} - v_y^s\frac{\partial v_y^s}{\partial y} + \frac{1}{\rho^s}\frac{\partial P^{s+1}}{\partial y} \tag{143}$$

$$R_4 = v_x^{s+1}\frac{\partial P^s}{\partial x} + \gamma P^s\frac{\partial v_x^{s+1}}{\partial x} - \gamma P^s\frac{\partial v_x^s}{\partial x} + v_y^{s+1}\frac{\partial P^s}{\partial y} + \gamma P^s\frac{\partial v_y^{s+1}}{\partial y} - \gamma P^s\frac{\partial v_y^s}{\partial y} + \frac{P^{s+1}}{\Delta t} - \frac{P^s}{\Delta t} + \gamma P^{s+1}\frac{\partial v_x^s}{\partial x} + v_x^s\frac{\partial P^{s+1}}{\partial x} - v_x^s\frac{\partial P^s}{\partial x} + \gamma P^{s+1}\frac{\partial v_y^s}{\partial y} + v_y^s\frac{\partial P^{s+1}}{\partial y} - v_y^s\frac{\partial P^s}{\partial y} \tag{144}$$

The residuals are now linearized. The products of dependent variables have either been converted to products of dependent variables from a previous iteration step or products of one dependent variable from the current iteration step and a dependent variable from a previous iteration step.

4.2 Least-Squares Finite Element Method

In this section, the linearized residuals are converted to a finite element matrix. The first step in deriving the element equation is to define a functional of residuals.

$$I = \frac{1}{2}\int_\Omega \left(R_1^2 + R_2^2 + R_3^2 + R_4^2\right)\mathrm{d}x\mathrm{d}y \tag{145}$$

The solution to a problem is found by minimizing I over the domain of the problem which is accomplished by iteratively approaching $\delta I = 0$ on the elements that make up the domain.

$$\delta I = \int_\Omega \sum_{k=1}^{4} R_k \delta R_k \mathrm{d}x\mathrm{d}y \tag{146}$$

$$
\begin{aligned}
\delta R_k =& \frac{\partial R_k}{\partial \rho^{s+1}}\delta\rho^{s+1} + \frac{\partial R_k}{\partial\left(\partial\rho^{s+1}/\partial x\right)}\delta\left(\partial\rho^{s+1}/\partial x\right) + \frac{\partial R_k}{\partial\left(\partial\rho^{s+1}/\partial y\right)}\delta\left(\partial\rho^{s+1}/\partial y\right) + \\
& \frac{\partial R_k}{\partial v_x^{s+1}}\delta v_x^{s+1} + \frac{\partial R_k}{\partial\left(\partial v_x^{s+1}/\partial x\right)}\delta\left(\partial v_x^{s+1}/\partial x\right) + \frac{\partial R_k}{\partial\left(\partial v_x^{s+1}/\partial y\right)}\delta\left(\partial v_x^{s+1}/\partial y\right) + \\
& \frac{\partial R_k}{\partial v_y^{s+1}}\delta v_y^{s+1} + \frac{\partial R_k}{\partial\left(\partial v_y^{s+1}/\partial x\right)}\delta\left(\partial v_y^{s+1}/\partial x\right) + \frac{\partial R_k}{\partial\left(\partial v_y^{s+1}/\partial y\right)}\delta\left(\partial v_y^{s+1}/\partial y\right) + \\
& \frac{\partial R_k}{\partial P^{s+1}}\delta P^{s+1} + \frac{\partial R_k}{\partial\left(\partial P^{s+1}/\partial x\right)}\delta\left(\partial P^{s+1}/\partial x\right) + \frac{\partial R_k}{\partial\left(\partial P^{s+1}/\partial y\right)}\delta\left(\partial P^{s+1}/\partial y\right)
\end{aligned}
\tag{147}
$$

Equation (147) is substituted into Eq. (146).

$$
\begin{aligned}
&\delta I = \\
&\int_\Omega \sum_{k=1}^{4} R_k \left(\frac{\partial R_k}{\partial \rho^{s+1}}\delta\rho^{s+1} + \frac{\partial R_k}{\partial\left(\partial\rho^{s+1}/\partial x\right)}\delta\left(\partial\rho^{s+1}/\partial x\right) + \frac{\partial R_k}{\partial\left(\partial\rho^{s+1}/\partial y\right)}\delta\left(\partial\rho^{s+1}/\partial y\right) \right) \mathrm{d}x\mathrm{d}y \\
&+ \int_\Omega \sum_{k=1}^{4} R_k \left(\frac{\partial R_k}{\partial v_x^{s+1}}\delta v_x^{s+1} + \frac{\partial R_k}{\partial\left(\partial v_x^{s+1}/\partial x\right)}\delta\left(\partial v_x^{s+1}/\partial x\right) + \frac{\partial R_k}{\partial\left(\partial v_x^{s+1}/\partial y\right)}\delta\left(\partial v_x^{s+1}/\partial y\right) \right) \mathrm{d}x\mathrm{d}y \\
&+ \int_\Omega \sum_{k=1}^{4} R_k \left(\frac{\partial R_k}{\partial v_y^{s+1}}\delta v_y^{s+1} + \frac{\partial R_k}{\partial\left(\partial v_y^{s+1}/\partial x\right)}\delta\left(\partial v_y^{s+1}/\partial x\right) + \frac{\partial R_k}{\partial\left(\partial v_y^{s+1}/\partial y\right)}\delta\left(\partial v_y^{s+1}/\partial y\right) \right) \mathrm{d}x\mathrm{d}y \\
&+ \int_\Omega \sum_{k=1}^{4} R_k \left(\frac{\partial R_k}{\partial P^{s+1}}\delta P^{s+1} + \frac{\partial R_k}{\partial\left(\partial P^{s+1}/\partial x\right)}\delta\left(\partial P^{s+1}/\partial x\right) + \frac{\partial R_k}{\partial\left(\partial P^{s+1}/\partial y\right)}\delta\left(\partial P^{s+1}/\partial y\right) \right) \mathrm{d}x\mathrm{d}y
\end{aligned}
\tag{148}
$$

The dependent variables, partial derivatives, and variations are approxmiated in an element by Eq. (149), in which σ is used as a substitute for ρ, v_x, v_y, and P. Equal order interpolation is used for all variables because the discrete spaces do not have compatibility conditions.[3,4]

$$
\begin{aligned}
\sigma &\approx \sum_{j=1}^{N} \psi_j \sigma_j & \delta\sigma &\approx \sum_{i=1}^{N} \psi_i \delta\sigma_i \\
\frac{\partial\sigma}{\partial x} &\approx \sum_{j=1}^{N} \frac{\partial\psi_j}{\partial x}\sigma_j & \delta\left(\frac{\partial\sigma}{\partial x}\right) &\approx \sum_{i=1}^{N} \frac{\partial\psi_i}{\partial x}\delta\sigma_i \\
\frac{\partial\sigma}{\partial y} &\approx \sum_{j=1}^{N} \frac{\partial\psi_j}{\partial y}\sigma_j & \delta\left(\frac{\partial\sigma}{\partial y}\right) &\approx \sum_{i=1}^{N} \frac{\partial\psi_i}{\partial y}\delta\sigma_i
\end{aligned}
\tag{149}
$$

The dependent variable approximations are substituted into the linearized residuals, Eqs. (141) to (144). Only the dependent variables from iteration step $s+1$ are approximated. The variables without an i or j index are known from either an initial guess or the previous Newton iteration. Once approxmimations are substituted into a linearized equation the s and $s+1$ indices are dropped.

$$R_1 \approx \sum_{j=1}^{N} \left\{ \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] \rho_j + \left[\frac{\partial \rho}{\partial x} \psi_j + \rho \frac{\partial \psi_j}{\partial x} \right] v_{x_j} + \left[\frac{\partial \rho}{\partial y} \psi_j + \rho \frac{\partial \psi_j}{\partial y} \right] v_{y_j} - \left[\frac{1}{\Delta t} \rho + \frac{\partial v_x}{\partial x} \rho + \frac{\partial \rho}{\partial x} v_x + \frac{\partial v_y}{\partial y} \rho + \frac{\partial \rho}{\partial y} v_y \right] \right\} \tag{150}$$

$$R_2 \approx \sum_{j=1}^{N} \left\{ \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial x} \psi_j \right] \rho_j + \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] v_{x_j} + \left[\frac{\partial v_x}{\partial y} \psi_j \right] v_{y_j} + \left[\frac{1}{\rho} \frac{\partial \psi_j}{\partial x} \right] P_j - \left[\frac{1}{\Delta t} v_x + \frac{\partial v_x}{\partial x} v_x + \frac{\partial v_x}{\partial y} v_y - \frac{1}{\rho} \frac{\partial P}{\partial x} \right] \right\} \tag{151}$$

$$R_3 \approx \sum_{j=1}^{N} \left\{ \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial y} \psi_j \right] \rho_j + \left[\frac{\partial v_y}{\partial x} \psi_j \right] v_{x_j} + \left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] v_{y_j} + \left[\frac{1}{\rho} \frac{\partial \psi_j}{\partial y} \right] P_j - \left[\frac{1}{\Delta t} v_y + \frac{\partial v_y}{\partial x} v_x + \frac{\partial v_y}{\partial y} v_y - \frac{1}{\rho} \frac{\partial P}{\partial y} \right] \right\} \tag{152}$$

$$R_4 \approx \sum_{j=1}^{N} \left\{ \left[\frac{\partial P}{\partial x} \psi_j + \gamma P \frac{\partial \psi_j}{\partial x} \right] v_{x_j} + \left[\frac{\partial P}{\partial y} \psi_j + \gamma P \frac{\partial \psi_j}{\partial y} \right] v_{y_j} + \left[\left(\frac{1}{\Delta t} + \gamma \frac{\partial v_x}{\partial x} + \gamma \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] P_j - \left[\frac{1}{\Delta t} P + \frac{\partial P}{\partial x} v_x + \gamma P \frac{\partial v_x}{\partial x} + \frac{\partial P}{\partial y} v_y + \gamma P \frac{\partial v_y}{\partial y} \right] \right\} \tag{153}$$

The partial derivatives in the first integrand of Eq. (148) are calculated and then the dependent variable approximations are substituted.

$$\begin{aligned}
&\frac{\partial R_1}{\partial \rho^{s+1}}\delta\rho^{s+1} + \frac{\partial R_1}{\partial\left(\partial\rho^{s+1}/\partial x\right)}\delta\left(\partial\rho^{s+1}/\partial x\right) + \frac{\partial R_1}{\partial\left(\partial\rho^{s+1}/\partial y\right)}\delta\left(\partial\rho^{s+1}/\partial y\right) \\
&\quad = \left[\frac{1}{\Delta t} + \frac{\partial v_x^s}{\partial x} + \frac{\partial v_y^s}{\partial y}\right]\delta\rho^{s+1} + v_x^s\delta\left(\partial\rho^{s+1}/\partial x\right) + v_y^s\left(\partial\rho^{s+1}/\partial y\right) \\
&\quad \approx \sum_{i=1}^{N}\left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}\right)\psi_i + v_x\frac{\partial\psi_i}{\partial x} + v_y\frac{\partial\psi_i}{\partial y}\right]\delta\rho_i
\end{aligned} \tag{154}$$

$$\begin{aligned}
&\frac{\partial R_2}{\partial \rho^{s+1}}\delta\rho^{s+1} + \frac{\partial R_2}{\partial\left(\partial\rho^{s+1}/\partial x\right)}\delta\left(\partial\rho^{s+1}/\partial x\right) + \frac{\partial R_2}{\partial\left(\partial\rho^{s+1}/\partial y\right)}\delta\left(\partial\rho^{s+1}/\partial y\right) \\
&\qquad = \left[-\frac{1}{\left(\rho^s\right)^2}\frac{\partial P^s}{\partial x}\right]\delta\rho^{s+1} \approx \sum_{i=1}^{N}\left[-\frac{1}{\rho^2}\frac{\partial P}{\partial x}\psi_i\right]\delta\rho_i
\end{aligned} \tag{155}$$

$$\begin{aligned}
&\frac{\partial R_3}{\partial \rho^{s+1}}\delta\rho^{s+1} + \frac{\partial R_3}{\partial\left(\partial\rho^{s+1}/\partial x\right)}\delta\left(\partial\rho^{s+1}/\partial x\right) + \frac{\partial R_3}{\partial\left(\partial\rho^{s+1}/\partial y\right)}\delta\left(\partial\rho^{s+1}/\partial y\right) \\
&\qquad = \left[-\frac{1}{\left(\rho^s\right)^2}\frac{\partial P^s}{\partial y}\right]\delta\rho^{s+1} \approx \sum_{i=1}^{N}\left[-\frac{1}{\rho^2}\frac{\partial P}{\partial y}\psi_i\right]\delta\rho_i
\end{aligned} \tag{156}$$

$$\frac{\partial R_4}{\partial \rho^{s+1}}\delta\rho^{s+1} + \frac{\partial R_4}{\partial\left(\partial\rho^{s+1}/\partial x\right)}\delta\left(\partial\rho^{s+1}/\partial x\right) + \frac{\partial R_4}{\partial\left(\partial\rho^{s+1}/\partial y\right)}\delta\left(\partial\rho^{s+1}/\partial y\right) = 0 \tag{157}$$

The first integral in Eq. (148) is approximated by substituting the approximations in Eqs. (154) to (157) and (150) to (153) and rearranging the integrand. The sum over i and $\delta\rho_i$ are factored out of the integral and the sum over j is replaced by matrix multiplication. The domain of integration is a single element.

$$\begin{aligned}
&\int_{\Omega_e}\sum_{k=1}^{4} R_k\left(\frac{\partial R_k}{\partial \rho^{s+1}}\delta\rho^{s+1} + \frac{\partial R_k}{\partial\left(\partial\rho^{s+1}/\partial x\right)}\delta\left(\partial\rho^{s+1}/\partial x\right) + \frac{\partial R_k}{\partial\left(\partial\rho^{s+1}/\partial y\right)}\delta\left(\partial\rho^{s+1}/\partial y\right)\right)\mathrm{d}x\mathrm{d}y \\
&\qquad = \sum_{i=1}^{N}\left(\left[K_{11}\right]\{\rho\} + \left[K_{12}\right]\{v_x\} + \left[K_{13}\right]\{v_y\} + \left[K_{14}\right]\{P\} - \{f_1\}\right)\delta\rho_i
\end{aligned} \tag{158}$$

$$\left[K_{11_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] + \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial x} \psi_i \right] \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial x} \psi_j \right] + \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial y} \psi_i \right] \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial y} \psi_j \right] \right\} \mathrm{d}x\mathrm{d}y \tag{159}$$

$$\left[K_{12_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[\frac{\partial \rho}{\partial x} \psi_j + \rho \frac{\partial \psi_j}{\partial x} \right] + \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial x} \psi_i \right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] + \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial y} \psi_i \right] \left[\frac{\partial v_y}{\partial x} \psi_j \right] \right\} \mathrm{d}x\mathrm{d}y \tag{160}$$

$$\left[K_{13_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[\frac{\partial \rho}{\partial y} \psi_j + \rho \frac{\partial \psi_j}{\partial y} \right] + \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial x} \psi_i \right] \left[\frac{\partial v_x}{\partial y} \psi_j \right] + \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial y} \psi_i \right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] \right\} \mathrm{d}x\mathrm{d}y \tag{161}$$

$$\left[K_{14_{ij}}\right] = \int_{\Omega_e} \left\{ \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial x} \psi_i \right] \left[\frac{1}{\rho} \frac{\partial \psi_j}{\partial x} \right] + \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial y} \psi_i \right] \left[\frac{1}{\rho} \frac{\partial \psi_j}{\partial y} \right] \right\} \mathrm{d}x\mathrm{d}y \tag{162}$$

$$\left\{f_{1_i}\right\} = \int_{\Omega_e} \left\{ \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[\frac{1}{\Delta t} \rho + \frac{\partial v_x}{\partial x} \rho + \frac{\partial \rho}{\partial x} v_x + \frac{\partial v_y}{\partial y} \rho + \frac{\partial \rho}{\partial y} v_y \right] + \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial x} \psi_i \right] \left[\frac{1}{\Delta t} v_x + \frac{\partial v_x}{\partial x} v_x + \frac{\partial v_x}{\partial y} v_y - \frac{1}{\rho} \frac{\partial P}{\partial x} \right] + \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial y} \psi_i \right] \left[\frac{1}{\Delta t} v_y + \frac{\partial v_y}{\partial x} v_x + \frac{\partial v_y}{\partial y} v_y - \frac{1}{\rho} \frac{\partial P}{\partial y} \right] \right\} \mathrm{d}x\mathrm{d}y \tag{163}$$

The partial derivatives in the second integrand of Eq. (148) are calculated and then the dependent variable approximations are substituted.

$$\begin{aligned}\frac{\partial R_1}{\partial v_x^{s+1}}\delta v_x^{s+1} + \frac{\partial R_1}{\partial\left(\partial v_x^{s+1}/\partial x\right)}\delta\left(\partial v_x^{s+1}/\partial x\right) + \frac{\partial R_1}{\partial\left(\partial v_x^{s+1}/\partial y\right)}\delta\left(\partial v_x^{s+1}/\partial y\right) \\ = \frac{\partial \rho^s}{\partial x}\delta v_x^{s+1} + \rho^s\delta\left(\partial v_x^{s+1}/\partial x\right) \approx \sum_{i=1}^{N}\left[\frac{\partial \rho}{\partial x}\psi_i + \rho\frac{\partial \psi_i}{\partial x}\right]\delta v_{x_i}\end{aligned} \tag{164}$$

$$\begin{aligned}\frac{\partial R_2}{\partial v_x^{s+1}}\delta v_x^{s+1} + \frac{\partial R_2}{\partial\left(\partial v_x^{s+1}/\partial x\right)}\delta\left(\partial v_x^{s+1}/\partial x\right) + \frac{\partial R_2}{\partial\left(\partial v_x^{s+1}/\partial y\right)}\delta\left(\partial v_x^{s+1}/\partial y\right) \\ = \left[\frac{1}{\Delta t} + \frac{\partial v_x^s}{\partial x}\right]\delta v_x^{s+1} + v_x^s\delta\left(\partial v_x^{s+1}/\partial x\right) + v_y^s\delta\left(\partial v_x^{s+1}/\partial y\right) \\ \approx \sum_{i=1}^{N}\left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x}\right)\psi_i + v_x\frac{\partial \psi_i}{\partial x} + v_y\frac{\partial \psi_i}{\partial y}\right]\delta v_{x_i}\end{aligned} \tag{165}$$

$$\begin{aligned}\frac{\partial R_3}{\partial v_x^{s+1}}\delta v_x^{s+1} + \frac{\partial R_3}{\partial\left(\partial v_x^{s+1}/\partial x\right)}\delta\left(\partial v_x^{s+1}/\partial x\right) + \frac{\partial R_3}{\partial\left(\partial v_x^{s+1}/\partial y\right)}\delta\left(\partial v_x^{s+1}/\partial y\right) \\ = \frac{\partial v_y^s}{\partial x}\delta v_x^{s+1} \approx \sum_{i=1}^{N}\left[\frac{\partial v_y}{\partial x}\psi_i\right]\delta v_{x_i}\end{aligned} \tag{166}$$

$$\begin{aligned}\frac{\partial R_4}{\partial v_x^{s+1}}\delta v_x^{s+1} + \frac{\partial R_4}{\partial\left(\partial v_x^{s+1}/\partial x\right)}\delta\left(\partial v_x^{s+1}/\partial x\right) + \frac{\partial R_4}{\partial\left(\partial v_x^{s+1}/\partial y\right)}\delta\left(\partial v_x^{s+1}/\partial y\right) \\ = \frac{\partial P^s}{\partial x}\delta v_x^{s+1} + \gamma P^s\delta\left(\partial v_x^{s+1}/\partial x\right) \approx \sum_{i=1}^{N}\left[\frac{\partial P}{\partial x}\psi_i + \gamma P\frac{\partial \psi_i}{\partial x}\right]\delta v_{x_i}\end{aligned} \tag{167}$$

The second integral in Eq. (148) is approximated by substituting the approximations in Eqs. (164) to (167) and (150) to (153) and rearranging the integrand. The sum over i and δv_{x_i} are factored out of the integral and the sum over j is replaced by matrix multiplication. The domain of integration is a single element.

$$\int_{\Omega_e} \sum_{k=1}^{4} R_k \left(\frac{\partial R_k}{\partial v_x^{s+1}} \delta v_x^{s+1} + \frac{\partial R_k}{\partial \left({}^{\partial v_x^{s+1}}/_{\partial x} \right)} \delta \left({}^{\partial v_x^{s+1}}/_{\partial x} \right) + \frac{\partial R_k}{\partial \left({}^{\partial v_x^{s+1}}/_{\partial y} \right)} \delta \left({}^{\partial v_x^{s+1}}/_{\partial y} \right) \right) \mathrm{d}x\mathrm{d}y$$
$$= \sum_{i=1}^{N} \left([K_{21}] \{\rho\} + [K_{22}] \{v_x\} + [K_{23}] \{v_y\} + [K_{24}] \{P\} - \{f_2\} \right) \delta v_{x_i} \tag{168}$$

$$\left[K_{21_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\frac{\partial \rho}{\partial x} \psi_i + \rho \frac{\partial \psi_i}{\partial x} \right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] + \right.$$
$$\left. \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial x} \psi_j \right] + \left[\frac{\partial v_y}{\partial x} \psi_i \right] \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial y} \psi_j \right] + \right\} \mathrm{d}x\mathrm{d}y \tag{169}$$

$$\left[K_{22_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\frac{\partial \rho}{\partial x} \psi_i + \rho \frac{\partial \psi_i}{\partial x} \right] \left[\frac{\partial \rho}{\partial x} \psi_j + \rho \frac{\partial \psi_j}{\partial x} \right] + \right.$$
$$\left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] + \tag{170}$$
$$\left. \left[\frac{\partial v_y}{\partial x} \psi_i \right] \left[\frac{\partial v_y}{\partial x} \psi_j \right] + \left[\frac{\partial P}{\partial x} \psi_i + \gamma P \frac{\partial \psi_i}{\partial x} \right] \left[\frac{\partial P}{\partial x} \psi_j + \gamma P \frac{\partial \psi_j}{\partial x} \right] \right\} \mathrm{d}x\mathrm{d}y$$

$$\left[K_{23_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\frac{\partial \rho}{\partial x} \psi_i + \rho \frac{\partial \psi_i}{\partial x} \right] \left[\frac{\partial \rho}{\partial y} \psi_j + \rho \frac{\partial \psi_j}{\partial y} \right] + \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[\frac{\partial v_x}{\partial y} \psi_j \right] + \right.$$
$$\left. \left[\frac{\partial v_y}{\partial x} \psi_i \right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] + \left[\frac{\partial P}{\partial x} \psi_i + \gamma P \frac{\partial \psi_i}{\partial x} \right] \left[\frac{\partial P}{\partial y} \psi_j + \gamma P \frac{\partial \psi_j}{\partial y} \right] \right\} \mathrm{d}x\mathrm{d}y \tag{171}$$

$$\left[K_{24_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[\frac{1}{\rho} \frac{\partial \psi_j}{\partial x} \right] + \left[\frac{\partial v_y}{\partial x} \psi_i \right] \left[\frac{1}{\rho} \frac{\partial \psi_j}{\partial y} \right] + \right.$$
$$\left. \left[\frac{\partial P}{\partial x} \psi_i + \gamma P \frac{\partial \psi_i}{\partial x} \right] \left[\left(\frac{1}{\Delta t} + \gamma \frac{\partial v_x}{\partial x} + \gamma \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] \right\} \mathrm{d}x\mathrm{d}y \tag{172}$$

$$
\begin{aligned}
\{f_{2_i}\} = \int_{\Omega_e} \Bigg\{ & \left[\frac{\partial \rho}{\partial x}\psi_i + \rho\frac{\partial \psi_i}{\partial x}\right]\left[\frac{1}{\Delta t}\rho + \frac{\partial v_x}{\partial x}\rho + \frac{\partial \rho}{\partial x}v_x + \frac{\partial v_y}{\partial y}\rho + \frac{\partial \rho}{\partial y}v_y\right] + \\
& \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x}\right)\psi_i + v_x\frac{\partial \psi_i}{\partial x} + v_y\frac{\partial \psi_i}{\partial y}\right]\left[\frac{1}{\Delta t}v_x + \frac{\partial v_x}{\partial x}v_x + \frac{\partial v_x}{\partial y}v_y - \frac{1}{\rho}\frac{\partial P}{\partial x}\right] + \\
& \left[\frac{\partial v_y}{\partial x}\psi_i\right]\left[\frac{1}{\Delta t}v_y + \frac{\partial v_y}{\partial x}v_x + \frac{\partial v_y}{\partial y}v_y - \frac{1}{\rho}\frac{\partial P}{\partial y}\right] + \\
& \left[\frac{\partial P}{\partial x}\psi_i + \gamma P\frac{\partial \psi_i}{\partial x}\right]\left[\frac{1}{\Delta t}P + \frac{\partial P}{\partial x}v_x + \gamma P\frac{\partial v_x}{\partial x} + \frac{\partial P}{\partial y}v_y + \gamma P\frac{\partial v_y}{\partial y}\right]\Bigg\}\,\mathrm{d}x\mathrm{d}y
\end{aligned}
\tag{173}
$$

The partial derivatives in the third integrand of Eq. (148) are calculated and then the dependent variable approximations are substituted.

$$\begin{aligned}\frac{\partial R_1}{\partial v_y^{s+1}}\delta v_y^{s+1} + \frac{\partial R_1}{\partial\left(\partial v_y^{s+1}/\partial x\right)}\delta\left(\partial v_y^{s+1}/\partial x\right) + \frac{\partial R_1}{\partial\left(\partial v_y^{s+1}/\partial y\right)}\delta\left(\partial v_y^{s+1}/\partial y\right) \\ = \frac{\partial \rho^s}{\partial y}\delta v_y^{s+1} + \rho^s\delta\left(\partial v_y^{s+1}/\partial y\right) \approx \sum_{i=1}^{N}\left[\frac{\partial \rho}{\partial y}\psi_i + \rho\frac{\partial \psi_i}{\partial y}\right]\delta v_{y_i}\end{aligned} \tag{174}$$

$$\begin{aligned}\frac{\partial R_2}{\partial v_y^{s+1}}\delta v_y^{s+1} + \frac{\partial R_2}{\partial\left(\partial v_y^{s+1}/\partial x\right)}\delta\left(\partial v_y^{s+1}/\partial x\right) + \frac{\partial R_2}{\partial\left(\partial v_y^{s+1}/\partial y\right)}\delta\left(\partial v_y^{s+1}/\partial y\right) \\ = \frac{\partial v_x^s}{\partial y}\delta v_y^{s+1} \approx \sum_{i=1}^{N}\left[\frac{\partial v_x}{\partial y}\psi_i\right]\delta v_{y_i}\end{aligned} \tag{175}$$

$$\begin{aligned}\frac{\partial R_3}{\partial v_y^{s+1}}\delta v_y^{s+1} + \frac{\partial R_3}{\partial\left(\partial v_y^{s+1}/\partial x\right)}\delta\left(\partial v_y^{s+1}/\partial x\right) + \frac{\partial R_3}{\partial\left(\partial v_y^{s+1}/\partial y\right)}\delta\left(\partial v_y^{s+1}/\partial y\right) \\ = \left[\frac{1}{\Delta t} + \frac{\partial v_y^s}{\partial y}\right]\delta v_y^{s+1} + v_x^s\delta\left(\partial v_y^{s+1}/\partial x\right) + v_y^s\delta\left(\partial v_y^{s+1}/\partial y\right) \\ \approx \sum_{i=1}^{N}\left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y}\right)\psi_i + v_x\frac{\partial \psi_i}{\partial x} + v_y\frac{\partial \psi_i}{\partial y}\right]\delta v_{y_i}\end{aligned} \tag{176}$$

$$\begin{aligned}\frac{\partial R_4}{\partial v_y^{s+1}}\delta v_y^{s+1} + \frac{\partial R_4}{\partial\left(\partial v_y^{s+1}/\partial x\right)}\delta\left(\partial v_y^{s+1}/\partial x\right) + \frac{\partial R_4}{\partial\left(\partial v_y^{s+1}/\partial y\right)}\delta\left(\partial v_y^{s+1}/\partial y\right) \\ = \frac{\partial P^s}{\partial y}\delta v_y^{s+1} + \gamma P^s\delta\left(\partial v_y^{s+1}/\partial y\right) \approx \sum_{i=1}^{N}\left[\frac{\partial P}{\partial y}\psi_i + \gamma P\frac{\partial \psi_i}{\partial y}\right]\delta v_{y_i}\end{aligned} \tag{177}$$

The third integral in Eq. (148) is approximated by substituting the approximations in Eqs. (174) to (177) and (150) to (153) and rearranging the integrand. The sum over i and δv_{y_i} are factored out of the integral and the sum over j is replaced by matrix multiplication. The domain of integration is a single element.

$$\int_{\Omega_e} \sum_{k=1}^{4} R_k \left(\frac{\partial R_k}{\partial v_y^{s+1}} \delta v_y^{s+1} + \frac{\partial R_k}{\partial \left({}^{\partial v_y^{s+1}}/_{\partial x} \right)} \delta \left({}^{\partial v_y^{s+1}}/_{\partial x} \right) + \frac{\partial R_k}{\partial \left({}^{\partial v_y^{s+1}}/_{\partial y} \right)} \delta \left({}^{\partial v_y^{s+1}}/_{\partial y} \right) \right) \mathrm{d}x\mathrm{d}y$$
$$= \sum_{i=1}^{N} \left([K_{31}] \{\rho\} + [K_{32}] \{v_x\} + [K_{33}] \{v_y\} + [K_{34}] \{P\} - \{f_3\} \right) \delta v_{y_i} \tag{178}$$

$$\left[K_{31_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\frac{\partial \rho}{\partial y} \psi_i + \rho \frac{\partial \psi_i}{\partial y} \right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] + \left[\frac{\partial v_x}{\partial y} \psi_i \right] \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial x} \psi_j \right] + \left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[-\frac{1}{\rho^2} \frac{\partial P}{\partial y} \psi_j \right] \right\} \mathrm{d}x\mathrm{d}y \tag{179}$$

$$\left[K_{32_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\frac{\partial \rho}{\partial y} \psi_i + \rho \frac{\partial \psi_i}{\partial y} \right] \left[\frac{\partial \rho}{\partial x} \psi_j + \rho \frac{\partial \psi_j}{\partial x} \right] + \left[\frac{\partial v_x}{\partial y} \psi_i \right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] + \left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[\frac{\partial v_y}{\partial x} \psi_j \right] + \left[\frac{\partial P}{\partial y} \psi_i + \gamma P \frac{\partial \psi_i}{\partial y} \right] \left[\frac{\partial P}{\partial x} \psi_j + \gamma P \frac{\partial \psi_j}{\partial x} \right] \right\} \mathrm{d}x\mathrm{d}y \tag{180}$$

$$\left[K_{33_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\frac{\partial \rho}{\partial y} \psi_i + \rho \frac{\partial \psi_i}{\partial y} \right] \left[\frac{\partial \rho}{\partial y} \psi_j + \rho \frac{\partial \psi_j}{\partial y} \right] + \left[\frac{\partial v_x}{\partial y} \psi_i \right] \left[\frac{\partial v_x}{\partial y} \psi_j \right] + \left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] + \left[\frac{\partial P}{\partial y} \psi_i + \gamma P \frac{\partial \psi_i}{\partial y} \right] \left[\frac{\partial P}{\partial y} \psi_j + \gamma P \frac{\partial \psi_j}{\partial y} \right] \right\} \mathrm{d}x\mathrm{d}y \tag{181}$$

$$\left[K_{34_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\frac{\partial v_x}{\partial y} \psi_i \right] \left[\frac{1}{\rho} \frac{\partial \psi_j}{\partial x} \right] + \left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y} \right) \psi_i + v_x \frac{\partial \psi_i}{\partial x} + v_y \frac{\partial \psi_i}{\partial y} \right] \left[\frac{1}{\rho} \frac{\partial \psi_j}{\partial y} \right] + \left[\frac{\partial P}{\partial y} \psi_i + \gamma P \frac{\partial \psi_i}{\partial y} \right] \left[\left(\frac{1}{\Delta t} + \gamma \frac{\partial v_x}{\partial x} + \gamma \frac{\partial v_y}{\partial y} \right) \psi_j + v_x \frac{\partial \psi_j}{\partial x} + v_y \frac{\partial \psi_j}{\partial y} \right] \right\} \mathrm{d}x\mathrm{d}y \tag{182}$$

$$\{f_{3_i}\} = \int_{\Omega_e} \left\{ \left[\frac{\partial \rho}{\partial y}\psi_i + \rho\frac{\partial \psi_i}{\partial y}\right] \left[\frac{1}{\Delta t}\rho + \frac{\partial v_x}{\partial x}\rho + \frac{\partial \rho}{\partial x}v_x + \frac{\partial v_y}{\partial y}\rho + \frac{\partial \rho}{\partial y}v_y\right] + \left[\frac{\partial v_x}{\partial y}\psi_i\right] \left[\frac{1}{\Delta t}v_x + \frac{\partial v_x}{\partial x}v_x + \frac{\partial v_x}{\partial y}v_y - \frac{1}{\rho}\frac{\partial P}{\partial x}\right] + \left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y}\right)\psi_i + v_x\frac{\partial \psi_i}{\partial x} + v_y\frac{\partial \psi_i}{\partial y}\right] \left[\frac{1}{\Delta t}v_y + \frac{\partial v_y}{\partial x}v_x + \frac{\partial v_y}{\partial y}v_y - \frac{1}{\rho}\frac{\partial P}{\partial y}\right] + \left[\frac{\partial P}{\partial y}\psi_i + \gamma P\frac{\partial \psi_i}{\partial y}\right] \left[\frac{1}{\Delta t}P + \frac{\partial P}{\partial x}v_x + \gamma P\frac{\partial v_x}{\partial x} + \frac{\partial P}{\partial y}v_y + \gamma P\frac{\partial v_y}{\partial y}\right] \right\} \mathrm{d}x\mathrm{d}y \tag{183}$$

The partial derivatives in the fourth integrand of Eq. (148) are calculated and then the dependent variable approximations are substituted.

$$\frac{\partial R_1}{\partial P^{s+1}}\delta P^{s+1} + \frac{\partial R_1}{\partial \left(\partial P^{s+1}/\partial x\right)}\delta\left(\partial P^{s+1}/\partial x\right) + \frac{\partial R_1}{\partial \left(\partial P^{s+1}/\partial y\right)}\delta\left(\partial P^{s+1}/\partial y\right) = 0 \tag{184}$$

$$\begin{aligned}\frac{\partial R_2}{\partial P^{s+1}}\delta P^{s+1} + \frac{\partial R_2}{\partial \left(\partial P^{s+1}/\partial x\right)}\delta\left(\partial P^{s+1}/\partial x\right) + \frac{\partial R_2}{\partial \left(\partial P^{s+1}/\partial y\right)}\delta\left(\partial P^{s+1}/\partial y\right) \\ = \frac{1}{\rho^s}\delta\left(\partial P^{s+1}/\partial x\right) \approx \sum_{i=1}^{N}\left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial x}\right]\delta P_i\end{aligned} \tag{185}$$

$$\begin{aligned}\frac{\partial R_3}{\partial P^{s+1}}\delta P^{s+1} + \frac{\partial R_3}{\partial \left(\partial P^{s+1}/\partial x\right)}\delta\left(\partial P^{s+1}/\partial x\right) + \frac{\partial R_3}{\partial \left(\partial P^{s+1}/\partial y\right)}\delta\left(\partial P^{s+1}/\partial y\right) \\ = \frac{1}{\rho^s}\delta\left(\partial P^{s+1}/\partial y\right) \approx \sum_{i=1}^{N}\left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial y}\right]\delta P_i\end{aligned} \tag{186}$$

$$\begin{aligned}&\frac{\partial R_4}{\partial P^{s+1}}\delta P^{s+1} + \frac{\partial R_4}{\partial \left(\partial P^{s+1}/\partial x\right)}\delta\left(\partial P^{s+1}/\partial x\right) + \frac{\partial R_4}{\partial \left(\partial P^{s+1}/\partial y\right)}\delta\left(\partial P^{s+1}/\partial y\right) \\ &= \left[\frac{1}{\Delta t} + \gamma\frac{\partial v_x^s}{\partial x} + \gamma\frac{\partial v_y^s}{\partial y}\right]\delta P^{s+1} + v_x^s\delta\left(\partial P^{s+1}/\partial x\right) + v_y^s\delta\left(\partial P^{s+1}/\partial y\right) \\ &\approx \sum_{i=1}^{N}\left[\left(\frac{1}{\Delta t} + \gamma\frac{\partial v_x}{\partial x} + \gamma\frac{\partial v_y}{\partial y}\right)\psi_i + v_x\frac{\partial \psi_i}{\partial x} + v_y\frac{\partial \psi_i}{\partial y}\right]\delta P_i\end{aligned} \tag{187}$$

The fourth integral in Eq. (148) is approximated by substituting the approximations in Eqs. (184) to (187) and (150) to (153) and rearranging the integrand. The sum over i and δP_i are factored out of the integral and the sum over j is replaced by matrix multiplication. The domain of integration is a single element.

$$\begin{aligned}\int_{\Omega_e}\sum_{k=1}^{4} R_k\left(\frac{\partial R_k}{\partial P^{s+1}}\delta P^{s+1} + \frac{\partial R_k}{\partial \left(\partial P^{s+1}/\partial x\right)}\delta\left(\partial P^{s+1}/\partial x\right) + \frac{\partial R_k}{\partial \left(\partial P^{s+1}/\partial y\right)}\delta\left(\partial P^{s+1}/\partial y\right)\right)\mathrm{d}x\mathrm{d}y \\ = \sum_{i=1}^{N}\left(\left[K_{41}\right]\{\rho\} + \left[K_{42}\right]\{v_x\} + \left[K_{43}\right]\{v_y\} + \left[K_{44}\right]\{P\} - \{f_4\}\right)\delta P_i\end{aligned} \tag{188}$$

$$\left[K_{41_{ij}}\right] = \int_{\Omega_e} \left\{ \left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial x}\right] \left[-\frac{1}{\rho^2}\frac{\partial P}{\partial x}\psi_j\right] + \left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial y}\right] \left[-\frac{1}{\rho^2}\frac{\partial P}{\partial y}\psi_j\right] \right\} \mathrm{d}x\mathrm{d}y \tag{189}$$

$$\begin{aligned}\left[K_{42_{ij}}\right] = \int_{\Omega_e} \Bigg\{ & \left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial x}\right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x}\right)\psi_j + v_x\frac{\partial \psi_j}{\partial x} + v_y\frac{\partial \psi_j}{\partial y}\right] + \left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial y}\right] \left[\frac{\partial v_y}{\partial x}\psi_j\right] + \\ & \left[\left(\frac{1}{\Delta t} + \gamma\frac{\partial v_x}{\partial x} + \gamma\frac{\partial v_y}{\partial y}\right)\psi_i + v_x\frac{\partial \psi_i}{\partial x} + v_y\frac{\partial \psi_i}{\partial y}\right] \left[\frac{\partial P}{\partial x}\psi_j + \gamma P\frac{\partial \psi_j}{\partial x}\right] \Bigg\} \mathrm{d}x\mathrm{d}y\end{aligned} \tag{190}$$

$$\begin{aligned}\left[K_{43_{ij}}\right] = \int_{\Omega_e} \Bigg\{ & \left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial x}\right] \left[\frac{\partial v_x}{\partial y}\psi_j\right] + \left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial y}\right] \left[\left(\frac{1}{\Delta t} + \frac{\partial v_y}{\partial y}\right)\psi_j + v_x\frac{\partial \psi_j}{\partial x} + v_y\frac{\partial \psi_j}{\partial y}\right] + \\ & \left[\left(\frac{1}{\Delta t} + \gamma\frac{\partial v_x}{\partial x} + \gamma\frac{\partial v_y}{\partial y}\right)\psi_i + v_x\frac{\partial \psi_i}{\partial x} + v_y\frac{\partial \psi_i}{\partial y}\right] \left[\frac{\partial P}{\partial y}\psi_j + \gamma P\frac{\partial \psi_j}{\partial y}\right] \Bigg\} \mathrm{d}x\mathrm{d}y\end{aligned} \tag{191}$$

$$\begin{aligned}\left[K_{44_{ij}}\right] = \int_{\Omega_e} \Bigg\{ & \left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial x}\right] \left[\frac{1}{\rho}\frac{\partial \psi_j}{\partial x}\right] + \left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial y}\right] \left[\frac{1}{\rho}\frac{\partial \psi_j}{\partial y}\right] + \\ & \left[\left(\frac{1}{\Delta t} + \gamma\frac{\partial v_x}{\partial x} + \gamma\frac{\partial v_y}{\partial y}\right)\psi_i + v_x\frac{\partial \psi_i}{\partial x} + v_y\frac{\partial \psi_i}{\partial y}\right] \left[\left(\frac{1}{\Delta t} + \gamma\frac{\partial v_x}{\partial x} + \gamma\frac{\partial v_y}{\partial y}\right)\psi_j + v_x\frac{\partial \psi_j}{\partial x} + v_y\frac{\partial \psi_j}{\partial y}\right] \Bigg\} \mathrm{d}x\mathrm{d}y\end{aligned} \tag{192}$$

$$\begin{aligned}\left\{f_{4_i}\right\} = \int_{\Omega_e} \Bigg\{ & \left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial x}\right] \left[\frac{1}{\Delta t}v_x + \frac{\partial v_x}{\partial x}v_x + \frac{\partial v_x}{\partial y}v_y - \frac{1}{\rho}\frac{\partial P}{\partial x}\right] + \left[\frac{1}{\rho}\frac{\partial \psi_i}{\partial y}\right] \left[\frac{1}{\Delta t}v_y + \frac{\partial v_y}{\partial x}v_x + \frac{\partial v_y}{\partial y}v_y - \frac{1}{\rho}\frac{\partial P}{\partial y}\right] + \\ & \left[\left(\frac{1}{\Delta t} + \gamma\frac{\partial v_x}{\partial x} + \gamma\frac{\partial v_y}{\partial y}\right)\psi_i + v_x\frac{\partial \psi_i}{\partial x} + v_y\frac{\partial \psi_i}{\partial y}\right] \left[\frac{1}{\Delta t}P + \frac{\partial P}{\partial x}v_x + \gamma P\frac{\partial v_x}{\partial x} + \frac{\partial P}{\partial y}v_y + \gamma P\frac{\partial v_y}{\partial y}\right] \Bigg\} \mathrm{d}x\mathrm{d}y\end{aligned} \tag{193}$$

Equations (158) to (188) are substituted into (148).

$$\begin{aligned}\delta I_e \approx & \sum_{i=1}^{N} \left([K_{11}]\{\rho\} + [K_{12}]\{v_x\} + [K_{13}]\{v_y\} + [K_{14}]\{P\} - \{f_1\} \right) \delta\rho_i + \\ & \sum_{i=1}^{N} \left([K_{21}]\{\rho\} + [K_{22}]\{v_x\} + [K_{23}]\{v_y\} + [K_{24}]\{P\} - \{f_2\} \right) \delta v_{x_i} + \\ & \sum_{i=1}^{N} \left([K_{31}]\{\rho\} + [K_{32}]\{v_x\} + [K_{33}]\{v_y\} + [K_{34}]\{P\} - \{f_3\} \right) \delta v_{y_i} + \\ & \sum_{i=1}^{N} \left([K_{41}]\{\rho\} + [K_{42}]\{v_x\} + [K_{43}]\{v_y\} + [K_{44}]\{P\} - \{f_4\} \right) \delta P_i\end{aligned} \tag{194}$$

The variations of the dependent variables are arbitrary numbers and, in general, they are not zero. In order for δI_e to approach zero the terms in parentheses inside the summations must approach 0. The result is four sets of N linear equations that are arranged in matrix form for each element, as shown in Eq. (195).

$$\begin{bmatrix} [K_{11}] & [K_{12}] & [K_{13}] & [K_{14}] \\ [K_{21}] & [K_{22}] & [K_{23}] & [K_{24}] \\ [K_{31}] & [K_{32}] & [K_{33}] & [K_{34}] \\ [K_{41}] & [K_{42}] & [K_{43}] & [K_{44}] \end{bmatrix} \begin{Bmatrix} \{\rho\} \\ \{v_x\} \\ \{v_y\} \\ \{P\} \end{Bmatrix} = \begin{Bmatrix} \{f_1\} \\ \{f_2\} \\ \{f_3\} \\ \{f_4\} \end{Bmatrix} \tag{195}$$

The system of equations is solved by using either an initial guess or the result of the previous iteration to calculate $[K_{11}]$ to $[K_{44}]$ and $\{f_1\}$ to $\{f_4\}$ for each element, assembling the element equations, imposing boundary conditions, and solving the linear system for the entire domain. The assembled element equations are solved using the preconditioned conjugate gradient (PCG) method. For the calculations presented in Chapter 5, a Jacobi preconditioner is used. If the PCG method does not converge within a specified number of iterations, the PCG solver is restarted using a symmetric Gauss-Seidel preconditioner and the solution vector with the lowest residual from the previous PCG attempt.

The solution method can be succinctly described as a PCG iteration loop nested inside a Newton iteration loop with the finite element equations recalculated for each Newton iteration. The Newton iterations proceed until Eq. (196) is satisfied. The magnitude of velocity at node j is V_j and $\|V_j\|$ is the Euclidean norm of the magnitude of velocity for every node in the domain.

$$\frac{\|V_j^{s+1} - V_j^s\|}{\|V_j^s\|} \leq 1 \times 10^{-5} \tag{196}$$

4.3 Boundary Conditions

Two common types of boundary conditions used for inviscid flows are explained here. Either all of the fluid properties are specified on a boundary or a boundary is treated as a solid object. Usually, all of the free stream properties are known and they are used as boundary conditions for the upstream boundary of a fluid dynamics problem. If the problem at hand is a flow in free space, such as an airfoil moving through a fluid without any other objects nearby, the free stream boundary conditions may be imposed on other boundaries of the problem as long as the domain is large enough to prevent the disturbed flow around the airfoil from interacting with the boundary.

Solid wall boundary conditions are imposed on boundaries that do not allow the fluid to pass through. Using the airfoil example again, if the airfoil were placed in a wind tunnel, instead of specifying free stream conditions on boundaries far away from the airfoil, the walls of the wind tunnel would be treated as solid objects. In that case, the boundaries may be moved closer to the airfoil so that they are coincident with the wind tunnel walls, but there may be some interaction between the flow around the airfoil and the flow near the walls. In both hypothetical problems, the edges of the airfoil would also be treated as solid walls. At the interface between a solid object and a fluid, the fluid flow is either parallel to the edge of the solid object or static. For an inviscid flow, a solid wall boundary condition is imposed by applying Eq. (197). The velocity vector of the fluid is $\vec{v}$ and the outward pointing unit normal vector of a fluid element is $\vec{n}$.

$$\vec{v} \cdot \vec{n} = 0 \tag{197}$$

A third type of boundary condition that is not used in this thesis is the outflow or downstream boundary condition. For the examples presented in Chapter 5, there is no boundary condition imposed on the downstream boundary. For a discussion of downstream boundary conditions, see Reference 7.

At any point where the flow properties are known, the flow properties are treated as essential boundary conditions. Equations (198) to (202) are taken from Reference 8 and essential boundary conditions are implemented as described therein. The index s is used to refer to the index of a specified degree of freedom.

$$\widehat{F}_i = F_i - K_{is}U_s \quad \forall i \neq s \tag{198}$$

$$K_{is} = K_{si} = 0 \quad \forall i \neq s \tag{199}$$

$$K_{ss} = 1 \tag{200}$$

Equations (198) to (200) are applied to the linear system in (201). For $s = 2$ the result is (202).

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} & \cdots & K_{1n} \\ K_{21} & K_{22} & K_{23} & K_{24} & \cdots & K_{2n} \\ K_{31} & K_{32} & K_{33} & K_{34} & \cdots & K_{3n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{n1} & K_{n2} & K_{n3} & K_{n4} & \cdots & K_{nn} \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_n \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_n \end{Bmatrix} \tag{201}$$

$$\begin{bmatrix} K_{11} & 0 & K_{13} & K_{14} & \cdots & K_{1n} \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ K_{31} & 0 & K_{33} & K_{34} & \cdots & K_{3n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{n1} & 0 & K_{n3} & K_{n4} & \cdots & K_{nn} \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_n \end{Bmatrix} = \begin{Bmatrix} \widehat{F}_1 \\ U_2 \\ \widehat{F}_3 \\ \vdots \\ \widehat{F}_n \end{Bmatrix} \tag{202}$$

If a solid boundary is straight and parallel to an axis in the global coordinate system, the solid wall boundary condition may be imposed by requiring the velocity component perpendicular to the boundary to be zero and applying Eqs. (198) to (200). Another option that can handle more complex boundaries is to use the least-squares finite element method. Equation (197) is written as a residual and a functional is defined with a penalty weight (ϑ).

$$R_5 = \vec{v} \cdot \vec{n} = v_x n_x + v_y n_y \tag{203}$$

$$I = \vartheta \frac{1}{2} \int_{\Gamma_{\text{wall}}} R_5^2 \mathrm{d}x\mathrm{d}y \tag{204}$$

The same logic used in Section 4.2 to develop the element coefficient matrix is applied here. Equation (205) is the resulting finite element matrix for a solid wall boundary condition applied to an element.

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \vartheta\left[Q_{xx}\right] & \vartheta\left[Q_{xy}\right] & 0 \\ 0 & \vartheta\left[Q_{yx}\right] & \vartheta\left[Q_{yy}\right] & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \{\rho\} \\ \{v_x\} \\ \{v_y\} \\ \{P\} \end{Bmatrix} = \begin{Bmatrix} \{0\} \\ \{0\} \\ \{0\} \\ \{0\} \end{Bmatrix} \tag{205}$$

$$\left[Q_{xx_{ij}}\right] = \int_{\Gamma_{\text{wall}}} \left[n_x\psi_i\right]\left[n_x\psi_j\right] \mathrm{d}x\mathrm{d}y \tag{206}$$

$$\left[Q_{xy_{ij}}\right] = \int_{\Gamma_{\text{wall}}} \left[n_x\psi_i\right]\left[n_y\psi_j\right] \mathrm{d}x\mathrm{d}y \tag{207}$$

$$\left[Q_{yx_{ij}}\right] = \int_{\Gamma_{\text{wall}}} \left[n_y\psi_i\right]\left[n_x\psi_j\right] \mathrm{d}x\mathrm{d}y \tag{208}$$

$$\left[Q_{yy_{ij}}\right] = \int_{\Gamma_{\text{wall}}} \left[n_y\psi_i\right]\left[n_y\psi_j\right] \mathrm{d}x\mathrm{d}y \tag{209}$$

Equation (205) is added to the element coefficient matrix for elements that have solid wall boundary conditions.

$$\begin{bmatrix} [K_{11}] & [K_{12}] & [K_{13}] & [K_{14}] \\ [K_{21}] & [K_{22}] + \vartheta\left[Q_{xx}\right] & [K_{23}] + \vartheta\left[Q_{xy}\right] & [K_{24}] \\ [K_{31}] & [K_{32}] + \vartheta\left[Q_{yx}\right] & [K_{33}] + \vartheta\left[Q_{yy}\right] & [K_{34}] \\ [K_{41}] & [K_{42}] & [K_{43}] & [K_{44}] \end{bmatrix} \begin{Bmatrix} \{\rho\} \\ \{v_x\} \\ \{v_y\} \\ \{P\} \end{Bmatrix} = \begin{Bmatrix} \{f_1\} \\ \{f_2\} \\ \{f_3\} \\ \{f_4\} \end{Bmatrix} \tag{210}$$

4.4 Conversion to Local Coordinates

To facilitate numerical integration, the components of Eq. (210) are converted to the local coordinate system of a master element. The information and formulas presented in this section, with the exception of those pertaining to the solid wall boundary conditions, are from Reference 8. As stated earlier, ξ and η represent orthogonal coordinate directions in the local coordinate system of a master element. A hat over a variable, $\widehat{\phi}$ for example, indicates that the variable represents a function in the local coordinate system.

The global coordinates of an element are approximated using Eqs. (211) and (212). The

global coordinates of the element nodes are represented by (x_j, y_j). The shape functions, $\widehat{\phi}_j$, are products of one-dimensional Lagrange polynomials. The use of exponential functions to convert from global coordinates to local coordinates was not investigated.

$$x = \sum_{j=1}^{N} \widehat{\phi}_j x_j \tag{211}$$

$$y = \sum_{j=1}^{N} \widehat{\phi}_j y_j \tag{212}$$

The Jacobian is calculated by arranging the partial derivatives of (x, y) with respect to (ξ, η) in a matrix.

$$[J] = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \widehat{\phi}_1}{\partial \xi} & \dfrac{\partial \widehat{\phi}_2}{\partial \xi} & \cdots & \dfrac{\partial \widehat{\phi}_N}{\partial \xi} \\ \dfrac{\partial \widehat{\phi}_1}{\partial \eta} & \dfrac{\partial \widehat{\phi}_2}{\partial \eta} & \cdots & \dfrac{\partial \widehat{\phi}_N}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_N & y_N \end{bmatrix} \tag{213}$$

The interpolation functions are already known in the local coordinate system. To find the derivatives of the interpolation functions with respect to the global coordinate system, the chain rule of differentiation is applied.

$$\frac{\partial \widehat{\psi}_i}{\partial \xi} = \frac{\partial \psi_i}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial \psi_i}{\partial y}\frac{\partial y}{\partial \xi} \tag{214}$$

$$\frac{\partial \widehat{\psi}_i}{\partial \eta} = \frac{\partial \psi_i}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial \psi_i}{\partial y}\frac{\partial y}{\partial \eta} \tag{215}$$

The partial derivatives are written in matrix form.

$$\begin{Bmatrix} \dfrac{\partial \widehat{\psi}_i}{\partial \xi} \\ \dfrac{\partial \widehat{\psi}_i}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial \psi_i}{\partial x} \\ \dfrac{\partial \psi_i}{\partial y} \end{Bmatrix} = [J] \begin{Bmatrix} \dfrac{\partial \psi_i}{\partial x} \\ \dfrac{\partial \psi_i}{\partial y} \end{Bmatrix} \tag{216}$$

The Jacobian is inverted in order to express the partial derivatives of the interpolation functions with respect to the global coordinate system. The determinant of the Jacobian is $\|J\|$.

$$\begin{Bmatrix} \dfrac{\partial \psi_i}{\partial x} \\ \dfrac{\partial \psi_i}{\partial y} \end{Bmatrix} = \frac{1}{\|J\|} \begin{bmatrix} \dfrac{\partial y}{\partial \eta} & -\dfrac{\partial y}{\partial \xi} \\ -\dfrac{\partial x}{\partial \eta} & \dfrac{\partial x}{\partial \xi} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial \widehat{\psi}_i}{\partial \xi} \\ \dfrac{\partial \widehat{\psi}_i}{\partial \eta} \end{Bmatrix} \tag{217}$$

Equation (217) is substituted directly into the components of (210) and the change of variables method of integration is applied. The matrix, $\left[K_{14_{ij}}\right]$ and the vector $\{f_{1_i}\}$ are shown as examples.

$$J_{11} = \frac{\partial x}{\partial \xi} \qquad J_{12} = \frac{\partial y}{\partial \xi} \qquad J_{21} = \frac{\partial x}{\partial \eta} \qquad J_{22} = \frac{\partial y}{\partial \eta} \tag{218}$$

$$\frac{\partial \psi_i}{\partial x} = \frac{1}{\|J\|} \left(J_{22} \frac{\partial \widehat{\psi}_i}{\partial \xi} - J_{12} \frac{\partial \widehat{\psi}_i}{\partial \eta} \right) \tag{219}$$

$$\frac{\partial \psi_i}{\partial y} = \frac{1}{\|J\|} \left(J_{11} \frac{\partial \widehat{\psi}_i}{\partial \eta} - J_{21} \frac{\partial \widehat{\psi}_i}{\partial \xi} \right) \tag{220}$$

$$\left[K_{14_{ij}}\right] = \int_{-1}^{1}\int_{-1}^{1}\left\{\left[-\frac{1}{\rho^2}\frac{\partial P}{\partial x}\widehat{\psi}_i\right]\left[\frac{1}{\rho}\frac{1}{\|J\|}\left(J_{22}\frac{\partial\widehat{\psi}_j}{\partial\xi} - J_{12}\frac{\partial\widehat{\psi}_j}{\partial\eta}\right)\right] + \left[-\frac{1}{\rho^2}\frac{\partial P}{\partial y}\widehat{\psi}_i\right]\left[\frac{1}{\rho}\frac{1}{\|J\|}\left(J_{11}\frac{\partial\widehat{\psi}_j}{\partial\eta} - J_{21}\frac{\partial\widehat{\psi}_j}{\partial\xi}\right)\right]\right\}\|J\|\,\mathrm{d}\xi\mathrm{d}\eta \tag{221}$$

$$\begin{aligned}\{f_{1_i}\} = &\\ \int_{-1}^{1}\int_{-1}^{1}\Bigg\{&\left[\left(\frac{1}{\Delta t} + \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}\right)\widehat{\psi}_i + \frac{v_x}{\|J\|}\left(J_{22}\frac{\partial\widehat{\psi}_i}{\partial\xi} - J_{12}\frac{\partial\widehat{\psi}_i}{\partial\eta}\right) + \frac{v_y}{\|J\|}\left(J_{11}\frac{\partial\widehat{\psi}_i}{\partial\eta} - J_{21}\frac{\partial\widehat{\psi}_i}{\partial\xi}\right)\right]\times\\ &\left[\frac{1}{\Delta t}\rho + \frac{\partial v_x}{\partial x}\rho + \frac{\partial\rho}{\partial x}v_x + \frac{\partial v_y}{\partial y}\rho + \frac{\partial\rho}{\partial y}v_y\right] +\\ &\left[-\frac{1}{\rho^2}\frac{\partial P}{\partial x}\widehat{\psi}_i\right]\left[\frac{1}{\Delta t}v_x + \frac{\partial v_x}{\partial x}v_x + \frac{\partial v_x}{\partial y}v_y - \frac{1}{\rho}\frac{\partial P}{\partial x}\right] + \left[-\frac{1}{\rho^2}\frac{\partial P}{\partial y}\widehat{\psi}_i\right]\left[\frac{1}{\Delta t}v_y + \frac{\partial v_y}{\partial x}v_x + \frac{\partial v_y}{\partial y}v_y - \frac{1}{\rho}\frac{\partial P}{\partial y}\right]\Bigg\}\|J\|\,\mathrm{d}\xi\mathrm{d}\eta\end{aligned} \tag{222}$$

The Jacobian is also used to express the normal vector in terms of the local coordinate system.

$$\begin{Bmatrix} \hat{n}_1 \\ \hat{n}_2 \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \begin{Bmatrix} n_x \\ n_y \end{Bmatrix} \tag{223}$$

The Jacobian is inverted to express the normal vector in terms of the global coordinate system.

$$\begin{Bmatrix} n_x \\ n_y \end{Bmatrix} = \frac{1}{\|J\|} \begin{bmatrix} \dfrac{\partial y}{\partial \eta} & -\dfrac{\partial y}{\partial \xi} \\ -\dfrac{\partial x}{\partial \eta} & \dfrac{\partial x}{\partial \xi} \end{bmatrix} \begin{Bmatrix} \hat{n}_1 \\ \hat{n}_2 \end{Bmatrix} = \frac{1}{\|J\|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \begin{Bmatrix} \hat{n}_1 \\ \hat{n}_2 \end{Bmatrix} \tag{224}$$

Equation (224) is substituted into the components of (205). The components of Eq. (205) are shown for the bottom edge of an element. The integrals are evaluated at $\eta = -1$.

$$\begin{Bmatrix} n_x \\ n_y \end{Bmatrix} = \frac{1}{\|J\|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \begin{Bmatrix} 0 \\ -1 \end{Bmatrix} = \frac{1}{\|J\|} \begin{Bmatrix} J_{12} \\ -J_{11} \end{Bmatrix} \tag{225}$$

$$\left[Q_{xx_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{12}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{12}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\xi \tag{226}$$

$$\left[Q_{xy_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{12}}{\|J\|}\widehat{\psi}_i\right] \left[-\frac{J_{11}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\xi \tag{227}$$

$$\left[Q_{yx_{ij}}\right] = \int_{-1}^{1} \left[-\frac{J_{11}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{12}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\xi \tag{228}$$

$$\left[Q_{yy_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{11}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{11}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\xi \tag{229}$$

The components of Eq. (205) are shown for the top edge of an element. The integrals are evaluated at $\eta = 1$.

$$\begin{Bmatrix} n_x \\ n_y \end{Bmatrix} = \frac{1}{\|J\|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \begin{Bmatrix} 0 \\ 1 \end{Bmatrix} = \frac{1}{\|J\|} \begin{Bmatrix} -J_{12} \\ J_{11} \end{Bmatrix} \tag{230}$$

$$\left[Q_{xx_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{12}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{12}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\xi \tag{231}$$

$$\left[Q_{xy_{ij}}\right] = \int_{-1}^{1} \left[-\frac{J_{12}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{11}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\xi \tag{232}$$

$$\left[Q_{yx_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{11}}{\|J\|}\widehat{\psi}_i\right] \left[-\frac{J_{12}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\xi \tag{233}$$

$$\left[Q_{yy_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{11}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{11}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\xi \tag{234}$$

The components of Eq. (205) are shown for the left edge of an element. The integrals are evaluated at $\xi = -1$.

$$\begin{Bmatrix} n_x \\ n_y \end{Bmatrix} = \frac{1}{\|J\|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \begin{Bmatrix} -1 \\ 0 \end{Bmatrix} = \frac{1}{\|J\|} \begin{Bmatrix} -J_{22} \\ J_{21} \end{Bmatrix} \tag{235}$$

$$\left[Q_{xx_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{22}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{22}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\eta \tag{236}$$

$$\left[Q_{xy_{ij}}\right] = \int_{-1}^{1} \left[-\frac{J_{22}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{21}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\eta \tag{237}$$

$$\left[Q_{yx_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{21}}{\|J\|}\widehat{\psi}_i\right] \left[-\frac{J_{22}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\eta \tag{238}$$

$$\left[Q_{yy_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{21}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{21}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\eta \tag{239}$$

The components of Eq. (205) are shown for the right edge of an element. The integrals are evaluated at $\xi = 1$.

$$\begin{Bmatrix} n_x \\ n_y \end{Bmatrix} = \frac{1}{\|J\|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \begin{Bmatrix} 1 \\ 0 \end{Bmatrix} = \frac{1}{\|J\|} \begin{Bmatrix} J_{22} \\ -J_{21} \end{Bmatrix} \tag{240}$$

$$\left[Q_{xx_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{22}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{22}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\eta \tag{241}$$

$$\left[Q_{xy_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{22}}{\|J\|}\widehat{\psi}_i\right] \left[-\frac{J_{21}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\eta \tag{242}$$

$$\left[Q_{yx_{ij}}\right] = \int_{-1}^{1} \left[-\frac{J_{21}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{22}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\eta \tag{243}$$

$$\left[Q_{yy_{ij}}\right] = \int_{-1}^{1} \left[\frac{J_{21}}{\|J\|}\widehat{\psi}_i\right] \left[\frac{J_{21}}{\|J\|}\widehat{\psi}_j\right] \|J\| \, \mathrm{d}\eta \tag{244}$$

4.5 Mesh Adaptation

As described in Section 2.2, a suitable set of exponential parameters was only found for cases where a discontinuity is located at the edge of an element. Therefore a mesh adaptation scheme is necessary to align element edges with shock waves. The method described here was written by Ait-Ali-Yahia et al.,[9] with the exception of a few minor changes. A preliminary step is calculating the second derivative of a dependent variable, which will be used to estimate the error along the element edges. Density, velocity, pressure, or any other variable that experiences a discontinuity at a shock wave would be a suitable selection. In this description, σ is used to represent the chosen dependent variable and it is approximated by Eq. (245).

$$\sigma \approx \sum_{j=1}^{N} \psi_j \sigma_j \tag{245}$$

$$\frac{\partial^2 \sigma}{\partial x^2} \approx \sum_{j=1}^{N} \frac{\partial^2 \psi_j}{\partial x^2} \sigma_j \tag{246}$$

$$\frac{\partial^2 \sigma}{\partial x \partial y} \approx \sum_{j=1}^{N} \frac{\partial^2 \psi_j}{\partial x \partial y} \sigma_j \tag{247}$$

$$\frac{\partial^2 \sigma}{\partial y^2} \approx \sum_{j=1}^{N} \frac{\partial^2 \psi_j}{\partial y^2} \sigma_j \tag{248}$$

In Reference 9, the second derivative is estimated using a weak formulation and mass lumping. Here, the second order chain rule is used. Application of the formulas in Reference 10 yields Eq. (249). First and second derivatives of the global coordinates with respect to the local coordinates can be calculated by differentiating the shape functions in Eqs. (211) and (212). The first derivatives of ψ_j with respect to the global coordinates are calculated using Eq. (217). The only unknown variables in Eq. (249) are the second derivatives of ψ_j, which can be calculated by inverting the equation, resulting in Eq. (252).

$$\begin{Bmatrix} \dfrac{\partial^2 \widehat{\psi}_j}{\partial \xi^2} \\ \dfrac{\partial^2 \widehat{\psi}_j}{\partial \eta^2} \\ \dfrac{\partial^2 \widehat{\psi}_j}{\partial \xi \partial \eta} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi}\dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi}\dfrac{\partial y}{\partial \xi} & 2\dfrac{\partial x}{\partial \xi}\dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta}\dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta}\dfrac{\partial y}{\partial \eta} & 2\dfrac{\partial x}{\partial \eta}\dfrac{\partial y}{\partial \eta} \\ \dfrac{\partial x}{\partial \xi}\dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \xi}\dfrac{\partial y}{\partial \eta} & \dfrac{\partial x}{\partial \xi}\dfrac{\partial y}{\partial \eta} + \dfrac{\partial y}{\partial \xi}\dfrac{\partial x}{\partial \eta} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial^2 \psi_j}{\partial x^2} \\ \dfrac{\partial^2 \psi_j}{\partial y^2} \\ \dfrac{\partial^2 \psi_j}{\partial x \partial y} \end{Bmatrix} + \begin{bmatrix} \dfrac{\partial^2 x}{\partial \xi^2} & \dfrac{\partial^2 y}{\partial \xi^2} \\ \dfrac{\partial^2 x}{\partial \eta^2} & \dfrac{\partial^2 y}{\partial \eta^2} \\ \dfrac{\partial^2 x}{\partial \xi \partial \eta} & \dfrac{\partial^2 y}{\partial \xi \partial \eta} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial \psi_j}{\partial x} \\ \dfrac{\partial \psi_j}{\partial y} \end{Bmatrix} \tag{249}$$

$$\begin{bmatrix} \mathcal{H}_x^{11} & \mathcal{H}_x^{12} \\ \mathcal{H}_x^{21} & \mathcal{H}_x^{22} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial^2 x}{\partial \xi^2} & \dfrac{\partial^2 x}{\partial \xi \partial \eta} \\ \dfrac{\partial^2 x}{\partial \eta \partial \xi} & \dfrac{\partial^2 x}{\partial \eta^2} \end{bmatrix} \tag{250}$$

$$\begin{bmatrix} \mathcal{H}_y^{11} & \mathcal{H}_y^{12} \\ \mathcal{H}_y^{21} & \mathcal{H}_y^{22} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial^2 y}{\partial \xi^2} & \dfrac{\partial^2 y}{\partial \xi \partial \eta} \\ \dfrac{\partial^2 y}{\partial \eta \partial \xi} & \dfrac{\partial^2 y}{\partial \eta^2} \end{bmatrix} \tag{251}$$

$$\begin{Bmatrix} \dfrac{\partial^2 \psi_j}{\partial x^2} \\ \dfrac{\partial^2 \psi_j}{\partial y^2} \\ \dfrac{\partial^2 \psi_j}{\partial x \partial y} \end{Bmatrix} = \frac{1}{\|J\|} \begin{bmatrix} J_{22}^2 & J_{12}^2 & -2J_{12}J_{22} \\ J_{21}^2 & J_{11}^2 & -2J_{11}J_{21} \\ -J_{21}J_{22} & -J_{11}J_{12} & J_{11}J_{22} + J_{12}J_{21} \end{bmatrix} \left\{ \begin{Bmatrix} \dfrac{\partial^2 \widehat{\psi}_j}{\partial \xi^2} \\ \dfrac{\partial^2 \widehat{\psi}_j}{\partial \eta^2} \\ \dfrac{\partial^2 \widehat{\psi}_j}{\partial \xi \partial \eta} \end{Bmatrix} - \begin{bmatrix} \mathcal{H}_x^{11} & \mathcal{H}_y^{11} \\ \mathcal{H}_x^{22} & \mathcal{H}_y^{22} \\ \mathcal{H}_x^{12} & \mathcal{H}_y^{12} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial \psi_j}{\partial x} \\ \dfrac{\partial \psi_j}{\partial y} \end{Bmatrix} \right\} \tag{252}$$

The Hessian of σ is reconstructed using the absolute values of its eigenvalues. The eigenvalues are represented by λ, the eigenvectors are $\{\kappa\}$, and the unit eigenvectors are $\{\widehat{\kappa}\}$.

$$\begin{bmatrix} \dfrac{\partial^2\sigma}{\partial x^2} & \dfrac{\partial^2\sigma}{\partial x\partial y} \\ \dfrac{\partial^2\sigma}{\partial y\partial x} & \dfrac{\partial^2\sigma}{\partial y^2} \end{bmatrix} = \begin{bmatrix} \mathcal{H}_\sigma^{11} & \mathcal{H}_\sigma^{12} \\ \mathcal{H}_\sigma^{21} & \mathcal{H}_\sigma^{22} \end{bmatrix} \tag{253}$$

$$\{\kappa_1\} = \begin{Bmatrix} \mathcal{H}_\sigma^{11} - \mathcal{H}_\sigma^{22} - \sqrt{\mathcal{H}_\sigma^{11}\mathcal{H}_\sigma^{11} - 2\mathcal{H}_\sigma^{11}\mathcal{H}_\sigma^{22} + 4\mathcal{H}_\sigma^{12}\mathcal{H}_\sigma^{12} + \mathcal{H}_\sigma^{22}\mathcal{H}_\sigma^{22}} \\ 2\mathcal{H}_\sigma^{12} \end{Bmatrix} \tag{254}$$

$$\{\kappa_2\} = \begin{Bmatrix} \mathcal{H}_\sigma^{11} - \mathcal{H}_\sigma^{22} + \sqrt{\mathcal{H}_\sigma^{11}\mathcal{H}_\sigma^{11} - 2\mathcal{H}_\sigma^{11}\mathcal{H}_\sigma^{22} + 4\mathcal{H}_\sigma^{12}\mathcal{H}_\sigma^{12} + \mathcal{H}_\sigma^{22}\mathcal{H}_\sigma^{22}} \\ 2\mathcal{H}_\sigma^{12} \end{Bmatrix} \tag{255}$$

$$\lambda_1 = \frac{1}{2}\left(\mathcal{H}_\sigma^{11} + \mathcal{H}_\sigma^{22} - \sqrt{\mathcal{H}_\sigma^{11}\mathcal{H}_\sigma^{11} - 2\mathcal{H}_\sigma^{11}\mathcal{H}_\sigma^{22} + 4\mathcal{H}_\sigma^{12}\mathcal{H}_\sigma^{12} + \mathcal{H}_\sigma^{22}\mathcal{H}_\sigma^{22}}\right) \tag{256}$$

$$\lambda_2 = \frac{1}{2}\left(\mathcal{H}_\sigma^{11} + \mathcal{H}_\sigma^{22} + \sqrt{\mathcal{H}_\sigma^{11}\mathcal{H}_\sigma^{11} - 2\mathcal{H}_\sigma^{11}\mathcal{H}_\sigma^{22} + 4\mathcal{H}_\sigma^{12}\mathcal{H}_\sigma^{12} + \mathcal{H}_\sigma^{22}\mathcal{H}_\sigma^{22}}\right) \tag{257}$$

$$\begin{bmatrix} \bar{\mathcal{H}}_\sigma^{11} & \bar{\mathcal{H}}_\sigma^{12} \\ \bar{\mathcal{H}}_\sigma^{21} & \bar{\mathcal{H}}_\sigma^{22} \end{bmatrix} = \begin{Big\{} \{\widehat{\kappa}_1\} & \{\widehat{\kappa}_2\} \end{Big\} \begin{bmatrix} |\lambda_1| & 0 \\ 0 & |\lambda_2| \end{bmatrix} \begin{Bmatrix} \{\widehat{\kappa}_1^T\} \\ \{\widehat{\kappa}_2^T\} \end{Bmatrix} \tag{258}$$

To explain how the mesh modification method works the element edges sharing a vertex node can be thought of as an assembly of springs.[9] The vertex being moved is assigned the index i and every vertex that shares an edge with vertex i is assigned the index j. The spring potential is minimized by moving node i. The stiffness of each spring is an error estimate along the edge represented by the spring divided by the length, L, of the edge. The spring stiffness, Eq. (259), is calculated for each element edge between node i and nodes j. For the top and bottom edges of an element (along $\eta_2 = \pm 1$), $\Gamma_x = J_{11}$ and $\Gamma_y = J_{12}$. For the left and right edges of an element (along $\eta_1 = \pm 1$), $\Gamma_x = J_{21}$ and $\Gamma_y = J_{22}$.

$$k_{ij} = \frac{1}{L}\int_\Gamma \sqrt{\Gamma_x^2\bar{\mathcal{H}}_\sigma^{11} + 2\Gamma_x\Gamma_y\bar{\mathcal{H}}_\sigma^{12} + \Gamma_y^2\bar{\mathcal{H}}_\sigma^{22}}\,\mathrm{d}x\mathrm{d}y \tag{259}$$

The spring potential is minimized by taking the first derivatives of Eq. (260) with respect to the global coordinates of node i and setting them equal to 0. Although the spring stiffness is a function of (x_i, y_i), it is treated as a constant.

$$\mathcal{P} = \sum_j \left[(x_i - x_J)^2 + (y_i - y_j)^2 \right] k_{ij} \tag{260}$$

$$\frac{\partial \mathcal{P}}{\partial x_i} = \sum_j (x_i - x_J)\, k_{ij} \tag{261}$$

$$\frac{\partial \mathcal{P}}{\partial y_i} = \sum_j (y_i - y_J)\, k_{ij} \tag{262}$$

Newton's method is applied to find the new global coordinates, (x_i, y_i), that satisfy Eqs. (261) and (262). Iterations of Newton's method are noted with the index s and ω is a relaxation parameter.

$$\frac{\partial^2 \mathcal{P}}{\partial x_i^2} \Delta x_i = -\frac{\partial \mathcal{P}}{\partial x_i} \tag{263}$$

$$\frac{\partial^2 \mathcal{P}}{\partial y_i^2} \Delta y_i = -\frac{\partial \mathcal{P}}{\partial y_i} \tag{264}$$

$$\Delta x_i = \frac{\sum_j \left(x_j^s - x_i^s \right) k_{ij}}{\sum_j k_{ij}} \tag{265}$$

$$\Delta y_i = \frac{\sum_j \left(y_j^s - y_i^s \right) k_{ij}}{\sum_j k_{ij}} \tag{266}$$

$$x_i^{s+1} = x_i^s + \omega \Delta x_i \tag{267}$$

$$y_i^{s+1} = y_i^s + \omega \Delta y_i \tag{268}$$

The mesh modification scheme is only applied to element vertices and only one vertex is updated at a time. Once a new vertex position is calculated, the Jacobians of the surrounding elements are calculated to verify mesh quality. If the resulting elements are too skewed or the determinant of the Jacobian of any element is negative, the relaxation

parameter is decreased and Eqs. (267) and (268) are used to calculate a new vertex position. After 50 iterations, if desirable elements are not created, the new vertex position is discarded and the algorithm moves on to the next vertex. If the new vertex position is kept, the coordinates of the interior nodes are generated and the nodal values of σ are calculated by interpolating on the original, unmodified mesh.

To ensure that vertices do not drift away from shock waves a constraint was imposed on the sum of the absolute value of the components of the gradient of σ at the new vertex coordinates. For simplicity, this constraint is referred to as the absolute gradient constraint and the absolute gradient is calculated using Eq. (269). The absolute gradient constraint is imposed by calculating $\|\nabla\sigma\|$ at (x_i^s, y_i^s) and $\left(x_i^{s+1}, y_i^{s+1}\right)$. If $\|\nabla\sigma\|$ at $\left(x_i^{s+1}, y_i^{s+1}\right)$ is less than $\|\nabla\sigma\|$ at (x_i^s, y_i^s), the relaxation parameter is decreased and the new vertex position is recalculated. The absolute gradient constraint is imposed in the same loop as the calculation of the Jacobian.

$$\|\nabla\sigma\| = \left|\frac{\partial\sigma}{\partial x}\right| + \left|\frac{\partial\sigma}{\partial y}\right| \tag{269}$$

Another change was made to the mesh modification scheme that improved performance. The components of the Hessian in Eq. (259), were squared. That change was applied to all elements but it gave the edges of exponential elements much greater stiffness than the edges of polynomial elements. This change improved mesh adaptation in the region of the shock reflection for both polynomial and exponential interpolation.

The mesh modification scheme yields better meshes if, for vertices on the edges of the domain, only the stiffness due to adjacent vertices on the edge of the domain are used to update (x_i, y_i). For vertices on a horizontal boundary, only the x-coordinate is updated. For vertices on a vertical boundary, only the y-coordinate is updated. For problems with vertices on boundaries that are not parallel to the global axes, the new x-coordinate could be calculated using the mesh modification scheme and then the new y-coordinate could be calculated using some known function of x that defines the shape of the boundary, or vice versa.

CHAPTER 5
SHOCK REFLECTION EXAMPLE

In this chapter, an oblique shock reflection example is presented. First, in Section 5.1 the exact solution is calculated. The exact solution is used in subsequent sections as a benchmark to evaluate the quality of numerical solutions calculated using polynomial and exponential interpolation. In Section 5.2, the shock reflection problem is solved on a uniform bilinear mesh, which serves to ensure that functions written for this paper give results that match previous work.[1,4] The numerical solution is compared to the exact solution to show areas where the finite element method performs poorly. In Section 5.3, polynomial and exponential interpolation functions are compared on a shock-aligned mesh. The shock reflection example is solved using the mesh modification scheme and both polynomial and exponential interpolation in Section 5.4.

5.1 Analytical Solution

This example induces an oblique shock in a supersonic flow using the essential boundary conditions on the left and top edges of the domain. The bottom edge of the domain is a solid wall. The first oblique shock is reflected off the wall and forms a second oblique shock. The oblique shock and normal shock relations used here are taken from Reference 11. Figure 11 shows the domain and shock locations for this example. The domain is subdivided into three parts; part 0 is upstream of the first shock, part 1 is downstream of the first shock and upstream of the reflected shock, and part 2 is downstream of the reflected shock.

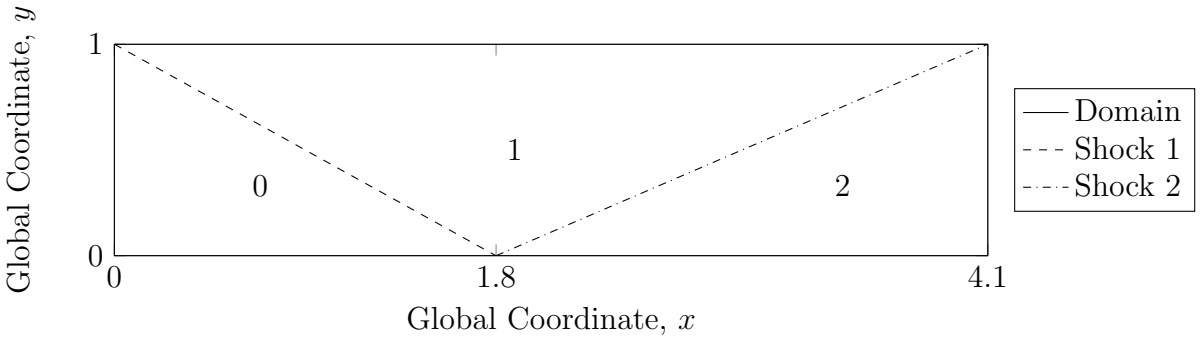

Figure 11. Domain and shock wave locations.

The free stream properties at the left edge of the domain are selected to obtain a Mach

number of 2.9. The flow properties are constant throughout region 0.

$$\begin{aligned}
\text{Speed of sound:} \quad & a_0 = 1 \\
\text{Density:} \quad & \rho_0 = 1 \\
\text{Velocity, } x\text{-component:} \quad & v_{x_0} = 2.9 \\
\text{Velocity, } y\text{-component:} \quad & v_{y_0} = 0 \\
\text{Pressure:} \quad & P_0 = \frac{\rho_0 a_0^2}{\gamma} = 0.7143 \\
\text{Mach number:} \quad & M_0 = \frac{\sqrt{v_{x_0}^2 + v_{y_0}^2}}{a_0} = 2.9
\end{aligned} \tag{270}$$

The velocity components in region 1 are selected to make the acute angle between shock 1 and the x-axis (β_1) 29 degrees. The acute angle between the velocity vector and the x-axis is θ_1.

$$\theta_1 = \tan^{-1}\left\{2\cot\beta_1\left[\frac{M_0^2\sin^2\beta_1 - 1}{M_0^2\left(\gamma + \cos 2\beta_1\right) + 2}\right]\right\} = 10.9404^\circ \tag{271}$$

The Mach numbers of the velocity components normal to shock 1 are calculated for region 0 ($M_{n_{01}}$) and region 1 ($M_{n_{11}}$).

$$M_{n_{01}} = M_0 \sin\beta_1 = 1.4059 \tag{272}$$

$$M_{n_{11}} = \left[\frac{1 + \dfrac{\gamma - 1}{2}M_{n_{01}}^2}{\gamma M_{n_{01}}^2 - \dfrac{\gamma - 1}{2}}\right]^{\frac{1}{2}} = 0.7372 \tag{273}$$

The flow properties in region 1 are calculated using normal shock equations.

$$\rho_1 = \rho_0\frac{\left(\gamma + 1\right)M_{n_{01}}^2}{2 + \left(\gamma - 1\right)M_{n_{01}}^2} = 1.7000 \tag{274}$$

$$P_1 = P_0\left[1 + \frac{2\gamma}{\gamma + 1}\left(M_{n_{01}}^2 - 1\right)\right] = 1.5282 \tag{275}$$

$$a_1 = \left[\frac{\gamma P_1}{\rho_1}\right]^{\frac{1}{2}} = 1.1218 \tag{276}$$

$$M_1 = \frac{M_{n_{11}}}{\sin(\beta_1 - \theta_1)} = 2.3781 \tag{277}$$

$$v_{x_1} = M_1 a_1 \cos\theta_1 = 2.6193 \tag{278}$$

$$v_{y_1} = -M_1 a_1 \sin\theta_1 = -0.5063 \tag{279}$$

In region 2, the flow is deflected by the solid wall that forms the bottom edge of the domain. The flow deflection angle in region 2 is equal in magnitude to the flow deflection angle in region 1 ($\theta_2 = \theta_1$). The acute angle between the velocity vector in region 1 and shock wave 2 (β_2) is calculated by solving Eq. (280) with the `fzero` function in MATLAB.

$$2\cot\beta_2 \left[\frac{M_1^2 \sin^2\beta_2 - 1}{M_1^2(\gamma + \cos 2\beta_2) + 2}\right] - \tan\theta_2 = 0 \tag{280}$$

$$\beta_2 = 34.2195^\circ \tag{281}$$

The Mach numbers of the velocity components normal to shock 2 are calculated for region 1 ($M_{n_{12}}$) and region 2 ($M_{n_{22}}$).

$$M_{n_{12}} = M_1 \sin\beta_2 = 1.3373 \tag{282}$$

$$M_{n_{22}} = \left[\frac{1 + \dfrac{\gamma - 1}{2} M_{n_{12}}^2}{\gamma M_{n_{12}}^2 - \dfrac{\gamma - 1}{2}}\right]^{\frac{1}{2}} = 0.7677 \tag{283}$$

The flow properties in region 2 are calculated using normal shock equations.

$$\rho_2 = \rho_1 \frac{(\gamma + 1) M_{n_{12}}^2}{2 + (\gamma - 1) M_{n_{12}}^2} = 2.6872 \tag{284}$$

$$P_2 = P_1 \left[1 + \frac{2\gamma}{\gamma + 1}\left(M_{n_{12}}^2 - 1\right)\right] = 2.934 \tag{285}$$

$$a_2 = \left[\frac{\gamma P_2}{\rho_2}\right]^{\frac{1}{2}} = 1.2363 \tag{286}$$

$$M_2 = \frac{M_{n_{22}}}{\sin(\beta_2 - \theta_2)} = 1.9424 \tag{287}$$

$$v_{x_2} = M_2 a_2 \cos(\theta_2 - \theta_1) = 2.4015 \tag{288}$$

$$v_{y_2} = M_2 a_2 \sin(\theta_2 - \theta_1) = 0 \tag{289}$$

5.2 Solution Using Uniform Grid and Polynomial Interpolation

The reflected shock example was solved numerically using a mesh composed of 1200 bilinear rectangular elements; 60 elements along the x-direction and 20 along the y-direction. The mesh is shown in Figure 12. Lagrange polynomial interpolation functions are used to approximate the dependent variables. The flow properties in region 0 of the analytical solution are used as essential boundary conditions on the left edge of the domain. The flow properties in region 1 of the analytical solution are used as the essentail boundary conditions on the top edge of the domain. The bottom edge is a solid boundary which is implemented by imposing $v_y = 0$ as an essential boundary condition. The time step is 0.05. The solution, shown in Figures 13 and 14, matches the results obtained by Taghaddosi et al.[1] and Potanza et al.[4]

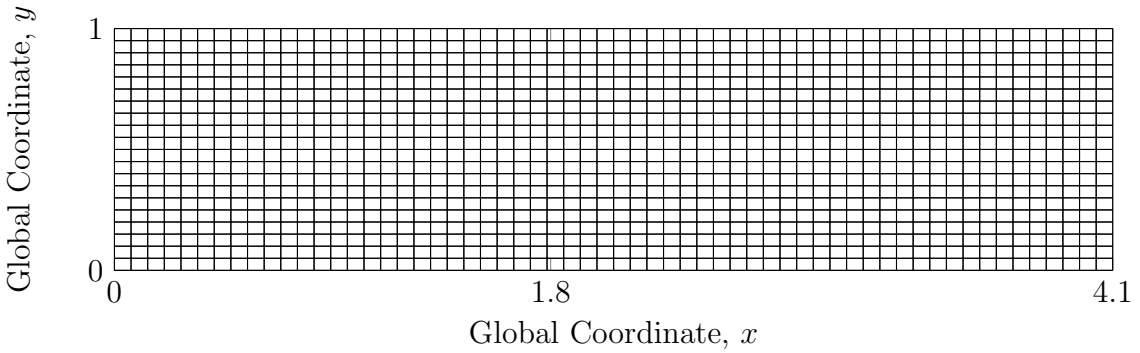

Figure 12. Uniform bilinear mesh.

Some noteworthy features of the numerical solution are the shock positions. Both shock waves form downstream of their analytical positions and the flow properties exhibit unnatural fluctuations immediately upstream and downstream of the shock waves. As noted in References 1 and 4, the shock waves are smeared.

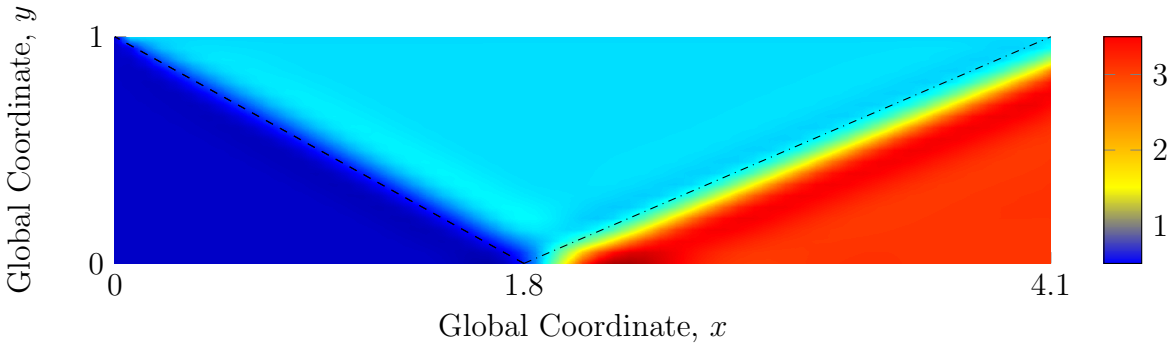

Figure 13. Pressure calculated using uniform mesh, polynomial interpolation, and $\Delta t = 0.05$ with dashed lines showing the analytical shock positions.

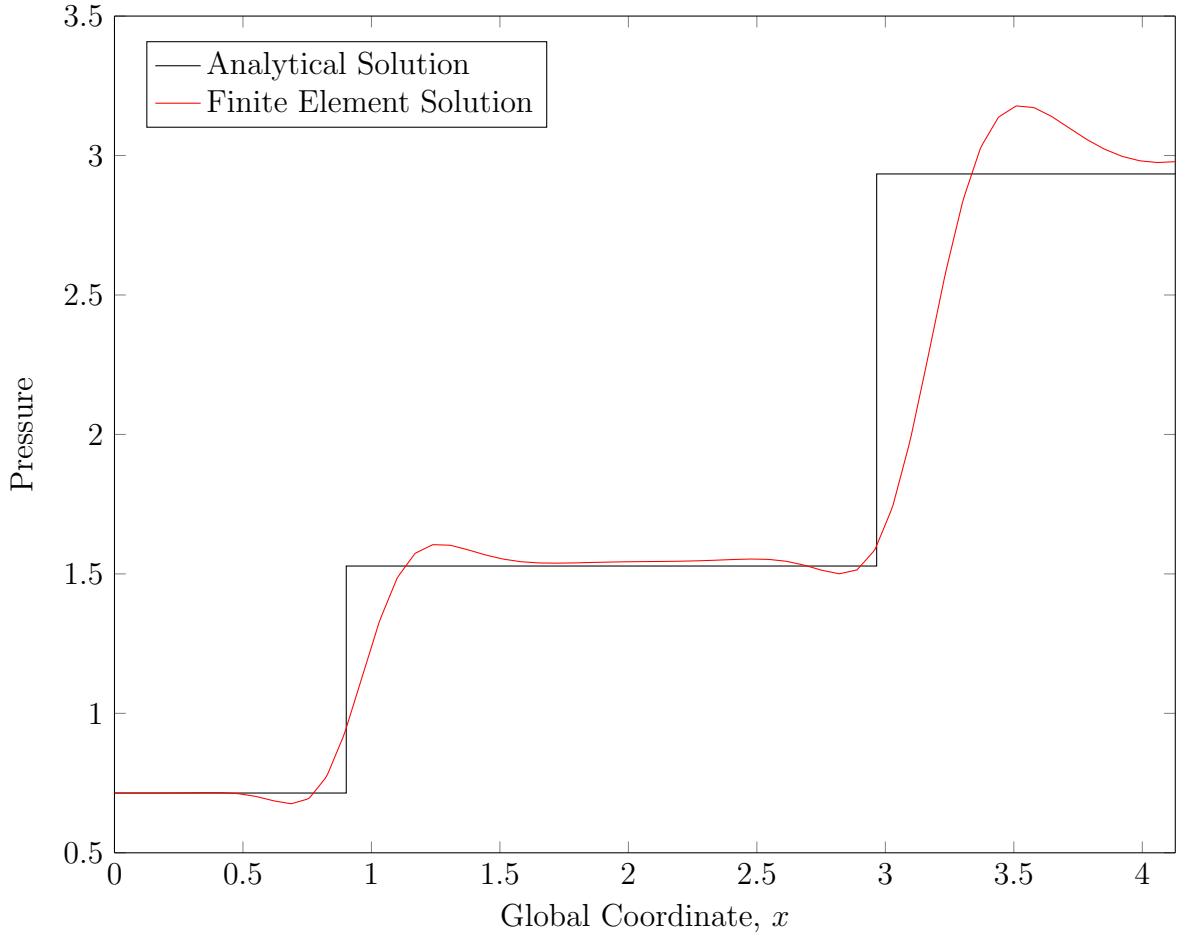

Figure 14. Pressure vs x at $y = 0.5$.

5.3 Comparison of Interpolation Functions on a Shock-Aligned Mesh

An initial investigation into the utility of exponential interpolation functions was conducted by solving the reflected shock problem on a mesh with element edges coincident with the analytical shock position. The mesh, shown in Figure 15, is composed of bicubic elements. The solution was calculated for four different selections for the interpolation functions in the elements adjacent to the shock waves; Lagrange polynomials both upstream and downstream, exponential functions upstream and Lagrange polynomials downstream, Lagrange polynomials upstream and exponential functions downstream, and exponential functions both upstream and downstream. Additionally, solutions were calculated using two time steps, $\Delta t = 0.05$ and $\Delta t = 0.01$. This approach to the shock reflection problem is a full factorial unreplicated experiment. Since there are no random variables in any of the calculations, replicates are not necessary because the same input would give the same output. The factors and the level of each factor are listed in Table 1. The response is the solution error calculated using Eq. (290). In Eq. (290), σ is a fluid property calculated using the finite element method, $\bar{\sigma}$ is the fluid property from the analytical solution, and Ω_j is the area of zone j in the analytical solution. The error calculation was repeated for each fluid property. To simplify notation, V is used to represent the magnitude of the velocity vector.

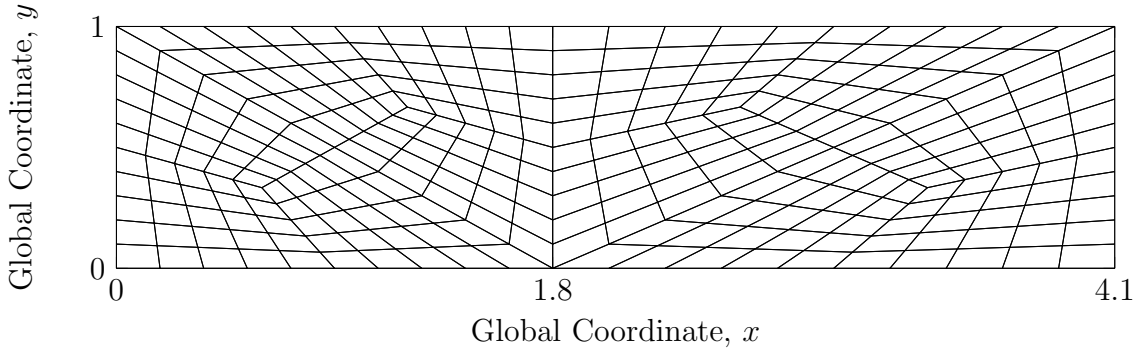

Figure 15. Shock-aligned bicubic mesh.

$$E_\sigma = \frac{\left|\sum_{j=0}^{2}\left(\int \sigma_j \mathrm{d}\Omega_j\right) - \sum_{j=0}^{2}\left(\bar{\sigma}_j \Omega_j\right)\right|}{\sum_{j=0}^{2}\left(\bar{\sigma}_j \Omega_j\right)} \tag{290}$$

Table 1. Factors and levels used to compare interpolation functions.

Factor	Levels
Δt	0.01
	0.05
Upstream Interpolation (Up Int.)	Exponential (Exp.)
	Polynomial (Poly.)
Downstream Interpolation (Dn. Int.)	Exponential (Exp.)
	Polynomial (Poly.)

The factor levels and error are shown in Tables 2 to 4. It would have been possible to include the flow property as a factor but the levels of that factor would not be independent. For example, pressure cannot be calculated without also calculating density and velocity. Therefore the error for each flow property was analyzed separately.

Table 2. Factor combinations and error for density.

	Interpolation		
Δt	Upstream	Downstream	$\log_{10}(E_\rho)$
0.01	Poly.	Exp.	-3.6176
0.05	Poly.	Exp.	-2.7484
0.01	Exp.	Poly.	-3.6765
0.05	Exp.	Poly.	-2.1754
0.01	Exp.	Exp.	-2.6942
0.05	Exp.	Exp.	-2.4246
0.01	Poly.	Poly.	-2.4917
0.05	Poly.	Poly.	-2.3114

Table 3. Factor combinations and error for magnitude of velocity.

	Interpolation		
Δt	Upstream	Downstream	$\log_{10}(E_V)$
0.01	Poly.	Exp.	-3.3313
0.05	Poly.	Exp.	-3.7894
0.01	Exp.	Poly.	-3.0722
0.05	Exp.	Poly.	-3.1170
0.01	Exp.	Exp.	-3.4672
0.05	Exp.	Exp.	-3.3591
0.01	Poly.	Poly.	-3.0883
0.05	Poly.	Poly.	-2.6991

Table 4. Factor combinations and error for pressure.

	Interpolation		
Δt	Upstream	Downstream	$\log_{10}(E_P)$
0.01	Poly.	Exp.	-2.3162
0.05	Poly.	Exp.	-2.3117
0.01	Exp.	Poly.	-2.9206
0.05	Exp.	Poly.	-2.2087
0.01	Exp.	Exp.	-2.4608
0.05	Exp.	Exp.	-2.4134
0.01	Poly.	Poly.	-2.3128
0.05	Poly.	Poly.	-2.0602

Table 5 shows analysis of variance results for the velocity magnitude. The analysis was performed using a type III sum of squares and a 0.05 significance level. The downstream interpolation function was the only significant factor. Exponential interpolation downstream of the shocks resulted in significantly less error than polynomial interpolation. Analysis of variance for density and pressure showed that none of the factors had a significant effect on the error.

Table 5. Significant factors for magnitude of velocity.

Source	Sum of Squares	Degrees of Freedom	Mean Square	F-Value	P-Value
Dn. Int.	0.4854	1	0.4854	11.67	0.0142
Error	0.2495	6	0.0416		
Total	0.7349	7			

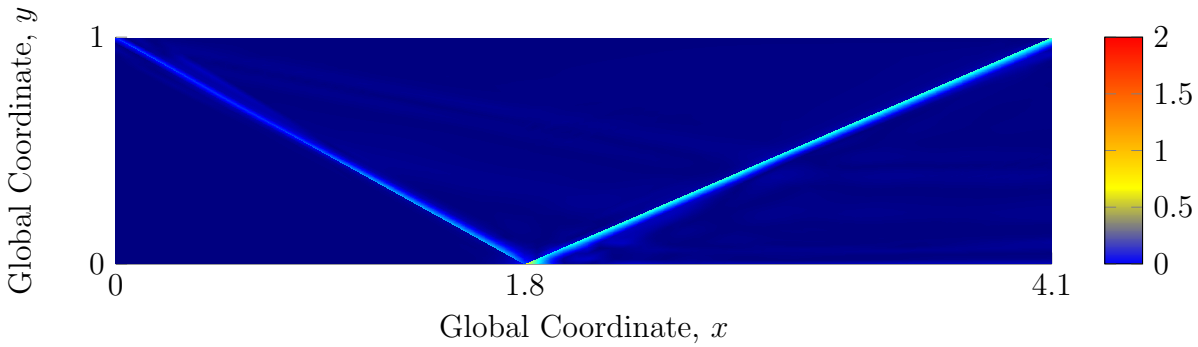

Figure 16. $|\rho - \bar{\rho}|$ **using** $\Delta t = 0.05$ **and polynomial interpolation.**

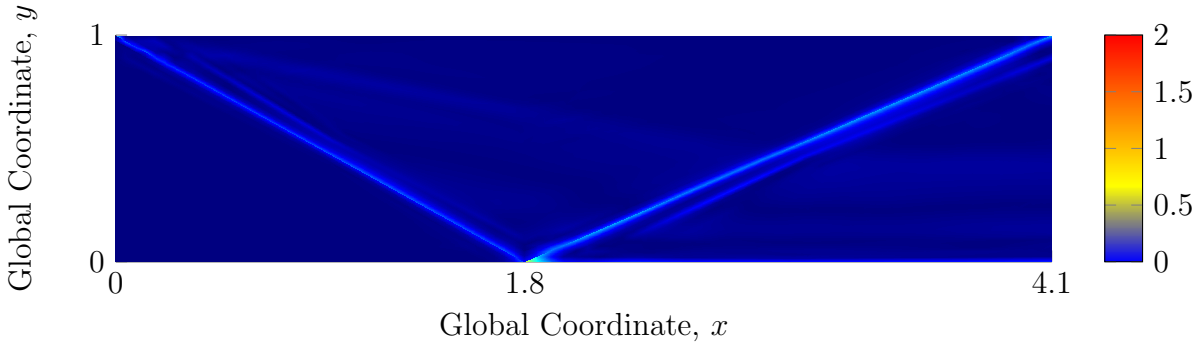

Figure 17. $|\rho - \bar{\rho}|$ **using** $\Delta t = 0.05$ **and exponential interpolation downstream of the shock waves.**

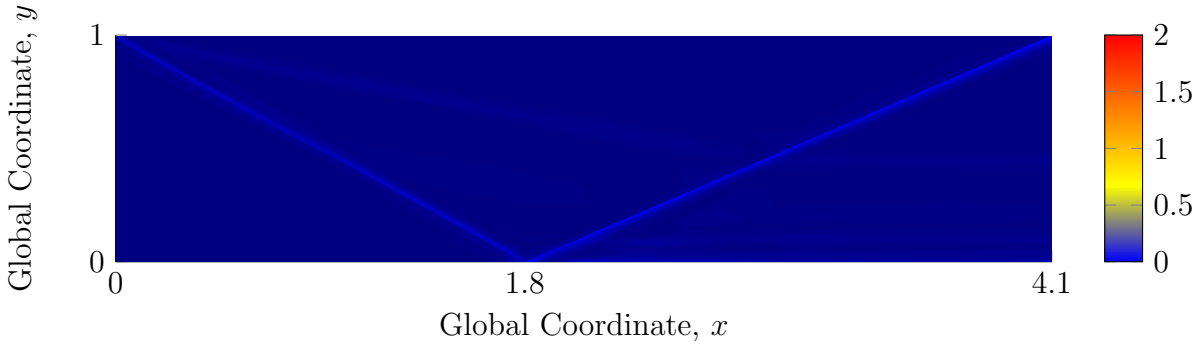

Figure 18. $|V - \bar{V}|$ **using** $\Delta t = 0.05$ **and polynomial interpolation.**

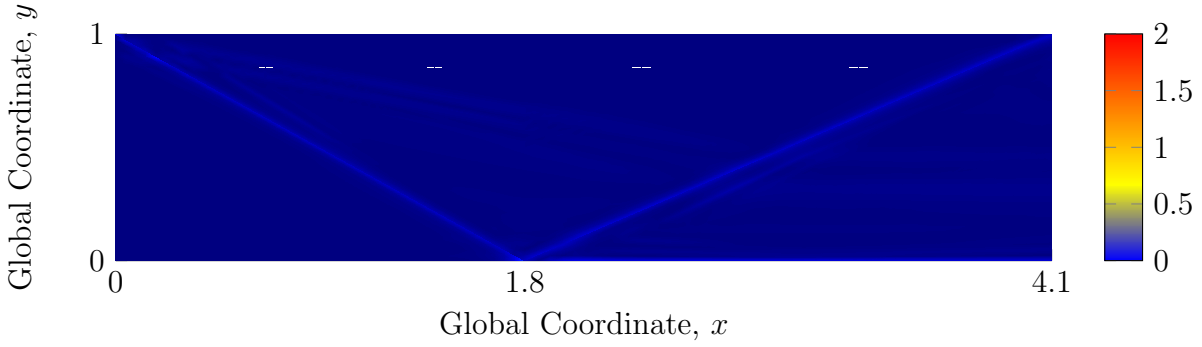

Figure 19. $|V - \bar{V}|$ **using** $\Delta t = 0.05$ **and exponential interpolation downstream of the shock waves.**

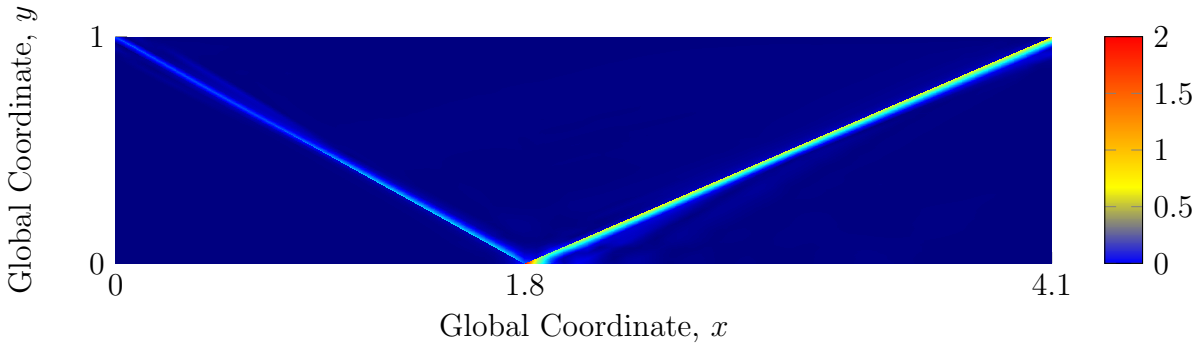

Figure 20. $|P - \bar{P}|$ **using** $\Delta t = 0.05$ **and polynomial interpolation.**

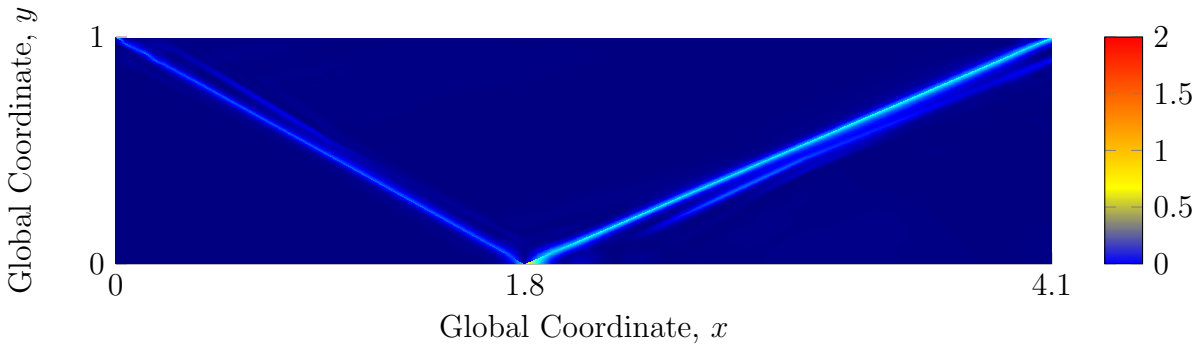

Figure 21. $|P - \bar{P}|$ **using** $\Delta t = 0.05$ **and exponential interpolation downstream of the shock waves.**

Figures 16 to 21 show the absolute value of the difference between the analytical solution and the finite element solutions calculated using polynomial interpolation and downstream exponential interpolation. Although the analysis of variance shows that exponential interpolation functions downstream of a shock reduce the error in velocity, there appears to be very little qualitative difference due to the choice of interpolation function. Figures 22 to 27 show all of the flow properties along $y = 0.5$. Exponential interpolation functions did not reduce oscillations and the shock waves are still smeared no matter which interpolation functions are used. For all choices of interpolation function on the shock-aligned mesh, the shock waves are much closer to their analytical positions when compared to the uniform mesh solution in the previous section. In Figures 23, 25, and 27 the reflected shock appears to be slightly closer to its exact position when exponential interpolation is used downstream of the shocks. Still, the choice of interpolation function does not appear to have a significant qualitative effect on the shock wave positions.

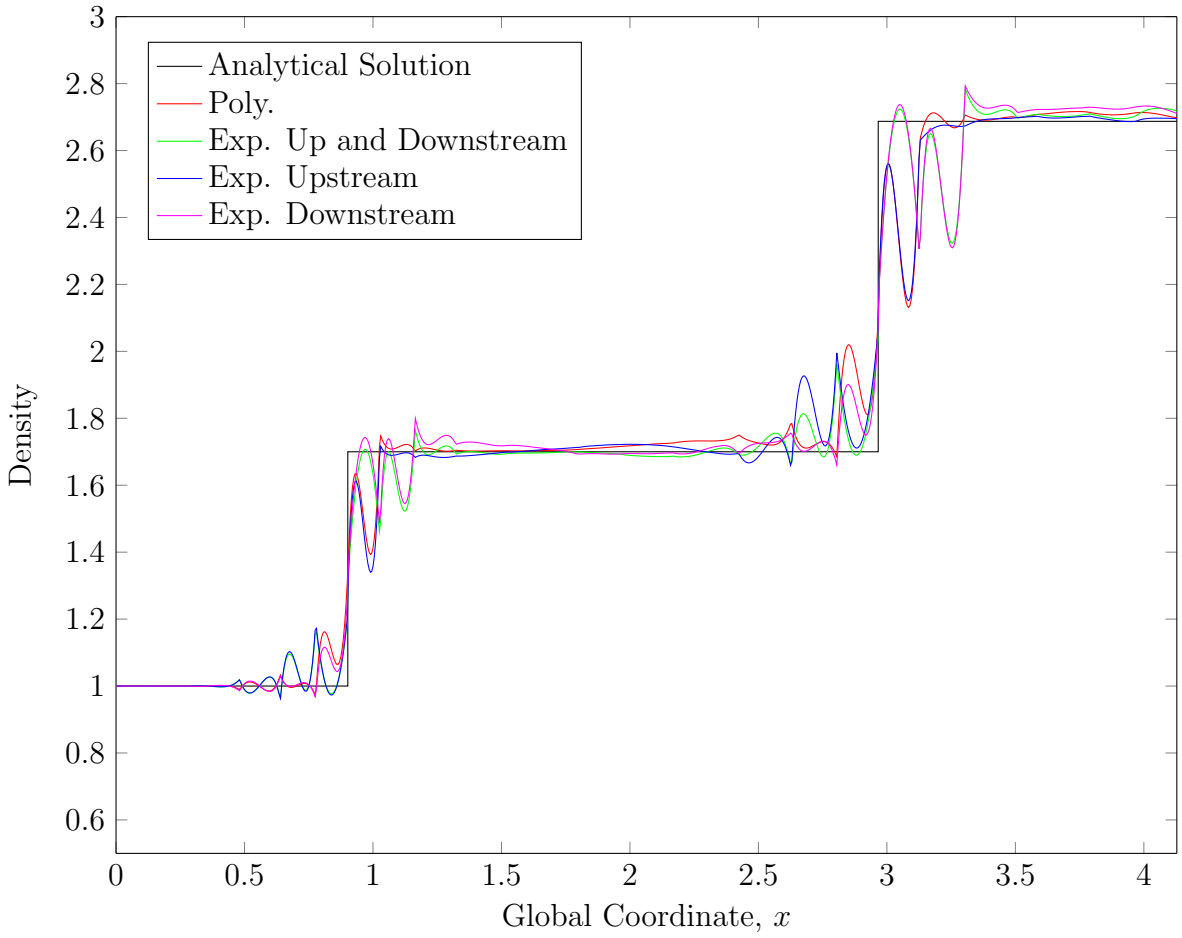

Figure 22. Density vs. x at $y = 0.5$, $\Delta t = 0.01$.

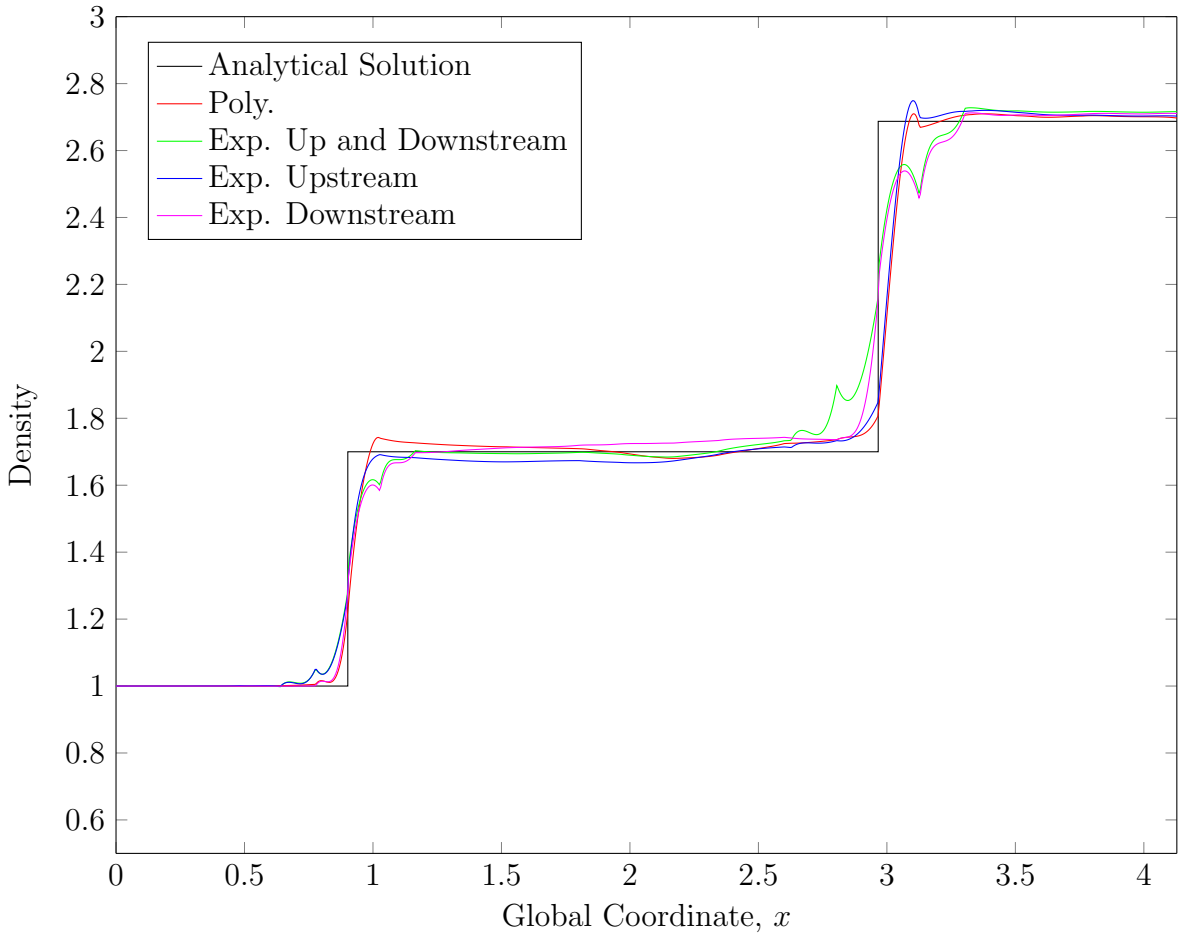

Figure 23. Density vs. x at $y = 0.5$, $\Delta t = 0.05$.

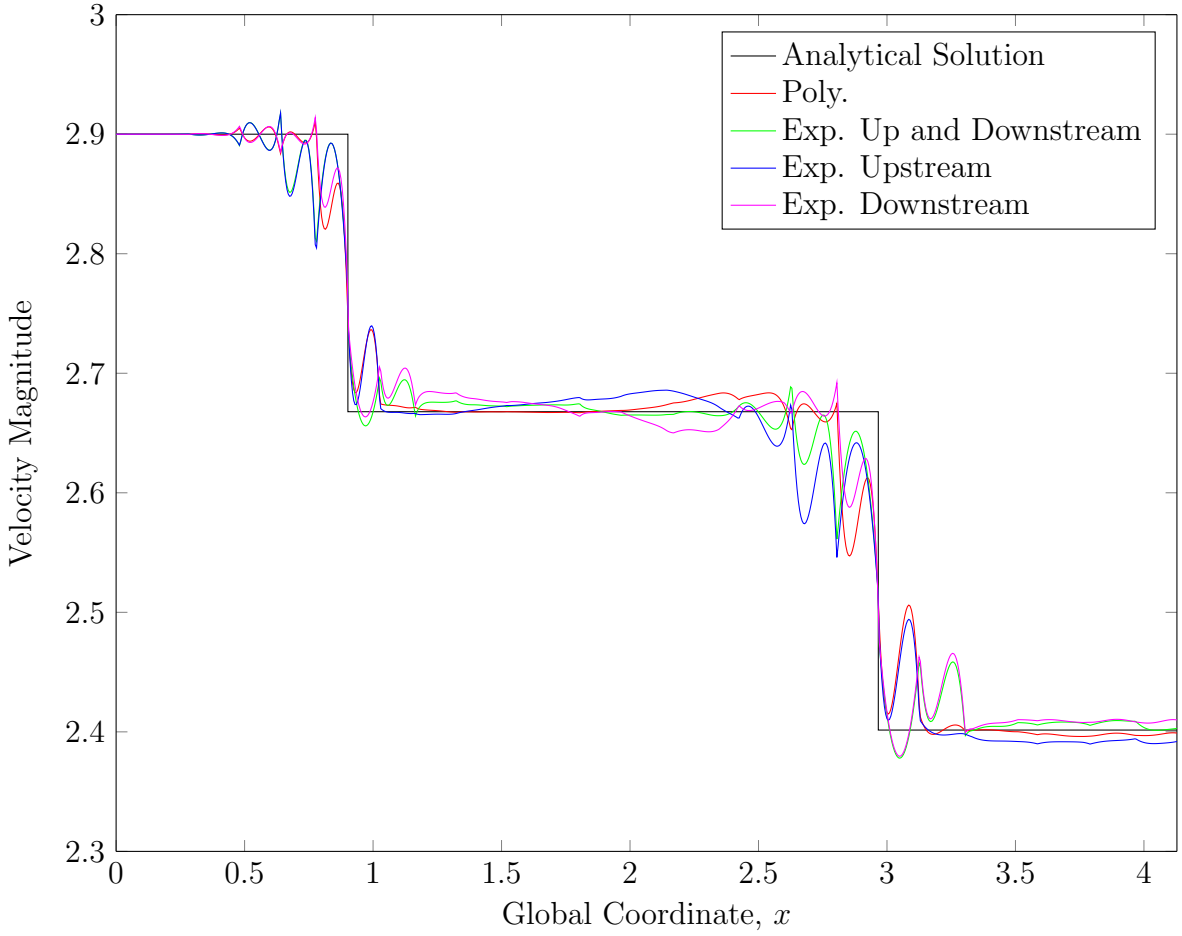

Figure 24. Velocity magnitude vs. x at $y = 0.5$, $\Delta t = 0.01$.

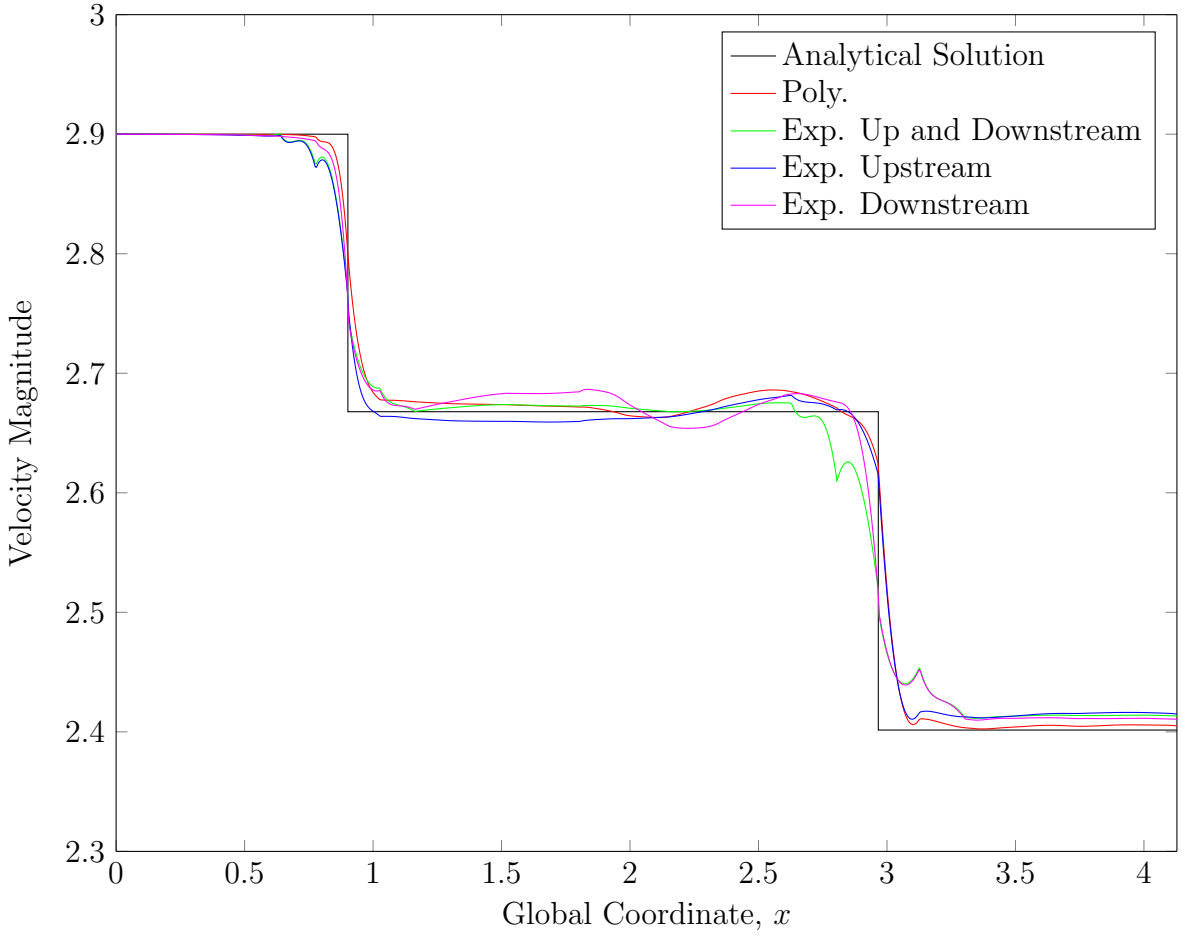

Figure 25. Velocity magnitude vs. x at $y = 0.5$, $\Delta t = 0.05$.

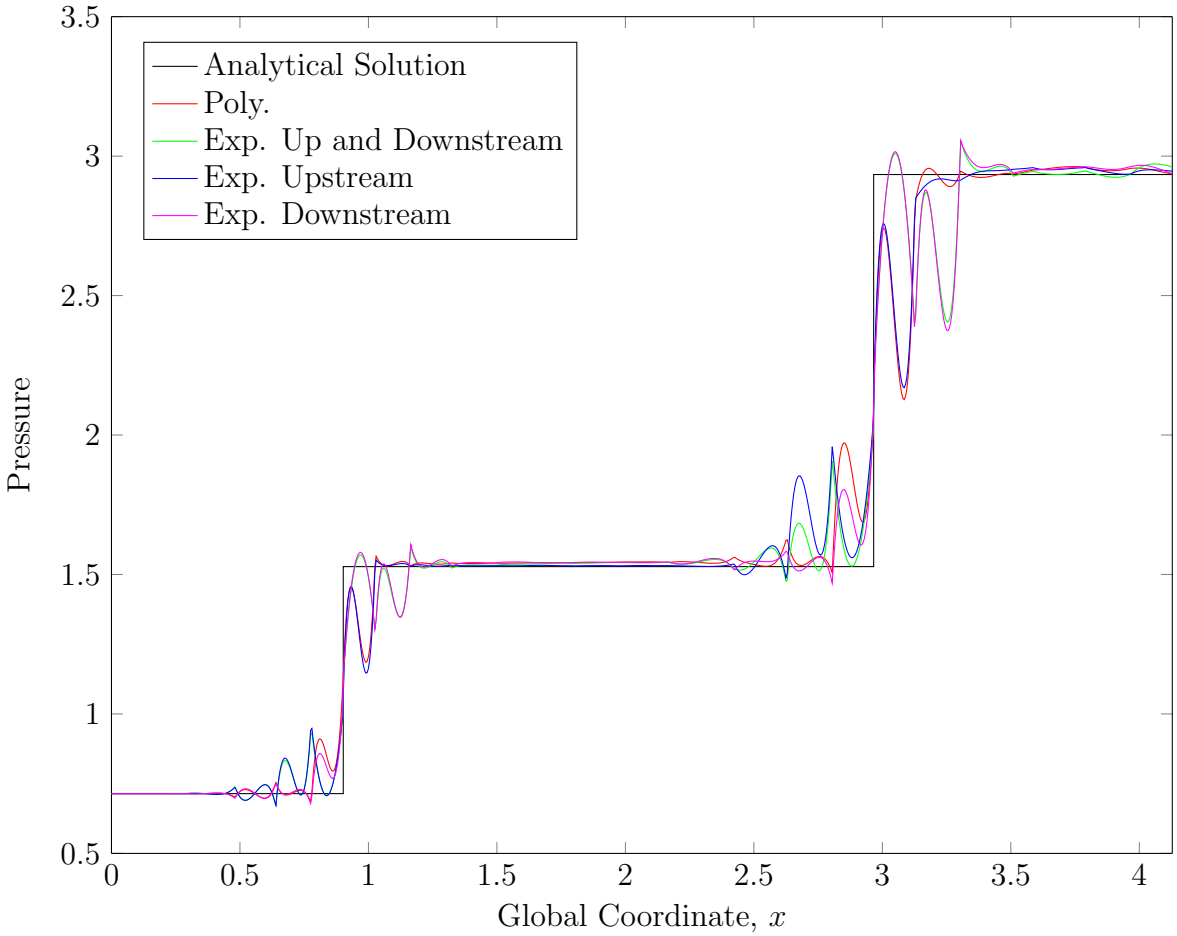

Figure 26. Pressure vs. x at $y = 0.5$, $\Delta t = 0.01$.

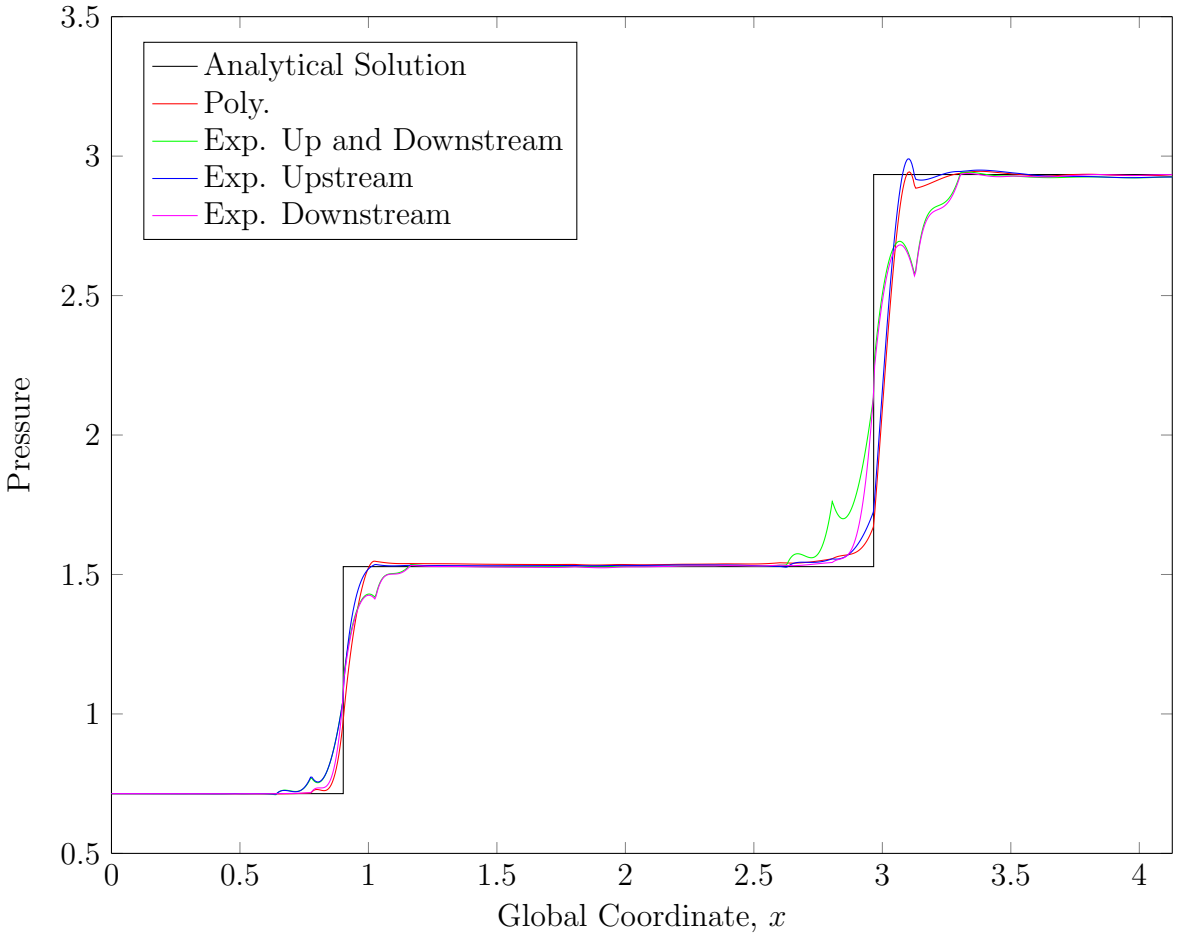

Figure 27. Pressure vs. x at $y = 0.5$, $\Delta t = 0.05$.

5.4 Comparison of Interpolation Functions Using Mesh Adaptation

Using the terminology of Ait-Ali-Yahia et al.,[9] execution of a mesh adaptation scheme and calculation of a finite element solution on the new mesh is one adaptive cycle. The initial mesh and finite element solutions presented in Section 5.3 were used as inputs to the first adaptive cycle and four adaptive cycles were calculated. The mesh adaptation scheme was limited to 50 iterations per adaptive cycle. As mentioned in Section 4.5, the mesh adaptation scheme was modified when it was used with exponential interpolation. Without the absolute gradient constraint, vertices of exponential elements could drift away from the shock waves, which would nullify any potential advantage of exponential interpolation. Squaring the components of the Hessian improved mesh adaptation near the point of reflection for both exponential and polynomial interpolation.

The absolute gradient constraint inhibited the mesh adaptation scheme from concentrating nodes near the shock waves. Even in regions where the analytical solution is

constant, there is still some small variation in the finite element solution. The small variations in the finite element solution acted as barriers that prevented element vertices from moving closer to the shocks. As can be seen in Figures 28, 33, 38, and 43 the elements adjacent to the shock waves became very small while elements in other areas remained approximately the same size. As a result, there is not much improvement in the exponential interpolation solution after the first adaptive cycle. However, the reflected shock did stay very close to it's analytical location. The solution using polynomial interpolation without the absolute gradient constraint allowed the angle between the reflected shock and the x-axis to decrease, which moved the shock downstream of it's analytical location. Figures 29, 34, 39, and 44 show that elements originally located near the shock reflection point drifted to the right. Figures 30 to 32, 35 to 37, 40 to 42, and 45 to 47 show the fluid properties along $y = 0.5$ after each adaptive cycle. The downstream movement of the reflected shock is apparent in the fluid property plots. Since the shock position before the first adaptive cycle was very close to its analytical location no matter which interpolation functions were used, it seems that the absolute gradient constraint has a much larger effect on shock location than the type of interpolation functions used.

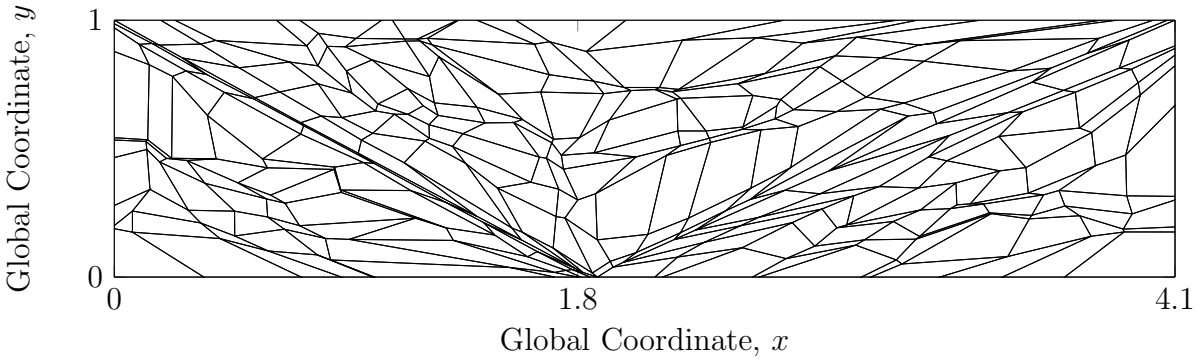

Figure 28. Mesh containing exponential elements downstream of the shocks after the first adaptive cycle.

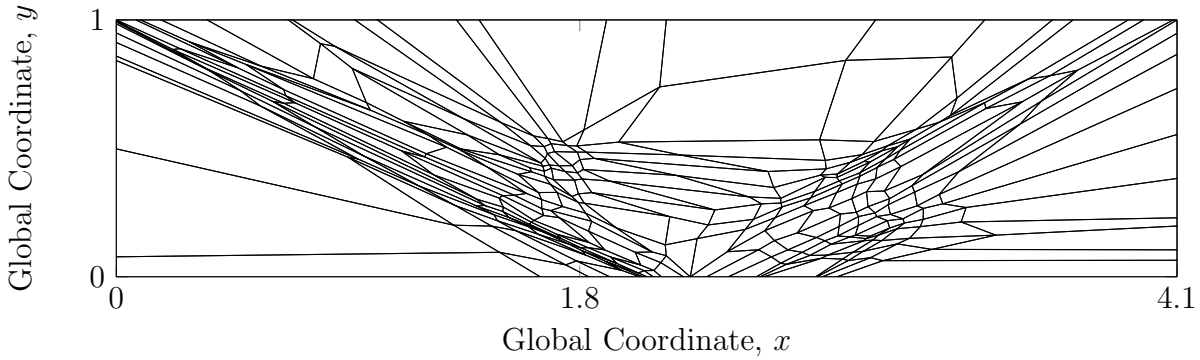

Figure 29. Mesh containing only polynomial elements after the first adaptive cycle.

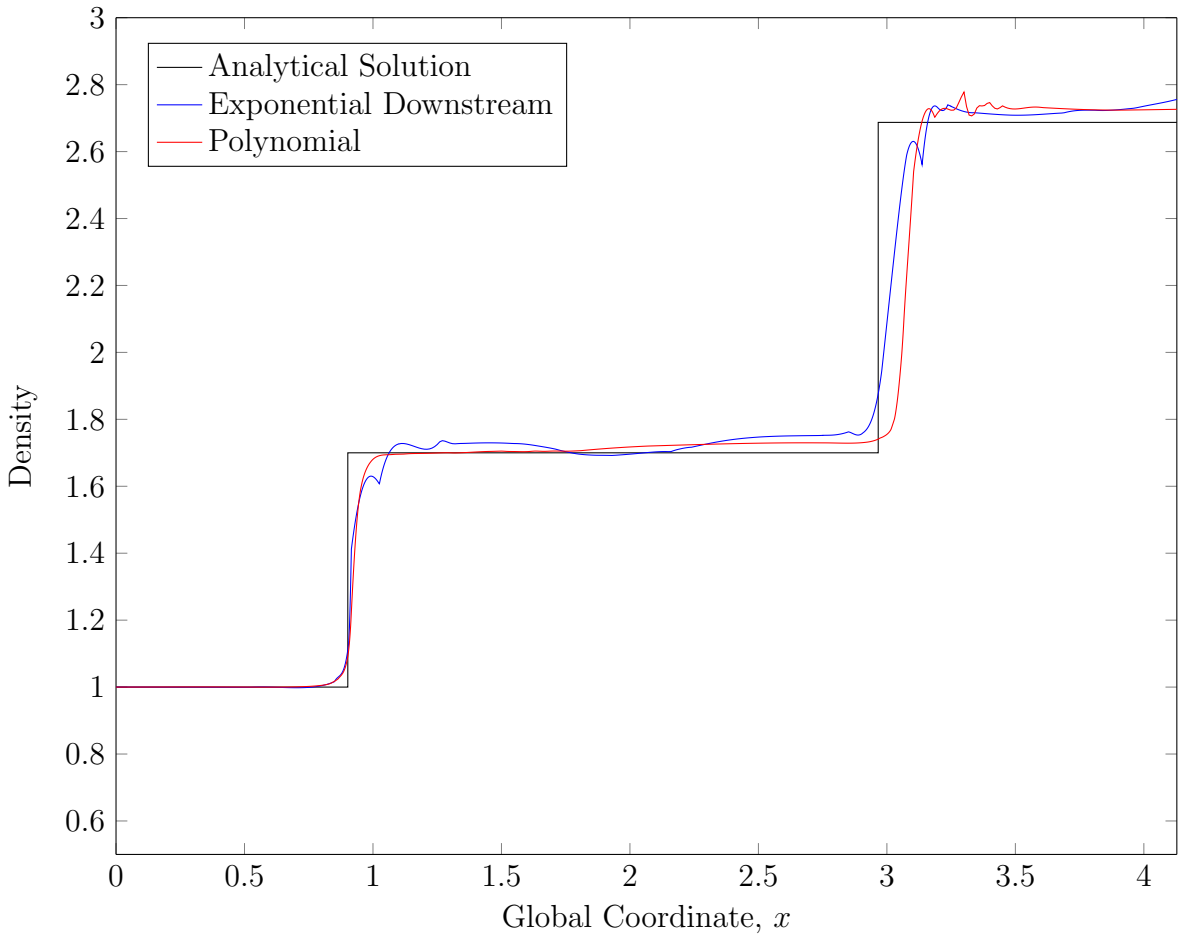

Figure 30. Density vs. x at $y = 0.5$ after the first adaptive cycle.

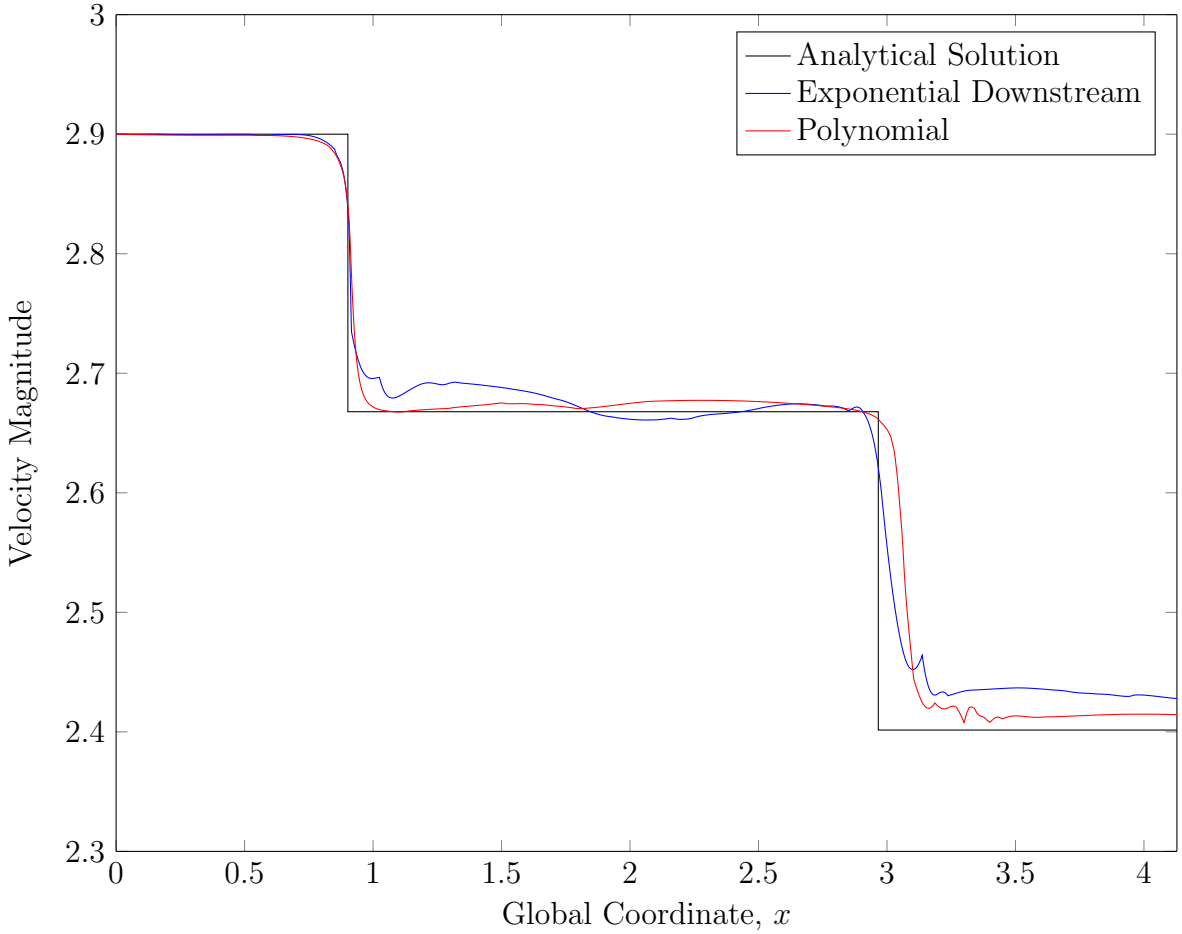

Figure 31. Velocity magnitude vs. x at $y = 0.5$ after the first adaptive cycle.

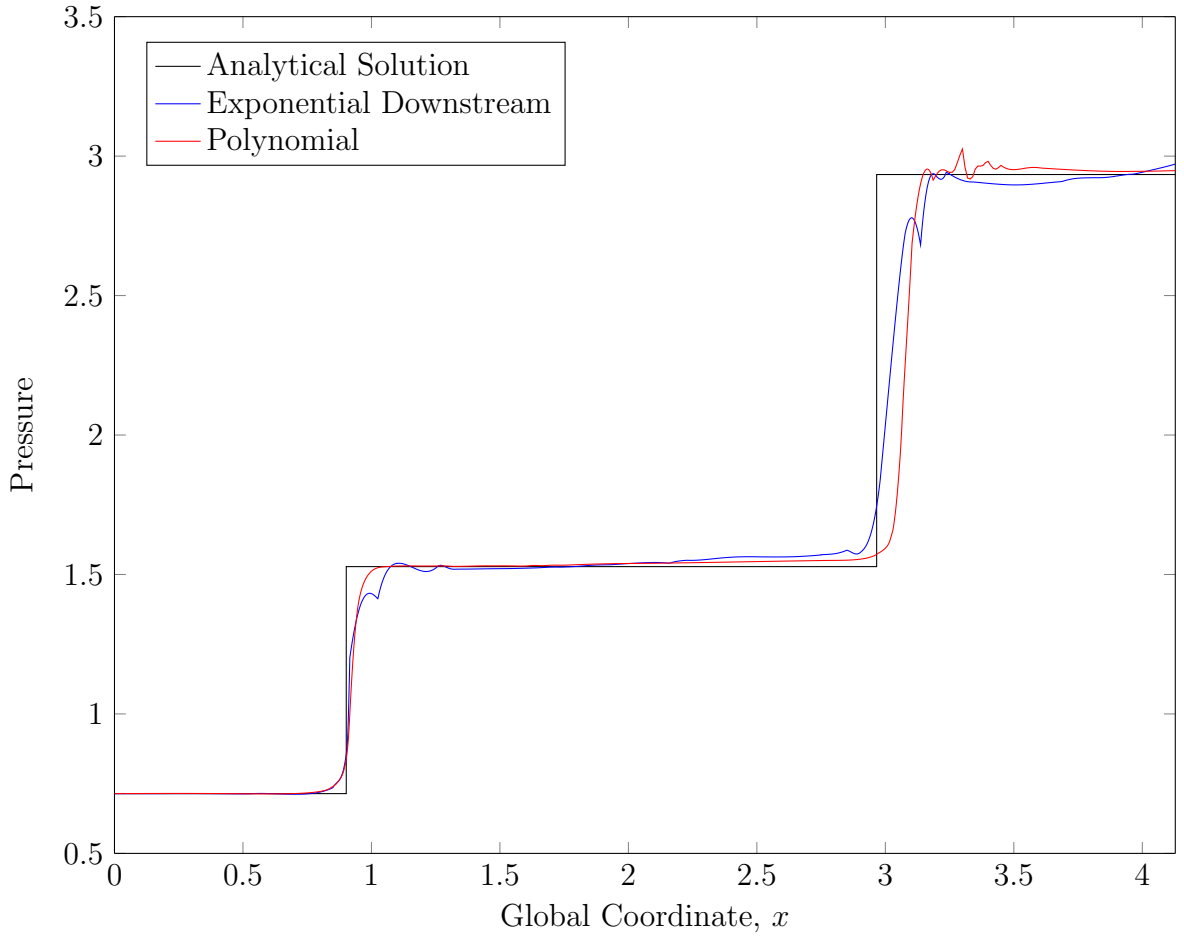

Figure 32. Pressure vs. x at $y = 0.5$ after the first adaptive cycle.

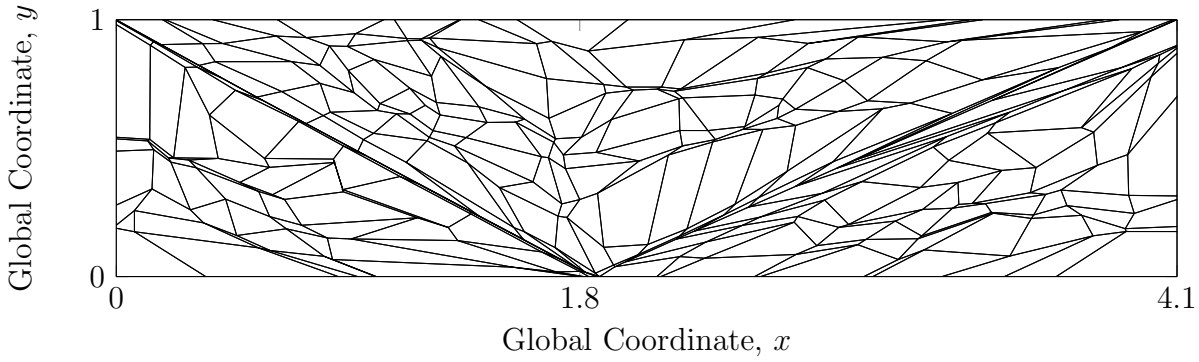

Figure 33. Mesh containing exponential elements downstream of the shocks after the second adaptive cycle.

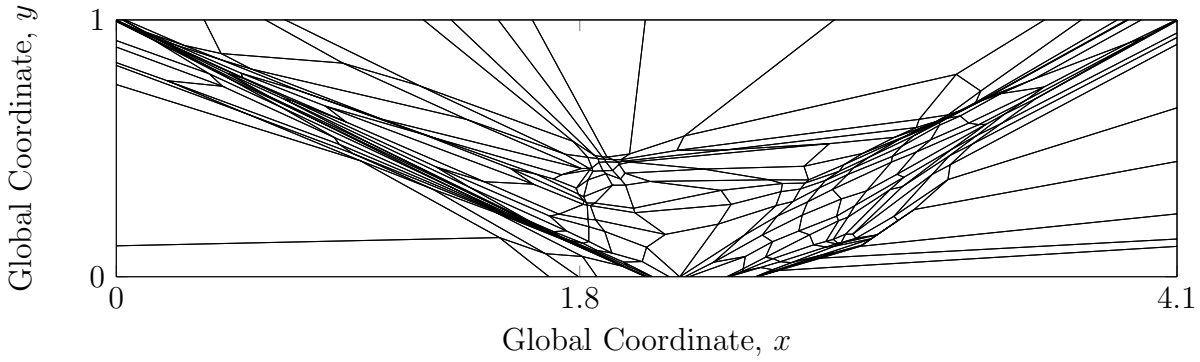

Figure 34. Mesh containing only polynomial elements after the second adaptive cycle.

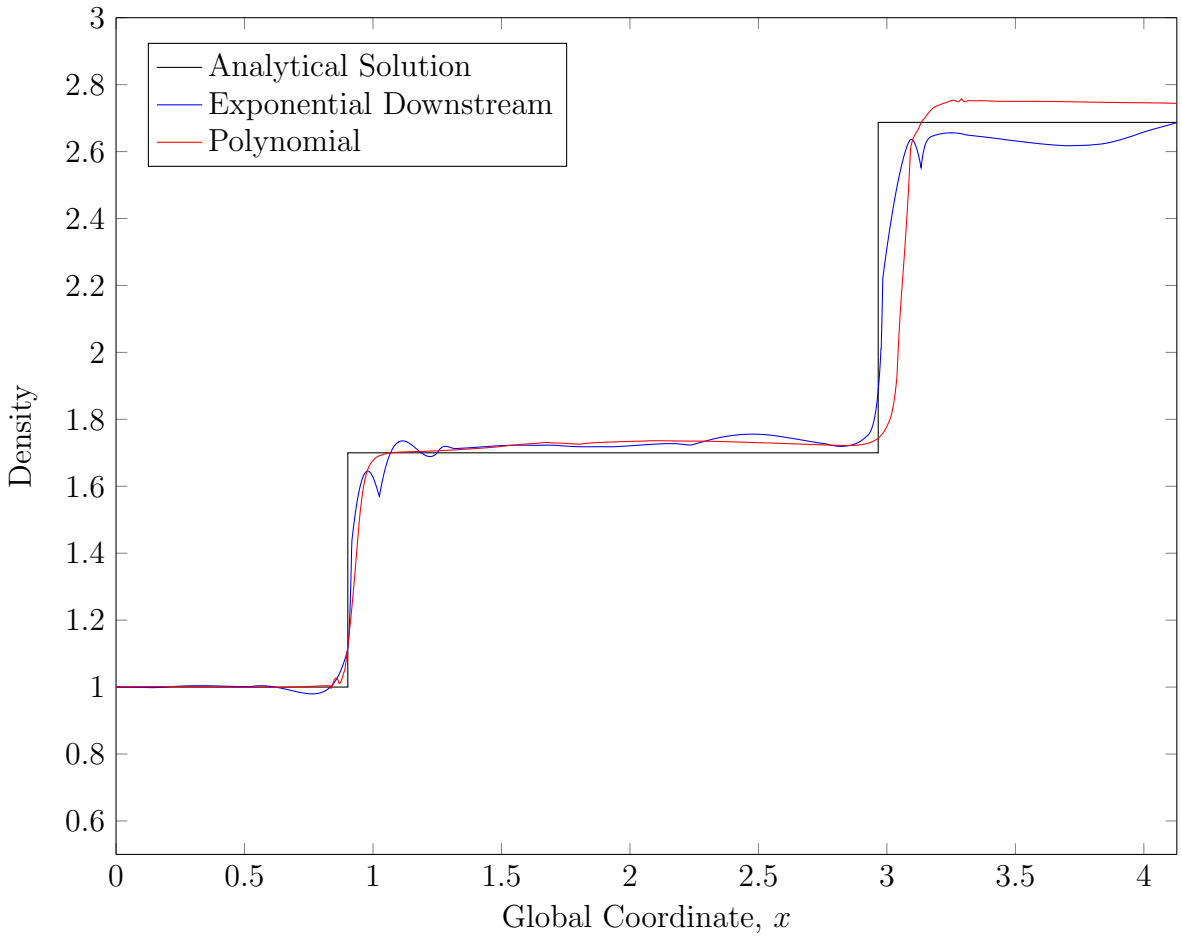

Figure 35. Density vs. x at $y = 0.5$ after the second adaptive cycle.

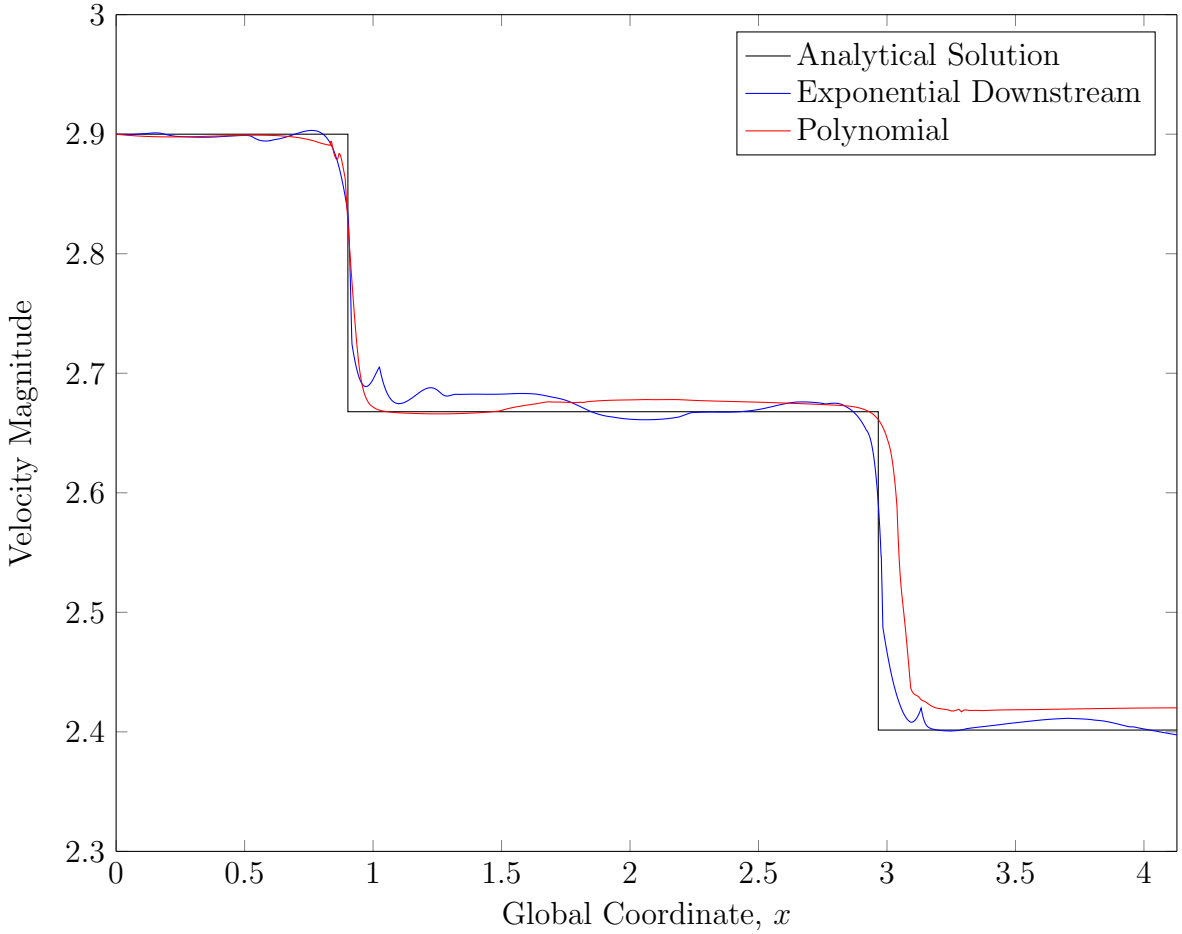

Figure 36. Velocity magnitude vs. x at $y = 0.5$ after the second adaptive cycle.

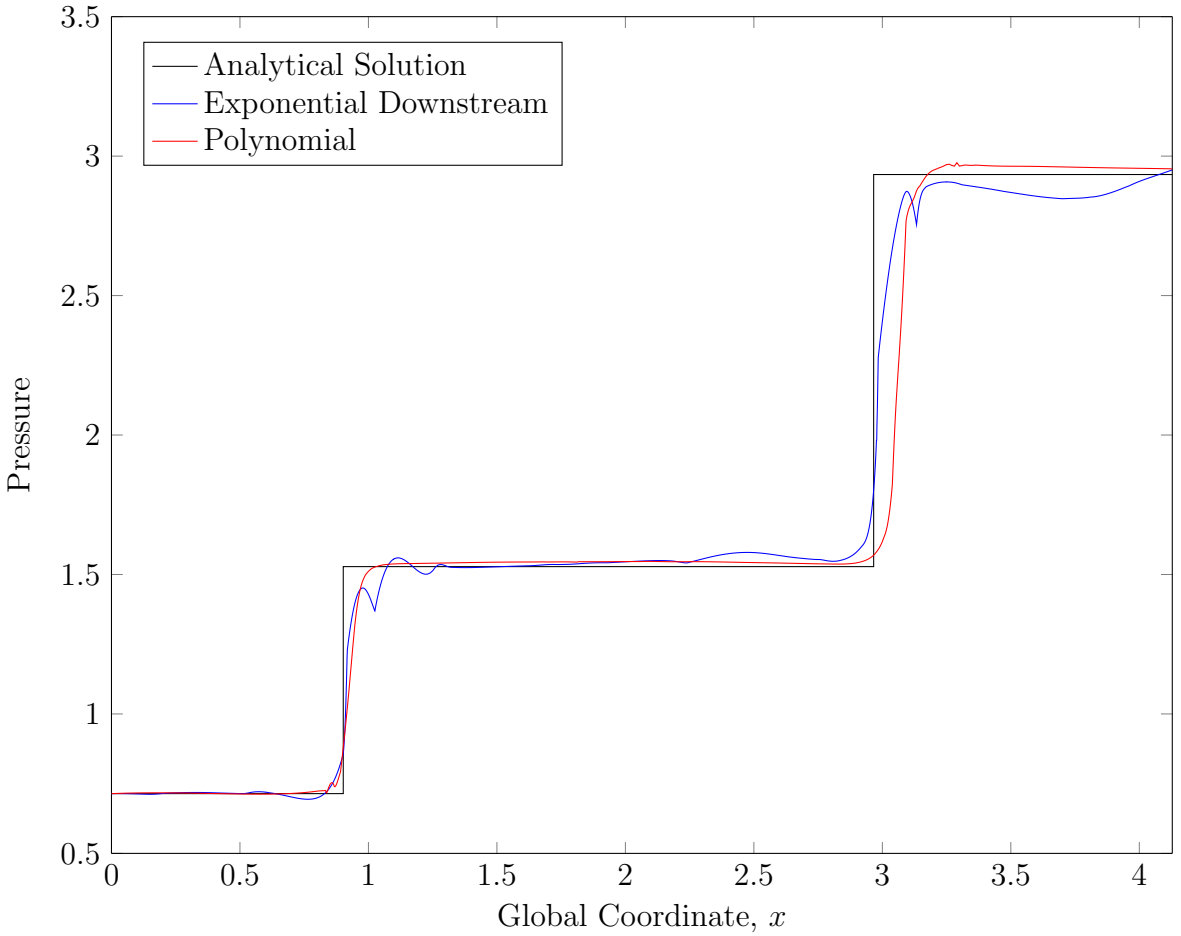

Figure 37. Pressure vs. x at $y = 0.5$ after the second adaptive cycle.

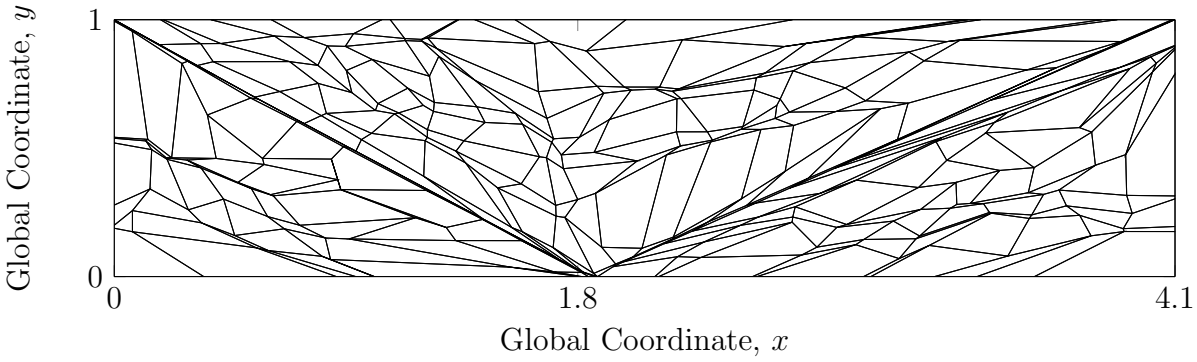

Figure 38. Mesh containing exponential elements downstream of the shocks after the third adaptive cycle.

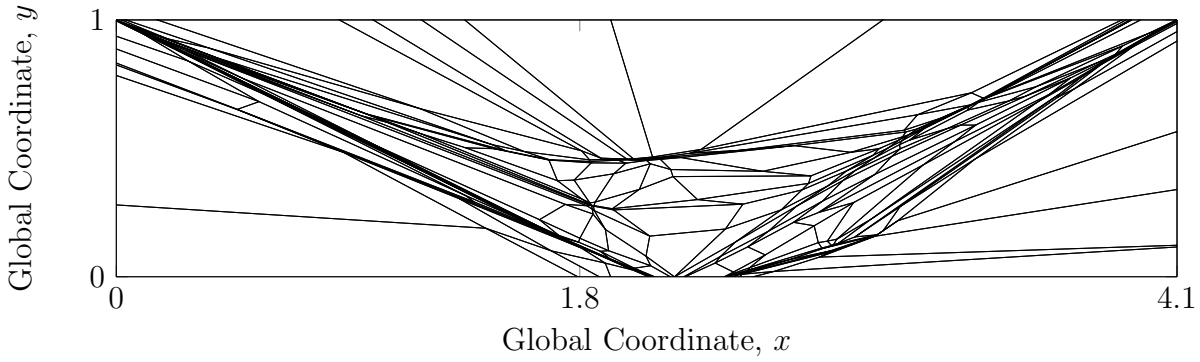

Figure 39. Mesh containing only polynomial elements after the third adaptive cycle.

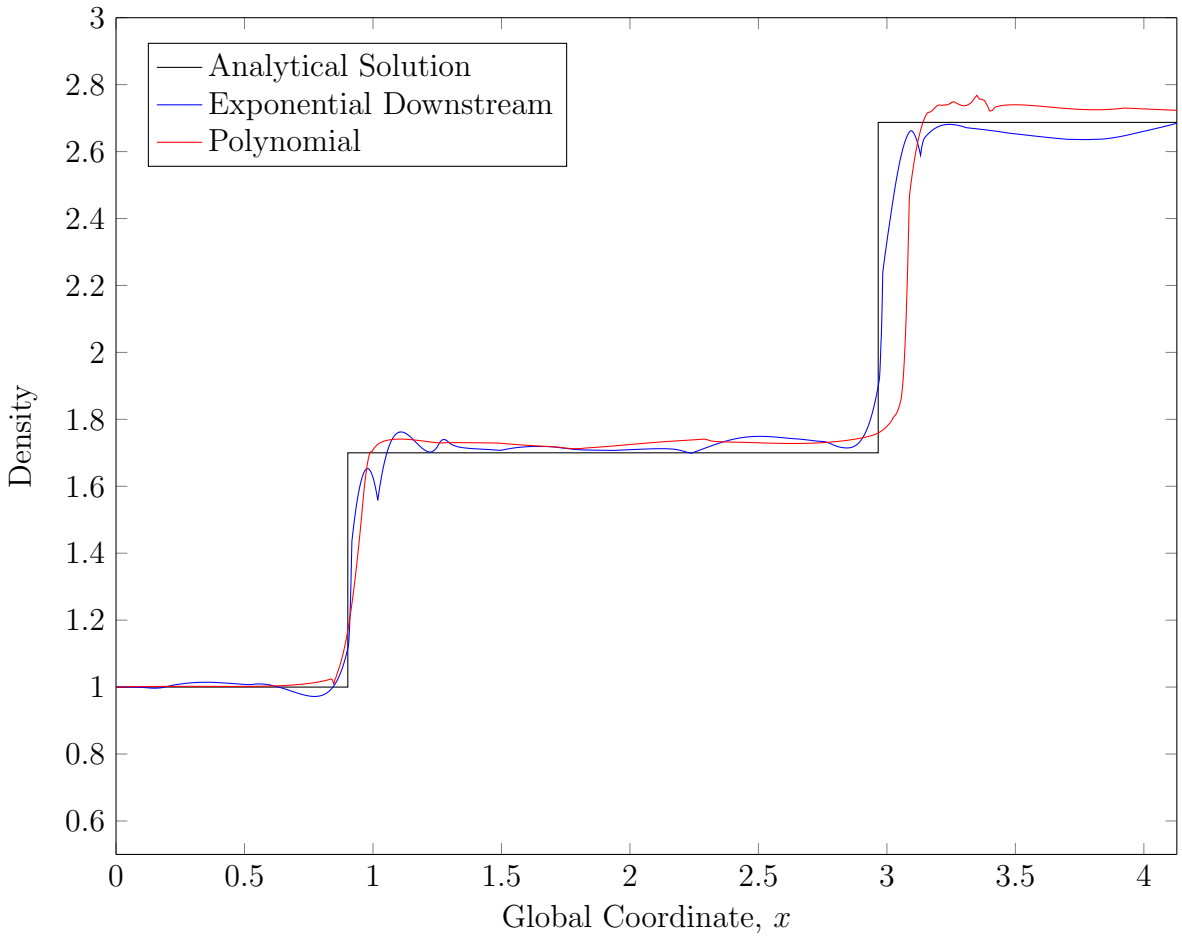

Figure 40. Density vs. x at $y = 0.5$ after the third adaptive cycle.

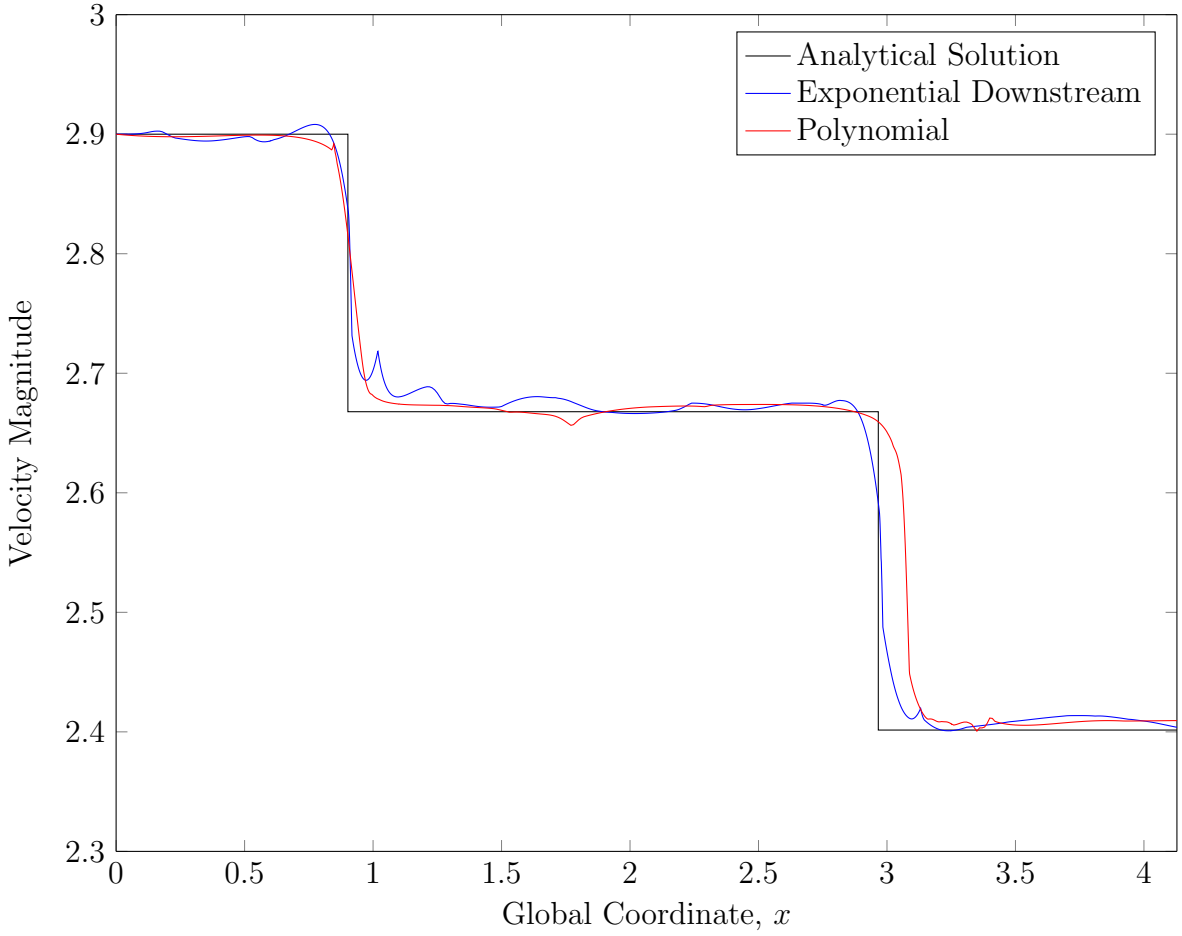

Figure 41. Velocity magnitude vs. x at $y = 0.5$ after the third adaptive cycle.

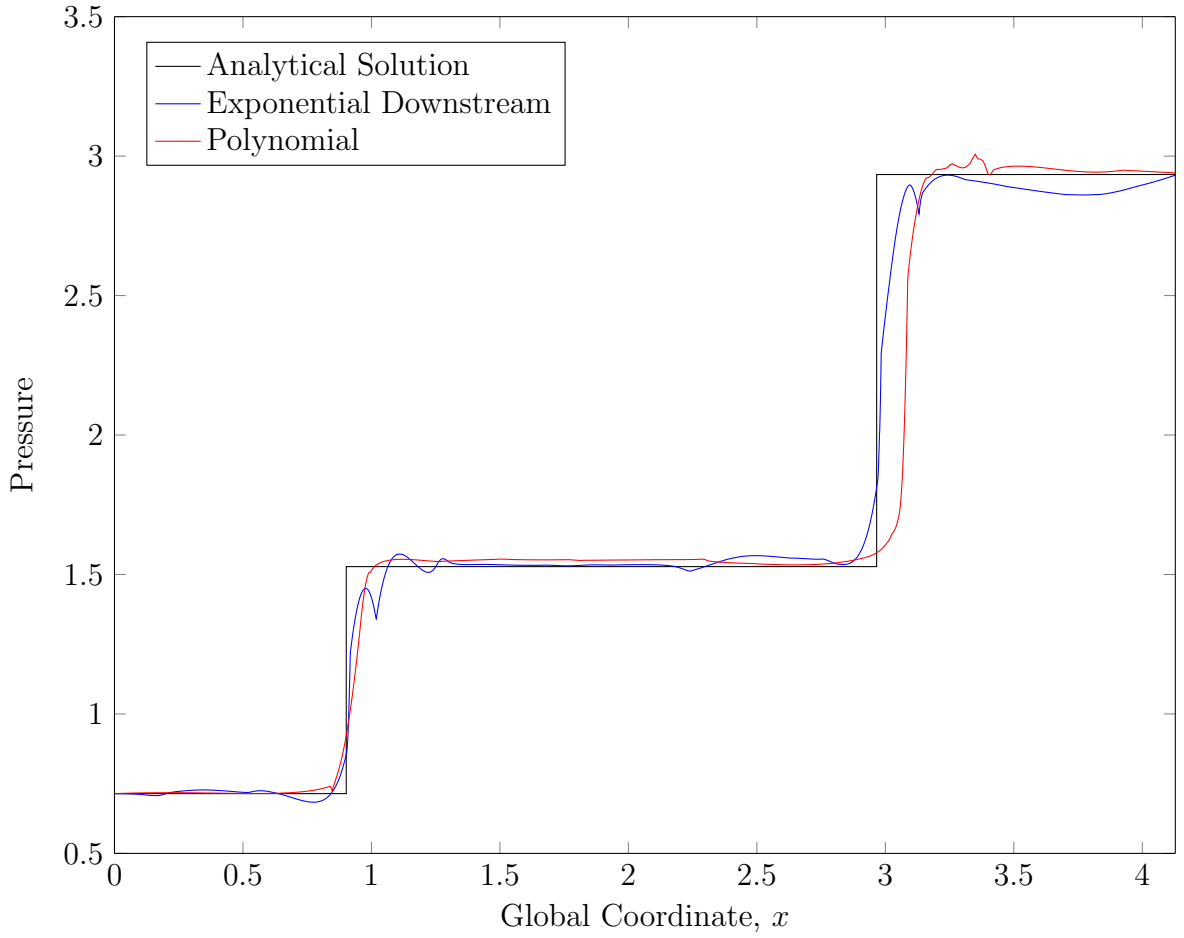

Figure 42. Pressure vs. x at $y = 0.5$ after the third adaptive cycle.

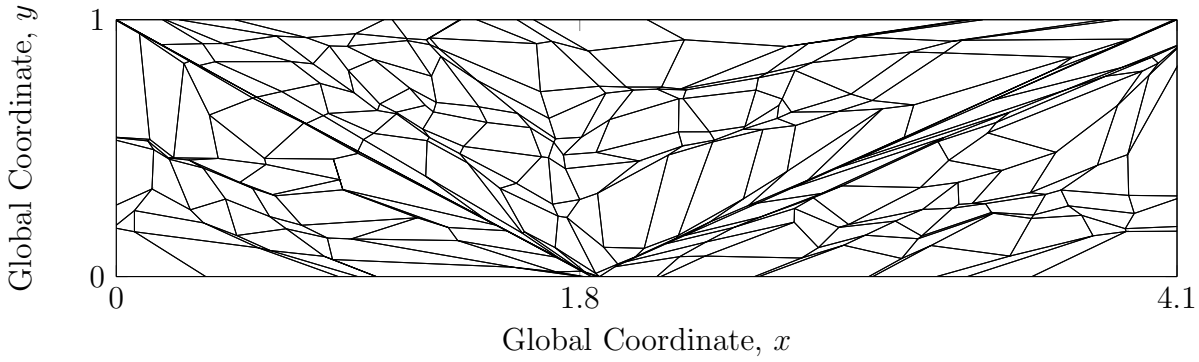

Figure 43. Mesh containing exponential elements downstream of the shocks after the fourth adaptive cycle.

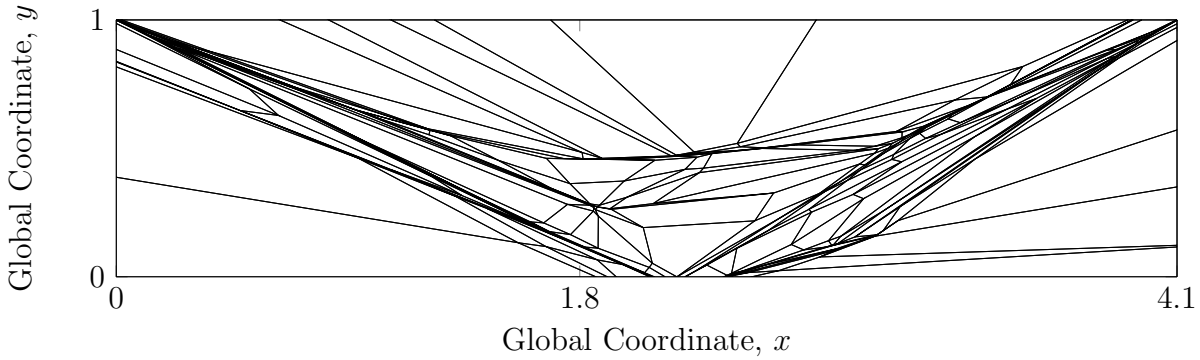

Figure 44. Mesh containing only polynomial elements after the fourth adaptive cycle.

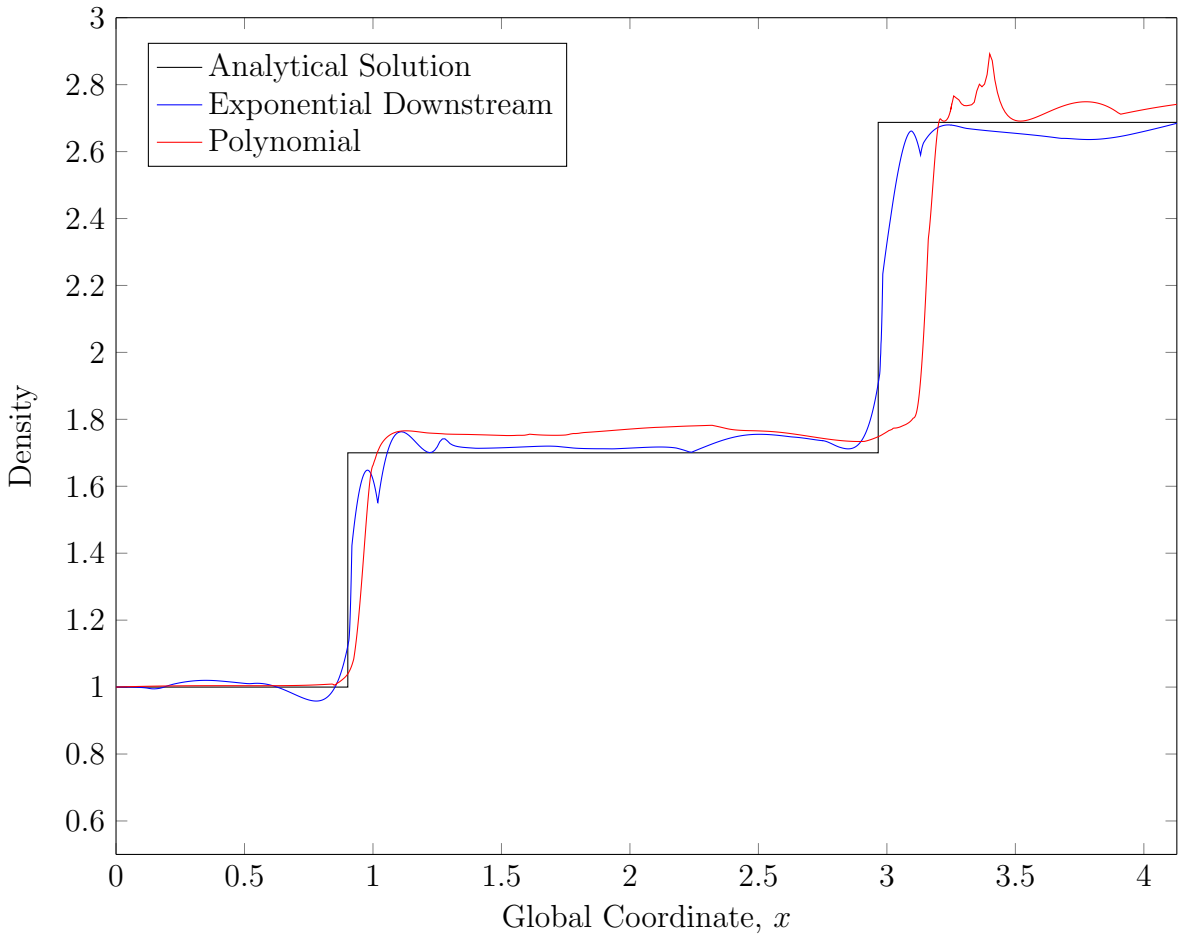

Figure 45. Density vs. x at $y = 0.5$ after the fourth adaptive cycle.

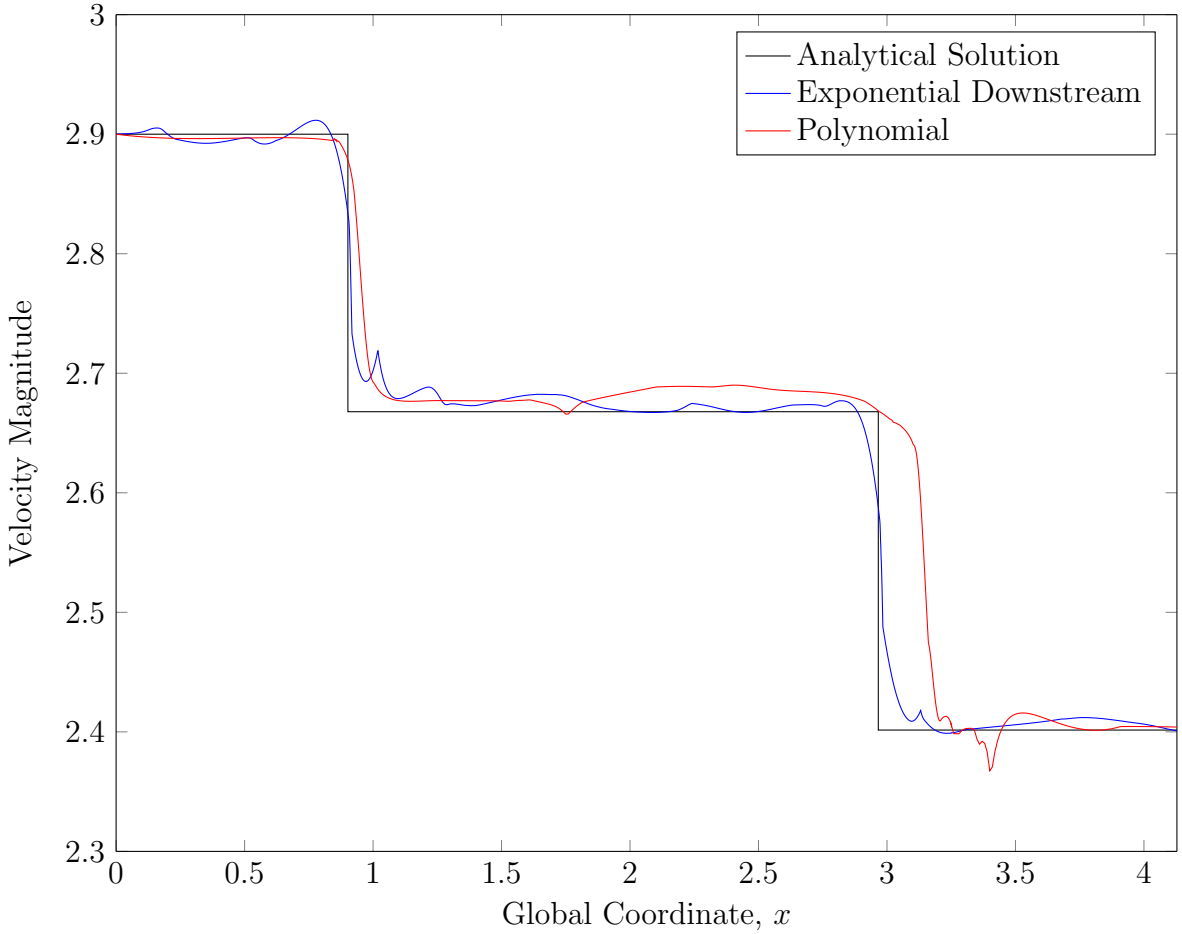

Figure 46. Velocity magnitude vs. x at $y = 0.5$ after the fourth adaptive cycle.

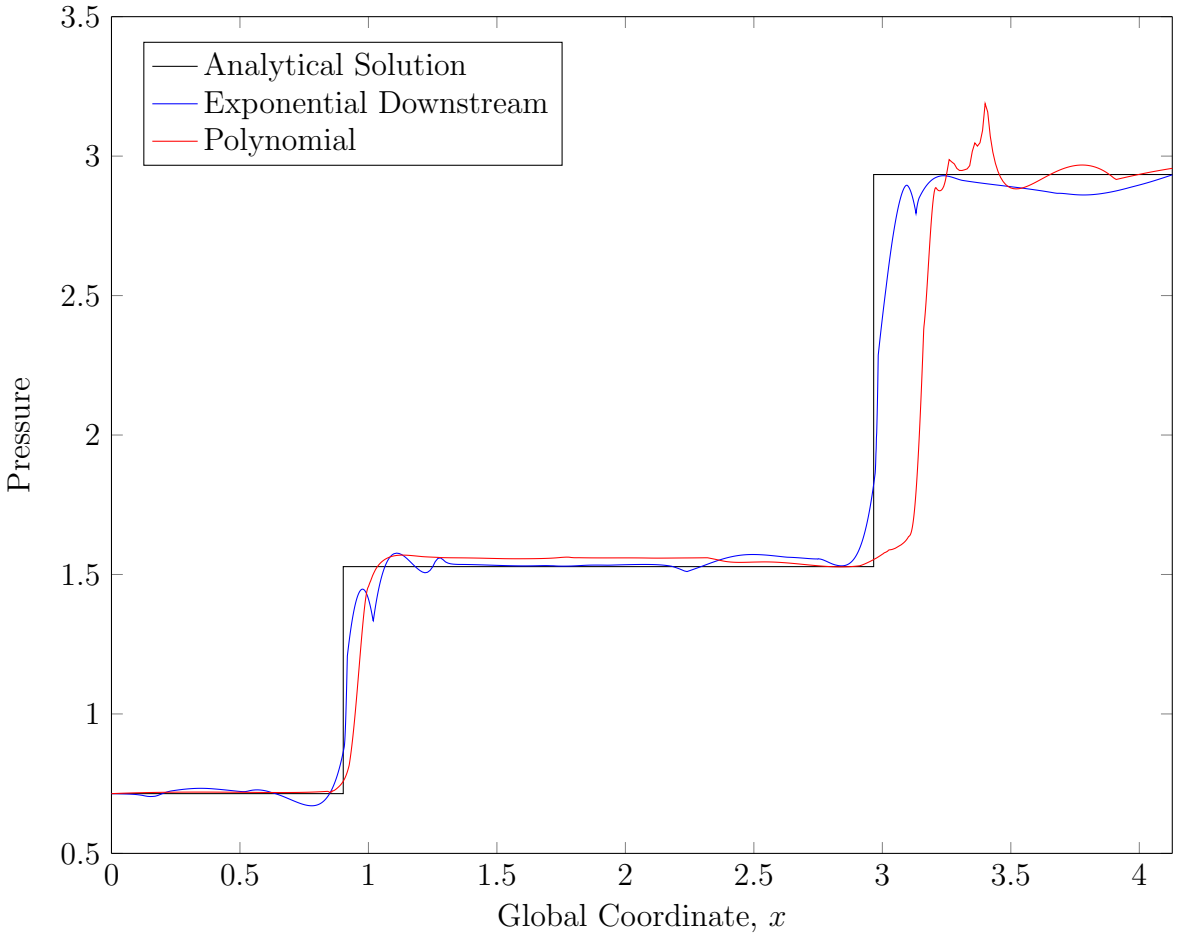

Figure 47. Pressure vs. x at $y = 0.5$ after the fourth adaptive cycle.

To make the absolute gradient constraint less restrictive while still keeping element edges coincident with shock waves, $\|\nabla\sigma\|$ was rounded to the nearest tenth. In regions where the analytical solution is constant but the finite element solution contained small variations, rounding the absolute gradient constraint allowed element vertices to migrate closer to the shocks. A much better mesh was produced in the first adaptive cycle but when exponential interpolation was used in the elements downstream of the shocks, the Newton iterations in the finite element solver diverged. The Newton iterations converged when the same mesh was used with polynomial interpolation in all elements. There appears to be a lower bound to the element size when using exponential interpolation but no further investigation to find the exact lower bound was conducted. The results of this adaptive cycle are shown in Figures 48 to 51.

Upon final editing of the MATLAB code for inclusion in the Appendix, it was discovered that when calculating the gradients in the absolute gradient constraint, division by $\|J\|$ was left out. That mistake was corrected and all of the calculations in this section,

except those presented in Figures 48 to 51, were corrected. Neglecting to divide by $\|J\|$ only affected mesh adaptation, not the finite element solution on a given mesh. Therefore, the observation that small exponential elements may lead to divergent Newton iterations is still true.

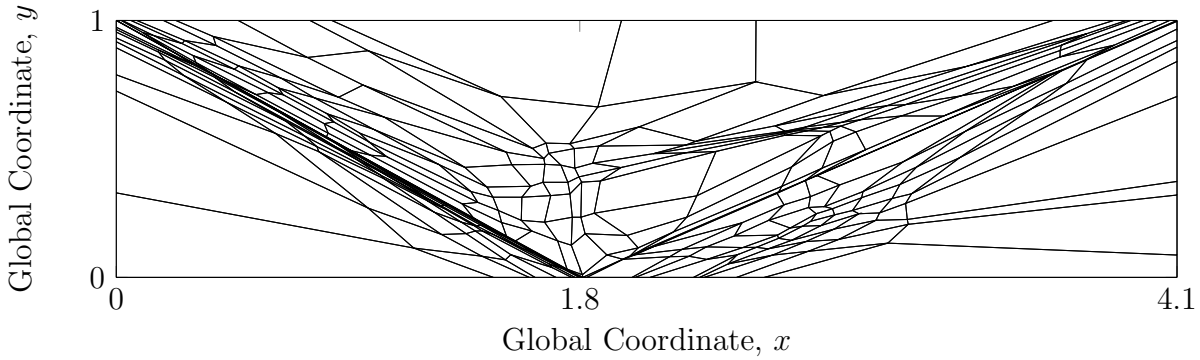

Figure 48. Mesh containing exponential elements downstream of the shocks that were changed to polynomial elements during the first adaptive cycle.

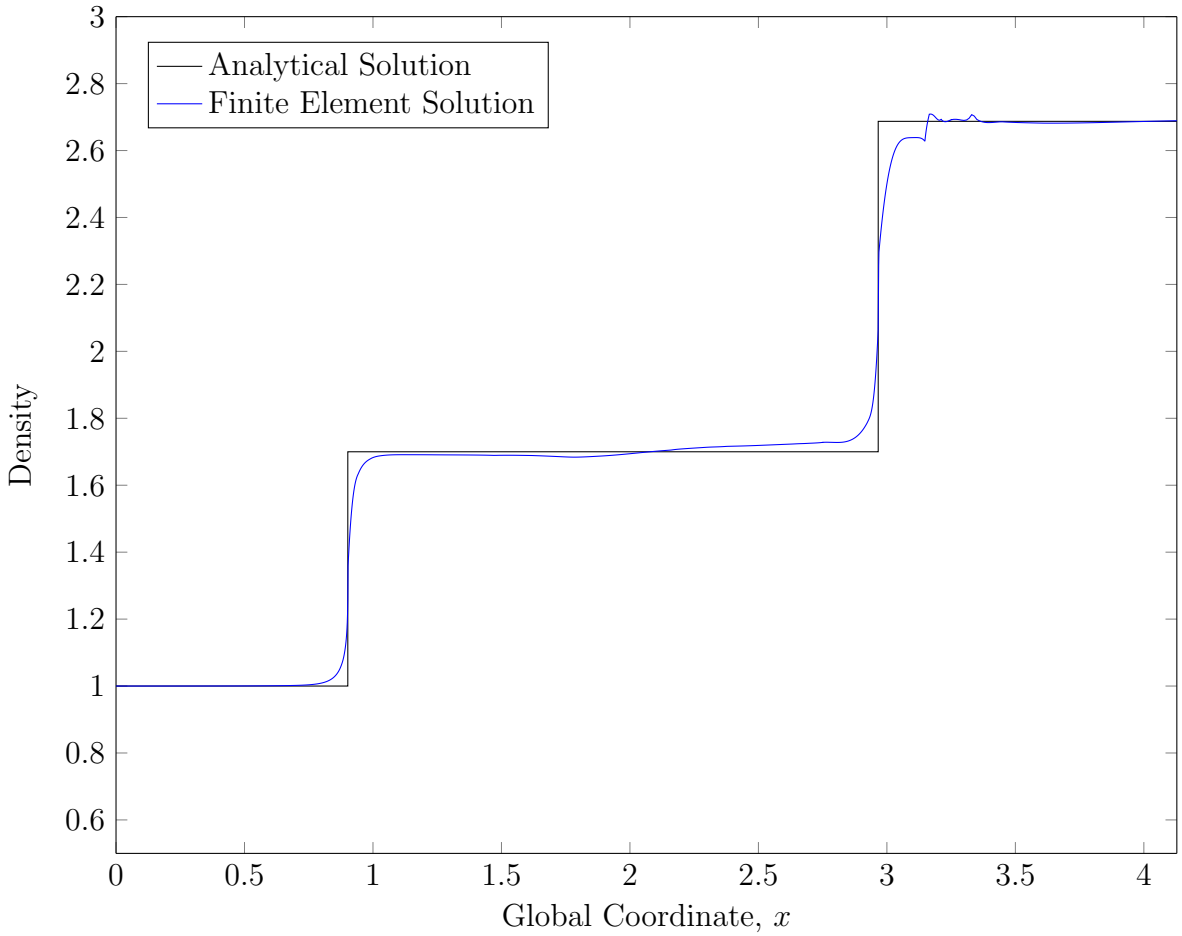

Figure 49. Density vs. x at $y = 0.5$ after the exponential elements were changed to polynomial elements.

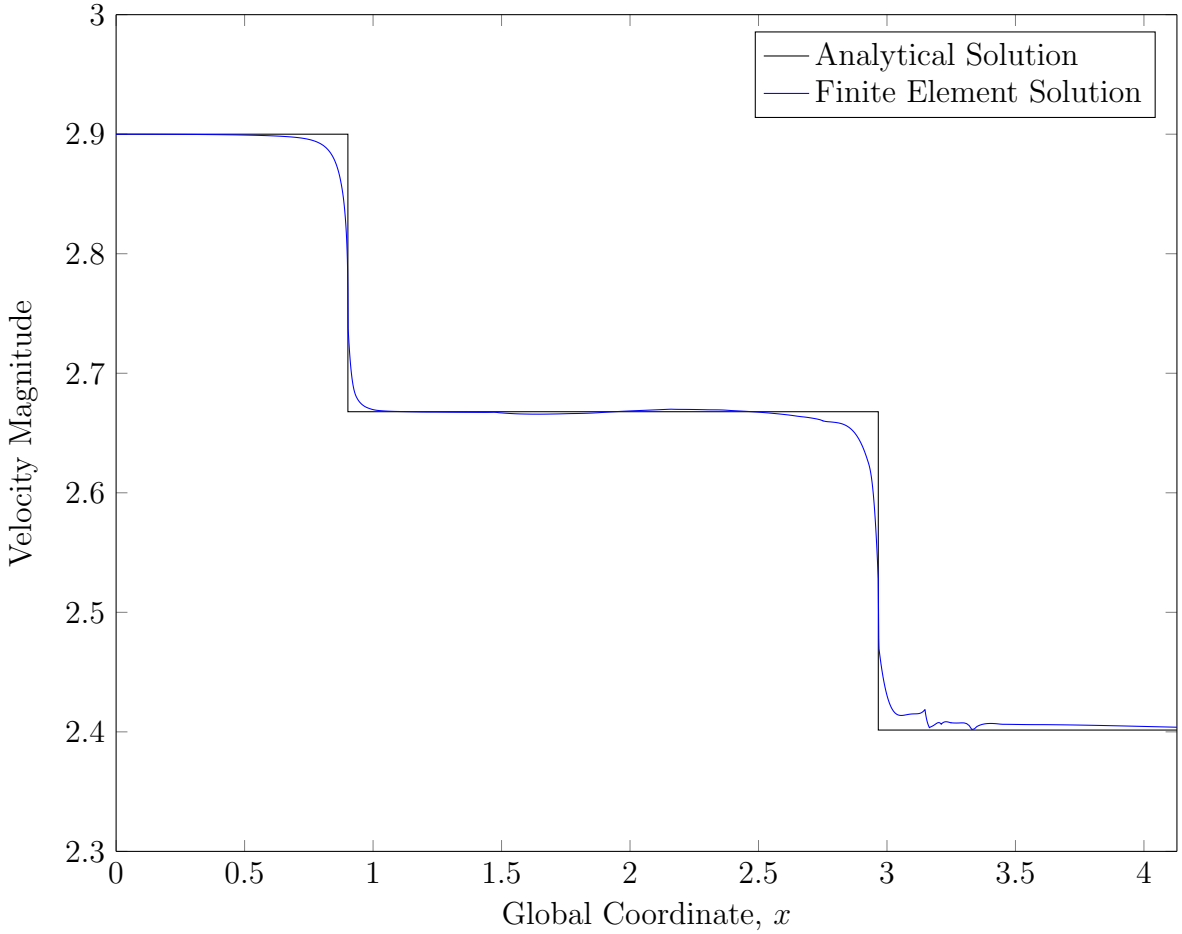

Figure 50. Velocity magnitude vs. x at $y = 0.5$ after the exponential elements were changed to polynomial elements.

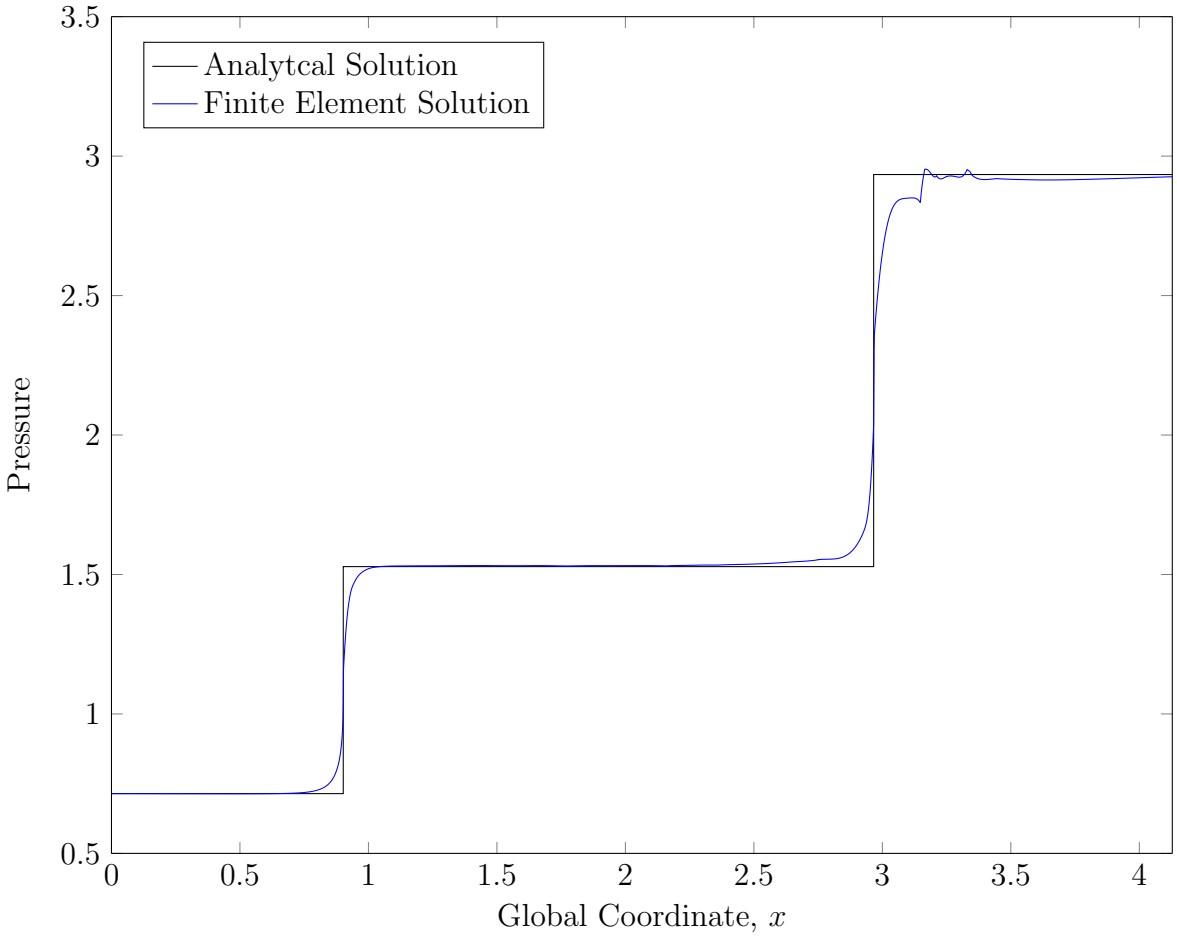

Figure 51. Pressure vs. x at $y = 0.5$ after the exponential elements were changed to polynomial elements.

CHAPTER 6
CONCLUSION

Exponential interpolation only reduced error in velocity magnitude when compared to polynomial interpolation. Although, any qualitative difference between the two types of interpolation is difficult to see. Only Δt had any obvious effect on the sharpness of the shocks and the oscillation adjacent to the shocks. The restrictive absolute gradient constraint was necessary to keep exponential elements adjacent to the shocks but it also stifled mesh adaptation in other regions of the domain. With the absolute gradient constraint weakened by rounding, smaller exponential elements were created that caused the Newton iterations in the finite element solver to diverge.

The absolute gradient constraint was necessary because the exponential interpolation functions can only approximate sharp changes in the dependent variables at element edges. Whereas polynomial interpolation cannot approximate large gradients well, it can approximate gradients in any direction, which makes polynomial interpolation more versatile than exponential interpolation. The other serious disadvantage to using exponential interpolation is the vastly higher number of quadrature points required for numerical integration compared to integration of polynomials. Therefore, if exponential interpolation is used, it should be used sparingly so that calculation time does not increase too much.

There are no advantages to using exponential interpolation. Even with the benefit of a known analytical solution and a shock-aligned mesh, exponential interpolation only showed a slight reduction in solution error. In order for exponential interpolation to be of use in a setting where shock locations are unknown, the mesh modification scheme would have to be updated to determine which elements should use exponential interpolation and to determine what the exponential parameters should be for each element. Exponential interpolation lacks the versatility of polynomial interpolation and introduces complexity without adding anything advantageous.

REFERENCES

[1]Taghaddosi, F., Habashi, W. G., Guèvremont, G., and Ait-Ali-Yahia, D., "An Adaptive Least-Squares Method for the Compressible Euler Equations," *International Journal for Numerical Methods in Fluids*, Vol. 31, 1999, pp. 1121–1139.

[2]Cockburn, B., Karniadakis, G. E., and Shu, C.-W., "The Development of Discontinuous Galerkin Methods," *Discontinuous Galerkin Methods: Theory, Computation and Applications*, edited by B. Cockburn, G. E. Karniadakis, and C.-W. Shu, Vol. 11 of *Lecture Notes in Computational Science and Engineering*, Springer, New York, 2000, pp. 3–50.

[3]Potanza, J. P. and Reddy, J. N., "Least-Squares Finite Element Formulations for Viscous Incompressible and Compressible Fluid Flows," *Computer Methods in Applied Mechanics and Engineering*, Vol. 195, 2006, pp. 2454–2494.

[4]Potanza, J. P., Diao, X., Reddy, J. N., and Surana, K. S., "Least-Squares Finite Element Models of Two-Dimensional Compressible Flows," *Finite Elements in Analysis and Design*, Vol. 40, 2004, pp. 629–644.

[5]Meiring, A. F. and Rosinger, E. E., "Discontinuous Finite Element Basis Functions for Nonlinear Partial Differential Equations," *Applied Mathematical Modelling*, Vol. 13, 1989, pp. 544–549.

[6]Ichimura, T., Hori, M., and Wijerathne, M. L. L., "Linear Finite Elements with Orthogonal Discontinuous Basis Functions for Explicit Earthquake Ground Motion Modeling," *International Journal for Numerical Methods in Engineering*, Vol. 86, 2011, pp. 286–300.

[7]Sani, R. L. and Gresho, P. M., "Resume and Remarks on the Open Boundary Condition Minisymposium," *International Journal for Numerical Methods in Fluids*, Vol. 18, 1994, pp. 983–1008.

[8]Reddy, J. N., *An Introduction to the Finite Element Method*, McGraw-Hill, New York, NY, 3rd ed., 2006.

[9]Ait-Ali-Yahia, D., Habashi, W. G., and Tam, A., "A Directionally Adaptive Methodology Using an Edge-Based Error Estimate on Quadrilateral Grids," *International Journal for Numerical Methods in Fluids*, Vol. 23, 1996, pp. 673–690.

[10]Ma, T.-W., "Higher Chain Formula Proved by Combinatorics," *The Electronic Journal of Combinatorics*, Vol. 16, 2009.

[11]Anderson, J. D., *Modern Compressible Flow with Historical Perspective*, McGraw-Hill, New York, NY, 3rd ed., 2003.

APPENDIX

The MATLAB code in this Appendix, with the exception of `changem_fea`, `chebpts`, and `legpts` was written by the author and used for the calculations presented in this paper. The first part of this appendix is a list that covers the three basic parts of a finite element program; the preprocessor, processor, and postprocessor. Under each basic part, the function names used to compute that part of the finite element problem are listed. If one function calls other functions, the names of those functions are indented under the name of the calling function. The next part of this appendix shows the source code for each function. The functions are listed in alphabetical order by function name and each function starts on a new page. Comments at the beginning of each function briefly explain the function's purpose, inputs, and outputs.

Preprocessor Functions

```
RectDomain
   Element_Mesh
      chebpts
TriDomain
   Element_Mesh_eql
RefineElements
   Element_Mesh
      chebpts
```

Processor Functions

```
EulerSolver
   GlobalMat
      Euler_K
         LegendrePts2D
            Exp_1D_GLquad_Num
            legpts
         ElementJacobian
            Master_int2D
               chebpts
               Lagrange_int1D
               Exp_int1D
         Master_int2D
            chebpts
            Lagrange_int1D
```

```
            Exp_int1D
         changem_fea
      GlobalF
         EulerF
            LegendrePts2D
               Exp_1D_GLquad_Num
               legpts
            ElementJacobian
               Master_int2D
                  chebpts
                  Lagrange_int1D
                  Exp_int1D
            Master_int2D
               chebpts
               Lagrange_int1D
               Exp_int1D
         changem_fea
      ImposeBC
         WallBC
            Num_LegendrePts
               Exp_1D_GLquad_Num
            legpts
            ElementJacobian
               Master_int2D
                  chebpts
                  Lagrange_int1D
                  Exp_int1D
            Master_int2D
               chebpts
               Lagrange_int1D
               Exp_int1D
         changem_fea
      GS_Precondition
   UpdateMesh
      Element2Grid
         ElementJacobian
```

```
      Master_int2D
         chebpts
         Lagrange_int1D
         Exp_int1D
   ElementHessian
      Master_int2D
         chebpts
         Lagrange_int1D
         Exp_int1D
   Master_int2D
      chebpts
      Lagrange_int1D
      Exp_int1D
El_springK
   Num_LegendrePts
      Exp_1D_GLquad_Num
   ElementJacobian
      Master_int2D
         chebpts
         Lagrange_int1D
         Exp_int1D
   ElementHessian
      Master_int2D
         chebpts
         Lagrange_int1D
         Exp_int1D
   Master_int2D
      chebpts
      Lagrange_int1D
      Exp_int1D
ElementJacobian
   Master_int2D
      chebpts
      Lagrange_int1D
      Exp_int1D
Master_int2D
```

```
      chebpts
      Lagrange_int1D
      Exp_int1D
    Element_Mesh
      chebpts
```

Postprocessor Functions

```
  Eval_Dofs
    Master_int2D
      chebpts
      Lagrange_int1D
      Exp_int1D
  FEASurf
    Element2Grid
      ElementJacobian
        Master_int2D
          chebpts
          Lagrange_int1D
          Exp_int1D
      ElementHessian
        Master_int2D
          chebpts
          Lagrange_int1D
          Exp_int1D
      Master_int2D
        chebpts
        Lagrange_int1D
        Exp_int1D
  Globaal2Local
    Local2Global
      Master_int2D
        chebpts
        Lagrange_int1D
        Exp_int1D
  PlotElements
```

```
function [array_out] = changem_fea(array_in,new_val,old_val)
% This function replaces values in an array with new values.  It mimics the
% MATLAB function changem.  This function was written with the help of a
% post on
% http://stackoverflow.com/questions/13812656/
% elegant-vectorized-version-of-changem-substitute-values-matlab
% by Rody Oldenhuis.
%
% INPUT:
% array_in = the array in which values will be replaced
% new_val = the new values that will be placed in the array
% old_val = the values in the array that will be replaced
%
% OUTPUT:
% array_out = the input array with old values replaced by new values
```

```
function [x w v] = chebpts(n,d,kind)
% Obtained from
%
% http://www.mathworks.com/matlabcentral/fileexchange/23972-chebfun-v4/
% content/chebfun/chebpts.m
%
% on 8/20/2015 at 17:24.
%
% Copyright (c) 2015, The Chancellor, Masters and Scholars of the University
% of Oxford, and the Chebfun Developers
% All rights reserved.
%
% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided that the following conditions are
% met:
%
%     * Redistributions of source code must retain the above copyright
%       notice, this list of conditions and the following disclaimer.
%     * Redistributions in binary form must reproduce the above copyright
%       notice, this list of conditions and the following disclaimer in
%       the documentation and/or other materials provided with the
%       distribution
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
% AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
% IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
% ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
% LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
% CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
% SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
% INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
% CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
% ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
% POSSIBILITY OF SUCH DAMAGE.
%
%CHEBPTS  Chebyshev points in [-1,1].
%   CHEBPTS(N) returns N Chebyshev points of the 2nd-kind in [-1,1].
%
%   CHEBPTS(N,D), where D is vector of length 2 and N is a scalar integer,
%   scales the nodes and weights for the interval [D(1) D(2)]. If the
%   interval is infinite, the map is chosen to be the default 'unbounded
%   map' with mappref('parinf') = [1 0] and mappref('adaptinf') = 0. If
%   length(D) > 2 and N a vector of length(D)-1, then CHEBPTS returns a
```

```
44 %   column vector of the stacked N(k) Chebyshev points on the subintervals
45 %   D(k:k+1). If length(N) is 1, then D is treated as [D(1) D(end)].
46 %
47 %   [X W] = CHEBPTS(N,D) returns also a row vector of the (scaled) weights
48 %   for Clenshaw—Curtis quadrature (computed using [1]). (For nodes and
49 %   weights of Gauss—Chebyshev quadrature, use [X W] = JACPTS(N,—.5,—.5,D))
50 %
51 %   [X W V] = CHEBPTS(N,D) returns, in addition to X and W, the barycentric
52 %   weights V corresponding to the Chebyshev points X.
53 %
54 %   [X W V] = CHEBPTS(F) returns the Chebyshev nodes and weights
55 %   corresponding to the domain and length of the chebfun F.
56 %
57 %   [X W V] = CHEBPTS(N,KIND) or CHEBPTS(N,D,KIND) returns Chebyshev points
58 %   and weights of the 1st—kind if KIND = 1 and 2nd—kind if KIND = 2
59 %   (default). (Note that if KIND is not suplpied, chebpts will always
60 %   return 2nd—kind points, regardless of the value of 'chebkind' in
61 %   chebfunpref.).
62 %
63 %   See also legpts, jacpts, lagpts, and hermpts.
64
65 %   Copyright 2011 by The University of Oxford and The Chebfun Developers.
66 %   See http://www.maths.ox.ac.uk/chebfun/ for Chebfun information.
67
68 %   [1] J rg Waldvogel, "Fast construction of the Fejr and Clenshaw—Curtis
69 %   quadrature rules", BIT Numerical Mathematics 46 (2006), pp 195—202.
```

```
function [K] = El_springK(xdim,ydim,Gnodes,type,param,U1,Brow,Gvert_ind)
% This function calculates the spring stiffnesses for the edges of a single
% element.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the local node number.
% type = a scalar indicating what type of interpolation is used in the
%    element.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a vector of exponential parameters.  If type = 0, param is
%    ignored.
% U1 = the values of a dependent variable at each node location arranged in
%    a Nx1 vector.
% Brow = the row of the connectivity matrix associated with the element.
% Gvert_ind = the global index of an element vertex.  This function will
%    only calculate stiffnesses along the element edges that share
%    Gvert_ind.
%
% OUTPUT:
% K = a 4x4 matrix of stiffness values.  K(i,j) is the stiffness for the
%    edge between vertices i and j.  K is symmetric and the main diagonal
%    of K is all zeros.

% Generate the Gauss-Legendre quadrature points and weights.
[Qpoints] = Num_LegendrePts(xdim,ydim,type,param);
[p,w] = legpts(Qpoints);
w = w';

% Initialize the output matrix.
K = zeros(4,4);

for Q = 1:1:4
```

```
44
45      switch Q
46          case 1 % Bottom edge
47
48              index = 1:1:xdim;
49              % Skip this edge if it is not needed.
50              if max(Brow(1,index) == Gvert_ind) ~= 1
51                  continue
52              end
53
54              % Quadrature points
55              N1 = p;
56              N2 = -1.*ones(size(p));
57
58              % Components of the Jacobian evaluated at the quadrature points
59              [J11,J12,J21,J22] = ElementJacobian(N1,N2,xdim,ydim,Gnodes);
60
61              % Values to be integrated to find the edge length
62              Ex = J11;
63              Ey = J12;
64
65              % Index of the output matrix where the stiffness value will be
66              % stored
67              K_i = 1;
68              K_j = 2;
69
70          case 2 % Top edge
71
72              index = (xdim.*ydim - xdim + 1):1:(xdim.*ydim);
73              % Skip this edge if it is not needed.
74              if max(Brow(1,index) == Gvert_ind) ~= 1
75                  continue
76              end
77
78              % Quadrature points
79              N1 = p;
80              N2 = ones(size(p));
81
82              % Components of the Jacobian evaluated at the quadrature points
83              [J11,J12,J21,J22] = ElementJacobian(N1,N2,xdim,ydim,Gnodes);
84
85              % Values to be integrated to find the edge length
86              Ex = J11;
```

```
87              Ey = J12;
88
89              % Index of the output matrix where the stiffness value will be
90              % stored
91              K_i = 3;
92              K_j = 4;
93
94          case 3 % Left edge
95
96              index = 1:xdim:(xdim.*ydim - xdim + 1);
97              % Skip this edge if it is not needed.
98              if max(Brow(1,index) == Gvert_ind) ~= 1
99                  continue
100             end
101
102             % Quadrature points
103             N1 = -1.*ones(size(p));
104             N2 = p;
105
106             % Components of the Jacobian evaluated at the quadrature points
107             [J11,J12,J21,J22] = ElementJacobian(N1,N2,xdim,ydim,Gnodes);
108
109             % Values to be integrated to find the edge length
110             Ex = J21;
111             Ey = J22;
112
113             % Index of the output matrix where the stiffness value will be
114             % stored
115             K_i = 1;
116             K_j = 3;
117
118         case 4 % Right edge
119
120             index = xdim:xdim:(xdim.*ydim);
121             % Skip this edge if it is not needed.
122             if max(Brow(1,index) == Gvert_ind) ~= 1
123                 continue
124             end
125
126             % Quadrature points
127             N1 = ones(size(p));
128             N2 = p;
129
```

```
130             % Components of the Jacobian evaluated at the quadrature points
131             [J11,J12,J21,J22] = ElementJacobian(N1,N2,xdim,ydim,Gnodes);
132 
133             % Values to be integrated to find the edge length
134             Ex = J21;
135             Ey = J22;
136 
137             % Index of the output matrix where the stiffness value will be
138             % stored
139             K_i = 2;
140             K_j = 4;
141 
142     end
143 
144     % Calculate the determinant of the Jacobian
145     Jdet = J11.*J22 - J12.*J21;
146 
147     % Calculate components of the Hessian
148     [Hx11,Hx12,Hx22,Hy11,Hy12,Hy22] = ElementHessian(N1,N2,xdim,ydim,Gnodes);
149 
150     % Evaluate the X and Y coordinates of the edge and the second
151     % derivatives of the dependent variable at each natural coordinate
152     % (N1,N2).
153     H11 = zeros(size(N1)); % d2U1_dx2
154     H12 = zeros(size(N1)); % d2U1_dxdy
155     H22 = zeros(size(N1)); % d2U1_dy2
156 
157     for S = 1:1:(xdim.*ydim)
158 
159         % First derivatives of the interpolation function
160         dint_d1 = Master_int2D(N1,N2,xdim,ydim,S,[1,0],type,param,'cheb');
161         dint_d2 = Master_int2D(N1,N2,xdim,ydim,S,[0,1],type,param,'cheb');
162 
163         dint_dx = (J22.*dint_d1 - J12.*dint_d2)./Jdet;
164         dint_dy = (J11.*dint_d2 - J21.*dint_d1)./Jdet;
165 
166         % Second derivatives of the interpolation function
167         d2int_d12 = Master_int2D(N1,N2,xdim,ydim,S,[2,0],type,param,'cheb');
168         d2int_d1d2 = Master_int2D(N1,N2,xdim,ydim,S,[1,1],type,param,'cheb');
169         d2int_d22 = Master_int2D(N1,N2,xdim,ydim,S,[0,2],type,param,'cheb');
170 
171         d2int_dx2 = (J22.^2.*(d2int_d12 - Hx11.*dint_dx - Hy11.*dint_dy) +...
172             J12.^2.*(d2int_d22 - Hx22.*dint_dx - Hy22.*dint_dy) -...
```

```
173             2.*J12.*J22.*(d2int_d1d2 - Hx12.*dint_dx - Hy12.*dint_dy))./...
174             (Jdet.^2);
175
176         d2int_dy2 = (J21.^2.*(d2int_d12 - Hx11.*dint_dx - Hy11.*dint_dy) +...
177             J11.^2.*(d2int_d22 - Hx22.*dint_dx - Hy22.*dint_dy) -...
178             2.*J11.*J21.*(d2int_d1d2 - Hx12.*dint_dx - Hy12.*dint_dy))./...
179             (Jdet.^2);
180
181         d2int_dxdy = (-J21.*J22.*(d2int_d12 - Hx11.*dint_dx - Hy11.*dint_dy) -...
182             J11.*J12.*(d2int_d22 - Hx22.*dint_dx - Hy22.*dint_dy) +...
183             (J11.*J22 + J12.*J21).*...
184             (d2int_d1d2 - Hx12.*dint_dx - Hy12.*dint_dy))./(Jdet.^2);
185
186         % Second derivatives of the dependent variable
187         H11 = H11 + d2int_dx2.*U1(S,1);
188         H12 = H12 + d2int_dxdy.*U1(S,1);
189         H22 = H22 + d2int_dy2.*U1(S,1);
190
191     end
192
193     % Calculate the components of the eigenvectors of the Hessian.
194     V11 = H11 - H22 - sqrt(H11.^2 - 2.*H11.*H22 + 4.*H12.^2 + H22.^2);
195     V12 = H11 - H22 + sqrt(H11.^2 - 2.*H11.*H22 + 4.*H12.^2 + H22.^2);
196     V21 = 2.*H12;
197     V22 = 2.*H12;
198
199     % Calculate the magnitude of the eigenvectors.
200     mag1 = sqrt(V11.^2 + V21.^2);
201     mag2 = sqrt(V12.^2 + V22.^2);
202
203     % Calculate the components of the unit eigenvectors.
204     if min(abs(mag1)) ~= 0
205         V11 = V11./mag1;
206         V21 = V21./mag1;
207     end
208
209     if min(abs(mag2)) ~= 0
210         V12 = V12./mag2;
211         V22 = V22./mag2;
212     end
213
214     % Calculate the eigenvalues of the Hessian.
215     Lambda1 = (H11 + H22 - sqrt(H11.^2 - 2.*H11.*H22 + 4.*H12.^2 + H22.^2))./2;
```

```
216     Lambda2 = (H11 + H22 + sqrt(H11.^2 - 2.*H11.*H22 + 4.*H12.^2 + H22.^2))./2;
217
218     % Calculate the components of the Hessian using the absolute value of
219     % the eigenvalues.  Square each component of the Hessian.
220     Hbar11 = (V11.^2.*abs(Lambda1) + V12.^2.*abs(Lambda2)).^2;
221     Hbar12 = (V11.*V21.*abs(Lambda1) + V12.*V22.*abs(Lambda2)).^2;
222     Hbar22 = (V21.^2.*abs(Lambda1) + V22.^2.*abs(Lambda2)).^2;
223
224     % Spring stiffness for the element edge.
225     Spr_k = sum(sum(w.*Jdet.*...
226         sqrt(Ex.^2.*Hbar11 + 2.*Ex.*Ey.*Hbar12 + Ey.^2.*Hbar22))./...
227         sum(sum(w.*Jdet.*sqrt(Ex.^2 + Ey.^2)));
228
229     % Put the spring stiffness for the edge in the appropriate location in
230     % the output matrix.
231     K(K_i,K_j) = Spr_k;
232     K(K_j,K_i) = Spr_k;
233
234 end
235
236 end
```

```
function [Nodes] = Element_Mesh(Gvert,xdim,ydim,Snodes)
% This function generates the global coordinates of the element nodes for a
% single element.  The nodes are placed at Chebyshev points of the
% 2nd-kind.
%
% INPUT:
% Gvert =  Global coordinates of the element vertices arranged in a 4x2
%    matrix.  Column 1 is the global x-coordinates.  Column 2 is the global
%    y-coordinates.  Row 1 is the lower left node, Row 2 is the lower right
%    node, row 3 is the upper left node, row 4 is the upper right node.
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% Snodes = a matrix of node coordinates along one side of the element.  The
%    first and last row of the matrix must be a point contained in Gvert.
%    Each side node is assumed to lie between the verticies that are above
%    and below it in Snodes.  If Snodes are not specified, the element is
%    assumed to have straight sides.  The number of rows in Snodes must
%    equal either xdim or ydim.  If Snodes are not specified, [] must be
%    used in place of Snodes when this function is called.
% OUTPUT:
% Nodes = (xdim*ydim)x2 matrix of node coordinates in the global coordinate
%    system.  Column 1 contains the x coordinates.  Column 2 contains the
%    y coordinates.  The row index is equal to the local node number.

% Initialize the output variable.
Nodes = ones(xdim.*ydim,2).*NaN;

% Assume the edges are straight and fill in the coordinates of the nodes.
% If one edge is not straight (Snodes is not empty), the coordinates of
% that edge will be replaced by Snodes later.

% Bottom edge
Nodes([1:1:xdim],1) = chebpts(xdim,[Gvert(1,1) Gvert(2,1)]);

if Gvert(1,2) == Gvert(2,2)

    Nodes([1:1:xdim],2) = ones(xdim,1).*Gvert(1,2);

else

    Nodes([1:1:xdim],2) = chebpts(xdim,[Gvert(1,2) Gvert(2,2)]);
```

```
44
45 end
46
47 % Left edge
48 Nodes([1:xdim:end],2) = chebpts(ydim,[Gvert(1,2) Gvert(3,2)]);
49
50 if Gvert(1,1) == Gvert(3,1)
51
52     Nodes([1:xdim:end],1) = ones(ydim,1).*Gvert(1,1);
53
54 else
55
56     Nodes([1:xdim:end],1) = chebpts(ydim,[Gvert(1,1) Gvert(3,1)]);
57
58 end
59
60 % Right edge
61 Nodes([xdim:xdim:end],2) = chebpts(ydim,[Gvert(2,2) Gvert(4,2)]);
62
63 if Gvert(2,1) == Gvert(4,1)
64
65     Nodes([xdim:xdim:end],1) = ones(ydim,1).*Gvert(2,1);
66
67 else
68
69     Nodes([xdim:xdim:end],1) = chebpts(ydim,[Gvert(2,1) Gvert(4,1)]);
70
71 end
72
73 % Top edge
74 Nodes([(xdim.*(ydim-1)+1):1:end],1) = chebpts(xdim,[Gvert(3,1) Gvert(4,1)]);
75
76 if Gvert(3,2) == Gvert(4,2)
77
78     Nodes([(xdim.*(ydim-1)+1):1:end],2) = ones(xdim,1).*Gvert(3,2);
79
80 else
81
82     Nodes([(xdim.*(ydim-1)+1):1:end],2) =...
83         chebpts(xdim,[Gvert(3,2) Gvert(4,2)]);
84
85 end
86
```

```
% If Snodes is not empty, replace the coordinates of the appropriate edge
% with Snodes.
if ~isempty(Snodes)

    [Svert1,~] = find((Gvert(:,1) == Snodes(1,1)) &...
        (Gvert(:,2) == Snodes(1,2)));
    [Svert2,~] = find((Gvert(:,1) == Snodes(end,1)) &...
        (Gvert(:,2) == Snodes(end,2)));

    if Svert1 > Svert2

        Snodes([1:1:end],:) = Snodes([end:-1:1],:);
        Svert_temp = Svert1;
        Svert1 = Svert2;
        Svert2 = Svert_temp;

    end

    switch Svert1

        case 1

            if Svert2 == 2

                % Bottom edge
                Nodes([1:1:xdim],:) = Snodes;

            else

                % Left edge
                Nodes([1:xdim:end],:) = Snodes;

            end

        case 2

            % Right edge
            Nodes([xdim:xdim:end],:) = Snodes;

        case 3

            % Top edge
            Nodes([(xdim.*(ydim-1)+1):1:end],:) = Snodes;
```

```
130
131     end
132 end
133
134 % Define the coordinates of the interior nodes
135 % x coordinates
136 for row = 2:1:(ydim-1)
137
138     End_ind = row.*xdim;
139     Start_ind = End_ind - xdim + 1;
140
141     if Nodes(Start_ind,1) == Nodes(End_ind,1)
142
143         Nodes([Start_ind:1:End_ind],1) = ones(xdim,1).*Nodes(Start_ind,1);
144
145     else
146
147         Nodes([Start_ind:1:End_ind],1) =...
148             chebpts(xdim,[Nodes(Start_ind,1) Nodes(End_ind,1)]);
149
150     end
151 end
152
153 % y coordinates
154 for col = 2:1:(xdim-1)
155
156     End_ind = xdim.*(ydim - 1) + col;
157     Start_ind = col;
158
159     if Nodes(Start_ind,2) == Nodes(End_ind,2)
160
161         Nodes([Start_ind:xdim:End_ind],2) =...
162             ones(ydim,1).*Nodes(Start_ind,2);
163
164     else
165
166         Nodes([Start_ind:xdim:End_ind],2) =...
167             chebpts(ydim,[Nodes(Start_ind,2) Nodes(End_ind,2)]);
168
169     end
170 end
171 end
```

```
function [Nodes] = Element_Mesh_eql(Gvert,xdim,ydim,Snodes)
% This function generates the global coordinates of the element nodes.  The
% nodes are placed at equal intervals.
%
% INPUT:
% Gvert =  Global coordinates of the element vertices arranged in a 4x2
%    matrix.  Column 1 is the global x-coordinates.  Column 2 is the global
%    y-coordinates.  Row 1 is the lower left node, Row 2 is the lower right
%    node, row 3 is the upper left node, row 4 is the upper right node.
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% Snodes = a matrix of node coordinates along one side of the element.  The
%    first and last row of the matrix must be a point contained in Gvert.
%    Each side node is assumed to lie between the verticies that are above
%    and below it in Snodes.  If Snodes are not specified, the element is
%    assumed to have straight sides.  The number of rows in Snodes must
%    equal either xdim or ydim.  If Snodes are not specified, [] must be
%    used in place of Snodes when this function is called.
% OUTPUT:
% Nodes = (xdim*ydim)x2 matrix of node coordinates in the global coordinate
%    system.  Column 1 contains the x coordinates.  Column 2 contains the
%    y coordinates.  The row index is equal to the local node number.

% Initialize the output variable.
Nodes = ones(xdim.*ydim,2).*NaN;

% Assume the edges are straight and fill in the coordinates of the nodes.
% If one edge is not straight (Snodes is not empty), the coordinates of
% that edge will be replaced by Snodes later.

% Bottom edge
Nodes([1:1:xdim],1) = linspace(Gvert(1,1),Gvert(2,1),xdim);

if Gvert(1,2) == Gvert(2,2)

    Nodes([1:1:xdim],2) = ones(xdim,1).*Gvert(1,2);

else

    Nodes([1:1:xdim],2) = linspace(Gvert(1,2),Gvert(2,2),xdim);

```

```
44 end
45
46 % Left edge
47 Nodes([1:xdim:end],2) = linspace(Gvert(1,2),Gvert(3,2),ydim);
48
49 if Gvert(1,1) == Gvert(3,1)
50
51     Nodes([1:xdim:end],1) = ones(ydim,1).*Gvert(1,1);
52
53 else
54
55     Nodes([1:xdim:end],1) = linspace(Gvert(1,1),Gvert(3,1),ydim);
56
57 end
58
59 % Right edge
60 Nodes([xdim:xdim:end],2) = linspace(Gvert(2,2),Gvert(4,2),ydim);
61
62 if Gvert(2,1) == Gvert(4,1)
63
64     Nodes([xdim:xdim:end],1) = ones(ydim,1).*Gvert(2,1);
65
66 else
67
68     Nodes([xdim:xdim:end],1) = linspace(Gvert(2,1),Gvert(4,1),ydim);
69
70 end
71
72 % Top edge
73 Nodes([(xdim.*(ydim-1)+1):1:end],1) = linspace(Gvert(3,1),Gvert(4,1),xdim);
74
75 if Gvert(3,2) == Gvert(4,2)
76
77     Nodes([(xdim.*(ydim-1)+1):1:end],2) = ones(xdim,1).*Gvert(3,2);
78
79 else
80
81     Nodes([(xdim.*(ydim-1)+1):1:end],2) =...
82         linspace(Gvert(3,2),Gvert(4,2),xdim);
83
84 end
85
86 % If Snodes is not empty, replace the coordinates of the appropriate edge
```

```
87 % with Snodes.
88 if ~isempty(Snodes)
89
90     [Svert1,~] = find((Gvert(:,1) == Snodes(1,1)) &...
91         (Gvert(:,2) == Snodes(1,2)));
92     [Svert2,~] = find((Gvert(:,1) == Snodes(end,1)) &...
93         (Gvert(:,2) == Snodes(end,2)));
94
95     if Svert1 > Svert2
96
97         Snodes([1:1:end],:) = Snodes([end:-1:1],:);
98         Svert_temp = Svert1;
99         Svert1 = Svert2;
100        Svert2 = Svert_temp;
101
102    end
103
104    switch Svert1
105
106        case 1
107
108            if Svert2 == 2
109
110                % Bottom edge
111                Nodes([1:1:xdim],:) = Snodes;
112
113            else
114
115                % Left edge
116                Nodes([1:xdim:end],:) = Snodes;
117
118            end
119
120        case 2
121
122            % Right edge
123            Nodes([xdim:xdim:end],:) = Snodes;
124
125        case 3
126
127            % Top edge
128            Nodes([(xdim.*(ydim-1)+1):1:end],:) = Snodes;
129
```

```
130     end
131 end
132 
133 % Define the coordinates of the interior nodes
134 % x coordinates
135 for row = 2:1:(ydim-1)
136 
137     End_ind = row.*xdim;
138     Start_ind = End_ind - xdim + 1;
139 
140     if Nodes(Start_ind,1) == Nodes(End_ind,1)
141 
142         Nodes([Start_ind:1:End_ind],1) = ones(xdim,1).*Nodes(Start_ind,1);
143 
144     else
145 
146         Nodes([Start_ind:1:End_ind],1) =...
147             linspace(Nodes(Start_ind,1),Nodes(End_ind,1),xdim);
148 
149     end
150 end
151 
152 % y coordinates
153 for col = 2:1:(xdim-1)
154 
155     End_ind = xdim.*(ydim - 1) + col;
156     Start_ind = col;
157 
158     if Nodes(Start_ind,2) == Nodes(End_ind,2)
159 
160         Nodes([Start_ind:xdim:End_ind],2) =...
161             ones(ydim,1).*Nodes(Start_ind,2);
162 
163     else
164 
165         Nodes([Start_ind:xdim:End_ind],2) =...
166             linspace(Nodes(Start_ind,2),Nodes(End_ind,2),ydim);
167 
168     end
169 end
170 end
```

```
function [Xout,Yout,Uout,dUout_dx,dUout_dy,d2Uout_dx2,d2Uout_dxdy,...
    d2Uout_dy2,Jdet] = Element2Grid(xdim,ydim,Gnodes,type,param,U,inc)
% This function calculates the value of a dependent variable and
% derivatives of the dependent variable in a single element.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the local node number.
% type = a scalar indicating what type of interpolation is used in the
%    element.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a vector of exponential parameters.  If type = 0, param is
%    ignored.
% U = the values of a dependent variable at each node location arranged in
%    a Nx1 vector.
% inc = the increment between local coordinates where the value of the
%    dependent variable and its derivatives will be calculated.
%
% OUTPUT:
% Xout = the global x-coordinate corresponding to each local coordinate.
% Yout = the global y-coordinate corresponding to each local coordinate.
% Uout = the dependent variable at each local coordinate.
% dUout_dx = the derivative of the dependent variable with respect to x at
%    each local coordinate.
% dUout_dy = the derivative of the dependent variable with respect to y at
%    each local coordinate.
% d2Uout_dx2 = the second derivative of the dependent variable with respect
%    to x at each local coordinate.
% d2Uout_dxdy = the second derivative of the dependent variable with
%    respect to x and y at each local coordinate.
% d2Uout_dy2 = the second derivative of the dependent variable with respect
%    to y at each local coordinate.
% Jdet = the determinant of the Jacobian at each local coordinate.
```

```
44
45 % Generate the local coordinates.
46 [N1,N2] = meshgrid(-1:inc:1,-1:inc:1);
47
48 % Calculate the number of nodes in the element.
49 n_nodes = xdim.*ydim;
50
51 % Calculate components of the Jacobian.
52 [J11,J12,J21,J22] = ElementJacobian(N1,N2,xdim,ydim,Gnodes);
53
54 % Calculate components of the Hessian.
55 [Hx11,Hx12,Hx22,Hy11,Hy12,Hy22] = ElementHessian(N1,N2,xdim,ydim,Gnodes);
56
57 % Define the determinant of the Jacobian.
58 Jdet = J11.*J22 - J12.*J21;
59
60 % Initialize the output variables.
61 Xout = zeros(size(N1));
62 Yout = zeros(size(N1));
63 Uout = zeros(size(N1));
64 dUout_dx = zeros(size(N1));
65 dUout_dy = zeros(size(N1));
66 d2Uout_dx2 = zeros(size(N1));
67 d2Uout_dxdy = zeros(size(N1));
68 d2Uout_dy2 = zeros(size(N1));
69
70 % Evaluate the dependent variable and its derivatives at each point (N1,N2).
71 for k = 1:1:n_nodes
72
73     % Interpolation function
74     int = Master_int2D(N1,N2,xdim,ydim,k,[0,0],type,param,'cheb');
75
76     % Dependent variable
77     Uout = Uout + int.*U(k,1);
78
79     % Global coordinates
80     Xout = Xout + int.*Gnodes(k,1);
81     Yout = Yout + int.*Gnodes(k,2);
82
83     % First derivatives of the interpolation function
84     dint_d1 = Master_int2D(N1,N2,xdim,ydim,k,[1,0],type,param,'cheb');
85     dint_d2 = Master_int2D(N1,N2,xdim,ydim,k,[0,1],type,param,'cheb');
86
```

```
87      dint_dx = (J22.*dint_d1 - J12.*dint_d2)./Jdet;
88      dint_dy = (J11.*dint_d2 - J21.*dint_d1)./Jdet;
89
90      % First derivatives of the dependent variable
91      dUout_dx = dUout_dx + dint_dx.*U(k,1);
92      dUout_dy = dUout_dy + dint_dy.*U(k,1);
93
94      % Second derivatives of the interpolation function
95      d2int_d12 = Master_int2D(N1,N2,xdim,ydim,k,[2,0],type,param,'cheb');
96      d2int_d1d2 = Master_int2D(N1,N2,xdim,ydim,k,[1,1],type,param,'cheb');
97      d2int_d22 = Master_int2D(N1,N2,xdim,ydim,k,[0,2],type,param,'cheb');
98
99      d2int_dx2 = (J22.^2.*(d2int_d12 - Hx11.*dint_dx - Hy11.*dint_dy) +...
100         J12.^2.*(d2int_d22 - Hx22.*dint_dx - Hy22.*dint_dy) -...
101         2.*J12.*J22.*(d2int_d1d2 - Hx12.*dint_dx - Hy12.*dint_dy))./...
102         (Jdet.^2);
103
104     d2int_dy2 = (J21.^2.*(d2int_d12 - Hx11.*dint_dx - Hy11.*dint_dy) +...
105         J11.^2.*(d2int_d22 - Hx22.*dint_dx - Hy22.*dint_dy) -...
106         2.*J11.*J21.*(d2int_d1d2 - Hx12.*dint_dx - Hy12.*dint_dy))./...
107         (Jdet.^2);
108
109     d2int_dxdy = (-J21.*J22.*(d2int_d12 - Hx11.*dint_dx - Hy11.*dint_dy) -...
110         J11.*J12.*(d2int_d22 - Hx22.*dint_dx - Hy22.*dint_dy) +...
111         (J11.*J22 + J12.*J21).*...
112         (d2int_d1d2 - Hx12.*dint_dx - Hy12.*dint_dy))./(Jdet.^2);
113
114     % Second derivatives of the dependent variable
115     d2Uout_dx2 = d2Uout_dx2 + d2int_dx2.*U(k,1);
116     d2Uout_dxdy = d2Uout_dxdy + d2int_dxdy.*U(k,1);
117     d2Uout_dy2 = d2Uout_dy2 + d2int_dy2.*U(k,1);
118
119 end
120
121 end
```

```
function [Hx11,Hx12,Hx22,Hy11,Hy12,Hy22] =...
    ElementHessian(N1,N2,xdim,ydim,Gnodes)
% This function calculates the Hessian for a 2-dimensional finite element.
%
% INPUT:
% N1 = horizontal coordinate in the local coordinate system where the
%    Hessian is to be evaluated.  N1 may be a scalar, vector, or matrix.
%    If N1 is a vector or a matrix it must have the same dimensions as N2.
% N2 = vertical coordinate in the local coordinate system where the Hessian
%    is to be evaluated.  N2 may be a scalar, vector, or matrix.  If N2 is
%    a vector or a matrix it must have the same dimensions as N1.
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the local node number.
%
% OUTPUT:
% Hx11,Hx12,Hx22 = the components of the 2x2 Hessian of x with respect to
%    the local coordinates of a master element.  Each component is
%    evaluated at each local coordinate in the inputs.
% Hy11,Hy12,Hy22 = the components of the 2x2 Hessian of y with respect to
%    the local coordinates of a master element.  Each component is
%    evaluated at each local coordinate in the inputs.

% N is the number of nodes in the element.
N = size(Gnodes,1);

% Initialize the output variables.
Hx11 = zeros(size(N1));
Hx12 = zeros(size(N1));
Hx22 = zeros(size(N1));

Hy11 = zeros(size(N1));
Hy12 = zeros(size(N1));
Hy22 = zeros(size(N1));

for k = 1:1:N

    Hx11 = Hx11 +...
        Master_int2D(N1,N2,xdim,ydim,k,[2,0],0,[],'cheb').*Gnodes(k,1);
```

```
44
45      Hx12 = Hx12 +...
46          Master_int2D(N1,N2,xdim,ydim,k,[1,1],0,[],'cheb').*Gnodes(k,1);
47
48      Hx22 = Hx22 +...
49          Master_int2D(N1,N2,xdim,ydim,k,[0,2],0,[],'cheb').*Gnodes(k,1);
50
51      Hy11 = Hy11 +...
52          Master_int2D(N1,N2,xdim,ydim,k,[2,0],0,[],'cheb').*Gnodes(k,2);
53
54      Hy12 = Hy12 +...
55          Master_int2D(N1,N2,xdim,ydim,k,[1,1],0,[],'cheb').*Gnodes(k,2);
56
57      Hy22 = Hy22 +...
58          Master_int2D(N1,N2,xdim,ydim,k,[0,2],0,[],'cheb').*Gnodes(k,2);
59
60  end
61  end
```

```
function [J11,J12,J21,J22] = ElementJacobian(N1,N2,xdim,ydim,Gnodes)
% This function calculates the Jacobian for a 2 dimensional finite element.
%
% INPUT:
% N1 = horizontal coordinate in the local coordinate system where the
%    Hessian is to be evaluated.  x may be a scalar, vector, or matrix.  If
%    x is a vector or a matrix it must have the same dimensions as y.
% N2 = vertical coordinate in the local coordinate system where the Hessian
%    is to be evaluated.  y may be a scalar, vector, or matrix.  If y is a
%    vector or a matrix it must have the same dimensions as x.
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the local node number.
%
% OUTPUT:
% [J11,J12,J21,J22] are the components of the 2x2 Jacobian matrix.  Each
%    component is evaluated at each local coordinate in the inputs.

% N is the number of nodes in the element.
N = size(Gnodes,1);

% Initialize the output variables.
J11 = zeros(size(N1));
J12 = zeros(size(N1));
J21 = zeros(size(N1));
J22 = zeros(size(N1));

for k = 1:1:N
    J11 = J11 +...
        Master_int2D(N1,N2,xdim,ydim,k,[1,0],0,[],'cheb').*Gnodes(k,1);
    J12 = J12 +...
        Master_int2D(N1,N2,xdim,ydim,k,[1,0],0,[],'cheb').*Gnodes(k,2);
    J21 = J21 +...
        Master_int2D(N1,N2,xdim,ydim,k,[0,1],0,[],'cheb').*Gnodes(k,1);
    J22 = J22 +...
        Master_int2D(N1,N2,xdim,ydim,k,[0,1],0,[],'cheb').*Gnodes(k,2);
end
end
```

```
function [F] = Euler_F(xdim,ydim,Gnodes,type,param,U,dt)
% This function generates the right side vector in [K]{U}={F} for an
% element.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the local node number.
% type = a scalar indicating what type of interpolation is used in the
%    element.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a vector of exponential parameters.  If type = 0, param is
%    ignored.
% U = the values of the dependent variables at each node from either an
%    initial guess or the previous solution iteration.
% dt = time step.
%
% OUTPUT:
% F = right side vector for the element.

% Generate the Gauss-Legendre quadrature points and weights.
[X,Y,W] = LegendrePts2D(xdim,ydim,type,param);

% N is the number of nodes in the element.
N = xdim.*ydim;

% Ratio of specific heats for air, assumed to be constant.
gamma = 1.4;

% Calculate components of the Jacobian.
[J11,J12,J21,J22] = ElementJacobian(X,Y,xdim,ydim,Gnodes);

% Calculate the determinant of the Jacobian.
Jdet = J11.*J22 - J12.*J21;
```

```
44
45 % Split the input vector U into separate vectors for each fluid property.
46 % These are the flow properties at the element nodes.
47 rho_k = U(1:4:end,1);
48 vx_k = U(2:4:end,1);
49 vy_k = U(3:4:end,1);
50 P_k = U(4:4:end,1);
51
52 % Initialize vectors to contain the fluid property values and the
53 % derivatives of the fluid properties at each quadrature point.
54 rho = zeros(size(X));
55 drho_dx = zeros(size(X));
56 drho_dy = zeros(size(X));
57
58 vx = zeros(size(X));
59 dvx_dx = zeros(size(X));
60 dvx_dy = zeros(size(X));
61
62 vy = zeros(size(X));
63 dvy_dx = zeros(size(X));
64 dvy_dy = zeros(size(X));
65
66 P = zeros(size(X));
67 dP_dx = zeros(size(X));
68 dP_dy = zeros(size(X));
69
70 % Evaluate the fluid properties and the derivatives of the fluid properties
71 % at each quadrature point.
72 for k = 1:1:N
73
74     int = Master_int2D(X,Y,xdim,ydim,k,[0,0],type,param,'cheb');
75     dint_d1 = Master_int2D(X,Y,xdim,ydim,k,[1,0],type,param,'cheb');
76     dint_d2 = Master_int2D(X,Y,xdim,ydim,k,[0,1],type,param,'cheb');
77
78     rho = rho + int.*rho_k(k,1);
79     drho_dx = drho_dx + (J22.*dint_d1 - J12.*dint_d2).*rho_k(k,1);
80     drho_dy = drho_dy + (J11.*dint_d2 - J21.*dint_d1).*rho_k(k,1);
81
82     vx = vx + int.*vx_k(k,1);
83     dvx_dx = dvx_dx + (J22.*dint_d1 - J12.*dint_d2).*vx_k(k,1);
84     dvx_dy = dvx_dy + (J11.*dint_d2 - J21.*dint_d1).*vx_k(k,1);
85
86     vy = vy + int.*vy_k(k,1);
```

```
87      dvy_dx = dvy_dx + (J22.*dint_d1 - J12.*dint_d2).*vy_k(k,1);
88      dvy_dy = dvy_dy + (J11.*dint_d2 - J21.*dint_d1).*vy_k(k,1);
89
90      P = P + int.*P_k(k,1);
91      dP_dx = dP_dx + (J22.*dint_d1 - J12.*dint_d2).*P_k(k,1);
92      dP_dy = dP_dy + (J11.*dint_d2 - J21.*dint_d1).*P_k(k,1);
93  end
94
95  drho_dx = drho_dx./Jdet;
96  drho_dy = drho_dy./Jdet;
97
98  dvx_dx = dvx_dx./Jdet;
99  dvx_dy = dvx_dy./Jdet;
100
101 dvy_dx = dvy_dx./Jdet;
102 dvy_dy = dvy_dy./Jdet;
103
104 dP_dx = dP_dx./Jdet;
105 dP_dy = dP_dy./Jdet;
106
107 % Initialize vectors to contain the values of the output vector components.
108 F1 = zeros(N,1);
109 F2 = zeros(N,1);
110 F3 = zeros(N,1);
111 F4 = zeros(N,1);
112
113 % Calculate the components of the output vector.
114 for i = 1:1:N
115
116     int_i = Master_int2D(X,Y,xdim,ydim,i,[0,0],type,param,'cheb');
117     dint_d1_i = Master_int2D(X,Y,xdim,ydim,i,[1,0],type,param,'cheb');
118     dint_d2_i = Master_int2D(X,Y,xdim,ydim,i,[0,1],type,param,'cheb');
119
120     dint_dx_i = (J22.*dint_d1_i - J12.*dint_d2_i)./Jdet;
121     dint_dy_i = (J11.*dint_d2_i - J21.*dint_d1_i)./Jdet;
122
123     F1(i,1) = sum(sum(W.*Jdet.*((rho./dt + rho.*dvx_dx + vx.*drho_dx +...
124         rho.*dvy_dy + vy.*drho_dy).*(int_i./dt + dvx_dx.*int_i +...
125         vx.*dint_dx_i + dvy_dy.*int_i + vy.*dint_dy_i) - (vx./dt +...
126         vx.*dvx_dx + vy.*dvx_dy - (1./rho).*dP_dx).*...
127         ((1./rho.^2).*dP_dx.*int_i) - (vy./dt + vx.*dvy_dx +...
128         vy.*dvy_dy - (1./rho).*dP_dy).*((1./rho.^2).*dP_dy.*int_i))));
129
```

```
130     F2(i,1) = sum(sum(W.*Jdet.*((rho./dt + rho.*dvx_dx + vx.*drho_dx +...
131         rho.*dvy_dy + vy.*drho_dy).*(drho_dx.*int_i + rho.*dint_dx_i) +...
132         (vx./dt + vx.*dvx_dx + vy.*dvx_dy - (1./rho).*dP_dx).*...
133         (int_i./dt + dvx_dx.*int_i + vx.*dint_dx_i + vy.*dint_dy_i) +...
134         (vy./dt + vx.*dvy_dx + vy.*dvy_dy - (1./rho).*dP_dy).*...
135         (dvy_dx.*int_i) + (P./dt + vx.*dP_dx + vy.*dP_dy +...
136         gamma.*P.*dvx_dx + gamma.*P.*dvy_dy).*(dP_dx.*int_i +...
137         gamma.*P.*dint_dx_i))));
138
139     F3(i,1) = sum(sum(W.*Jdet.*((rho./dt + rho.*dvx_dx + vx.*drho_dx +...
140         rho.*dvy_dy + vy.*drho_dy).*(drho_dy.*int_i + rho.*dint_dy_i) +...
141         (vx./dt + vx.*dvx_dx + vy.*dvx_dy - (1./rho).*dP_dx).*...
142         (dvx_dy.*int_i) + (vy./dt + vx.*dvy_dx + vy.*dvy_dy -...
143         (1./rho).*dP_dy).*(int_i./dt + vx.*dint_dx_i + dvy_dy.*int_i +...
144         vy.*dint_dy_i) + (P./dt + vx.*dP_dx + vy.*dP_dy +...
145         gamma.*P.*dvx_dx + gamma.*P.*dvy_dy).*(dP_dy.*int_i +...
146         gamma.*P.*dint_dy_i))));
147
148     F4(i,1) = sum(sum(W.*Jdet.*((vx./dt + vx.*dvx_dx + vy.*dvx_dy -...
149         (1./rho).*dP_dx).*((1./rho).*dint_dx_i) + (vy./dt + vx.*dvy_dx +...
150         vy.*dvy_dy - (1./rho).*dP_dy).*((1./rho).*dint_dy_i) +...
151         (P./dt + vx.*dP_dx + vy.*dP_dy + gamma.*P.*dvx_dx +...
152         gamma.*P.*dvy_dy).*(int_i./dt + gamma.*dvx_dx.*int_i +...
153         vx.*dint_dx_i + gamma.*dvy_dy.*int_i + vy.*dint_dy_i))));
154
155 end
156
157 % Assemble the components of the output vector.
158 F = zeros(4.*N,1);
159
160 F(1:4:end,1) = F1;
161 F(2:4:end,1) = F2;
162 F(3:4:end,1) = F3;
163 F(4:4:end,1) = F4;
164
165 end
```

```
function [K] = Euler_K(xdim,ydim,Gnodes,type,param,U,dt)
% This function generates the coefficient matrix in [K]{U}={F} for an
% element.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the local node number.
% type = a scalar indicating what type of interpolation is used in the
%    element.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a vector of exponential parameters.  If type = 0, param is
%    ignored.
% U = the values of the dependent variables at each node from either an
%    initial guess or the previous solution iteration.
% dt = time step.
%
% OUTPUT:
% K = coefficient matrix for the element.

% Generate the Gauss-Legendre quadrature points and weights.
[X,Y,W] = LegendrePts2D(xdim,ydim,type,param);

% N is the number of nodes in the element.
N = xdim.*ydim;

% Ratio of specific heats for air, assumed to be constant.
gamma = 1.4;

% Calculate components of the Jacobian.
[J11,J12,J21,J22] = ElementJacobian(X,Y,xdim,ydim,Gnodes);

% Calculate the determinant of the Jacobian.
Jdet = J11.*J22 - J12.*J21;
```

```
44
45 % Split the input vector U into separate vectors for each fluid property.
46 % These are the flow properties at the element nodes.
47 rho_k = U(1:4:end,1);
48 vx_k = U(2:4:end,1);
49 vy_k = U(3:4:end,1);
50 P_k = U(4:4:end,1);
51
52 % Initialize vectors to contain the fluid property values and the
53 % derivatives of the fluid properties at each quadrature point.
54 rho = zeros(size(X));
55 drho_dx = zeros(size(X));
56 drho_dy = zeros(size(X));
57
58 vx = zeros(size(X));
59 dvx_dx = zeros(size(X));
60 dvx_dy = zeros(size(X));
61
62 vy = zeros(size(X));
63 dvy_dx = zeros(size(X));
64 dvy_dy = zeros(size(X));
65
66 P = zeros(size(X));
67 dP_dx = zeros(size(X));
68 dP_dy = zeros(size(X));
69
70 % Evaluate the fluid properties and the derivatives of the fluid properties
71 % at each quadrature point.
72 for k = 1:1:N
73
74     int = Master_int2D(X,Y,xdim,ydim,k,[0,0],type,param,'cheb');
75     dint_d1 = Master_int2D(X,Y,xdim,ydim,k,[1,0],type,param,'cheb');
76     dint_d2 = Master_int2D(X,Y,xdim,ydim,k,[0,1],type,param,'cheb');
77
78     rho = rho + int.*rho_k(k,1);
79     drho_dx = drho_dx + (J22.*dint_d1 - J12.*dint_d2).*rho_k(k,1);
80     drho_dy = drho_dy + (J11.*dint_d2 - J21.*dint_d1).*rho_k(k,1);
81
82     vx = vx + int.*vx_k(k,1);
83     dvx_dx = dvx_dx + (J22.*dint_d1 - J12.*dint_d2).*vx_k(k,1);
84     dvx_dy = dvx_dy + (J11.*dint_d2 - J21.*dint_d1).*vx_k(k,1);
85
86     vy = vy + int.*vy_k(k,1);
```

```
87      dvy_dx = dvy_dx + (J22.*dint_d1 - J12.*dint_d2).*vy_k(k,1);
88      dvy_dy = dvy_dy + (J11.*dint_d2 - J21.*dint_d1).*vy_k(k,1);
89
90      P = P + int.*P_k(k,1);
91      dP_dx = dP_dx + (J22.*dint_d1 - J12.*dint_d2).*P_k(k,1);
92      dP_dy = dP_dy + (J11.*dint_d2 - J21.*dint_d1).*P_k(k,1);
93  end
94
95  drho_dx = drho_dx./Jdet;
96  drho_dy = drho_dy./Jdet;
97  dvx_dx = dvx_dx./Jdet;
98  dvx_dy = dvx_dy./Jdet;
99  dvy_dx = dvy_dx./Jdet;
100 dvy_dy = dvy_dy./Jdet;
101 dP_dx = dP_dx./Jdet;
102 dP_dy = dP_dy./Jdet;
103
104 % Initialize vectors to contain the values of the coefficient matrix
105 % components.  Only the upper triangular components are calculated because
106 % the coefficient matrix is symmetric.
107 K11 = zeros(N,N);
108 K12 = zeros(N,N);
109 K13 = zeros(N,N);
110 K14 = zeros(N,N);
111 K22 = zeros(N,N);
112 K23 = zeros(N,N);
113 K24 = zeros(N,N);
114 K33 = zeros(N,N);
115 K34 = zeros(N,N);
116 K44 = zeros(N,N);
117
118 % Calculate the components of the coefficient matrix.
119 for i = 1:1:N
120
121     int_i = Master_int2D(X,Y,xdim,ydim,i,[0,0],type,param,'cheb');
122     dint_d1_i = Master_int2D(X,Y,xdim,ydim,i,[1,0],type,param,'cheb');
123     dint_d2_i = Master_int2D(X,Y,xdim,ydim,i,[0,1],type,param,'cheb');
124
125     dint_dx_i = (J22.*dint_d1_i - J12.*dint_d2_i)./Jdet;
126     dint_dy_i = (J11.*dint_d2_i - J21.*dint_d1_i)./Jdet;
127
128     dR1_drho_i = (1./dt + dvx_dx + dvy_dy).*int_i +...
129         vx.*dint_dx_i + vy.*dint_dy_i;
```

```
130
131     dR1_dvx_i = drho_dx.*int_i + rho.*dint_dx_i;
132
133     dR1_dvy_i = drho_dy.*int_i + rho.*dint_dy_i;
134
135     dR2_drho_i = (-1./rho.^2).*dP_dx.*int_i;
136
137     dR2_dvx_i = (1./dt + dvx_dx).*int_i +...
138         vx.*dint_dx_i + vy.*dint_dy_i;
139
140     dR2_dvy_i = dvx_dy.*int_i;
141
142     dR2_dP_i = (1./rho).*dint_dx_i;
143
144     dR3_drho_i = (-1./rho.^2).*dP_dy.*int_i;
145
146     dR3_dvx_i = dvy_dx.*int_i;
147
148     dR3_dvy_i = (1./dt + dvy_dy).*int_i +...
149         vx.*dint_dx_i + vy.*dint_dy_i;
150
151     dR3_dP_i = (1./rho).*dint_dy_i;
152
153     dR4_dvx_i = dP_dx.*int_i + gamma.*P.*dint_dx_i;
154
155     dR4_dvy_i = dP_dy.*int_i + gamma.*P.*dint_dy_i;
156
157     dR4_dP_i = (1./dt + gamma.*dvx_dx + gamma.*dvy_dy).*int_i +...
158         vx.*dint_dx_i + vy.*dint_dy_i;
159
160     for j = 1:1:N
161
162         int_j = Master_int2D(X,Y,xdim,ydim,j,[0,0],type,param,'cheb');
163         dint_d1_j = Master_int2D(X,Y,xdim,ydim,j,[1,0],type,param,'cheb');
164         dint_d2_j = Master_int2D(X,Y,xdim,ydim,j,[0,1],type,param,'cheb');
165
166         dint_dx_j = (J22.*dint_d1_j - J12.*dint_d2_j)./Jdet;
167         dint_dy_j = (J11.*dint_d2_j - J21.*dint_d1_j)./Jdet;
168
169         R1_rho_j = (1./dt + dvx_dx + dvy_dy).*int_j +...
170             vx.*dint_dx_j + vy.*dint_dy_j;
171
172         R1_vx_j = drho_dx.*int_j + rho.*dint_dx_j;
```

```
173
174         R1_vy_j = drho_dy.*int_j + rho.*dint_dy_j;
175
176         R2_rho_j = (-1./rho.^2).*dP_dx.*int_j;
177
178         R2_vx_j = (1./dt + dvx_dx).*int_j +...
179             vx.*dint_dx_j + vy.*dint_dy_j;
180
181         R2_vy_j = dvx_dy.*int_j;
182
183         R2_P_j = (1./rho).*dint_dx_j;
184
185         R3_rho_j = (-1./rho.^2).*dP_dy.*int_j;
186
187         R3_vx_j = dvy_dx.*int_j;
188
189         R3_vy_j = (1./dt + dvy_dy).*int_j +...
190             vx.*dint_dx_j + vy.*dint_dy_j;
191
192         R3_P_j = (1./rho).*dint_dy_j;
193
194         R4_vx_j = dP_dx.*int_j + gamma.*P.*dint_dx_j;
195
196         R4_vy_j = dP_dy.*int_j + gamma.*P.*dint_dy_j;
197
198         R4_P_j = (1./dt + gamma.*dvx_dx + gamma.*dvy_dy).*int_j +...
199             vx.*dint_dx_j + vy.*dint_dy_j;
200
201         K11(i,j) = sum(sum(W.*Jdet.*(dR1_drho_i.*R1_rho_j +...
202             dR2_drho_i.*R2_rho_j + dR3_drho_i.*R3_rho_j)));
203
204         K12(i,j) = sum(sum(W.*Jdet.*(dR1_drho_i.*R1_vx_j +...
205             dR2_drho_i.*R2_vx_j + dR3_drho_i.*R3_vx_j)));
206
207         K13(i,j) = sum(sum(W.*Jdet.*(dR1_drho_i.*R1_vy_j +...
208             dR2_drho_i.*R2_vy_j + dR3_drho_i.*R3_vy_j)));
209
210         K14(i,j) = sum(sum(W.*Jdet.*(dR2_drho_i.*R2_P_j +...
211             dR3_drho_i.*R3_P_j)));
212
213         K22(i,j) = sum(sum(W.*Jdet.*(dR1_dvx_i.*R1_vx_j +...
214             dR2_dvx_i.*R2_vx_j + dR3_dvx_i.*R3_vx_j + dR4_dvx_i.*R4_vx_j)));
215
```

```
216         K23(i,j) = sum(sum(W.*Jdet.*(dR1_dvx_i.*R1_vy_j +...
217             dR2_dvx_i.*R2_vy_j + dR3_dvx_i.*R3_vy_j + dR4_dvx_i.*R4_vy_j)));
218
219         K24(i,j) = sum(sum(W.*Jdet.*(dR2_dvx_i.*R2_P_j +...
220             dR3_dvx_i.*R3_P_j + dR4_dvx_i.*R4_P_j)));
221
222         K33(i,j) = sum(sum(W.*Jdet.*(dR1_dvy_i.*R1_vy_j +...
223             dR2_dvy_i.*R2_vy_j + dR3_dvy_i.*R3_vy_j + dR4_dvy_i.*R4_vy_j)));
224
225         K34(i,j) = sum(sum(W.*Jdet.*(dR2_dvy_i.*R2_P_j +...
226             dR3_dvy_i.*R3_P_j + dR4_dvy_i.*R4_P_j)));
227
228         K44(i,j) = sum(sum(W.*Jdet.*(dR2_dP_i.*R2_P_j +...
229             dR3_dP_i.*R3_P_j + dR4_dP_i.*R4_P_j)));
230     end
231 end
232 % Assemble the components of the coefficient matrix.
233 K = zeros(4.*N,4.*N);
234
235 K(1:4:end,1:4:end) = K11;
236 K(2:4:end,1:4:end) = K12';
237 K(3:4:end,1:4:end) = K13';
238 K(4:4:end,1:4:end) = K14';
239
240 K(1:4:end,2:4:end) = K12;
241 K(2:4:end,2:4:end) = K22;
242 K(3:4:end,2:4:end) = K23';
243 K(4:4:end,2:4:end) = K24';
244
245 K(1:4:end,3:4:end) = K13;
246 K(2:4:end,3:4:end) = K23;
247 K(3:4:end,3:4:end) = K33;
248 K(4:4:end,3:4:end) = K34';
249
250 K(1:4:end,4:4:end) = K14;
251 K(2:4:end,4:4:end) = K24;
252 K(3:4:end,4:4:end) = K34;
253 K(4:4:end,4:4:end) = K44;
254
255 end
```

```
function [U_out,Vrel_vector] =...
    EulerSolver(xdim,ydim,Gnodes,type,param,B,U,dt,BC,maxIt)
% This function iteratively solves the nonlinear finite element equations.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same xdim.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.  It is assumed that every element has the
%    same ydim.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the global node number.
%    This matrix contains coordinates for all of the nodes in the domain.
% type = a vector indicating what type of interpolation is used in the
%    element.  Each entry in the vector is either a 0, 1, or 2.  The row
%    index of the vector corresponds to the element number.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a matrix of exponential parameters.  The row index of the matrix
%    corresponds to the element number.  Each row of the matrix contains
%    the exponential parameters for an element.  If an element type is 0,
%    the row in param for that element should contain all zeros.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.
% U = the values of the dependent variables at each node from either an
%    initial guess or the previous solution iteration.
% dt = time step.
% BC = a structure that holds the boundary conditions.
%       .essential = a vector the same size as U.  If an essential boundary
%           condition is specified for U(i), the value of the boundary
%           condition is stored in BC.essential(i).  All other elements of
%           BC.essential = NaN.
%       .wall = a matrix the same size as B.  BC.wall(i,j) = B(i,j) for
%           node j of element i if that node is on a solid wall.  All other
%           elements of BC.wall = 0.
%       .mesh = a column vector with twice the number of rows in Gnodes.
%           If the x-coordinate of node i does not move during mesh
```

```
44 %           adaptation, BC.mesh(2*i-1,1) = Gnodes(i,1).  If the
45 %           y-coordinate of node i does not move during mesh adaptation,
46 %           BC.mesh(2*i,1) = Gnodes(i,2).  All other elements of
47 %           BC.mesh = NaN.
48 % maxIt = the maximum number of Newton iterations.
49 %
50 % OUTPUT
51 % U_out = the values of the dependent variables after the final Newton
52 %    iteration.
53 % Vrel_vector = a vector containing the value of the norm of the relative
54 %    velocity divided by the norm of the velocity after each Newton
55 %    iteration.
56
57 Vrel_vector = ones(maxIt,1).*NaN;
58
59 for j = 1:1:maxIt
60
61     % Calculate and assemble the coefficient matrix for the entire domain.
62     K = GlobalMat(xdim,ydim,Gnodes,type,param,B,U,dt);
63
64     % Calculate and assemble the right side vector for the entire domain.
65     F = GlobalF(xdim,ydim,Gnodes,type,param,B,U,dt);
66
67     % Impose the boundary conditions.
68     [K,F] = ImposeBC(xdim,ydim,Gnodes,type,param,B,K,F,BC);
69
70     % Generate the Jacobi preconditioner.
71     M = spdiags(diag(K),0,size(K,1),size(K,2));
72
73     % Update the solution vector using the preconditioned conjugate
74     % gradient method.
75     [U1,flag,~,~,~] = pcg(K,F,1e-8,10000,M,[],U);
76
77     % If the solution does not converge using a Jacobi preconditioner, use
78     % a symmetric Gauss-Seidel preconditioner.
79     if flag ~= 0
80
81         % Generate the symmetric Gauss-Seidel preconditioner.
82         [D,Dinv,L] = GS_Precondition(K);
83         M = (D - L)*Dinv*(D - L');
84
85         % Update the solution vector using the preconditioned conjugate
86         % gradient method.
```

```
87          [U1,flag,~,~,~] = pcg(K,F,1e-8,10000,M,[],U1);
88
89      end
90
91      % Check convergence of the Newton iteration.
92      V = sqrt(U(2:4:end).^2 + U(3:4:end).^2);
93      V1 = sqrt(U1(2:4:end).^2 + U1(3:4:end).^2);
94      Vrel = norm(V1 - V)./norm(V);
95      Vrel_vector(j,1) = Vrel;
96
97      % Plot the convergence history.
98      figure(1)
99      semilogy(Vrel_vector(~isnan(Vrel_vector)),'ok')
100
101     % Stop execution of the Newton iterations if the convergence criterion
102     % is met.
103     if (Vrel <= 1e-5) && (flag == 0)
104         break
105     else
106         U = U1;
107     end
108
109     % Save the solver's progress to a .mat file.  If this function is
110     % interrupted, it can be started again at the last complete Newton
111     % iteration.
112     save('Euler_Solver_temp.mat','U1','Vrel_vector')
113
114 end
115
116 % Create the output variable and save the solver's progress to a .mat file.
117 U_out = U1;
118 save('Euler_Solver_temp.mat','U1','Vrel_vector')
119
120 end
```

```
function Uout =...
    Eval_Dofs(xdim,ydim,el_type,param,B,U_node,N1,N2,el_num,Nnode_Dofs)
% Evaluate a dependent variable at a set of local coordinates and element
% numbers.  N1, N2, and el_num may be matrices or vectors.  If N1, N2, and
% el_num are vectors they must be column vectors.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same xdim.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.  It is assumed that every element has the
%    same ydim.
% el_type = a vector indicating what type of interpolation is used in the
%    element.  Each entry in the vector is either a 0, 1, or 2.  The row
%    index of the vector corresponds to the element number.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a matrix of exponential parameters.  The row index of the matrix
%    corresponds to the element number.  Each row of the matrix contains
%    the exponential parameters for an element.  If an element type is 0,
%    the row in param for that element should contain all zeros.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.
% U_node = the values of the dependent variables at each node.
% N1 = a vector of horizontal coordinates in the local coordinate system
%    where the dependent variables are to be evaluated.
% N2 = a vector of vertical coordinates in the local coordinate system
%    where the dependent variables are to be evaluated.
% el_num = a vector of element numbers where the dependent variables are to
%    be evaluated.  N1, N2, and el_num should be the same size.
% Nnode_Dofs = a scalar defining the number of dependent variables at each
%    node.
%
% OUTPUT:
% Uout = a vector of dependent variables evaluated at each (N1,N2,el_num).

% Initialize the output variable.
Uout = zeros(Nnode_Dofs.*size(N1,1),1);
```

```
44
45 % Iterate through each element in the domain.
46 for Brow = 1:1:size(B,1)
47
48     % Find all indices of el_num that belong to the current element.
49     [row,~,~] = find(el_num == Brow);
50
51     % If there are no occurrences of the current element index in el_num
52     % move on to the next loop iteration.
53     if isempty(row)
54         continue
55     end
56
57     % Iterate through each occurrence of the current element number in
58     % el_num.
59     for Q = 1:1:max(size(row))
60
61         % Iterate through each node number in the current element.
62         for k = 1:1:(xdim.*ydim)
63
64             % Evaluate the interpolation function at (N1,N2).
65             int = Master_int2D(N1(row(Q)),N2(row(Q)),xdim,ydim,k,[0,0],...
66                 el_type(Brow,1),param(Brow,:),'cheb');
67
68             % Iterate through each degree of freedom at the node.
69             for Dof = Nnode_Dofs:-1:1
70
71                 % Evaluate the degree of freedom at (N1,N2).
72                 U_node_k = U_node((Nnode_Dofs.*B(Brow,k)-(Dof-1)),1);
73                 Uout(Nnode_Dofs.*row(Q)-(Dof-1),1) =...
74                     Uout(Nnode_Dofs.*row(Q)-(Dof-1),1) + int.*U_node_k;
75
76             end
77         end
78     end
79 end
80
81 end
```

```
function N = Exp_1D_GLquad_Num(param)
% This function calculates the number of Gauss–Legendre quadrature points
% necessary to integrate a one–dimensional exponential interpolation
% function.  It is assumed that only one of the exponential parameters is
% nonzero.
%
% INPUT:
% param = a vector of exponential parameters for a single interpolation
%    function.
%
% OUTPUT:
% N = the number of Gauss–Legendre quadrature points necessary to integrate
%    a one–dimensional exponential interpolation function.

% Find the index of the parameter with the highest absolute value.
[~,pnum,~] = find(abs(param) > 0.01,1,'last');

pval = param(1,pnum);

if isempty(pnum)
    pnum = 0;
    pval = 0;
end

% Calculate N based on an empirical formula.  Round up to make N a whole
% number.
N = ceil(4.339423015247898.*sqrt(abs(pval).*pnum));

end
```

```
function [f] = Exp_int1D(x,xi,param,fnum,deriv)
% This function returns the value of an exponential interpolation function
% for a 1 dimensional C^0 element.  The function index is equal to
% the node index.  Nodes must be numbered sequentially starting with the
% node located at the lowest xi.
%
% INPUT:
% x = a scalar, vector, or matrix of the independent variable values where
%    the function is to be evaluated.
% xi = a 1xN vector of the node locations where N is the number of nodes.
% param = a 1x(N-1) vector of the exponential parameters that affect the
%    shape of the function.
% fnum =  the interpolation function index.
% deriv = a number indicating whether to evaluate the interpolation
%    function indicated by fnum or its derivative.
%       0 returns the value of the interpolation function.
%       1 returns the value of the interpolation function's first
%         derivative with respect to x.
%       2 returns the value of the interpolation function's second
%         derivative with respect to x.
%
% OUTPUT:
% f = the value of the function at each x.

N = size(xi,2); % The number of nodes.

% Construct a matrix to use with Cramer's rule to find the coefficients of
% the interpolation function.
A = zeros(N,N);
A(:,1) = ones(N,1);

for j = 2:1:N

    if abs(param(1,(j-1))) <= 0.001 % Limit if the parameter is near zero.
        A(:,j) = xi.^(j-1).*exp(param(1,(j-1)).*(xi.^(j-1)));
    else
        A(:,j) = exp(param(1,(j-1)).*(xi.^(j-1)));
    end

end

% Calculate the coefficients of the interpolation function using Cramer's
% rule.
```

```
D = det(A);
Di = zeros(1,N);

for k = 1:1:N

    Aminor = A([1:(fnum-1),(fnum+1):end],[1:(k-1),(k+1):end]);
    Di(1,k) = (-1).^(fnum + k).*det(Aminor);

end

Di = Di./D;

% Initialize a variable for the output.
if (deriv == 1) || (deriv == 2)
    f = zeros(size(x));
else
    f = ones(size(x)).*Di(1,1);
end

for m = 2:1:N

    if deriv == 2 % Second derivative

        if m == 2
            if abs(param(1,(m-1))) <= 0.001 % Limit if the parameter is
                                            % near zero.

                f = f + Di(1,m).*(m-1).*exp(param(1,(m-1)).*x.^(m-1)).*...
                    ((m-1).*2.*param(1,(m-1)).*x.^(2.*m-4) + ...
                    (m-1).*param(1,(m-1)).^2.*x.^(2.*m-4).*x.^(m-1));

            else

                f = f + Di(1,m).*exp(param(1,(m-1)).*x.^(m-1)).*(m-1).*...
                    param(1,(m-1)).*((m-1).*param(1,(m-1)).*x.^(2.*m-4));

            end
        else
            if abs(param(1,(m-1))) <= 0.001 % Limit if the parameter is
                                            % near zero.

                f = f + Di(1,m).*(m-1).*exp(param(1,(m-1)).*x.^(m-1)).*(...
                    (m-2).*x.^(m-3) + (m-1).*2.*param(1,(m-1)).*...
```

```
87                      x.^(2.*m-4) + (m-1).*param(1,(m-1)).^2.*x.^(2.*m-4).*...
88                      x.^(m-1) + (m-2).*param(1,(m-1)).*x.^(m-1).*x.^(m-3));
89
90              else
91
92                  f = f + Di(1,m).*exp(param(1,(m-1)).*x.^(m-1)).*(m-1).*...
93                      param(1,(m-1)).*((m-1).*param(1,(m-1)).*x.^(2.*m-4) +...
94                      (m-2).*x.^(m-3));
95
96              end
97          end
98
99      elseif deriv == 1 % First derivative
100
101         if abs(param(1,(m-1))) <= 0.001 % Limit if the parameter is near
102                                         % zero.
103
104             f = f + Di(1,m).*(m-1).*x.^(m-2).*exp(param(1,(m-1)).*...
105                 x.^(m-1)).*(1 + x.^(m-1).*param(1,(m-1)));
106
107         else
108
109             f = f + Di(1,m).*param(1,(m-1)).*(m-1).*x.^(m-2).*...
110                 exp(param(1,(m-1)).*x.^(m-1));
111
112         end
113
114     else % Interpolation function (no derivative)
115
116         if abs(param(1,(m-1))) <= 0.001 % Limit if the parameter is near
117                                         % zero.
118
119             f = f + Di(1,m).*x.^(m-1).*exp(param(1,(m-1)).*x.^(m-1));
120
121         else
122
123             f = f + Di(1,m).*exp(param(1,(m-1)).*x.^(m-1));
124
125         end
126     end
127 end
128 end
```

```
function FEASurf(xdim,ydim,Gnodes,type,param,B,U,inc)
% This function plots a solution variable and its derivatives on the entire
% domain.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same xdim.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.  It is assumed that every element has the
%    same ydim.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the global node number.
%    This matrix contains coordinates for all of the nodes in the domain.
% type = a vector indicating what type of interpolation is used in the
%    element.  Each entry in the vector is either a 0, 1, or 2.  The row
%    index of the vector corresponds to the element number.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a matrix of exponential parameters.  The row index of the matrix
%    corresponds to the element number.  Each row of the matrix contains
%    the exponential parameters for an element.  If an element type is 0,
%    the row in param for that element should contain all zeros.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.
% U = the value of ONE of the dependent variables at each node.
% inc = the increment between local coordinates where the value of the
%    dependent variable and its derivatives will be calculated.

N_el = size(B,1); % Number of elements in the domain.

% Create figure handles for the output figures and the axes in each figure.
f1 = figure('Visible','off');
a1 = gca;

f2 = figure('Visible','off');
a2 = gca;

```

```
44 f3 = figure('Visible','off');
45 a3 = gca;
46
47 f4 = figure('Visible','off');
48 a4 = gca;
49
50 f5 = figure('Visible','off');
51 a5 = gca;
52
53 f6 = figure('Visible','off');
54 a6 = gca;
55
56 f7 = figure('Visible','off');
57 a7 = gca;
58
59 % Execute this loop for each element.
60 for j = 1:1:N_el
61
62     % Get the global degrees of freedom for the current element.
63     U_el = U(B(j,:),1);
64
65     % Get the global coordinates of the nodes for the current element.
66     enodes = Gnodes(B(j,:),:);
67
68     % Calculate everything needed to plot the dependent variable and its
69     % derivatives on the current element.
70     [Xplot,Yplot,Uplot,dUplot_dx,dUplot_dy,d2Uplot_dx2,d2Uplot_dxdy,...
71         d2Uplot_dy2,Jdet] = Element2Grid(xdim,ydim,enodes,type(j,1),...
72         param(j,:),U_el,inc);
73
74     % Plot the dependent variable on the current element.
75     set(0,'CurrentFigure',f1)
76     surf(Xplot,Yplot,Uplot,'edgecolor','none','facecolor','interp')
77     hold on
78
79     % Plot the first derivatives of the dependent variable on the current
80     % element.
81     set(0,'CurrentFigure',f2)
82     surf(Xplot,Yplot,dUplot_dx,'edgecolor','none','facecolor','interp')
83     hold on
84
85     set(0,'CurrentFigure',f3)
86     surf(Xplot,Yplot,dUplot_dy,'edgecolor','none','facecolor','interp')
```

```
87      hold on
88
89      % Plot the determinant of the Jacobian on the current element.
90      set(0,'CurrentFigure',f4)
91      surf(Xplot,Yplot,Jdet,'edgecolor','none','facecolor','interp')
92      hold on
93
94      % Plot the second derivatives of the dependent variable on the current
95      % element.
96      set(0,'CurrentFigure',f5)
97      surf(Xplot,Yplot,d2Uplot_dx2,'edgecolor','none','facecolor','interp')
98      hold on
99
100     set(0,'CurrentFigure',f6)
101     surf(Xplot,Yplot,d2Uplot_dxdy,'edgecolor','none','facecolor','interp')
102     hold on
103
104     set(0,'CurrentFigure',f7)
105     surf(Xplot,Yplot,d2Uplot_dy2,'edgecolor','none','facecolor','interp')
106     hold on
107
108 end
109
110 % Adjust the x and y axes of each figure so that the increments along each
111 % axis are the same size.  This is similar to using the "axis equal"
112 % command except it excludes the z axis.
113 d1 = daspect(a1);
114 d1(1,2) = d1(1,1);
115 daspect(a1,d1)
116
117 d2 = daspect(a2);
118 d2(1,2) = d2(1,1);
119 daspect(a2,d2)
120
121 d3 = daspect(a3);
122 d3(1,2) = d3(1,1);
123 daspect(a3,d3)
124
125 d4 = daspect(a4);
126 d4(1,2) = d4(1,1);
127 daspect(a4,d4)
128
129 d5 = daspect(a5);
```

```
d5(1,2) = d5(1,1);
daspect(a5,d5)

d6 = daspect(a6);
d6(1,2) = d6(1,1);
daspect(a6,d6)

d7 = daspect(a7);
d7(1,2) = d7(1,1);
daspect(a7,d7)

% Make all of the figures visible and place a title on each figure.
set(f1,'Visible','on')
colormap jet(256)
title('$U$','interpreter','latex')

set(f2,'Visible','on')
colormap jet(256)
title('$\frac{\partial U}{\partial x}$','interpreter','latex')

set(f3,'Visible','on')
colormap jet(256)
title('$\frac{\partial U}{\partial y}$','interpreter','latex')

set(f4,'Visible','on')
colormap jet(256)
title('$\|J\|$','interpreter','latex')

set(f5,'Visible','on')
colormap jet(256)
title('$\frac{\partial^2 U}{\partial x^2}$','interpreter','latex')

set(f6,'Visible','on')
colormap jet(256)
title('$\frac{\partial^2 U}{\partial x \partial y}$','interpreter','latex')

set(f7,'Visible','on')
colormap jet(256)
title('$\frac{\partial^2 U}{\partial y^2}$','interpreter','latex')

end
```

```
function [N1_out,N2_out,elnum_out] = Global2Local(xdim,ydim,Gnodes,B,x,y)
% This function finds local coordinates and element numbers that correspond
% to a set of global coordinates.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same xdim.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.  It is assumed that every element has the
%    same ydim.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the global node number.
%    This matrix contains coordinates for all of the nodes in the domain.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.
% x = a set of x-coordinates in the global coordinate system.
% y = a set of y-coordinates in the global coordinate system.
%
% OUTPUT
% N1_out = horizontal coordinates in the local coordinate system.
% N2_out = vertical coordinates in the local coordinate system.
% elnum_out = the element number for each (N1,N2) coordinate.

% If x and y are not column vectors, reshape them into column vectors.
reshape_out = 0;

if size(x,2) ~= 1
    reshape_out = 1;
    size_out = size(x);

    x = reshape(x,numel(x),1);
    y = reshape(y,numel(y),1);

end

% Initialize output variables.
N1_out = NaN.*ones(size(x));
N2_out = NaN.*ones(size(x));
elnum_out = NaN.*ones(size(x));

% Arrange the local node indices so that the edge nodes will form a convex
```

```
44 % polygon.
45 edge_ind = [1:1:xdim,(2.*xdim):xdim:(xdim.*ydim),...
46     (xdim.*ydim-1):-1:(xdim.*ydim-xdim+1),...
47     (xdim.*ydim-2.*xdim+1):-xdim:(xdim+1)];
48
49 fmin_opts = optimset('Algorithm','active-set','Display','off');
50
51 % Execute this loop for each element in the domain.
52 for el_num = 1:1:size(B,1)
53
54     % Find out which (x,y) points are inside or on the boundary of the
55     % current element.
56     IN = inpolygon(x,y,Gnodes(B(el_num,edge_ind),1),...
57         Gnodes(B(el_num,edge_ind),2));
58
59     [in_row,~,~] = find(IN);
60
61     % If none of the (x,y) coordinates are inside the current element, skip
62     % the rest of the loop and move to the next element.
63     if isempty(in_row)
64         continue
65     end
66
67     elnum_out(in_row,1) = el_num;
68
69     % Define a function handle for fmincon to minimize.
70     min_fun = @(N)find_local(xdim,ydim,Gnodes(B(el_num,:),:),x(in_row,1),y(in_row,1),N);
71
72     % Define an initial guess, upper bounds, and lower bounds to input into
73     % fmincon.
74     ini_guess = zeros(2.*max(size(in_row)),1);
75     min_bound = -1.*ones(2.*max(size(in_row)),1);
76     max_bound = ones(2.*max(size(in_row)),1);
77
78     % Find the local coordinates for each global coordinate by minimizing
79     % the distance between a guess for (N1,N2) and the (x,y) coordinates.
80     [N_out,~,~] = fmincon(min_fun,ini_guess,[],[],[],[],min_bound,...
81         max_bound,[],fmin_opts);
82
83     % Put the local coordinates into the output variable.
84     N1_out(in_row,1) = N_out(1:2:end,1);
85     N2_out(in_row,1) = N_out(2:2:end,1);
86
```

```
87 end
88
89 % Reshape the output variables to match the dimensions of x and y in the
90 % inputs.
91 if reshape_out == 1
92
93     N1_out = reshape(N1_out,size_out);
94     N2_out = reshape(N2_out,size_out);
95     elnum_out = reshape(elnum_out,size_out);
96
97 end
98
99 end
100
101 function dist = find_local(xdim,ydim,elnodes,x,y,N)
102 % This function calculates the distance between a set of local coordinates
103 % and a set of global coordinates.  This is the function that fmincon
104 % minimizes in Global2Local.
105
106 [Xguess,Yguess] = Local2Global(xdim,ydim,elnodes,N(1:2:end,1),...
107     N(2:2:end,1),'cheb');
108
109 dist = max(size(N)).*sum(sqrt((x - Xguess).^2 + (y - Yguess).^2));
110 end
```

```
function [F] = GlobalF(xdim,ydim,Gnodes,type,param,B,U,dt)
% This function generates the right side vector in [K]{U}={F} for the
% entire domain.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same xdim.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.  It is assumed that every element has the
%    same ydim.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the global node number.
%    This matrix contains coordinates for all of the nodes in the domain.
% type = a vector indicating what type of interpolation is used in the
%    element.  Each entry in the vector is either a 0, 1, or 2.  The row
%    index of the vector corresponds to the element number.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a matrix of exponential parameters.  The row index of the matrix
%    corresponds to the element number.  Each row of the matrix contains
%    the exponential parameters for an element.  If an element type is 0,
%    the row in param for that element should contain all zeros.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.
% U = the values of the dependent variables at each node from either an
%    initial guess or the previous solution iteration.
% dt = time step.
%
% OUTPUT:
% F = right side vector for the entire domain.

N_el = size(B,1); % Number of elements in the domain.

% Initialize the output variable.
F = sparse([],[],[],size(U,1),1,4.*size(Gnodes,1));

% Execute this loop for each element in the domain.
```

```
44 for j = 1:1:N_el
45
46     % Collect the global degree of freedom indices for the current element
47     % and convert them to local degree of freedom indices.
48     u_ind = zeros(4.*size(B,2),1);
49
50     u_ind(1:4:end) = 4.*B(j,:) - 3;
51     u_ind(2:4:end) = 4.*B(j,:) - 2;
52     u_ind(3:4:end) = 4.*B(j,:) - 1;
53     u_ind(4:4:end) = 4.*B(j,:);
54
55     % Get the global degrees of freedom for the current element.
56     U_el = U(u_ind,1);
57
58     % Generate the right side vector for the element.
59     enodes = Gnodes([B(j,:)]',:);
60     [F_el] = Euler_F(xdim,ydim,enodes,type(j,1),param(j,:),U_el,dt);
61
62     % Convert the right side vector to vectors of indices and values.
63     [Fi_el,~,F_el_val] = find(F_el);
64
65     % Convert the element row indices to global row indices.
66     Fi_el = changem_fea(Fi_el,u_ind',1:1:4.*size(B,2));
67
68     % Make a sparse vector for the element that has the same size as the
69     % global right side vector.
70     % Add the element right side vector to the global right side vector.
71     F = F + sparse(Fi_el,1,F_el_val,size(U,1),1);
72
73 end
74 end
```

```
function [K] = GlobalMat(xdim,ydim,Gnodes,type,param,B,U,dt)
% This function generates the coefficient matrix in [K]{U}={F} for the
% entire domain.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same xdim.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.  It is assumed that every element has the
%    same ydim.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the global node number.
%    This matrix contains coordinates for all of the nodes in the domain.
% type = a vector indicating what type of interpolation is used in the
%    element.  Each entry in the vector is either a 0, 1, or 2.  The row
%    index of the vector corresponds to the element number.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a matrix of exponential parameters.  The row index of the matrix
%    corresponds to the element number.  Each row of the matrix contains
%    the exponential parameters for an element.  If an element type is 0,
%    the row in param for that element should contain all zeros.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.
% U = the values of the dependent variables at each node from either an
%    initial guess or the previous solution iteration.
% dt = time step.
%
% OUTPUT:
% K = coefficient matrix for the entire domain.

N_el = size(B,1); % Number of elements in the domain.

% Initialize the output variable.
K = sparse([],[],[],size(U,1),size(U,1),size(B,1).*(4.*xdim.*ydim).^2);

% Execute this loop for each element in the domain.
```

```
44 for j = 1:1:N_el
45
46     % Collect the global degree of freedom indices for the current element
47     % and convert them to local degree of freedom indices.
48     u_ind = zeros(4.*size(B,2),1);
49
50     u_ind(1:4:end) = 4.*B(j,:) - 3;
51     u_ind(2:4:end) = 4.*B(j,:) - 2;
52     u_ind(3:4:end) = 4.*B(j,:) - 1;
53     u_ind(4:4:end) = 4.*B(j,:);
54
55     % Get the global degrees of freedom for the current element.
56     U_el = U(u_ind,1);
57
58     % Generate the coefficient matrix for the element.
59     [K_el] = Euler_K(xdim,ydim,Gnodes([B(j,:)]',:),type(j,1),param(j,:),U_el,dt);
60
61     % Convert the element row and column indices to global row and column
62     % indices.
63     [Ki_el,Kj_el,K_el_val] = find(K_el);
64     Ki_el = changem_fea(Ki_el,u_ind',1:1:4.*size(B,2));
65     Kj_el = changem_fea(Kj_el,u_ind',1:1:4.*size(B,2));
66
67     % Make a sparse matrix for the element that has the same size as the
68     % global coefficient matrix and add the element coefficient matrix to
69     % the global coefficient matrix.
70     K = K + sparse(Ki_el,Kj_el,K_el_val,size(U,1),size(U,1));
71
72 end
73 end
```

```
function [D,Dinv,L] = GS_Precondition(K)
% This function generates the matrices necessary to construct a symmetric
% Gauss-Seidel preconditioner for the input matrix K.  It is assumed that
% K is symmetric.  K = D - L - L'
%
% INPUT:
% K = a square matrix from which the preconditioner will be constructed.
%
% OUTPUT:
% D = a matrix the same size as K that contains the entries on the main
%    diagonal of K.
% Dinv = the inverse of D.
% L = the lower triangular portion of K*(-1)

m = size(K,1);

% Initialize the output variables.
D = sparse([],[],[],m,m,m);
Dinv = sparse([],[],[],m,m,m);
L = sparse([],[],[],m,m,nnz(K)./2);

% Place the elements of K into the output variables.
D = D + sparse(1,1,K(1,1),m,m);
Dinv = Dinv + sparse(1,1,1./K(1,1),m,m);

for s = 2:1:m

    D = D + sparse(s,s,K(s,s),m,m);
    Dinv = Dinv + sparse(s,s,1./K(s,s),m,m);
    L = L - sparse(s,1:1:(s-1),K(s,1:1:(s-1)),m,m);

end

end
```

```
function [K,F] = ImposeBC(xdim,ydim,Gnodes,type,param,B,K,F,BC)
% This function imposes the essential and solid wall boundary conditions on
% the coefficient matrix [K] and the right side vector {F} in the equation
% [K]{U}={F}.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same xdim.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.  It is assumed that every element has the
%    same ydim.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the global node number.
%    This matrix contains coordinates for all of the nodes in the domain.
% type = a vector indicating what type of interpolation is used in the
%    element.  Each entry in the vector is either a 0, 1, or 2.  The row
%    index of the vector corresponds to the element number.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a matrix of exponential parameters.  The row index of the matrix
%    corresponds to the element number.  Each row of the matrix contains
%    the exponential parameters for an element.  If an element type is 0,
%    the row in param for that element should contain all zeros.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.
% K = The assembled global coefficient matrix.
% F = The assembled global right side vector.
% BC = a structure that holds the boundary conditions.
%       .essential = a vector the same size as U.  If an essential boundary
%           condition is specified for U(i), the value of the boundary
%           condition is stored in BC.essential(i).  All other elements of
%           BC.essential = NaN.
%       .wall = a matrix the same size as B.  BC.wall(i,j) = B(i,j) for
%           node j of element i if that node is on a solid wall.  All other
%           elements of BC.wall = 0.
%       .mesh = a column vector with twice the number of rows in Gnodes.
%           If the x-coordinate of node i does not move during mesh
```

```
44 %           adaptation, BC.mesh(2*i-1,1) = Gnodes(i,1).  If the
45 %           y-coordinate of node i does not move during mesh adaptation,
46 %           BC.mesh(2*i,1) = Gnodes(i,2).  All other elements of
47 %           BC.mesh = NaN.  BC.mesh is not used in this function.
48 %
49 % OUTPUT:
50 % K = The global coefficient matrix after boundary conditions have been
51 %    imposed.
52 % F = The global right side vector after boundary conditions have been
53 %    imposed.
54
55 % Impose solid wall boundary conditions.
56 % Execute this loop for each element in the domain.
57 for j = 1:1:size(BC.wall,1)
58
59     % Get the global node coordinates for the current element.
60     enodes = Gnodes(B(j,:),:);
61
62     % Determine if the bottom edge of the element is a solid wall.
63     if BC.wall(j,1:1:xdim) ~= 0
64
65         % Calculate the solid wall boundary condition coefficient matrices
66         % for the current element.
67         [xxQ,xyQ,yyQ] = WallBC('bottom',xdim,ydim,enodes,type(j,1),...
68             param(j,:));
69
70         % Break the matrices into vectors of row indices, column indices,
71         % and values.
72         [xxQ_i,xxQ_j,xxQ_val] = find(xxQ);
73         [xyQ_i,xyQ_j,xyQ_val] = find(xyQ);
74         [yyQ_i,yyQ_j,yyQ_val] = find(yyQ);
75
76         % Change the local row and column indices to global row and column
77         % indices.
78         xxQ_i = changem_fea(xxQ_i,4.*BC.wall(j,1:1:xdim) - 2,1:1:xdim);
79         xxQ_j = changem_fea(xxQ_j,4.*BC.wall(j,1:1:xdim) - 2,1:1:xdim);
80
81         xyQ_i = changem_fea(xyQ_i,4.*BC.wall(j,1:1:xdim) - 2,1:1:xdim);
82         xyQ_j = changem_fea(xyQ_j,4.*BC.wall(j,1:1:xdim) - 1,1:1:xdim);
83
84         yyQ_i = changem_fea(yyQ_i,4.*BC.wall(j,1:1:xdim) - 1,1:1:xdim);
85         yyQ_j = changem_fea(yyQ_j,4.*BC.wall(j,1:1:xdim) - 1,1:1:xdim);
86
```

```
87          % Add the solid wall boundary condition coefficient matrices to the
88          % global coefficient matrix.
89          K = K + sparse(xxQ_i,xxQ_j,5.*xxQ_val,size(K,1),size(K,2));
90          K = K + sparse(xyQ_i,xyQ_j,5.*xyQ_val,size(K,1),size(K,2));
91          K = K + sparse(xyQ_j,xyQ_i,5.*xyQ_val,size(K,1),size(K,2));
92          K = K + sparse(yyQ_i,yyQ_j,5.*yyQ_val,size(K,1),size(K,2));
93
94      end
95
96      % Determine if the top edge of the element is a solid wall.
97      if BC.wall(j,(xdim.*ydim - xdim + 1):1:(xdim.*ydim)) ~= 0
98
99          % Calculate the solid wall boundary condition coefficient matrices
100         % for the current element.
101         [xxQ,xyQ,yyQ] = WallBC('top',xdim,ydim,enodes,type(j,1),param(j,:));
102
103         % Break the matrices into vectors of row indices, column indices,
104         % and values.
105         [xxQ_i,xxQ_j,xxQ_val] = find(xxQ);
106         [xyQ_i,xyQ_j,xyQ_val] = find(xyQ);
107         [yyQ_i,yyQ_j,yyQ_val] = find(yyQ);
108
109         % Change the local row and column indices to global row and column
110         % indices.
111         xxQ_i = changem_fea(xxQ_i,...
112             4.*BC.wall(j,(xdim.*ydim - xdim + 1):1:(xdim.*ydim)) - 2,...
113             (xdim.*ydim - xdim + 1):1:(xdim.*ydim));
114         xxQ_j = changem_fea(xxQ_j,...
115             4.*BC.wall(j,(xdim.*ydim - xdim + 1):1:(xdim.*ydim)) - 2,...
116             (xdim.*ydim - xdim + 1):1:(xdim.*ydim));
117
118         xyQ_i = changem_fea(xyQ_i,...
119             4.*BC.wall(j,(xdim.*ydim - xdim + 1):1:(xdim.*ydim)) - 2,...
120             (xdim.*ydim - xdim + 1):1:(xdim.*ydim));
121         xyQ_j = changem_fea(xyQ_j,...
122             4.*BC.wall(j,(xdim.*ydim - xdim + 1):1:(xdim.*ydim)) - 1,...
123             (xdim.*ydim - xdim + 1):1:(xdim.*ydim));
124
125         yyQ_i = changem_fea(yyQ_i,...
126             4.*BC.wall(j,(xdim.*ydim - xdim + 1):1:(xdim.*ydim)) - 1,...
127             (xdim.*ydim - xdim + 1):1:(xdim.*ydim));
128         yyQ_j = changem_fea(yyQ_j,...
129             4.*BC.wall(j,(xdim.*ydim - xdim + 1):1:(xdim.*ydim)) - 1,...
```

```
130             (xdim.*ydim - xdim + 1):1:(xdim.*ydim));
131
132         % Add the solid wall boundary condition coefficient matrices to the
133         % global coefficient matrix.
134         K = K + sparse(xxQ_i,xxQ_j,xxQ_val,size(K,1),size(K,2));
135         K = K + sparse(xyQ_i,xyQ_j,xyQ_val,size(K,1),size(K,2));
136         K = K + sparse(xyQ_j,xyQ_i,xyQ_val,size(K,1),size(K,2));
137         K = K + sparse(yyQ_i,yyQ_j,yyQ_val,size(K,1),size(K,2));
138
139     end
140
141     % Determine if the left edge of the element is a solid wall.
142     if BC.wall(j,1:xdim:(xdim.*ydim - xdim + 1)) ~= 0
143
144         % Calculate the solid wall boundary condition coefficient matrices
145         % for the current element.
146         [xxQ,xyQ,yyQ] = WallBC('left',xdim,ydim,enodes,type(j,1),param(j,:));
147
148         % Break the matrices into vectors of row indices, column indices,
149         % and values.
150         [xxQ_i,xxQ_j,xxQ_val] = find(xxQ);
151         [xyQ_i,xyQ_j,xyQ_val] = find(xyQ);
152         [yyQ_i,yyQ_j,yyQ_val] = find(yyQ);
153
154         % Change the local row and column indices to global row and column
155         % indices.
156         xxQ_i = changem_fea(xxQ_i,...
157             4.*BC.wall(j,1:xdim:(xdim.*ydim - xdim + 1)) - 2,...
158             1:xdim:(xdim.*ydim - xdim + 1));
159         xxQ_j = changem_fea(xxQ_j,...
160             4.*BC.wall(j,1:xdim:(xdim.*ydim - xdim + 1)) - 2,...
161             1:xdim:(xdim.*ydim - xdim + 1));
162
163         xyQ_i = changem_fea(xyQ_i,...
164             4.*BC.wall(j,1:xdim:(xdim.*ydim - xdim + 1)) - 2,...
165             1:xdim:(xdim.*ydim - xdim + 1));
166         xyQ_j = changem_fea(xyQ_j,...
167             4.*BC.wall(j,1:xdim:(xdim.*ydim - xdim + 1)) - 1,...
168             1:xdim:(xdim.*ydim - xdim + 1));
169
170         yyQ_i = changem_fea(yyQ_i,...
171             4.*BC.wall(j,1:xdim:(xdim.*ydim - xdim + 1)) - 1,...
172             1:xdim:(xdim.*ydim - xdim + 1));
```

```
173         yyQ_j = changem_fea(yyQ_j,...
174             4.*BC.wall(j,1:xdim:(xdim.*ydim - xdim + 1)) - 1,...
175             1:xdim:(xdim.*ydim - xdim + 1));
176 
177         % Add the solid wall boundary condition coefficient matrices to the
178         % global coefficient matrix.
179         K = K + sparse(xxQ_i,xxQ_j,xxQ_val,size(K,1),size(K,2));
180         K = K + sparse(xyQ_i,xyQ_j,xyQ_val,size(K,1),size(K,2));
181         K = K + sparse(xyQ_j,xyQ_i,xyQ_val,size(K,1),size(K,2));
182         K = K + sparse(yyQ_i,yyQ_j,yyQ_val,size(K,1),size(K,2));
183 
184     end
185 
186     % Determine if the right edge of the element is a solid wall.
187     if BC.wall(j,xdim:xdim:(xdim.*ydim)) ~= 0
188 
189         % Calculate the solid wall boundary condition coefficient matrices
190         % for the current element.
191         [xxQ,xyQ,yyQ] = WallBC('right',xdim,ydim,enodes,type(j,1),param(j,:));
192 
193         % Break the matrices into vectors of row indices, column indices,
194         % and values.
195         [xxQ_i,xxQ_j,xxQ_val] = find(xxQ);
196         [xyQ_i,xyQ_j,xyQ_val] = find(xyQ);
197         [yyQ_i,yyQ_j,yyQ_val] = find(yyQ);
198 
199         % Change the local row and column indices to global row and column
200         % indices.
201         xxQ_i = changem_fea(xxQ_i,...
202             4.*BC.wall(j,xdim:xdim:(xdim.*ydim)) - 2,...
203             xdim:xdim:(xdim.*ydim));
204         xxQ_j = changem_fea(xxQ_j,...
205             4.*BC.wall(j,xdim:xdim:(xdim.*ydim)) - 2,...
206             xdim:xdim:(xdim.*ydim));
207 
208         xyQ_i = changem_fea(xyQ_i,...
209             4.*BC.wall(j,xdim:xdim:(xdim.*ydim)) - 2,...
210             xdim:xdim:(xdim.*ydim));
211         xyQ_j = changem_fea(xyQ_j,...
212             4.*BC.wall(j,xdim:xdim:(xdim.*ydim)) - 1,...
213             xdim:xdim:(xdim.*ydim));
214 
215         yyQ_i = changem_fea(yyQ_i,...
```

```
216             4.*BC.wall(j,xdim:xdim:(xdim.*ydim)) - 1,...
217             xdim:xdim:(xdim.*ydim));
218         yyQ_j = changem_fea(yyQ_j,...
219             4.*BC.wall(j,xdim:xdim:(xdim.*ydim)) - 1,...
220             xdim:xdim:(xdim.*ydim));
221 
222         % Add the solid wall boundary condition coefficient matrices to the
223         % global coefficient matrix.
224         K = K + sparse(xxQ_i,xxQ_j,xxQ_val,size(K,1),size(K,2));
225         K = K + sparse(xyQ_i,xyQ_j,xyQ_val,size(K,1),size(K,2));
226         K = K + sparse(xyQ_j,xyQ_i,xyQ_val,size(K,1),size(K,2));
227         K = K + sparse(yyQ_i,yyQ_j,yyQ_val,size(K,1),size(K,2));
228 
229     end
230 
231 end
232 
233 % Impose the essential boundary conditions.
234 % Execute this loop for each row in F.
235 for j = 1:1:size(F,1)
236 
237     % If there is no essential boundary condition for the current dependent
238     % variable, skip the rest of the loop and move on to the next dependent
239     % variable.
240     if isnan(BC.essential(j,1))
241         continue
242     else
243 
244         q = [1:1:(j-1),(j+1):1:size(F,1)];
245 
246         F(q,1) = F(q,1) - K(q,j).*BC.essential(j,1);
247 
248         K = K - sparse(q,j.*ones(size(q)),K(q,j),size(K,1),size(K,2));
249         K = K - sparse(j.*ones(size(q)),q,K(j,q),size(K,1),size(K,2));
250 
251         K(j,j) = 1;
252         F(j,1) = BC.essential(j,1);
253 
254     end
255 end
256 
257 end
```

```
function [f] = Lagrange_int1D(x,xi,fnum,deriv)
% This function returns the value of a Lagrange interpolation function
% for a 1 dimensional C^0 element.  The function index is equal to
% the node index.  Nodes must be numbered sequentially starting with the
% node located at the lowest xi.
%
% INPUT:
% x = a scalar, vector, or matrix of the independent variable values where
%    the function is to be evaluated.
% xi = a 1xN vector of the node locations where N is the number of nodes.
% fnum =  the interpolation function index.
% deriv = a number indicating whether to evaluate the interpolation
%    function indicated by fnum or its derivative.
%       0 returns the value of the interpolation function.
%       1 returns the value of the interpolation function's first
%         derivative with respect to x.
%       2 returns the value of the interpolation function's second
%         derivative with respect to x.
%
% OUTPUT:
% f = the value of the function at each x.

N = size(xi,2); % The number of nodes.

% Construct a matrix to use with Cramer's rule to find the coefficients of
% the interpolation function.
A = zeros(N,N);
A(:,1) = ones(N,1);

for j = 2:1:N

    A(:,j) = xi.^(j-1);

end

% Calculate the coefficients of the interpolation function using Cramer's
% rule.
D = det(A);
Di = zeros(1,N);

for k = 1:1:N

    Aminor = A([1:(fnum-1),(fnum+1):end],[1:(k-1),(k+1):end]);
```

```
44
45      Di(1,k) = (-1).^(fnum + k).*det(Aminor);
46
47  end
48
49  Di = Di./D;
50
51  % Initialize a variable for the output.
52  if (deriv == 1) || (deriv == 2)
53      f = zeros(size(x));
54  else
55      f = ones(size(x)).*Di(1,1);
56  end
57
58  for m = 2:1:N
59
60      if deriv == 2 % Second derivative
61
62          if m == 2
63              continue
64          else
65              f = f + Di(1,m).*(m-1).*(m-2).*x.^(m-3);
66          end
67
68      elseif deriv == 1 % First derivative
69
70          f = f + Di(1,m).*(m-1).*x.^(m-2);
71
72      else % Interpolation function (no derivative)
73
74          f = f + Di(1,m).*x.^(m-1);
75
76      end
77
78  end
79
80  end
```

```
function [X,Y,W] = LegendrePts2D(xdim,ydim,type,param)
% This function generates the Gauss-Legendre quadrature points and weights
% for a two-dimensional interpolation function.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% type = a scalar indicating what type of interpolation is used in the
%    element.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a vector of exponential parameters.
%
% OUTPUT:
% X = a matrix of quadrature points along the horizontal axis of the local
%    coordinate system.
% Y = a matrix of quadrature points along the vertical axis of the local
%    coordinate system.
% W = a matrix of quadrature weights for each quadrature point.

% Calculate the number of quadrature points needed to integrate the
% exponential term of a one-dimensional exponential interpolation function.
% The parameters are multiplied by 4 because the components of [K] and {F}
% contain products of 4 interpolation functions.
Exp_N = Exp_1D_GLquad_Num(4.*param);

% Calculate the number of quadrature points needed to integrate a
% one-dimensional polynomial interpolation function.
Poly_degree = max(xdim,ydim) - 1;
Poly_N = ceil((6.*Poly_degree + 1)./2);

% Determine the number of quadrature points needed along each local
% coordinate direction based on the element type.
switch type

    case 0

```

```
44          N1 = Poly_N;
45          N2 = Poly_N;
46
47      case 1
48
49          N1 = Poly_N + Exp_N;
50          N2 = Poly_N;
51
52      case 2
53
54          N1 = Poly_N;
55          N2 = Poly_N + Exp_N;
56
57  end
58
59  % Calculate the quadrature points and weights along the horizontal local
60  % coordinate direction.
61  [x,wx] = legpts(N1);
62  x = x';
63
64  % Calculate the quadrature points and weights along the vertical local
65  % coordinate direction.
66  [y,wy] = legpts(N2);
67  y = y';
68
69  % Assemble the quadrature points and weights into matrices for
70  % two-dimensional numerical integration.
71  [X,Y] = meshgrid(x,y);
72  W = wy'*wx;
73
74  end
```

```
function [x w v] = legpts(n,int,meth)
% Obtained from
%
% http://www.mathworks.com/matlabcentral/fileexchange/23972-chebfun/
% content/chebfun/legpts.m
%
% on 1/29/2015 at 17:22.
%
% Copyright (c) 2015, The Chancellor, Masters and Scholars of the University
% of Oxford, and the Chebfun Developers
% All rights reserved.
%
% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided that the following conditions are
% met:
%
%     * Redistributions of source code must retain the above copyright
%       notice, this list of conditions and the following disclaimer.
%     * Redistributions in binary form must reproduce the above copyright
%       notice, this list of conditions and the following disclaimer in
%       the documentation and/or other materials provided with the
%       distribution
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
% AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
% IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
% ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
% LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
% CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
% SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
% INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
% CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
% ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
% POSSIBILITY OF SUCH DAMAGE.
%
%LEGPTS  Legendre points and Gauss Quadrature Weights.
%  LEGPTS(N) returns N Legendre points X in (-1,1).
%
%  [X,W] = LEGPTS(N) returns also a row vector W of weights for Gauss
%  quadrature.
%
%  LEGPTS(N,D) scales the nodes and weights for the domain D. D can be
%  either a domain object or a vector with two components. If the interval
```

```
44 %  is infinite, the map is chosen to be the default 'unbounded map' with
45 %  mappref('parinf') = [1 0] and mappref('adaptinf') = 0.
46 %
47 %  [X,W,V] = LEGPTS(N) returns additionally a column vector V of weights in
48 %  the barycentric formula corresponding to the points X.
49 %
50 %  [X,W] = LEGPTS(N,METHOD) allows the user to select which method to use.
51 %    METHOD = 'FASTSMALL' uses the recurrence relation for the Legendre
52 %       polynomials and their derivatives to perform Newton iteration
53 %       on the WKB approximation to the roots.
54 %    METHOD = 'FAST' uses the Glaser–Liu–Rokhlin fast algorithm, which
55 %       is much faster for large N.
56 %    METHOD = 'GW' will use the traditional Golub–Welsch eigenvalue method,
57 %       which is maintained mostly for historical reasons.
58 %       By default LEGPTS uses 'FASTSMALL' when N<=256 and FAST when N>256
59 %
60 %  See also chebpts and jacpts.
61
62 %  Copyright 2011 by The University of Oxford and The Chebfun Developers.
63 %  See http://www.maths.ox.ac.uk/chebfun/ for Chebfun information.
64
65 %  'GW' by Nick Trefethen, March 2009 — algorithm adapted from [1].
66 %  'FAST' by Nick Hale, April 2009 — algorithm adapted from [2].
67 %
68 %  References:
69 %    [1] G. H. Golub and J. A. Welsch, "Calculation of Gauss quadrature
70 %       rules", Math. Comp. 23:221–230, 1969,
71 %    [2] A. Glaser, X. Liu and V. Rokhlin, "A fast algorithm for the
72 %       calculation of the roots of special functions", SIAM Journal
73 %       on Scientific Computing", 29(4):1420–1438:, 2007.
```

```
function [Xout,Yout] = Local2Global(xdim,ydim,Gnodes,N1,N2,spacing)
% This function calculates global coordinates for a set of local
% coordinates in an element.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the local node number.
% N1 = horizontal coordinate in the local coordinate system where the
%    Hessian is to be evaluated.  N1 may be a scalar, vector, or matrix.
%    If N1 is a vector or a matrix it must have the same dimensions as N2.
% N2 = vertical coordinate in the local coordinate system where the Hessian
%    is to be evaluated.  N2 may be a scalar, vector, or matrix.  If N2 is
%    a vector or a matrix it must have the same dimensions as N1.
% spacing = a string that determines how the nodes are distributed along
%    each dimension.
%       'equal' = the nodes are equally spaced in [-1,1] in each coordinate
%           direction.
%       'cheb' = the nodes are placed at Chebyshev points of the 2nd-kind
%           in [-1,1] in each coordinate direction.
%
% OUTPUT:
% X_out = Global x-coordinates.
% Y_out = Global y-coordinates.

n_nodes = xdim.*ydim;

Xout = zeros(size(N1));
Yout = zeros(size(N1));

for k = 1:1:n_nodes

    int = Master_int2D(N1,N2,xdim,ydim,k,[0,0],0,[],spacing);

    Xout = Xout + int.*Gnodes(k,1);
    Yout = Yout + int.*Gnodes(k,2);
end

end
```

```
function [f] = Master_int2D(x,y,xdim,ydim,fnum,deriv,type,param,spacing)
% This function calculates the value of an interpolation function on a 2
% dimensional master element.
%
% INPUT:
% x = horizontal coordinate in the local coordinate system where the
%    function is to be evaluated.  x may be a scalar, vector, or matrix.
%    If x is a vector or a matrix it must have the same dimensions as y.
% y = vertical coordinate in the local coordinate system where the function
%    is to be evaluated.  y may be a scalar, vector, or matrix.  If y is a
%    vector or a matrix it must have the same dimensions as x.
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% fnum =  the index of the interpolation function to be evaluated.  The
%    interpolation functions are numbered in the same way as nodes.
% deriv = A 1x2 vector.
%    deriv(1,1) is the partial derivative of the interpolation function
%    with respect to x.
%    deriv(1,2) is the partial derivative of the interpolation function
%    with respect to y.
%       0 = returns the value of the interpolation function.
%       1 = returns the value of the interpolation function's first
%           derivative.
%       2 = returns the value of the interpolation function's second
%           derivative.
% type = a scalar indicating what type of interpolation is used in the
%    element.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a 1x(N-1) vector of the exponential parameters that affect the
%    shape of the function.  If type = 0, param = [] or all zeros.  If
%    type = 1, the size of param is 1x(xdim-1).  If type = 2, the size of
%    param is 1x(ydim-1).
% spacing = a string that determines how the nodes are distributed along
%    each dimension.
%       'equal' = the nodes are equally spaced in [-1,1] in each coordinate
%           direction.
```

```
%       'cheb' = the nodes are placed at Chebyshev points of the 2nd-kind
%           in [-1,1] in each coordinate direction.
%
% OUTPUT:
% f = the value of the function for each (x,y) coordinate.

% Determine the indices of the one-dimensional functions that will be used
% to construct the two-dimensional function.
yfnum = ceil(fnum./xdim);
xfnum = fnum - (yfnum-1).*xdim;

% Calculate node positions along each coordinate direction based on the
% specified spacing.
if strcmp(spacing,'equal')

    xi = -1:(2./(xdim-1)):1;
    yi = -1:(2./(ydim-1)):1;

elseif strcmp(spacing,'cheb')

    xi = chebpts(xdim);
    xi = xi';
    yi = chebpts(ydim);
    yi = yi';

end

% Calculate the values of the one-dimensional functions at x and y that
% will be used to construct the two-dimensional function.
switch type

    case 0

        % f is Lagrange in x and y direction.
        fx = Lagrange_int1D(x,xi,xfnum,deriv(1,1));
        fy = Lagrange_int1D(y,yi,yfnum,deriv(1,2));
        f = fx.*fy;

    case 1

        % f is exponential in x direction and Lagrange in y direction.
        fx = Exp_int1D(x,xi,param,xfnum,deriv(1,1));
        fy = Lagrange_int1D(y,yi,yfnum,deriv(1,2));
```

```
87          f = fx.*fy;
88
89      case 2
90
91          % f is Lagrange in x direction and exponential in y direction.
92          fx = Lagrange_int1D(x,xi,xfnum,deriv(1,1));
93          fy = Exp_int1D(y,yi,param,yfnum,deriv(1,2));
94          f = fx.*fy;
95
96  end
97
98  end
```

```
function [N] = Num_LegendrePts(xdim,ydim,type,param)
% This function calculates the number of Gauss-Legendre quadrature points
% required to integrate a one-dimensional interpolation function.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% type = a scalar indicating what type of interpolation is used in the
%    element.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a 1x(N-1) vector of the exponential parameters that affect the
%    shape of the function.  If type = 0, param = [] or all zeros.  If
%    type = 1, the size of param is 1x(xdim-1).  If type = 2, the size of
%    param is 1x(ydim-1).
%
% OUTPUT:
% N = the number of quadrature points.

% Calculate the number of quadrature points needed to integrate the
% exponential term of a one-dimensional exponential interpolation function.
% The parameters are multiplied by 4 because the components of [K] and {F}
% contain products of 4 interpolation functions.
Exp_N = Exp_1D_GLquad_Num(4.*param);

% Calculate the number of quadrature points needed to integrate a
% one-dimensional polynomial interpolation function.
Poly_degree = max(xdim,ydim) - 1;
Poly_N = ceil((6.*Poly_degree + 1)./2);

if (type == 1) || (type == 2)
    N = Poly_N + Exp_N;
else
    N = Poly_N;
end
end
```

```
function PlotElements(xdim,ydim,Gnodes,B)
% This function plots all of the elements in the domain and shows their
% element numbers, center point, and positive local axes.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same xdim.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.  It is assumed that every element has the
%    same ydim.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the global node number.
%    This matrix contains coordinates for all of the nodes in the domain.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.

% Repeat this loop for each element.
for j = 1:1:size(B,1)

    % Get the nodes for the current element.
    ElNodes = [Gnodes(B(j,:),1),Gnodes(B(j,:),2)];

    % Plot the element boundaries
    vertex = zeros(5,2);
    vertex(1,:) = ElNodes(1,:);
    vertex(2,:) = ElNodes(xdim,:);
    vertex(3,:) = ElNodes((xdim.*ydim),:);
    vertex(4,:) = ElNodes((xdim.*ydim-xdim+1),:);
    vertex(5,:) = ElNodes(1,:);

    plot(vertex(:,1),vertex(:,2),'k')
    hold on

end

axis equal
end
```

```
function [Gnodes,B] = RectDomain(xVert,yVert,x_el_dim,y_el_dim)
% This function generates node coordinates and a connectivity matrix for a
% rectangular region.
%
% INPUT:
% xVert = a vector of x-coordinates for the element vertices.
% yVert = a vector of y-coordinates for the element vertices.
% x_el_dim = the number of nodes along the horizontal direction of an
%    element in the local coordinate system.
% y_el_dim = the number of nodes along the vertical direction of an element
%    in the local coordinate system.
%
% OUTPUT:
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the global node number.
%    This matrix contains coordinates for all of the nodes in the domain.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.

% Number of elements along the x-direction.
N_el_x = max(size(xVert)) - 1;

% Number of elements along the y-direction.
N_el_y = max(size(yVert)) - 1;

% Initialize the output variables.
Gnodes = ones(N_el_x.*N_el_y.*x_el_dim.*y_el_dim,2).*NaN;
B = ones((N_el_x.*N_el_y),(x_el_dim.*y_el_dim)).*NaN;

% Initialize an index that is used to populate the rows of B.
B_row = 1;

for j = 1:1:N_el_y

    % Define the y-coordinates of the vertices for the current element.
    Gvert = zeros(4,2);
    Gvert(1,2) = yVert(1,j);
    Gvert(2,2) = yVert(1,j);
    Gvert(3,2) = yVert(1,(j+1));
    Gvert(4,2) = yVert(1,(j+1));

    for k = 1:1:N_el_x
```

```
44
45          % Define the x-coordinates of the vertices for the current element.
46          Gvert(1,1) = xVert(1,k);
47          Gvert(2,1) = xVert(1,(k+1));
48          Gvert(3,1) = xVert(1,k);
49          Gvert(4,1) = xVert(1,(k+1));
50
51          % Fill in the edge and interior nodes of the element.
52          [G_el_nodes] = Element_Mesh(Gvert,x_el_dim,y_el_dim,[]);
53
54          % Find out which of the element nodes are new.
55          [~,new_node_ind,~] =...
56              setxor(roundn(G_el_nodes,-5),roundn(Gnodes,-5),'rows');
57          new_node_ind = sort(new_node_ind);
58          new_nodes = G_el_nodes(new_node_ind,:);
59
60          % Put the new nodes into Gnodes.
61          next_ind = find(isnan(Gnodes(:,1)),1,'first');
62          Gnodes(next_ind:1:(next_ind+size(new_nodes,1)-1),:) = new_nodes;
63
64          % Get the global indices for all of the new nodes and put the
65          % indices into the next row of B.
66          [~,b] = ismember(roundn(G_el_nodes,-5),roundn(Gnodes,-5),'rows');
67          B(B_row,:) = b';
68
69          % Add 1 to B_row so that the next element fills in a vacant row in
70          % B.
71          B_row = B_row + 1;
72
73      end
74
75  end
76
77  % Delete any NaNs remaining in Gnodes.
78  nan_ind = find(isnan(Gnodes(:,1)),1,'first');
79  Gnodes = Gnodes(1:1:(nan_ind-1),:);
80
81  end
```

```
function [Gnodes_out,B_out] =...
    RefineElements(Gnodes_in,B_in,xdim_in,ydim_in,xdim_out,ydim_out)
% This function refines all of the elements in a domain.
%
% INPUT:
% Gnodes_in = the location of each node in the gobal coordinate system
%    before the elements are refined arranged in a Nx2 matrix.  Column 1 is
%    the x-coordinate.  Column 2 is the y-coordinate.  The row index is
%    equal to the global node number.  This matrix contains coordinates for
%    all of the nodes in the domain.
% B_in = the connectivity matrix.  B(i,j) = the global node number for
%    local node number j of element i.
% xdim_in = the number of nodes along the horizontal direction of an
%    element in the local coordinate system.  It is assumed that every
%    element has the same xdim_in.
% ydim_in = the number of nodes along the vertical direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same ydim_in.
% xdim_out = the number of nodes along the horizontal direction of an
%    element in the local coordinate system for the output elements.  It is
%    assumed that every element in the output has the same xdim_out.
% ydim_out = the number of nodes along the vertical direction of an element
%    in the local coordinate system for the output elements.  It is assumed
%    that every element in the output has the same ydim_out.
%
% OUTPUT:
% Gnodes_out = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix for the refined mesh.  Column 1 is the
%    x-coordinate.  Column 2 is the y-coordinate.  The row index is equal
%    to the global node number.  This matrix contains coordinates for all
%    of the nodes in the domain.
% B_out = the connectivity matrix for the refined mesh.  B(i,j) = the
%    global node number for local node number j of element i.

% Initialize the output variables.
Gnodes_out =...
    ones((size(B_in,1).*xdim_in.*ydim_in.*xdim_out.*ydim_out),2).*NaN;
B_out = ones((size(B_in,1).*xdim_in.*ydim_in),(xdim_out.*ydim_out)).*NaN;

% Initialize an index that is used to populate the rows of B.
B_row = 1;

for h = 1:1:size(B_in,1)
```

```
44
45      for j = 1:xdim_in:(xdim_in.*ydim_in - xdim_in - 1)
46
47          for k = j:1:(j + xdim_in - 2)
48
49              % Define the vertices of a new element.
50              Gvert = [Gnodes_in(B_in(h,k),:);...
51                       Gnodes_in(B_in(h,(k + 1)),:);...
52                       Gnodes_in(B_in(h,(k + xdim_in)),:);...
53                       Gnodes_in(B_in(h,(k + xdim_in + 1)),:)];
54
55              % Fill in the edge and interior nodes of the element.
56              [G_el_nodes] = Element_Mesh(Gvert,xdim_out,ydim_out,[]);
57
58              % Find out which of the element nodes are new.
59              [~,new_node_ind,~] =...
60                  setxor(roundn(G_el_nodes,-10),roundn(Gnodes_out,-10),'rows');
61              new_node_ind = sort(new_node_ind);
62              new_nodes = G_el_nodes(new_node_ind,:);
63
64              % Put the new nodes into Gnodes_out.
65              next_ind = find(isnan(Gnodes_out(:,1)),1,'first');
66              Gnodes_out(next_ind:1:(next_ind+size(new_nodes,1)-1),:) =...
67                  new_nodes;
68
69              % Get the global indices for all of the new nodes and put the
70              % indices into the next row of B_out.
71              [~,b] =...
72                  ismember(roundn(G_el_nodes,-10),roundn(Gnodes_out,-10),'rows');
73              B_out(B_row,:) = b';
74
75              % Add 1 to B_row so that the next element fills in a vacant row
76              % in B_out.
77              B_row = B_row + 1;
78
79          end
80      end
81  end
82
83  % Delete any NaNs remaining in Gnodes_out.
84  node_nan_ind = find(isnan(Gnodes_out(:,1)),1,'first');
85
86  if node_nan_ind ~= 0
```

```
87      Gnodes_out = Gnodes_out(1:1:(node_nan_ind-1),:);
88  end
89
90  % Delete any NaNs remaining in B_out.
91  B_nan_ind = find(isnan(B_out(:,1)),1,'first');
92
93  if B_nan_ind ~= 0
94      B_out = B_out(1:1:(B_nan_ind-1),:);
95  end
96
97  end
```

```
function [Gnodes,B] = TriDomain(TriVert,el_dim)
% This function generates node coordinates and a connectivity matrix for a
% triangular region.  It divides the triangle into 3 quadrilateral
% elements.
%
% INPUT:
% TriVert = the global coordinates of the triangular region arranged in a
%    Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is the
%    y-coordinate.
% el_dim = the number of nodes along each direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same number of nodes along the horizontal and vertical direction
%    and each element has the same number of nodes.
%
% OUTPUT:
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the global node number.
%    This matrix contains coordinates for all of the nodes in the domain.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.

% Initialize the output variables.
Gnodes = ones(3.*el_dim.^2,2).*NaN;
B = ones(3,el_dim.^2).*NaN;

% Calculate the coordinates of the center of the triangle.
Center = [sum(TriVert(:,1))./3, sum(TriVert(:,2))./3];

% Append the first two vertices to the end of TriVert to facilitate the
% loop iterations.
TriVert = [TriVert; TriVert(1:2,:)];

for j = 1:1:3

    % The midpoints of the sides are used as element vertices.
    midSide1 = [(TriVert(j,1) + TriVert(j+1,1))./2,...
        (TriVert(j,2) + TriVert(j+1,2))./2];
    midSide2 = [(TriVert(j,1) + TriVert(j+2,1))./2,...
        (TriVert(j,2) + TriVert(j+2,2))./2];

    % Define the coordinates of the vertices for the current element.
    Gvert = zeros(4,2);
```

```
44      Gvert(1,:) = TriVert(j,:);
45      Gvert(2,:) = midSide1;
46      Gvert(3,:) = midSide2;
47      Gvert(4,:) = Center;
48
49      % Fill in the edge and interior nodes of the element.
50      [G_el_nodes] = Element_Mesh_eql(Gvert,el_dim,el_dim,[]);
51
52      % Find out which of the element nodes are new.
53      [~,new_node_ind,~] = setxor(roundn(G_el_nodes,-10),roundn(Gnodes,-10),'rows');
54      new_node_ind = sort(new_node_ind);
55      new_nodes = G_el_nodes(new_node_ind,:);
56
57      % Put the new nodes into Gnodes.
58      next_ind = find(isnan(Gnodes(:,1)),1,'first');
59      Gnodes(next_ind:1:(next_ind+size(new_nodes,1)-1),:) = new_nodes;
60
61      % Get the global indices for all of the new nodes and put the
62      % indices into the next row of B.
63      [~,b] = ismember(roundn(G_el_nodes,-10),roundn(Gnodes,-10),'rows');
64      B(j,:) = b';
65
66  end
67
68  % Delete any NaNs remaining in Gnodes.
69  nan_ind = find(isnan(Gnodes(:,1)),1,'first');
70  Gnodes = Gnodes(1:1:(nan_ind-1),:);
71
72  end
```

```
function [Gnodes_out,U_out,Ave_out,Max_out,Norm_out] =...
    UpdateMesh(xdim,ydim,Gnodes,el_type,param,B,U,BC)
% This function implements a mesh adaptation scheme that is a slightly
% modified version of the one presented in:
%
% Ait-Ali-Yahia, D., Habashi, W. G., and Tam, A., "A Directionally Adaptive
% Methodology Using an Edge-Based Error Estimate on Quadrilateral Grids,"
% International Journal for Numerical Methods in Fluids, Vol. 23, 1996,
% pp. 673-690.
%
% INPUT:
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.  It is assumed that every element has
%    the same xdim.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.  It is assumed that every element has the
%    same ydim.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the global node number.
%    This matrix contains coordinates for all of the nodes in the domain.
% el_type = a vector indicating what type of interpolation is used in the
%    element.  Each entry in the vector is either a 0, 1, or 2.  The row
%    index of the vector corresponds to the element number.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a matrix of exponential parameters.  The row index of the matrix
%    corresponds to the element number.  Each row of the matrix contains
%    the exponential parameters for an element.  If an element type is 0,
%    the row in param for that element should contain all zeros.
% B = the connectivity matrix.  B(i,j) = the global node number for local
%    node number j of element i.
% U = the values of the dependent variables at each node.
% BC = a structure that holds the boundary conditions.
%       .essential = a vector the same size as U.  If an essential boundary
%           condition is specified for U(i), the value of the boundary
%           condition is stored in BC.essential(i).  All other elements of
%           BC.essential = NaN.  BC.essential is not used in this function.
%       .wall = a matrix the same size as B.  BC.wall(i,j) = B(i,j) for
```

```
44 %           node j of element i if that node is on a solid wall.  All other
45 %           elements of BC.wall = 0.  BC.wall is not used in this function.
46 %       .mesh = a column vector with twice the number of rows in Gnodes.
47 %           If the x-coordinate of node i does not move during mesh
48 %           adaptation, BC.mesh(2*i-1,1) = Gnodes(i,1).  If the
49 %           y-coordinate of node i does not move during mesh adaptation,
50 %           BC.mesh(2*i,1) = Gnodes(i,2).  All other elements of
51 %           BC.mesh = NaN.
52 %
53 % OUTPUT:
54 % Gnodes_out = the location of each node in the gobal coordinate system
55 %    for the new mesh.
56 % U_out = the values of the dependent variables at each node in the new
57 %    mesh.
58 % Ave_out = a vector containing the average distance the element vertices
59 %    moved during each iteration.
60 % Max_out = a vector containing the maximum distance the element vertices
61 %    moved during each iteration.
62 % Norm_out = a vector containing the norm of the distance the element
63 %    vertices moved divided by the number of element vertices for each
64 %    iteration.
65
66 % Initialize loop termination variables.
67 MaxIt = 18;
68
69 % Initialize output variables.
70 Gnodes_out = Gnodes;
71 U_out = U;
72 Ave_out = zeros(MaxIt,1);
73 Max_out = zeros(MaxIt,1);
74 Norm_out = zeros(MaxIt,1);
75
76 % Get the connectivity matrix for just the element vertices.
77 local_vert = [1,xdim,(xdim.*ydim - xdim + 1),(xdim.*ydim)];
78 Bvert = B(:,local_vert);
79
80 % Make a vector containing just the node numbers of the element vertices.
81 vert_index = unique(Bvert);
82
83 % Select the degree of freedom in U that will be used to modify the mesh.
84 Usub = U_out(4:4:end);
85
86 % Create interpolation structures to interpolate the degrees of freedom at
```

```
% new global coordinate locations.
el_inc = 0.1;  % interpolation increment on the local coordinate system
el_xy_size = max(size(-1:el_inc:1)).^2;
N_el = size(B,1); % number of elements

% Initialize vectors to contain the values of the interpolation points.
Xpoints = NaN.*ones((N_el.*el_xy_size),1);
Ypoints = NaN.*ones((N_el.*el_xy_size),1);
U1points = NaN.*ones((N_el.*el_xy_size),1);
U2points = NaN.*ones((N_el.*el_xy_size),1);
U3points = NaN.*ones((N_el.*el_xy_size),1);
U4points = NaN.*ones((N_el.*el_xy_size),1);

for j = 1:1:N_el

    % Get the global degrees of freedom for the current element.
    U1_el = U((4.*B(j,:)-3),1);
    U2_el = U((4.*B(j,:)-2),1);
    U3_el = U((4.*B(j,:)-1),1);
    U4_el = U((4.*B(j,:)),1);

    % Get the global coordinates of the nodes for the current element.
    enodes = Gnodes(B(j,:),:);

    % Get the interpolation points for the current element.
    [X,Y,U1,~,~,~,~,~,~] = Element2Grid(xdim,ydim,enodes,el_type(j,1),...
        param(j,:),U1_el,el_inc);
    [~,~,U2,~,~,~,~,~,~] = Element2Grid(xdim,ydim,enodes,el_type(j,1),...
        param(j,:),U2_el,el_inc);
    [~,~,U3,~,~,~,~,~,~] = Element2Grid(xdim,ydim,enodes,el_type(j,1),...
        param(j,:),U3_el,el_inc);
    [~,~,U4,~,~,~,~,~,~] = Element2Grid(xdim,ydim,enodes,el_type(j,1),...
        param(j,:),U4_el,el_inc);

    X = round(1e10.*X)./1e10;
    Y = round(1e10.*Y)./1e10;

    % Arrange the interpolation points into vectors.
    X = reshape(X,el_xy_size,1);
    Y = reshape(Y,el_xy_size,1);
    U1 = reshape(U1,el_xy_size,1);
    U2 = reshape(U2,el_xy_size,1);
    U3 = reshape(U3,el_xy_size,1);
```

```
130     U4 = reshape(U4,el_xy_size,1);
131
132     % Store the interpolation points.
133     Xpoints((j.*el_xy_size-(el_xy_size-1)):1:(j.*el_xy_size),1) = X;
134     Ypoints((j.*el_xy_size-(el_xy_size-1)):1:(j.*el_xy_size),1) = Y;
135     U1points((j.*el_xy_size-(el_xy_size-1)):1:(j.*el_xy_size),1) = U1;
136     U2points((j.*el_xy_size-(el_xy_size-1)):1:(j.*el_xy_size),1) = U2;
137     U3points((j.*el_xy_size-(el_xy_size-1)):1:(j.*el_xy_size),1) = U3;
138     U4points((j.*el_xy_size-(el_xy_size-1)):1:(j.*el_xy_size),1) = U4;
139
140 end
141
142 % Create interpolation structures that can be used later to interpolate the
143 % dependent variables at new global coordinates.
144 U1interp = TriScatteredInterp(Xpoints,Ypoints,U1points);
145 U2interp = TriScatteredInterp(Xpoints,Ypoints,U2points);
146 U3interp = TriScatteredInterp(Xpoints,Ypoints,U3points);
147 U4interp = TriScatteredInterp(Xpoints,Ypoints,U4points);
148
149 % Clear some variables to save memory.
150 clear Xpoints Ypoints U1points U2points U3points U4points
151
152 Gnodes_last = Gnodes;
153
154 % Modify the mesh for the specified number of iterations.
155 for R = 1:1:MaxIt
156
157     for S = 1:1:max(size(vert_index)) % repeat this for every element vertex
158
159         % This is the global index of the vertex that is being moved.  Only
160         % one vertex position is updated at a time.
161         vert = vert_index(S);
162
163         % Find the connectivity matrix for each element that shares the
164         % current vertex.
165         [Brow,~,~] = find(Bvert == vert);
166         Brow = sort(Brow);
167         Bsub = Bvert(Brow,:);
168
169         % Find the indices of vertices that are connected to the vertex
170         % that is being moved.
171         Bsub1 = Bsub;
172         for N = 1:1:size(Bsub1,1)
```

```
173
174             if Bsub1(N,1) == vert % bottom and left edges connect to node
175                 Bsub1(N,4) = 0;
176             elseif Bsub(N,2) == vert % bottom and right edges connect to node
177                 Bsub1(N,3) = 0;
178             elseif Bsub(N,3) == vert % top and left edges connect to node
179                 Bsub1(N,2) = 0;
180             else % top and right edges connect to node
181                 Bsub1(N,1) = 0;
182             end
183
184         end
185
186         node_index = unique(Bsub1);
187
188         % This deletes the 0 from node_index.
189         node_index = node_index(2:end);
190
191         % Create a vector of the spring stiffnesses for the vertices
192         % connected to the vertex that is being moved.
193         K_vector = zeros(max(size(node_index)),1);
194
195         for Q = 1:1:size(Bsub,1)
196
197             el_num = Brow(Q);
198             all_el_nodes = Gnodes(B(el_num,:),:);
199             U_in = Usub(B(el_num,:),1);
200
201             [K_el] = El_springK(xdim,ydim,all_el_nodes,...
202                 el_type(el_num,1),param(el_num,:),U_in,B(el_num,:),vert);
203
204             % Row and column indices for K_el
205             Ldofs = 1:1:4;
206
207             % Global degrees of freedom corresponding to Ldofs
208             Gdofs = Bvert(el_num,:);
209
210             % Ldof corresponding to the current moving vertex
211             vert_local = Ldofs(Gdofs == vert);
212
213             [tf,loc] = ismember(Gdofs,node_index);
214
215             K_vector(loc(tf),1) = K_vector(loc(tf),1) + K_el(tf,vert_local);
```

```
216
217         end
218
219         % Define separate stiffness vectors to update the x and y
220         % coordinates.
221         Kx_vector = K_vector;
222         Ky_vector = K_vector;
223
224         % Calculate the changes in x and y
225         x_vector = Gnodes_out(node_index,1);
226         y_vector = Gnodes_out(node_index,2);
227
228         % if the x coordinate is constrained:
229         if ~isnan(BC.mesh(2.*vert-1,1))
230
231             % do not move the coordinate in the x-direction.
232             delta_x = 0;
233
234             % only use the vertices at the edge of the domain to update the
235             % y coordinate.
236             Ky_vector = ~isnan(BC.mesh(2.*node_index-1,1)).*Ky_vector;
237
238         end
239
240         % if the y coordinate is constrained:
241         if ~isnan(BC.mesh(2.*vert,1))
242
243             % do not move the coordinate in the y-direction.
244             delta_y = 0;
245
246             % only use the vertices at the edge of the domain to update the
247             % x coordinate.
248             Kx_vector = ~isnan(BC.mesh(2.*node_index,1)).*Kx_vector;
249
250         end
251
252         % Calculate the distance that the x-coordinate moves.
253         if sum(Kx_vector) == 0
254             % Do this to prevent division by zero.
255             delta_x = 0;
256
257         elseif isnan(BC.mesh(2.*vert-1,1))
258
```

```
259             delta_x = sum((x_vector - ones(size(x_vector)).*...
260                 Gnodes_out(vert,1)).*Kx_vector)./sum(Kx_vector);
261 
262         end
263 
264         % Calculate the distance that the y-coordinate moves.
265         if sum(Ky_vector) == 0
266             % Do this to prevent division by zero.
267             delta_y = 0;
268 
269         elseif isnan(BC.mesh(2.*vert,1))
270 
271             delta_y = sum((y_vector - ones(size(y_vector)).*...
272                 Gnodes_out(vert,2)).*Ky_vector)./sum(Ky_vector);
273 
274         end
275 
276         % Calculate the sum of the absolute values of the components of the
277         % gradient of the dependent variable at the old node locations.
278         % This will be used later to make sure that new node positions are
279         % placed at locations with equal or greater gradients.
280         div_max = 0;
281 
282         for C = 1:1:size(Bsub,1)
283 
284             el_num = Brow(C);
285 
286             % Local vertex coordinates.
287             Eta1 = [-1 1 -1 1];
288             Eta2 = [-1 -1 1 1];
289 
290             % Global degrees of freedom corresponding to Ldofs
291             Gdofs = Bvert(el_num,:);
292 
293             % Ldof corresponding to the current moving vertex
294             Eta1 = Eta1(Gdofs == vert);
295             Eta2 = Eta2(Gdofs == vert);
296 
297             [J11,J12,J21,J22] = ElementJacobian(Eta1,Eta2,2,2,...
298                 Gnodes_out(Bsub(C,:),:));
299 
300             Jdet = J11.*J22 - J12.*J21;
301 
```

```
302             div_el = 0;
303
304             for A = 1:1:4
305
306                 dint_d1 = Master_int2D(Eta1,Eta2,2,2,A,[1,0],...
307                     el_type(el_num,1),param(el_num,end),'cheb');
308                 dint_d2 = Master_int2D(Eta1,Eta2,2,2,A,[0,1],...
309                     el_type(el_num,1),param(el_num,end),'cheb');
310
311                 div_el = div_el +...
312                     abs(((J22.*dint_d1 - J12.*dint_d2)./Jdet).*...
313                     Usub(Bsub(C,A),1)) +...
314                     abs(((J11.*dint_d2 - J21.*dint_d1)./Jdet).*...
315                     Usub(Bsub(C,A),1));
316             end
317
318             div_max = max(div_max,div_el);
319         end
320
321         % Uncomment the next line to make this constraint less restrictive.
322         %div_max = round(10.*div_max)./10;
323
324         % relaxation parameter
325         relax = 1;
326
327         for T = 1:1:15
328
329             % Calculate the new coordinate for the current vertex.
330             x_new = Gnodes_out(vert,1) + relax.*delta_x;
331             y_new = Gnodes_out(vert,2) + relax.*delta_y;
332
333             % Make sure the Jacobians are still positive and the elements
334             % are not too skewed.
335             Gnodes_temp = Gnodes_out;
336             Gnodes_temp(vert,1) = x_new;
337             Gnodes_temp(vert,2) = y_new;
338
339             % Interpolate the dependent variable at the new vertex
340             % location.
341             Usub_temp = Usub;
342             Usub_temp(vert,1) = U4interp(round(1e10.*[x_new,y_new])./1e10);
343
344             % If the interpolation returns NaN, use a different
```

```
345             % interpolation method.
346             if isnan(Usub_temp(vert,1))
347
348                 U4interp.Method = 'nearest';
349                 Usub_temp(vert,1) =...
350                     U4interp(round(1e10.*[x_new,y_new])./1e10);
351                 U4interp.Method = 'linear';
352
353             end
354
355             % Check the Jacobian of the newly formed elements and the
356             % gradient at the new vertex location.
357             [N1,N2] = meshgrid(-1:1:1,-1:1:1);
358             Jdet_min = 1;
359             JdetRatio_max = 1;
360             div_max_new = 0;
361
362             for Q = 1:1:size(Bsub,1)
363
364                 el_num = Brow(Q);
365
366                 % Local vertex coordinates.
367                 Eta1 = [-1 1 -1 1];
368                 Eta2 = [-1 -1 1 1];
369
370                 % Global degrees of freedom corresponding to Ldofs
371                 Gdofs = Bvert(el_num,:);
372
373                 % Ldof corresponding to the current moving vertex
374                 Eta1 = Eta1(Gdofs == vert);
375                 Eta2 = Eta2(Gdofs == vert);
376
377                 [J11_E,J12_E,J21_E,J22_E] =...
378                     ElementJacobian(Eta1,Eta2,2,2,Gnodes_temp(Bsub(Q,:),:));
379                 Jdet_E = J11_E.*J22_E - J12_E.*J21_E;
380
381                 [J11_N,J12_N,J21_N,J22_N] =...
382                     ElementJacobian(N1,N2,2,2,Gnodes_temp(Bsub(Q,:),:));
383                 Jdet_N = J11_N.*J22_N - J12_N.*J21_N;
384
385                 div_el = 0;
386
387                 for A = 1:1:4
```

```
388
389                     dint_d1 = Master_int2D(Eta1,Eta2,2,2,A,[1,0],...
390                         el_type(el_num,1),param(el_num,end),'cheb');
391                     dint_d2 = Master_int2D(Eta1,Eta2,2,2,A,[0,1],...
392                         el_type(el_num,1),param(el_num,end),'cheb');
393
394                     div_el = div_el +...
395                         abs(((J22_E.*dint_d1 - J12_E.*dint_d2)./Jdet_E).*...
396                         Usub_temp(Bsub(Q,A),1)) +...
397                         abs(((J11_E.*dint_d2 - J21_E.*dint_d1)./Jdet_E).*...
398                         Usub_temp(Bsub(Q,A),1));
399                 end
400
401                 div_max_new = max(div_max_new,div_el);
402
403                 Jdet_min = min(min(min(Jdet_N)),Jdet_min);
404
405                 JdetRatio = max(max(abs(Jdet_N)))./min(min(abs(Jdet_N)));
406                 JdetRatio_max = max(JdetRatio,JdetRatio_max);
407
408             end
409
410             % Uncomment the next line to make this constraint less restrictive.
411             %div_max_new = round(10.*div_max_new)./10;
412
413             % If a Jacobian determinant is negative, if the element is too
414             % skewed, or if the divergence is lower decrease the relaxation
415             % parameter.
416             if (Jdet_min <= 0) || (JdetRatio_max > 7.6) || (div_max_new < div_max)
417
418                 if T == 14
419                     relax = 0;
420                 else
421                     relax = 0.5.*relax;
422                 end
423             else
424                 break
425             end
426         end
427
428         for N = 1:1:size(Brow,1)
429
430             % Fill in the interior nodes for each element in Bsub and update
```

```
431             % Gnodes.
432             NewNodes = Element_Mesh(Gnodes_temp(Bsub(N,:),:),xdim,ydim,[]);
433             Gnodes_out(B(Brow(N,1),:),:) = NewNodes;
434 
435             % Find the value of Usub at each of the new nodes.
436             Usub(B(Brow(N,1),:),1) = U4interp(round(1e10.*NewNodes)./1e10);
437 
438             if max(max(isnan(Usub(B(Brow(N,1),:),1))))
439 
440                 U4interp.Method = 'nearest';
441                 Usub(B(Brow(N,1),:),1) = U4interp(NewNodes);
442                 U4interp.Method = 'linear';
443 
444             end
445 
446         end
447 
448     end
449 
450     % Calculate how far the nodes moved.
451     Diff_vector = sqrt((Gnodes_out(:,1) - Gnodes_last(:,1)).^2 + ...
452         (Gnodes_out(:,1) - Gnodes_last(:,1)).^2);
453 
454     AveDiff = sum(Diff_vector)./size(Diff_vector,1);
455     MaxDiff = max(Diff_vector);
456     DiffNorm = norm(Diff_vector)./size(Diff_vector,1);
457 
458     Ave_out(R,1) = AveDiff;
459     Max_out(R,1) = MaxDiff;
460     Norm_out(R,1) = DiffNorm;
461 
462     % Plot the difference.
463     figure(2)
464     plot(R,DiffNorm,'or')
465     hold on
466 
467     figure(3)
468     hold on
469     plot(R,MaxDiff,'*k')
470 
471     figure(4)
472     hold on
473     plot(R,AveDiff,'^b')
```

```
474
475     Gnodes_last = Gnodes_out;
476
477 end
478
479 % Interpolate all of the degrees of freedom on the new mesh.
480 U_out(1:4:end) = U1interp(round(1e10.*Gnodes_out)./1e10);
481 U_out(2:4:end) = U2interp(round(1e10.*Gnodes_out)./1e10);
482 U_out(3:4:end) = U3interp(round(1e10.*Gnodes_out)./1e10);
483 U_out(4:4:end) = U4interp(round(1e10.*Gnodes_out)./1e10);
484
485 end
```

```
function [xxQ,xyQ,yyQ] = WallBC(edge,xdim,ydim,Gnodes,type,param)
% This function generates the coefficient matrices for a solid wall
% boundary on a single element.
%
% INPUT:
% edge = a string that indicates which element edge is the solid wall
%    boundary when the element is transformed to local coordinates.  The
%    string should be either 'bottom', 'top', 'left', or 'right'.
% xdim = the number of nodes along the horizontal direction of an element
%    in the local coordinate system.
% ydim = the number of nodes along the vertical direction of an element in
%    the local coordinate system.
% Gnodes = the location of each node in the gobal coordinate system
%    arranged in a Nx2 matrix.  Column 1 is the x-coordinate.  Column 2 is
%    the y-coordinate.  The row index is equal to the local node number.
% type = a scalar indicating what type of interpolation is used in the
%    element.
%       0 = Lagrange polynomial interpolation in both the horizontal and
%           vertical direction
%       1 = Exponential interpolation in the horizontal direction and
%           Lagrange polynomial interpolation in the vertical direction
%       2 = Lagrange polynomial interpolation in the horizontal direction
%           and exponential interpolation in the vertical direction
% param = a vector of exponential parameters.  If type = 0, param is
%    ignored.
%
% OUTPUT:
% xxQ,xyQ,yyQ = the coefficient matrices for a solid wall boundary on a
%    single element.  yxQ is not calculated because yxQ = xyQ'.

% Generate the one dimensional Gauss-Legendre quadrature points and
% weights.
Qpoints = Num_LegendrePts(xdim,ydim,type,param);
[p,w] = legpts(Qpoints);
w = w';

% Generate the two dimensional quadrature points for integration along the
% bottom edge and the components of the unit normal vector on the bottom
% edge.
if strcmp(edge,'bottom')

    N1 = p;
    N2 = -1.*ones(size(p));
```

```
44
45      % Calculate components of the Jacobian
46      [J11,J12,J21,J22] = ElementJacobian(N1,N2,xdim,ydim,Gnodes);
47
48      % Define the determinant of the Jacobian
49      Jdet = J11.*J22 - J12.*J21;
50
51      % Calculate the normal vector components in global coordinates
52      RnormX = (1./Jdet).*J12;
53      RnormY = (1./Jdet).*-J11;
54
55      mag = sqrt(RnormX.^2 + RnormY.^2);
56
57      RnormX = RnormX./mag;
58      RnormY = RnormY./mag;
59
60  % Generate the two dimensional quadrature points for integration along the
61  % top edge and the components of the unit normal vector on the top edge.
62  elseif strcmp(edge,'top')
63
64      N1 = p;
65      N2 = ones(size(p));
66
67      % Calculate components of the Jacobian
68      [J11,J12,J21,J22] = ElementJacobian(N1,N2,xdim,ydim,Gnodes);
69
70      % Define the determinant of the Jacobian
71      Jdet = J11.*J22 - J12.*J21;
72
73      % Calculate the normal vector components in global coordinates
74      RnormX = (1./Jdet).*-J12;
75      RnormY = (1./Jdet).*J11;
76
77      mag = sqrt(RnormX.^2 + RnormY.^2);
78
79      RnormX = RnormX./mag;
80      RnormY = RnormY./mag;
81
82  % Generate the two dimensional quadrature points for integration along the
83  % left edge and the components of the unit normal vector on the left edge.
84  elseif strcmp(edge,'left')
85
86      N1 = -1.*ones(size(p));
```

```
87      N2 = p;
88
89      % Calculate components of the Jacobian
90      [J11,J12,J21,J22] = ElementJacobian(N1,N2,xdim,ydim,Gnodes);
91
92      % Define the determinant of the Jacobian
93      Jdet = J11.*J22 - J12.*J21;
94
95      % Calculate the normal vector components in global coordinates
96      RnormX = (1./Jdet).*-J22;
97      RnormY = (1./Jdet).*J21;
98
99      mag = sqrt(RnormX.^2 + RnormY.^2);
100
101     RnormX = RnormX./mag;
102     RnormY = RnormY./mag;
103
104 % Generate the two dimensional quadrature points for integration along the
105 % right edge and the components of the unit normal vector on the right
106 % edge.
107 elseif strcmp(edge,'right')
108
109     N1 = ones(size(p));
110     N2 = p;
111
112     % Calculate components of the Jacobian
113     [J11,J12,J21,J22] = ElementJacobian(N1,N2,xdim,ydim,Gnodes);
114
115     % Define the determinant of the Jacobian
116     Jdet = J11.*J22 - J12.*J21;
117
118     % Calculate the normal vector components in global coordinates
119     RnormX = (1./Jdet).*J22;
120     RnormY = (1./Jdet).*-J21;
121
122     mag = sqrt(RnormX.^2 + RnormY.^2);
123
124     RnormX = RnormX./mag;
125     RnormY = RnormY./mag;
126
127 end
128
129 % Number of nodes in the element.
```

```
130 N = xdim.*ydim;
131
132 % Initialize the output variables.
133 xxQ = zeros(N,N);
134 xyQ = zeros(N,N);
135 yyQ = zeros(N,N);
136
137 % Calculate the elements of the output vectors.
138 for j = 1:1:N
139
140     int_j = Master_int2D(N1,N2,xdim,ydim,j,[0,0],type,param);
141
142     parfor k = 1:1:N
143
144         int_k = Master_int2D(N1,N2,xdim,ydim,k,[0,0],type,param);
145
146         xxQ(j,k) = sum(w.*RnormX.^2.*int_j.*int_k.*Jdet);
147         xyQ(j,k) = sum(w.*RnormX.*RnormY.*int_j.*int_k.*Jdet);
148         yyQ(j,k) = sum(w.*RnormY.^2.*int_j.*int_k.*Jdet);
149
150     end
151
152 end
153
154 end
```

VITA

Bradford Scott Smith Jr.

Department of Mechanical and Aerospace Engineering
238 Kaufman Hall
Norfolk, VA 23529

Bradford Scott Smith Jr. earned a Bachelor of Science in Aerospace Engineering with a minor in Mathematics from Virginia Polytechnic Institute and State University in May 2007. He served in the United States Navy and currently works as an engineer at the Naval Surface Warfare Center, Dahlgren Division in Dahlgren, Virginia.