2018

A Localized Blended RBF Collocation Method for Effective Shock Capturing

Michael Harris

A LOCALIZED BLENDED RBF COLLOCATION METHOD FOR EFFECTIVE SHOCK CAPTURING

by

MICHAEL F. HARRIS

B.S. Florida Institute of Technology, 2007

M.S. Florida Institute of Technology, 2009

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Mechanical and Aerospace Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2018

Major Professor: Alain Kassab

ABSTRACT

Solving partial differential equations (PDEs) can require numerical methods, especially for non-linear problems and complex geometry. Common numerical methods used today are the finite difference method (FDM), finite element method (FEM) and the finite volume method (FVM). These methods require a mesh or grid before a solution is attempted. Developing the mesh can require expensive preprocessing time and the quality of the mesh can have major effects on the solution.

In recent years, meshless methods have become a research interest due to the simplicity of using scattered data points. Many types of meshless methods exist stemming from the spectral or pseudo-spectral methods, but the focus of this research involves a meshless method using radial basis function (RBF) interpolation. Radial basis functions (RBF) interpolation is a class of meshless method and can be used in solving partial differential equations. Radial basis functions are impressive because of the capability of multivariate interpolation over scattered data, even for data with discontinuities. Also, radial basis function interpolation is capable of spectral accuracy and exponential convergence. For infinitely smooth radial basis functions such as the Hardy Multiquadric and inverse Multiquadric, the RBF is dependent on a shape parameter that must be chosen properly to obtain accurate approximations. The optimum shape parameter can vary depending on the smoothness of the field. Typically, the shape parameter is chosen to be a large value rendering the RBF flat and yielding high condition number interpolation matrix. This strategy works well for smooth data and as

shown to produce phenomenal results for problems in heat transfer and incompressible fluid dynamics. The approach of flat RBF or high condition matrices tends to fail for steep gradients and shocks. Instead, a low-value shape parameter rendering the RBF steep and the condition number of the interpolation matrix small should be used in the presence of steep gradients or shocks.

This work demonstrates a method to capture steep gradients and shocks using a blended RBF approach. The method switches between flat and steep RBF interpolation depending on the smoothness of the data. Flat RBF or high condition number RBF interpolation is used for smooth regions maintaining high accuracy. Steep RBF or low condition number RBF interpolation provides stability for steep gradients and shocks. This method is demonstrated using several numerical experiments such as 1-D advection equation, 2-D advection equation, Burgers equation, 2-D inviscid compressible Euler equations, and the Navier-Stokes equations.

To my family.

Foster Harris, Gail Harris, Mindy Harris

Earl Harris

Courtney Harris

For giving me their support and encouragement.

ACKNOWLEDGMENTS

The author would like to thank Dr. Alain J. Kassab and Dr. Eduardo A. Divo for suggesting this interesting topic. Their expertise and guidance in the subject made this research a success.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The governing equations typically used in science and engineering, especially in fluid dynamics and heat transfer, are represented in the form of partial differential equations (PDE). For most engineering applications, we seek solutions to nonlinear partial differential equations over complex geometry. These types of problems require the use of numerical methods. The methods used today include finite difference method (FDM), finite element method (FEM), and the finite volume method (FVM). These methods are well developed and are the most common methods used today for engineering analysis, but these methods require a preprocessing step of developing a mesh. The mesh, grid or connectivity within the domain is used to discretely solve the governing partial differential equations. These equations are iteratively solved along the mesh or grid until convergence. The focus of research in computational methods has been to reduce the computational time of the solution while mesh generation remained a bottleneck. This meshing preprocessing step can be expensive requiring significant development time. Mesh generation can be even more time-consuming in the modeling of shocks since the location of shocks are unknown before the solution, sometimes requiring the mesh to be adapted based on the solution. Also, the quality of the mesh is important where a poor-quality mesh could cause convergence issues. Generating a high-quality mesh while balancing computation resource and still provide enough resolution to capture the physics can be an iterative process.

In the past few decades, meshless methods have become a research area of interest for solving partial differential equations, especially in the use of radial basis functions to interpolate scattered data. Radial basis functions (RBF) such as the Hardy multiquadrics, Hardy inverse multiquadrics, Gaussian, and thin plate splines are a few RBFs used in literature. The Hardy multiquadrics and inverse multiquadrics are the most impressive functions as these radial basis functions produce the best accuracy. However, Hardy multiquadrics and inverse multiquadrics radial basis functions depend on a shape parameter that must be chosen properly for accurate approximations. Researchers tend to choose a shape parameter to render the RBF flat and the interpolation matrix close to ill-conditioned. This approach works well for smooth functions, but this approach tends to cause oscillations for data where steep gradients or shocks exist. Reducing the shape parameter to render the RBF steep and the interpolation matrix condition number low eliminates oscillations capturing the steep gradients. A blended radial basis function interpolation scheme is proposed in this dissertation. The scheme seeks to use the accuracy of flat RBF interpolation in smooth regions., while benefiting from the stability of low shape parameter radial basis function for regions where steep gradient and shocks appear.

The structure of this dissertation is the following. Chapter 2, provides a review of the literature in radial basis function interpolation starting with the introduction of the Hardy multiquadrics and continuing with the RBF interpolation encounters of Kansa and others. The literature on RBF interpolation has shown impressive capabilities of the

Hardy multiquadrics as this radial basis function outperforms other interpolation methods. However, research over the years has focused heavily on flat RBF interpolation and finding the optimal shape parameter.

In Chapter 3, we present the methodology of radial basis function interpolation. A detailed explanation of the mathematical steps taken for RBF interpolation is shown and how the RBF interpolation is used to solve partial differential equations. The method of moving least squares is also presented for completeness. Moving least squares can add stability if oscillations are produced as this method does not interpolate, although this method can add significant dissipation to the solution.

Chapter 4 provides the formulation of the blended RBF interpolation scheme and criterion used to switch between high and low shape parameter interpolation. For clarity, the author would like to distinguish the definition of high and low-value shape parameter interpolation. We use the traditional form of the Hardy Multiquadrics written as $\chi_j(x) = \sqrt{r_j^2(x) + c^2}$, where c is noted as the shape parameter. Therefore, we vary the shape parameter c between some limits of low or high value depending on the smoothness of the field. High value shape parameter interpolation allows for high accuracy, while low value shape parameter interpolation allows for stability near steep gradients and shocks. The criterion for switching the RBF interpolation is presented using a few different approaches.

Chapter 5 discusses the upwinding schemes used to solve advective derivatives. The inviscid flux terms are evaluated using a variety of different schemes such as Steger-Warming, Van Leer, and Liu AUSM schemes. Flux splitting schemes of Steger-

Warming and Van Leer are used because the formulation is favorable for RBF direct differentiation. The advection upwinding splitting method (AUSM) schemes are also studied to show the versatility of using RBF interpolation. RBF interpolation allows for interpolation of field variables as input to flux splitting schemes or to directly solve for the derivatives.

Chapter 6 presents several numerical experiment results that demonstrate the advantage of using high-value shape parameter RBF interpolation for smooth data and low condition RBF interpolation for discontinuous data. The chapter begins by demonstrating the effect of shape parameter in reconstructing a test function using RBF interpolation. Large shape parameter interpolation works well for smooth function reconstructions producing very low errors in comparison, but as the gradient is increased the RBF interpolation diverges producing oscillations. A case follows showing how the oscillations can be eliminated by lowering the shape parameter. Next, the RBF interpolation is used to solve partial differential equations beginning with 1-D examples, such as the 1-D advection and Burgers' equation. In these examples, we show how the local reconstruction of the field can become unstable by using a constant high-value shape parameter causing overshoots and undershoots. Eventually, these overshoots and undershoots of the field reconstructions cause the solution to become unstable. The instability can be prevented by allowing the solver to switch to low shape parameter RBF interpolation. The one-dimensional numerical experiments show that the switch allows for RBF interpolation for steep gradients indicating that this approach could work for the solution of convective dominant flows.

The one-dimensional and the two-dimensional numerical experiments show the performance of the blended RBF interpolation. The classic problems of a diagonal 45° wave and the turning wave provide a qualitative measure of the dissipation introduced by the blending scheme. These 2-D numerical experiments prove the RBF interpolation must use some form of blending between high and low-value RBF interpolation. Constant low shape parameter interpolation causes dissipation of solution smearing out the wave. By using the blended RBF interpolation scheme, the magnitude of the wave remains sharp giving retaining the shape of the initial wave introduced into the domain.

Finally, the blended RBF interpolation is used to solve the inviscid compressible Euler equations and the Navier-Stokes equations. We make use of flux limiters and smoothness indicators found in developed schemes to detect steep gradients and blend the radial basis function interpolation. The methodology again shows the added accuracy and stability in solving problems with steep gradients and shocks. The RBF interpolation schemes can now be used for high convective flows using the blended RBF approach, allowing for the versatility of using scatter data and reconstruction of data to create stencils. The conclusions and potential opportunities to carry this research further are summarized in Chapter 7. A MATLAB script and all functions are provided in the appendices to guide future researchers.

CHAPTER 2

LITERATURE REVIEW

Meshless methods have been a research interest for many years and shown to have advantages over other numerical methods. These methods can be used to solve the governing equations for many applications in science and engineering, such as fluid dynamics, heat transfer, and structural mechanics [1]. There are many different types of meshless methods that have been developed over the years, and a large portion of literature can be found on the subject [2, 3, 4, 5, 6, 7, 1, 8, 9, 10]. Meshless methods are a group of numerical techniques that use a unifying concept of global or local interpolation methods. However, the type of interpolation is different for each method. Meshless methods originate from spectral or pseudo-spectral methods that are based on Legendre and Chebyshev polynomials, which require the point or nodal distribution to be uniform [11, 12, 13]. However, the adoption of radial basis functions (RBF) allows meshless methods to use irregular spatial distribution of points or nodes [11, 12, 13]. The focus of this work will be on meshless methods involving radial basis function (RBF) interpolation. Examples of radial basis functions in literature are listed in Table 1 and shown graphically in Figure 1.

Table 1: Radial Basis Functions

Name	Expression	Shape Parameter
Hardy Multiquadric (MQ)	$\chi(r) = (r^2 + c^2)^q$	$c, q = \frac{1}{2}$
Inverse Hardy Multiquadric (IMQ)	$\chi(r) = \frac{1}{(r^2 + c^2)^q}$	$c, q = \frac{1}{2}$
Gaussian	$\chi(r) = e^{-cr^2}$	c
Thin Plate Splines (TPS)	$\chi(r) = r^{\eta}$	η

Radial basis functions have been a subject of research for decades as a global collocation method in solving partial differential equations. The Hardy Multiquadrics RBF was developed for mapping topographical data and has proven to be an impressive function for multivariate interpolation [14, 15]. To determine to the best approach for scattered data interpolation, Franke tested 29 different algorithms [16]. Franke determined that the Hardy Multiquadric RBF produce stable results and yields the most accurate results of all other methods studied. These interpolation methods were soon used to solve partial differential equations (PDEs). Kansa developed the RBF collocation meshless method using Hardy Multiquadric to solve partial differential equations. [5, 6] The Kansa method was used to solve hyperbolic and elliptic equations such as the linear advection-diffusion equation and the Poisson equation. These numerical experiments demonstrated the efficiency of using RBF multiquadric in contrast to the finite difference method. In the Kansa method, the RBFs are applied globally in strong form to solve partial differential equations. However, the method becomes intractable when using large datasets due to ill-conditioning of the interpolation matrix. Domain decomposition

[11] and matrix preconditioning [17, 18] have been employed to address the difficulties of large data sets and ill-conditioned matrices. Many researchers have switched from global collocation and are using a local RBF collocation developed by Sarler [19, 20, 21, 22, 23, 24, 25, 26, 27, 28] [29]. Divo and Kassab have shown success in using the localized RBF collocation in heat transfer and fluid dynamics [27, 28, 29, 30].

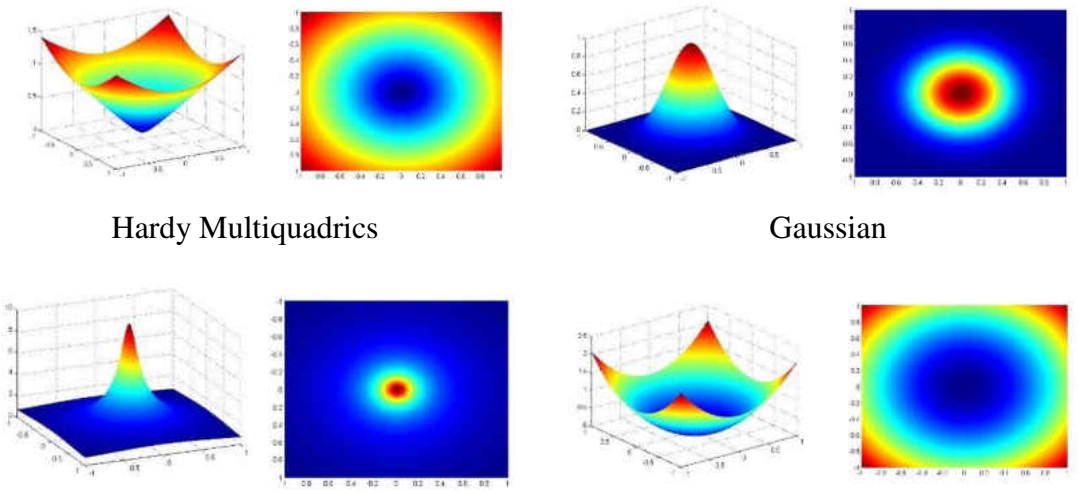


Figure 1: Graphical representation of typical radial basis functions

RBF interpolation using Hardy Multiquadric RBF has shown to provide high accuracy and exponential convergence [27, 28, 29, 30]. However, the Hardy multiquadrics and inverse multiquadrics are dependent on the shape parameter, c. The optimal shape parameter for a given problem can be difficult to determine, and many researchers have focused their attention on finding a solution for the shape parameter

problem [7, 31, 32, 33, 34, 29, 35, 36]. For best accuracy, the shape parameter should be chosen to be a large value causing the condition number of the interpolation matrix to be close to ill-conditioned, rendering the RBF flat [27, 37, 28, 29, 38]. The effect on shape parameter on the Hardy Multiquadrics is shown in Figure 2 and Figure 3. These figures illustrate the fact that the Hardy Multiquadric RBF tends to flatten as the shape parameter, c, is increased. Spectral convergence can be reached as the shape parameter $c \to \infty$. [27, 39, 40]. As $c \to 0$, the Multiquadric RBF becomes steep and the accuracy is piecewise linear. However, low shape parameter can be used to add stability in areas of steep gradients and shocks.

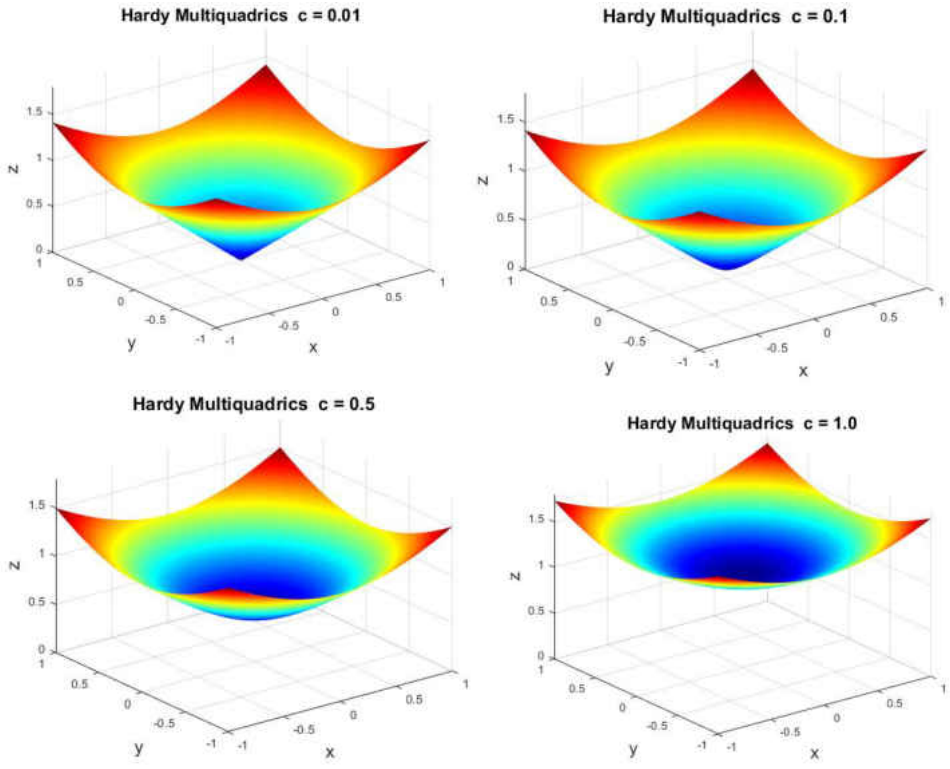


Figure 2: Hardy Multiquadrics for varying shape parameter, c

Many algorithms have been developed to maintain stability for flat RBF interpolation. Rippa minimized a cost function to mimic the error of the radial basis interpolant and the unknown function [41]. Fornberg and Wright developed a method that provides stable results for all values of the shape parameter, but the shape parameter must be transformed to the complex space [42]. Fornberg, Larsson, and Flyer used the RBF-QR and implemented this algorithm for general domains [43]. Kansa and Carlson have proposed using a variable shape parameter, but this approach causes the coefficient matrix to be no longer symmetric [44]. Siraj-ul-Islam, Vertnik, and Sarler used scaled the shape parameter or the local support to maintain stability [45, 46, 47]. Golberg, Chen, and Karur suggest using a cross-validation technique to determine the shape parameter [48].

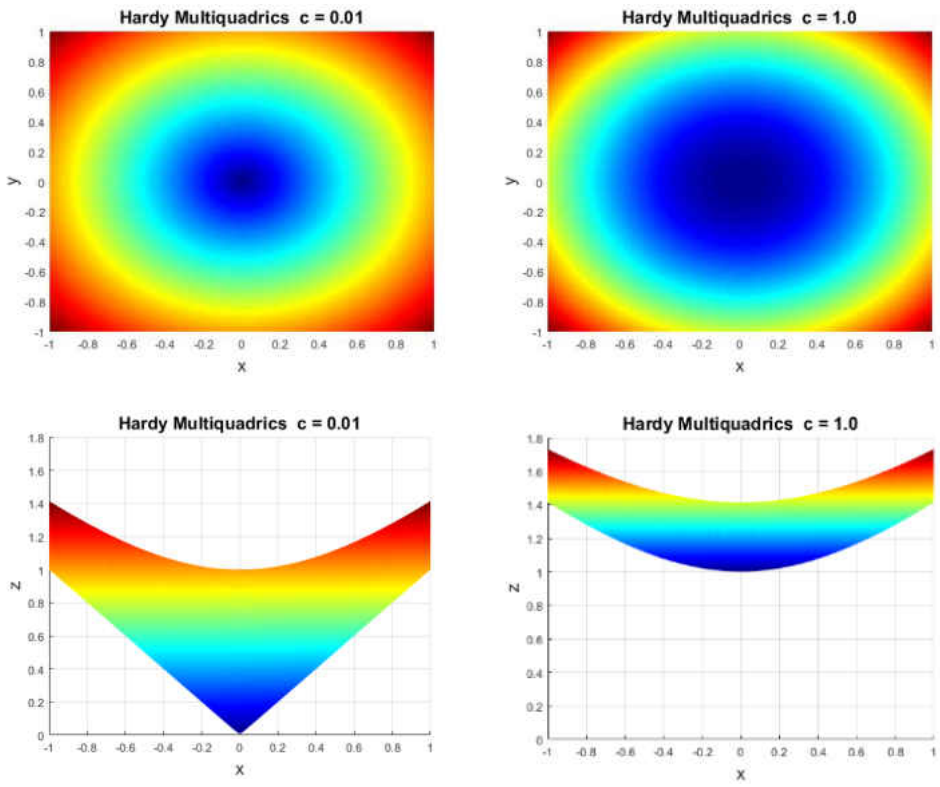


Figure 3: Hardy Multiquadrics for varying shape parameter, *c* in XY and XZ plane

However, the research has focused heavily on overcoming the ill-conditioning of RBF interpolation always seeking flat RBF interpolation for all types of problems. The choice of shape parameter is affected by many factors. The number of data points and the location of the data points used tends to affect the value of the shape parameter needed for accurate approximations [41]. Carlson and Foley observed that the shape parameter value is dependent on the function and is independent of the number and location of data points [49, 50]. Their work demonstrates Hardy Multiquadric interpolation on topography that lies along the same plane and the distance between the data points can vary by orders of magnitude. As the shape parameter approached zero, they observed the Hardy Multiquadrics interpolation yielding better accuracy in those

regions. For univariate interpolation, as the shape parameter goes to zero, the Multiquadrics RBF simply yields a piecewise linear interpolant [49, 50]. The issue was due to steep gradient oscillations within the measured data while using relatively high shape parameter values of $c = 0.1$ and $c = 0.01$. Carlson and Foley recommended the shape parameters to be reduced enough to remove any excursions caused by the interpolation due to some measurement's high gradients. With some simple numerical experiments, one could determine that the shape parameter for the Hardy Multiquadrics depends on the smoothness of the function to be interpolated. The authors observed that small values of the shape parameter could resolve the issue of these excursions while using RBF interpolation on rapidly varying data. Although the shape parameter must be reduced, they found that multiquadrics RBF yielded accurate results producing smaller RMS errors than all other methods investigated. The suggestion of using a linear interpolant may not be a favorable strategy for some since spectral accuracy is lost, but using steep RBF interpolation can provide stability in specific regions of high gradient.

Jung and Durante [51] also recommended the notion of reducing the shape parameter to zero for the Multiquadrics RBF near local discontinuities. Their algorithm sweeps through the data and lowers the shape parameter to zero based on a developed smoothness criterion consisting of the derivative weighted with the RBF expansion coefficients at the data center, x_i. Simply, the algorithm switches the shape parameter to zero for each collocation point if a jump discontinuity is detected. This iterative procedure proposed by the authors works very well eliminating oscillations in the data and capturing the discontinuities.

For many engineering problems, we seek solutions to the fluid dynamics governing equation such as the Navier-Stokes equations. The Navier-Stokes equations can be difficult to solve especially for highly convective problems. Meshless methods using RBFs have been used to solve the Navier-Stokes equations for low Reynolds number flows that accommodates the RBF behavior for smooth functions. Highly convective flows tend to have steep gradients and shocks. As mentioned before, high gradients can cause instabilities with RBF interpolation and caution must be used in these high gradient regions. Many different approaches using RBF schemes have been reported in solving highly convective flows using RBFs [46, 47, 52, 53, 54, 55]. Most of these proposed schemes used scaling, slope limiters, unsymmetric radial basis functions to solve hyperbolic problems.

Shock capturing schemes capable of capturing the discontinuities while limiting numerical dissipation was a challenge in the computational fluid dynamics community until the 1980s saw advancement in upwind schemes. Two types of upwinding techniques were produced during this time such as the flux differencing schemes and the flux vector splitting schemes. Roe introduced flux difference scheme that is still used in commercial CFD today [56]. The Roe scheme provides good accuracy but can be computationally intensive, and entropy corrections are needed to overcome expansion shocks. Steger-Warming [57] introduced flux vector splitting methods which are computationally efficient but tends to give inaccurate results near sonic or stagnation regions. Van Leer [58] attempted to correct the discontinuous issues of the Steger-Warming scheme, but the scheme can introduce significant dissipation, especially for

viscous problems. The advection upwinded splitting method, or AUSM scheme was developed by Liou [59] has proven to be a robust scheme even for near vacuum flows.

A new approach for solving hyperbolic PDEs using a localized blended RBF collocation method [60, 61, 62, 63] is proposed in this dissertation. The blended RBF strategy is to combine the benefits of flat RBF and steep RBF interpolation to form a high-resolution scheme [64, 65, 66] for effective shock capturing. The methodology of this scheme is described in the following chapters, and governing equations for high-speed compressible flow are solved.

CHAPTER 3

MESHLESS METHODOLOGY

3.1 Radial Basis Function Interpolation

Radial basis function interpolation is implemented by setting the solution of the function equal to the expansion described by [5, 14, 67].

$$f(x) = \sum_{j=1}^{N} \alpha_j \chi_j(x) \quad (1)$$

where α_j are the expansion coefficients, $\chi_j(x)$ is the expansion function and N is the number of collocation points. The coefficients α_j are determined through a collocation process of the function $f(x)$ at the collocation points, $f(x_i)$ for $i = 1 \ldots N$.

$$f(x_i) = \sum_{j=1}^{N} \alpha_j \chi_j(x_i) \quad (2)$$

The equations can be presented in matrix form as

$$\{f\} = [\mathrm{C}]\{\alpha\} \quad (3)$$

where the collocation matrix $[\mathrm{C}]$ is of the form

$$[\mathrm{C}] = \begin{bmatrix} \chi_1(x_1) & \cdots & \chi_N(x_1) \\ \vdots & \ddots & \vdots \\ \chi_1(x_N) & \cdots & \chi_N(x_N) \end{bmatrix} \quad (4)$$

The expansion functions, $\chi_j(x)$, are chosen to be a type of radial basis function. Many radial basis functions exist, but for the scope of this disseration, only the Multiquadric and the Inverse Multiquadric are considered.

Multiquadric RBF:

$$\chi_j(x) = \sqrt{\left[r_j^2(x) + c^2\right]} \tag{5}$$

Inverse multiquadric RBF:

$$\chi_j(x) = \frac{1}{\sqrt{\left[r_j^2(x) + c^2\right]}} \tag{6}$$

The coefficients, α_j, are then found by solving the linear system by

$$\{\alpha\} = [\mathrm{C}]^{-1}\{f\} \tag{7}$$

With expansion coefficients known, the function or field variable can be reconstructed by recalling Eqn. (3)

For constant and linear fields, the RBF can be augmented by a polynomial expansion given by, see [67].

$$f(x) = \sum_{j=1}^{N} \alpha_j \chi_j(x) + \sum_{j=1}^{NP} \alpha_j P_j(x) \tag{8}$$

where N is the number collocation points and NP is the number of polynomials. The polynomials $P_j(x)$ can be for example $P = [1, x]$. For two-dimensional problems, $P = [1, x, y]$. This addition of the polynomial expansion provides stability in constant and

linear fields and along the boundary of the domain. Although, the added polynomial expansion can reduce accuracy in some cases [49, 50]

The procedure is the same as the RBF interpolation described previously by solving for the expansion coefficients α_j. The collocation matrix is of the form

$$[\mathrm{C}] = \begin{bmatrix} \chi_1(x_1) & \dots & \chi_N(x_1) & P_1(x_1) & \dots & P_{NP}(x_1) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \chi_1(x_N) & \dots & \chi_N(x_N) & P_1(x_N) & \dots & P_{NP}(x_N) \\ P_1(x_1) & \dots & P_1(x_N) & 0 & \dots & 0 \\ \dots & \ddots & \dots & \vdots & \ddots & \vdots \\ P_{NP}(x_1) & \dots & P_{NP}(x_N) & 0 & \dots & 0 \end{bmatrix}_{N+NP,N+NP} \tag{9}$$

moreover, the vector of know values is modified by

$$\{f\} = \begin{Bmatrix} f(x_1) \\ \vdots \\ f(x_N) \\ 0 \\ \vdots \\ 0 \end{Bmatrix}_{N+NP,1} \tag{10}$$

3.2 Local RBF Collocation Meshless Method

Early RBF collocation methods such as the Kansa method, are global methods and can be difficult to use for large-scale problems. Domain decomposition has been used to solve the difficulties of global methods. Another strategy is to use a local RBF collocation meshless method [11, 12, 13]. The localized RBF collocation meshless method (LRCMM) is described briefly in this section and for more details, see [67]. The LRCMM uses a localized RBF expansion to interpolate the field variables and

approximate the derivatives. A set of data points is sampled around the data center (x_i, y_i) as shown in Figure 4.

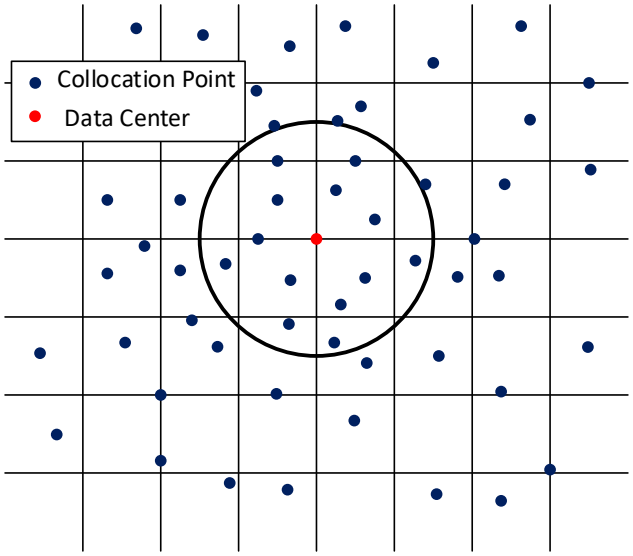


Figure 4: Local RBF Collocation Meshless Method for 2-D problem

The function $f(x, y)$ can be approximated by the expansion

$$f(x,y) = \sum_{j=1}^{N} \alpha_j \chi_j(x,y) + \sum_{j=1}^{NP} \alpha_{j+N} P_j(x) \qquad (11)$$

Rewriting the equation in the vector-matrix notation, the function is evaluated by the product.

$$f(x,y) = \{C(x,y)\}^T \{\alpha\} \qquad (12)$$

where the expansion function $\chi_j(x, y)$ can be chosen to be the Hardy Multiquadric given by

$$\chi_j(x,y) = \sqrt{(x - x_j)^2 + (y - y_j)^2 + c^2} \qquad (13)$$

and c is the shape parameter which is a chosen value that provides the best accuracy. A linear system of equations can be formed from the local nodes near the data center (x_i, y_i). The coefficients $\{\alpha\}$ are solved by matrix inversion.

$$\{f\} = [\mathrm{C}]\{\alpha\} \quad (14)$$

$$\{\alpha\} = [\mathrm{C}]^{-1}\{f\} \quad (15)$$

Substituting Eqn. (15)into Eqn. (12), we can interpolate the field variable value anywhere within the domain by

$$f(x,y) = \{C(x,y)\}^T[C]^{-1}\{f\} \quad (16)$$

The derivatives can be approximated by applying the derivative operator over the local expansion as

$$\frac{\partial f(x,y)}{\partial x} = \left\{\frac{\partial C(x,y)}{\partial x}\right\}^T [C]^{-1}\{f\} \quad (17)$$

$$\frac{\partial f(x,y)}{\partial y} = \left\{\frac{\partial C(x,y)}{\partial y}\right\}^T [C]^{-1}\{f\} \quad (18)$$

The vectors $\{C(x,y)\}^T[C]^{-1}\{f\}$, $\left\{\frac{\partial \mathrm{C}(x,y)}{\partial \mathrm{x}}\right\}^T [\mathrm{C}]^{-1}$ and $\left\{\frac{\partial \mathrm{C}(x,y)}{\partial \mathrm{y}}\right\}^T [\mathrm{C}]^{-1}$ can be pre-computed and saved for each data centre prior to the computation. Therefore, the derivatives of field variables can be approximated by the product of $\left\{\frac{\partial \mathrm{C}(x,y)}{\partial \mathrm{x}}\right\}^T [\mathrm{C}]^{-1}$ and the field variable values within the subdomain.

$$f(x,y) = \{C(x,y)\}^T[C]^{-1}\{f\} \quad (19)$$

$$\frac{\partial f(x,y)}{\partial x} = \left\{\frac{\partial C(x,y)}{\partial x}\right\}^T [C]^{-1}\{f\} \quad (20)$$

$$\frac{\partial f(x,y)}{\partial y} = \left\{\frac{\partial C(x,y)}{\partial y}\right\}^T [C]^{-1}\{f\} \quad (21)$$

3.3 Radial Basis Function Enhanced Finite Difference

The RBF enhanced finite difference uses the impressive capabilities of the localized RBF interpolation to interpolate "virtual" points within the domain to formulate a finite difference or finite volume stencil [67, 68]. For high-speed flow applications, the direction which information is passed is important to maintaining a stable solution. The enhanced finite difference is an elegant strategy developed to solve this issue with RBF interpolation when approximating the derivatives in regions where highly convective flow, steep gradients or discontinuities are present. By the creation of finite difference stencil, the well-developed upwind schemes in computational fluid dynamics can be used. For example, this strategy allows for the use of schemes such as Roe's scheme, and AUSM schemes and other high-resolution schemes as well as flux limiters. Figure 5 illustrates how the RBF enhanced finite difference is implemented.

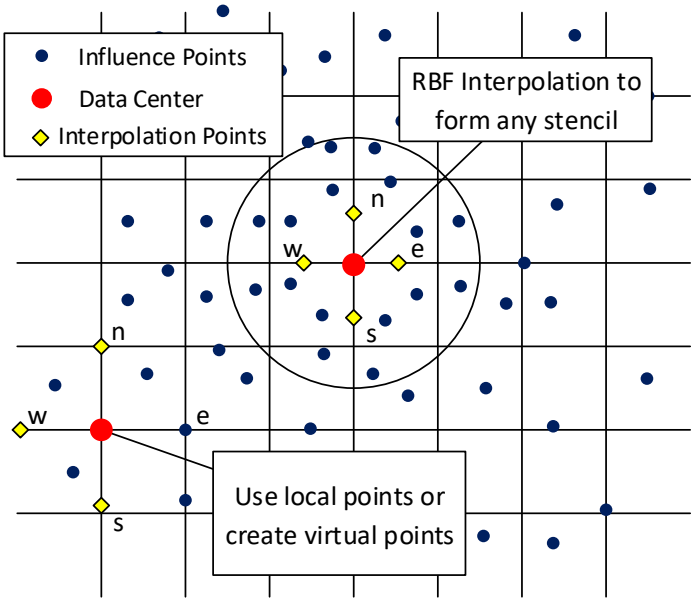


Figure 5: RBF Enhanced Finite Difference for Two Dimensional problems

We interpolate the values to form a finite difference stencil, i.e., the east, west, north, and south points. The localized expansion is evaluated at the stencil locations.

$$f(x_e, y_e) = \{C(x_e, y_e)\}^T [C]^{-1} \{f\} \quad (22)$$

$$f(x_w, y_w) = \{C(x_w, y_w)\}^T [C]^{-1} \{f\} \quad (23)$$

$$f(x_n, y_n) = \{C(x_n, y_n)\}^T [C]^{-1} \{f\} \quad (24)$$

$$f(x_s, y_s) = \{C(x_s, x_s)\}^T [C]^{-1} \{f\} \quad (25)$$

3.4 Moving Least Squares

The localized RBF blended scheme can still produce oscillations in some scenarios. If instabilities occur, the scheme can switch to a moving least square (MLS) scheme to approximate the derivatives [69, 67]. The MLS formulation is described in this section.

The field variable $f(x)$ can be approximated by the expansion

$$f(x) = \sum_{j=1}^{NP} \alpha_j P_j \qquad (26)$$

where α_j are the expansion coefficients, NP are the number of polynomials, P_j. NP must be less than the number of influence points N used in the topology to give a least squares approximation [67].

To determine the expansion coefficients, α_j, a least squares minimization is performed on the data set of points N giving

$$\sum_{j=1}^{NP} \alpha_j \left[\sum_{k=1}^{N} P_i(x_k) P_j(x_k) \right] = \sum_{k=1}^{N} P_i(x_k) f(x_k) \qquad (27)$$

Expressed in the matrix-vector form we have

$$[C]_{NP,.N}\{\alpha\}_{NP,1} = [P]_{NP,.N}\{f\}_{N,1} \qquad (28)$$

where

$$C_{i,j} = \sum_{k=1}^{N} P_i(x_k) P_j(x_k) \qquad (29)$$

The expansion coefficients can be solved by

$$\{\alpha\}_{NP,1} = [C]^{-1}_{NP,NP}[C]_{NP,N}\{f\}_{NF,1} \quad (30)$$

Recalling the approximation for the field variable $f(x)$ is

$$f(x) = \sum_{j=1}^{NP} \alpha_j P_j \quad (31)$$

Writing in matrix-vector notation, we have

$$f(x) = \{P_c\}^T\{\alpha\} \quad (32)$$

After substitution, the field variable $f(x)$ can be approximated by the expression

$$f_c = \{P_c\}^T_{1,NP}[C]^{-1}_{NP,NP}[C]_{NP,N}\{f\}_{N,1} \quad (33)$$

The derivatives can be approximated by differentiation of the field variable approximation as

$$\frac{\partial f}{\partial x} = \sum_{j=1}^{NP} \alpha_j \frac{\partial P_j}{\partial x} \quad (34)$$

$$\frac{\partial f_c}{\partial x} = \left\{\frac{\partial P_c}{\partial x}\right\}^T_{1,NP} [C]^{-1}_{NP,NP}[C]_{NP,N}\{f\}_{N,1} \quad (35)$$

We can rewrite the expression for any linear differential operator

$$Lf(x) = \{L_{ls}\}^T_{1,N}\{f\}_{N,1} \quad (36)$$

The operator vector $\{L_{ls}\}^T_{1,N} = \{LP_c\}^T_{1,NP}[C]^{-1}_{NP,NP}[C]_{NP,N}$ can be precomputed and stored in a preprocessing stage similar to RBF interpolation.

CHAPTER 4

LOCALIZED BLENDED RADIAL BASIS FUNCTION COLLOCATION METHOD

4.1 Methodology

The local RBF collocation method has shown potential to solve partial differential equations over irregularly spaced data points accurately [13, 12, 19], but the field is usually smooth allowing for the shape parameter to be chosen to give high conditioning providing best accuracy. However, low conditioning allows for steep gradients and discontinuities to be captured using these RBF meshless techniques. Therefore, the RBF interpolation for high and low conditioning must be blended not only to capture the gradients but to keep the solution stable. The local blended RBF scheme is executed by forming two localized expansions using different shape parameter values, one for high conditioning and the other for low conditioning. We begin with the blending expression written as

$$f(x,y) = f_L + \phi(f_H - f_L) \tag{37}$$

alternatively,

$$f(x,y) = (1-\phi)f_L + \phi f_H \tag{38}$$

where f_H may be the high value shape paramter interpolation and f_L is the low value shape parameter interpolation.

$$f_H(x,y) = \sum_{j=1}^{N} \alpha_{H_j} \chi_{H_j}(x,y) + \sum_{j=1}^{NP} \alpha_{j+N} P_j(x) \quad (39)$$

$$f_L(x,y) = \sum_{j=1}^{N} \alpha_{L_j} \chi_{L_j}(x,y) + \sum_{j=1}^{NP} \alpha_{j+N} P_j(x) \quad (40)$$

Defining two RBF with high and low valued shape parameters c_L and c_H. The Hardy Multiquadrics RBF is used as an example

$$\chi_{L_j}(x,y) = \sqrt{(x-x_j)^2 + (y-y_j)^2 + c_L^2} \quad (41)$$

$$\chi_{H_j}(x,y) = \sqrt{(x-x_j)^2 + (y-y_j)^2 + c_H^2} \quad (42)$$

By introducing discrete points at locations x_i and y_i, the functions can be evaluated as

$$f_H(x_i,y_i) = \sum_{j=1}^{N} \alpha_{H_j} \chi_{H_j}(x_i,y_i) + \sum_{j=1}^{NP} \alpha_{H_{j+N}} P_j(x_i,y_i) \quad (43)$$

$$f_L(x_i,y_i) = \sum_{j=1}^{N} \alpha_{L_j} \chi_{L_j}(x_i,y_i) + \sum_{j=1}^{NP} \alpha_{L_{j+N}} P_j(x_i,y_i) \quad (44)$$

Next, the weights $\alpha_{H_j}$and α_{L_j} must be determined. A system of equations can be formed using all the field variable values within the local domain.

$$[\mathrm{C}_H]\{\alpha_H\} = \{f\} \quad (45)$$

$$[C_L]\{\alpha_L\} = \{f\} \quad (46)$$

Inverting the collocation matrices $[\mathrm{C}_L]$ and $[\mathrm{C}_H]$ and solving for the expanison coefficients

$$\{\alpha_H\} = [C_H]^{-1}\{f\} \tag{47}$$

$$\{\alpha_L\} = [C_L]^{-1}\{f\} \tag{48}$$

Substituting and rearranging, we have a blended expression.

$$f(x,y) = (1-\phi)C_L(x,y)^T[C_L]^{-1}\{f\} + \phi\left(C_H(x,y)^T[C_H]^{-1}\{f\}\right) \tag{49}$$

Similarly, the derivatives can simply be approximated by differentiation of the RBF vector

$$\begin{aligned}\frac{\partial f(x,y)}{\partial x} = (1-\phi)&\left\{\frac{\partial C_L(x,y)}{\partial x}\right\}^T [C_L]^{-1}\{f\} \\ &+ \phi\left\{\frac{\partial C_H(x,y)}{\partial x}\right\}^T [C_H]^{-1}\{f\}\end{aligned} \tag{50}$$

$$\begin{aligned}\frac{\partial f(x,y)}{\partial y} = (1-\phi)&\left\{\frac{\partial C_L(x,y)}{\partial y}\right\}^T [C_L]^{-1}\{f\} \\ &+ \phi\left\{\frac{\partial C_H(x,y)}{\partial y}\right\}^T [C_H]^{-1}\{f\}\end{aligned} \tag{51}$$

The term ϕ is the blending term where in this case $\phi = 1$ will provide high shape parameter RBF interpolation and $\phi = 0$ will give low shape parameter interpolation. A method is needed to determine the appropriate way of determining the blending parameter ϕ. Methods using flux limiters or smoothness indicator, such a total variation diminishing (TVD) and weighted essentially non-oscillatory (WENO) schemes are good candidates to start with for numerical experiments.

4.2 Blending Parameter

A method of determining how to blend between high and low shape parameter is needed. Fortunately, the schemes that exist for high-resolution schemes in computational fluid dynamics give a starting point on how this should be performed. In CFD, flux or slope limiters are used to limit the gradients in the solver to realistic values. These flux/slope limiters are functions of the successive gradients found within the solution domain. Many flux limiters have been developed over the years, and more details can be found in these references [70, 71, 72]. For this research, a simple minmod flux limiter is used. The minmod flux limiter can vary linearly between [0,1] depending on the successive gradient r. For example, the flux/slope is calculated by

$$f_{i+\frac{1}{2}} = f_{L_{i+\frac{1}{2}}} + \phi(r_i)(f_{H_{i+\frac{1}{2}}} - f_{L_{i+\frac{1}{2}}}) \quad (52)$$

$$f_{i-\frac{1}{2}} = f_{L_{i-\frac{1}{2}}} + \phi(r_{i-1})(f_{H_{i-\frac{1}{2}}} - f_{L_{i-\frac{1}{2}}}) \quad (53)$$

where the blending parameter is a function of the successive gradient r. The successive gradient is evaluated by the expression.

$$r_i = \frac{u_i - u_{i-1}}{u_{i+1} - u_i} \quad (54)$$

$$r_{i-1} = \frac{u_{i-1} - u_{i-2}}{u_i - u_{i-1}} \quad (55)$$

The local RBF collocation method gives the versatility of interpolating the field variable for the successive gradients. The minmod limiter is used for the blending parameter and is calculated by

$$\phi(r_i) = \max(0, \min(r_i, 1)) \quad (56)$$

$$\phi(r_{i-1}) = \max(0, \min(r_{i-1}, 1)) \quad (57)$$

The blending parameter in regions where high gradients, shocks or discontinuities are present the minmod limiter reduces a value less between [0,1] and if the value of the successive gradient is less than zero the minmod limiter is equal to zero switching to low shape parameter interpolation.

Another approach can be taken that involves using the smoothness indicator developed for the weighted essentially non-oscillatory or WENO scheme [65]. The WENO scheme uses polynomials to interpolate the fluxes at the midpoints of the stencil. Many stencils are used in the formulation to provide high order accuracy, and the smoothness indicator determines the stencil that provides the smoothest flux approximation by comparing the L^2 norm values of the derivatives of the polynomial used. This flux is then used to approximate the derivatives in the governing equations. An approach for the case of RBF interpolation is performed by replacing the method of using different stencils by using different shape parameter RBF collocations and determine the smoothest interpolation. For example, the smoothness indicator is first determined by computing the first, second, and third derivative for the RBF collocation for each shape parameter used.

$$\frac{\partial f_H}{\partial x} = \left\{\frac{\partial \mathrm{C}_H}{\partial x}\right\}^T [C_H]^{-1}\{f\} \quad (58)$$

$$\frac{\partial^2 f_H}{\partial x^2} = \left\{\frac{\partial^2 C_H}{\partial x^2}\right\}^T [C_H]^{-1}\{f\} \quad (59)$$

$$\frac{\partial^3 f_H}{\partial x^3} = \left\{\frac{\partial C_H}{\partial x}\right\}^T [C_H]^{-1}\{f\} \quad (60)$$

and

$$\frac{\partial f_L}{\partial x} = \left\{\frac{\partial C_L}{\partial x}\right\}^T [C_L]^{-1}\{f\} \quad (61)$$

$$\frac{\partial^2 f_L}{\partial x^2} = \left\{\frac{\partial^2 C_L}{\partial x^2}\right\}^T [C_L]^{-1}\{f\} \quad (62)$$

$$\frac{\partial^3 f_L}{\partial x^3} = \left\{\frac{\partial^3 C_L}{\partial x^3}\right\}^T [C_L]^{-1}\{f\} \quad (63)$$

Many derivatives can be approximated and used in the smoothness formulation because of the infinitely smooth property of the multiquadric RBF. With the derivatives known, we can attempt to approximate the smoothness indicator for each RBF approximation.

$$\beta_H = \sqrt{\Delta x^2 \left(\frac{\partial f_H}{\partial x}\right)^2 + \Delta x^4 \left(\frac{\partial^2 f_H}{\partial x^2}\right)^2 + \Delta x^6 \left(\frac{\partial^3 f_H}{\partial x^3}\right)^2} \quad (64)$$

$$\beta_L = \sqrt{\Delta x^2 \left(\frac{\partial f_L}{\partial x}\right)^2 + \Delta x^4 \left(\frac{\partial^2 f_L}{\partial x^2}\right)^2 + \Delta x^6 \left(\frac{\partial^3 f_L}{\partial x^3}\right)^2} \quad (65)$$

The weighting factors can be calculated by

$$\omega_H = \frac{\gamma_H}{\beta_H + \varepsilon} \quad (66)$$

$$\omega_L = \frac{\gamma_L}{\beta_L + \varepsilon} \quad (67)$$

where γ_H and γ_L are linear weights and in this case are chosen by the user. The linear weights are chosen so that the high shape parameter interpolation is favoured, i.e. $\gamma_H = 0.9$ and $\gamma_L = 0.1$. The parameter ε is chosen to be small number to avoid zero in the denominator, such as $\varepsilon = 1e^{-8}$. Finally, the weights can be determined by the ratios

$$\phi_H = \frac{\omega_H}{\omega_H + \omega_L} \quad (68)$$

$$\phi_L = \frac{\omega_L}{\omega_H + \omega_L} \quad (69)$$

The flux is then calculated by

$$f_{i+\frac{1}{2}} = \phi_L f_{L_{i+\frac{1}{2}}} + \phi_H f_{H_{i+\frac{1}{2}}} \quad (70)$$

The WENO scheme approach still needs to be studied, but initial results using the WENO blending approach looks promising. The flux limiter approach is used heavily for this research and is recommended to be used as a starting point for future research using the blended RBF approach.

CHAPTER 5

INVISCID FLUX SCHEMES

5.1 Steger Warming Flux Splitting

The Steger-Warming flux splitting scheme [57, 73, 74, 75, 76] is an upwinding scheme which splits the convective flux terms into right and left running components. The value of the eigenvalues determines the direction of the fluxes. The formulation is explained well in Hirsch [77], but summarized is this section.

For one-dimensional inviscid flow, the governing equation can be written as

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} = 0 \qquad (71)$$

where Q is a vector of the primitive variables and F is a vector of the fluxes.

$$Q = \begin{vmatrix} \rho \\ \rho u \\ \rho e_t \end{vmatrix} \quad F = \begin{vmatrix} \rho u \\ \rho u^2 + P \\ (\rho e_t + P)u \end{vmatrix}$$

The total energy is calculated by $e_t = e + \frac{1}{2}u^2$ and the ideal gas assumption is used as the equation of state given as $P = \rho RT$

Applying the flux splitting scheme, the governing equation can be rewritten as

$$\frac{\partial Q}{\partial t} + \frac{\partial F^+}{\partial x} + \frac{\partial F^-}{\partial x} = 0 \qquad (72)$$

The flux components can be calculated by

$$F^{\pm} = \frac{\rho}{2\gamma}\begin{Bmatrix} 2(\gamma-1)\lambda_1^{\pm} + \lambda_2^{\pm} + \lambda_3^{\pm} \\ 2(\gamma-1)\lambda_1^{\pm}u + \lambda_2^{\pm}(u+a) + \lambda_3^{\pm}(u-c) \\ (\gamma-1)\lambda_1^{\pm}u^2 + \frac{\lambda_2^{\pm}}{2}(u+a)^2 + \frac{\lambda_3^{\pm}}{2}(u-a)^2 + \frac{3-\gamma}{2(\gamma-1)}(\lambda_2^{\pm} + \lambda_3^{\pm})a^2 \end{Bmatrix} \quad (73)$$

where for supersonic flow the eigenvalues are

$$\lambda_1^+ = u,\ \lambda_2^+ = (u+a),\ \lambda_3^+ = (u-a)$$

$$\lambda_1^- = \lambda_2^- = \lambda_3^- = 0$$

For subsonic flow, the eigenvalues are

$$\lambda_1^+ = u,\ \lambda_2^+ = (u+a),\ \lambda_3^+ = 0$$

$$\lambda_1^- = \lambda_2^- = 0,\ \lambda_3^- = (u-a)$$

For two-dimensional flow, we can write the split flux governing equation as

$$\frac{\partial Q}{\partial t} + \frac{\partial F^+}{\partial x} + \frac{\partial F^-}{\partial x} + \frac{\partial G^+}{\partial y} + \frac{\partial G^-}{\partial y} = 0 \quad (74)$$

The split flux can then be written [77]

$$F^+ = \begin{bmatrix} \alpha \\ \alpha u + a(\lambda_2^+ - \lambda_3^+) \\ \alpha v \\ \alpha\frac{u^2+v^2}{2} + ua(\lambda_2^+ - \lambda_3^+) + a^2\frac{\lambda_2^+ + \lambda_3^+}{\gamma-1} \end{bmatrix} \quad (75)$$

$$F^- = \begin{bmatrix} \alpha \\ \alpha u + a(\lambda_2^- - \lambda_3^-) \\ \alpha v \\ \alpha\frac{u^2+v^2}{2} + \mathrm{u}a(\lambda_2^- - \lambda_3^-) + a^2\frac{\lambda_2^- + \lambda_3^-}{\gamma-1} \end{bmatrix} \quad (76)$$

where

$$\alpha = 2(\gamma - 1)\lambda_1 + \lambda_2 + \lambda_3$$

For supersonic flow, the eigenvalues for the split fluxes are all positive. Therefore

$$\lambda_1^+ = u, \lambda_2^+ = u + a, \lambda_3^+ = u - a$$

$$\lambda_1^- = \lambda_2^- = \lambda_3^- = 0$$

The eigenvalue $\lambda_3 = u - a$ is negative. For subsonic flow, we have one negative component.

$$\lambda_1^+ = u, \lambda_2^+ = u + a, \lambda_3^+ = 0$$

$$\lambda_1^- = \lambda_2^- = 0,\ \lambda_3^- = u - a$$

The y-component of the fluxes can be calculated similarly by [77]

$$G^+ = \begin{bmatrix} \alpha \\ \alpha u \\ \alpha v + a(\lambda_2^+ - \lambda_3^+) \\ \alpha \dfrac{u^2 + v^2}{2} + va(\lambda_2^+ - \lambda_3^+) + a^2 \dfrac{\lambda_2^+ + \lambda_3^+}{\gamma - 1} \end{bmatrix} \quad (77)$$

$$G^- = \begin{bmatrix} \alpha \\ \alpha u \\ \alpha v + a(\lambda_2^- - \lambda_3^-) \\ \alpha \dfrac{u^2 + v^2}{2} + va(\lambda_2^- - \lambda_3^-) + a^2 \dfrac{\lambda_2^- + \lambda_3^-}{\gamma - 1} \end{bmatrix} \quad (78)$$

where

$$\alpha = 2(\gamma - 1){\lambda_1}^{\pm} + {\lambda_2}^{\pm} + {\lambda_3}^{\pm}$$

For supersonic flow, the eigenvalues for the split fluxes are positive. Therefore,

$$\lambda_1^+ = v, \lambda_2^+ = v + a, \lambda_3^+ = v - a$$

$$\lambda_1^- = \lambda_2^- = \lambda_3^- = 0$$

For subsonic flow, the eigenvalues for the split fluxes are

$$\lambda_1^+ = v, \lambda_2^+ = v + a, \lambda_3^+ = 0$$

$$\lambda_1^- = \lambda_2^- = 0,\ \lambda_3^- = v - a$$

The governing equation can be solved since the flux components are known. An explicit forward differencing in time is used giving the scheme below

$$Q^{n+1} = Q^n - \Delta t \left[\frac{\partial F^+}{\partial x} + \frac{\partial F^-}{\partial x} + \frac{\partial G^+}{\partial y} + \frac{\partial G^-}{\partial y} \right] \quad (79)$$

5.2 Van Leer Flux Vector Splitting Scheme

The Steger-Warming flux splitting methods can give inaccurate results at sonic and stagnation points. These errors are due to the Jacobian of the split fluxes, i.e. $\frac{\partial f^+}{\partial u}$ and $\frac{\partial f^-}{\partial u}$ for 1-D flux splitting, are not continuously differentiable. The Van Leer flux splitting scheme [58, 73, 74, 75] seeks to correct the discontinuity at sonic and stagnation points by letting the Jacobian be continuous functions of Mach number and the function be lowest order polynomial possible [77]. This is an important concept because it forms a basis used in other schemes, such as the AUSM schemes.

Applying the flux splitting scheme, the governing equations can be rewritten in the form

$$\frac{\partial Q}{\partial t} + \frac{\partial F^+}{\partial x} + \frac{\partial F^-}{\partial x} = 0 \quad (80)$$

The flux components can be calculated by [77]

$$f_{VL}^{+} = \frac{\rho a}{4}(M+1)^2 \begin{Bmatrix} 1 \\ \frac{2a}{\gamma}\left(1+\frac{\gamma-1}{2}M\right) \\ \frac{2a^2}{\gamma^2-1}\left(1+\frac{\gamma-1}{2}M\right)^2 \end{Bmatrix} \quad (81)$$

$$f_{VL}^{-} = \frac{\rho a}{4}(M+1)^2 \begin{Bmatrix} 1 \\ \frac{2a}{\gamma}\left(-1+\frac{\gamma-1}{2}M\right) \\ \frac{2a^2}{\gamma^2-1}\left(1-\frac{\gamma-1}{2}M\right)^2 \end{Bmatrix} \quad (82)$$

For two-dimensional flow, the governing equations can be written as

$$\frac{\partial Q}{\partial t} + \frac{\partial F^{+}}{\partial x} + \frac{\partial F^{-}}{\partial x} + \frac{\partial G^{+}}{\partial y} + \frac{\partial G^{-}}{\partial y} = 0 \quad (83)$$

The flux splitting for the Van Leer scheme [77]

$$f_{VL}^{\pm} = \pm\frac{\rho}{4a}(u \pm c)^2 \begin{Bmatrix} 1 \\ \frac{(\gamma-1)u \pm 2a}{\gamma} \\ \frac{v^2}{2} + \frac{[(\gamma-1)u \pm 2a]^2}{2(\gamma^2-1)} \end{Bmatrix} \quad (84)$$

$$g_{VL}^{\pm} = \pm\frac{\rho}{4a}(v \pm c)^2 \begin{Bmatrix} 1 \\ u \\ \frac{(\gamma-1)v \pm 2a}{\gamma} \\ \frac{u^2}{2} + \frac{[(\gamma-1)v \pm 2a]^2}{2(\gamma^2-1)} \end{Bmatrix} \quad (85)$$

5.3 Advection Upstream Splitting Method (AUSM)

The AUSM scheme begins by noticing the convection and acoustic waves are physically separate processes, and therefore the flux can be written by splitting the pressure from the inviscid flux [78, 59].

$$F = F_c + F_p \tag{86}$$

where

$$F_c = \begin{pmatrix} \rho u \\ \rho u^2 \\ \rho(e_t + P)u \end{pmatrix} \quad F_p = \begin{pmatrix} 0 \\ P \\ 0 \end{pmatrix} \tag{87}$$

In two dimensions, the inviscid flux is written similarly as

$$F_c = \begin{pmatrix} \rho u \\ \rho u^2 \\ \rho uv \\ \rho(et + P)u \end{pmatrix} \quad F_p = \begin{pmatrix} 0 \\ P \\ 0 \\ 0 \end{pmatrix} \tag{88}$$

$$G_c = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 \\ \rho(et + P)v \end{pmatrix} \quad G_p = \begin{pmatrix} 0 \\ 0 \\ P \\ 0 \end{pmatrix} \tag{89}$$

The convective flux is expressed regarding Mach number

$$F_c = Ma \begin{pmatrix} \rho \\ \rho u \\ \rho(e_t + P) \end{pmatrix} \quad F_p = \begin{pmatrix} 0 \\ P \\ 0 \end{pmatrix} \tag{90}$$

or

$$F_c = Ma \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho(et + P) \end{pmatrix} \quad F_p = \begin{pmatrix} 0 \\ P \\ 0 \\ 0 \end{pmatrix} \tag{91}$$

$$G_c = Ma\begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho(et+P) \end{pmatrix} \quad G_p = \begin{pmatrix} 0 \\ 0 \\ P \\ 0 \end{pmatrix} \tag{92}$$

We can then write the flux at the face $i+\frac{1}{2}$

$$f_{i+\frac{1}{2}} = f_{c_{i+\frac{1}{2}}} + \boldsymbol{P}_{i+\frac{1}{2}} \tag{93}$$

$$f_{C_{i+\frac{1}{2}}} = m_{i+\frac{1}{2}} \Phi_{i+\frac{1}{2}} \tag{94}$$

$$\boldsymbol{P}_{i+\frac{1}{2}} = \begin{pmatrix} 0 \\ p_{i+\frac{1}{2}} \\ 0 \end{pmatrix} \tag{95}$$

where

$$m_{i+\frac{1}{2}} = \mathcal{M}_i^+ + \mathcal{M}_{i+1}^-, \quad \mathcal{M}_i^\pm = \mathcal{M}^\pm(M_i) \tag{96}$$

$$p_{i+\frac{1}{2}} = \wp_j^+ p_j + \wp_{j+1}^- p_{j+1}, \quad \wp_i^\pm = \wp^\pm(M_i) \tag{97}$$

And

$$\mathcal{M}_i^\pm = \begin{cases} \frac{1}{2}(M \pm |M|), & if\ |M| > 1 \\ \pm\frac{1}{4}(M \pm 1)^2, & otherwise \end{cases} \tag{98}$$

And

$$\wp_i^\pm = \begin{cases} \frac{1}{2}(M \pm |M|)/M, & if\ |M| > 1 \\ \frac{1}{4}(M \pm 1)^2(2 \mp M), & otherwise \end{cases} \tag{99}$$

Finally, upwinding is applied to $\Phi_{i+\frac{1}{2}}$

$$\Phi_{i+\frac{1}{2}} = \begin{cases} \Phi_i, & if\ m_{j+\frac{1}{2}} > 1 \\ \Phi_{i+1}, & otherwise \end{cases} \quad (100)$$

$$\Phi = \begin{pmatrix} \rho a \\ \rho u a \\ \rho(et + p)a \end{pmatrix} \quad (101)$$

Therefore

$$f_{i+\frac{1}{2}} = m_{i+\frac{1}{2}} \Phi_{i+\frac{1}{2}} + \boldsymbol{P}_{i+\frac{1}{2}} \quad (102)$$

5.4 AUSM$^+$

The AUSM+ scheme is an update to the AUSM scheme to fix issues near stagnation points. The AUSM scheme discussed in the previous section is a special case of the AUSM+ scheme by allowing the speed of sound to be upwinded with the fluxes as presented before [59]. The AUSM+ scheme works by defining a common speed of sound denoted by $a_{i+\frac{1}{2}}$.

The flux can then be determined by the expression

$$f_{c_{i+\frac{1}{2}}} = m_{i+\frac{1}{2}}\, a_{i+\frac{1}{2}} \Phi_{i+\frac{1}{2}} \quad (103)$$

The critical speed of sound then calculates the common speed of sound a^*which can be expressed in terms on the total enthalpy h_t, specific heat ratio γ, and u. For an ideal gas, we have

$$h_t = \frac{a^2}{\gamma - 1} + \frac{1}{2}u^2 = \frac{(\gamma + 1)a^{*2}}{2(\gamma - 1)} \quad (104)$$

and

$$a_{i+\frac{1}{2}} = \frac{a^{*^2}}{u_j} \tag{105}$$

To account for the flow is subsonic, the following approach is taken

$$a_{i+\frac{1}{2}} = \min(\tilde{a}_L, \tilde{a}_R) \tag{106}$$

Where

$$\tilde{a} = \frac{a^{*^2}}{\max(a^*, |u|)} \tag{107}$$

The equations used to blend the fluxes and pressure based on Mach number are

$$\mathcal{M}_i^{\pm} = \begin{cases} \frac{1}{2}(M \pm |M|), & if\ |M| > 1 \\ \pm\frac{1}{2}(M \pm 1)^2 \pm \beta(M^2 - 1)^2, & otherwise \end{cases} \tag{108}$$

where $-\frac{1}{16} \leq \beta \leq \frac{1}{2}$

$$\wp_i^{\pm} = \begin{cases} \frac{1}{2}(M \pm |M|)/M, & if\ |M| > 1 \\ \frac{1}{4}(M \pm 1)^2(2 \mp M) \pm \alpha M(M^2 - 1)^2, & otherwise \end{cases} \tag{109}$$

where $-\frac{3}{4} \leq \alpha \leq \frac{3}{16}$

The recommended values for α and β are

$$\beta = \frac{1}{8} \quad \alpha = \frac{3}{16} \tag{110}$$

5.5 Inviscid Flux Scheme Code Validation

The flux schemes used can be complicated to implement and must be well understood before adding the blended RBF interpolation. A finite difference solver is developed for two-dimensional compressible flow to ensure the upwind schemes are executed correctly. The geometry is a two-dimensional channel with a 10° wedge to induce an oblique shock wave and is shown in Figure 6 along with the boundary conditions [73, 74, 79, 80]. A Mach 2 flow is prescribed at the inlet, and a slip boundary condition is used at the wall. Extrapolation from the interior of the domain is used for the outlet. The freestream inlet conditions are used as an initial condition set throughout the domain. Since we are seeking a steady state solution, we implement local time stepping to accelerate convergence. First order finite differencing is used for the Steger-Warming and Van Leer flux splitting for each flux component.

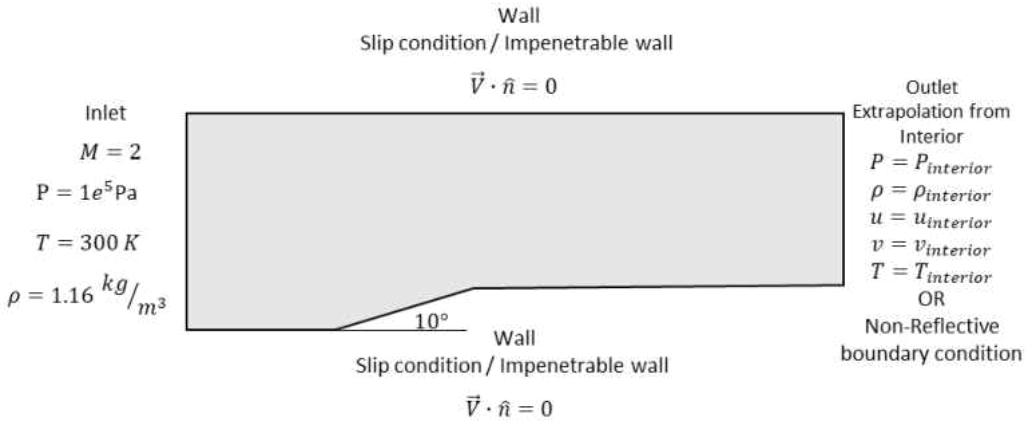


Figure 6: Boundary Conditions for FDM Validation

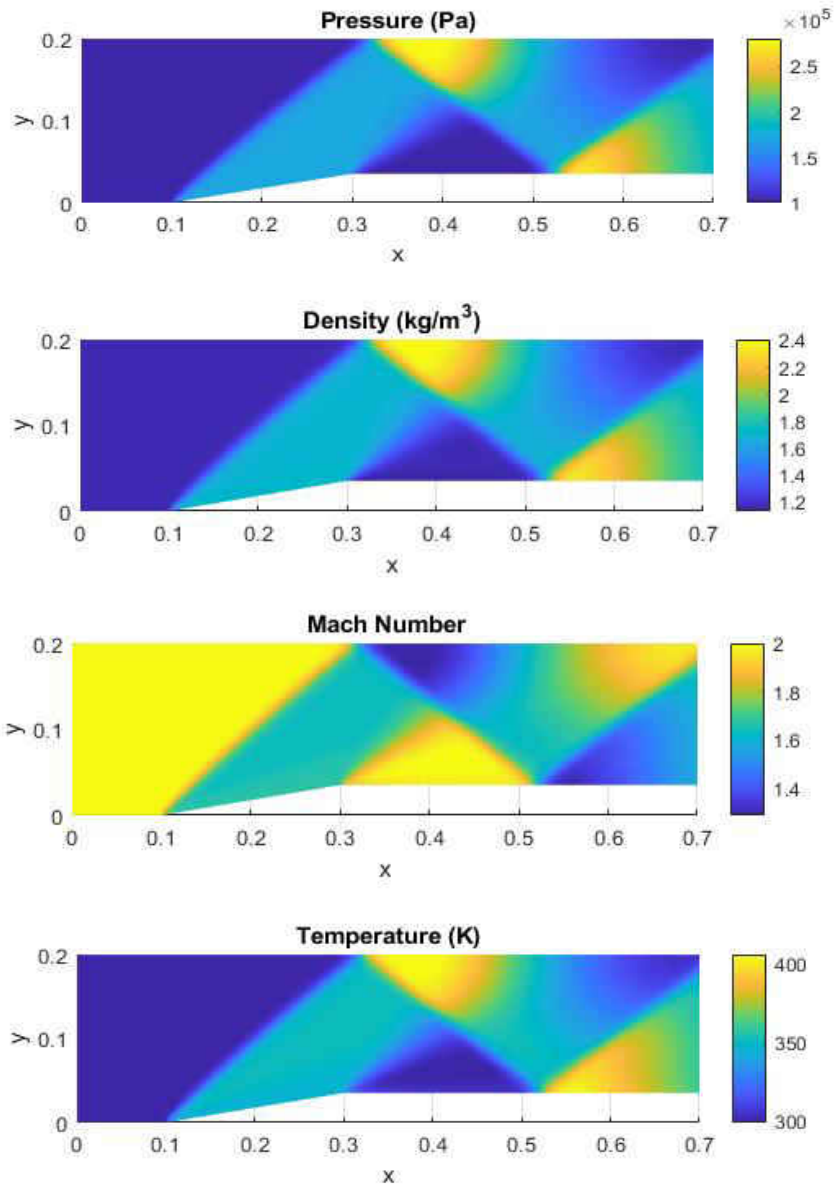


Figure 7: Steger-Warming using Finite Difference Method

The results from the finite difference scheme are in good agreement with the oblique shock relations. The analytical solutions are listed in Table 2.

Table 2: Solutions using Oblique Shock Relations

Location	Mach	Pressure (Pa)	Density (kg/m^3)	Temperature (K)
1	2.00	1.000E+05	1.16	300.00
2	1.64	1.707E+05	1.69	351.05
3	1.29	2.803E+05	2.41	405.96

The code using the finite difference method and the inviscid flux schemes has been validated. The next goal is to use these inviscid flux schemes with the localized RBF collocation method. Implementing the finite difference method using these schemes is beneficial because all these inviscid schemes can be coded and saved in subroutines or functions. These functions are used later in the implementation of the localized RBF collocation method.

CHAPTER 6

NUMERICAL EXPERIMENTS

6.1 Shape Parameter Effects on RBF Interpolation

Radial basis functions, like the Hardy multiquadric, are dependent on the shape parameter, c, which has a significant effect on the behavior of the approximation of the function and its derivatives. The shape parameter is typically chosen to be a large value so that the matrix $[\mathrm{C}]$ is close to ill-conditioning [27]. In the following sections, it is shown this methodology fails for problems with steep gradients and shocks.

Figure 8, illustrates the use of a high-value shape parameter when interpolating smooth functions. An RBF reconstruction of the test function $f(x) = -\arctan[a(x - 0.5)]$ using a shape parameter of $c = 0.001$ providing a low condition number for the interpolation matrix is presented in the figure. The test function has the parameter a to easily change the slope of the function giving the capability of testing different slopes quickly. For $a = 5$, the function is smooth, and the interpolation tends to approximate the function as piecewise linear giving relatively high L^2 norm errors. Increasing the shape parameter to $c = 1$, the RBF reconstruction matches the test function well providing significantly smaller L^2 errors. Ideally, the RBF interpolation matrix needs to have a high condition number to benefit from the spectral accuracy capabilities.

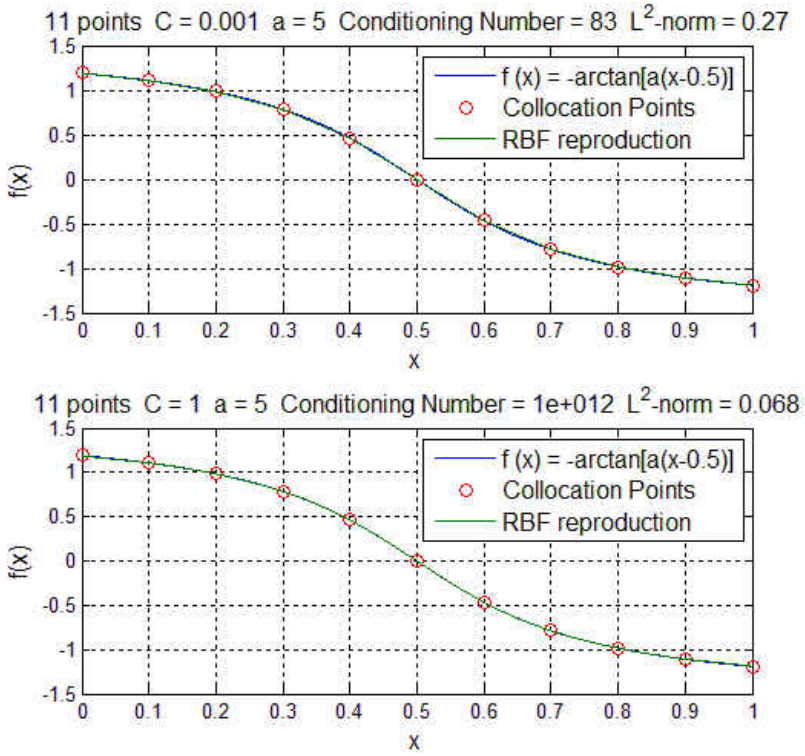


Figure 8: Interpolation of smooth function using c = 1 and c = 0.001 for the test function f(x) = -arctan[a(x-0.5)]. Test Function and RBF Reproduction (Top). Zoomed in on the plot (Bottom).

Next, the advantages of using low condition RBF interpolation are demonstrated. In Figure 9, a step function is used as the next test function, and the step is introduced at $x = 0.5$. Using a shape parameter $c = 0.001$, the RBF reconstruction approximates the test function as piecewise linear. With the addition of collocation points near the step, the error in the approximation can be reduced. Although the low shape parameter RBF interpolation gives a first order approximation, the use of low shape parameter interpolation does not introduce oscillations. Increasing the shape parameter to $c = 1$ causes the RBF reconstruction to be oscillatory. This unstable behavior is common for

high order schemes used to solving hyperbolic partial differential equations. These oscillations will ultimately cause a partial differential equation to produce unphysical values eventually giving an unstable solution.

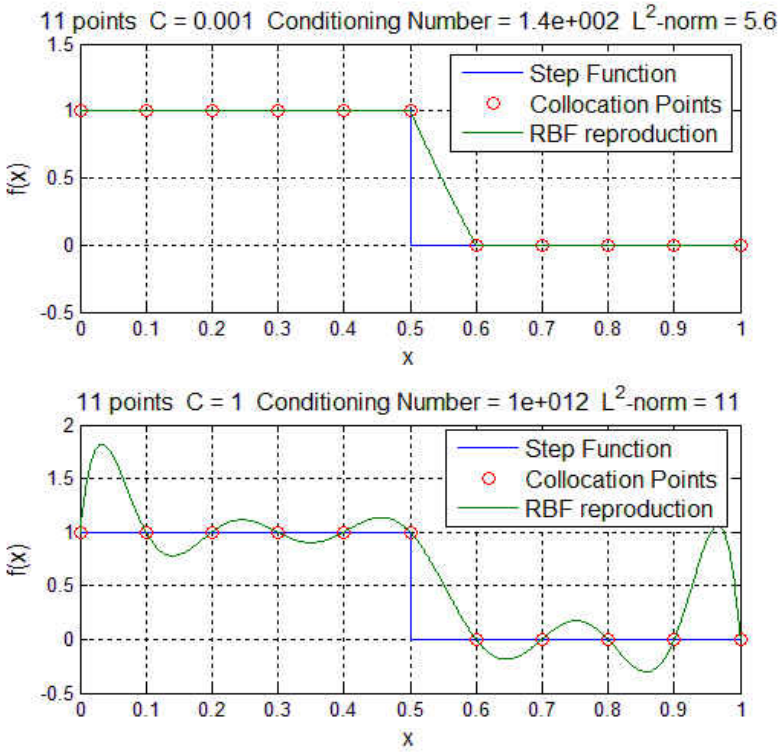


Figure 9: Interpolation of step function using c=1 and c=0.001

RBF interpolation is further investigated by comparing the RBF interpolation approximated derivatives for different shape parameter values. Figure 10 compares the test function $f(x) = Atan^{-1}[\omega(x - x_0)]$ and the derivatives for shape parameters $c = 0.1,\ 0.5,\ 1.0$ for smooth function reconstruction.

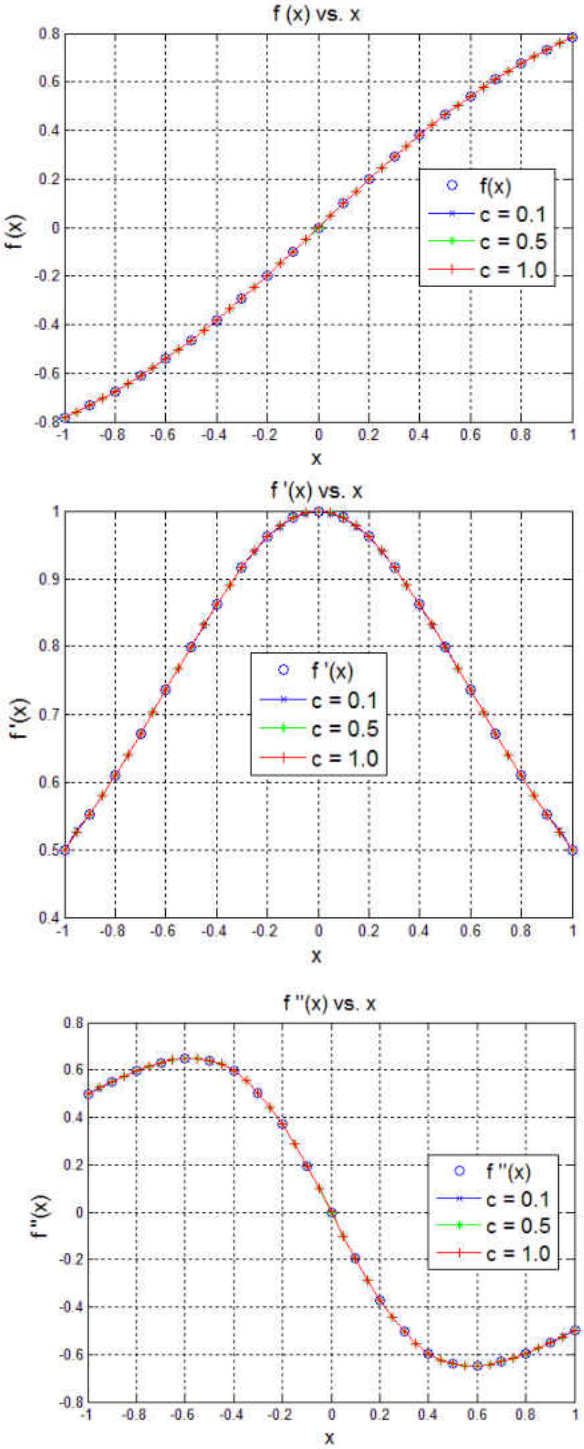


Figure 10: Test function $f(x) = Atan^{-1}[\omega(x - x_0)]$ $A = 1\ \omega = 1$

Smooth function interpolation for $f(x), f'(x), f''(x)$

We observe that the RBF interpolation works well for reproducing the function and derivatives for all shape parameter used. However, the RBF interpolation using a higher shape parameter produces significantly lower errors. The errors for smooth function are in Table 3

Table 3: L^2 Norms for Smooth Function RBF Interpolation

c	$f(x)$	$f'(x)$	$f''(x)$
0.1	$2.52e^{-4}$	0.0016	$3.46e^{-4}$
0.5	$3.20e^{-6}$	$1.64e^{-5}$	$5.71e^{-6}$
1.0	$1.41e^{-8}$	$4.41e^{-8}$	$1.45e^{-8}$

We can increase the slope of the test function by the parameter ω. For $\omega = 10$, the RBF rendering produces oscillation for the test function and derivatives using a shape parameter value of $c = 1$. Using a low shape RBF interpolation, the oscillations are minimum and produces relatively low error compared to high shape parameter results. The errors for the steep gradient case of $\omega = 10$ are shown in Table 4.

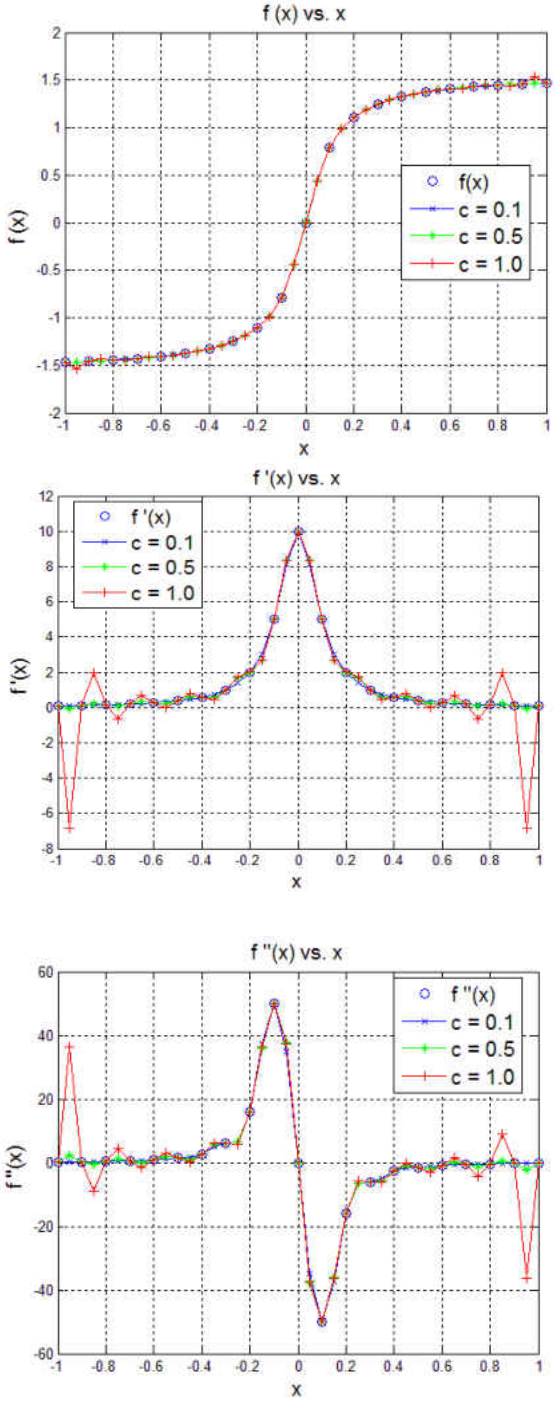


Figure 11: Test function $f(x) = Atan^{-1}[\omega(x - x_0)]$ $A = 1\ \omega = 10$

Smooth function interpolation for $f(x), f'(x), f''(x)$

Table 4: L^2 Norms for function with Steep Gradient RBF Interpolation

c	$f(x)$	$f'(x)$	$f''(x)$
0.1	0.0021	0.0069	0.0526
0.5	0.0017	0.0167	0.0480
1.0	0.0060	0.2369	0.0810

Continuing with the study of shape parameter effects on RBF interpolation, we focus our attention on reproducing and determining the derivatives for a two-dimensional surface. The surface used for the numerical experiment is shown in Figure 12.

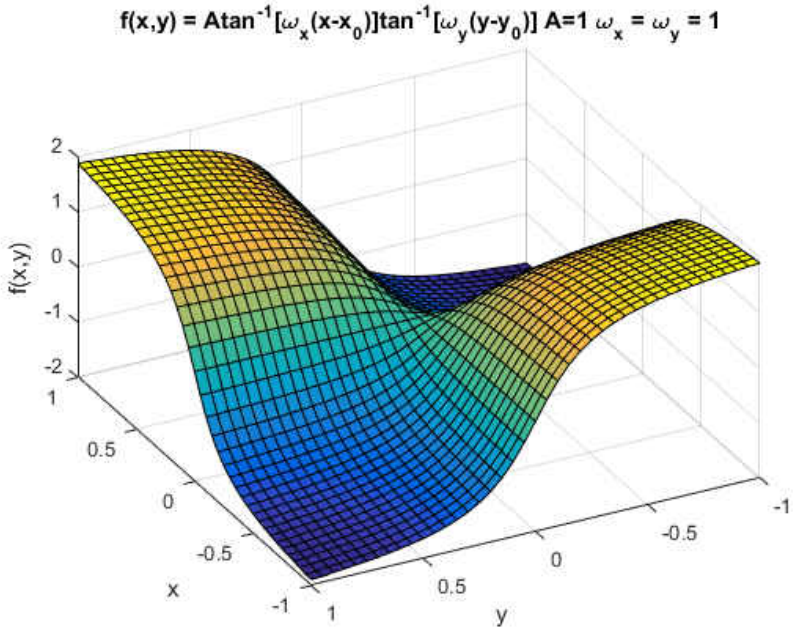


Figure 12: Two-Dimensional Test using RBF Interpolation

Again, a low shape parameter RBF in provides better approximations for the test function reproduction and approximating derivatives. Errors for the function reproduction and the partial derivatives for the two-dimensional case are shown in Table 5.

Table 5: L^2 Norms for 2-D function with Steep Gradient RBF Interpolation

c	$f(x,y)$	$f_x(x,y)$	$f_y(x,y)$	$\nabla^2 f(x,y)$
0.1	0.0051	0.0522	0.0522	0.0964
0.5	0.0053	0.0436	0.0436	0.1363
1.0	0.0098	0.0437	0.0437	0.1763

6.2 One-Dimensional Inviscid Burgers Equation

The solution of the 1-D inviscid Burgers equation is computed using the blended RBF approach and RBF enhanced finite difference. This numerical experiment demonstrates the concept of blended RBF interpolation applied to the solution of a PDE with a shock or discontinuity. The governing equation and the initial condition are

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \qquad (111)$$

$$u(x,0) = 1 \; for \; x < 0.5$$

$$u(x,0) = 0 \; for \; x \geq 0.5$$

The collocation points are randomly distributed across the domain of [0,1] over x as shown in Figure 13. The local collocation points are specified to be the points $i+1$, i, and $i-1$ where i is the data center, but the number of collocation points can be increased for better approximations. These local collocation points are used for the RBF interpolation along the subdomain $[x_{i-1}, x_{i+1}]$. A distance Δx is used to interpolate for the value of u^n at $x_i - \Delta x$ to create an upwind stencil. With the upstream values of u^n known, a first order explicit scheme in time, t and first order finite difference in x is used at some constant distance Δx to calculate the next value of u at the next time step, u^{n+1}.

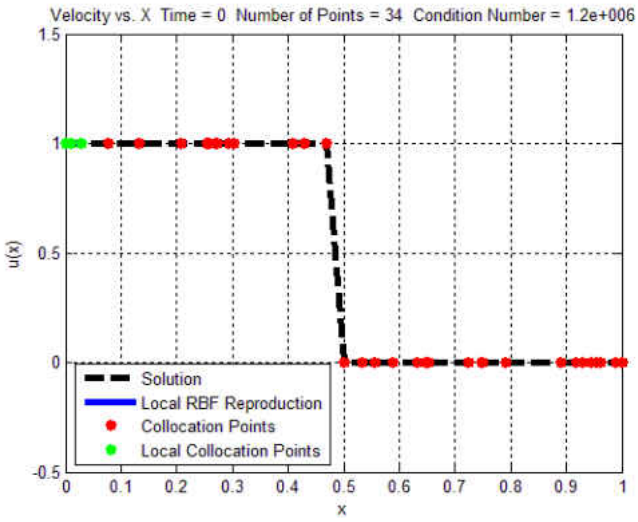


Figure 13: Initial Condition for the Non-linear Burgers' Equation Solution

A high shape parameter value rendering the RBF flat is used to demonstrate the instabilities produced by high condition number interpolation near shocks and discontinuities. Condition numbers range from $K = 2e^7 - 2e^{10}$ as the scheme proceeds through the algorithm. A local RBF reproduction is evaluated for each data center and

within the collocation point limits. As the solution marches in along space and time, the high shape parameter RBF reproduction causes overshoots near steep gradients. This behavior will eventually lead to instabilities like other high order methods seen in computational fluid dynamics. This is shown in Figure 14.

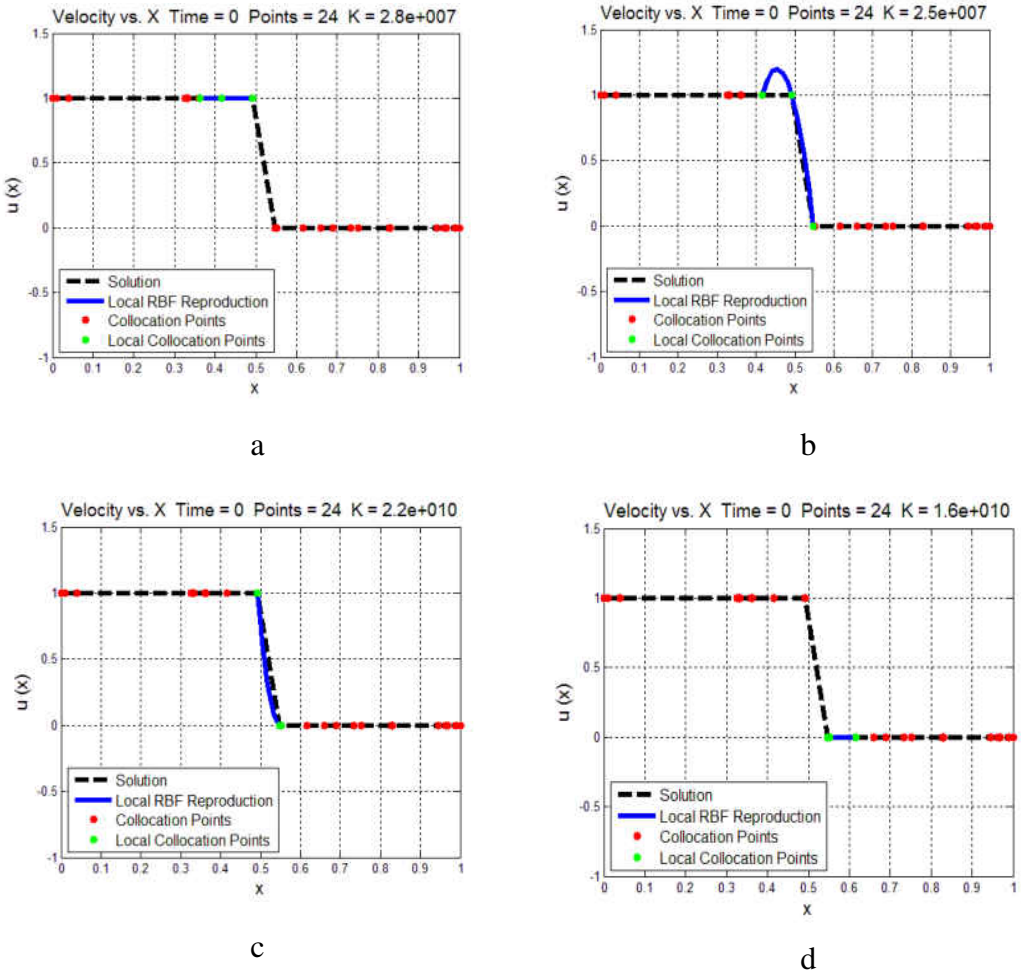


Figure 14: Solution sweep with constant shape parameter interpolation approach a) Calculation before discontinuity K = 2.8 x 10^7 b) At top of discontinuity K = 2.5 x 10^7 c) At bottom of discontinuity K = 2.2 x 10^{10} d) After discontinuity K = 1.6 x 10^{10}

Next, the local blended RBF collocation method is used allowing the scheme to switch between high or low shape parameter RBF interpolation vectors. In Figure 15, it

is shown that the condition number of the collocation matrix is high near smooth regions. As the scheme marches towards the discontinuity, the scheme switches to a low shape parameter RBF and low condition number. For Hardy multiquadric RBF, a low shape parameter causes the interpolation to be piecewise linear providing stability near steep gradients and shocks.

In Figure 15, the condition number varies between $K = 6.4 - 6e^6$ depending on the smoothness of the function. Early in the development of the blending scheme, the second derivative was monitored and if the curvature of the function was found to be too high based some criterion, the scheme can switch to low RBF interpolation. This strategy worked well, but a robust blending criterion based on the second derivative of the field proved to be difficult to determine. A better approach is adopted to use the successive gradients and flux limiter. This strategy gave a robust criterion for blending high and low shape parameter RBF.

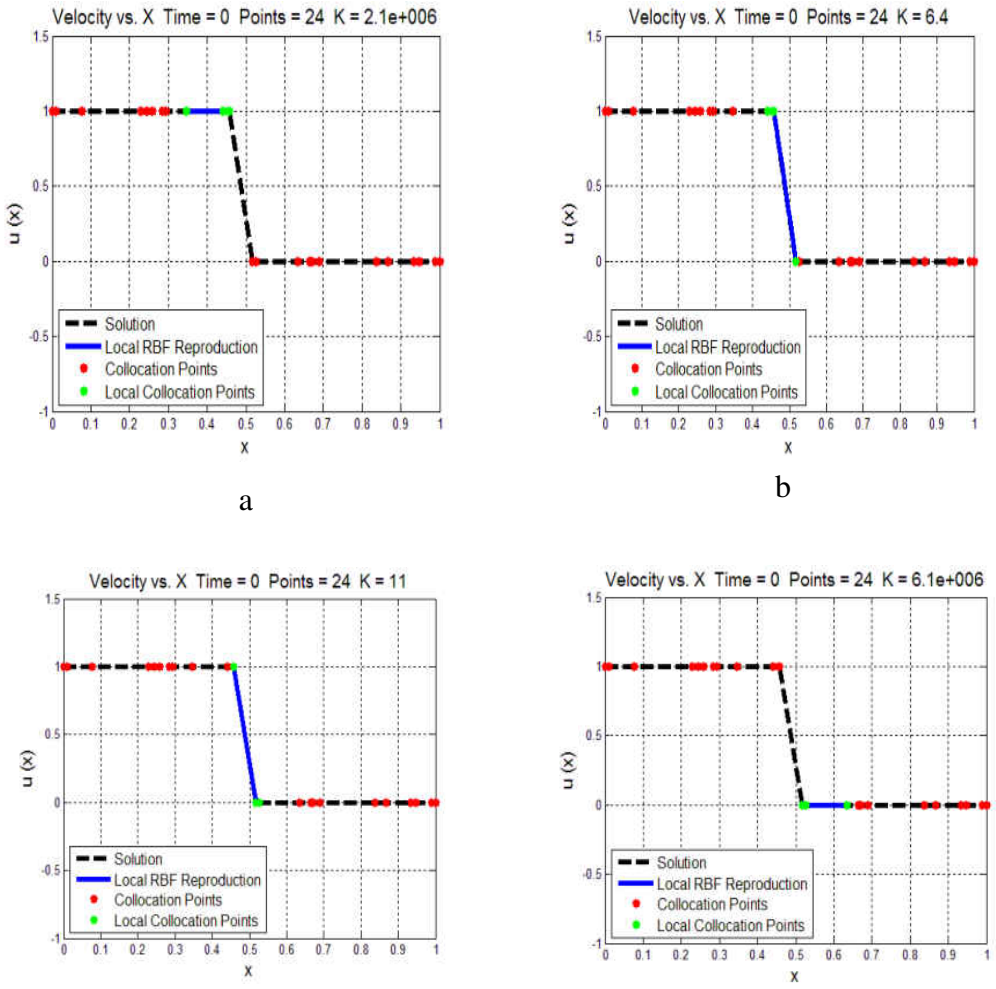


Figure 15 Solution sweep with blended interpolation approach a) Calculation before discontinuity K = 2.1 x 10^6 b) At top of discontinuity K = 6.4 c) At bottom of discontinuity K = 11 d) After discontinuity K = 6.1 x 10^6

The next figures further demonstrate the stability of the blended RBF scheme as the solution progresses through time. As the wave travels from left to right, the wave maintains its initial shape. The scheme avoids introducing any oscillations into the solution maintaining a stable solution. The solution of the Burgers' equation is presented in Figure 16 and Figure 17.

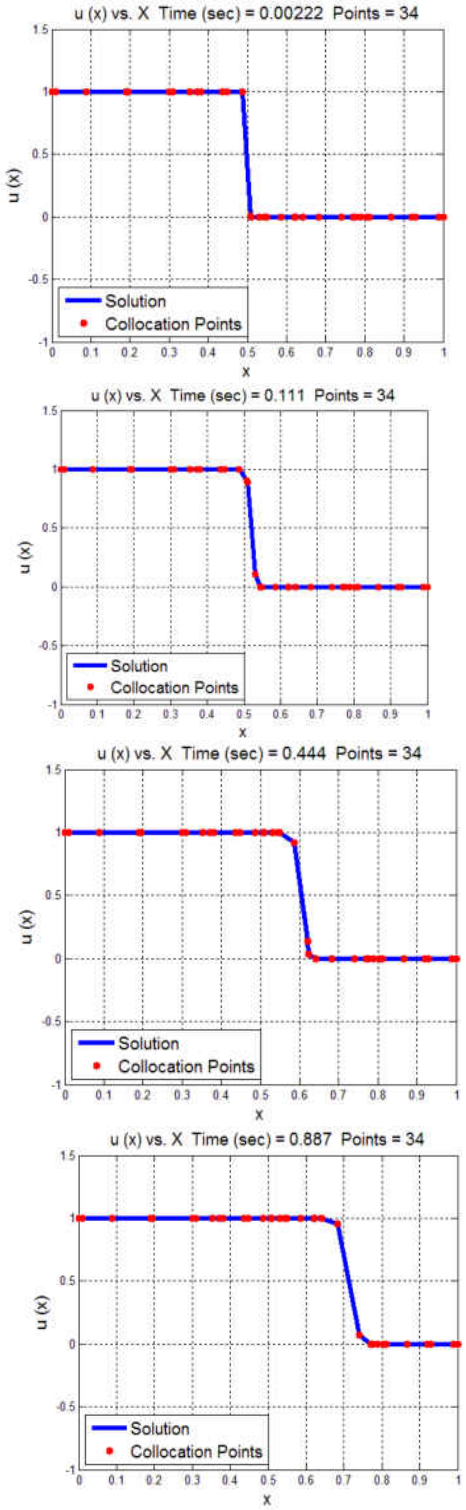


Figure 16 Non-linear Burgers' Equation Solution using the RBF blended interpolation with the RBF enhanced finite difference method

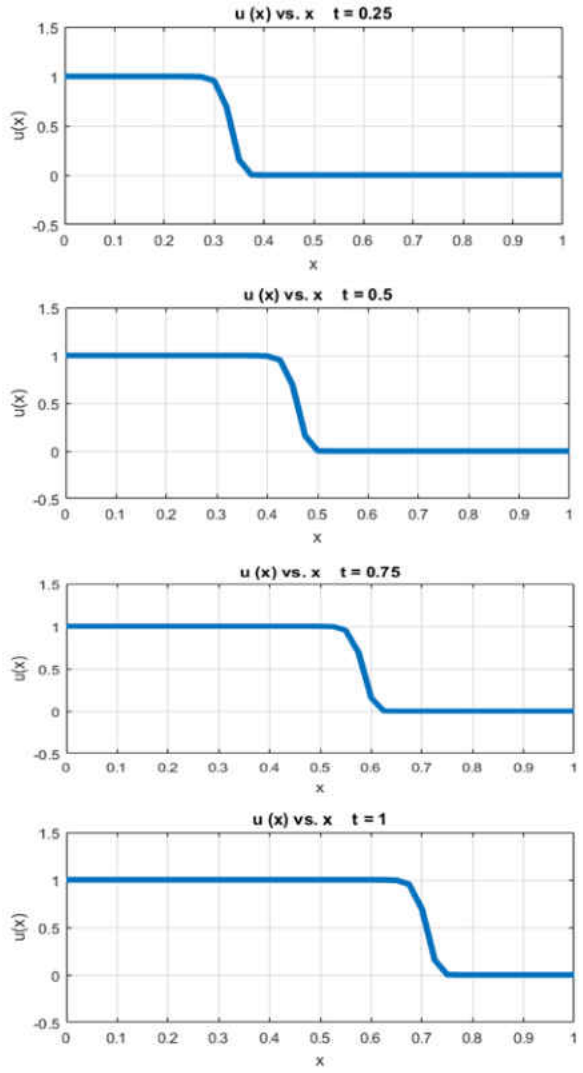


Figure 17: Results for the 1-D Burgers Equation using the blended RBF

In the next example, we investigate reducing the amount of dissipation introduced into the solution using blended RBF interpolation. We seek to benefit from the high accuracy of RBF interpolation and form a high-resolution scheme for shock capturing. We observe that the low shape parameter becomes dissipative as the solution progresses. For high shape parameter, the solution does well at keeping the gradients steep with no

dissipations, but this strategy tends to cause overshoots and undershoots near steep gradients. This is analogous to low order schemes being dissipative and high order schemes being dispersive commonly observed with other mesh-based schemes in CFD. Methods like total variation diminishing (TVD) and weighted essentially non-oscillatory schemes seek to minimize or eliminate oscillations by switching schemes near steep gradient and shocks [64]. The blended RBF scheme uses the attributes of low and high shape parameter. We want to capture the steep gradients as accurately as possible without oscillations. A 1- D advection equation is used for this numerical experiment. The governing equation can be written as

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0 \qquad (112)$$

For the first example, a top-hat profile is used for the initial condition shown in Table 6.

Table 6: Initial Conditions for Example 1

$u(x,0) = 0$	$x < 0.2$
$u(x,0) = 1$	$0.2 \leq x \leq 0.5$
$u(x,0) = 0$	$x > 0.5$

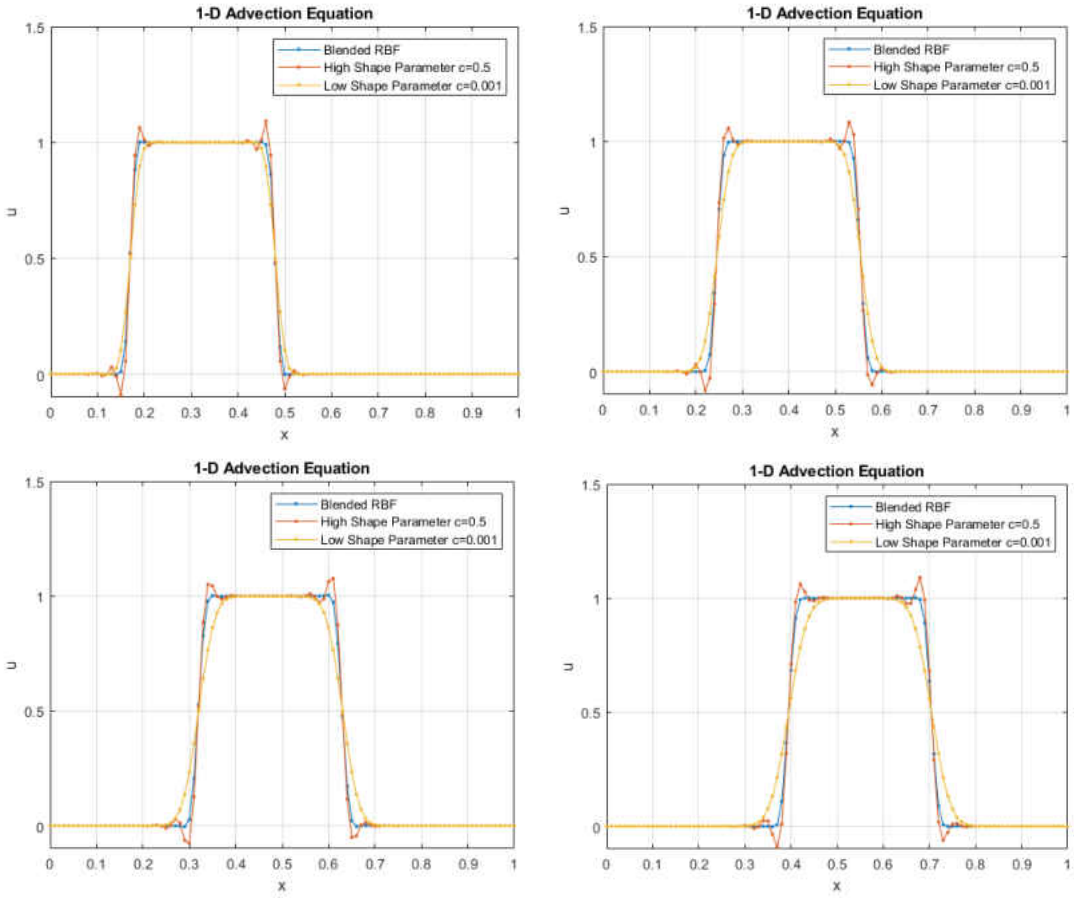


Figure 18: 1-D Advection using Example 1 initial conditions

The solution is evaluated for three different cases of low, high, and a blended shape parameter. Also, the solution is present in Figure 18 and Figure 19. The minmod flux limiter is used for the blended solution. The shape parameter values used are $c = 0.5$ and $c = 0.001$ for the high and low interpolation, respectively. The solution of the one-dimensional advection equation using the blended RBF collocation method works well capturing the gradient as close to the high shape parameter solution while using the low shape parameter solution when oscillations are detected. Also, the results provide proof that the scheme is a high-order scheme for hyperbolic partial differential equations. To

demonstrate the robustness of the blending approach a different set of initial conditions are used giving the u(x) profile represented by

Table 7: Example 2 Initial Conditions

$u(x,0) = 0$	$x < 0.1$
$u(x,0) = 2$	$0.1 \leq x \leq 0.3$
$u(x,0) = 1$	$0.3 < x \leq 0.5$
$u(x,0) = 0$	$x > 0.5$

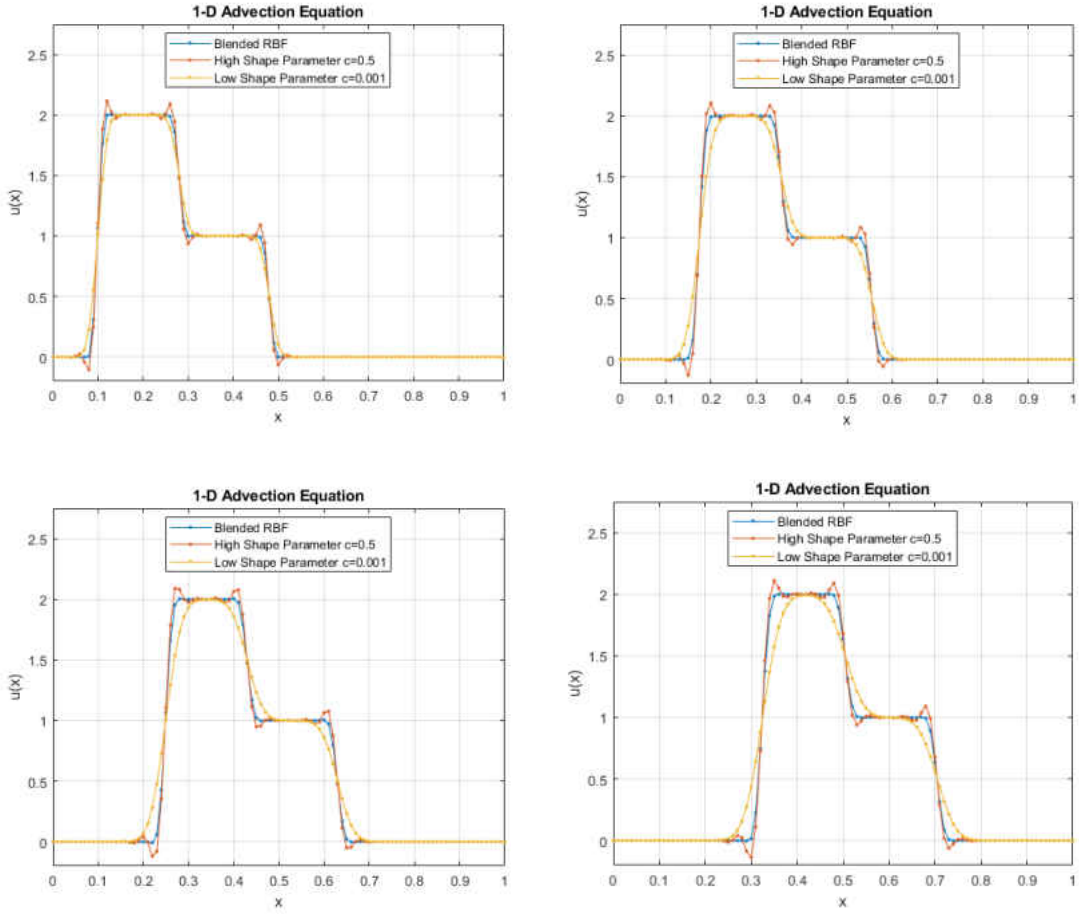


Figure 19: 1-D Advection Equation using Example 2 initial conditions

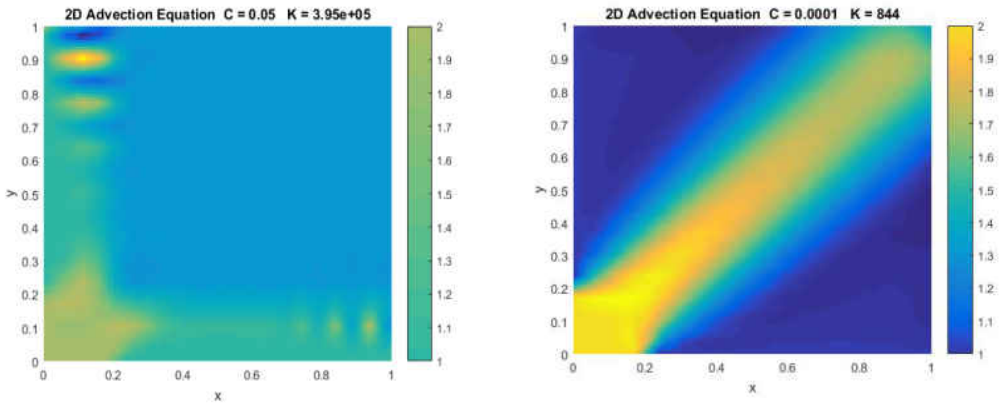
6.3 Two-Dimensional Linear Advection Equation

The two-dimensional linear advection equation is solved using the RBF blended interpolation. Two cases are presented to demonstrate this approach. The first case is the 45° diagonally traveling wave which will be referred to as Case 1 [67]. The governing equation, boundary conditions and initial conditions for this case are

$$\frac{\partial u}{\partial t} + U_1 \frac{\partial u}{\partial x} + U_2 \frac{\partial u}{\partial y} = 0 \qquad (113)$$

$$u(x,0,t) = 2 \; for \; x \leq 0.2$$
$$u(x,0,t) = 1 \; for \; x \geq 0.2$$
$$u(0,y,t) = 2 \; for \; y \leq 0.2$$
$$u(0,y,t) = 1 \; for \; y \geq 0.2$$
$$u(x,0) = 0$$
$$U_1 = \frac{\sqrt{2}}{2}, U_2 = \frac{\sqrt{2}}{2}$$

In Figure 20, the solution to case 1 is shown for constant shape parameter providing a high conditioning number for the interpolation matrix compared to a solution using the RBF blended approach. The solution obtained using the constant shape parameter becomes oscillatory and unstable early in the simulation, while the blended shape parameter solution is stable.



Unstable Stable

Figure 20: Comparison of Case 1 with a constant shape parameter with high condition number (Left) to the RBF blended interpolation approach (Right).

The second case is the turning wave which will be referred to as Case 2 [67]. The governing equation, boundary conditions and initial conditions for this case are

$$\frac{\partial u}{\partial t} + U_1 \frac{\partial u}{\partial x} + U_2 \frac{\partial u}{\partial y} = 0 \qquad (114)$$

where

$$U_1 = 2y[1 - (x-1)^2]$$
$$U_2 = -2(x-1)(1-y^2)$$

The solution of the turning wave is shown in Figure 21 and Figure 22. Again, for constant shape parameter and high conditioning number, the solution eventually becomes oscillatory and unstable. The solution remains stable by allowing the RBF interpolation to be blended between high and low shape parameter.

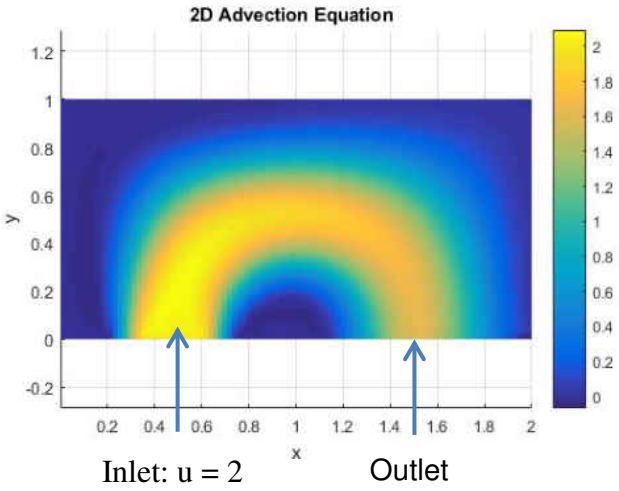


Figure 21: Turning Wave prescribed boundary conditions.

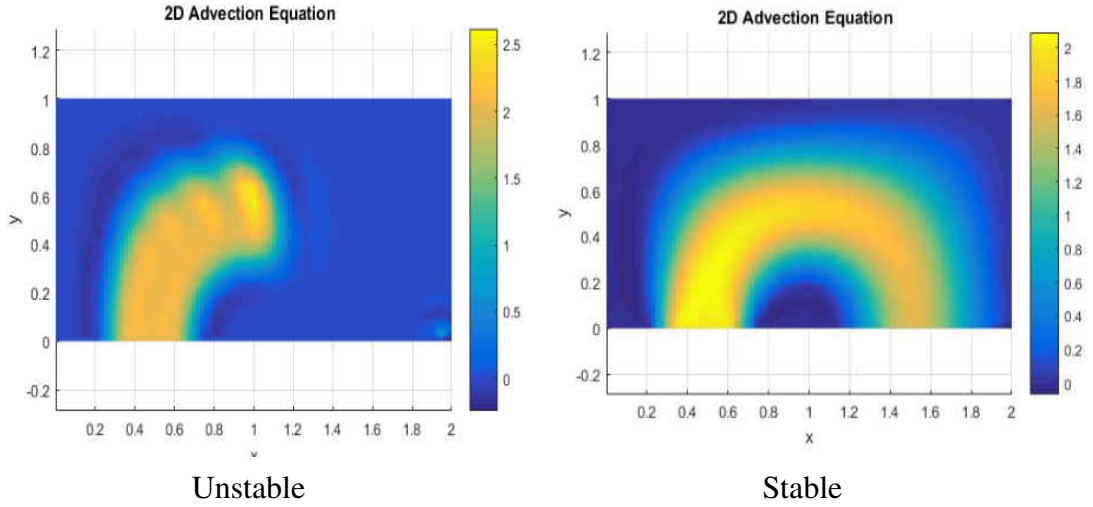


Figure 22 Comparison of Case 2 with a constant shape parameter with high condition number (Left) to the RBF blended interpolation approach (Right).

6.4 Inviscid Compressible Euler Equations

The local blended RBF collocation method has shown to work well for one and two-dimensional hyperbolic equations. The next numerical experiment will be to test the blended RBF scheme on the inviscid Euler equations. Supersonic flow in two-dimensional channel flow will be used for the next test case. A 10° wedge is introduced in the domain to induce an oblique shock wave. Inlet boundary conditions are prescribed as Mach number $M = 2$, pressure $P = 1e^5$ Pa, density $\rho = 1.16\ \frac{kg}{s}$ and temperature $T = 300\ K$. The outlet boundary conditions use extrapolated numerical boundary conditions that are update with each timestep [73, 79, 80]. The number of points within the local topology for the RBF collocation is apprInitially the field is smooth, with no steep gradients or shocks present in the domain. The goal of this numerical experiment is to use the localized blended RBF collocation method to capture the shock. The governing equations for inviscid compressible flow are

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0 \qquad (115)$$

where

$$Q = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_t \end{Bmatrix} \qquad F = \begin{Bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ (\rho e_t + P)u \end{Bmatrix} \qquad G = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ (\rho e_t + P)v \end{Bmatrix}$$

The total energy and the pressure can be found by

$$e_t = e + \frac{1}{2}(u^2 + v^2) \qquad P = \rho(\gamma - 1)e$$

The equations are solved using the Steger-Warming [57, 73, 74, 75, 76, 77], Van Leer [58, 77, 75, 73, 74], and the AUSM/AUSM+ [78, 59] inviscid flux schemes as described in the previous sections with the use of blended RBF collocation method. The domain is shown in Figure 24.

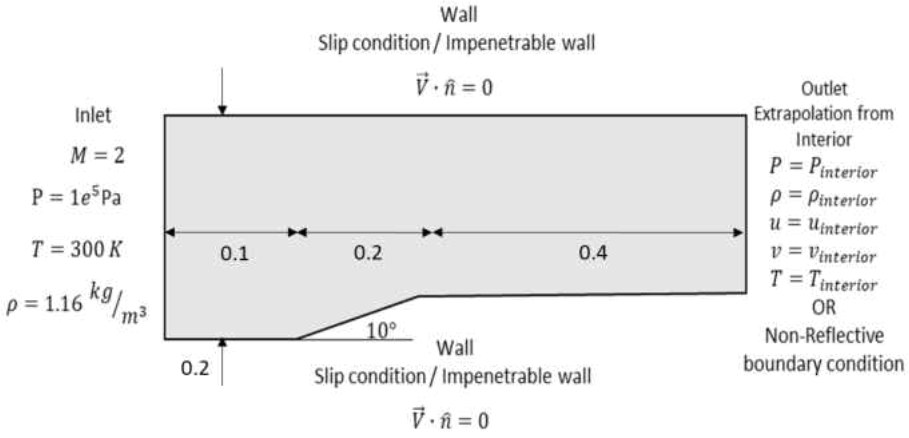


Figure 23: Problem domain and boundary conditions for the blended RBF scheme with Flux Splitting

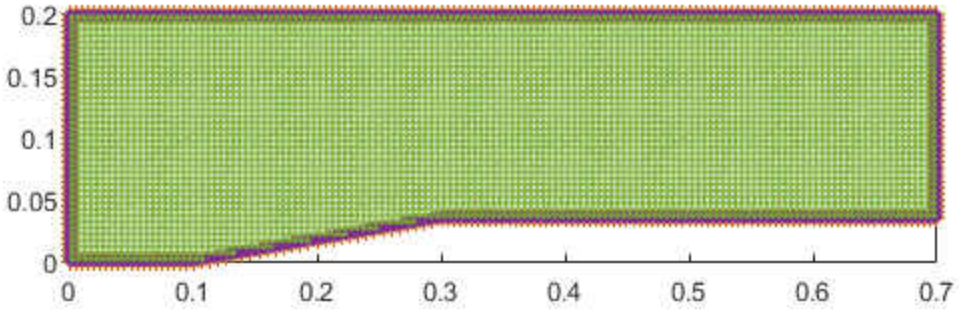


Figure 24: Point distribution for supersonic channel flow example

For the first attempt of solving the Euler equations, the Steger-Warming flux splitting method is used. After surveying many upwind schemes, the formulation of flux

splitting schemes is favorable for using RBF derivatives to solve the partial differential equation. For example, the Steger-Warming flux splitting method is given by

$$\frac{\partial Q}{\partial t}+\frac{\partial F^{+}}{\partial x}+\frac{\partial F^{-}}{\partial x}+\frac{\partial G^{+}}{\partial y}+\frac{\partial G^{-}}{\partial y}=0$$

The flux components F^{+}, F^{-}, G^{+}, and G^{-} can be calculated for the collocation points within the local support domain. With the flux components known, the derivatives, $\frac{\partial F^{+}}{\partial x}, \frac{\partial F^{-}}{\partial x}$ and $\frac{\partial G^{+}}{\partial y}, \frac{\partial G^{-}}{\partial y}$ can be approximated using low and high shape parameter RBF derivative interpolation vectors. The blending of the high and low RBF interpolation is either determined using a flux limiter or a smoothness indicator approach. The method is illustrated in Figure 25 and Figure 26. The derivatives must be approximated upwind of the direction of the flow. In other words, the derivatives $\frac{\partial F^{+}}{\partial x}, \frac{\partial F^{-}}{\partial x}$ must be approximated at the west and east nodes, respectively, as shown in Figure 25. Likewise, the y-component derivatives $\frac{\partial G^{+}}{\partial y}, \frac{\partial G^{-}}{\partial y}$ must be solved in a upwinded fashion at the south and north nodes. The time derivative is a forward difference and the PDE is solved explicitly as.

$$Q^{n+1}=Q^{n}-\Delta t\left[\frac{\partial F^{+}}{\partial x}+\frac{\partial F^{-}}{\partial x}+\frac{\partial G^{+}}{\partial y}+\frac{\partial G^{-}}{\partial y}\right]$$

The advantage of this approach is the increase in accuracy of using blended RBF derivatives to solve the PDE. In this approach, no finite differencing is performed therefore maintaining the spectral accuracy capabilities of the RBF collocation method.

Using this approach provides a fully meshless method for solving the inviscid compressible flow governing equations.

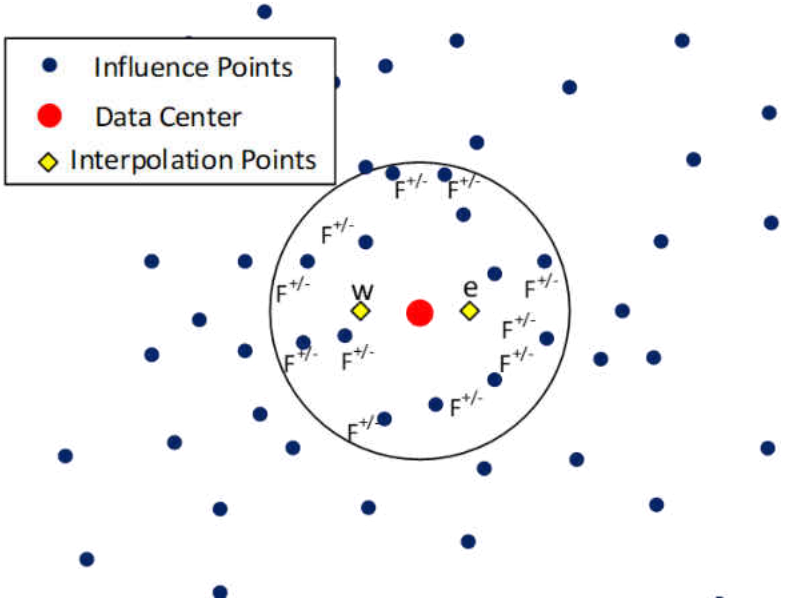


Figure 25: RBF Meshless Method for Flux Splitting for x component

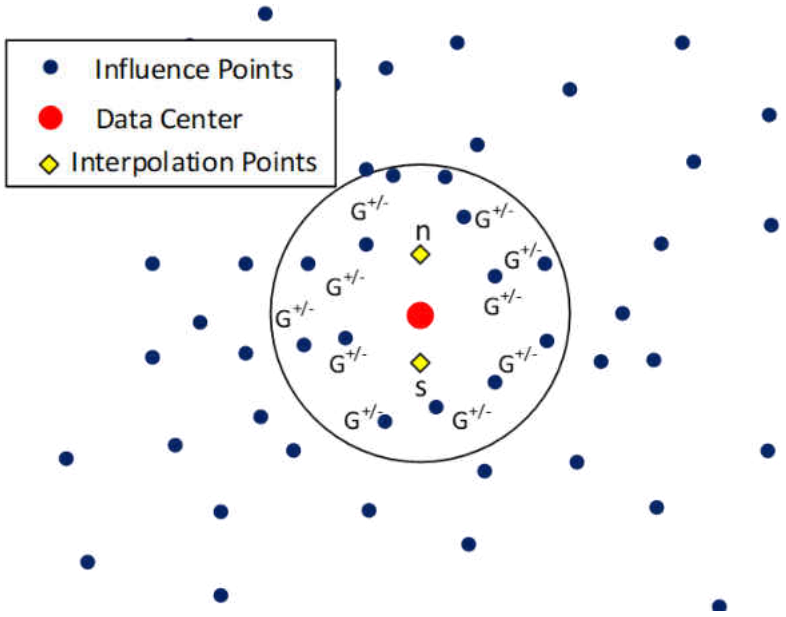


Figure 26: RBF Meshless Method for Flux Splitting for y component

The results for the Steger-Warming and Van Leer flux splitting schemes using blended RBF interpolation are shown in the following figures. The approach works well capturing the shock with minimal dissipation. Results are compared to a finite volume method solver. The number of points used for the localized RBF collocation method is 20,279, and the number of cells used for the finite volume solution is 41,514. The localized blended collocation method captures the shock and reflected shocks well using less than half of the number points used for the finite volume solution. The static pressures, density, and temperature can be obtained from oblique shock relations to verify the numerical results. Qualitative and quantitative comparisons are presented in the following figures and tables.

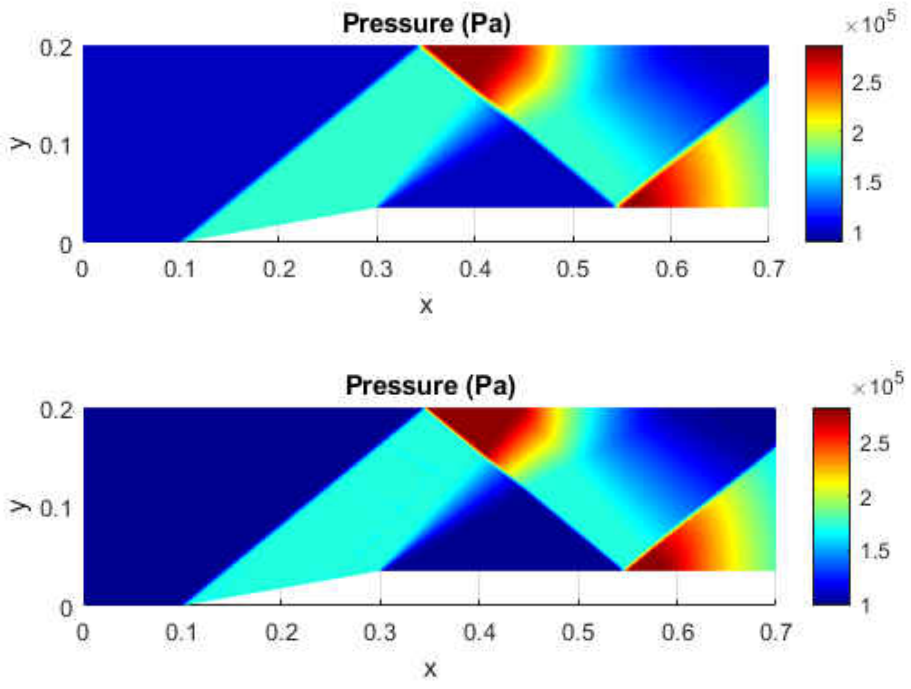


Figure 27: Comparison between Localized Blended RBF Collocation Method (top) and finite volume method (bottom)

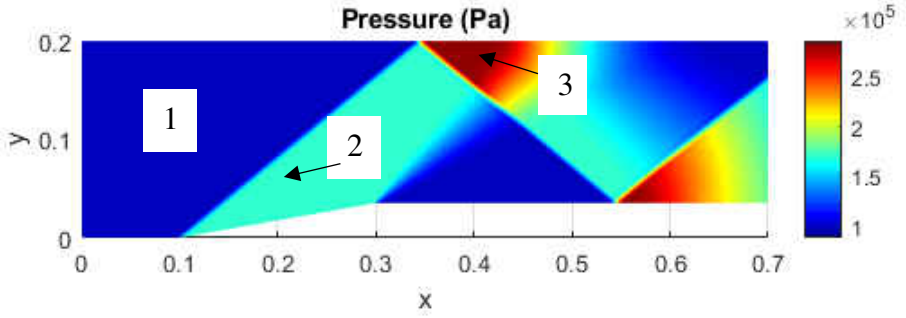


Figure 28: Locations for analytical and numerical comparisons

Table 8: Comparison of results for localized blended RBF collocation

Oblique Shock Relations				
Location	Mach	Pressure (Pa)	Density (kg/m^3)	Temperature (K)
1	2.000	1.00E+05	1.161	300.000
2	1.641	1.71E+05	1.693	351.045
3	1.285	2.80E+05	2.405	405.958
Localized Blended RBF Collocation				
Location	Mach	Pressure (Pa)	Density (kg/m^3)	Temperature (K)
1	2	1.00E+05	1.161	300
2	1.641	1.71E+05	1.694	351.05
3	1.286	2.80E+05	2.406	405.9
Finite Volume Method (FVM)				
Location	Mach	Pressure (Pa)	Density (kg/m^3)	Temperature (K)
1	2	1.00E+05	1.16198	300
2	1.63953	1.70E+05	1.68475	350.544
3	1.2887	2.79E+05	2.40177	405.132

Table 9: Percent difference of results compared to oblique shock relations

Localized Blended RBF Collocation Method vs. Theory				
Location	Mach	Pressure (Pa)	Density (kg/m^3)	Temperature (K)
2	0.029%	0.025%	0.045%	0.001%
3	0.086%	0.007%	0.039%	0.014%
Finite Volume Method (FVM) vs. Theory				
Location	Mach	Pressure (Pa)	Density (kg/m^3)	Temperature (K)
2	0.060%	0.351%	0.501%	0.143%
3	0.297%	0.310%	0.137%	0.203%

The percent difference between the localized blended RBF and analytical results are shown in Table 9. Also, the finite volume method is compared with analytical results to further illustrate the localized blended RBF capability. Localized blended RBF results give significantly lower error compared to the finite volume method. Low errors using the localized RBF collocation method are expected because the RBF is capable of spectral accuracy.

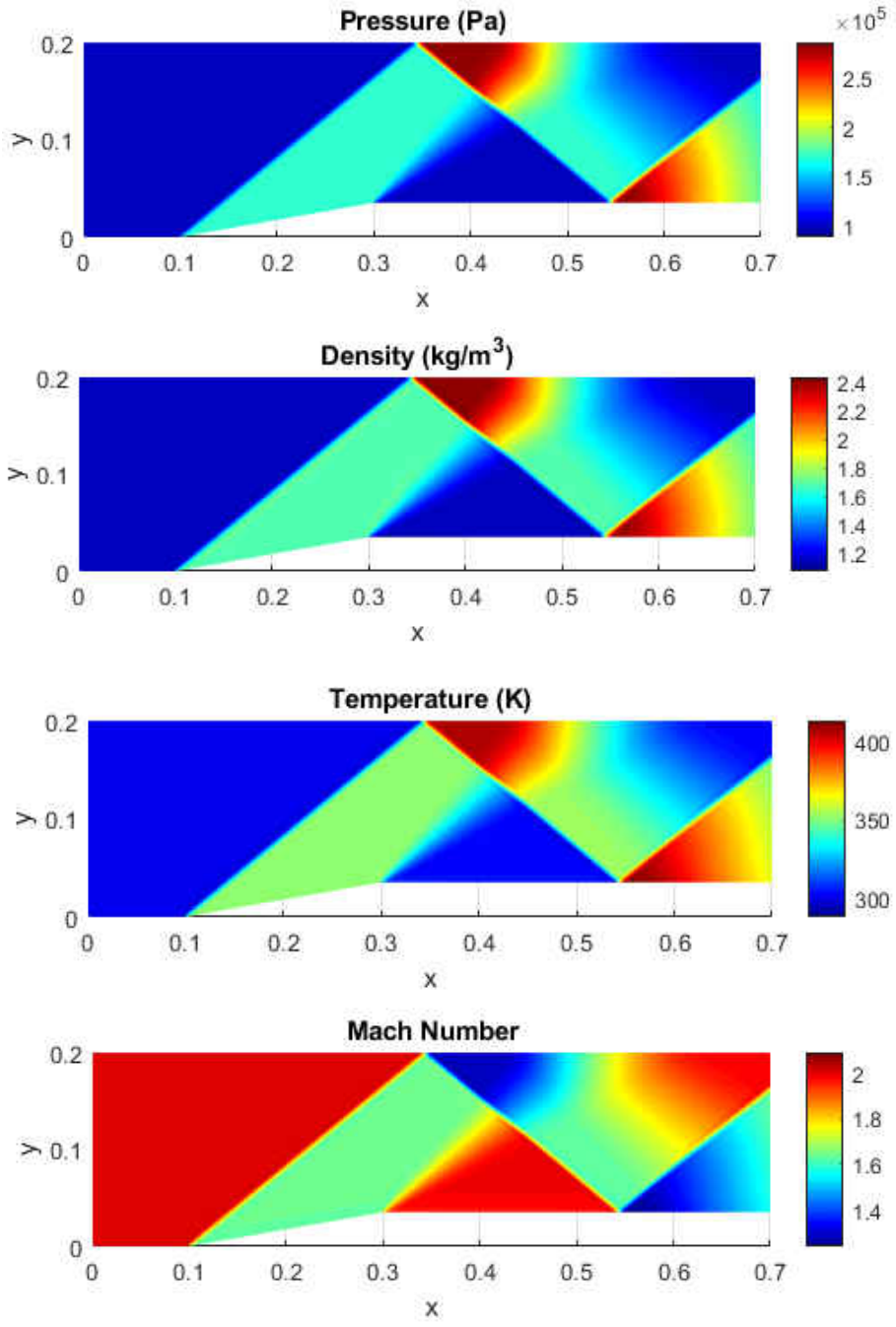


Figure 29: Steger-Warming flux splitting for supersonic wedge

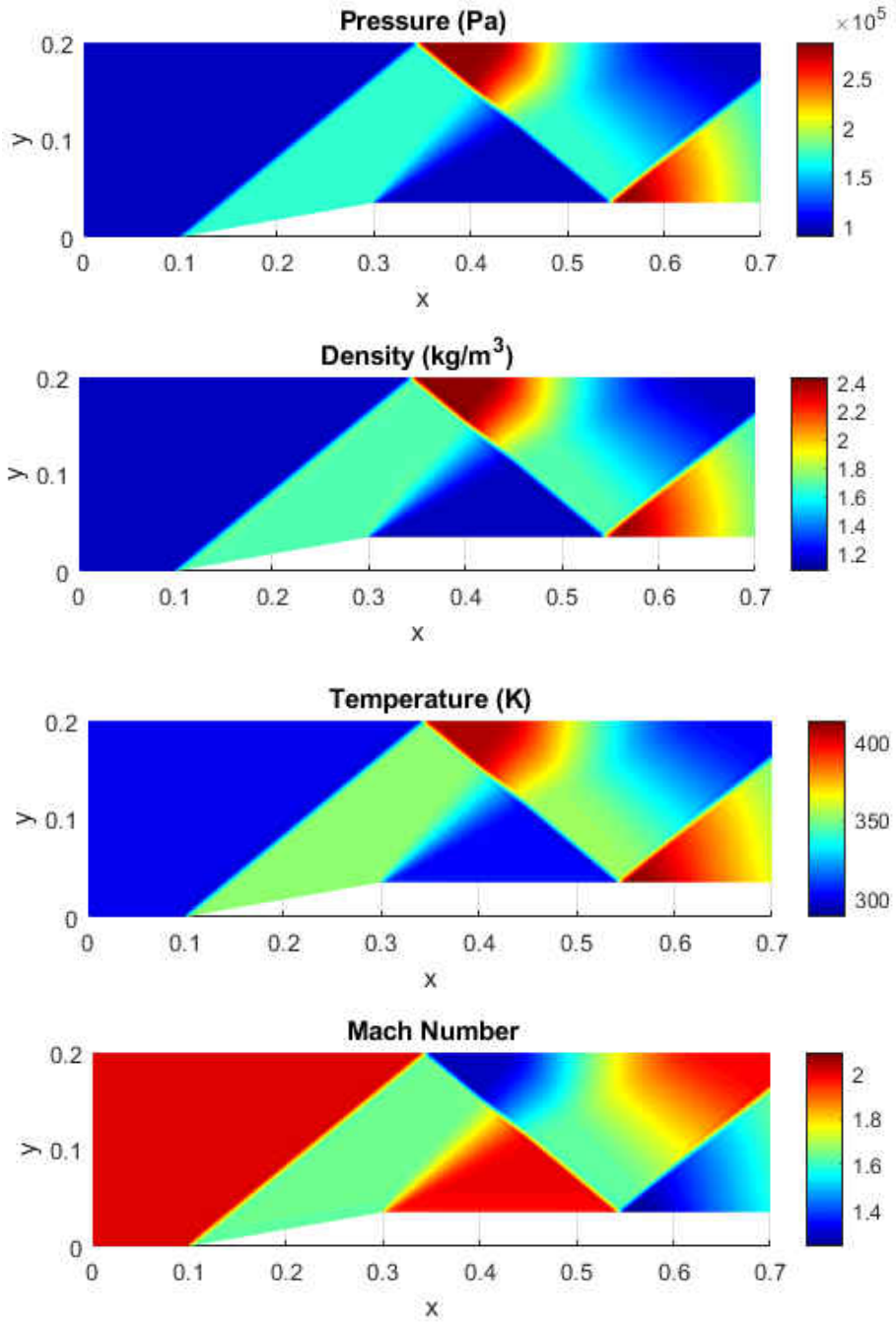


Figure 30: Van Leer flux splitting for supersonic flow over a wedge

A case of constant high shape parameter was performed to ensure that the blending scheme was providing stability. Figure 28, shows early results for a Mach 3 test case. Oscillations near the wall emerge due to the high-pressure gradients and the solution quickly becomes unstable due to unphysical values.

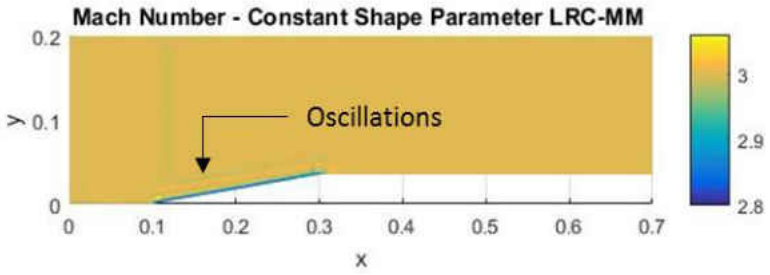


Figure 31: Solution using constant high conditioning RBF interpolation

Again, the localized blended RBF collocation method adds stability near steep gradients and shocks

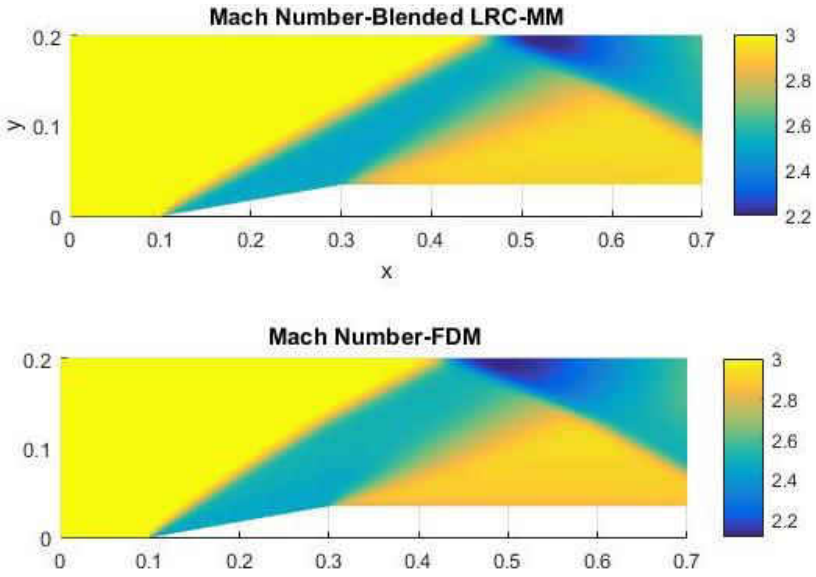


Figure 32: Mach number comparison of the local blended RBF collocation solution with finite difference method for M = 3

A case for the supersonic airfoil is presented showing good results compared to analytical results.

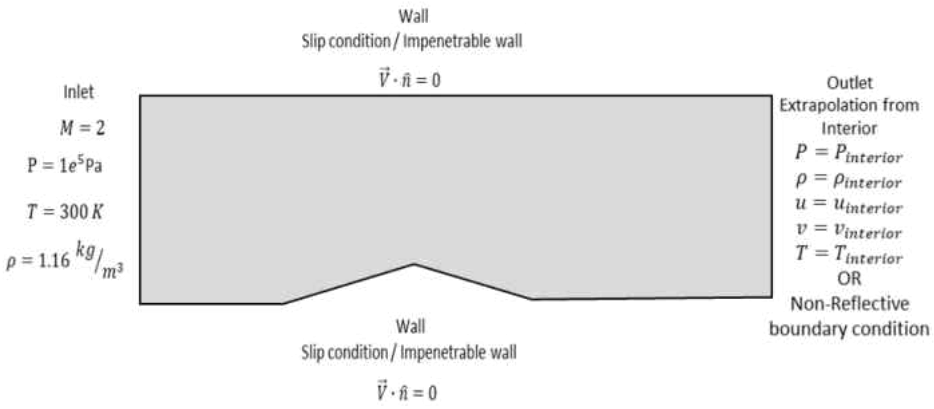


Figure 33: Boundary conditions for supersonic airfoil example

The oblique shock is captured well at the leading edge followed by an expansion wave. This example further demonstrates the local blended RBF scheme performance by testing the scheme on rapidly varying fields. The results are presented in the following figures.

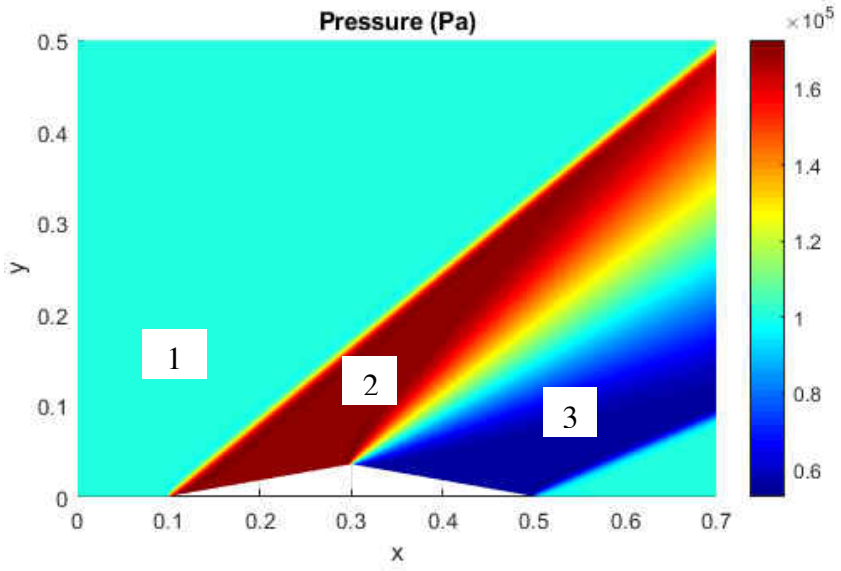


Figure 34: Locations for analytical and numerical comparisons

Table 10: Comparison of results for supersonic airfoil

Localized Blended RBF Collocation				
Location	Mach	Pressure (Pa)	Density (kg/m^3)	Temperature (K)
1	2.000	100000.0	1.161	300.000
2	1.640	170663.6	1.693	351.401
3	2.360	55387.9	0.755	255.796
Oblique Shock Relations				
Location	Mach	Pressure (Pa)	Density (kg/m^3)	Temperature (K)
1	2.000	100000.0	1.161	300.000
2	1.641	170657.9	1.694	351.401
3	2.372	55078.5	0.755	254.116

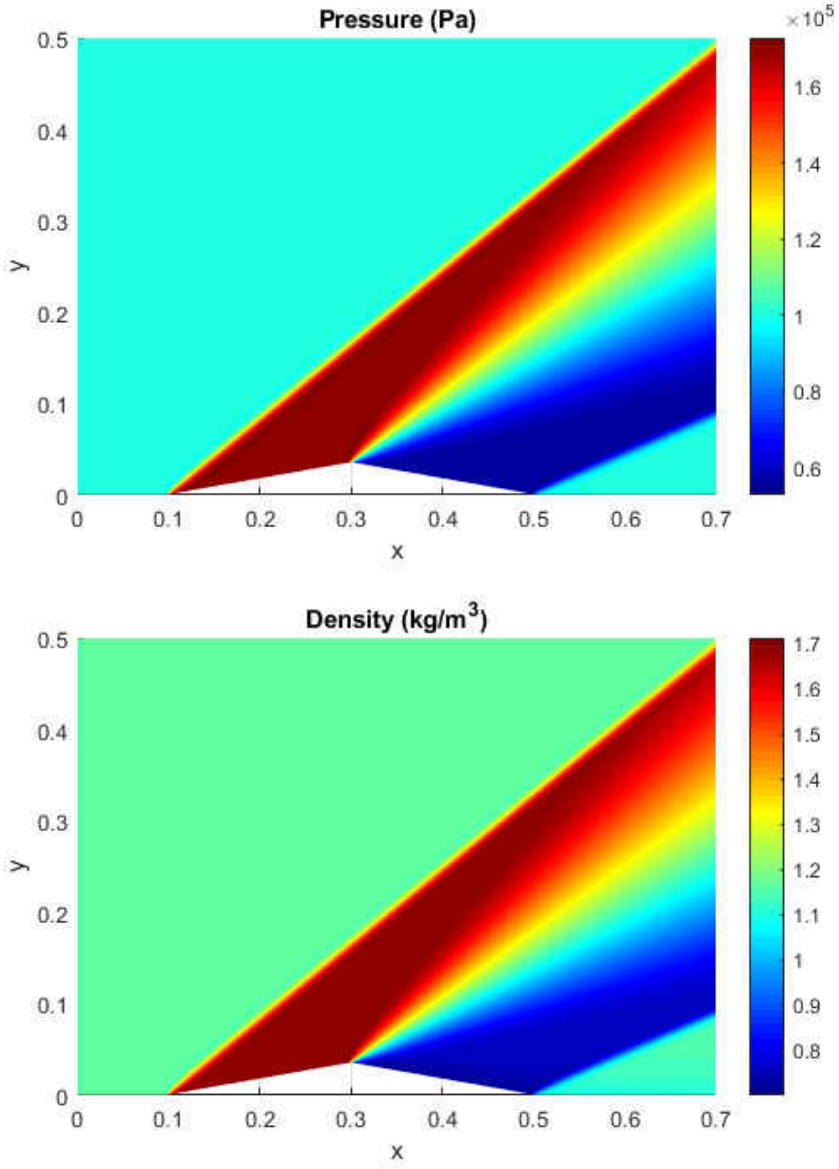


Figure 35: Pressure and density for supersonic airfoil

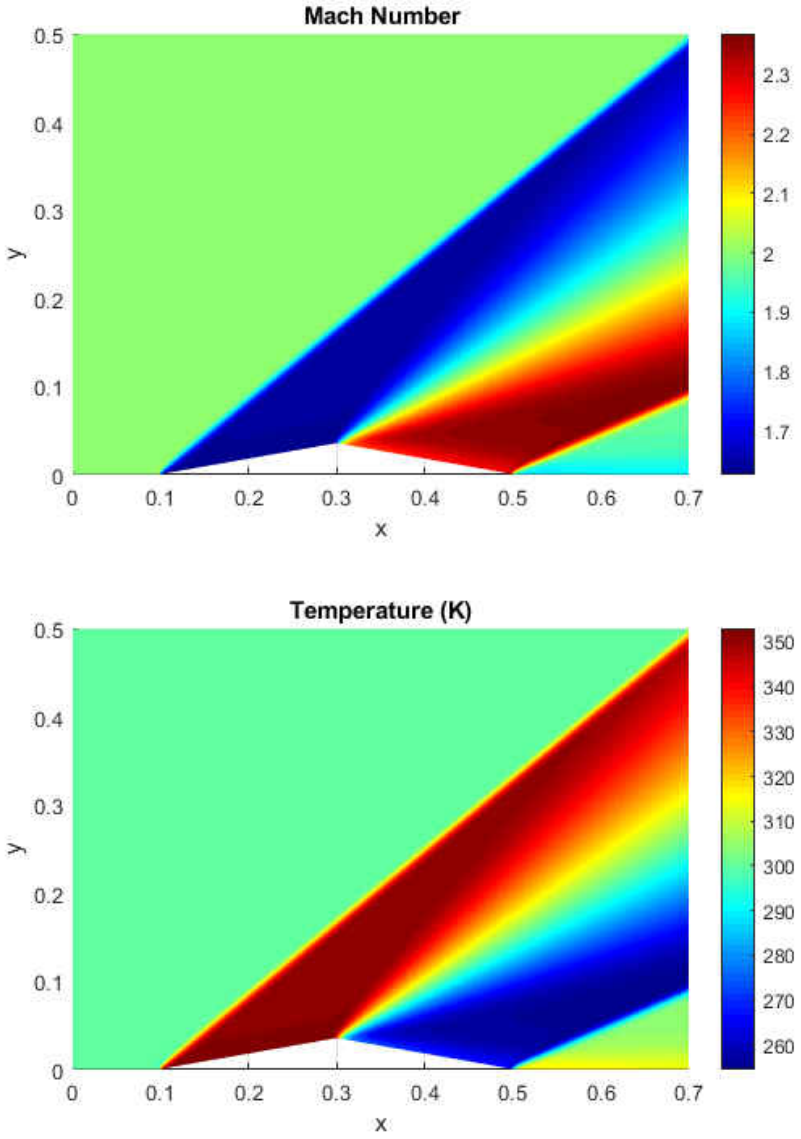


Figure 36: Mach number and temperature for supersonic airfoil using Steger-Warming flux splitting

The AUSM/AUSM+ scheme is used to solve the Mach 2 supersonic wedge test case. A different RBF interpolation approach to accommodate the AUSM scheme is needed. To accomplish this, the enhanced RBF finite difference method is used along with the

blending methodology. The field variables are interpolated using high and low shape parameter interpolation. The flux values are blended between high and low interpolation depending on the smoothness of the field measured by the successive gradient or smoothness indicator. The blended field values are then used to calculate the fluxes for the designed finite difference stencil. Figure 37 further illustrates this strategy.

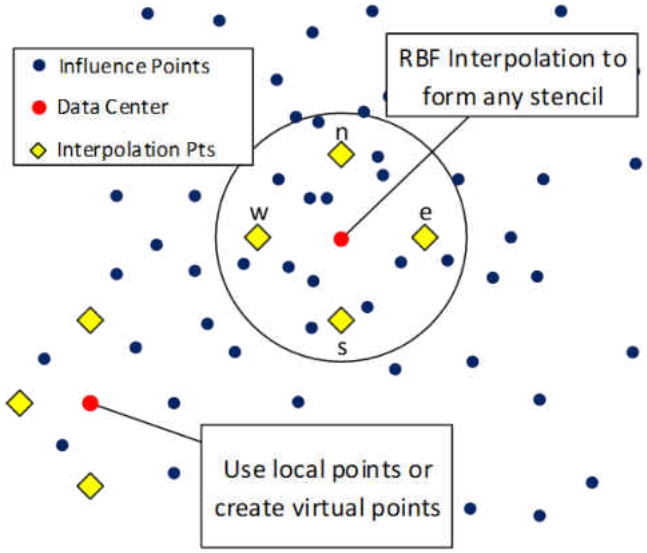


Figure 37: RBF enhanced finite difference stencil for AUSM scheme.

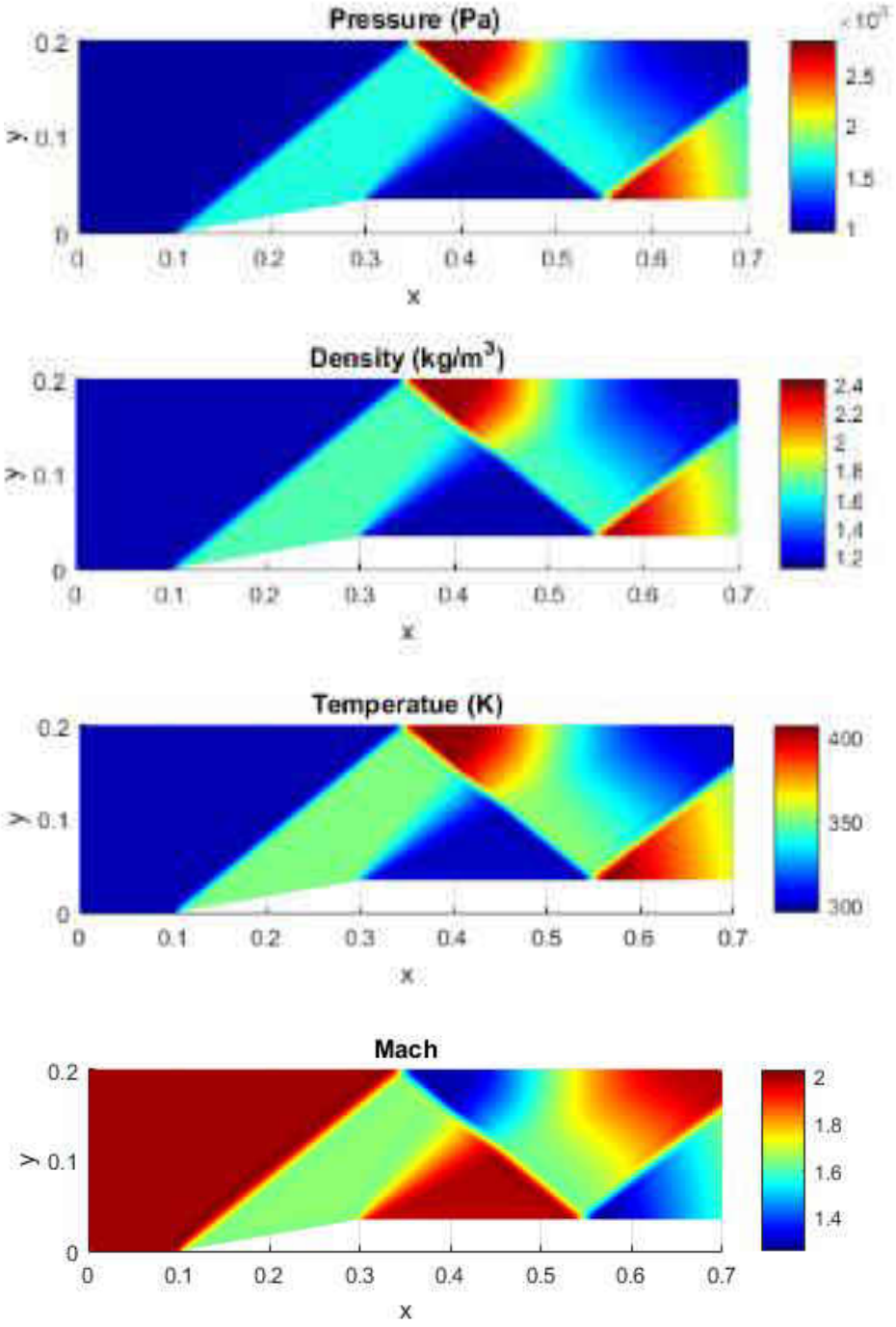


Figure 38: AUSM+ results for 2-D supersonic wedge test case

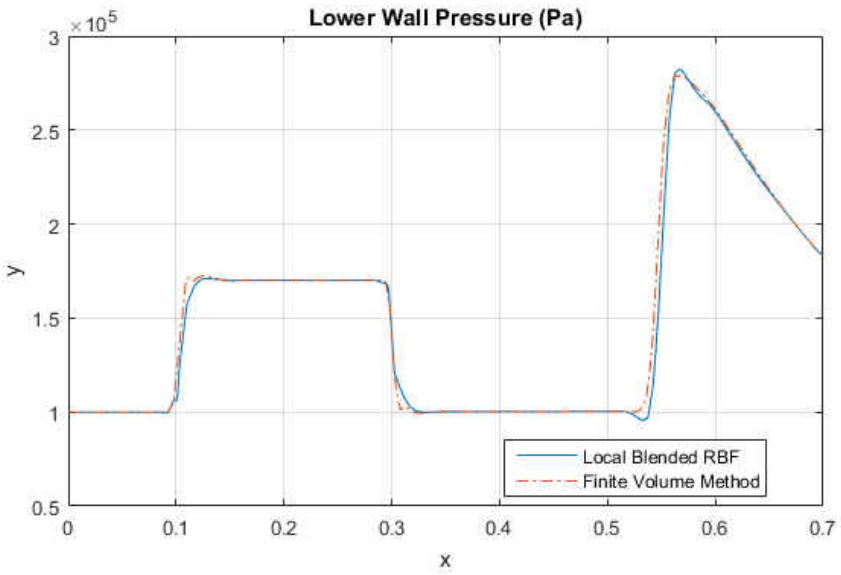


Figure 39: Lower Pressure Comparison for Local Blended RBF and FVM

The results for the AUSM/AUSM+ scheme using the local blended RBF interpolation match well with oblique shock relations, and finite volume method solutions. The enhanced finite difference combined with blended RBF is a good strategy to unite the developed high-resolution CFD schemes for shock capturing with RBF interpolation.

Mass was verified to be conserved in the domain. The mass balance confirms that the solution converged and is accurate. The following plot shows the difference of the mass entering and exiting at the outlet reaching a value close to zero approximately 1e-4.

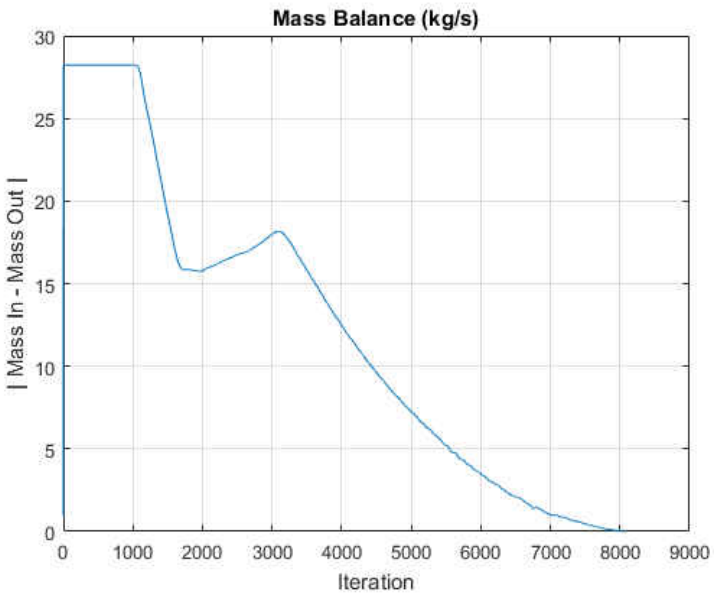


Figure 40: Mass conservation of the Local Blended RBF scheme

6.5 Navier-Stokes Equations & Conjugate Heat Transfer

The blended RBF interpolation scheme gave satisfactory results for the Euler equations,. The next step is to add viscosity to the fluid flow; therefore, we would like to solve the Navier-Stokes equations. Due to the nature of high Reynolds number flows, the boundary is thin, making high-speed viscous flow one of the most difficult problems in computational fluid dynamics. The governing equations can be written in vector notation as

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = \frac{\partial F_v}{\partial x} + \frac{\partial G_v}{\partial x} \qquad (116)$$

where

$$Q = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_t \end{Bmatrix} \qquad F = \begin{Bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ (\rho e_t + P)u \end{Bmatrix} \qquad G = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ (\rho e_t + P)v \end{Bmatrix}$$

and

$$F_v = \begin{Bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} - q_x \end{Bmatrix} \qquad G_v = \begin{Bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xy} + v\tau_{yy} - q_y \end{Bmatrix}$$

$$e_t = e + \frac{1}{2}(u^2 + v^2) \qquad P = \rho(\gamma - 1)e$$

The shear stresses are given by [75]

$$\tau_{xx} = \frac{2}{3}\mu\left(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right) \quad (117)$$

$$\tau_{yy} = \frac{2}{3}\mu\left(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x}\right) \quad (118)$$

$$\tau_{xy} = \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \quad (119)$$

$$q_x = -k\frac{\partial T}{\partial x} \quad (120)$$

$$q_y = -k\frac{\partial T}{\partial y} \quad (121)$$

where k is the thermal conductivity.

The inlet and outlet boundary conditions remain the same as the Euler equation examples and are provided in the following figure. The wall boundary condition is using

the no-slip condition. Therefore, the velocity at the wall is zero. A conjugate heat transfer boundary condition is applied to the lower fluid domain wall, and the temperature is solved within the solid domain using RBF direct differentiation. The solid material chosen for this case is titanium. The boundary conditions for solid side walls are prescribed to be adiabatic, and the lower solid wall is maintained at a constant temperature of 300 K.

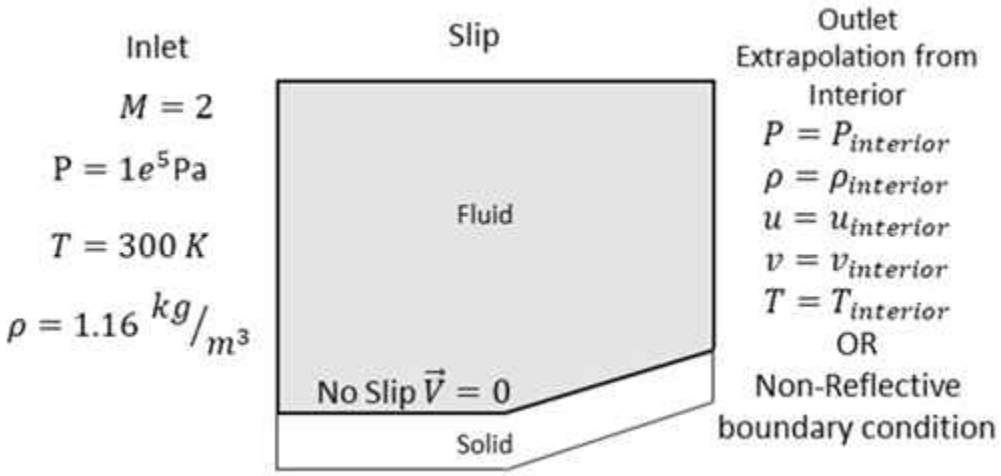


Figure 41: Boundary Conditions for Navier-Stokes with CHT

Points are closely distributed near the wall boundaries to capture the boundary layer and the approximate the shear stresses. To overcome non-uniform nodal arrangement and choice of shape parameter, the local support of the RBF should be scaled based on the maximum distance of nodes within the local support [45, 46, 81, 82]. This approach reduces the shape parameter dependency when using non-uniform node distribution. The derivatives for the diffusion terms are solved using moving least squares or weighted

least squares approach [73, 87, 88, 89]. The moving least squares or weighted least squares methods can provide stability opposed to RBF collocation second derivatives.

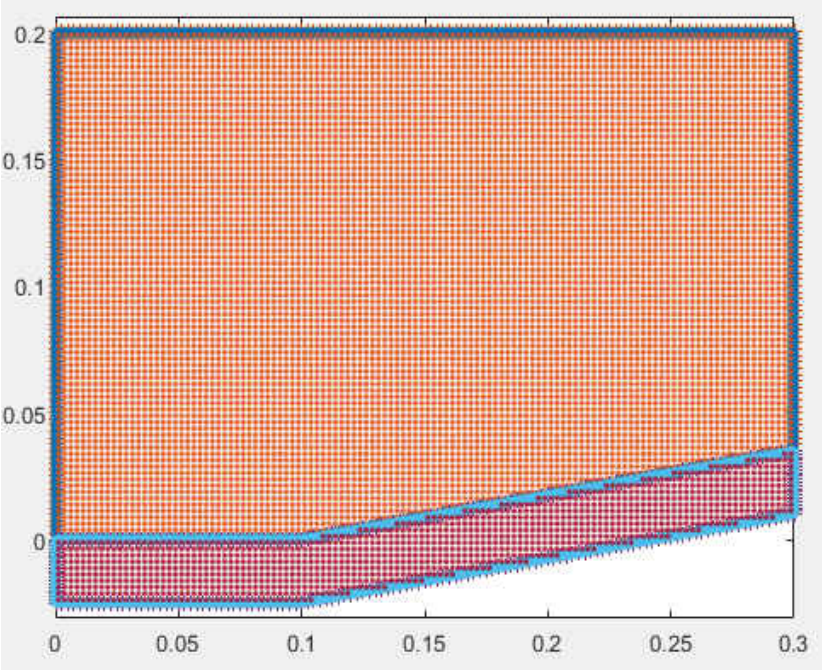


Figure 42: Domain for Navier-Stokes Solution with CHT

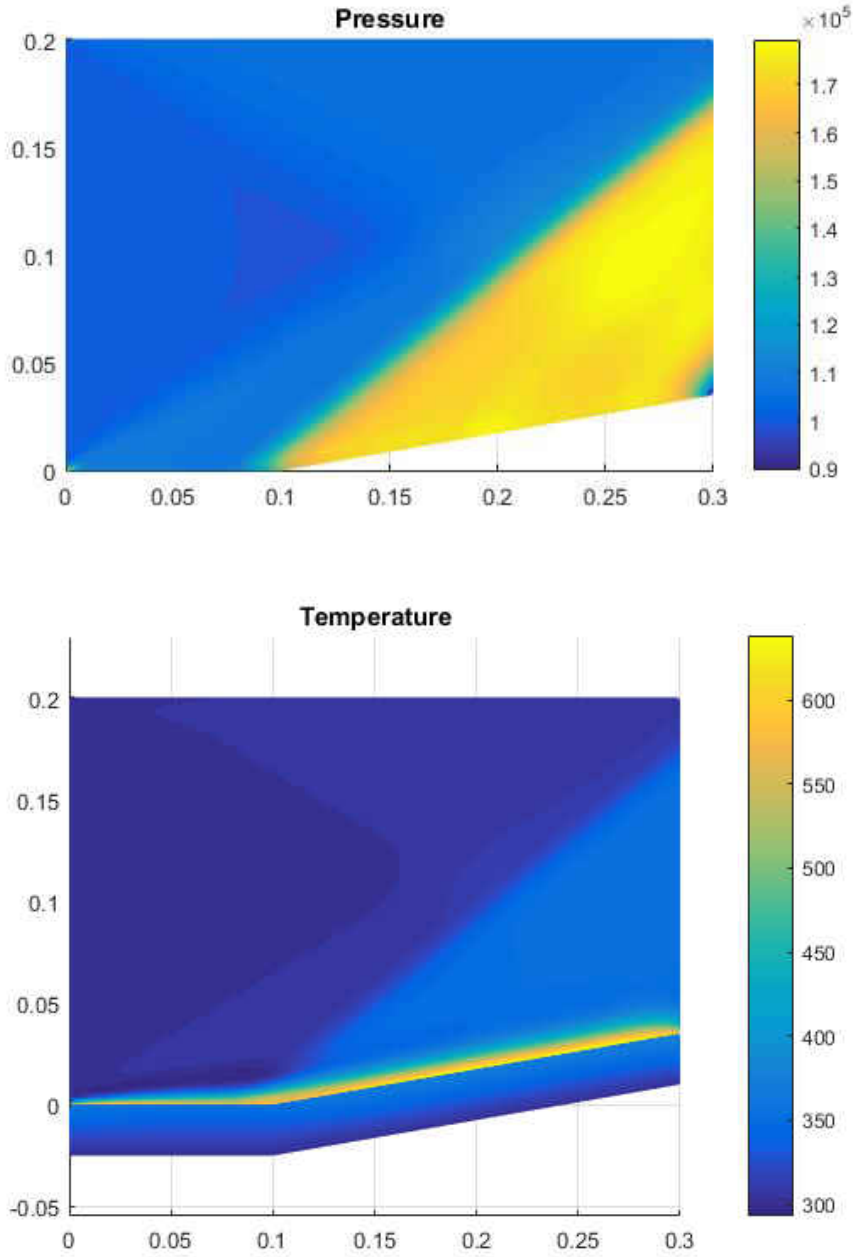


Figure 43: Navier-Stokes Solution using blended RBF interpolation for Pressure (Pa) and Temperature (K)

The local blended RBF collocation method captures the shock well for the Navier-Stokes solution. The stagnation temperature at the wall downstream of the shock for a

freestream Mach number M=2 is approximately 632 K and the static pressure across the oblique shock is expected to be 1.707e5 Pa using shock relations. The numerical results are close to shock relation theoretical values. However, the solution has numerical dissipation that needs to be reduced, especially near the wall boundary and will be future work for this research.

CHAPTER 7

CONCLUSIONS

The localized blended radial basis function (RBF) collocation method has shown to solve hyperbolic partial differential equations by solving several numerical experiments in one and two dimensions. This approach made use of radial basis function interpolation using different shape parameters to create low and high conditioning interpolation vectors. The high shape parameter interpolation was used for smooth regions while the low shape parameter was used when the steep gradients or shocks formed. This approach is analogous to other high-resolution schemes in CFD where blending between high and low order schemes are used to prevent oscillations near steep gradients.

The effects of shape parameter on RBF interpolation is discussed, and examples are provided. For smooth functions, the RBF interpolation using a high condition collocation matrix does well in reproducing the test function and determining the derivatives. As the shape parameter is increased, the error for the function and derivatives decrease exponentially. For most problems, the high shape parameter RBF should be used most often, if possible. However, increasing the slope of the function causes the high shape parameter RBF interpolation to produce oscillations. We seek to use the RBF interpolation for the solution of PDEs, therefore, the oscillations must be reduced or eliminated for a stable solution. Fortunately, the use of low shape parameter or low condition number RBF interpolation dampens oscillations providing a piecewise

linear reproduction of the test function. By allowing the numerical scheme to switch between high and low RBF interpolation, a robust high-resolution scheme is developed benefiting from the spectral accuracy capabilities of the RBF collocation method.

The blended RBF scheme was able to solve many different PDE example problems, such as the inviscid Burgers equation, two-dimensional advection equation to demonstrate the performance of the blended RBF scheme. Studying the blended RBF scheme using these simple numerical test cases allowed for the experimentation of different blending parameters strategy and ultimately finding a criterion that eliminates oscillation while minimizing the numerical dissipation introduced into the solution. Also, the 2-D inviscid compressible Euler equations and Navier-Stokes equations are solved using the blended RBF scheme. The Euler equations are solved using a few different flux schemes, such as Steger-Warming, Van Leer and, AUSM/AUSM+. The results for all schemes showed good agreement with analytical shock relations and commercial CFD. This research provides qualitative and quantitative comparisons for most test cases observed.

The blended RBF scheme can also be applied to mesh-based methods to improve the solution. The well-developed upwind schemes found in CFD are adopted and are used with the blended RBF scheme. Therefore, many CFD codes can use the blended RBF interpolation without significant changes or additions to the existing algorithm. Most mesh-based methods use some form of interpolation, especially in finite volume methods. A blended RBF interpolation could be added to a finite volume scheme to

increase the accuracy of the method. Overset or Chimera grids could also benefit from this type of interpolation since these methods interpolate data between grids.

The localized blended RBF collocation method provides a stable method that can be used to solve hyperbolic PDEs and introduces a new tool for future research. The many advantages of using meshless methods can now be used for hyperbolic PDEs. For example, the capability of adding nodes during the simulation to refine the solution around the shock can now be used effectively. Mesh-based methods have this capability, but the user must re-mesh the entire domain unless an adaptive mesh refinement algorithm is used. The meshless capability of adding and removing nodes with the addition of localized Blended RBF collocation allows improvements for many multiphysics applications such as fluid-structure interaction (FSI), biomedical applications, solidification, melting/ablation, and two-phase flow for example. The localized blended RBF collocation method can contribute to all types of applications mentioned.

Although the localized blended RBF collocation method worked well providing excellent results for shock capturing problems, further work is needed to improve upon the scheme. An implicit formulation or parallel processing needs to be implemented in the solver. An implicit solver or adding parallel processing capability to the solver should provide a significant speed up, especially for the Navier-Stokes equations. Also, the scheme should be attempted for problems in three-dimensions.

APPENDIX A

LOCALIZED BLENDED RBF COLLOCATION STEGER-WARMING/ VAN LEER EULER SOLVER

```
clc
close all


new_geo = 1;
if new_geo == 1

    clearvars -except new_geo;

else

    clearvars -except new_geo eta*;

end
%*****************Setting up Geometry and Point
Cloud*********************
% x and y vertices that make up the domain
xv = [0.0,0.1,0.3,0.7,0.7,0.3,0.1,0,0];
yv = [0.0,0.0,0.035265,0.035265,0.2,0.2,0.2,0.2,0.0];

% Number of points along each boundary
NB = [40,80,160,66,160,80,40,80];

% Number of edges that make up the domain
nume = 8;

% Boundary condition type  1 = Wall, 2 = Inlet, 3 = Outlet
BC = [1,1,1,3,1,2];

% Offset for the shadow numbers
offset = 0.0002;
offset2 = 0.0002;

% Number of interior points
NP = 50;

% Plotting the domain
figure(1)
plot(xv,yv)
title('Domain')
xlabel('x')
ylabel('y')
axis equal

layers = 1;
[xb,yb,xm,ym,tx,ty,nx,ny,xs,ys,xint,yint] =
preprocessing(xv,yv,NB,offset,offset2,NP);

figure(2)
plot(xv,yv)
hold all
```

```
plot(xb,yb,'+')
plot(xm,ym,'.','MarkerSize',15)
% plot(xs,ys,'.','MarkerSize',15)
axis equal

Nx = 80;

Ny1 = 30;
Ny2 = 10;
k=1;
for i = 1:3
    for j = 1:NB(i)
        delta_y(k) = (max(ym) - ym(k))/Ny1;
        delta_x(k) = (max(xs)-min(xs))/Nx;
        k = k + 1;
    end
end

k = 1;
m = 1;
for i = 1:3
    for j = 1:NB(i)

        delta = (max(ys) - ys(k));

        for l = 1:Ny1-1

            yint2(m) = ym(k) + l*delta_y(k);
%             yint2(m) = ys(k) + delta/2*(1-
cos(pi*l*delta_y(k)/(delta)));
            xint2(m) = xs(k) ;
            m = m+1;

            %             hold off
            %             figure(3)
            %             plot(xv,yv)
            %             hold all
            %             plot(xb,yb,'+')
            %             plot(xm,ym,'.','MarkerSize',15)
            %             plot(xs,ys,'.','MarkerSize',15)
            %             plot(xint2,yint2,'.','MarkerSize',5)
            %             axis equal
            %             pause(0.001)

        end
        k = k + 1;
    end
end

buffer = offset;
for i = 1:length(xs)
```

```
    xpoly(i) = xs(i) - nx(i)*buffer;
    ypoly(i) = ys(i) - ny(i)*buffer;

end

[Xint,Yint] = meshgrid(0.00125:0.0025:0.3,0.00125:0.0025:0.35);
[Xint2,Yint2] = meshgrid(0.30125:0.0025:0.7,0.03125:0.0025:0.2);

% [Xint3,Yint3] = meshgrid(0:0.0025:0.7,0.062:0.0025:0.2);

% [xint,yint] = inDomain(xpoly,ypoly,Xint,Yint,1);
% % [xint2,yint2] = inDomain(xpoly,ypoly,Xint2,Yint2,1);
% [xint_s,yint_s] =
inDomain(xpoly_solid,ypoly_solid,Xint_solid,Yint_solid,-1);

[in,on] = inpolygon(Xint,Yint,xpoly,ypoly);
[in2,on2] = inpolygon(Xint2,Yint2,xpoly,ypoly);
% [in3,on3] = inpolygon(Xint3,Yint3,xpoly,ypoly);

% [Xint,Yint] = meshgrid(0.00025:0.0005:0.3,0.00025:0.0005:0.2);
% [Xint2,Yint2] = meshgrid(0.3025:0.001:0.7,0.0375:0.001:0.2);
% % [Xint3,Yint3] = meshgrid(0:0.0025:0.7,0.062:0.0025:0.2);
% [in,on] = inpolygon(Xint,Yint,xpoly,ypoly);
% [in2,on2] = inpolygon(Xint2,Yint2,xpoly,ypoly);
xint = [Xint(in);Xint2(in2)];
yint = [Yint(in);Yint2(in2)];

% xint = xint2';
% yint = yint2';

hold off
figure(3)
plot(xv,yv)
hold all
plot(xb,yb,'+')
plot(xm,ym,'.','MarkerSize',15)
plot(xs,ys,'.','MarkerSize',15)
plot(xint,yint,'.','MarkerSize',5)
axis equal
axis([0.0 0.7 0.0 0.2])

pause(0.001)

%*************************************************************************
****



%**********************Input
values***************************************
```

```
CFL  = 0.9;
Mach = 2.0;
Pinit = 100e3;
Tinit = 300;
MW = 28;
gama = 1.4;
R = 287.05;
mu = 1.846e-5;
kcond = 0.02624;
rhoinit = Pinit/(R*Tinit);
ainit = sqrt(gama*R*Tinit);
uinit = Mach*ainit;
vinit = 0.0;
etinit = (R*Tinit)/(gama-1) + 0.5*(uinit^2 + vinit^2);

%Subdomain Parameters
theta = [0:0.01:2*pi,0]; %Angles in radians for circular subdomain
rsub = 0.007;             %Radius of the circular subdomain

% dx = offset;          %step size in the x direction
% dy = offset;          %step size in the y direction

%Shape paramters
sf  = 0.05;            %shape parameter for smooth RBF interpolation
sf2 = 1e-3;            %shape parameter for discontinuous interpolation
sf3 = 0.001;
sf4 = 0.01;
%Time stepping paramters
dt = 0.001;             %time step
time = 10;              %end time
Nsteps = 1;           %total number of time steps

%*****************************************************************
****


%**********************Initialization*********************************
****

% Applying intial condtion to domain and applying boundary conditions

[rhom,rhos,rhoint,rho,rhoall] =
intializeRho(rhoinit,Nsteps,nume,NB,length(xint),8,layers);
[um,us,uint,u,uall] =
intializeU(uinit,Nsteps,nume,NB,length(xint),8,layers);
[vm,vs,vint,v,vall] =
intializeV(vinit,Nsteps,nume,NB,length(xint),8,layers);
[Pm,Ps,Pint,P,Pall] =
intializeP(Pinit,Nsteps,nume,NB,length(xint),8,layers);
[etm,ets,etint,et,etall] =
intializeEt(etinit,Nsteps,nume,NB,length(xint),8,layers);
```

```
[am,as,aint,a,aall] =
intializeSoS(ainit,Nsteps,nume,NB,length(xint),8,layers);


Vmagm = sqrt(um.^2+vm.^2);
Vmags = sqrt(us.^2+vs.^2);
Vmagint = sqrt(uint.^2+vint.^2);
Vmagall = sqrt(uall.^2+vall.^2);

Mm = Vmagm./am;
Ms = Vmags./as;
Mint = Vmagint./aint;
Mall = Vmagall./aall;

M = [Ms;Mint];

Tm = Pm./(R*rhom);
Ts = Ps./(R*rhos);
Tint = Pint./(R*rhoint);
Tall = Pall./(rhoall*R);

T = [Ts;Tint];

Q(1,:) = rho;
Q(2,:) = rho.*u;
Q(3,:) = rho.*v;
Q(4,:) = rho.*et;

Qall(1,:) = rhoall;
Qall(2,:) = rhoall.*uall;
Qall(3,:) = rhoall.*vall;
Qall(4,:) = rhoall.*etall;

% rho(:,:) = Q(1,:,:);
% u(:,:) =   Q(2,:,:)./Q(1,:,:);
% v(:,:) =   Q(3,:,:)./Q(1,:,:);
% et(:,:) =  Q(4,:,:)./Q(1,:,:);
% P(:,:) =   rho(:,:)*(gama-1).*(et(:,:) - ...
%     0.5*(u(:,:).^2 + v(:,:).^2));
% a(:,:) = sqrt( gama*P(:,:)./rho(:,:) );
%
% Vmag(:,:) = sqrt( u(:,:).^2 + v(:,:).^2 );
%
% M(:,:) = Vmag(:,:)./a(:,:);

xall = [xm';xs';xint];
yall = [ym';ys';yint];

x = [xs';xint];
y = [ys';yint];
```

```
npbs = length(x);



%**********************************************************************
****

% dx = offset;          %step size in the x direction
% dy = offset;          %step size in the y direction
%**************Computing Interpolation
Vectors*****************************
%The interpolation vectors only need to computed once for the domain.
%Interpolation vectors are to be saved and

tic
i = 0;
if new_geo == 1
    n=1;
    parfor i = 1:npbs

        xsub = x(i) + rsub*cos(theta);
        ysub = y(i) + rsub*sin(theta);

%             hold off
%             figure(3)
%             plot(xint,yint,'b.')
%             hold all
%             plot(xm,ym,'black.')
%             plot(xs,ys,'r.')
%             plot(x(i),y(i),'*')
%             plot(xsub,ysub)
%             axis equal
%             legend('Interior Nodes','Boundary Nodes', 'Shadow Nodes')
%             axis([0.0 0.7 0.0 0.2])
%             pause(0.01)

        [xo,yo,rhoo,uo,vo,Po,eto,ao,indices{i}] =
subdomain_fast(rsub,xall,yall,xsub,ysub,rhoall,uall,vall,Pall,etall,aal
l);
        [xi,yi,rhoi,ui,vi,Pi,eti,ai,indices2{i}] =
subdomain_fast(rsub,x,y,xsub,ysub,rho,u,v,P,et,a);
%
%                 if length(xo) < 5;
%
%             rsub = rsub * 1.02;
%
%
%             %dx = rsub/20;
%
%             i = i-1;
%
%             continue;
```

```
%
%         elseif length(xo) > 6;
%
%             rsub = rsub * 0.98;
%
%
%             %dy = rsub/20;
%
%             i = i-1;
%
%             continue;
%         end

        %dx_save(i)  = dx;
        %dy_save(i)  = dy;


        r{i} = sqrt( (x(i)-xo).^2 + (y(i)-yo).^2 );
        ri{i} = sqrt( (x(i)-xi).^2 + (y(i)-yi).^2 );

        ith_index(i) = find(ri{i}==0);

        mean_dist(i) = mean(nonzeros(r{i}));
        min_dist(i) = min(nonzeros(r{i}));

        x_dist = (nonzeros(abs(x(i) - xo)));
        y_dist = (nonzeros(abs(y(i) - yo)));

        x_dist = min(x_dist(x_dist >= offset));
        y_dist = min(y_dist(y_dist >= offset));

        min_dist = min(x_dist,y_dist);

%         if i <= sum(NB)

            dx(i) = offset;
            dy(i) = offset;

%         else
%             dx(i) = offset;
%             dy(i) = offset;
%
%         end

%         hold off
%         figure(3)
%         plot(xint,yint,'b.')
%         hold all
%         plot(xm,ym,'black.')
%         plot(xs,ys,'r.')
%         plot(x(i)+dx(i),y(i),'*')
```

```
%         plot(x(i),y(i)+dy(i),'*')
%         plot(x(i)-dx(i),y(i),'*')
%         plot(x(i)-dx(i)*-nx(i),y(i)-dy(i)*-ny(i),'*')
%         plot(x(i),y(i),'*')
%         plot(xsub,ysub)
%         axis equal
%         legend('Interior Nodes','Boundary Nodes', 'Shadow Nodes')
%         axis([0.0 0.7 0.0 0.2])
%         pause(0.01)
%         hold off




%         k = 1;
%         for j = 1:length(xo)
%
%             if yo(j) < y(i)+ dy && yo(j) > y(i) - dy
%
%                 xo_horiz(k) = xo(j);
%                 yo_horiz(k) = yo(j);
%                 k = k+1;
%             end
%
%             if xo(j) < x(i)+ dx && xo(j) > x(i) - dx
%
%                 xo_vert(k) = xo(j);
%                 yo_vert(k) = yo(j);
%                 k=k+1;
%             end
%
%         end


%         dx = min(nonzeros(r{i}));
%         dy = min(nonzeros(r{i}));
%
%         dx_save(i)  = dx;
%         dy_save(i)  = dy;

%         dx_new = abs( x(i) - min(xall(indices{i})) );
%         dy_new = abs( y(i) - min(yall(indices{i})) );
%
%         dx = min(nonzeros(dx_new),nonzeros(dy_new))/2;
%         dy = min(nonzeros(dx_new),nonzeros(dy_new))/2;

%         sf = 0.815*min_dist(i)/2;
%         sf3 = 0.815*min_dist(i)/2;

        indices{i};
```

```
        [RBF_s,CNs(i)] = call_RBF(xo,xo,yo,yo,rsub,sf,x(i),y(i));
        [RBF_s2,~] = call_RBF(xi,xi,yi,yi,rsub,sf3,x(i),y(i));
        [RBF_s3,~] = call_RBF(xo,xo,yo,yo,rsub,sf4,x(i),y(i));
        [RBF_d,CNd(i)] = call_RBF(xo,xo,yo,yo,rsub,sf2,x(i),y(i));
        [RBF_i,CNi(i)] = call_RBF(xi,xi,yi,yi,rsub,sf,x(i),y(i));


        [RBF_dc,~]  = call_RBF(x(i),xo,y(i),yo,rsub,sf,x(i),y(i));

        eta{i} = (RBF_dc/RBF_s);

        [RBF_E,CNE] = call_RBF(x(i) +
dx(i)/2,xo,y(i),yo,rsub,sf,x(i),y(i));
        [RBF_W,CNW] = call_RBF(x(i) -
dx(i)/2,xo,y(i),yo,rsub,sf,x(i),y(i));

        [RBF_N,CNN] = call_RBF(x(i),xo,y(i) +
dy(i)/2,yo,rsub,sf,x(i),y(i));
        [RBF_S,CNS] = call_RBF(x(i),xo,y(i) -
dy(i)/2,yo,rsub,sf,x(i),y(i));

        eta_E{i} = RBF_E/RBF_s;
        eta_W{i} = RBF_W/RBF_s;
        eta_N{i} = RBF_N/RBF_s;
        eta_S{i} = RBF_S/RBF_s;


        [RBF_E2,CNE] = call_RBF(x(i) +
dx(i),xo,y(i),yo,rsub,sf,x(i),y(i));
        [RBF_W2,CNW] = call_RBF(x(i) -
dx(i),xo,y(i),yo,rsub,sf,x(i),y(i));

        [RBF_N2,CNN] = call_RBF(x(i),xo,y(i) +
dy(i),yo,rsub,sf,x(i),y(i));
        [RBF_S2,CNS] = call_RBF(x(i),xo,y(i) -
dy(i),yo,rsub,sf,x(i),y(i));

        eta_E2{i} = RBF_E2/RBF_s;
        eta_W2{i} = RBF_W2/RBF_s;
        eta_N2{i} = RBF_N2/RBF_s;
        eta_S2{i} = RBF_S2/RBF_s;



%         [RBF_E2,CNE] = call_RBF(x(i) + 4*dx,xo,y(i),yo,sf2);
%         [RBF_W2,CNW] = call_RBF(x(i) - 4*dx,xo,y(i),yo,sf2);
%
%         [RBF_N2,CNN] = call_RBF(x(i),xo,y(i) + 4*dy,yo,sf2);
%         [RBF_S2,CNS] = call_RBF(x(i),xo,y(i) - 4*dy,yo,sf2);
```

```
        %[RBF_dx1,RBF_dy1,RBF_dx21,RBF_dy21] =
derivatives(xo,xo,yo,yo,sf);

        if i <= sum(NB)

            %Backward derivative with smooth interpolation for X
            [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] = derivatives(x(i)-
dx(i),xo,y(i),yo,sf,rsub);

            eta_dx_sb{i} = RBF_dx2/RBF_s;
            eta_dx2_sb{i} = RBF_dx22/RBF_s;

            %Backward derivative with smooth interpolation for Y
            [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i),xo,y(i)-dy(i),yo,sf,rsub);

            eta_dy_sb{i} = RBF_dy2/RBF_s;
            eta_dy2_sb{i} = RBF_dy22/RBF_s;

            %Foward derivative with smooth interpolation for X
            [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i)+dx(i),xo,y(i),yo,sf,rsub);

            eta_dx_sf{i} = RBF_dx2/RBF_s;
            eta_dx2_sf{i} = RBF_dx22/RBF_s;


            %Foward derivative with smooth interpolation for Y
            [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i),xo,y(i)+dy(i),yo,sf,rsub);

            eta_dy_sf{i} = RBF_dy2/RBF_s;
            eta_dy2_sf{i} = RBF_dy22/RBF_s;

            %[RBF_dx1,RBF_dy1,RBF_dx21,RBF_dy21] =
derivatives(xo,xo,yo,yo,sf2);

            %Backward derivative with discontinuous interpolation for X
            [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] = derivatives(x(i)-
dx(i),xo,y(i),yo,sf2,rsub);

            eta_dx_db{i} = RBF_dx2/RBF_d;
            eta_dx2_db{i} = RBF_dx22/RBF_d;

            %Backward derivative with discontinuous interpolation for Y
            [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i),xo,y(i)-dy(i),yo,sf2,rsub);
            eta_dy_db{i} = RBF_dy2/RBF_d;
```

```
            eta_dy2_db{i} = RBF_dy22/RBF_d;

            %Forward derivative with discontinuous interpolation for X
            [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i)+dx(i),xo,y(i),yo,sf2,rsub);

            eta_dx_df{i} = RBF_dx2/RBF_d;
            eta_dx2_df{i} = RBF_dx22/RBF_d;

            %Foward derivative with discontinuous interpolation for Y
            [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i),xo,y(i)+dy(i),yo,sf2,rsub);
            eta_dy_df{i} = RBF_dy2/RBF_d;
            eta_dy2_df{i} = RBF_dy22/RBF_d;

        else


        %Backward derivative with smooth interpolation for X
        [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] = derivatives(x(i)-
dx(i)/2,xo,y(i),yo,sf,rsub);

        eta_dx_sb{i} = RBF_dx2/RBF_s;
        eta_dx2_sb{i} = RBF_dx22/RBF_s;

        %Backward derivative with smooth interpolation for Y
        [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] = derivatives(x(i),xo,y(i)-
dy(i)/2,yo,sf,rsub);

        eta_dy_sb{i} = RBF_dy2/RBF_s;
        eta_dy2_sb{i} = RBF_dy22/RBF_s;

        %Foward derivative with smooth interpolation for X
        [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i)+dx(i)/2,xo,y(i),yo,sf,rsub);

        eta_dx_sf{i} = RBF_dx2/RBF_s;
        eta_dx2_sf{i} = RBF_dx22/RBF_s;


        %Foward derivative with smooth interpolation for Y
        [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i),xo,y(i)+dy(i)/2,yo,sf,rsub);

        eta_dy_sf{i} = RBF_dy2/RBF_s;
        eta_dy2_sf{i} = RBF_dy22/RBF_s;

        %[RBF_dx1,RBF_dy1,RBF_dx21,RBF_dy21] =
derivatives(xo,xo,yo,yo,sf2);
```

```
        %Backward derivative with discontinuous interpolation for X
        [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] = derivatives(x(i)-
dx(i)/2,xo,y(i),yo,sf2,rsub);

        eta_dx_db{i} = RBF_dx2/RBF_d;
        eta_dx2_db{i} = RBF_dx22/RBF_d;

        %Backward derivative with discontinuous interpolation for Y
        [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] = derivatives(x(i),xo,y(i)-
dy(i)/2,yo,sf2,rsub);
        eta_dy_db{i} = RBF_dy2/RBF_d;
        eta_dy2_db{i} = RBF_dy22/RBF_d;

        %Forward derivative with discontinuous interpolation for X
        [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i)+dx(i)/2,xo,y(i),yo,sf2,rsub);

        eta_dx_df{i} = RBF_dx2/RBF_d;
        eta_dx2_df{i} = RBF_dx22/RBF_d;

        %Foward derivative with discontinuous interpolation for Y
        [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i),xo,y(i)+dy(i)/2,yo,sf2,rsub);
        eta_dy_df{i} = RBF_dy2/RBF_d;
        eta_dy2_df{i} = RBF_dy22/RBF_d;



        end


        %Second derivative with smooth interpolation
        [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(x(i),xo,y(i),yo,sf4,rsub);
        eta_dx_c{i} = RBF_dx2/RBF_s3;
        eta_dy_c{i} = RBF_dy2/RBF_s3;
        eta_dx2{i} = RBF_dx22/RBF_s3;
        eta_dy2{i} = RBF_dy22/RBF_s3;


%         %Centered derivative with smooth interpolation
%         [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22] =
derivatives(xo,xo,yo,yo,sf);
%         eta_dx_c{i} = RBF_dx2/RBF_s;
%         eta_dy_c{i} = RBF_dy2/RBF_s;
        %


    end
end
```

```
toc
%*************************************************************************
****


L1 = 0.1;
L2 = 0.2;
L3 = 0.4;
H1 = 0.2;
H2 = H1-0.035;
dx_mesh = 0.005;
dy_mesh = 0.005;
[X,Y,testValue] = mesh(L1,L2,L3,H1,H2,dx_mesh,dy_mesh);

%************************Solver*************************************
****

count = 1;
report = 1000;
maxLaplace_old = 0.0;
sigma = 0.7;

Pres_old = sum(Pall);
Q_old = Q;
conv(:,1) = [1;1;1;1];
n  = 0;
% for n = 1:20000 %Nsteps-1

load CFD

% while conv >= 0.01
m = 1;
for n = 1:30000
    %     n = n + 1;

    %for in_iter = 1:5
    for i = 1:npbs

        QP = [Qall(:,indices{i}),zeros(4,3)];

        Q_E{i} = QP*eta_E{i}';
        Q_W{i} = QP*eta_W{i}';
        Q_N{i} = QP*eta_N{i}';
        Q_S{i} = QP*eta_S{i}';

        Q_E2{i} = QP*eta_E2{i}';
        Q_W2{i} = QP*eta_W2{i}';
        Q_N2{i} = QP*eta_N2{i}';
        Q_S2{i} = QP*eta_S2{i}';
```

```
if u(i) >= 0
    rxb(:,i) = (Q_W{i} - Q_W2{i})./( Q(:,i) - Q_W{i} );
    rxf(:,i) = (Q(:,i) - Q_W{i})./( Q_E{i} - Q(:,i) );
else
    rxb(:,i) = (Q_E{i} - Q(:,i))./( Q(:,i) - Q_W{i} );
    rxf(:,i) = (Q_E2{i} - Q_E{i})./( Q_E{i} - Q(:,i) );
end

if v(i) >= 0
    ryb(:,i) = (Q_S{i} - Q_S2{i})./( Q(:,i) - Q_S{i} );
    ryf(:,i) = (Q(:,i) - Q_S{i})./( Q_N{i} - Q(:,i) );
else
    ryb(:,i) = (Q_N{i} - Q(:,i))./( Q(:,i) - Q_S{i} );
    ryf(:,i) = (Q_N2{i} - Q_N{i})./( Q_N{i} - Q(:,i) );
end



%             if u(i) >= 0
%                 rxb1 = (Q_W{i} - Q_W2{i})./( Q(:,i) - Q_W{i} );
%                 rxb2 = ( Q(:,i) - Q_W{i} )./(Q_W{i} - Q_W2{i});
%
%                 rxf1 = (Q(:,i)-Q_W{i})./( Q_E{i} - Q(:,i) );
%                 rxf2 = (Q_E{i} - Q(:,i) )./(Q(:,i)-Q_W{i});
%
%             else
%                 rxb1 = ( Q(:,i)-Q_E)./( Q_W - Q(:,i) );
%                 rxb2 = ( Q_W - Q(:,i) )./( Q(:,i)-Q_E);
%
%                 rxf1 = (Q_E - Q_E2{i})./( Q(:,i) - Q_E{i} );
%                 rxf2 = ( Q(:,i) - Q_E{i} )./(Q_E{i} - Q_E2{i});
%             end
%
%             if v(i) >= 0
%                 ryb1 = (Q_S{i} - Q_S2{i})./( Q(:,i) - Q_S{i} );
%                 ryb2 = ( Q(:,i) - Q_S{i} )./(Q_S{i} - Q_S2{i});
%
%
%                 ryf1 = (Q(:,i) - Q_S{i} )./( Q_N{i} - Q(:,i) );
%                 ryf2 = ( Q_N{i} - Q(:,i) )./(Q(:,i) - Q_S{i} );
%
%             else
%                 ryb1 = (Q(:,i) - Q_N{i} )./( Q_S{i} - Q(:,i) );
%                 ryb2 = ( Q_S{i} - Q(:,i) )./(Q(:,i) - Q_N{i} );
%
%                 ryf1 = (Q_N{i} - Q_N2{i} )./( Q(:,i) - Q_N{i} );
%                 ryf2 = ( Q(:,i) - Q_N{i} )./(Q_N{i} - Q_N2{i});
%
%             end
```

```
        Qpos = max([Q_W{i},Qall(:,indices{i})],[],2);
        Qneg = min([Q_W{i},Qall(:,indices{i})],[],2);

%         rxb(:,i) = (Qpos-Q_W{i})./(Qneg-Q_W{i});
%         ryb(:,i) = (Qpos-Q_S{i})./(Qneg-Q_S{i});
%         rxf(:,i) = (Qpos-Q_E{i})./(Qneg-Q_E{i});
%         ryf(:,i) = (Qpos-Q_N{i})./(Qneg-Q_N{i});

        for j = 1:4


            if  Q_W{i}(j) < Q(j,i)

                rxb(j,i) = (Qpos(j) - Q_W{i}(j))/(Q(j,i)-Q_W{i}(j));


            elseif Q_W{i}(j) > Q(j,i)

                rxb(j,i) = (Qneg(j) - Q_W{i}(j))/(Q(j,i)-Q_W{i}(j));

            else
                rxb(j,i) = 1;

            end
        end

        Qpos = max([Q_E{i},Qall(:,indices{i})],[],2);
        Qneg = min([Q_E{i},Qall(:,indices{i})],[],2);



        for j = 1:4

            if Q_E{i}(j) < Q(j,i)

                rxf(j,i) = (Qpos(j) - Q_E{i}(j))/(Q(j,i)-Q_E{i}(j));


            elseif Q_E{i}(j) > Q(j,i)

                rxf(j,i) = (Qneg(j) - Q_E{i}(j))/(Q(j,i)-Q_E{i}(j));

            else

                rxf(j,i) = 1;

            end
        end
```

```
Qpos = max([Q_S{i},Qall(:,indices{i})],[],2);
Qneg = min([Q_S{i},Qall(:,indices{i})],[],2);

for j = 1:4

    if Q_S{i}(j) < Q(j,i)

        ryb(j,i) = (Qpos(j) - Q_S{i}(j))/(Q(j,i)-Q_S{i}(j));


    elseif Q_S{i}(j) > Q(j,i)

        ryb(j,i) = (Qneg(j) - Q_S{i}(j))/(Q(j,i)-Q_S{i}(j));


    else

         ryb(j,i) = 1;
    end
end

Qpos = max([Q_N{i},Qall(:,indices{i})],[],2);
Qneg = min([Q_N{i},Qall(:,indices{i})],[],2);

for j = 1:4

    if Q_N{i}(j) < Q(j,i)

        ryf(j,i) = (Qpos(j) - Q_N{i}(j))/(Q(j,i)-Q_N{i}(j));

    elseif Q_N{i}(j) > Q(j,i)

        ryf(j,i) = (Qneg(j) - Q_N{i}(j))/(Q(j,i)-Q_N{i}(j));

    else
        ryf(j,i) = 1;

    end
end


  minmod_xb{i} = max( 0,min(rxb(:,i),1.0) );
  minmod_yb{i} = max( 0,min(ryb(:,i),1.0) );

  minmod_xf{i} = max( 0,min(rxf(:,i),1.0) );
  minmod_yf{i} = max( 0,min(ryf(:,i),1.0) );

  %average_blend = mean(minmod_xb,minmod_yb,minmod_xf,minmod_yf);
```

```
%          if minmod_xb(i) < 0.1
%
% %              if xnew(m) ~= x(i)+dx || xnew(m+1) ~= x(i)-dx
%
%                  xnew(m) = x(i)-2*dx;
%                  ynew(m) = y(i);
%
%                  m = m + 1;
%
% %              end
%
%          end


         %xsub = x(i) + rsub*cos(theta);
         %ysub = y(i) + rsub*sin(theta);

         %[xo,yo,rhoo,uo,vo,Po,eto,ao] =
subdomain(xall,yall,xsub,ysub,rhoall,uall,vall,Pall,etall,aall);

         clear Fp Fm Gp Gm F G Qc

         Fp(1:4,1:length(indices{i})) = 0.0;
         Fm(1:4,1:length(indices{i})) = 0.0;
         Gp(1:4,1:length(indices{i})) = 0.0;
         Gm(1:4,1:length(indices{i})) = 0.0;
         F(1:4,1:length(indices{i})) = 0.0;
         G(1:4,1:length(indices{i})) = 0.0;
         Qc(1:length(indices{i})) = 0.0;

         for j = 1:length(indices{i})

             [Fp(:,j),Fm(:,j),Gp(:,j),Gm(:,j),F(:,j),G(:,j),Qc(j)] =
upwindedFluxes(rhoall(indices{i}(j)),uall(indices{i}(j)),vall(indices{i
}(j)),aall(indices{i}(j)),gama);

         end

         % %
         %                          hold off
         %                          figure(3)
         %                          plot(x,y,'o')
         %                          plot(xall,yall,'.')
         %                          hold all
         %                          plot(x(i),y(i),'o')
         %
plot(xall(indices{i}),yall(indices{i}),'*')
         %                          axis equal
         %                          pause(0.05)

         %          ur = eta{i}*uo';
         %
```

```
%         d2rho_dx2 = Qc*eta_dx2{i}';
%         d2rho_dy2 = Qc*eta_dy2{i}';
%
%         %d2G_dx2 = G*eta_dx2{i}';
%         %d2G_dy2 = G*eta_dy2{i}';
%
%         %                  d2Q_dx2_db =
Q(:,indices{i})*eta_dx2_db{i}';
%         %                  d2Q_dx2_df =
Q(:,indices{i})*eta_dx2_df{i}';
%         %
%         %                  d2Q_dy2_db =
Q(:,indices{i})*eta_dy2_db{i}';
%         %                  d2Q_dy2_df =
Q(:,indices{i})*eta_dy2_df{i}';
%         %
%         laplace_s(i) =  abs(d2rho_dx2) + abs(d2rho_dy2);
%         maxLaplace = max(laplace_s);

        %                  if maxLaplace > maxLaplace_old
        %
        %                      maxLaplace_old = maxLaplace;
        %
        %                  end
        %
        %                  laplace_s(i) = laplace_s(i)/maxLaplace;
        %                  laplace_s(i)


        %         dFp_dxb = Fp*eta_dx_sb{i}' + phi(i) *
(Fp*eta_dx_db{i}' - Fp*eta_dx_sb{i}');
        %         dFm_dxf = Fm*eta_dx_sb{i}' + phi(i) *
(Fm*eta_dx_db{i}' - Fm*eta_dx_sb{i}');
        %
        %         dGp_dyb = Gp*eta_dy_sb{i}' + phi(i) *
(Gp*eta_dy_db{i}' - Gp*eta_dy_sb{i}');
        %         dGm_dyf = Gm*eta_dy_sb{i}' + phi(i) *
(Gm*eta_dy_db{i}' - Gm*eta_dy_sb{i}');
        %

        %         if C_hat(i) > 0.5

%         dFp_dxb = Fp*eta_dx_db{i}';
%         dFm_dxf = Fm*eta_dx_df{i}';
%
%         dGp_dyb = Gp*eta_dy_db{i}';
%         dGm_dyf = Gm*eta_dy_df{i}';

        %             CN(i) = 1;
        %             %disp('discontinuous')
        % %         else
        %
```

```
        Fp = [Fp,zeros(4,3)];
        Fm = [Fm,zeros(4,3)];
        Gp = [Gp,zeros(4,3)];
        Gm = [Gm,zeros(4,3)];

        dFp_dxb = Fp*eta_dx_db{i}' + minmod_xb{i}.*(Fp*eta_dx_sb{i}' -
Fp*eta_dx_db{i}');
        dFm_dxf = Fm*eta_dx_df{i}' + minmod_xf{i}.*(Fm*eta_dx_sf{i}' -
Fm*eta_dx_df{i}');

        dGp_dyb = Gp*eta_dy_db{i}' + minmod_yb{i}.*(Gp*eta_dy_sb{i}' -
Gp*eta_dy_db{i}');
        dGm_dyf = Gm*eta_dy_df{i}' + minmod_yf{i}.*(Gm*eta_dy_sf{i}' -
Gm*eta_dy_df{i}');

        dt1 = CFL*dx(i)/(abs(u(i))+a(i));
        dt2 = CFL*dy(i)/(abs(v(i))+a(i));
        dt = min(dt1,dt2); %CFL/( (abs(u(i))+a(i))/dx(i) +
(abs(v(i))+a(i))/dy(i)); %min(dt1,dt2);


        Qf(:,i) = Q(:,i) - dt * (dFp_dxb + dFm_dxf + dGp_dyb +
dGm_dyf);

        rho(i) = Qf(1,i);
        u(i) =   Qf(2,i)/Qf(1,i);
        v(i) =   Qf(3,i)/Qf(1,i);
        et(i) =  Qf(4,i)/Qf(1,i);
        P(i) =   rho(i)*(gama-1)*(et(i) - ...
            0.5*(u(i)^2 + v(i)^2));
        a(i) = sqrt( gama*P(i)/rho(i) );

        Vmag(i) = sqrt( u(i)^2 + v(i)^2 );

        M(i) = Vmag(i)/a(i);

        rho(i);
        u(i);
        v(i);
        Vmag(i);

    end
    conv_L1(:,n) = sum(abs(Qf-Q),2);
    conv_max = max(conv_L1,[],2);
    conv(:,n) = conv_L1(:,n)./conv_max;
    iter_time(n) = n;
    Q = Qf;


    k=1;
```

```
    for i = 1:nume
        for j = 1:NB(i)

            if i == 1 || i == 2 || i == 3 || i == 5 || i == 6 || i == 7

                rhom(k) = rho(k);
                vm(k) = -u(k)*nx(k)/ny(k);
                um(k) = sqrt(Vmag(k)^2-vm(k)^2);
                Pm(k) = P(k);
                am(k) = a(k);
                Tm(k) = T(k);
                etm(k) = et(k);

                Vmagm(k) = sqrt( um(k)^2 + vm(k)^2 );
                Mm(k) = Vmagm(k)/am(k);


            elseif i == 4

%                 Aplus(1,1) = 0;
%                 Aplus(1,2) = 1;
%                 Aplus(1,3) = 0;
%                 Aplus(1,4) = 0;
%
%                 Aplus(2,1) = (gama-3)/2*um(k)^2 + (gama-1)/2*vm(k)^2;
%                 Aplus(2,2) = (3-gama)*um(k);
%                 Aplus(2,3) = -(gama-1)*vm(k);
%                 Aplus(2,4) = (gama-1);
%
%                 Aplus(3,1) = -um(k)*vm(k);
%                 Aplus(3,2) = vm(k);
%                 Aplus(3,3) = um(k);
%                 Aplus(3,4) = 0;
%
%                 Aplus(4,1) = -gama*um(k)*etm(k) + (gama-
1)*um(k)*Vmagm(k)^2;
%                 Aplus(4,2) = gama*etm(k) - (gama-
1)/2*(vm(k)^2+3*um(k)^2);
%                 Aplus(4,3) = -(gama-1)*um(k)*vm(k);
%                 Aplus(4,4) = gama*um(k);
%
%                 Qm = [rhom(k); rhom(k)*um(k); rhom(k)*vm(k);
rhom(k)*etm(k)];
%                 Qs = [rho(k); rho(k)*u(k); rho(k)*v(k);
rho(k)*et(k)];
%
%                 %[Fpm,Fmm,Gpm,Gmm,Fm,Gm,Qcm] =
upwindedFluxesVL(rhom(k),um(k),vm(k),am(k),gama);
%                 %[Fps,Fms,Gps,Gms,Fs,Gs,Qcs] =
upwindedFluxesVL(rhos(k),us(k),vs(k),as(k),gama);
%
%                 dt1 = CFL*offset/(um(i)+am(i));
```

```
%                   dt2 = CFL*offset/(vm(i)+am(i));
%                   dt = min(dt1,dt2);
%
%                   Qbnp1 = Qm - dt/offset * Aplus * (Qm - Qs);

%                   rhom(k) = Qbnp1(1);
%                   um(k) =   Qbnp1(2)/Qbnp1(1);
%                   vm(k) =   Qbnp1(3)/Qbnp1(1);
%                   etm(k) =  Qbnp1(4)/Qbnp1(1);
%                   Pm(k) =   rhom(k)*(gama-1)*(etm(k) - 0.5*(um(k)^2 +
vm(k)^2));
%                   am(k) = sqrt( gama*Pm(k)/rhom(k) );


                    rhom(k) = rho(k);
                    um(k) = u(k);
                    vm(k) = v(k);
                    Pm(k) = P(k);
                    etm(k) = et(k);
                    am(k) = a(k);
                    Tm(k) = T(k);
%
                    Vmagm(k) = sqrt( um(k)^2 + vm(k)^2 );
                    Mm(k) = Vmagm(k)/am(k);
                else
                    rhom(k) = rhoinit;
                    um(k) = uinit;
                    vm(k) = vinit;
                    Pm(k) = Pinit;
                    etm(k) = etinit;
                    am(k) = ainit;
                    Tm(k) = Tinit;

                    Vmagm(k) = sqrt( uinit^2 + vinit^2 );
                    Mm(k) = sqrt( uinit^2 + vinit^2 )/ainit;
                end

%                                       hold off
%                           figure(3)
%                           plot(x(k),y(k),'o')
%                           hold all
%                           plot(xall,yall,'.')
%                           %plot(x(i),y(i),'o')
%                           %plot(xall(indices{i}),yall(indices{i}),'*')
%                           axis equal
%                           pause(0.1)
                k = k+1;

            end
        end

        rhoall = [rhom;rho];
```

```
    uall = [um;u];
    vall = [vm;v];
    etall = [etm;et];
    Pall = [Pm;P];
    aall = [am;a];
    Mall = [Mm;M];


    Qall(1,:) = rhoall;
    Qall(2,:) = rhoall.*uall;
    Qall(3,:) = rhoall.*vall;
    Qall(4,:) = rhoall.*etall;



%     Pres = sum(Pall);
%     pres_norm = abs(Pres-Pres_old)/Pres_old
%     Pres_old = Pres;


    %     xlin = linspace(min(xv),max(xv),100);
    %     ylin = linspace(min(yv),max(yv),100);
    %
    %     [X,Y] = meshgrid(xlin,ylin);

%     f = scatteredInterpolant(xall,yall,Pall(:),'natural');
%     Z = f(X,Y);

    %     k=1;
    %     for i = 1:length(xall)
    %         for j = 1:length(yall)
    %
    %             u_new(i,j) = uall(k);
    %             k=k+1;
    %         end
    %     end

    %end
    if n == report*count || n == 1
        conv(:,n)
        max(Pall)
        %     figure(7);
        %     f = scatteredInterpolant(xall,yall,Pall(:),'natural');
        %     Z = f(X,Y);
        %     h=surf(X,Y,Z);
        %     view(0,90)
        %     h.EdgeColor = 'none';
        %     S1 = ['2D Advection Equation  '];
        %     %     S2 = ['C = ', num2str(sf)];
        %     %     S3 = ['   K = ', num2str(CN,3)];
        %
        %     title_string = [S1];
```

```
%     hold all
%     %     plot3(xall,yall,uall(:,n+1),'r.','MarkerSize',15)
%     title(title_string,'FontSize',14)
%     xlabel('x')
%     ylabel('y')
%     zlabel('u(x,y)')
%     %   axis([-0.1 1.0 -0.1 1.1])
%     c=colorbar;
%     w = c.Limits;
%c.Limits = [0 2];
%
%     axis equal
%     pause(0.001)
% %     M(n) = getframe(gcf);
%     hold off
count = count+1;

%     figure(4)
%     scatter(x,y,[],laplace_s','filled');
%     colorbar
%     axis equal
%     pause(0.001)

figure(5)
scatter(xall,yall,[],Mall,'filled');
colorbar
axis equal
pause(0.001)

figure(6)
scatter(xall,yall,[],rhoall,'filled');
colorbar
axis equal
pause(0.001)


figure(7)
scatter(xall,yall,[],Pall,'filled');
colorbar
axis equal
pause(0.001)


minmod_xbMat = cell2mat(minmod_xb);

figure(8)
scatter(x,y,[],minmod_xbMat(2,1:end)','filled');
colorbar
axis equal
pause(0.001)

minmod_ybMat = cell2mat(minmod_yb);
```

```
        figure(9)
        scatter(x,y,[],minmod_ybMat(2,1:end)','filled');
        colorbar
        axis equal
        pause(0.0001)
        %
        figure(10)
%         scatter(x,y,[],C_hat','filled');
%         colorbar
%         axis equal
%         pause(0.001)
%
        f = scatteredInterpolant(xall,yall,Pall(:),'natural');
        Z = f(X,Y);
        h=surf(X,Y,Z);

        figure(11)
        plot(X(:,1),Z(:,1))
        hold all
        plot(CFD(:,1),CFD(:,2))
        hold off

        figure(12)
        semilogy(iter_time,conv(1,:)')
        hold all
        semilogy(iter_time,conv(2,:)')
        semilogy(iter_time,conv(3,:)')
        semilogy(iter_time,conv(4,:)')
        hold off
        legend('mass','x','y','energy')

        n
    end

end
% movie2avi(M , 'diagonalunstable.avi');

%*********************************************************************
****
    Tall = Pall./(rhoall*R);

    figure(7);
    f = scatteredInterpolant(xall,yall,Pall(:),'natural');
    Z = f(X,Y);
    h=surf(X,Y,Z);
    view(0,90)
    h.EdgeColor = 'none';
    S1 = ['Prssure(Pa)'];
        S2 = ['C = ', num2str(sf)];
        S3 = ['   K = ', num2str(CN,3)];

    title_string = [S1];
```

```
    hold all
%         plot3(xall,yall,uall(:,n+1),'r.','MarkerSize',15)
    title(title_string,'FontSize',14)
    xlabel('x')
    ylabel('y')
    zlabel('u(x,y)')
    axis equal
    axis([0.0 0.7 0.0 0.2])
    c=colorbar;
%         w = c.Limits;
%     c.Limits = [0 2];

hold off

figure(8)
plot(X(:,1),Z(:,1))

    figure(9);
    f = scatteredInterpolant(xall,yall,rhoall(:),'natural');
    Z = f(X,Y);
    h=surf(X,Y,Z);
    view(0,90)
    h.EdgeColor = 'none';
    S1 = ['Density (kg/m^3)'];
%         S2 = ['C = ', num2str(sf)];
%         S3 = ['   K = ', num2str(CN,3)];

    title_string = [S1];
    hold all
%         plot3(xall,yall,uall(:,n+1),'r.','MarkerSize',15)
    title(title_string,'FontSize',14)
    xlabel('x')
    ylabel('y')
%     zlabel('u(x,y)')
    axis equal
    axis([0.0 0.7 0.0 0.2])
    c=colorbar;
%         w = c.Limits;
%     c.Limits = [0 2];


    figure(10);
    f = scatteredInterpolant(xall,yall,Mall(:),'linear');
    Z = f(X,Y);
    h=surf(X,Y,Z);
    view(0,90)
    h.EdgeColor = 'none';
    S1 = ['Mach Number'];
%         S2 = ['C = ', num2str(sf)];
%         S3 = ['   K = ', num2str(CN,3)];

    title_string = [S1];
```

```
    hold all
    title(title_string,'FontSize',14)
    xlabel('x')
    ylabel('y')
%     zlabel('u(x,y)')
    axis equal
    axis([0.0 0.7 0.0 0.2])
    c=colorbar;
%         w = c.Limits;
%     c.Limits = [0 2];

    figure(11);
    f = scatteredInterpolant(xall,yall,Tall(:),'natural');
    Z = f(X,Y);
    h=surf(X,Y,Z);
    view(0,90)
    h.EdgeColor = 'none';
    S1 = ['Mach Number'];
%         S2 = ['C = ', num2str(sf)];
%         S3 = ['   K = ', num2str(CN,3)];
%
    title_string = [S1];
    hold all
    title(title_string,'FontSize',14)
    xlabel('x')
    ylabel('y')
%     zlabel('u(x,y)')
    axis equal
    axis([0.0 0.7 0.0 0.2])
    c=colorbar;
%         w = c.Limits;
%     c.Limits = [0 2];
```

APPENDIX B

LOCALIZED BLENDED RBF COLLOCATION AUSM/AUSM+ SOLVER

```
%Author: Michael F. Harris
clc
close all
clear all

%*****************Setting up Geometry and Point
Cloud**********************

xv = [0.0, 0.1, 0.300,    0.7, 0.7, 0.3, 0.1, 0.0, 0.0];
yv = [0.0, 0.0, 0.035,  0.035, 0.2, 0.2, 0.2, 0.2, 0.0];

L = 0.3;
H =0.2;

NB = [40,80,160,66,160,80,40,80];


NBtot = sum(NB);


% Number of edges that make up the domain
nume = 8;


% Boundary condition type  1 = Wall, 2 = Inlet, 3 = Outlet
BC = [1,1,1,3,1,2];

% Offset for the shadow numbers
offset  = 5e-4;
offset2 = 5e-4;
% Number of interior points
NP = 50;

% Plotting the domain
figure(1)
plot(xv,yv)
title('Domain')
xlabel('x')
ylabel('y')
axis equal

%Fluid Nodes
layers = 1;
growth = 1.0;



[xb,yb,xm,ym,tx,ty,nx,ny,xs,ys,xg,yg,~,~] =
preprocessing(xv,yv,NB,offset,NP,layers,growth,-1);
```

```
rx = zeros(1,NBtot);
ry = zeros(1,NBtot);
rmag = zeros(1,NBtot);

for i = 1:NBtot

    rx(i) = ( xb(i+1)-xb(i) );
    ry(i) = ( yb(i+1)-yb(i) );

    rmag(i) = sqrt(rx(i)^2+ry(i)^2);
end

dndx = rx./rmag;
dndy = ry./rmag;



figure(2)
%Fluids Nodes
plot(xv,yv)
hold all
plot(xb,yb,'+')
plot(xm,ym,'.','MarkerSize',15)
plot(xs,ys,'.','MarkerSize',15)
axis equal


xpoly = zeros(1,NBtot);
ypoly = zeros(1,NBtot);


buffer = offset;


for i = 1:length(xs(:,layers))

    xpoly(i) = xs(i,layers) - nx(i)*buffer;
    ypoly(i) = ys(i,layers) - ny(i)*buffer;

end



[Xint,Yint] = meshgrid(0.00125:0.0025:0.3,0.00125:0.0025:0.35);

[Xint2,Yint2] = meshgrid(0.30125:0.0025:0.7,0.03625:0.0025:0.3);
```

```
[in,on] = inpolygon(Xint,Yint,xpoly,ypoly);
[in2,on2] = inpolygon(Xint2,Yint2,xpoly,ypoly);


xint = [Xint(in&~on);Xint2(in2&~on2)];
yint = [Yint(in&~on);Yint2(in2&~on2)];







%
%  for i = 1:length(xint)
%     xint(i) = xint(i) + 0.00025*(-1)^i;
%     yint(i) = yint(i) + 0.00025*(-1)^(i+1);
% end

hold off
figure(3)
%Fluid Nodes
plot(xv,yv)
hold all
plot(xb,yb,'+')
plot(xm,ym,'.','MarkerSize',15)
plot(xs,ys,'.','MarkerSize',15)
plot(xint,yint,'.','MarkerSize',5)
axis equal
% axis([0.0 0.7 0.0 0.2])

pause(0.001)

%**********************Input
values**************************************

CFL  = 0.001;
Mach = 2.0;
Pinit = 100e3;
Tinit = 300;
MW = 28;
gama = 1.4;
R = 287.05;
mu = 1.846e-5;
kcond = 0.02624;
rhoinit = Pinit/(R*Tinit);
ainit = sqrt(gama*R*Tinit);
uinit = Mach*ainit;
vinit = 0.0;
etinit = (R*Tinit)/(gama-1) + 0.5*(uinit^2 + vinit^2);


%Subdomain Parameters
```

```
theta = [0:0.01:2*pi,0]; %Angles in radians for circular subdomain
rsub = 0.015;             %Radius of the circular subdomain
rsub_mls = 0.01;

%Shape paramters
sf  = 0.01;            %shape parameter for smooth RBF interpolation
sf2 = 0.001;            %shape parameter for discontinuous
interpolation
sf3 = 5;
sf4 = 0.00001;
sfi = 1e-3;


% Applying intial condtion to domain and applying boundary conditions
Nsteps=1;
[rhom,rhos,rhoint,rho,rhoall] =
intializeRho(rhoinit,Nsteps,nume,NB,length(xint),nume,layers);
[um,us,uint,u,uall] =
intializeU(uinit,Nsteps,nume,NB,length(xint),nume,layers);
[vm,vs,vint,v,vall] =
intializeV(vinit,Nsteps,nume,NB,length(xint),nume,layers);
[Pm,Ps,Pint,P,Pall] =
intializeP(Pinit,Nsteps,nume,NB,length(xint),nume,layers);
[etm,ets,etint,et,etall] =
intializeEt(etinit,Nsteps,nume,NB,length(xint),nume,layers);
[am,as,aint,a,aall] =
intializeSoS(ainit,Nsteps,nume,NB,length(xint),nume,layers);


Vmagm = sqrt(um.^2+vm.^2);
Vmags = sqrt(us.^2+vs.^2);
Vmagint = sqrt(uint.^2+vint.^2);
Vmagall = sqrt(uall.^2+vall.^2);

Mm = Vmagm./am;
Ms = Vmags./as;
Mint = Vmagint./aint;
Mall = Vmagall./aall;

M = [Ms;Mint];

Tm = Pm./(R*rhom);
Ts = Ps./(R*rhos);
Tint = Pint./(R*rhoint);
Tall = Pall./(rhoall*R);

mu_ref = 1.716e-5;
Tref = 273.15;
Sconst = 110.4;
mu_s =  mu_ref*((Tall/Tref).^(3/2)).*( (Tref+Sconst)./(Tall+Sconst) );

T = [Ts;Tint];
```

```
NQ = length(rho);
NQall = length(rhoall);
Q = zeros(4,NQ);
Qall = zeros(4,NQall);

Q(1,:) = rho;
Q(2,:) = rho.*u;
Q(3,:) = rho.*v;
Q(4,:) = rho.*et;

Qall(1,:) = rhoall;
Qall(2,:) = rhoall.*uall;
Qall(3,:) = rhoall.*vall;
Qall(4,:) = rhoall.*etall;


Nsall = sum(NB)*layers;
xsall = zeros(1,Nsall);
ysall = zeros(1,Nsall);


k = 1;
for i = 1:layers
    for j = 1:NBtot

        xsall(k) = xs(j,i);
        ysall(k) = ys(j,i);
        k = k+1;

    end
end



xall = [xm';xsall';xint];
yall = [ym';ysall';yint];

x = [xsall';xint];
y = [ysall';yint];


%**************Computing Interpolation
Vectors*****************************
%The interpolation vectors only need to computed once for the domain.
%Interpolation vectors are to be saved and
npbs = length(x);
dx = zeros(npbs,1);
dy = zeros(npbs,1);
```

```
indices = cell(npbs,1);
indices2 = cell(npbs,1);
indicesLS = cell(npbs,1);
r = cell(npbs,1);

eta_err_S = cell(npbs,1);
eta_err_D = cell(npbs,1);

eta_E = cell(npbs,1);
eta_W = cell(npbs,1);
eta_N = cell(npbs,1);
eta_S = cell(npbs,1);

eta_EE = cell(npbs,1);
eta_WW = cell(npbs,1);
eta_NN = cell(npbs,1);
eta_SS = cell(npbs,1);

eta_EEE = cell(npbs,1);
eta_WWW = cell(npbs,1);
eta_NNN = cell(npbs,1);
eta_SSS = cell(npbs,1);

eta_Es = cell(npbs,1);
eta_Ws = cell(npbs,1);
eta_Ns = cell(npbs,1);
eta_Ss = cell(npbs,1);

eta_Ed = cell(npbs,1);
eta_Wd = cell(npbs,1);
eta_Nd = cell(npbs,1);
eta_Sd = cell(npbs,1);

ith_index = zeros(1,npbs);

CNs = zeros(npbs,1);
CNd = zeros(npbs,1);


parfor i = 1:npbs

    xsub = x(i) + rsub*cos(theta);
    ysub = y(i) + rsub*sin(theta);

    xsub_mls = x(i) + rsub_mls*cos(theta);
    ysub_mls = y(i) + rsub_mls*sin(theta);

    [xo,yo,rhoo,uo,vo,Po,eto,ao,indices{i}] =
subdomain_fast(rsub,xall,yall,xsub,ysub,rhoall,uall,vall,Pall,etall,aal
l);
```

```
    [xls,yls,rhols,uls,vls,Pls,etls,als,indicesLS{i}] =
subdomain_fast(rsub_mls,xall,yall,xsub_mls,ysub_mls,rhoall,uall,vall,Pa
ll,etall,aall);

    r{i} = sqrt( (x(i)-xo).^2 + (y(i)-yo).^2 );

    ith_index(i) = find(r{i}==0);

    Rmax = max(r{i});

    xo_e = xo;
    yo_e = yo;

    xo_e(ith_index(i))=[];
    yo_e(ith_index(i))=[];

    x_dist = (nonzeros(abs(x(i) - xo)));
    y_dist = (nonzeros(abs(y(i) - yo)));

    x_dist = max(x_dist(x_dist >= offset));
    y_dist = max(y_dist(y_dist >= offset));

    mean_dist = mean(nonzeros(r{i}));
    min_dist = min(x_dist,y_dist);


    dx(i) = offset;
    dy(i) = offset;



    %
    %             hold off
    %             figure(3)
    %             plot(xint,yint,'b.')
    %             hold all
    %             plot(xm,ym,'black.')
    %             plot(xs,ys,'r.')
    %             plot(xo,yo,'*')
    %             plot(xls,yls,'*')
    %             plot(x(i)+dx(i),y(i),'*')
    %             plot(x(i),y(i)+dy(i),'*')
    %             plot(x(i)-dx(i),y(i),'*')
    %             plot(x(i),y(i)-dy(i),'*')
    %             plot(x(i),y(i),'o')
    %             plot(xsub_mls,ysub_mls)
    %             axis equal
    %             legend('Interior Nodes','Boundary Nodes', 'Shadow
Nodes')
    %             axis([0.0 0.75 0.0 0.2])
    %             pause(0.001)
```

```
    %             hold off


    xstencil_1 = [x(i)+dx(i)/2,x(i)-dx(i)/2,x(i),x(i)];
    ystencil_1 = [y(i),y(i),y(i)+dy(i)/2,y(i)-dy(i)/2];

    [eta_E{i},eta_W{i},eta_N{i},eta_S{i},~,~] =
call_RBF_comp(xo,xo,yo,yo,sf,xstencil_1,ystencil_1,offset,x(i),y(i),x_d
ist,y_dist);

    xstencil_1 = [x(i)+dx(i),x(i)-dx(i),x(i),x(i)];
    ystencil_1 = [y(i),y(i),y(i)+dy(i),y(i)-dy(i)];

    [eta_EE{i},eta_WW{i},eta_NN{i},eta_SS{i},~,~] =
call_RBF_comp(xo,xo,yo,yo,sf,xstencil_1,ystencil_1,offset,x(i),y(i),x_d
ist,y_dist);

    xstencil_1 = [x(i)+ dx(i),x(i)- dx(i),x(i),x(i)];
    ystencil_1 = [y(i),y(i),y(i)+ dy(i),y(i)- dy(i)];

    [eta_EEE{i},eta_WWW{i},eta_NNN{i},eta_SSS{i},~,~] =
call_RBF_comp(xo,xo,yo,yo,sf,xstencil_1,ystencil_1,offset,x(i),y(i),x_d
ist,y_dist);



    xstencil_1 = [x(i)+dx(i)/2,x(i)-dx(i)/2,x(i),x(i)];
    ystencil_1 = [y(i),y(i),y(i)+dy(i)/2,y(i)-dy(i)/2];


    xstencil_2 = [x(i)+dx(i),x(i)-dx(i),x(i),x(i)];
    ystencil_2 = [y(i),y(i),y(i)+dy(i),y(i)-dy(i)];

    [eta_Es{i},eta_Ws{i},eta_Ns{i},eta_Ss{i},~,CNs(i)] =
call_RBF_comp(xo,xo,yo,yo,sf,xstencil_1,ystencil_1,offset,x(i),y(i),x_d
ist,y_dist);
    [eta_Ed{i},eta_Wd{i},eta_Nd{i},eta_Sd{i},~,CNd(i)] =
call_RBF_comp(xo,xo,yo,yo,sf2,xstencil_1,ystencil_1,offset,x(i),y(i),x_
dist,y_dist);
    [eta_EEs{i},eta_WWs{i},eta_NNs{i},eta_SSs{i},~,~] =
call_RBF_comp(xo,xo,yo,yo,sf,xstencil_2,ystencil_2,offset,x(i),y(i),x_d
ist,y_dist);
    [eta_EEd{i},eta_WWd{i},eta_NNd{i},eta_SSd{i},~,~] =
call_RBF_comp(xo,xo,yo,yo,sf2,xstencil_2,ystencil_2,offset,x(i),y(i),x_
dist,y_dist);

    [RBFpoly,~] = call_RBFwPoly(xo,xo,yo,yo,offset,sf,x(i),y(i));
    [RBFpoly_d,~] = call_RBFwPoly(xo,xo,yo,yo,offset,sf2,x(i),y(i));

    %Second derivative with smooth interpolation
```

```
    [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22,RBF_dx33,RBF_dy33,~,~] =
derivativesPoly(x(i)+0.5*dx(i),xo,y(i),yo,sf,offset,x(i),y(i));
    eta_dx_cf{i} = RBF_dx2/RBFpoly;

    eta_dx2_cf{i} = RBF_dx22/RBFpoly;

    eta_dx3_cf{i} = RBF_dx33/RBFpoly;


    [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22,RBF_dx33,RBF_dy33,~,~] =
derivativesPoly(x(i)+0.5*dx(i),xo,y(i),yo,sf2,offset,x(i),y(i));

    eta_dx_cdf{i} = RBF_dx2/RBFpoly_d;

    eta_dx2_cdf{i} = RBF_dx22/RBFpoly_d;

    eta_dx3_cdf{i} = RBF_dx33/RBFpoly_d;

    %Second derivative with smooth interpolation
    [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22,RBF_dx33,RBF_dy33,~,~] =
derivativesPoly(x(i)-0.5*dx(i),xo,y(i),yo,sf,offset,x(i),y(i));
    eta_dx_cb{i} = RBF_dx2/RBFpoly;

    eta_dx2_cb{i} = RBF_dx22/RBFpoly;

    eta_dx3_cb{i} = RBF_dx33/RBFpoly;

    [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22,RBF_dx33,RBF_dy33,~,~] =
derivativesPoly(x(i)-0.5*dx(i),xo,y(i),yo,sf2,offset,x(i),y(i));

    eta_dx_cdb{i} = RBF_dx2/RBFpoly_d;

    eta_dx2_cdb{i} = RBF_dx22/RBFpoly_d;

    eta_dx3_cdb{i} = RBF_dx33/RBFpoly_d;




    %Second derivative with smooth interpolation
    [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22,RBF_dx33,RBF_dy33,~,~] =
derivativesPoly(x(i),xo,y(i)+0.5*dy(i),yo,sf,offset,x(i),y(i));
    eta_dy_cf{i} = RBF_dx2/RBFpoly;

    eta_dy2_cf{i} = RBF_dx22/RBFpoly;

    eta_dy3_cf{i} = RBF_dx33/RBFpoly;
```

```
    [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22,RBF_dx33,RBF_dy33,~,~] =
derivativesPoly(x(i),xo,y(i)+0.5*dy(i),yo,sf2,offset,x(i),y(i));

    eta_dy_cdf{i} = RBF_dx2/RBFpoly_d;

    eta_dy2_cdf{i} = RBF_dx22/RBFpoly_d;

    eta_dy3_cdf{i} = RBF_dx33/RBFpoly_d;

    %Second derivative with smooth interpolation
    [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22,RBF_dx33,RBF_dy33,~,~] =
derivativesPoly(x(i),xo,y(i)-0.5*dy(i),yo,sf,offset,x(i),y(i));
    eta_dy_cb{i} = RBF_dx2/RBFpoly;

    eta_dy2_cb{i} = RBF_dx22/RBFpoly;

    eta_dy3_cb{i} = RBF_dx33/RBFpoly;

    [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22,RBF_dx33,RBF_dy33,~,~] =
derivativesPoly(x(i),xo,y(i)-0.5*dy(i),yo,sf2,offset,x(i),y(i));

    eta_dy_cdb{i} = RBF_dx2/RBFpoly_d;

    eta_dy2_cdb{i} = RBF_dx22/RBFpoly_d;

    eta_dy3_cdb{i} = RBF_dx33/RBFpoly_d;


end


L1 = 0.1;
L2 = 0.2;
L3 = 0.4;
H1 = 0.2;
H2 = H1-0.035;
dx_mesh = 0.0025;
dy_mesh = 0.0025;
[X,Y,testValue] = mesh(L1,L2,L3,H1,H2,dx_mesh,dy_mesh);
load CFD

count = 1;
report = 50;


cfl = CFL;

txx = zeros(length(xall),1);
tyy = zeros(length(xall),1);
txy = zeros(length(xall),1);
```

```
qx = zeros(length(xall),1);
qy = zeros(length(xall),1);

Qf = Q;


disp('Pre-processing completed')
disp('Solving...')

minmod_xb = cell(1,npbs);
minmod_yb = cell(1,npbs);
minmod_xf = cell(1,npbs);
minmod_yf = cell(1,npbs);


for n = 1:60000

    if cfl >= 1.0

        cfl = 1.0;
    else
        cfl = CFL*2^((n-1)/4);
    end

    for i = 1:npbs

        dt  = cfl/( (abs(u(i))+a(i))/dx(i) + (abs(v(i)) + a(i))/dy(i)
);

        QP = [Qall(:,indices{i}),zeros(4,3)];

        Q_E = QP*eta_E{i}';
        Q_W = QP*eta_W{i}';
        Q_N = QP*eta_N{i}';
        Q_S = QP*eta_S{i}';

        Q_EE = QP*eta_EE{i}';
        Q_WW = QP*eta_WW{i}';
        Q_NN = QP*eta_NN{i}';
        Q_SS = QP*eta_SS{i}';

        Q_EEE = QP*eta_EEE{i}';
        Q_WWW = QP*eta_WWW{i}';
        Q_NNN = QP*eta_NNN{i}';
        Q_SSS = QP*eta_SSS{i}';


        if u(i) >= 0
            rxb1 = (Q_W - Q_WW)./( Q(:,i) - Q_W );
```

```
    rxb2 = ( Q(:,i) - Q_W )./(Q_W - Q_WW);

    rxf1 = (Q(:,i)-Q_W)./( Q_E - Q(:,i) );
    rxf2 = ( Q_E - Q(:,i) )./(Q(:,i)-Q_W);
else
    rxb1 = (Q_E - Q(:,i))./(Q(:,i)- Q_W);
    rxb2 = ( Q_W - Q(:,i) )./( Q(:,i)-Q_E);

    rxf1 = (Q_EE - Q_E )./( Q_E - Q(:,i));
    rxf2 = ( Q(:,i) - Q_E )./(Q_E - Q_EE);
end

if v(i) >= 0
    ryb1 = (Q_S - Q_SS)./( Q(:,i) - Q_S );
    ryb2 = ( Q(:,i) - Q_S )./(Q_S - Q_SS);

    ryf1 = (Q(:,i) - Q_S )./( Q_N - Q(:,i) );
    ryf2 = ( Q_N - Q(:,i) )./(Q(:,i) - Q_S );

else
    ryb1 = (Q_N - Q(:,i))./( Q(:,i) - Q_S );
    ryb2 = ( Q_S - Q(:,i) )./(Q(:,i) - Q_N );

    ryf1 = (Q_NN - Q_N)./( Q_N - Q(:,i) );
    ryf2 = ( Q(:,i) - Q_N )./(Q_N - Q_NN);
end


minmod_xb{i}  = max( 0,min( rxb1,1.0 ));
minmod_yb{i}  = max( 0,min( ryb1,1.0 ));
minmod_xf{i}  = max( 0,min( rxf1,1.0 ));
minmod_yf{i}  = max( 0,min( ryf1,1.0 ));

%         rxb1(isnan(rxb1)) = 1.0;
%         rxf1(isnan(rxf1)) = 1.0;
%         ryb1(isnan(ryb1)) = 1.0;
%         ryf1(isnan(ryf1)) = 1.0;
%
%         rxb1(isinf(rxb1)) = 1.0;
%         rxf1(isinf(rxf1)) = 1.0;
%         ryb1(isinf(ryb1)) = 1.0;
%         ryf1(isinf(ryf1)) = 1.0;
%
%         minmod_xb{i} = ((rxb1 <= 1).*rxb1);
%         minmod_xb{i} = ((minmod_xb{i}>0).*minmod_xb{i});
%
%         minmod_yb{i} = ((ryb1 <= 1).*ryb1);
%         minmod_yb{i} = ((minmod_yb{i}>0).*minmod_yb{i});
%
%         minmod_xf{i} = ((rxf1 <= 1).*rxf1);
%         minmod_xf{i} = ((minmod_xf{i}>0).*minmod_xf{i});
%
```

```
        %         minmod_yf{i} = ((ryf1 <= 1).*ryf1);
        %         minmod_yf{i} = ((minmod_yf{i}>0).*minmod_yf{i});

        %         dEv_dx_mls = [0; dtxx_dx; dtxy_dx; dEv4_dx];
        %         dFv_dy_mls = [0; dtxy_dy; dtyy_dy; dFv4_dy];
        %
        %         QRK = [Q(:,i),zeros(4,3)];
        %         QallRK = [Qall,zeros(4,3)];

        dQ_dxf = QP*eta_dx_cf{i}';
        d2Q_dx2f = QP*eta_dx2_cf{i}';
        d3Q_dx3f = QP*eta_dx3_cf{i}';

        dQ_dxb = QP*eta_dx_cb{i}';
        d2Q_dx2b = QP*eta_dx2_cb{i}';
        d3Q_dx3b = QP*eta_dx3_cb{i}';

        dQ_dx_df = QP*eta_dx_cdf{i}';
        d2Q_dx2_df = QP*eta_dx2_cdf{i}';
        d3Q_dx3_df = QP*eta_dx3_cdf{i}';

        dQ_dx_db = QP*eta_dx_cdb{i}';
        d2Q_dx2_db = QP*eta_dx2_cdb{i}';
        d3Q_dx3_db = QP*eta_dx3_cdb{i}';

        dQ_dyf = QP*eta_dy_cf{i}';
        d2Q_dy2f = QP*eta_dy2_cf{i}';
        d3Q_dy3f = QP*eta_dy3_cf{i}';

        dQ_dyb = QP*eta_dy_cb{i}';
        d2Q_dy2b = QP*eta_dy2_cb{i}';
        d3Q_dy3b = QP*eta_dy3_cb{i}';

        dQ_dy_df = QP*eta_dy_cdf{i}';
        d2Q_dy2_df = QP*eta_dy2_cdf{i}';
        d3Q_dy3_df = QP*eta_dy3_cdf{i}';

        dQ_dy_db = QP*eta_dy_cdb{i}';
        d2Q_dy2_db = QP*eta_dy2_cdb{i}';
        d3Q_dy3_db = QP*eta_dy3_cdb{i}';

        %         beta_xs = sum(dQ_dx.^2,2).^0.5 +
sum(d2Q_dx2.^2,2).^0.5 + sum(d3Q_dx3.^2,2).^(0.5);
        %         beta_xd = sum(dQ_dx_d.^2,2).^0.5 +
sum(d2Q_dx2_d.^2,2).^0.5 + sum(d3Q_dx3_d.^2,2).^(0.5);
        %
        %         beta_ys = sum( dQ_dy.^2,2).^0.5 +
sum(d2Q_dy2.^2,2).^0.5 + sum(d3Q_dy3.^2,2).^(0.5);
        %         beta_yd = sum( dQ_dy_d.^2,2).^0.5 +
sum(d2Q_dy2_d.^2,2).^0.5 + sum(d3Q_dy3_d.^2,2).^(0.5);
```

```
        %         dQxs = 0.5*(dQ_dxf + dQ_dxb);
        %         dQ2xs = 0.5*(d2Q_dx2f + d2Q_dx2b);
        %         dQ3xs = 0.5*(d3Q_dx3f + d3Q_dx3b);
        %
        %         dQxd = 0.5*(dQ_dx_df + dQ_dx_db);
        %         dQ2xd = 0.5*(d2Q_dx2_df + d2Q_dx2_db);
        %         dQ3xd = 0.5*(d3Q_dx3_df + d3Q_dx3_db);
        %
        %         dQys = 0.5*(dQ_dyf + dQ_dyb);
        %         dQ2ys = 0.5*(d2Q_dy2f + d2Q_dy2b);
        %         dQ3ys = 0.5*(d3Q_dy3f + d3Q_dy3b);
        %
        %         dQyd = 0.5*(dQ_dy_df + dQ_dy_db);
        %         dQ2yd = 0.5*(d2Q_dy2_df + d2Q_dy2_db);
        %         dQ3yd = 0.5*(d3Q_dy3_df + d3Q_dy3_db);
        %
        %         beta_xs = ( dx(i)^2*dQxs.^2 + dx(i)^4*dQ2xs.^2 +
dx(i)^6*dQ3xs.^2);
        %         beta_xd = ( dx(i)^2*dQxd .^2 + dx(i)^4*dQ2xd.^2 +
dx(i)^6*dQ3xd.^2);
        %
        %         beta_ys = ( dy(i)^2*dQys.^2 + dy(i)^4*dQ2ys.^2 +
dy(i)^6*dQ3ys.^2);
        %         beta_yd = ( dy(i)^2*dQyd.^2 + dy(i)^4*dQ2yd.^2 +
dy(i)^6*dQ3yd.^2);
        %
        %         wxs = (0.9)./(beta_xs + 1e-6).^2;
        %         wxd = (0.1)./(beta_xd + 1e-6).^2;
        %
        %         wys = (0.9)./(beta_ys + 1e-6).^2;
        %         wyd = (0.1)./(beta_yd + 1e-6).^2;
        %
        %         Wxs{i} = wxs./(wxs+wxd);
        %         Wxd{i} = wxd./(wxs+wxd);
        %
        %         Wys{i} = wys./(wys+wyd);
        %         Wyd{i} = wyd./(wys+wyd);

        QE = (QP*eta_Ed{i}').*(1-minmod_xf{i}) +
minmod_xf{i}.*(QP*eta_Es{i}') ;
        QN = (QP*eta_Nd{i}').*(1-minmod_yf{i}) +
minmod_yf{i}.*(QP*eta_Ns{i}') ;
        QW = (QP*eta_Wd{i}').*(1-minmod_xb{i}) +
minmod_xb{i}.*(QP*eta_Ws{i}') ;
        QS = (QP*eta_Sd{i}').*(1-minmod_yb{i}) +
minmod_yb{i}.*(QP*eta_Ss{i}') ;

        QE = (QP*eta_Es{i}') ;
        QN = (QP*eta_Ns{i}') ;
        QW = (QP*eta_Ws{i}') ;
        QS = (QP*eta_Ss{i}') ;

        QEd = (QP*eta_Ed{i}') ;
```

```
        QNd = (QP*eta_Nd{i}') ;
        QWd = (QP*eta_Wd{i}') ;
        QSd = (QP*eta_Sd{i}') ;



        QEE = QP*eta_EEd{i}'.*(1 - minmod_xf{i}) +
minmod_xf{i}.*(QP*eta_EEs{i}') ;
        QNN = QP*eta_NNd{i}'.*(1 - minmod_yf{i}) +
minmod_yf{i}.*(QP*eta_NNs{i}') ;
        QWW = QP*eta_WWd{i}'.*(1 - minmod_xb{i}) +
minmod_xb{i}.*(QP*eta_WWs{i}') ;
        QSS = QP*eta_SSd{i}'.*(1 - minmod_yb{i}) +
minmod_yb{i}.*(QP*eta_SSs{i}') ;

        %         QEE = QP*eta_EEd{i}' + minmod_xf{i}.*(QP*eta_EEs{i}'
- QP*eta_EEd{i}');
        %         QNN = QP*eta_NNd{i}' + minmod_yf{i}.*(QP*eta_NNs{i}'
- QP*eta_NNd{i}');
        %         QWW = QP*eta_WWd{i}' + minmod_xb{i}.*(QP*eta_WWs{i}'
- QP*eta_WWd{i}');
        %         QSS = QP*eta_SSd{i}' + minmod_yb{i}.*(QP*eta_SSs{i}'
- QP*eta_SSd{i}');



        rho_e = QE(1);
        u_e =   QE(2)/QE(1);
        v_e =   QE(3)/QE(1);
        et_e =  QE(4)/QE(1);
        P_e =   rho_e*(gama-1)*(et_e - 0.5*(u_e^2 + v_e^2));
        a_e = sqrt( gama*P_e/rho_e );
        T_e = P_e/(R*rho_e);
        MuE =  mu_ref*(T_e/Tref)^(3/2)*( (Tref+Sconst)/(T_e+Sconst) );

        Mach_e = u_e/sqrt(gama*R*T_e);

        if P_e < 0
            disp('pressure negative')
        end

        rho_n = QN(1);
        u_n =   QN(2)/QN(1);
        v_n =   QN(3)/QN(1);
        et_n =  QN(4)/QN(1);
        P_n =   rho_n*(gama-1)*(et_n - 0.5*(u_n^2 + v_n^2));
        a_n = sqrt( gama*P_n/rho_n );
        T_n = P_n/(R*rho_n);
        MuN =  mu_ref*(T_n/Tref)^(3/2)*( (Tref+Sconst)/(T_n+Sconst) );

        Mach_n = v_n/sqrt(gama*R*T_n);
```

```
if P_n < 0
    disp('pressure negative')
end

rho_w = QW(1);
u_w =   QW(2)/QW(1);
v_w =   QW(3)/QW(1);
et_w =  QW(4)/QW(1);
P_w =   rho_w*(gama-1)*(et_w - 0.5*(u_w^2 + v_w^2));
a_w = sqrt( gama*P_w/rho_w );
T_w = P_w/(R*rho_w);
MuW =  mu_ref*(T_w/Tref)^(3/2)*( (Tref+Sconst)/(T_w+Sconst) );

Mach_w = u_w/sqrt(gama*R*T_w);

if P_w < 0
    disp('pressure negative')
end

rho_s = QS(1);
u_s =   QS(2)/QS(1);
v_s =   QS(3)/QS(1);
et_s =  QS(4)/QS(1);
P_s =   rho_s*(gama-1)*(et_s - 0.5*(u_s^2 + v_s^2));
a_s = sqrt( gama*P_s/rho_s );
T_s = P_s/(R*rho_s);
MuS =  mu_ref*(T_s/Tref)^(3/2)*( (Tref+Sconst)/(T_s+Sconst) );

Mach_s = v_s/sqrt(gama*R*T_s);


if P_s < 0
    disp('pressure negative')
end


%Low shape parameter interpolation
rho_ed = QEd(1);
u_ed =   QEd(2)/QEd(1);
v_ed =   QEd(3)/QEd(1);
et_ed =  QEd(4)/QEd(1);
P_ed =   rho_ed*(gama-1)*(et_ed - 0.5*(u_ed^2 + v_ed^2));
a_ed = sqrt( gama*P_ed/rho_ed );
T_ed = P_ed/(R*rho_ed);
MuE =  mu_ref*(T_e/Tref)^(3/2)*( (Tref+Sconst)/(T_e+Sconst) );

Mach_ed = u_ed/sqrt(gama*R*T_ed);

if P_ed < 0
    disp('pressure negative')
end
```

```
rho_nd = QNd(1);
u_nd =   QNd(2)/QNd(1);
v_nd =   QNd(3)/QNd(1);
et_nd =  QNd(4)/QNd(1);
P_nd =   rho_nd*(gama-1)*(et_nd - 0.5*(u_nd^2 + v_nd^2));
a_nd = sqrt( gama*P_nd/rho_nd );
T_nd = P_nd/(R*rho_nd);
MuN =  mu_ref*(T_n/Tref)^(3/2)*( (Tref+Sconst)/(T_n+Sconst) );

Mach_n = v_nd/sqrt(gama*R*T_nd);

if P_nd < 0
    disp('pressure negative')
end

rho_wd = QWd(1);
u_wd =   QWd(2)/QWd(1);
v_wd =   QWd(3)/QWd(1);
et_wd =  QWd(4)/QWd(1);
P_wd =   rho_wd*(gama-1)*(et_wd - 0.5*(u_wd^2 + v_wd^2));
a_wd = sqrt( gama*P_wd/rho_wd );
T_wd = P_wd/(R*rho_wd);
MuW =  mu_ref*(T_w/Tref)^(3/2)*( (Tref+Sconst)/(T_w+Sconst) );

Mach_wd = u_wd/sqrt(gama*R*T_wd);

if P_wd < 0
    disp('pressure negative')
end

rho_sd = QSd(1);
u_sd =   QSd(2)/QSd(1);
v_sd =   QSd(3)/QSd(1);
et_sd =  QSd(4)/QSd(1);
P_sd =   rho_sd*(gama-1)*(et_sd - 0.5*(u_sd^2 + v_sd^2));
a_sd = sqrt( gama*P_sd/rho_sd );
T_sd = P_sd/(R*rho_sd);
MuS =  mu_ref*(T_s/Tref)^(3/2)*( (Tref+Sconst)/(T_s+Sconst) );

Mach_sd = v_sd/sqrt(gama*R*T_sd);


if P_sd < 0
    disp('pressure negative')
end


rho_ee = QEE(1);
u_ee =   QEE(2)/QEE(1);
v_ee =   QEE(3)/QEE(1);
```

```
        et_ee =  QEE(4)/QEE(1);
        P_ee =   rho_ee*(gama-1)*(et_ee - 0.5*(u_ee^2 + v_ee^2));
        a_ee = sqrt( gama*P_ee/rho_ee );
        T_ee = P_ee/(R*rho_ee);
        MuEE =  mu_ref*(T_ee/Tref)^(3/2)*( (Tref+Sconst)/(T_ee+Sconst)
);

        if P_ee < 0
            disp('pressure negative')
        end

        rho_nn = QNN(1);
        u_nn =   QNN(2)/QNN(1);
        v_nn =   QNN(3)/QNN(1);
        et_nn =  QNN(4)/QNN(1);
        P_nn =   rho_nn*(gama-1)*(et_nn - 0.5*(u_nn^2 + v_nn^2));
        a_nn = sqrt( gama*P_nn/rho_nn );
        T_nn = P_nn/(R*rho_nn);
        MuNN =  mu_ref*(T_nn/Tref)^(3/2)*( (Tref+Sconst)/(T_nn+Sconst)
);

        if P_nn < 0
            disp('pressure negative')
        end

        rho_ww = QWW(1);
        u_ww =   QWW(2)/QWW(1);
        v_ww =   QWW(3)/QWW(1);
        et_ww =  QWW(4)/QWW(1);
        P_ww =   rho_ww*(gama-1)*(et_ww - 0.5*(u_ww^2 + v_ww^2));
        a_ww = sqrt( gama*P_ww/rho_ww );
        T_ww = P_ww/(R*rho_ww);
        MuWW =  mu_ref*(T_ww/Tref)^(3/2)*( (Tref+Sconst)/(T_ww+Sconst)
);

        if P_ww < 0
            disp('pressure negative')
        end

        rho_ss = QSS(1);
        u_ss =   QSS(2)/QSS(1);
        v_ss =   QSS(3)/QSS(1);
        et_ss =  QSS(4)/QSS(1);
        P_ss =   rho_ss*(gama-1)*(et_ss - 0.5*(u_ss^2 + v_ss^2));
        a_ss = sqrt( gama*P_ss/rho_ss );
        T_ss = P_ss/(R*rho_ss);
        MuSS =  mu_ref*(T_ss/Tref)^(3/2)*( (Tref+Sconst)/(T_ss+Sconst)
);

        if P_ss < 0
            disp('pressure negative')
        end
```

```
        [Fiphj] =
upwindedFluxesAUSMplus(rho(i),u(i),v(i),a(i),P(i),et(i),rho_e,u_e,v_e,a
_e,P_e,et_e,gama,1);
        [Gijph] =
upwindedFluxesAUSMplus(rho(i),u(i),v(i),a(i),P(i),et(i),rho_n,u_n,v_n,a
_n,P_n,et_n,gama,2);
        [Fimhj] =
upwindedFluxesAUSMplus(rho_w,u_w,v_w,a_w,P_w,et_w,rho(i),u(i),v(i),a(i)
,P(i),et(i),gama,1);
        [Gijmh] =
upwindedFluxesAUSMplus(rho_s,u_s,v_s,a_s,P_s,et_s,rho(i),u(i),v(i),a(i)
,P(i),et(i),gama,2);

        [Fiphj_d] =
upwindedFluxesAUSMplus(rho(i),u(i),v(i),a(i),P(i),et(i),rho_ed,u_ed,v_e
d,a_ed,P_ed,et_ed,gama,1);
        [Gijph_d] =
upwindedFluxesAUSMplus(rho(i),u(i),v(i),a(i),P(i),et(i),rho_nd,u_nd,v_n
d,a_nd,P_nd,et_nd,gama,2);
        [Fimhj_d] =
upwindedFluxesAUSMplus(rho_wd,u_wd,v_wd,a_wd,P_wd,et_wd,rho(i),u(i),v(i
),a(i),P(i),et(i),gama,1);
        [Gijmh_d] =
upwindedFluxesAUSMplus(rho_sd,u_sd,v_sd,a_sd,P_sd,et_sd,rho(i),u(i),v(i
),a(i),P(i),et(i),gama,2);
        %
        %
        Fiphj = Fiphj_d.*(1 - minmod_xf{i}) + minmod_xf{i}.*Fiphj ;
        Gijph=  Gijph_d.*(1 - minmod_yf{i}) + minmod_yf{i}.*Gijph ;
        Fimhj = Fimhj_d.*(1 - minmod_xb{i}) + minmod_xb{i}.*Fimhj ;
        Gijmh = Gijmh_d.*(1 - minmod_yb{i}) + minmod_yb{i}.*Gijmh ;

        dF_dx = (Fiphj - Fimhj)/(dx(i)/2);
        dG_dy = (Gijph - Gijmh)/(dy(i)/2);

        Qf(:,i) = Q(:,i) - dt * (dF_dx + dG_dy);

        rho(i) = Qf(1,i);
        u(i) =   Qf(2,i)/Qf(1,i);
        v(i) =   Qf(3,i)/Qf(1,i);
        et(i) =  Qf(4,i)/Qf(1,i);
        intern_e =   et(i) - 0.5*(u(i)^2 + v(i)^2);
        T(i) = (gama-1)*intern_e/R;
        P(i) = rho(i)*R*T(i);
        Vmag(i) = sqrt( u(i)^2 + v(i)^2 );
        M(i) = Vmag(i)/a(i);
        a(i) = sqrt( gama*P(i)/rho(i) );


    end
    conv_eq(:,n) = sum(abs(Qf-Q),2);
```

```
    conv_max = max(conv_eq,[],2);
    conv_eq(:,n) = conv_eq(:,n)./conv_max;

    Q = Qf;

    %Apply and update boundary conditions
    k=1;
    for i = 1:nume;
        for j = 1:NB(i)

            if i == 1 || i == 2 || i==3  || i == 5 || i == 6 || i == 7

                vm(k) = -u(k)*nx(k)/ny(k);
                um(k) = sqrt(Vmag(k)^2-vm(k)^2);
                Tm(k) = T(k);
                Pm(k) = P(k);
                rhom(k) = rho(k);
                am(k) = sqrt(gama*Pm(k)/rhom(k));
                etm(k) = Pm(k)/(rhom(k)*(gama-1))+ 0.5*(um(k)^2 +
vm(k)^2);
                Vmagm(k) = sqrt( um(k)^2 + vm(k)^2 );
                Mm(k) = Vmagm(k)/am(k);


            elseif i == 4

                Vn = um(k)*nx(k) + vm(k)*ny(k);

                rhom(k) = rho(k); % - dt*(um(k)+am(k)) *(rhom(k)-
rho(k))/offset;
                um(k) = u(k); % - dt*Vn *(um(k)-u(k))/offset;
                vm(k) = v(k); % - dt*Vn *(vm(k)-v(k))/offset;
                Pm(k) = P(k); % - dt*(um(k)+am(k)) *(Pm(k)-
P(k))/offset;
                etm(k) = et(k);% - dt*(um(k)+am(k)) *(etm(k)-
et(k))/offset;
                Tm(k) = T(k); %- dt*(um(k)+am(k)) *(Tm(k)-T(k))/offset;
                am(k) = a(k);

                Vmagm(k) = sqrt( um(k)^2 + vm(k)^2 );
                Mm(k) = Vmagm(k)/am(k);



            else

                rhom(k) = rhoinit;
                um(k) = uinit;
                vm(k) = vinit;
                Pm(k) = Pinit;
                etm(k) = etinit;
```

```
            am(k) = ainit;
            Tm(k) = Tinit;


            Vmagm(k) = sqrt( uinit^2 + vinit^2 );
            Mm(k) = sqrt( uinit^2 + vinit^2 )/ainit;
        end

        k = k+1;

    end
end



rhoall = [rhom;rho];
uall = [um;u];
vall = [vm;v];
etall = [etm;et];
Pall = [Pm;P];
aall = [am;a];
Mall = [Mm;M];
Tall = [Tm;T];


Qall(1,:) = rhoall;
Qall(2,:) = rhoall.*uall;
Qall(3,:) = rhoall.*vall;
Qall(4,:) = rhoall.*etall;

if n == report*count || n == 1


    %         P_old = P;
    conv_eq(:,n)
    max(Pall)
    n

    count = count+1;


    figure(5)
    scatter(xall,yall,[],Mall,'filled');
    colorbar
    axis equal
    pause(0.001)

    figure(6)
    scatter(xall,yall,[],rhoall,'filled');
    colorbar
```

```
axis equal
pause(0.001)


figure(7)
scatter(xall,yall,[],Pall,'filled');
colorbar
axis equal
pause(0.001)


minmod_xbMat = cell2mat(minmod_xb);

figure(8)
scatter(x,y,[],minmod_xbMat(1 ,:)','filled');
colorbar
axis equal
pause(0.001)

minmod_ybMat = cell2mat(minmod_yb);
figure(9)
scatter(x,y,[],minmod_ybMat(1 ,:)','filled');
colorbar
axis equal
pause(0.0001)

figure(10)
Nedge = length(xm)-1;
C = [(1:(Nedge-1))' (2:Nedge)'; Nedge 1];
tri = delaunayTriangulation(xall,yall,C);
IO = isInterior(tri);
h = trisurf(tri(IO,:),xall,yall,Pall);


f = scatteredInterpolant(xall,yall,Pall(:),'natural');
Z = f(X,Y);
% h=surf(X,Y,Z);

figure(11)
plot(X(:,1),Z(:,1))
hold all
plot(CFD(:,1),CFD(:,2))
hold off

figure(12)
semilogy(conv_eq(1,:))
hold all
semilogy(conv_eq(2,:))
semilogy(conv_eq(3,:))
semilogy(conv_eq(4,:))
hold off
```

```
        figure(4)
        scatter(xall,yall,[],Tall,'filled');
        colorbar

        axis equal

    end
e
```

APPENDIX C

LOCALIZED BLENDED RBF COLLOCATION NAVIER-STOKES SOLVER

```
%Author: Michael F. Harris
clc
close all

%*****************Setting up Geometry and Point
Cloud**********************

xv = [0.0,0.1,0.3,0.3,0.1,0,0];
yv = [0.0,0.0,0.035,0.2,0.2,0.2,0.0];

xv_solid = [ 0.0, 0.1,  0.3   ,  0.3  ,  0.1  ,  0.0  , 0.0];
yv_solid = [ 0.0, 0.0,  0.035 ,  0.01, -0.025, -0.025, 0.0];


% xv_solid = [ 0.0, 0.1, 0.3 ,  0.7, 0.7,  0.3,  0.1  ,  0.0  , 0.0];
% yv_solid = [ 0.0, 0.0,  0.035 ,  0.035, 0.01, 0.01, -0.025, -0.025,
0.0];

L = 0.3;
H =0.2;

NB = [40,80,66,80,40,80];
NBsolid = [40,60,10,80,40,10];

NBtot = sum(NB);
NBsolidtot = sum(NBsolid);


% Number of edges that make up the domain
nume = 6;
nume_solid = 6;

% Boundary condition type  1 = Wall, 2 = Inlet, 3 = Outlet
BC = [1,1,1,3,1,2];

% Offset for the shadow numbers
offset  = 1e-5;
offset2 = 1e-5;
offset_s = 5e-4;
% Number of interior points
NP = 50;

% Plotting the domain
figure(1)
plot(xv,yv)
title('Domain')
xlabel('x')
ylabel('y')
axis equal

%Fluid Nodes
```

```
layers = 5;
growth = 2.5;

%Solid Nodes
solid_layers = 1;
solid_growth = 1.0;

[xb,yb,xm,ym,tx,ty,nx,ny,xs,ys,xg,yg,~,~] =
preprocessing_viscous(xv,yv,NB,offset,NP,layers,growth,-1);
[xbs,ybs,xms,yms,txs,tys,nxs,nys,xss,yss,xgs,ygs,~,~] =
preprocessing_viscous(xv_solid,yv_solid,NBsolid,offset_s,NP,solid_layer
s,solid_growth,1);


rx = zeros(1,NBtot);
ry = zeros(1,NBtot);
rmag = zeros(1,NBtot);

for i = 1:NBtot

    rx(i) = ( xb(i+1)-xb(i) );
    ry(i) = ( yb(i+1)-yb(i) );

    rmag(i) = sqrt(rx(i)^2+ry(i)^2);
end

dndx = rx./rmag;
dndy = ry./rmag;

for i = 1:NBsolidtot

    rx(i) = ( xbs(i+1)-xbs(i) );
    ry(i) = ( ybs(i+1)-ybs(i) );

    rmag(i) = sqrt(rx(i)^2+ry(i)^2);
end

dndx_solid = rx./rmag;
dndy_solid = ry./rmag;

figure(2)
%Fluids Nodes
plot(xv,yv)
hold all
plot(xb,yb,'+')
plot(xm,ym,'.','MarkerSize',15)
plot(xs,ys,'.','MarkerSize',15)
%Solid Nodes
plot(xv_solid,yv_solid)
plot(xbs,ybs,'+')
plot(xms,yms,'.','MarkerSize',15)
```

```
plot(xss,yss,'.','MarkerSize',15)
axis equal

xpoly = zeros(1,NBtot);
ypoly = zeros(1,NBtot);
xpoly_solid = zeros(1,NBsolidtot);
ypoly_solid = zeros(1,NBsolidtot);

buffer = offset;
buffer_s = offset_s;

for i = 1:length(xs(:,layers))

    xpoly(i) = xs(i,layers) - nx(i)*buffer;
    ypoly(i) = ys(i,layers) - ny(i)*buffer;

end

xpoly2 = [0.0105, 0.1, 0.2895, 0.2895, 0.0105, 0.0105];
ypoly2 = [0.0105, 0.0105, 0.04375,0.1895, 0.1895, 0.0105];

for i = 1:length(xss(:,solid_layers))

    xpoly_solid(i) = xss(i,solid_layers) + nxs(i)*buffer_s;
    ypoly_solid(i) = yss(i,solid_layers) + nys(i)*buffer_s;

end

[Xint,Yint] = meshgrid(0.00125:0.0025:0.3,0.00125:0.0025:0.35);
% [Xint2,Yint2] = meshgrid(0.0005:0.001:0.3,0.0005:0.001:0.35);

% [Xint2,Yint2] = meshgrid(0.30125:0.0025:0.7,0.0375:0.00125:0.3);
[Xint_solid,Yint_solid] = meshgrid(0.00125:0.0025:0.7,-
0.02375:0.0025:0.035);

% [Xint3,Yint3] = meshgrid(0:0.0025:0.7,0.062:0.0025:0.2);

% [xint,yint] = inDomain(xpoly,ypoly,Xint,Yint,1);
% % [xint2,yint2] = inDomain(xpoly,ypoly,Xint2,Yint2,1);
% [xint_s,yint_s] =
inDomain(xpoly_solid,ypoly_solid,Xint_solid,Yint_solid,-1);

[in,on] = inpolygon(Xint,Yint,xpoly,ypoly);
% [in2,on2] = inpolygon(Xint2,Yint2,xpoly,ypoly);
[in_solid,on_solid] =
inpolygon(Xint_solid,Yint_solid,xpoly_solid,ypoly_solid);
% [in3,on3] = inpolygon(Xint3,Yint3,xpoly,ypoly);

% [Xint,Yint] = meshgrid(0.00025:0.0005:0.3,0.00025:0.0005:0.2);
% [Xint2,Yint2] = meshgrid(0.3025:0.001:0.7,0.0375:0.001:0.2);
% % [Xint3,Yint3] = meshgrid(0:0.0025:0.7,0.062:0.0025:0.2);
```

```
% [in,on] = inpolygon(Xint,Yint,xpoly,ypoly);
% [in2,on2] = inpolygon(Xint2,Yint2,xpoly,ypoly);

xint = Xint(in&~on);
yint = Yint(in&~on);

% xint2 = Xint2(in2);
% yint2 = Yint2(in2);
%
% [in3,on3] = inpolygon(xint2,yint2,xpoly,ypoly);

% xint3 = xint2(in3);
% yint3 = yint2(in3);

xint_s = Xint_solid(in_solid&~on_solid);
yint_s = Yint_solid(in_solid&~on_solid);

% for i = 1:length(xint)
%     xint(i) = xint(i) + 0.00025*(-1)^i;
% %     yint(i) = yint(i) + 0.0005*(-1)^i;
% end

%  xint = [xint;Xint2(in2)];
%  yint = [yint;Yint2(in2)];

%  xint = [xint];
%  yint = [yint];

hold off
figure(3)
%Fluid Nodes
plot(xv,yv)
hold all
plot(xb,yb,'+')
plot(xm,ym,'.','MarkerSize',15)
plot(xs,ys,'.','MarkerSize',15)
plot(xint,yint,'.','MarkerSize',5)

%Solid Nodes
plot(xv_solid,yv_solid)
plot(xbs,ybs,'+')
plot(xms,yms,'.','MarkerSize',15)
plot(xss,yss,'.','MarkerSize',15)
plot(xint_s,yint_s,'.','MarkerSize',5)
axis equal
% axis([0.0 0.7 0.0 0.2])

pause(0.001)

%**********************Input
values**************************************
```

```
CFL  = 0.001;
Mach = 2.0;
Pinit = 100e3;
Tinit = 300;
MW = 28;
gama = 1.4;
R = 287.05;
mu = 1.846e-5;
kcond = 0.02624;
k_solid = 21.9;
rhoinit = Pinit/(R*Tinit);
ainit = sqrt(gama*R*Tinit);
uinit = Mach*ainit;
vinit = 0.0;
etinit = (R*Tinit)/(gama-1) + 0.5*(uinit^2 + vinit^2);


%Subdomain Parameters
theta = [0:0.01:2*pi,0]; %Angles in radians for circular subdomain
rsub = 0.0075;            %Radius of the circular subdomain
rsub_mls = 0.007;
rsub_solid = 0.01;

%Shape paramters
sf  = 0.0005;            %shape parameter for smooth RBF interpolation
sf2 = 0.00001;            %shape parameter for discontinuous
interpolation
sf3 = 0.00001;
sf4 = 0.00001;
sfi = 1e-3;
sf_solid = 0.01;


% Applying intial condtion to domain and applying boundary conditions
Nsteps = 1;
[rhom,rhos,rhoint,rho,rhoall] =
intializeRho(rhoinit,Nsteps,nume,NB,length(xint),nume,layers);
[um,us,uint,u,uall] =
intializeU(uinit,Nsteps,nume,NB,length(xint),nume,layers);
[vm,vs,vint,v,vall] =
intializeV(vinit,Nsteps,nume,NB,length(xint),nume,layers);
[Pm,Ps,Pint,P,Pall] =
intializeP(Pinit,Nsteps,nume,NB,length(xint),nume,layers);
[etm,ets,etint,et,etall] =
intializeEt(etinit,Nsteps,nume,NB,length(xint),nume,layers);
[am,as,aint,a,aall] =
intializeSoS(ainit,Nsteps,nume,NB,length(xint),nume,layers);


Vmagm = sqrt(um.^2+vm.^2);
Vmags = sqrt(us.^2+vs.^2);
```

```
Vmagint = sqrt(uint.^2+vint.^2);
Vmagall = sqrt(uall.^2+vall.^2);

Mm = Vmagm./am;
Ms = Vmags./as;
Mint = Vmagint./aint;
Mall = Vmagall./aall;

M = [Ms;Mint];

Tm = Pm./(R*rhom);
Ts = Ps./(R*rhos);
Tint = Pint./(R*rhoint);
Tall = Pall./(rhoall*R);

mu_ref = 1.716e-5;
Tref = 273.15;
Sconst = 110.4;
mu_s =  mu_ref*((Tall/Tref).^(3/2)).*( (Tref+Sconst)./(Tall+Sconst) );

T = [Ts;Tint];

NQ = length(rho);
NQall = length(rhoall);
Q = zeros(4,NQ);
Qall = zeros(4,NQall);

Q(1,:) = rho;
Q(2,:) = rho.*u;
Q(3,:) = rho.*v;
Q(4,:) = rho.*et;

Qall(1,:) = rhoall;
Qall(2,:) = rhoall.*uall;
Qall(3,:) = rhoall.*vall;
Qall(4,:) = rhoall.*etall;


Nsall = sum(NB)*layers;
xsall = zeros(1,Nsall);
ysall = zeros(1,Nsall);


k = 1;
for i = 1:layers
    for j = 1:NBtot

        xsall(k) = xs(j,i);
        ysall(k) = ys(j,i);
        k = k+1;
```

```
    end
end

k = 1;
xsall_s = zeros(solid_layers*sum(NBsolid),1);
ysall_s = zeros(solid_layers*sum(NBsolid),1);
for i = 1:solid_layers
    for j = 1:sum(NBsolid)

        xsall_s(k) = xss(j,i);
        ysall_s(k) = yss(j,i);
        k = k+1;

    end
end

xall = [xm';xsall';xint];
yall = [ym';ysall';yint];

x = [xsall';xint];
y = [ysall';yint];

xall_solid = [xms';xsall_s;xint_s];
yall_solid = [yms';ysall_s;yint_s];

x_solid = [xsall_s;xint_s];
y_solid = [ysall_s;yint_s];

%CHT Boundary and Initial Conditions
Tm_solid = zeros(1,NBsolidtot);
Tall_solid = zeros(1,NBsolidtot);
T_solid = zeros(1,NBsolidtot);

Tm_solid(1:length(xms))  = 300;
Tall_solid(1:length(xall_solid)) = 300;
T_solid(1:length(x_solid)) = 300;

T_solid_F = T_solid;

npbs = length(x);
npbs_solid = length(x_solid);

%**************Computing Interpolation
Vectors*****************************
%The interpolation vectors only need to computed once for the domain.
%Interpolation vectors are to be saved and

dx = zeros(npbs,1);
dy = zeros(npbs,1);

indices = cell(npbs,1);
```

```
indices2 = cell(npbs,1);
indicesLS = cell(npbs,1);
r = cell(npbs,1);

eta_err_S = cell(npbs,1);
eta_err_D = cell(npbs,1);

eta_E = cell(npbs,1);
eta_W = cell(npbs,1);
eta_N = cell(npbs,1);
eta_S = cell(npbs,1);

eta_EE = cell(npbs,1);
eta_WW = cell(npbs,1);
eta_NN = cell(npbs,1);
eta_SS = cell(npbs,1);

eta_EEE = cell(npbs,1);
eta_WWW = cell(npbs,1);
eta_NNN = cell(npbs,1);
eta_SSS = cell(npbs,1);

eta_Es = cell(npbs,1);
eta_Ws = cell(npbs,1);
eta_Ns = cell(npbs,1);
eta_Ss = cell(npbs,1);

eta_Ed = cell(npbs,1);
eta_Wd = cell(npbs,1);
eta_Nd = cell(npbs,1);
eta_Sd = cell(npbs,1);

ith_index = zeros(1,npbs);

CNs = zeros(npbs,1);
CNd = zeros(npbs,1);

parfor i = 1:npbs

    xsub = x(i) + rsub*cos(theta);
    ysub = y(i) + rsub*sin(theta);

    xsub_mls = x(i) + rsub_mls*cos(theta);
    ysub_mls = y(i) + rsub_mls*sin(theta);

    [xo,yo,rhoo,uo,vo,Po,eto,ao,indices{i}] =
subdomain_fast(rsub,xall,yall,xsub,ysub,rhoall,uall,vall,Pall,etall,aal
l);
    [xls,yls,rhols,uls,vls,Pls,etls,als,indicesLS{i}] =
subdomain_fast(rsub_mls,xall,yall,xsub_mls,ysub_mls,rhoall,uall,vall,Pa
ll,etall,aall);
```

```
    r{i} = sqrt( (x(i)-xo).^2 + (y(i)-yo).^2 );

    ith_index(i) = find(r{i}==0);

    Rmax = max(r{i});

    xo_e = xo;
    yo_e = yo;

    xo_e(ith_index(i))=[];
    yo_e(ith_index(i))=[];

    x_dist = (nonzeros(abs(x(i) - xo)));
    y_dist = (nonzeros(abs(y(i) - yo)));

    x_distMLS = (nonzeros(abs(x(i) - xls)));
    y_distMLS = (nonzeros(abs(y(i) - yls)));

    x_distMLS = max(x_distMLS);
    y_distMLS = max(y_distMLS);

    x_dist2 = max(x_dist);
    y_dist2 = max(y_dist);

    x_dist = min(x_dist(x_dist >= offset));
    y_dist = min(y_dist(y_dist >= offset));



    mean_dist = mean(nonzeros(r{i}));
    min_dist = min(x_dist,y_dist);

    if i <= sum(NB)*layers
        dx(i) = offset;
        dy(i) = offset;
    else
        dx(i) = 100*offset;
        dy(i) = 100*offset;
    end

    [xi,yi,rhoi,ui,vi,Pi,eti,ai,indices2{i}] =
square_subdomain(x(i),y(i),x,y,rsub,rho,u,v,P,et,a,10);


%             hold off
%             figure(3)
%             plot(xint,yint,'b.')
%             hold all
%             plot(xm,ym,'black.')
```

```
%             plot(xs,ys,'r.')
%             plot(xo,yo,'*')
%             plot(xls,yls,'*')
%             plot(x(i)+dx(i),y(i),'*')
%             plot(x(i),y(i)+dy(i),'*')
%             plot(x(i)-dx(i),y(i),'*')
%             plot(x(i),y(i)-dy(i),'*')
%             plot(x(i),y(i),'o')
%             plot(xsub_mls,ysub_mls)
%             axis equal
%             legend('Interior Nodes','Boundary Nodes', 'Shadow Nodes')
%             axis([0.0 0.75 0.0 0.2])
%             pause(0.001)
%             hold off

%     [RBF_s,CNs(i)] = call_RBF(xo,xo,yo,yo,rsub,sf);
%
%     [RBF_sc,CNsc] = call_RBF(xo,xo,yo,yo,rsub,sf3);
%     [RBF_dc,CNdc] = call_RBF(xo,xo,yo,yo,rsub,sf4);
%
%     [RBF_s2,~] = call_RBF(xi,xi,yi,yi,rsub,sfi);
%
%     [RBF_d,CNd(i)] = call_RBF(xo,xo,yo,yo,rsub,sf2);
%
%     [RBF_i,~] = call_RBF(xi,xi,yi,yi,rsub,sfi);
%
%     xstencil_1 = [x(i),x(i),x(i),x(i)];
%     ystencil_1 = [y(i),y(i),y(i),y(i)];
%
%     [eta_err_S{i},~,~,~,~,~] =
call_RBF_comp(xo_e,xo_e,yo_e,yo_e,sf,xstencil_1,ystencil_1,rsub,x(i),y(
i));
%     [eta_err_D{i},~,~,~,~,~] =
call_RBF_comp(xo_e,xo_e,yo_e,yo_e,sf2,xstencil_1,ystencil_1,rsub,x(i),y
(i));

    xstencil_1 = [x(i)+dx(i)/2,x(i)-dx(i)/2,x(i),x(i)];
    ystencil_1 = [y(i),y(i),y(i)+dy(i)/2,y(i)-dy(i)/2];

    [eta_E{i},eta_W{i},eta_N{i},eta_S{i},~,~] =
call_RBF_comp(xo,xo,yo,yo,sf,xstencil_1,ystencil_1,offset,x(i),y(i),x_d
ist2,y_dist2);

    xstencil_1 = [x(i)+dx(i),x(i)-dx(i),x(i),x(i)];
    ystencil_1 = [y(i),y(i),y(i)+dy(i),y(i)-dy(i)];

    [eta_EE{i},eta_WW{i},eta_NN{i},eta_SS{i},~,~] =
call_RBF_comp(xo,xo,yo,yo,sf,xstencil_1,ystencil_1,offset,x(i),y(i),x_d
ist2,y_dist2);

    xstencil_1 = [x(i)+ dx(i),x(i)- dx(i),x(i),x(i)];
    ystencil_1 = [y(i),y(i),y(i)+ dy(i),y(i)- 0.75*dy(i)];
```

```
    [eta_EEE{i},eta_WWW{i},eta_NNN{i},eta_SSS{i},~,~] =
call_RBF_comp(xo,xo,yo,yo,sf,xstencil_1,ystencil_1,offset,x(i),y(i),x_d
ist2,y_dist2);



    xstencil_1 = [x(i)+dx(i)/2,x(i)-dx(i)/2,x(i),x(i)];
    ystencil_1 = [y(i),y(i),y(i)+dy(i)/2,y(i)-dy(i)/2];


    xstencil_2 = [x(i)+dx(i),x(i)-dx(i),x(i),x(i)];
    ystencil_2 = [y(i),y(i),y(i)+dy(i),y(i)-dy(i)];

    [eta_Es{i},eta_Ws{i},eta_Ns{i},eta_Ss{i},~,CNs(i)] =
call_RBF_comp(xo,xo,yo,yo,sf,xstencil_1,ystencil_1,offset,x(i),y(i),x_d
ist2,y_dist2);
    [eta_Ed{i},eta_Wd{i},eta_Nd{i},eta_Sd{i},~,CNd(i)] =
call_RBF_comp(xo,xo,yo,yo,sf2,xstencil_1,ystencil_1,offset,x(i),y(i),x_
dist2,y_dist2);
    [eta_EEs{i},eta_WWs{i},eta_NNs{i},eta_SSs{i},~,~] =
call_RBF_comp(xo,xo,yo,yo,sf,xstencil_2,ystencil_2,offset,x(i),y(i),x_d
ist2,y_dist2);
    [eta_EEd{i},eta_WWd{i},eta_NNd{i},eta_SSd{i},~,~] =
call_RBF_comp(xo,xo,yo,yo,sf2,xstencil_2,ystencil_2,offset,x(i),y(i),x_
dist2,y_dist2);

    [RBFpoly,CNs(i)] = call_RBFwPoly(xo,xo,yo,yo,offset,sf,x(i),y(i));
    [RBFpoly_d,~] = call_RBFwPoly(xo,xo,yo,yo,offset,sf2,x(i),y(i));

    %Second derivative with smooth interpolation
    [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22,RBF_dxdy,~] =
derivativesPoly(x(i),xo,y(i),yo,sf,offset,x(i),y(i));
    eta_dx_c{i} = RBF_dx2/RBFpoly;
    eta_dy_c{i} = RBF_dy2/RBFpoly;

    eta_dx2_c{i} = RBF_dx22/RBFpoly;
    eta_dy2_c{i} = RBF_dy22/RBFpoly;
    eta_dxdy_c{i} = RBF_dxdy/RBFpoly;

    [RBF_dx2,RBF_dy2,RBF_dx22,RBF_dy22,RBF_dxdy,~] =
derivativesPoly(x(i),xo,y(i),yo,sf2,offset,x(i),y(i));
    eta_dx_cd{i} = RBF_dx2/RBFpoly_d;
    eta_dy_cd{i} = RBF_dy2/RBFpoly_d;
    eta_dxdy_cd{i} = RBF_dxdy/RBFpoly_d;

    eta_dx2_cd{i} = RBF_dx22/RBFpoly_d;
    eta_dy2_cd{i} = RBF_dy22/RBFpoly_d;
    eta_dxdy_cd{i} = RBF_dxdy/RBFpoly_d;
```

```
    [~,LS_dx{i}, LS_dy{i}, LS_dx2{i}, LS_dy2{i}, LS_dxdy{i}] =
MLSoperator(x(i),y(i),xls,yls,x(i),y(i),rsub,x_distMLS,y_distMLS);

end


L1 = 0.1;
L2 = 0.2;
L3 = 0.4;
H1 = 0.2;
H2 = H1-0.035;
dx_mesh = 0.002;
dy_mesh = 0.002;
[X,Y,testValue] = mesh(L1,L2,L3,H1,H2,dx_mesh,dy_mesh);
load CFD

count = 1;
report = 1000;


parfor i = 1:length(xall)

%     xsub = xall(i) + rsub*cos(theta);
%     ysub = yall(i) + rsub*sin(theta);

    xsub_mls = xall(i) + rsub_mls*cos(theta);
    ysub_mls = yall(i) + rsub_mls*sin(theta);

    [xls_all,yls_all,~,~,~,~,~,~,indices_all_ls{i}] =
subdomain_fast(rsub_mls,xall,yall,xsub_mls,ysub_mls,rhoall,uall,vall,Pa
ll,etall,aall);

    x_dist = (nonzeros(abs(xall(i) - xls_all)));
    y_dist = (nonzeros(abs(yall(i) - yls_all)));

    x_dist2 = max(x_dist);
    y_dist2 = max(y_dist);

%     [xrbf_all,yrbf_all,~,~,~,~,~,~,indices_all_rbf{i}] =
subdomain_fast(rsub,xall,yall,xsub,ysub,rhoall,uall,vall,Pall,etall,aal
l);

%     [RBF_s,CNs(i)] =
call_RBFwPoly(xrbf_all,xrbf_all,yrbf_all,yrbf_all,offset,sf2,xall(i),ya
ll(i));
%     [RBF_dx2,RBF_dy2,~,~,~,~] =
derivativesPoly(xall(i),xrbf_all,yall(i),yrbf_all,sf2,offset,xall(i),ya
ll(i));
%
%     eta_dx_all{i} = RBF_dx2/RBF_s;
%     eta_dy_all{i} = RBF_dy2/RBF_s;
```

```
    [~,LS_dx_all{i}, LS_dy_all{i}, ~ ,~] =
MLSoperator(xall(i),yall(i),xls_all,yls_all,xall(i),yall(i),rsub,x_dist
2,y_dist2);
end

cfl = CFL;

txx = zeros(length(xall),1);
tyy = zeros(length(xall),1);
txy = zeros(length(xall),1);

qx = zeros(length(xall),1);
qy = zeros(length(xall),1);

Qf = Q;


disp('Pre-processing completed')
disp('Solving...')

minmod_xb = cell(1,npbs);
minmod_yb = cell(1,npbs);
minmod_xf = cell(1,npbs);
minmod_yf = cell(1,npbs);


for n = 1:60000

    if cfl >= 0.5

        cfl = 0.5;
    else
        cfl = CFL*2^((n-1)/4);
    end


%     for i = 1:length(xall)
%
%         %         if i <= sum(NB)
%         %
%         % %             plot(xall,yall,'.')
%         % %             hold all
%         % %             plot(xall(i),yall(i),'.')
%         % %             plot(u(i),v(i),'o')
%         % %             plot(um(i),vm(i),'o')
%         %
%         %             du_dx = (-3*um(i) + 4*u(i) -
u(sum(NB)+i))/(2*offset) * -nx(i);
%         %             du_dy = (-3*um(i) + 4*u(i) -
u(sum(NB)+i))/(2*offset) * -ny(i);
```

```
%          %              dv_dx = (-3*vm(i) + 4*v(i) -
v(sum(NB)+i))/(2*offset) * -nx(i);
%          %              dv_dy = (-3*vm(i) + 4*v(i) -
v(sum(NB)+i))/(2*offset) * -ny(i);
%          %
%          %              dT_dx = (-3*Tm(i) + 4*T(i) -
T(sum(NB)+i))/(2*offset) * -nx(i);
%          %              dT_dy = (-3*Tm(i) + 4*T(i) -
T(sum(NB)+i))/(2*offset) * -ny(i);
%          %
%          %          else
%
%
%                       du_dx = LS_dx_all{i}*uall(indices_all_ls{i});
%                       dv_dy = LS_dy_all{i}*vall(indices_all_ls{i});
%
%                       dv_dx = LS_dx_all{i}*vall(indices_all_ls{i});
%                       du_dy = LS_dy_all{i}*uall(indices_all_ls{i});
%
%                       dT_dx = LS_dx_all{i}*Tall(indices_all_ls{i});
%                       dT_dy = LS_dy_all{i}*Tall(indices_all_ls{i});
%
% %          up = [uall(indices_all_rbf{i});zeros(3,1)];
% %          vp = [vall(indices_all_rbf{i});zeros(3,1)];
% %          Tp = [Tall(indices_all_rbf{i});zeros(3,1)];
% %
% %          du_dx = eta_dx_all{i}*up;
% %          dv_dy = eta_dy_all{i}*vp;
% %
% %          dv_dx = eta_dx_all{i}*vp;
% %          du_dy = eta_dy_all{i}*up;
% %
% %          dT_dx = eta_dx_all{i}*Tp;
% %          dT_dy = eta_dy_all{i}*Tp;
%
%          %          end
%
%          txx(i) = 2/3*mu*(2*du_dx - dv_dy);
%          tyy(i) = 2/3*mu*(2*dv_dy - du_dx);
%          txy(i) = mu*(du_dy + dv_dx);
%
%          %          txxR(k) = 2/3*muT_all(k)*(2*du_dx - dv_dy);
%          %          tyyR(k) = 2/3*muT_all(k)*(2*dv_dy - du_dx);
%          %          txyR(k) = muT_all(k)*(du_dy + dv_dx);
%          %
%          %          kT(k) = muT_all(k)/(PrT*rhoall(k));
%          qx(i) = -(kcond)*dT_dx;
%          qy(i) = -(kcond)*dT_dy;
%
%
%
%      end
%      E4 = uall.*txx + vall.*txy - qx;
%      F4 = uall.*txy + vall.*tyy - qy;
```

```
    for i = 1:npbs

        dt  = cfl/( (abs(u(i))+a(i))/dx(i) + (abs(v(i)) + a(i))/dy(i) +
2*mu/(rhoall(i)*dy(i)^2) + 2*mu/(rhoall(i)*dx(i)^2)  );

%         hold off
%         plot(xall,yall,'.')
%         hold all
%         plot(xall(indices{i}),yall(indices{i}),'.')
%         pause(0.01)

        QP = [Qall(:,indices{i}),zeros(4,3)];

        Q_E = QP*eta_E{i}';
        Q_W = QP*eta_W{i}';
        Q_N = QP*eta_N{i}';
        Q_S = QP*eta_S{i}';

        Q_EE = QP*eta_EE{i}';
        Q_WW = QP*eta_WW{i}';
        Q_NN = QP*eta_NN{i}';
        Q_SS = QP*eta_SS{i}';

        Q_EEE = QP*eta_EEE{i}';
        Q_WWW = QP*eta_WWW{i}';
        Q_NNN = QP*eta_NNN{i}';
        Q_SSS = QP*eta_SSS{i}';


        if u(i) >= 0
            rxb1 = (Q_W - Q_WW)./( Q(:,i) - Q_W );
            rxb2 = ( Q(:,i) - Q_W )./(Q_W - Q_WW);

            rxf1 = (Q(:,i)-Q_W)./( Q_E - Q(:,i) );
            rxf2 = ( Q_E - Q(:,i) )./(Q(:,i)-Q_W);
        else
            rxb1 = (Q_E - Q(:,i))./(Q(:,i)- Q_W);
            rxb2 = ( Q_W - Q(:,i) )./( Q(:,i)-Q_E);

            rxf1 = (Q_EE - Q_E )./( Q_E - Q(:,i));
            rxf2 = ( Q(:,i) - Q_E )./(Q_E - Q_EE);
        end

        if v(i) >= 0
            ryb1 = (Q_S - Q_SS)./( Q(:,i) - Q_S );
            ryb2 = ( Q(:,i) - Q_S )./(Q_S - Q_SS);

            ryf1 = (Q(:,i) - Q_S )./( Q_N - Q(:,i) );
            ryf2 = ( Q_N - Q(:,i) )./(Q(:,i) - Q_S );
```

```
        else
            ryb1 = (Q_N - Q(:,i))./( Q(:,i) - Q_S );
            ryb2 = ( Q_S - Q(:,i) )./(Q(:,i) - Q_N );

            ryf1 = (Q_NN - Q_N)./( Q_N - Q(:,i) );
            ryf2 = ( Q(:,i) - Q_N )./(Q_N - Q_NN);
        end


        minmod_xb{i}  = max( 0,min( rxb1,1.0 ));
        minmod_yb{i}  = max( 0,min( ryb1,1.0 ));
        minmod_xf{i}  = max( 0,min( rxf1,1.0 ));
        minmod_yf{i}  = max( 0,min( ryf1,1.0 ));


%         txx_p = [txx(indices{i});zeros(3,1)];
%         tyy_p = [tyy(indices{i});zeros(3,1)];
%         txy_p = [txy(indices{i});zeros(3,1)];
%         Ev4_p = [E4(indices{i});zeros(3,1)];
%         Fv4_p = [F4(indices{i});zeros(3,1)];
%
%         dtxx_dx = eta_dx_cd{i}*txx_p;
%         dtyy_dy = eta_dy_cd{i}*tyy_p;
%         dtxy_dx = eta_dx_cd{i}*txy_p;
%         dtxy_dy = eta_dy_cd{i}*txy_p;
%
%         dEv4_dx = eta_dx_cd{i}*Ev4_p;
%         dFv4_dy = eta_dy_cd{i}*Fv4_p;


%         up = [uall(indices{i}); zeros(3,1)];
%         vp = [vall(indices{i}); zeros(3,1)];
%         Tp = [Tall(indices{i}); zeros(3,1)];
%
%         du_dx = eta_dx_c{i}*up;
%         dv_dx = eta_dx_c{i}*vp;
%
%         du_dy = eta_dy_c{i}*up;
%         dv_dy = eta_dy_c{i}*vp;
%
%         d2u_dx2 = eta_dx2_c{i}*up;
%         d2v_dx2 = eta_dx2_c{i}*vp;
%
%         d2u_dy2 = eta_dy2_c{i}*up;
%         d2v_dy2 = eta_dy2_c{i}*vp;
%
%         d2u_dxdy = eta_dxdy_c{i}*up;
%         d2v_dxdy = eta_dxdy_c{i}*vp;

        du_dx = LS_dx{i}*uall(indicesLS{i});
        dv_dx = LS_dx{i}*vall(indicesLS{i});
```

```
        du_dy = LS_dy{i}*uall(indicesLS{i});
        dv_dy = LS_dy{i}*vall(indicesLS{i});

        d2u_dx2 = LS_dx2{i}*uall(indicesLS{i});
        d2v_dx2 = LS_dx2{i}*vall(indicesLS{i});

        d2u_dy2 = LS_dy2{i}*uall(indicesLS{i});
        d2v_dy2 = LS_dy2{i}*vall(indicesLS{i});

        d2u_dxdy = LS_dxdy{i}*uall(indicesLS{i});
        d2v_dxdy = LS_dxdy{i}*vall(indicesLS{i});

        txx = 2/3*mu*(2*du_dx-dv_dy);
        tyy = 2/3*mu*(2*dv_dy-du_dx);

        dtxx_dx = 2/3*mu*(2*d2u_dx2-d2v_dy2);
        dtyy_dy = 2/3*mu*(2*d2v_dy2-d2u_dx2);
        dtxy_dx = mu*(d2u_dxdy + d2v_dx2);
        dtxy_dy = mu*(d2u_dy2 + d2v_dxdy);

        d2T_dx2 = kcond*LS_dx2{i}*Tall(indicesLS{i});
        d2T_dy2 = kcond*LS_dy2{i}*Tall(indicesLS{i});

        dEv4_dx = uall(i)*dtxx_dx + txx*du_dx + vall(i)*dtxy_dx +
txy*dv_dx + d2T_dx2;
        dFv4_dy = uall(i)*dtxy_dy + txy*du_dy + vall(i)*dtyy_dy +
tyy*dv_dy + d2T_dy2;


%                       dtxx_dx = LS_dx_all{i}*txx_mp; %(indicesLS{i});
%                       dtyy_dy = LS_dy_all{i}*tyy_mp; %(indicesLS{i});
%                       dtxy_dx = LS_dx_all{i}*txy_mp; %(indicesLS{i});
%                       dtxy_dy = LS_dy_all{i}*txy; %(indicesLS{i});
%
%                       dEv4_dx = LS_dx_mp{i}*Ev4_mp; %(indicesLS{i});
%                       dFv4_dy = LS_dy_mp{i}*Fv4_mp; %(indicesLS{i});

% dtxx_dx = LS_dx{i}*txx(indicesLS{i});
% dtyy_dy = LS_dy{i}*tyy(indicesLS{i});
% dtxy_dx = LS_dx{i}*txy(indicesLS{i});
% dtxy_dy = LS_dy{i}*txy(indicesLS{i});
%
% dEv4_dx = LS_dx{i}*E4(indicesLS{i});
% dFv4_dy = LS_dy{i}*F4(indicesLS{i});


dEv_dx_mls = [0; dtxx_dx; dtxy_dx; dEv4_dx];
dFv_dy_mls = [0; dtxy_dy; dtyy_dy; dFv4_dy];

nQRK = [Q(:,i),zeros(4,3)];
QallRK = [Qall,zeros(4,3)];
```

```
uallRK = uall;
vallRK = vall;
TallRK = Tall;


        QE = QP*eta_Ed{i}' + minmod_xf{i}.*(QP*eta_Es{i}' -
QP*eta_Ed{i}');
        QN = QP*eta_Nd{i}' + minmod_yf{i}.*(QP*eta_Ns{i}' -
QP*eta_Nd{i}');
        QW = QP*eta_Wd{i}' + minmod_xb{i}.*(QP*eta_Ws{i}' -
QP*eta_Wd{i}');
        QS = QP*eta_Sd{i}' + minmod_yb{i}.*(QP*eta_Ss{i}' -
QP*eta_Sd{i}');


        QEE = QP*eta_EEd{i}' + minmod_xf{i}.*(QP*eta_EEs{i}' -
QP*eta_EEd{i}');
        QNN = QP*eta_NNd{i}' + minmod_yf{i}.*(QP*eta_NNs{i}' -
QP*eta_NNd{i}');
        QWW = QP*eta_WWd{i}' + minmod_xb{i}.*(QP*eta_WWs{i}' -
QP*eta_WWd{i}');
        QSS = QP*eta_SSd{i}' + minmod_yb{i}.*(QP*eta_SSs{i}' -
QP*eta_SSd{i}');


        rho_e = QE(1);
        u_e =   QE(2)/QE(1);
        v_e =   QE(3)/QE(1);
        et_e =  QE(4)/QE(1);
        P_e =   rho_e*(gama-1)*(et_e - 0.5*(u_e^2 + v_e^2));
        a_e = sqrt( gama*P_e/rho_e );
        T_e = P_e/(R*rho_e);
        MuE =  mu_ref*(T_e/Tref)^(3/2)*( (Tref+Sconst)/(T_e+Sconst) );

        Mach_e = u_e/sqrt(gama*R*T_e);

        if P_e < 0
            disp('pressure negative')
        end

        rho_n = QN(1);
        u_n =   QN(2)/QN(1);
        v_n =   QN(3)/QN(1);
        et_n =  QN(4)/QN(1);
        P_n =   rho_n*(gama-1)*(et_n - 0.5*(u_n^2 + v_n^2));
        a_n = sqrt( gama*P_n/rho_n );
        T_n = P_n/(R*rho_n);
        MuN =  mu_ref*(T_n/Tref)^(3/2)*( (Tref+Sconst)/(T_n+Sconst) );

        Mach_n = v_n/sqrt(gama*R*T_n);
```

```
        if P_n < 0
            disp('pressure negative')
        end

        rho_w = QW(1);
        u_w =   QW(2)/QW(1);
        v_w =   QW(3)/QW(1);
        et_w =  QW(4)/QW(1);
        P_w =   rho_w*(gama-1)*(et_w - 0.5*(u_w^2 + v_w^2));
        a_w = sqrt( gama*P_w/rho_w );
        T_w = P_w/(R*rho_w);
        MuW =  mu_ref*(T_w/Tref)^(3/2)*( (Tref+Sconst)/(T_w+Sconst) );

        Mach_w = u_w/sqrt(gama*R*T_w);

        if P_w < 0
            disp('pressure negative')
        end

        rho_s = QS(1);
        u_s =   QS(2)/QS(1);
        v_s =   QS(3)/QS(1);
        et_s =  QS(4)/QS(1);
        P_s =   rho_s*(gama-1)*(et_s - 0.5*(u_s^2 + v_s^2));
        a_s = sqrt( gama*P_s/rho_s );
        T_s = P_s/(R*rho_s);
        MuS =  mu_ref*(T_s/Tref)^(3/2)*( (Tref+Sconst)/(T_s+Sconst) );

        Mach_s = v_s/sqrt(gama*R*T_s);


        if P_s < 0
            disp('pressure negative')
        end

        rho_ee = QEE(1);
        u_ee =   QEE(2)/QEE(1);
        v_ee =   QEE(3)/QEE(1);
        et_ee =  QEE(4)/QEE(1);
        P_ee =   rho_ee*(gama-1)*(et_ee - 0.5*(u_ee^2 + v_ee^2));
        a_ee = sqrt( gama*P_ee/rho_ee );
        T_ee = P_ee/(R*rho_ee);
        MuEE =  mu_ref*(T_ee/Tref)^(3/2)*( (Tref+Sconst)/(T_ee+Sconst)
);

        if P_ee < 0
            disp('pressure negative')
        end

        rho_nn = QNN(1);
        u_nn =   QNN(2)/QNN(1);
        v_nn =   QNN(3)/QNN(1);
```

```
        et_nn =  QNN(4)/QNN(1);
        P_nn =   rho_nn*(gama-1)*(et_nn - 0.5*(u_nn^2 + v_nn^2));
        a_nn = sqrt( gama*P_nn/rho_nn );
        T_nn = P_nn/(R*rho_nn);
        MuNN =  mu_ref*(T_nn/Tref)^(3/2)*( (Tref+Sconst)/(T_nn+Sconst)
);

        if P_nn < 0
            disp('pressure negative')
        end

        rho_ww = QWW(1);
        u_ww =   QWW(2)/QWW(1);
        v_ww =   QWW(3)/QWW(1);
        et_ww =  QWW(4)/QWW(1);
        P_ww =   rho_ww*(gama-1)*(et_ww - 0.5*(u_ww^2 + v_ww^2));
        a_ww = sqrt( gama*P_ww/rho_ww );
        T_ww = P_ww/(R*rho_ww);
        MuWW =  mu_ref*(T_ww/Tref)^(3/2)*( (Tref+Sconst)/(T_ww+Sconst)
);

        if P_ww < 0
            disp('pressure negative')
        end

        rho_ss = QSS(1);
        u_ss =   QSS(2)/QSS(1);
        v_ss =   QSS(3)/QSS(1);
        et_ss =  QSS(4)/QSS(1);
        P_ss =   rho_ss*(gama-1)*(et_ss - 0.5*(u_ss^2 + v_ss^2));
        a_ss = sqrt( gama*P_ss/rho_ss );
        T_ss = P_ss/(R*rho_ss);
        MuSS =  mu_ref*(T_ss/Tref)^(3/2)*( (Tref+Sconst)/(T_ss+Sconst)
);

        if P_ss < 0
            disp('pressure negative')
        end

        [Fiphj] =
upwindedFluxesAUSMplus(rho(i),u(i),v(i),a(i),P(i),et(i),rho_e,u_e,v_e,a
_e,P_e,et_e,gama,1);
        [Gijph] =
upwindedFluxesAUSMplus(rho(i),u(i),v(i),a(i),P(i),et(i),rho_n,u_n,v_n,a
_n,P_n,et_n,gama,2);
        [Fimhj] =
upwindedFluxesAUSMplus(rho_w,u_w,v_w,a_w,P_w,et_w,rho(i),u(i),v(i),a(i)
,P(i),et(i),gama,1);
        [Gijmh] =
upwindedFluxesAUSMplus(rho_s,u_s,v_s,a_s,P_s,et_s,rho(i),u(i),v(i),a(i)
,P(i),et(i),gama,2);
```

```
        dF_dx = (Fiphj - Fimhj)/(dx(i)/2);
        dG_dy = (Gijph - Gijmh)/(dy(i)/2);

        Qf(:,i) = Q(:,i) - dt * (dF_dx + dG_dy); % - dEv_dx_mls -
dFv_dy_mls);

        rho(i) = Qf(1,i);
        u(i) =   Qf(2,i)/Qf(1,i);
        v(i) =   Qf(3,i)/Qf(1,i);
        et(i) =  Qf(4,i)/Qf(1,i);
        intern_e =   et(i) - 0.5*(u(i)^2 + v(i)^2);
        T(i) = (gama-1)*intern_e/R;
        P(i) = rho(i)*R*T(i);
        Vmag(i) = sqrt( u(i)^2 + v(i)^2 );
        M(i) = Vmag(i)/a(i);
        a(i) = sqrt( gama*P(i)/rho(i) );


    end
    conv_eq(:,n) = sum(abs(Qf-Q),2);
    conv_max = max(conv_eq,[],2);
    conv_eq(:,n) = conv_eq(:,n)./conv_max;

    Q = Qf;
    %Solid iteration

    %           converge_Tsolid = 1;
    %           while converge_Tsolid > 0.01
    %               k = 1;
    %               for i = 1:nume_solid
    %                   for j = 1:NBsolid(i)
    %
    %                       if i == 3 || i == 6
    %                           Tm_solid(k) = T_solid(k);
    %                       elseif i == 1 || i == 2
    %                           Tm_solid(k) = T_solid(k) +
(kcond/k_solid)*(offset_s/offset)*(T(k)-Tm(k));
    %                           Tm(k) = Tm_solid(k);
    %                       else
    %                           Tm_solid(k) = 300;
    %                       end
    %                       k = k+1;
    %                   end
    %               end
    %
    %               for i = 1:length(x_solid)
    %
    %                   d2T_solid_dx2 =
Tall_solid(index_solid{i})*eta_dx2_c{i}';
    %                   d2T_solid_dy2 =
Tall_solid(index_solid{i})*eta_dy2_c{i}';
```

```
    %
    %                   T_solid_F(i) = T_solid(i) +
thdiff*0.01*(d2T_solid_dx2 + d2T_solid_dy2);
    %
    %               end
    %               converge_Tsolid = sum(abs(T_solid_F - T_solid));
    %               T_solid = T_solid_F;
    %
    %               Tall_solid = [Tm_solid,T_solid];
    %           end

    k=1;
    for i = 1:nume;
        for j = 1:NB(i)

            if i == 1 || i == 2

                vm(k) = 0; %-u(k)*nx(k)/ny(k);
                um(k) = 0; %sqrt(Vmag(k)^2-vm(k)^2);
                Tm(k) = T(k);
                Pm(k) = P(k);
                rhom(k) = rho(k);
                am(k) = sqrt(gama*Pm(k)/rhom(k));
                etm(k) = Pm(k)/(rhom(k)*(gama-1))+ 0.5*(um(k)^2 +
vm(k)^2);
                Vmagm(k) = sqrt( um(k)^2 + vm(k)^2 );
                Mm(k) = Vmagm(k)/am(k);

            elseif  i == 4 || i == 5

                vm(k) = -u(k)*nx(k)/ny(k);
                um(k) = sqrt(Vmag(k)^2-vm(k)^2);
                Tm(k) = T(k);
                Pm(k) = P(k);
                rhom(k) = rho(k);
                am(k) = a(k);
                etm(k) = Pm(k)/(rhom(k)*(gama-1))+ 0.5*(um(k)^2 +
vm(k)^2);
                Vmagm(k) = sqrt( um(k)^2 + vm(k)^2 );
                Mm(k) = Vmagm(k)/am(k);

            elseif i == 3

                Vn = um(k)*nx(k) + vm(k)*ny(k);

                rhom(k) = rho(k); % - dt*(um(k)+am(k)) *(rhom(k)-
rho(k))/offset;
                um(k) = u(k); % - dt*Vn *(um(k)-u(k))/offset;
                vm(k) = v(k); % - dt*Vn *(vm(k)-v(k))/offset;
                Pm(k) = P(k); % - dt*(um(k)+am(k)) *(Pm(k)-
P(k))/offset;
```

```
                etm(k) = et(k);% - dt*(um(k)+am(k)) *(etm(k)-
et(k))/offset;
                Tm(k) = T(k); %- dt*(um(k)+am(k)) *(Tm(k)-T(k))/offset;
                am(k) = a(k);

                Vmagm(k) = sqrt( um(k)^2 + vm(k)^2 );
                Mm(k) = Vmagm(k)/am(k);



            else


                rhom(k) = rhoinit;
                um(k) = uinit;
                vm(k) = vinit;
                Pm(k) = Pinit;
                etm(k) = etinit;
                am(k) = ainit;
                Tm(k) = Tinit;


                Vmagm(k) = sqrt( uinit^2 + vinit^2 );
                Mm(k) = sqrt( uinit^2 + vinit^2 )/ainit;
            end

            k = k+1;

        end
    end



    %     Tall_solid = [Tm_solid,T_solid];

    rhoall = [rhom;rho];
    uall = [um;u];
    vall = [vm;v];
    etall = [etm;et];
    Pall = [Pm;P];
    aall = [am;a];
    Mall = [Mm;M];
    Tall = [Tm;T];

    mu_s =  mu_ref*((Tall/Tref).^(3/2)).*( (Tref+Sconst)./(Tall+Sconst)
);

    Qall(1,:) = rhoall;
    Qall(2,:) = rhoall.*uall;
    Qall(3,:) = rhoall.*vall;
```

```
Qall(4,:) = rhoall.*etall;

if n == report*count || n == 1


    %         P_old = P;
    conv_eq(:,n)
    max(Pall)
    n

    count = count+1;


    figure(5)
    scatter(xall,yall,[],Mall,'filled');
    colorbar
    axis equal
    pause(0.001)

    figure(6)
    scatter(xall,yall,[],rhoall,'filled');
    colorbar
    axis equal
    pause(0.001)


    figure(7)
    scatter(xall,yall,[],Pall,'filled');
    colorbar
    axis equal
    pause(0.001)


    minmod_xbMat = cell2mat(minmod_xb);

    figure(8)
    scatter(x,y,[],minmod_xbMat(1 ,:)','filled');
    colorbar
    axis equal
    pause(0.001)

    minmod_ybMat = cell2mat(minmod_yb);
    figure(9)
    scatter(x,y,[],minmod_ybMat(1 ,:)','filled');
    colorbar
    axis equal
    pause(0.0001)

    figure(10)
    Nedge = length(xm)-1;
    C = [(1:(Nedge-1))' (2:Nedge)'; Nedge 1];
```

```
        tri = delaunayTriangulation(xall,yall,C);
        IO = isInterior(tri);
        h = trisurf(tri(IO,:),xall,yall,Pall);


        f = scatteredInterpolant(xall,yall,Pall(:),'natural');
        Z = f(X,Y);
        % h=surf(X,Y,Z);

        figure(11)
        plot(X(:,1),Z(:,1))
        hold all
        plot(CFD(:,1),CFD(:,2))
        hold off

        figure(12)
        semilogy(conv_eq(1,:))
        hold all
        semilogy(conv_eq(2,:))
        semilogy(conv_eq(3,:))
        semilogy(conv_eq(4,:))
        hold off


        figure(4)
        scatter(xall,yall,[],Tall,'filled');
        hold all
        scatter(xall_solid,yall_solid,[],Tall_solid,'filled');
        colorbar

        axis equal

    end
end
```

APPENDIX D

FUNCTIONS

```
function [xo,yo,rhoo,uo,vo,Po,eto,ao,index] =
subdomain_fast(rsub,x,y,xsub,ysub,rho,u,v,P,et,a)


xo = zeros(1,500);
yo = zeros(1,500);

rhoo = zeros(1,500);
uo = zeros(1,500);
vo = zeros(1,500);
Po = zeros(1,500);
eto = zeros(1,500);
ao = zeros(1,500);
index = zeros(1,500);


k=1;
for i = 1:length(x)

    for j = 1:length(xsub)-1

        tx = xsub(j+1) - xsub(j);
        ty = ysub(j+1) - ysub(j);
        t = sqrt(tx^2 + ty^2);

        nx = ty/t;
        ny= -tx/t;

        rx = xsub(j)-x(i);
        ry = ysub(j)-y(i);
%         r = sqrt( rx^2 + ry^2 );

        %             end

        %             for j = 1:length(xsub)-1
        %
        %                 dotproduct = rx(j)*rx(j+1) + ry(j)*ry(j+1);
        %                 theta2(j) = acos(dotproduct/(r(j)*r(j+1)));
        %
        %             end

        check = rx*nx+ry*ny;

        if check <= 0.0;

            flag = 0.0;
            break;

        else
```

```
            flag = 1.0;
        end
    end
    %             check = sum(theta2)/(2*pi);
    if flag == 1.0
        %                   xmax
        %                   xmin
        %                   ymax
        %                   ymin
        %                   x(i)
        %                   y(i)
        %                   check

        xo(k) = x(i);
        yo(k) = y(i);

        rhoo(k) = rho(i);
        uo(k) = u(i);
        vo(k) = v(i);
        Po(k) = P(i);
        eto(k) = et(i);
        ao(k) = a(i);
        index(k) = i;
        k = k + 1;
    end
end

xo(k:end) = [];
yo(k:end) = [];

rhoo(k:end) = [];
uo(k:end) = [];
vo(k:end) = [];
Po(k:end) = [];
eto(k:end) = [];
ao(k:end) = [];
index(k:end) = [];


end
```

```
function [RBF,CN] = call_RBF(xi,xj,yi,yj,rsub,sf)

RBF = zeros(length(xi),length(xj));
for i = 1:length(xi)
    for j = 1:length(xj)
```

```
        r = sqrt( (xi(i) - xj(j))^2 + (yi(i) - yj(j))^2 );

        RBF(i,j) = ( r^2 + sf^2 )^(1/2);

    end
end
CN = cond(RBF);
```

```
function [rhom,rhos,rhoint,rho,rhoall] =
intializeRho(rhoinit,Nsteps,nume,NB,NP,inletEdge,layers)

NBtot = sum(NB);
NBsn = sum(NB)*layers;


rhom = zeros(NBtot,1);
rhos = zeros(NBsn,1);
rhoint = zeros(NP,1);

%Initializing boundary nodes
k = 1;
for n = 1:Nsteps
    for i = 1:nume
        for j = 1:NB(i)
            if i == inletEdge
                rhom(k,n) = rhoinit;
            else
                rhom(k,n) = rhoinit;
            end
            k = k+1;
        end
    end
    k=1;
end

%Initializing shadow nodes
k=1;
for n = 1:Nsteps
    for p = 1:layers
        for i = 1:nume
            for j = 1:NB(i)
                rhos(k,n) = rhoinit;
                k = k+1;
            end
        end
    end
    k=1;
```

```
end

%Initializing Interior Points
for n = 1:Nsteps
    for i = 1:NP
        rhoint(i,n) = rhoinit;
    end
end

rhoall = [rhom;rhos;rhoint];

rho = [rhos;rhoint];




function [um,us,uint,u,uall] =
intializeU(uinit,Nsteps,nume,NB,NP,inletEdge,layers)

NBtot = sum(NB);
NBsn = sum(NB)*layers;


um = zeros(NBtot,1);
us = zeros(NBsn,1);
uint = zeros(NP,1);

%Initializing boundary nodes
k = 1;
for n = 1:Nsteps
    for i = 1:nume
        for j = 1:NB(i)
            if i == inletEdge
                um(k,n) = uinit;
            else
                um(k,n) = uinit;
            end
            k = k+1;
        end
    end
    k=1;
end

%Initializing shadow nodes
k=1;
for n = 1:Nsteps
    for p = 1:layers
        for i = 1:nume
            for j = 1:NB(i)
```

```
                    us(k,n) = uinit;
                    k = k+1;

            end
        end
    end
    k=1;
end

%Initializing Interior Points
for n = 1:Nsteps
    for i = 1:NP
        uint(i,n) = uinit;
    end
end

uall = [um;us;uint];

u = [us;uint]
```

```
function [vm,vs,vint,v,vall] =
intializeV(vinit,Nsteps,nume,NB,NP,inletEdge,layers)

NBtot = sum(NB);
NBsn = sum(NB)*layers;


vm = zeros(NBtot,1);
vs = zeros(NBsn,1);
vint = zeros(NP,1);
%Initializing boundary nodes
k = 1;
for n = 1:Nsteps
    for i = 1:nume
        for j = 1:NB(i)
            if i == inletEdge
                vm(k,n) = vinit;
            else
                vm(k,n) = 0;
            end
            k = k+1;
        end
    end
    k=1;
end

%Initializing shadow nodes
```

```
k=1;
for n = 1:Nsteps
    for p = 1:layers
        for i = 1:nume
            for j = 1:NB(i)
                    vs(k,n) = 0;
                    k = k+1;
            end
        end
    end
    k=1;
end

%Initializing Interior Points
for n = 1:Nsteps
    for i = 1:NP
        vint(i,n) = 0;
    end
end

vall = [vm;vs;vint];

v = [vs;vint];
```

```
function [Pm,Ps,Pint,P,Pall] =
intializeP(Pinit,Nsteps,nume,NB,NP,inletEdge,layers)


NBtot = sum(NB);
NBsn = sum(NB)*layers;


Pm = zeros(NBtot,1);
Ps = zeros(NBsn,1);
Pint = zeros(NP,1);

%Initializing boundary nodes
k = 1;
for n = 1:Nsteps
    for i = 1:nume
        for j = 1:NB(i)
            if i == inletEdge
                Pm(k,n) = Pinit;
            else
                Pm(k,n) = Pinit;
            end
            k = k+1;
```

```
        end
    end
    k=1;
end

%Initializing shadow nodes
k=1;
for n = 1:Nsteps
    for p = 1:layers
        for i = 1:nume
            for j = 1:NB(i)
                Ps(k,n) = Pinit;
                k = k+1;
            end
        end
    end
    k=1;
end

%Initializing Interior Points
for n = 1:Nsteps
    for i = 1:NP
        Pint(i,n) = Pinit;
    end
end

Pall = [Pm;Ps;Pint];

P = [Ps;Pint];




function [etm,ets,etint,et,etall] =
intializeEt(etinit,Nsteps,nume,NB,NP,inletEdge,layers)

NBtot = sum(NB);
NBsn = sum(NB)*layers;
etm = zeros(NBtot,1);
ets = zeros(NBsn,1);
etint = zeros(NP,1);

%Initializing boundary nodes
k = 1;
for n = 1:Nsteps
    for i = 1:nume
        for j = 1:NB(i)
            if i == inletEdge
                etm(k,n) = etinit;
            else
```

```
                etm(k,n) = etinit;
            end
            k = k+1;
        end
    end
    k=1;
end

%Initializing shadow nodes
k=1;
for n = 1:Nsteps
    for p = 1:layers
        for i = 1:nume
            for j = 1:NB(i)
                    ets(k,n) = etinit;
                    k = k+1;
            end
        end
    end
        k=1;
end

%Initializing Interior Points
for n = 1:Nsteps
    for i = 1:NP
        etint(i,n) = etinit;
    end
end

etall =[etm;ets;etint];

et = [ets;etint];




function [am,as,aint,a,aall] =
intializeSoS(ainit,Nsteps,nume,NB,NP,inletEdge,layers)
NBtot = sum(NB);
NBsn = sum(NB)*layers;


am = zeros(NBtot,1);
as = zeros(NBsn,1);
aint = zeros(NP,1);

%Initializing boundary nodes
k = 1;
for n = 1:Nsteps
    for i = 1:nume
```

```
        for j = 1:NB(i)
            if i == inletEdge
                am(k,n) = ainit;
            else
                am(k,n) = ainit;
            end
            k = k+1;
        end
    end
    k=1;
end

%Initializing shadow nodes
k=1;
for n = 1:Nsteps
    for p = 1:layers
        for i = 1:nume
            for j = 1:NB(i)

                    as(k,n) = ainit;
                    k = k+1;

            end
        end
    end
    k=1;
end

%Initializing Interior Points
for n = 1:Nsteps
    for i = 1:NP
        aint(i,n) = ainit;
    end
end

aall = [am;as;aint];
a = [as;aint];




function [RBF,CN] = call_RBFwPoly(xi,xj,yi,yj,rsub,sf,xc,yc)

RBF = zeros(length(xi),length(xj));
for i = 1:length(xi)
    for j = 1:length(xj)

        r = sqrt( (xi(i) - xj(j))^2 + (yi(i) - yj(j))^2 )/rsub;

        RBF(i,j) = ( r^2 + sf^2 )^(1/2);
```

```
    end
end
CN = cond(RBF);

NP = 3;

for i = 1:length(xi)

    P(i,:) = [1,(xi(i)-xc)/rsub,(yi(i)-yc)/rsub];
%,xi(i)*yi(i),xi(i)^2,yi(i)^2];

end


PP = P';
zeroMat = zeros(NP,NP);

P = [P;zeroMat];

RBF_P = [RBF;PP];

RBF = [RBF_P,P];



function
[RBF_dx,RBF_dy,RBF_dx2,RBF_dy2,RBF_dx3,RBF_dy3,RBF_dxdy,RBF_laplace] =
derivativesPoly(xi,xj,yi,yj,sf,rsub,xc,yc)

Ni = length(xi);
Nj = length(xj);
NP = 3;

RBF = zeros(Ni,Nj);
RBF_dx = zeros(Ni,Nj);
RBF_dy = zeros(Ni,Nj);
RBF_dx2 = zeros(Ni,Nj);
RBF_dy2 = zeros(Ni,Nj);
RBF_dxdy = zeros(Ni,Nj);

for i = 1:length(xi)
    for j = 1:length(xj)

        r = sqrt( ( xi(i) - xj(j) )^2 + ( yi(i) - yj(j) )^2 )/rsub ;


        RBF(i,j) = sqrt( (r^2 + sf^2) );
```

```
        RBF_dx(i,j) = (xi(i)-xj(j))/(rsub^2*sqrt( r^2 + sf^2 ));

        RBF_dy(i,j) = (yi(i)-yj(j))/(rsub^2*sqrt( r^2 + sf^2 ));

        RBF_dx2(i,j) = 1/(rsub^2*sqrt( (r^2 + sf^2 ))) ...
            - (xi(i)-xj(j))^2/( rsub^4*(r^2 + sf^2 )^(3/2));

        RBF_dy2(i,j) = 1/(rsub^2*sqrt( (r^2 + sf^2 ))) ...
            - (yi(i)-yj(j))^2/(rsub^4*( r^2 + sf^2 )^(3/2));

        RBF_dx3(i,j) = 3*(xi(i)-xj(j))/(rsub^4*(r^2+sf^2)^(3/2)) ...
             * ( (xi(i)-xj(j))^2/(rsub^2*(r^2 + sf^2 )) - 1);

        RBF_dy3(i,j) = 3*(yi(i)-yj(j))/(rsub^4*(r^2+sf^2)^(3/2)) ...
             * ( (yi(i)-yj(j))^2/(rsub^2*(r^2 + sf^2 )) - 1);

        RBF_dxdy(i,j) = -((xi(i)-xj(j))*(yi(i)-yj(j)))/(( r^2 + sf^2
)^(3/2));

        RBF_laplace = RBF_dx2 + RBF_dy2;

    end
end

for i = 1:Ni

    P(i,:) = [1, (xi(i)-xc)/rsub, (yi(i)-yc)/rsub];

end

RBF_P = [RBF,P];

for i = 1:Ni

    dP_dx(i,:) = [0, 1/rsub, 0];
    dP_dx2(i,:) = [0, 0, 0];
    dP_dx3(i,:) = [0, 0, 0];

    dP_dy(i,:) = [0, 0, 1/rsub];
    dP_dy2(i,:) = [0, 0, 0];
    dP_dy3(i,:) = [0, 0, 0];

    dP_dxdy(i,:) = [0, 0, 0];

end
%
RBF_dx = [RBF_dx,dP_dx];

RBF_dx2 = [RBF_dx2,dP_dx2];
```

```
RBF_dx3 = [RBF_dx3,dP_dx3];

RBF_dy = [RBF_dy,dP_dy];

RBF_dy2 = [RBF_dy2,dP_dy2];

RBF_dy3 = [RBF_dy3,dP_dy3];

RBF_dxdy = [RBF_dxdy,dP_dxdy];



function [Fp,Fm,Gp,Gm,F,G,Qc] = upwindedFluxes(rho,u,v,a,gama)

%Steger-Warming FVS
if u >= a
    Aeig1p = u;
    Aeig2p = u+a;
    Aeig3p = u-a;

    Aeig1m = 0;
    Aeig2m = 0;
    Aeig3m = 0;
else
    Aeig1p = u;
    Aeig2p = u+a;
    Aeig3p = 0;

    Aeig1m = 0;
    Aeig2m = 0;
    Aeig3m = u-a;
end

if v >= a
    Beig1p = v;
    Beig2p = v+a;
    Beig3p = v-a;

    Beig1m = 0;
    Beig2m = 0;
    Beig3m = 0;
else
    Beig1p = v;
    Beig2p = v+a;
    Beig3p = 0;

    Beig1m = 0;
    Beig2m = 0;
    Beig3m = v-a;
end
```

```
alpha = 2*(gama-1)*Aeig1p + Aeig2p + Aeig3p;

Fp = rho/(2*gama)*[ alpha;
                    alpha*u + a*(Aeig2p - Aeig3p);
                    alpha*v;
                    alpha*(u^2+v^2)/2 + u*a*(Aeig2p-Aeig3p) + ...
                        a^2*(Aeig2p + Aeig3p)/(gama-1)];


alpha = 2*(gama-1)*Aeig1m + Aeig2m + Aeig3m;

Fm = rho/(2*gama)*[ alpha;
                    alpha*u + a*(Aeig2m - Aeig3m);
                    alpha*v;
                    alpha*(u^2+v^2)/2 + u*a*(Aeig2m-Aeig3m) + ...
                        a^2*(Aeig2m + Aeig3m)/(gama-1)];

alpha = 2*(gama-1)*Beig1p + Beig2p + Beig3p;

Gp = rho/(2*gama)*[ alpha;
                    alpha*u;
                    alpha*v + a*(Beig2p - Beig3p);
                    alpha*(u^2+v^2)/2 + v*a*(Beig2p-Beig3p) + ...
                        a^2*(Beig2p + Beig3p)/(gama-1)];

alpha = 2*(gama-1)*Beig1m + Beig2m + Beig3m;

Gm = rho/(2*gama)*[ alpha;
                    alpha*u;
                    alpha*v + a*(Beig2m - Beig3m);
                    alpha*(u^2+v^2)/2 + v*a*(Beig2m-Beig3m) + ...
                        a^2*(Beig2m + Beig3m)/(gama-1)];

F = Fp + Fm;
G = Gp + Gm;
Qc = rho;



function [Fp,Fm,Gp,Gm,F,G,Qc] = upwindedFluxesVL(rho,u,v,a,gama)

%Van Leer FVS

for i = 1:length(rho)

Mx = u(i)/a(i);
My = v(i)/a(i);

fp1VL = rho(i)/(4*a(i))*(u(i)+a(i))^2;
```

```
fm1VL = -rho(i)/(4*a(i))*(u(i)-a(i))^2;

gp1VL = rho(i)/(4*a(i))*(v(i)+a(i))^2;
gm1VL = -rho(i)/(4*a(i))*(v(i)-a(i))^2;


if Mx < 1 && Mx > -1
    Fp(:,i) = fp1VL * [ 1; ((gama-1)*u(i)+2*a(i))/gama; v(i) ; v(i)^2/2
+ ((gama-1)*u(i) + 2*a(i))^2 / (2*(gama^2-1))];
    Fm(:,i) = fm1VL * [ 1; ((gama-1)*u(i)-2*a(i))/gama; v(i) ; v(i)^2/2
+ ((gama-1)*u(i) - 2*a(i))^2 / (2*(gama^2-1))];

elseif Mx >= 1

        Fp(:,i) = [ rho(i)*u(i); rho(i)*a(i)^2*(gama*Mx^2+1)/gama;
rho(i)*u(i)*v(i); (rho(i)*a(i)^2*u(i))/(gama*(gama-
1))+0.5*rho(i)*u(i)*(u(i)^2+v(i)^2)+rho(i)*u(i)*a(i)^2/gama];
        Fm(:,i) = zeros(4,1);

elseif Mx <= -1

        Fp(:,i) = zeros(4,1);
        Fm(:,i) = [ rho(i)*u(i); rho(i)*a(i)^2*(gama*Mx^2+1)/gama;
rho(i)*u(i)*v(i); (rho(i)*a(i)^2*u(i))/(gama*(gama-
1))+0.5*rho(i)*u(i)*(u(i)^2+v(i)^2)+rho(i)*u(i)*a(i)^2/gama];

end

if My < 1 && My > -1

    Gp(:,i)= gp1VL * [ 1; u(i) ; ((gama-1)*v(i)+2*a(i))/gama ; u(i)^2/2
+ ((gama-1)*v(i) + 2*a(i))^2 / (2*(gama^2-1))];
    Gm(:,i) = gm1VL * [ 1; u(i) ; ((gama-1)*v(i)-2*a(i))/gama ;
u(i)^2/2 + ((gama-1)*v(i) - 2*a(i))^2 / (2*(gama^2-1))];

elseif My >= 1

    Gp(:,i) = [ rho(i)*v(i); rho(i)*u(i)*v(i);
rho(i)*a(i)^2*(gama*My^2+1)/gama; (rho(i)*a(i)^2*v(i))/(gama*(gama-
1))+0.5*rho(i)*v(i)*(u(i)^2+v(i)^2)+rho(i)*v(i)*a(i)^2/gama];
    Gm(:,i) = zeros(4,1);

elseif My <= -1

    Gp(:,i) = zeros(4,1);
    Gm(:,i) = [ rho(i)*v(i); rho(i)*u(i)*v(i);
rho(i)*a(i)^2*(gama*My^2+1)/gama; (rho(i)*a(i)^2*v(i))/(gama*(gama-
1))+0.5*rho(i)*v(i)*(u(i)^2+v(i)^2)+rho(i)*v(i)*a(i)^2/gama];

end
```

```
% if u >= a
%     Aeig1p = u;
%     Aeig2p = u+a;
%     Aeig3p = u-a;
%
%     Aeig1m = 0;
%     Aeig2m = 0;
%     Aeig3m = 0;
% else
%     Aeig1p = u;
%     Aeig2p = u+a;
%     Aeig3p = 0;
%
%     Aeig1m = 0;
%     Aeig2m = 0;
%     Aeig3m = u-a;
% end
%
% if v >= a
%     Beig1p = v;
%     Beig2p = v+a;
%     Beig3p = v-a;
%
%     Beig1m = 0;
%     Beig2m = 0;
%     Beig3m = 0;
% else
%     Beig1p = v;
%     Beig2p = v+a;
%     Beig3p = 0;
%
%     Beig1m = 0;
%     Beig2m = 0;
%     Beig3m = v-a;
% end
%
% alpha = 2*(gama-1)*Aeig1p + Aeig2p + Aeig3p;
%
% Fp = rho/(2*gama)*[ alpha;
%                     alpha*u + a*(Aeig2p - Aeig3p);
%                     alpha*v;
%                     alpha*(u^2+v^2)/2 + u*a*(Aeig2p-Aeig3p) + ...
%                         a^2*(Aeig2p + Aeig3p)/(gama-1)];
%
%
% alpha = 2*(gama-1)*Aeig1m + Aeig2m + Aeig3m;
%
% Fm = rho/(2*gama)*[ alpha;
%                     alpha*u + a*(Aeig2m - Aeig3m);
%                     alpha*v;
```

```
%                       alpha*(u^2+v^2)/2 + u*a*(Aeig2m-Aeig3m) + ...
%                           a^2*(Aeig2m + Aeig3m)/(gama-1)];
%
% alpha = 2*(gama-1)*Beig1p + Beig2p + Beig3p;
%
% Gp = rho/(2*gama)*[ alpha;
%                       alpha*u;
%                       alpha*v + a*(Beig2p - Beig3p);
%                       alpha*(u^2+v^2)/2 + v*a*(Beig2p-Beig3p) + ...
%                           a^2*(Beig2p + Beig3p)/(gama-1)];
%
% alpha = 2*(gama-1)*Beig1m + Beig2m + Beig3m;
%
% Gm = rho/(2*gama)*[ alpha;
%                       alpha*u;
%                       alpha*v + a*(Beig2m - Beig3m);
%                       alpha*(u^2+v^2)/2 + v*a*(Beig2m-Beig3m) + ...
%                           a^2*(Beig2m + Beig3m)/(gama-1)];
end


F = Fp + Fm;
G = Gp + Gm;
Qc = rho;



function [F] =
upwindedFluxesAUSMplus(rhoL,uL,vL,aL,pL,etL,rhoR,uR,vR,aR,pR,etR,gama,f
lag,mach)


beta  = 0.1875;
alpha = 0.125;

beta  = 0.0;
alpha = 0.0;

if flag == 1;

    htL =  aL^2/(gama-1) + 0.5*(uL^2 + vL^2);
    htR =  aR^2/(gama-1) + 0.5*(uR^2 + vR^2);

    astarL = sqrt(2*(gama-1)*htL/(gama+1));
    astarR = sqrt(2*(gama-1)*htR/(gama+1));

    atempL = astarL^2/max(astarL,abs(uL));
    atempR = astarL^2/max(astarR,abs(uR));

    ahalf = min(atempL,atempR);
```

```
else
    htL =  aL^2/(gama-1) + 0.5*(uL^2 + vL^2);
    htR =  aR^2/(gama-1) + 0.5*(uR^2 + vR^2);

    astarL = sqrt(2*(gama-1)*htL/(gama+1));
    astarR = sqrt(2*(gama-1)*htR/(gama+1));

    atempL = astarL^2/max(astarL,abs(vL));
    atempR = astarL^2/max(astarR,abs(vR));

    ahalf = min(atempL,atempR);
end

 ahalf = 0.5*(aL+aR);


if flag == 1
    ML = uL/ahalf;
    MR = uR/ahalf;
else
    ML = vL/ahalf;
    MR = vR/ahalf;
end


if abs(ML) <= 1
    MLp = 0.25*(ML+1)^2 + beta*(ML^2-1)^2;
else
    MLp = 0.5*(ML+abs(ML));
end

if abs(ML) <= 1
    PLp = 0.25*(ML+1)^2*(2-ML)+alpha*ML*(ML^2-1)^2;
else
    PLp = 0.5*(ML+abs(ML))/ML;
end

if abs(MR) <= 1
    MRm = -0.25*(MR-1)^2 - beta*(MR^2-1)^2;
else
    MRm = 0.5*(MR-abs(MR));
end

if abs(MR) <= 1
    PRm = 0.25*(MR-1)^2*(2+MR)- alpha*MR*(MR^2-1)^2;
else
    PRm = 0.5*(MR-abs(MR))/MR;
end

Mh = MLp + MRm;
```

```
Mhp = 0.5*( Mh + abs(Mh));
Mhm = 0.5*( Mh - abs(Mh));

ph = PLp*pL + PRm*pR;

% if Mh >= 0

    Fcp = [rhoL; rhoL*uL; rhoL*vL; (rhoL*etL + pL)];

% else

    Fcm = [rhoR; rhoR*uR; rhoR*vR; (rhoR*etR + pR)];

% end

if flag == 1
    p = [0; ph; 0; 0];
else
    p = [0; 0; ph; 0];
end


F = ahalf*(Mhp*Fcp + Mhm*Fcm) + p;
```

```
function [LS_f,LS_dx, LS_dy, LS_dx2, LS_dy2,LS_dxdy] =
MLSoperator(xo,yo,x,y,xi,yi,rsub,x_dist,y_dist)

NP = 6;
NF = length(x);
C = zeros(NP,NP);
P = zeros(NP,NF);
theta = zeros(NF,1);
d = zeros(NF,1);
const = 0.001;
for i = 1:NF
    d(i) = sqrt((x(i)-xi)^2 + (y(i)^2-yi)^2);
    theta(i) = 1/(d(i)^2 + const^2);
end

for i = 1:NP
    for j = 1:NP
        for k = 1:NF

            xp = (x(k)-xo)/x_dist;
            yp = (y(k)-yo)/y_dist;
```

```
            phi = [1; xp; yp; xp^2; xp*yp; yp^2]; %; xp^3; xp^2*yp;
xp*yp^2; yp^3];

%             phi = [1; x(k); y(k); x(k)^2; x(k)*y(k); y(k)^2; x(k)^3;
x(k)^2*y(k); x(k)*y(k)^2; y(k)^3];

            C(i,j) = C(i,j) + phi(i)*phi(j);

        end
    end
    for k = 1:NF

        xp = (x(k)-xo)/x_dist;
        yp = (y(k)-yo)/y_dist;

        phi = [1; xp; yp; xp^2; xp*yp; yp^2]; %; xp^3; xp^2*yp;
xp*yp^2; yp^3];

%         phi = [1; x(k); y(k); x(k)^2; x(k)*y(k); y(k)^2; x(k)^3;
x(k)^2*y(k); x(k)*y(k)^2; y(k)^3];
        P(i,k) = phi(i);
    end
end

dxp_dx = 1/x_dist;
dyp_dy = 1/y_dist;

xp = (xo-xo)/x_dist;
yp = (yo-yo)/y_dist;


phi = [1; xp; yp; xp^2; xp*yp; yp^2]; %; xp^3; xp^2*yp; xp*yp^2; yp^3];

dphi_dx = [0; dxp_dx; 0; 2*xp*dxp_dx; yp*dxp_dx; 0]; %; 3*xp^2*dxp_dx;
2*xp*yp*dxp_dx; yp^2*dxp_dx; 0];
d2phi_dx2 = [0; 0; 0; 2*dxp_dx^2; 0; 0]; %; 6*xo; 2*yo; 0; 0];

dphi_dy = [0; 0; dyp_dy; 0; xp*dyp_dy; 2*yp*dyp_dy]; %; 0; xp^2*dyp_dy;
2*xp*yp*dyp_dy; 3*yp^2*dyp_dy];
d2phi_dy2 = [0; 0; 0; 0; 0; 2*dyp_dy^2]; %;  0;  0; 2*xo; 6*yo];

dphi_dxdy = [0; 0; 0; 0; dxp_dx*dyp_dy; 0];

% dphi_dx = [0; 1; 0; 2*xo; yo; 0; 3*xo^2; 2*xo*yo; yo^2; 0];
% d2phi_dx2 = [0; 0; 0; 2; 0; 0; 6*xo; 2*yo; 0; 0;];
%
% dphi_dy = [0; 0; 1; 0; xo; 2*yo; 0; xo^2; 2*xo*yo; 3*yo^2];
%d2phi_dy2 = [0; 0; 0; 0; 0; 2; 0; 0; 2*xo; 6*yo];

LS_f = phi'*(C\P);
```

```
LS_dx = dphi_dx'*(C\P);
LS_dy = dphi_dy'*(C\P);

LS_dx2 = d2phi_dx2'*(C\P);
LS_dy2 = d2phi_dy2'*(C\P);

LS_dxdy = dphi_dxdy'*(C\P);

end
```

LIST OF REFERENCES

[1] G. R. Liu, Mesh Free Methods, Boca Raton, FL: CRC Press, 2003.

[2] T. Belytschko, Y. Y. Lu and L. Gu, "Element -free Galerkin Methods," *International Journal Numerical Methods in Engineering,* pp. 229-256, 1994.

[3] T. Atluri and T. Zhu, "A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics," *Computational Mech,* pp. 117-127, 1998.

[4] J. M. Melenk and I. Babuska, "The partion of unity finite element method: basic theory and applications," *Computational Methods in Appl Mech Eng,* pp. 289-314, 1996.

[5] E. J. Kansa, "Multiquadrics - a scattered data approximation scheme with applications to computational fluid dynamics - I. Surface approximations and partial derivative estimates.," *Computers & Mathematics with Applications,* vol. 19, no. 8/9, pp. 127-145, 1990.

[6] E. J. Kansa, "Multiquadrics - A Scattered Data Approximation Scheme with Applications to Compuational Fluid Dynamics-II," *Computers Math. Appic.,* vol. 19, no. 8/9, pp. 147-161, 1990.

[7] E. J. Kansa and Y. C. Hon, "Circumventing the Ill-Conditioning Problem with Multiquadric Radial Basis Funtions: Applications to Elliptic Partial Differential Equations," *Computers and Mathematics with Applications ,* vol. 39, pp. 123-137, 2000.

[8] B. Nayroles, G. Touzot and P. Villon, "Generalizing the finite element method: diffuse approximation and diffuse elements," *Computation Mech,* pp. 307-318, 1992.

[9] C. A. Duarte and J. T. Oden, "Hp clouds- an hp meshless method," *Numer Method in Partial Diff Eq,* pp. 673-705, 1996.

[10] W. K. Liu, S. Jun, S. Li, J. Adee and T. Belytschko, "Reproducing kernel particle methods for structural dynamics," *International Journal of Numerical Methods in Eng,* pp. 1655-1679, 1995.

[11] E. A. Divo and A. J. Kassab, "Iterative Domain Decomposition Meshless Method Modeling of Incompressible Flows and Conjugate Heat Transfer," *Engineering Analysis,* vol. 30, no. 6, pp. 465-478, 2006.

[12] E. A. Divo and A. J. Kassab, "An Efficient Localized RBF Meshless Method for Fluid Flow and Conjugate Heat Transfer," *ASME Journal of Heat Transfer,* vol. 129, pp. 124-136, 2007.

[13] E. Divo and A. J. Kassab, "Localized Meshless Modeling of Natural Convective Viscous Flows," *Numerical Heat Transfer,* vol. 53, pp. 487-509, 2008.

[14] R. L. Hardy, "Multiquadric equations of topography and other irregular surfaces," *Journal of Geophysical Research,* vol. 76, no. 8, pp. 1905-1915, 1971.

[15] R. L. Hardy, "Research results in the application of multiquadric equations to surverying and mapping problems," *Surveying and Mapping,* vol. 35, pp. 321-332, 1975.

[16] R. Franke, "Scattered Data Interpolation: Tests of Some Methods," *Mathematics of Computation,* vol. 38, no. 157, pp. 181-200, 1982.

[17] N. Dyn, D. Levin and S. Rippa, "Numerical Procedures for Surface Fitting of Scattered Data by Radial Basis Funtions," *SIAM Journal of Sci. Stat. Computat.,* vol. 7, no. 2, pp. 639-659, 1986.

[18] A. Karageorghis, C. S. Chen and Y.-S. Smyrlos, "A matrix decomposition RBF algorithm: Approximation of functions and their derivatives," *Applied Numerical Mathematics ,* vol. 57, pp. 304-319, 2007.

[19] B. Sarler and R. Vertnik, "Local Explicit Radial Basis Funtion Collocation Method for Diffusion Problems," *Computational Math,* pp. 1269-1282, 2005.

[20] B. Sarler , T. Tran-Cong and C. S. Chen, "Meshfree Direct and Indirect Local Radial Basis Function Collocation Formulations for Transport Phenomena," *Boundary Elements XVII,* pp. 417-428, 2005.

[21] G. Yao, B. Sarler and C. S. Chen, "A comparison of three explicit local meshless methods using radial basis funtions," *Engineering Analysis with Boundary Elements,* vol. 35, pp. 600-609, 2011.

[22] S. u. Islam, R. Vertnik and B. Sarler, "Local radial basis funtion collocation method along with explicit time stepping for hyperbolic partial differential equations," *Applied Numerical Mathematics,* vol. 67, pp. 136-151, 2011.

[23] S.-u. Islam, B. Sarler and R. Vertnik, "Radial basis function collocation method for the numerical solution of the two-dimensional transient nonlinear coupled Burgers' equation," *Applied Mathematical Modelling ,* vol. 36, pp. 1148-1160, 2012.

[24] Y.-C. Hon, B. Sarler and D.-f. Yun, "Local radial basis function collocation method for solving thermo-driven fluid-flow problems with free surface," *Engineering Analysis with Boundary Elements,* vol. 57, pp. 2-8, 2015.

[25] B. Mavric and B. Sarler, "Local radial basis function collocation method for linear thermoelasticity in two dimensions," *Inter. Journal of Numerical Methods for Heat and Fluid Flow,* vol. 25, no. 6, pp. 1488-1510, 2015.

[26] B. Mavric and B. Sarler, "Application of the RBF collocation method to transient coupled thermoelasticity," *Inter. Journal of Numerical Methods for Heat and Fluid Flow,* vol. 27, no. 5, pp. 1064-1077, 2017.

[27] G. Kosec and B. Sarler, "Solution of a low Prandtl number convection benchmark by a local meshless method," *Inter. Journal of Numerical Methods for Heat and Fluid Flow,* vol. 23, no. 1, pp. 189-204, 2013.

[28] B. Sarler, "A Radial Basis Function Collocation Approach in Computational Fluid Dynamics," *Computer Modeling in Science & Engineering,* vol. 7, no. 2, pp. 185-193, 2005.

[29] G. Yao and B. Sarler, "Assessment of global and local meshless methods based on collocation with radial basis functions for parabolic partial differential equations in three dimensions," *Engineering Analysis and Boundary Elements,* vol. 36, no. 11, pp. 1640-1648, 2012.

[30] S. Gerace, K. Erhart, A. Kassab and E. Divo, ""A Model-Integrated Localized Collocation Meshless Method for Large Scale Three Dimensional Heat Transfer Problems," *Engineering Analysis,* vol. 45, pp. 2-19, 2014.

[31] S. Gerace, K. Erhart, E. Divo and A. Kassab, "Adaptively Refined Hybrid FDM/Meshless Scheme with Applications to Laminar and Turbulent Flows," *CMES Computer Modeling in Engineering and Science,* vol. 81, no. 1, pp. 35-68, 2011.

[32] K. Erhart , A. J. Kassab and E. Divo, "An Inverse Localized Meshless Technique for the Determination of Non-Linear Heat Generation Rates in Living Tissues," *International Journal of Heat and Fluid Flow,* vol. 18, no. 3, pp. 401-414, 2008.

[33] J. Kelly, E. Divo and A. J. Kassab, "Numerical Solution of the Two-Phase Incompressible Navier-Stokes Equations using a GPU-Accelerated Meshless Method Engineering Analysis with Boundary Elements," *Engineering Analysis,* vol. 40C, pp. 36-49, 2014.

[34] A. H. D. Cheng, M. A. Golberg, E. J. Kansa and G. Zammito, "Exponential Convergence and H-c Multiquadric Collocation Method for Partial Differenctial Equations," *Numerical Methods In Partial Differential Equations,* pp. 571-594, 2003.

[35] W. R. Madych, "Miscellaneous Error Bounds for Multiquadric and Related Interpolations," *Computers Math. Applic.,* vol. 24, no. 12, pp. 121-138, 1992.

[36] M. D. Buhmann, S. Dinew and E. Larsson, "A note on radial basis funtion interpolant limits," *IMA Journal of Numerical Analysis,* vol. 30, pp. 543-554, 2010.

[37] W. R. Madych, "Bounds on Multivariate Polynomials and Exponential Error Estimates for Multiquadric Interpolation," *Journal of Approximation Theory,* vol. 70, pp. 94-114, 1992.

[38] E. J. Kansa and P. Holoborodko, "On the ill-conditioned nature of C∞ RBF strong collocation," *Engineering Analysis with Boundary Elements,* vol. 78, pp. 26-30, 2017.

[39] C. M. C. Roque and A. J. M. Ferreira, "Numerical Experiments on Optimal Shape Parameters for Radial Basis Functions," *Numerical Methods for Partial Differential Equations,* pp. 1-14, 2009.

[40] G. B. Wright and B. Fornberg, "Scattered node compact finite difference-type formulas generated from radial basis funtions," *Journal of Computational Physics,* vol. 212, pp. 99-123, 2006.

[41] E. Larsson and B. Fornberg, "A Numerical Study of Some Radial Basis Function Based Solution Methods for Elliptic PDEs," *Computers and Mathematics with Applications,* vol. 46, pp. 891-902, 2003.

[42] J. G. Wang and G. R. Liu, "On the optimal shape parameters of radial basis functions used for 2-D meshless methods," *Comput. Methods Appl. Mech. Engrg.,* vol. 191, pp. 2611-2630, 2002.

[43] G. E. Fasshauer and J. G. Zhang, "On choosing "optimal" shape parametes for RBF approximations," *Numer Algor,* vol. 45, pp. 345-368, 2007.

[44] R. Schaback, "Error Estimates and Condition Numbers for Radial Basis Function Interpolation," *Advances in Computational Mathematics,* vol. 3, no. 3, pp. 251-264, 1995.

[45] T. A. Driscoll and B. Fornberg, "Interpolation in the Limit of Increasingly Flat Radial Basis Function," *Computers & Mathematics with Applications,* vol. 43, no. 3-5, pp. 413-422, 2002.

[46] M. Buhmann and N. Dyn, "Spectral Convergence of Multiquadric Interpolation," in *Proceeding of the Edinburgh Mathematical Society*, Edinburgh, 1993.

[47] E. Larsson and B. Fornberg, "Theoretical and Compurtational Aspects of Multivariate Interpolation with Increasingly Flat Radial Basis Funtions," *Computers and Math. with Appl.,* vol. 49, pp. 103-130, 2005.

[48] S. Rippa, "An algorithm for selecting a good value for the parameter c in radial basis function interpolation," *Advances in Computational Mathematics,* vol. 11, pp. 193-210, 1999.

[49] B. Fornberg and G. Wright, "Stable Computation of Multiquadric Interpolants for All Values of the Shape Parameter," *Int. J. Comp. and Math with Applications,* vol. 48, pp. 853-867, 2004.

[50] B. Fornberg, E. Larsson and N. Flyer, "Stable Computations with Gaussian Radial Basis Functions," *SIAM J. Sci. Comput.,* vol. 33, no. 2, pp. 869-892, 2011.

[51] E. J. Kansa and R. E. Carlson, "Improved Accuracy of Multiquadric Interpolation using Variable Shape Parameters," *Computers Math. Applic. ,* vol. 24, no. 12, pp. 99-120, 1992.

[52] R. Vertnik and B. Sarler, "Solution of Incompressible Turbulent Flow by a Mesh-Free Methdo," *Computer modeling in Engineering and Sciences,* vol. 44, no. 1, pp. 65-95, 2009.

[53] M. A. Golberg, C. S. Chen and S. R. Karur, "Improved multiquadric approximations for partial differential equations," *Engineering Analysis with Boundary Element Methods,* vol. 18, pp. 9-17, 1996.

[54] R. E. Carlson and T. A. Foley, "The Parameter R2 in Multiquadric Interpolation," *Computers Math Applic,* vol. 21, no. 9, pp. 29-42, 1991.

[55] R. E. Carlson and T. A. Foley, "Interpolation of Track Data with Radial Basis Methods," *Computer Math Applic.,* vol. 24, no. 12, pp. 27-34, 1992.

[56] J.-H. Jung and V. R. Durante, "An iterative adaptive multiquadric radial basis function method for the detection of local jump discontinuities," *Applied Numerical Methods,* vol. 59, pp. 1449-1466, 2008.

[57] F. Benkhaldoun, A. Halassi, D. Ouazar, M. Seaid and A. Taik, "Slope limiters for radial basis function applied to conservation laws with discontinuous flux funtion," *Enigneering Analysis with Boundary Elements,* vol. 66, pp. 49-65, 2016.

[58] J. Li and C. S. Chen, "Some observations on unsymmetric radial basis function collocation methods for convection-diffusion problems," *International Journal for Numerical Methods in Engineering,* vol. 57, pp. 1085-1094, 2003.

[59] D. Stevens, H. Power, M. Lees and H. Morvan, "The use of PDE centres in the local RBF Hermitian method for 3D convective-diffusion problems," *Journal of Computational Physics,* vol. 228, pp. 4606-4624, 2009.

[60] Y. L. Chan, C. T. Shen, C. T. Wu and D. L. Young, "A novel upwind-based local radial basis funtion differential quadrature," *Computers & Fluids,* vol. 89, pp. 157-166, 2014.

[61] P. L. Roe, "Approximate Riemann solvers, parameter vectors and difference schemes," *Journal of Computational Physics,* vol. 43, no. 357, 1981.

[62] J. L. Steger and R. F. Warming, "Flux vector spliting of the inviscid gas dynamics equations with applications to finite-difference methods," *Journal of Computational Physics,* vol. 40, no. 2, pp. 263-293, 1981.

[63] B. van Leer, "Flux-vector splitting for the Euler equations," *ICASE Report 82-30,* 1982.

[64] M.-S. Liou, "A Sequel to AUSM: AUSM+," *Journal of Computational Physics,* vol. 129, no. 0256, pp. 346-382, 1996.

[65] M. Harris, A. Kassab and E. Divo, "Application of an RBF blending interpolation method to problems with shocks," *Computer Assisted Methods in Engineering and Science,* vol. 22, pp. 229-241, 2015.

[66] M. Harris, E. Kassab and E. Divo, "An RBF interpolation blending scheme for effective shock-capturing," *International Journal of Computational Methods and Experimental Measurements,* vol. 5, no. 3, pp. 281-292, 2017.

[67] M. F. Harris, A. J. Kassab and E. A. Divo, "Meshless Method using a RBF Interpolation Scheme for Inviscid Compressible Flows," in *Second Thermal and Fluids Engineering Conference*, Las Vegas, NV, 2017.

[68] M. F. Harris, A. J. Kassab and E. A. Divo, "Viscous Compressible Flow Soluiton using RBF Blended Interpolation," in *3rd Thermal and Fluids Engineering Conference (TFEC)*, Fort Lauderdale, FL, 2018.

[69] A. Harten, "A high resolution scheme for the computation of weak solutions of hyperbolic conservation laws," *Journal of Compuational Physics,* vol. 49, no. 357, 1983.

[70] X. Liu, S. Osher and T. Chan, "Weighted Essentially Nonoscillatory Schemes," *Journal of Computational Physics,* vol. 115, pp. 200-212, 1994.

[71] S. Osher and F. Solomon, "Upwind difference schemes for hyperbolic systems of conservation laws," *Mathematics of Computation,* vol. 38, no. 158, pp. 339-374, 1982.

[72] D. W. Pepper, A. J. Kassab and E. A. Divo, Introduction to Finite Element, Boundary Element, and Meshless Methods, ASME Press, 2014.

[73] P. Lancaster and K. Salkauskas, "Surfaces Generated by Moving Least Squares Methods," *Mathematics of Computation,* vol. 37, no. 155, pp. 141-158, 1981.

[74] C. B. Laney, Computational Gasdynamics, Cambridge University Press, 1998.

[75] T. J. Chung, Computational Fluid Dynamics, New York, NY: Cambridge University Press, 2002.

[76] F. Moukalled, L. Mangani and M. Darwish, The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM and Matlab, Switzerland : Springer International Publishing, 2016.

[77] K. A. Hoffmann and S. T. Chiang, Computational Fluid Dynamics Volume I, Wichita: Engineering Education System, 2000.

[78] K. A. Hoffmann and S. T. Chiang, Computational Fluid Dynamics Volume II, Wichita: Engineering Education System, 2000.

[79] R. H. Pletcher, J. C. Tannehill and D. A. Anderson, Computational Fluid Mechanics and Heat Transfer, CRC Press Taylor & Francis Group, LLC, 2013.

[80] R. W. MacCormack, Numerical Computation of Compressible and Viscous Flow, Reston, VA: American Institute of Aeronautics and Astronautics, 2014.

[81] C. Hirsch, Numerical Computation of Internal and External Flows Volume 2: Computational Methods for Inviscid and Viscous Flows, John Wiley and Sons, 1984.

[82] M.-S. Liou and C. J. Steffen, Jr., "A New Flux Splitting Scheme," *Journal of Computational Physics,* vol. 107, pp. 23-39, 1993.

[83] J. D. Anderson, Jr., Computational Fluid Dynamics: The Basics with Applications, McGraw-Hill, Inc,, 1995.

[84] C. A. J. Fletcher, Computational Techniques for Fluid Dynamics Vol II, Berlin: Springer-Verlag, 1988.

[85] R. Vertnik and B. Sarler, "Simulation of continuous casting of steel by a meshless technique," *Int. J. Cast. Met. Res.,* vol. 22, pp. 311-313, 2009.

[86] K. Mramor, R. Vertnik and B. Sarler, "Simulation of laminar backward facing step flow under magnetic field with explicit local radial basis function collocation method," *Engineering Analysis with Boundary Elements,* vol. 49, pp. 37-47, 2014.

[87] N. Talat, B. Mavric, G. Belsak, V. Hatic, S. Bajt and B. Sarler, "Development of Meshless Phase Field Method for Two-Phase flow," *International Journal of Multiphase Flow,* vol. 108, pp. 169-180, 2018.

[88] N. Talat, B. Mavric, V. Hatic, S. Bajt and B. Sarler, "Phase field simulation of Rayleigh-Taylor instability with a meshless method," *Engineering Analysis with Boundary Elements,* vol. 87, pp. 78-89, 2018.

[89] V. Hatic, B. Mavric and B. Sarler, "Simulation of a macrosegregation benchmark with a meshless diffuse approximate method," *Inter. Journal of Numerical Methodas for Heat & Fluid Flow,* vol. 4, pp. 361-380, 2018.