MASTER'S THESIS

ACTIVE FLUX SCHEMES FOR HYPERBOLIC CONSERVATION LAWS

by

TANIYA KAPOOR

&

ABHISHEK CHANDRA

SAU
SOUTH ASIAN UNIVERSITY

DEPARTMENT OF MATHEMATICS
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SOUTH ASIAN UNIVERSITY
NEW DELHI, INDIA

MAY, 2019

ACTIVE FLUX SCHEMES FOR HYPERBOLIC CONSERVATION LAWS

A Master Thesis

Submitted to

SOUTH ASIAN UNIVERSITY

for the award of the degree of

Master of Science

in

Applied Mathematics

by

Taniya Kapoor & Abhishek Chandra

Under the Supervision of

Dr. Navnit Jha

Department of Mathematics
Faculty of Mathematics and Computer Science

South Asian University

New Delhi - India

CERTIFICATE

*This is to certify that thesis entitled "***ACTIVE FLUX SCHEMES FOR HYPERBOLIC CONSERVATION LAWS***" submitted by* **Taniya Kapoor & Abhishek Chandra** *to the South Asian University, New Delhi, India for the award of the degree of* **Master of Science**, *is a record of the partial fulfillment of the award of the degree of M.Sc.(Applied Mathematics), carried by them under my supervision and guidance. The thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.*

The results contained in this thesis have not been submitted either in part or full to any other university or institution for the award of any degree or diploma.

Place : New Delhi

Date : May 2019

Dr. Navnit Jha

Department of Mathematics
Faculty of Mathematics and Computer Science
South Asian University
New Delhi - India

Prof. Dr. Pranab k. Muhuri

Professor & Dean (FMCS)
Department of Computer Science
South Asian University
New Delhi - India

Declaration

We hereby declare that the masters project entitled **"ACTIVE FLUX SCHEMES FOR HYPERBOLIC CONSERVATION LAWS"** *Which is being submitted for the award of the degree of Master of Science to the South Asian University, is a record of theoretical survey with the insight to work out some new problems, by us, under the supervision of* **Dr. Navnit Jha.** *The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.*

Place : New Delhi

Date : May 2019

Taniya Kapoor **(SAU/AM(M)/2017/15)**
Abhishek Chandra **(SAU/AM(M)/2017/16)**
M.Sc Applied Mathematics
South Asian University
New Delhi

Dr. Navnit Jha
Department of Mathematics
Faculty of Mathematics and Computer Science
South Asian University
New Delhi - India

Dedicated to our parents and professors

Acknowledgement

It is a pleasant aspect that we now have the opportunity to express our gratitude to those who inspired and helped us in bringing this project to the present form.

I would first like to thank my thesis supervisor **Dr. Navnit Jha** *for giving me the opportunity to work with him and for his constant support and motivation.*

Words will always be less to express our heartfelt gratitude, our most sincere thanks to **Dr. Harish Kumar (IIT Delhi)**, *for his continuous support and motivation in our master's study and related research. We thank Dr. Kumar for his several important suggestions, advice and fruitful discussions that contributed to this project immensely. we would like to thank Dr. Kumar for being a true mentor and guiding in our tough times. For proofreading this thesis we are indebted to him.*

We would also like to thank our teachers Dr. Q.M. Danish Lohani, Prof R.K. Mohanty, Dr. Saroj Kumar Sahani, Dr. Deepa Sinha, Dr. Pankaj Jain, Dr. Jagdish Bansal and Dr. Kapil Sharma for their insightful comments and encouragement throughout our academic session.

We feel deeply indebted to our parents for their love, support and constant help and guidance in achieving our goal and who succeeded to develop in us a spirit of devotion and perseverance since our childhood which has encouraged us throughout the preparation of the project.

Finally, we would like to express our deepest gratitude to God who made all the things possible.

Taniya Kapoor , Abhishek Chandra
M.Sc Applied Mathematics
South Asian University
New Delhi

Abstract

ACTIVE FLUX SCHEMES

FOR

HYPERBOLIC CONSERVATION LAWS

by

Taniya Kapoor & Abhishek Chandra

Chair : Dr. Navnit Jha

The numerical solution of hyperbolic conservation laws has been the subject of extraordinary research during the last two decades or so. Conservation Laws model many exciting problems in the physical, biological, engineering and social sciences. In most of the cases, it is impossible to obtain analytic solutions for them. Even if the solutions exist, their evaluation is often a laborious task. Numerical methods are therefore useful almost exclusively to treat such problems.

The advances in computing and the presentation of scientific programming software have encouraged the figuring of different numerical techniques that are utilized. Consequently, many linear and nonlinear systems of hyperbolic conservation laws that were previously unsolved can now be unraveled by using suitable mathematical strategies.

This project is primarily concerned with the study of the advancement of active flux (AF) Schemes, quite a new class of methods for solving hyperbolic conservation laws, which are closely associated with the finite volume (FV) schemes. In FV schemes, a numerical inter-cell flux is required, which we can call as a passive flux as it is updated only from the conserved values. Whereas in the AF scheme the edge and vertex values are updated and evolved independently from the conserved cell average quantities. Later, the interface values *are used to calculate the fluxes that conservatively update the cell averages.*

The AF scheme uses continuous parabolic reconstruction with Lagrange basis functions inside each Cartesian cell with a cubic bubble function to maintain conservation in two dimensions which ensures third order spatial accuracy. Every reconstruction is local to a single element, which makes AF schemes very compact and far more suitable for random, unstructured meshes.

This project presents an implementation of AF on the one space dimension and the two-dimensional Cartesian grids for linear advection and linear acoustic equations. Sample problems of one dimension and two dimensions linear advection and linear acoustics have also been implemented on Cartesian grids. The AF scheme efficiently achieves third order accuracy for all the equations, for all courant number ranging between zero and one.

Additionally, the coding structure has been discussed in detail for the 2D implementation of AF schemes. Thorough specifics of the application of spherical means has been reviewed and is supplemented with the MATLAB *and* MATHEMATICA *codes*

AMS (MOS) Subject Classification: 35L65, 35L45, 35L03, 35L40, 65M08.

Keywords: Active flux; Advection equation; Acoustic equations; Hyperbolic conservation laws.

Taniya Kapoor , Abhishek Chandra
M.Sc Applied Matheatics
South Asian University
New Delhi

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction to Conservation Laws

This thesis is primarily concerned with the *numerical solution of hyperbolic conservation laws.* So, it might be a good idea to get acquainted at this point of time with the general notion of conservation laws, where they arise from, what is their conservative form and this section ends with some popular examples of this class.

Mathematically, conservation laws [2] is the name given to *first order hyperbolic partial differential equations in both space and time*. Physically, conservation laws state that particular measurable property of an isolated physical system does not change as the system evolves over time. Exact conservation laws include conservation of energy, conservation of momentum and conservation of electric charges, etc.

Now, we present a brief introduction from where the conservation laws originate. Conservation laws are a subset of a class called Balance laws. Balance laws are time-dependent. Consider a domain Ω and a quantity of interest $\boldsymbol{U}$ defined for all points $\boldsymbol{x} \in \boldsymbol{\mathcal{R}}$. This quantity of interest $\boldsymbol{U}$ is originally a physical quantity [3].

The time rate of change of $\boldsymbol{U}$ in any fixed sub-domain $\omega \subset \Omega$ is equal to the total amount of $\boldsymbol{U}$ produced or destroyed inside ω and the flux of $\boldsymbol{U}$ across the boundary $\partial\omega$. The observation can be mathematically written as,

$$\frac{\partial \int_{\omega} \boldsymbol{U} d\boldsymbol{x}}{\partial t} = -\int_{\partial \omega} \boldsymbol{F}.\nu d\sigma(\boldsymbol{x}) + \int_{\omega} \boldsymbol{S} d\boldsymbol{x}$$

These are called as the Balance laws. The first integral on the right hand side corresponds to the flux term and the second integral corresponds to the source or sink term. When this source or sink term becomes zero, that is, when the quantities tend to be conserved at all evaluation steps, then the equation obtained is,

$$\frac{\partial \int_{\omega} \boldsymbol{U} d\boldsymbol{x}}{\partial t} = -\int_{\partial \omega} \boldsymbol{F}.\nu d\sigma(\boldsymbol{x})$$

using integration by parts (or the Gauss divergence theorem) on the surface integral to obtain

$$\frac{\partial \int_{\omega} \boldsymbol{U} d\boldsymbol{x}}{\partial t} + \int_{\omega} div(\boldsymbol{F}) d\boldsymbol{x} = 0$$

This equation holds for all sub-domains $\omega \subset \Omega$. choosing an infinitesimal ω to obtain the differential equation

$$\boldsymbol{U}_t + div(\boldsymbol{F}) = 0 \qquad \forall(\boldsymbol{x}, t) \in (\Omega, \boldsymbol{\mathcal{R}}^{+}) \tag{1.1}$$

This equation is termed as conservation law where the only change in $\boldsymbol{U}$ comes from the quantity entering or leaving the domain of interest.

Examples of Conservation Laws

Now we extend our general discussion of conservation laws to the ones that have been discussed in this text. They are namely

- Linear advection equation also known as the transport equation in one and two space dimensions.
- System of linear Acoustic equations in one and two space dimensions.

Other Examples

Other examples which have not been discussed in the text but which are very important include the

- Burger′s equations
- Euler equations
- Equations of gas dynamics
- System of magnetic induction equations

The linear advection equations and the Burger′s equation are the starting point of any work related to linear and nonlinear conservation laws respectively. This project deals with only linear conservation equations, so, a thorough study of the advection equation has been done. A similar study with nonlinear equations can be started with Burger′s equation

1.2 Finite Volume Method

Finite volume schemes for hyperbolic conservation laws have been used for decades now. we present just its notion which shall help to discriminate between the AF schemes (about which we present a brief description in the next section) and FVM.

Like in any numerical approximation method, the first step in FVM is also to discretise the computational domain both in space $[x_L, x_R]$ as well as in time $[0, t]$. Computational cells are called as control volumes $\boldsymbol{C}_j = [x_{j-1/2}, x_{j+1/2})$, The FVM uses control volumes instead of the mesh points $\boldsymbol{x_j}$. The time levels are obtained by $t^n = n\Delta t$. Unlike finite difference method (FDM), FVM is based on using the perspective of cell averages u_j^n [3], [4], [5]. Where the cell averages are defined as

$$u_j^n \approx \frac{1}{\Delta x}\int_{x_j-1/2}^{x_j+1/2} u(x,t^n)dx \tag{1.2}$$

At each time level t^n, u_j^n is our main entity of scrutiny. For a conventional finite volume scheme the cell averages are updated by

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}(F_{j+1/2}^n - F_{j-1/2}^n) \tag{1.3}$$

where $F_{j+1/2}$ is the flux through the $(j+\frac{1}{2})^{th}$ interface at n^{th} time level. the prime element of any scheme is a proficient approximation of this inter-cell flux. lately, one method to obtain a numerical flux is by using a Riemann solver [6].

Considering the IVP of a general conservation law

$$u_t + f(u(x,t))_x = 0 \tag{1.4}$$

with initial data

$$u(x,0) = u_0(x) \tag{1.5}$$

a piecewise constant polynomial [3], [4], [5]

$$\overline{u}(x,t^n) = u_j^n, \qquad x_{j-\frac{1}{2}} < x < x_{j+\frac{1}{2}}$$

is considered.

The above process is called the reconstruction step, which results in the formation of Riemann problems across all the boundaries, that are to be solved numerically in time, One important requirement is that average

$$\frac{1}{\Delta x}\int_{x_j-1/2}^{x_j+1/2} \overline{u}(x,t^n)dx \approx u_j^n$$

over each such cell.

The contemporary method to increase the order of accuracy of the scheme in FVM are using the ENO (Essentially non-oscillatory schemes) and WENO (Weighted Essentially non-oscillatory schemes) along with the Runge Kutta time stepping. Without going much into any further details of ENO/WENO and FVM , we now shift our focus to AF schemes.

1.3 Introduction to Active Flux Schemes

In this section, AF schemes are introduced and the main points of differences from the finite volume schemes are discussed. The AF method differs from the usual finite volume scheme in the number of degrees of freedom (DOF) that both posses [7]. AF uses two DOF per element in one dimension and three DOF in per element in two dimensions. The additional degrees of freedom in the AF method is used to calculate the numerical flux across cells, which is the primary difference between FVM and AF scheme. without discussing any specific equation, we present a general structure of the AF method at this point.

The scheme starts with the internal continuous reconstruction which is piecewise parabolic in nature. It is well known that the accuracy of the vertex update and the average flux calculation is determined by the representation of the solution within the cell. AF chooses parabolic reconstruction (with a cubic bubble function to maintain conservation in two dimensions) which assures the scheme to be exact for quadratic data and hence gives third order spatial accuracy. Then for updates, different strategies are used according to the physics of the problem, like for example in linear advection - characteristics backtracing is preferred, whereas for linear acoustics - spherical means over disks are preferred for two space dimensional problems and spherical means over spheres are considered for three-dimensional problems. These will be discussed in detail in further sections.

After the vertex and edge, values have been calculated at $(n+1/2)^{th}$ and $(n+1)^{th}$

level, then a higher order numerical integration is applied and fluxes are calculated. AF scheme although being discrete in nature, that is the conserved cell averages are calculated at the n^{th} and $(n+1)^{th}$ level only and not at the $(n+1/2)^{th}$ level. But, edge and vertex values are calculated at the $(n+1/2)^{th}$ level also, which makes the scheme staggered in nature.

Remark: One important point to notice is that the reconstructions are piecewise parabolic in nature, which ensures third order accuracy in space. Hence to reach the same accuracy in time, the edge and vertex value at half time steps are also calculated. At last space-time, Simpson rule is used as a quadrature formula for numerical flux.

Now A general algorithm of active flux scheme is discussed further, to elaborate the understanding of modus operandi of active flux.

A General Algorithm of Active Flux

As shown in figure 1.1, a general algorithm for an AF flux scheme consists of three main steps that repeat itself at each evolution step. The first step is calculating the reconstructed cell average values from the given initial data. Once this is done, using these reconstructed values, the edge and vertex interface values are calculated at $(n+\frac{1}{2})^{th}$ and $(n+1)^{th}$ time step then using higher order numerical integration methods like the Simpson quadrature rules, intercell flux are calculated then with the help of equation the reconstructed values are calculated for the $(n+1)^{th}$ time step. $(n+1)^{th}$ level values are stored as nth level values are stored as an nth level value, and the process is repeated until the final time step.

1.4 Literature Review

Actor flux being quite a new class of method, we do not have a lot of literature about them. But, still the progress is rapid and the stature of the schemes is gaining

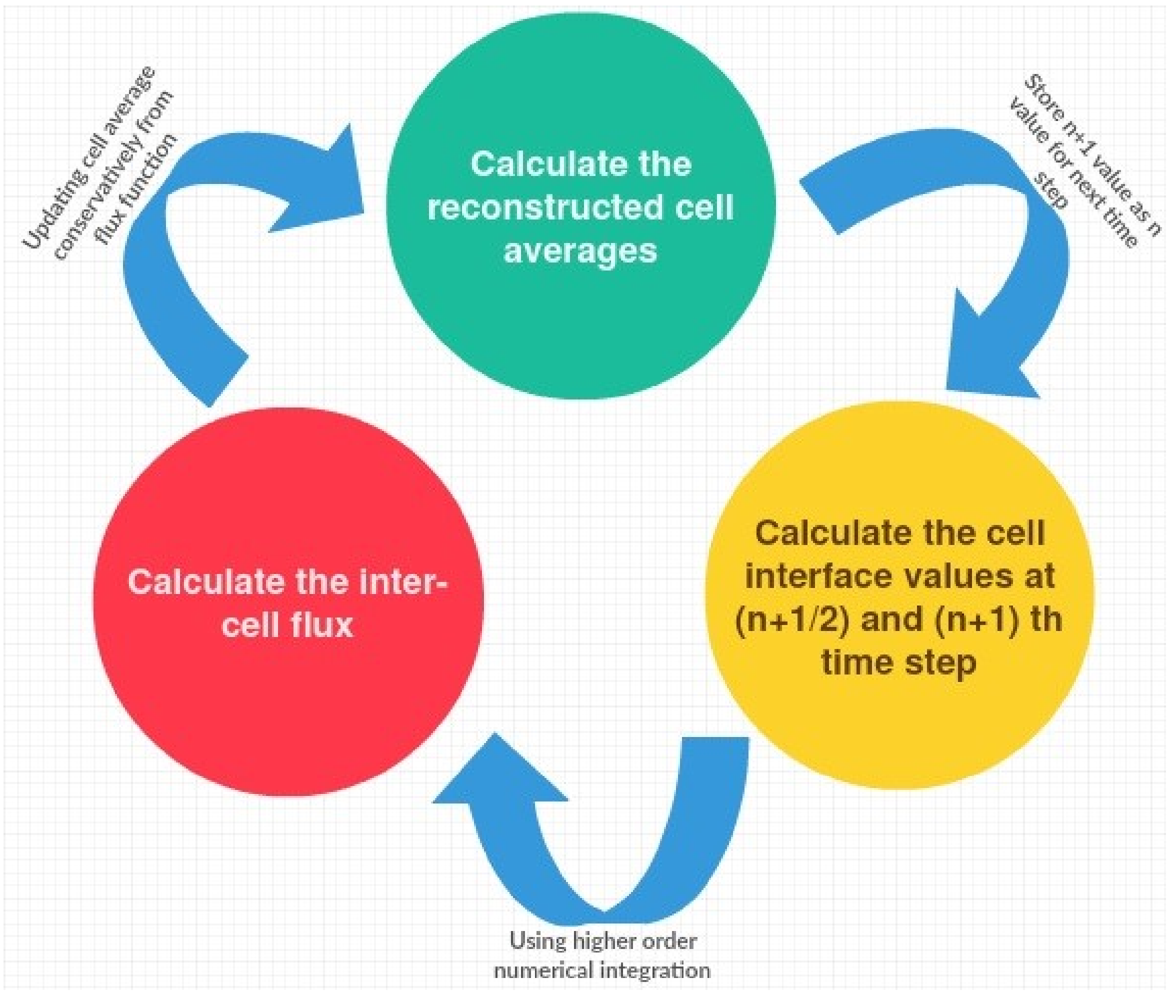

Calculate the reconstructed cell averages
Store n+1 value as n value for next time step
Calculate the cell interface values at (n+1/2) and (n+1) th time step
Using higher order numerical integration
Calculate the inter-cell flux
Updating cell average conservatively from flux function

Figure 1.1: A general algorithm of AF Scheme

importance slowly. Almost, all of the work previously has been carried out in the Department of Aerospace engineering, University of Michigan, USA under the supervision of Philip L. Roe. The works include dealing with the linear advection equation, Burgers equation, and Linear Acoustic equations on an unstructured triangular mesh in [7] [8] [9] [10]. Other works include testing of active flux schemes thoroughly on advection and acoustic problems in[11] and in[12] respectively. A latest joint work[13] has been carried out for active flux schemes on Cartesian grids. This thesis combines the ideas of [7] and [13] to extend the foundation of ideas laid by them.

1.5 Thesis Statement

This thesis is concerned with developing active flux schemes on Cartesian grids. Numerically solving the one-dimensional and two-dimensional problems of advection and acoustics, and achieving the desired order of accuracy is the prime focus of this project. The project also aims at simplifying the concept of spherical means on the Cartesian grid by listing the coding structure employed to obtain results.

1.6 Research Methodology

It is nearly impossible to carry out this project without the use of mathematical software for computing the results. MATLAB has been used to carry out the numerical mathematical calculations while MATHEMATICA has been employed to facilitate the symbolic mathematical simplifications.

1.7 Outline of the Thesis

This thesis has been divided into seven chapters and two important appendices. chapter 1 gives the introduction and discusses the tone of the project.

chapter 2 deals with the 1D Linear advection equation which is numerically simulated to obtain third order spatial accuracy by active flux scheme. The theory presented takes care of reference mapping and reconstruction. A numerical example is presented at the end of the chapter.

chapter 3 extends the concept of the 1D Linear advection equation and deals with the waves of negative advection speed as well. Strategy to decouple the equation and then solve the resulting equation has been disused. To sum up, the chapter numerical illustrations of simple and non-simple wave have been presented.

chapter 4 Is an important addition and it should be a pleasant experience to read the 2D coding structure in the Cartesian grid for active flux schemes.

chapter 5 Shifts the reader's attention to the two-dimensional framework of an advection equation. Along with the reference element and the mapping numerical example is presented. The chapter ends with an explanation to the abruptness shown by active flux scheme.

chapter 6 It is concerned with the system of two-dimensional acoustic equations. The theory of spherical means is discussed thoroughly and Morton's famous problem is solved.

chapter 7 Gives a concluding remark about the project and gives an insight into possible future work. The first two appendices have MATLAB and MATHEMATICA codes in them used for this project.

2 One Dimensional Linear Advection Equation

The most basic example of scalar hyperbolic conservation law is the one-dimensional linear advection equation. It is advisable and even customary to test and implement any new scheme or method on the linear advection equation as an explicit solution of this equation is known and proper analysis of any scheme can be done on the basis of how the scheme deals with linear advection.

2.1 The Equation

Scalar one-dimensional linear advection equation is given by the equation

$$u_t + au_x = 0 \tag{2.1}$$

First, let us understand this equation and its associated variables. The equation expresses the advection (transportation) of quantity u with advection speed of a. If a is positive, the quantity is said to be moving with positive speed and if a is negative the quantity is said to be transported with a negative pace. The quantity is only advected in the direction of a and other directions need not be taken care of.

The equation (2.1) must be complemented with an initial condition, like given by equation (1.5). Also, a proper boundary condition is to be supplied according to the physics of the problem, which will be explained further. AF scheme was first applied to the advection equation in [7]. Advection problem has also been dealt by AF in [11]

In linear advection as the information is advected in the direction depending upon the magnitude of $''a''$. The solution procedure used by AF is the back tracing of the characteristics curve. The theory of characteristics is explained in brief in later sections.

2.2 Anatomy of The Active Flux Schemes

In this section discuss the structure of AF schemes for 1D advection has been presented along with the data storage locations in one space dimension. The algorithm is given in the form of a flow chart to express the step by step procedure to deal with the problem.

Data Storage Locations in 1D Problems

AF uses more DOF than the usual FVM. In FVM the only storage locations are the cell centers, but in AF schemes the cell interfaces also end up being the data storage locations. As the FV method is staggered in nature, the interface values at half time steps are also recorded. But, AF method being fully discrete in nature, the conserved quantities are stored only at the n^{th} and $(n+1)^{th}$ level.

Figure 2.1 shows how data is stored efficiently in 1D. Star shows the location of the cell center and the circle shows the storage locations of edge or vertex values.

Algorithm For 1D Active Flux

Next, we proceed to discuss the algorithm for 1D advection to solve via AF that this text has followed. The problem-solving starts with discretizing the computational domain in space and time by following the CFL [14] condition. AF method is stable

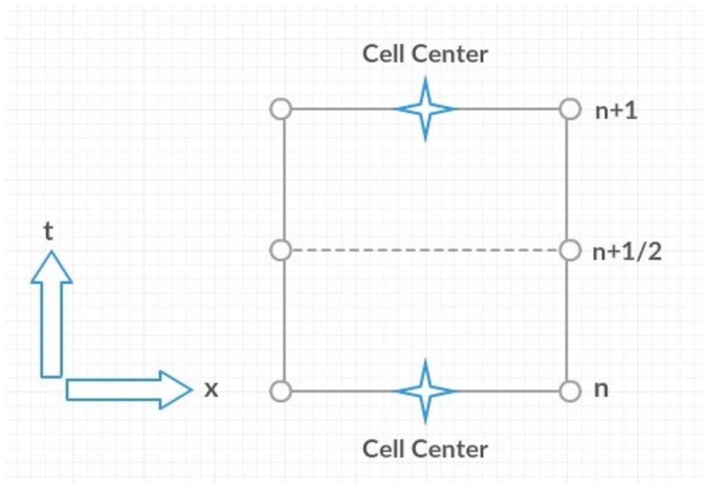


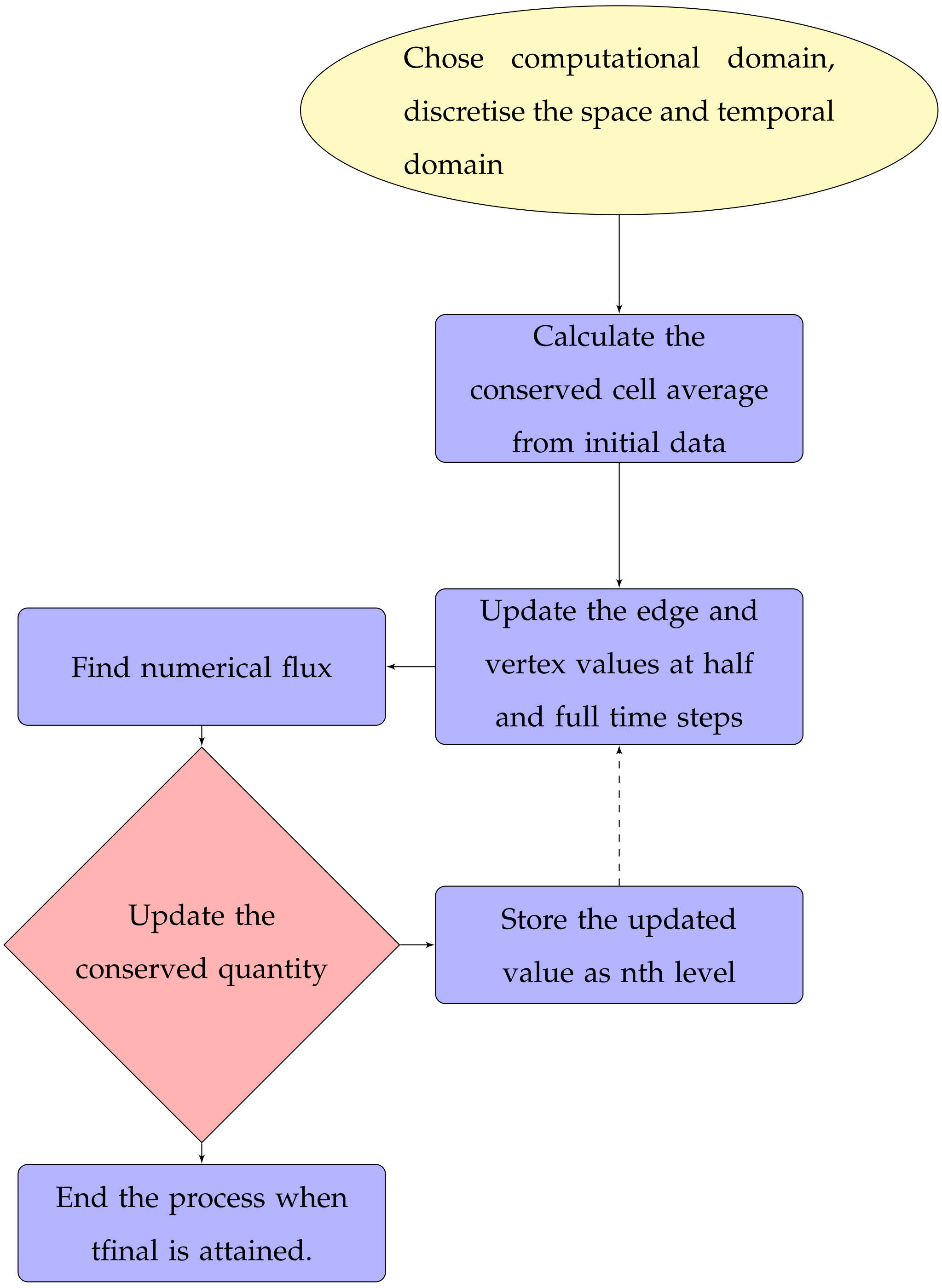
Figure 2.1: 1D Grid

for all Courant numbers (v) ranging from 0 to 1.

Next, a very important step to achieve third order accuracy is performed. Conserved cell average is calculated by using the parabolic reconstruction which is discussed in further section.

Next step is exclusive for AF and FVM does not use this kind of procedure, which is a calculation of the edge and vertex updates which are used up in calculating the numerical fluxes, whereas in FVM numerical flux like Roe's flux or the Enguist-Osher flux depends upon the conserved quantities.

In the semi-final step Conserved cell average is updated using the usual FVM update formula. Finally, a loop is run to reach through the end time, while storing the $(n+1)^{th}$ level values as n^{th} level values.


Chose computational domain, discretise the space and temporal domain
Calculate the conserved cell average from initial data
Update the edge and vertex values at half and full time steps
Find numerical flux
Update the conserved quantity
Store the updated value as nth level
End the process when tfinal is attained.

2.3 Reference Element and Mapping

We look for a general portrayal of the arrangement inside a given cell j. Throughout this thesis, the coordinates have been changed to reference coordinates for simplifying the formulas of basis functions. Working with non-dimensional coordinates facilitates the definition of arrangement inside an element. The cell coordinates are standardised with the end goal that $(j - 1/2)$ edge is at $\xi = 0$ and $(j + 1/2)$ edge is at $\xi = 1$

We then have the accompanying mapping from physical space to reference space given by

$$x = x_{j-\frac{1}{2}} + \xi \Delta x \tag{2.2}$$

$$\xi = \frac{x - x_{j-1/2}}{\Delta x} \tag{2.3}$$

As known, the solution of linear advection is constant along the characteristics (which are straight lines with inclination of $\frac{1}{a}$ on x-t plot). Hence, to know the solution at any interface coordinate ξ_i at time t, we can trace back the characteristics in time to a point where solution is known. This point will be referred as characteristics origin ξ_0. This is the theory that will be used to solve linear advection by AF scheme.

2.4 Internal Reconstruction

Unlike Van Leer's approach in [15], Eymann chose to follow the standard finite element construction inside each cell and represent the solution as the sum of basis functions.

$$u(\xi) = \sum_{i=1}^{3} c_i \phi_i(\xi) \tag{2.4}$$

Index 1 correspond to $(j - 1/2)^{th}$ interface. Index 2 correspond to midpoint. Index 3 correspond to $(j + 1/2)^{th}$ interface. For basis function standard one dimensional Lagrange basis function are used, which are valid for $\xi \in [0, 1]$. The coefficients and basis functions defining the reconstruction are listed in table 2.1

Table 2.1: Basis function and coefficient for 1D reconstruction

Index	c_i	ϕ_i
1	$u^n_{j-1/2}$	$(2\xi-1)(\xi-1)$
2	$\frac{1}{4}(6\overline{u}^n_j - u^n_{j-1/2} - u^n_{j+1/2})$	$4\xi(1-\xi)$
3	$u^n_{j+1/2}$	$\xi(2\xi-1)$

Now, we proceed to explain how these expression come and how to derive them [16]. First of all $u(\xi)$ is taken to be a quadratic polynomial,

$$u(\xi) = a\xi^2 + b\xi + c \tag{2.5}$$

then we find

$$u(0) = c$$

$$u(\frac{1}{2}) = \frac{a}{4} + \frac{b}{2} + c$$

$$u(1) = a + b + c$$

We also know that, $u(0) = u_{j-1/2}$; $u(\frac{1}{2}) = \overline{u}_j$; $u(1) = u_{j+1/2}$

This system of linear equations is easy to solve as the number of equations equal the number of variables. Thus after solving the system and re-arranging the terms we get the actual formulas and table as presented.

2.5 Updating the Edge and Vertex Values

In AF scheme the conserved cell average and edge, vertex values are independently updated. Also the edge and vertex values need not be conservative. Due to the linear nature of characteristics the following equation holds.

$$at = (\xi_i - \xi_0)\Delta x$$

$$\xi_0 = \xi_i - \frac{at}{\Delta x}$$

Therefore, to find the updated vertex value we only need to calculate $u(\xi_0)$. The average flux is determined by employing a sufficiently accurate numerical integration technique like Simpson's rule which is exact for quadratic data. For using this we require both ξ_0^{n+1} value at $t = \Delta t$ and $\xi_0^{n+1/2}$ value at $t = \Delta t/2$. Once we have them the average flux is given by

$$\overline{f} = \frac{1}{6}[f(u(\xi_i) + 4f(u(\xi_0^{n+1/2})) + f(u(\xi_0^{n+1}))] \tag{2.6}$$

for one dimensional linear advection, the characteristics origin depends upon the CFL number v.

$$v = \frac{a\Delta t}{\Delta x}$$

when the wave speed is positive the characteristics origin at $(n+1/2)^{th}$ and $(n+1)^{th}$ time level is given by

$$\xi_0^{n+1/2} = \xi_i - \frac{v}{2}$$

$$\xi_0^{n+1} = \xi_i - v$$

For positive wave speed right interface is considered, that is $\xi_i = 1$ substituting equation (13) in table 1 we obtain

$$u_{j+1/2}^{n+1} = v(3v-2)u_{j-1/2}^n + 6v(1-v)\overline{u}_j^n + (1-v)(1-3v)u_{j+1/2}^n \tag{2.7}$$

2.6 Calculating Numerical Flux and Time Step Marching

Next, after calculating the vertex updates, numerical flux are calculated using equation (13). For positive wave speed the flux obtained is

$$F_{j+1/2}^n = a[v(v-1)u_{j-1/2}^n + v(3-2v)\overline{u}_j^n + (1-v)^2 u_{j+1/2}^n] \tag{2.8}$$

The cell averages are then conservatively updated from the flux function by

$$\overline{u}_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}(F_{j+1/2}^n - F_{j-1/2}^n) \tag{2.9}$$

Substituting the flux function for left and right interface results in full update for conserved quantity.

$$\overline{u}_j^{n+1} = v^2(v-1)u_{j-3/2}^n + v^2(3-2v)\overline{u}_{j-1}^n + v(1-v)u_{j-1/2}^n + (1-v)^2(1+2v)\overline{u}_j^n - v(1-v)^2 u_{j+1/2}^n \tag{2.10}$$

After obtaining conserved values at $(n+1)^{th}$ level, the same procedure is followed to obtain values at next time step and the process continues till the final specified time.

We now have discussed all the steps, how to solve an advection problem in one dimension using the AF method.

We now present a numerical experiment carried out in [**?**], to ascertain the order of accuracy of the scheme.

All the plots have been compared with the exact solution of the advection problem [4]. Also $L_2(\overline{u})$ error plots have been plotted in the subsequent section.

2.7 Numerical Illustrations

In order to check the accuracy of AF method, first we must define the measure of error estimate. An error norm typically used to measure solution accuracy at given time level in FVM is $L_2(\overline{u})$ norm.

$$L_2(\overline{u}) = \left[\frac{1}{\Omega} \sum_{j=1}^{N} |\overline{u}_j \Omega_j - \overline{u}_j^{exact} \Omega_j| \right]^{1/2} \tag{2.11}$$

where N is total number of cells, $\sum_j^N \Omega_j = \Omega$ is the total volume of domain and $\overline{u}; \overline{u}^{exact}$ are the numerical and exact solution in the conservative variable $\overline{u}$.

To test the accuracy of AF scheme, we have taken a sinusoidal problem implemented along with the periodic boundary conditions . A sine wave initial condition is considered as shown in the equation

$$u_0(x) = \frac{1}{2\pi} \sin 2\pi x \tag{2.12}$$

The advection speed is set to be 1, that is $a = 1$ and the spatial domain is considered to be $x \in [0, 1]$. Numerical tests are presented at time $t = 1$ and 100 for 20 and 160 cells. For checking the order of accuracy the results are tabulated in table 2.2.

Table 2.2: Order Of accuracy Of 1D Linear Advection

Level	N	$L_2(\overline{u})$	order
1	20	$2.1363e-05$	–
2	40	$2.5482e-06$	3.06
3	80	$3.1047e-07$	3.02
4	160	$3.8311e-08$	3.03
5	320	$4.7579e-09$	3.00
6	640	$5.9278e-10$	3.00
7	1280	$7.3976e-11$	3.00
8	2560	$9.2394e-12$	3.00

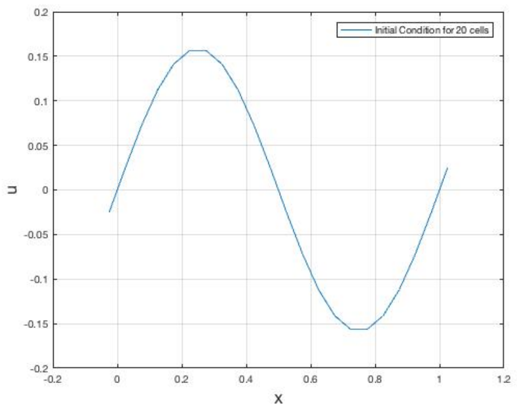


Figure 2.2: Initial Condition for *20* cells

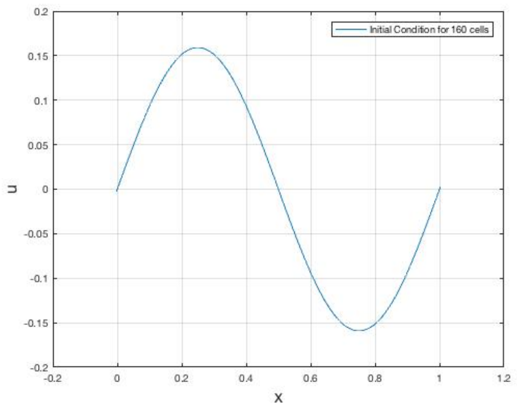


Figure 2.3: Initial Condition for *160* cells

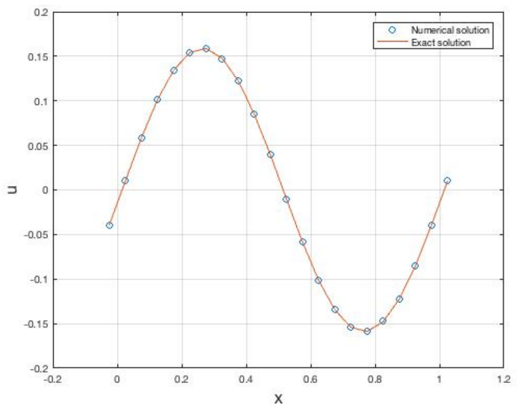


Figure 2.4: Solution at time *t=1* for *20* cells

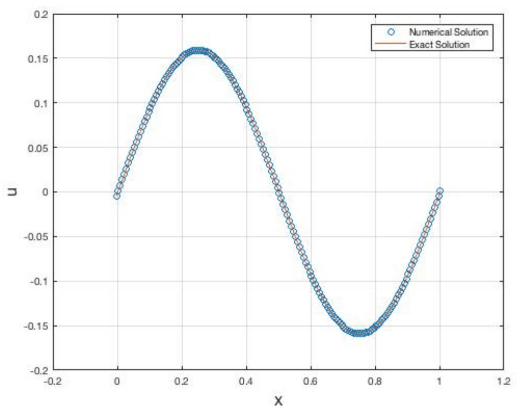


Figure 2.5: Solution at time *t=1* for *160* cells

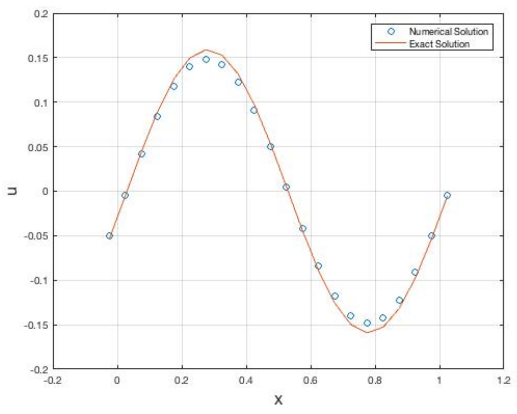


Figure 2.6: Solution at time *t=100* for *20* cells

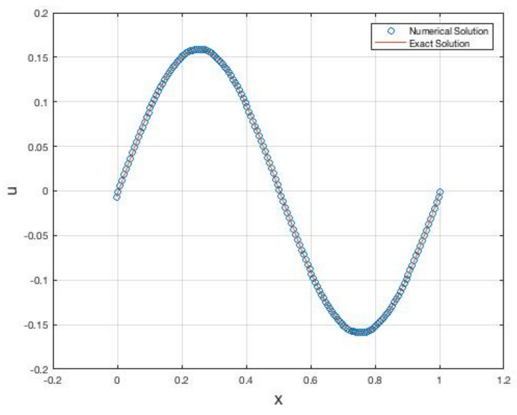


Figure 2.7: Solution at time *t=100* for *160* cells

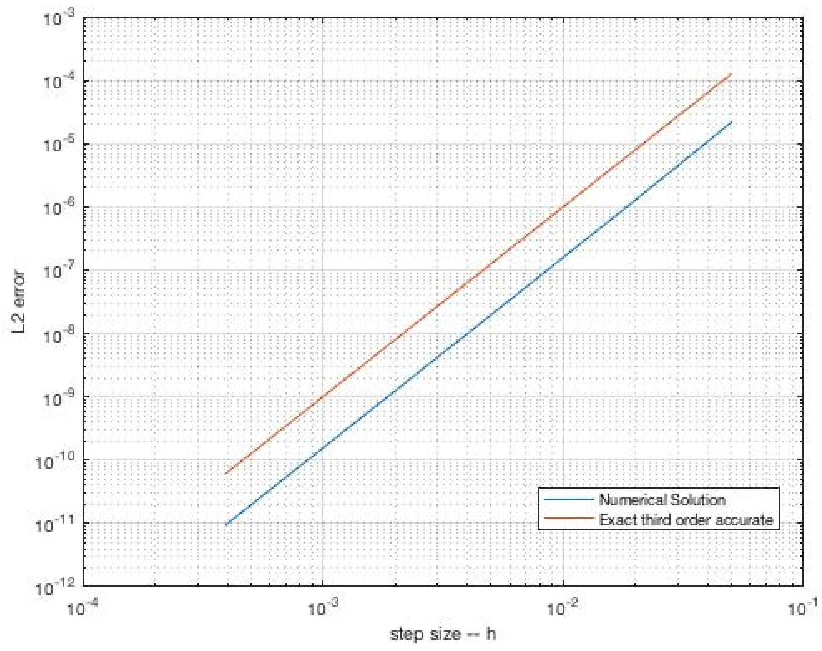


Figure 2.8: AF error convergence for advection test

3 System of One Dimensional Linear Acoustic Equations

Till now we have only presented scalar conservation laws. Now we begin to describe system of hyperbolic conservation laws. First we consider one dimensional linear acoustic equations.

3.1 The Equations

The acoustic equations is a system which shows relationship between pressure and velocity. Two equations comprise the system which are coupled in nature. The system is given by

$$\begin{cases} p_t + a_0 u_x = 0 \quad ; \\ u_t + a_0 p_x = 0 \end{cases} \tag{3.1}$$

The system can be re-written in the following form

$$\frac{\partial \boldsymbol{q}}{\partial t} + \boldsymbol{A}\frac{\partial \boldsymbol{q}}{\partial x} = 0 \tag{3.2}$$

where the state vector $\boldsymbol{q} = (p, u)$ and flux matrix $\boldsymbol{A} = \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix}$.

The equation must be complemented with an initial condition like given by

$$\begin{cases} p(x,0) = p_0(x) \quad ; \\ u(x,0) = u_0(x) \end{cases} \tag{3.3}$$

Also, a proper boundary condition is to be supplied according to the physics of the problem, which depends on which waves are moving in a positive direction and which waves are moving in a negative direction. Further, the AF scheme was first applied to the acoustic equation in [8]. Acoustic problem has also been dealt by AF in [12]

In this chapter, we will observe that 1D linear acoustic equations are nothing but a coupled system of 1D advection equations. Strategies have been discussed in the following text to convert the acoustic problem to the system of advection equations.

3.2 Strategy to Deal with the 1D System

The approach for solving one dimensional linear acoustics is trivial. The equation using the eigenvalue decomposition are switched into two advection equations with different variables, which do not depend on each other. Both equations are linear advection in nature, they are solved separately using the procedure discussed in chapter 2 earlier.

The Acoustic system can be diagonalised by performing an eigenvalue decomposition of the flux matrix $\boldsymbol{A}$.

$$\boldsymbol{A} = \boldsymbol{R}\boldsymbol{\Lambda}\boldsymbol{L}$$

Where $\boldsymbol{\Lambda}$,$\boldsymbol{L}$ and $\boldsymbol{R}$ are matrices of eigenvalues of $\boldsymbol{A}$, its corresponding eigenvectors and inverse of the matrix of eigenvectors respectively

Multiplying equation (3.2) throughout by $\boldsymbol{L}$ results in the system :

$$\boldsymbol{w_t} + \boldsymbol{\Lambda w_x} = 0 \tag{3.4}$$

where, $\boldsymbol{w} = \boldsymbol{Lq}$

which is, $\boldsymbol{w} = (w_1, w_2)$

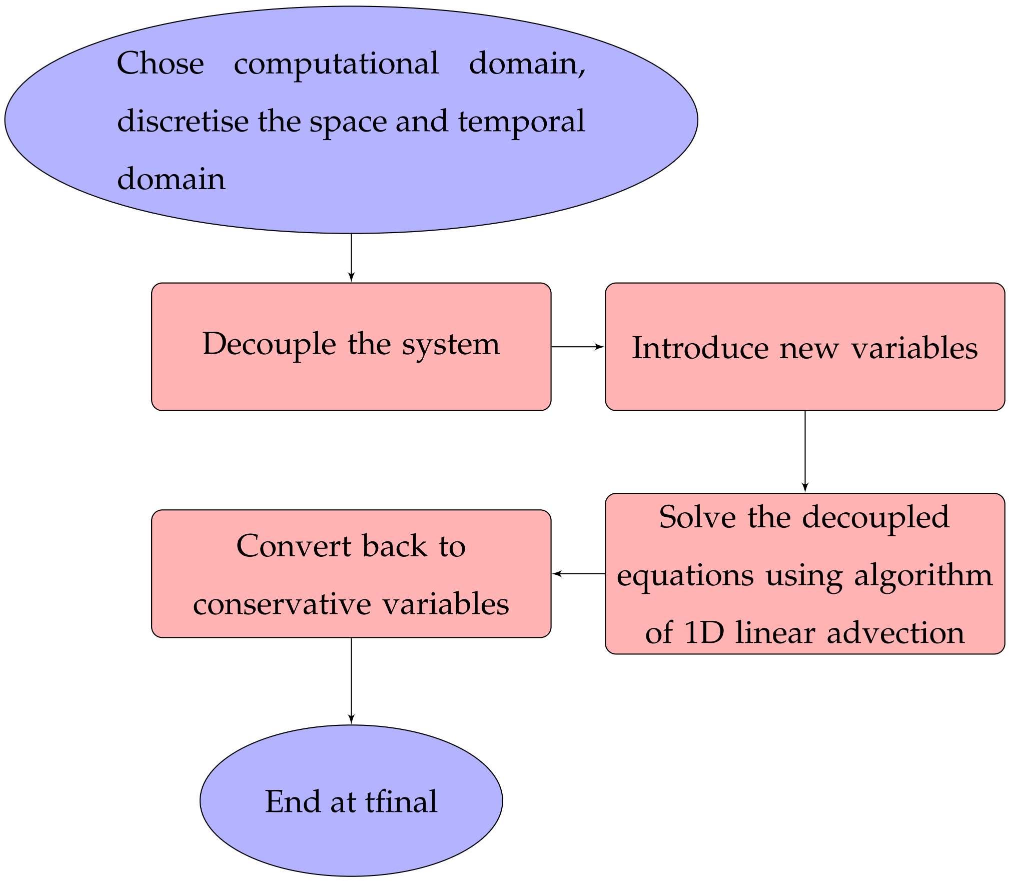
where, $w_1 = p + u$ and $w_2 = p - u$

After solving two state variable, w_1 and w_2 are obtained. Substituting back by the simple relationship gives the value of p and u.

Algorithm For 1D System of Acoustics

The Algorithm of 1D acoustic equation deals more with the *linear algebra* than concerning with numeric PDE. To start with, after discretising the computational domain, the system is decoupled. After that new variables (as discussed in the previous section) are introduced. The system is solved in these new variables (using the algorithm of 1D advection equation) and then substituted back to the original conserved variables.

Algorithm in the form of flow chart has been presented in the next page which guides through all these steps mentioned above.


Chose computational domain, discretise the space and temporal domain
Decouple the system
Introduce new variables
Solve the decoupled equations using algorithm of 1D linear advection
Convert back to conservative variables
End at tfinal

3.3 Discourse About the Solution Assembly

Acoustic equations after decoupling are much similar to the advection equations. so we discuss all of the solution strategies of acoustic in this section itself as much has already been discussed in chapter 2 .insights about reference element, mapping, reconstruction and updates have been presented below

Reference Element, Mapping & Reconstruction

The Reference element, Mapping and the reconstruction is exactly similar as in the advection case additionally the reconstruction step has been done twice, once for the pressure and once for the velocity. Pressure and velocity are first reconstructed then converted to the new state variables w_1 and w_2.

In AF scheme, as we know the conserved cell average and edge, vertex values are independently updated. Also the edge and vertex values need not be conservative. Due to the linear nature of characteristics the following equation holds for w_1 and w_2 as well

$$at = (\xi_i - \xi_0)\Delta x$$

$$\xi_0 = \xi_i - \frac{at}{\Delta x} \tag{3.5}$$

Therefore, to find the updated vertex value in w_1 and w_2 we only need to calculate $u(\xi_0)$. For using this we require both ξ_0^{n+1} value at $t = \Delta t$ and $\xi_0^{n+1/2}$ value at $t = \Delta t/2$. Once we have them the average flux is again given by

$$\overline{f} = \frac{1}{6}[f(u(\xi_i) + 4f(u(\xi_0^{n+1/2})) + f(u(\xi_0^{n+1}))] \tag{3.6}$$

for one dimensional linear advection, the characteristics origin depends upon the CFL number v. So, here as well the equations remain the same after decoupling

$$v = \frac{a\Delta t}{\Delta x}$$

Vertex and Edge Update Formulas for Negative Wave Speed

In addition to the formulas derived in chapter 2 we will be needing the formulas for negative advection speed as well because one of the state variables among w_1 or w_2 will be advected with positive wave speed and the other one will be travelling with the negative pace.

All the notations and symbols have their usual meaning as described in chapter 2 So, for negative wave speed we have

$$\xi_0^{n+1/2} = \xi_i + \frac{v}{2} \tag{3.7}$$

$$\xi_0^{n+1} = \xi_i + v \tag{3.8}$$

where left interface is considered $\xi_i = 0$, and in the same manner we obtain the vertex updates as

$$w_{j+1/2}^{n+1} = v(3v-2)w_{j+1/2}^n + 6v(1-v)\overline{w}_j^n + (1-v)(1-3v)w_{j-1/2}^n \tag{3.9}$$

When wave speed is negative the formula becomes

$$\overline{w}_j^{n+1} = v^2(v-1)w_{j+1/2}^n + v^2(3-2v)\overline{w}_j^n + v(1-v)w_{j-1/2}^n + (1-v)^2(1+2v)\overline{w}_{j-1}^n - v(1-v)^2 w_{j-3/2}^n \tag{3.10}$$

where, w is used for w_1 or w_2. The above formulas hold for both of them.

Updating the Substituted Variables

Next, after calculating the vertex updates for w_1 or w_2, numerical flux are calculated using equation (3.6) For positive wave speed the flux obtained is

$$F_{j+1/2}^n = a[v(v-1)w_{j-1/2}^n + v(3-2v)\overline{w}_j^n + (1-v)^2 w_{j+1/2}^n] \tag{3.11}$$

While, for the negative wave speed the flux obtained is

$$F_{j+1/2}^n = a[(1-v)^2 w_{j-1/2}^n + v(3-2v)\overline{w}_j^n + v(v-1)w_{j+1/2}^n] \tag{3.12}$$

The cell averages are then conservatively updated from the flux function by

$$\overline{w}_j^{n+1} = \overline{w}_j^n - \frac{\Delta t}{\Delta x}(F_{j+1/2}^n - F_{j-1/2}^n) \tag{3.13}$$

Substituting the flux function for left and right interface results in full update for conserved quantity. After obtaining conserved values at $(n+1)^{th}$ level, the same procedure is followed to obtain values at next time step and the process continues till the final specified time.

Results in Conserved Quantity

Once the solution has been obtained in variables w_1 and w_2, through a simple relationship between the $w's$ and the pressure and velocity gives back the values of pressure and velocity for the system. The final results are plotted in the conserved quantities pressure and velocity.

3.4 Numerical Examples

We now present a numerical experiment carried out in [**?**], to ascertain the order of accuracy of the scheme.

With $a_0 = 1$ and for $x \in [0, 2]$. Periodic boundary conditions are imposed. Numerical tests are presented at time $t = 10$ and 100 for 20 cells for both simple and non-simple wave. The Courant number has been fixed at 0.7. For checking order of accuracy, the results are tabulated.

Experiments on Simple Wave

For simple wave the initial condition is

$$p(x, 0) = \frac{1}{2}\sin \pi x \tag{3.14}$$

$$u(x, 0) = -\sin \pi x \tag{3.15}$$

For checking the order of accuracy the results are tabulated in table 3.1.

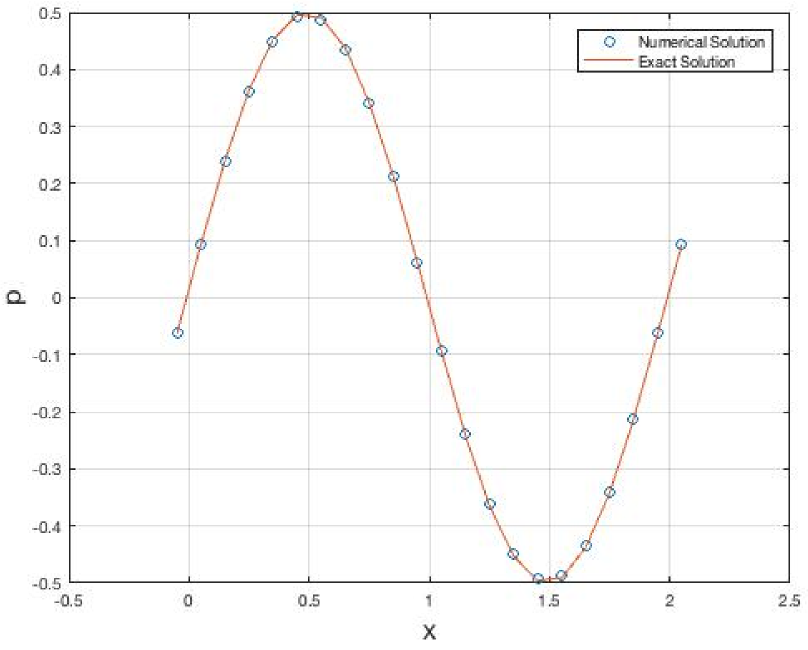


Figure 3.1: Pressure at *t=10* for *20* cells

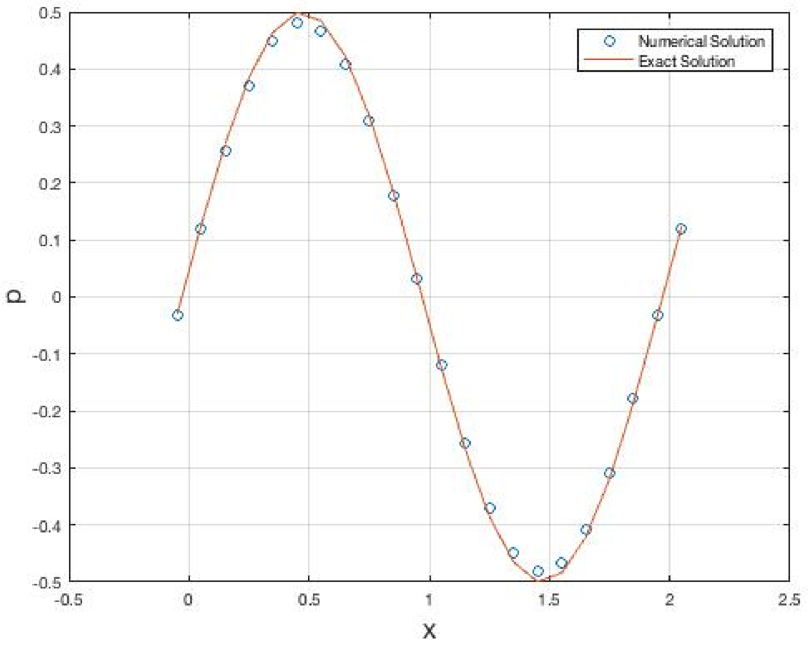


Figure 3.2: Pressure at *t=100* for *20* cells

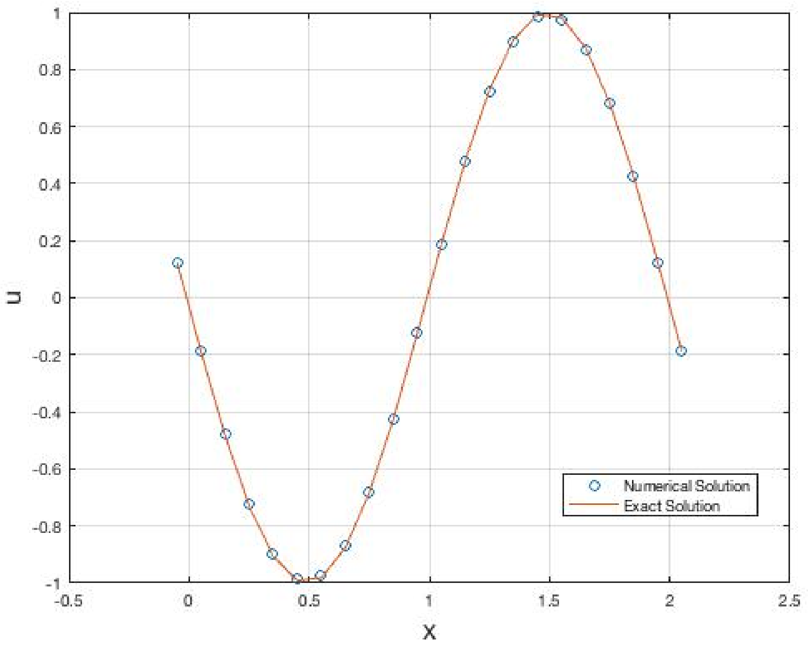


Figure 3.3: velocity at *t=10* for *20* cells

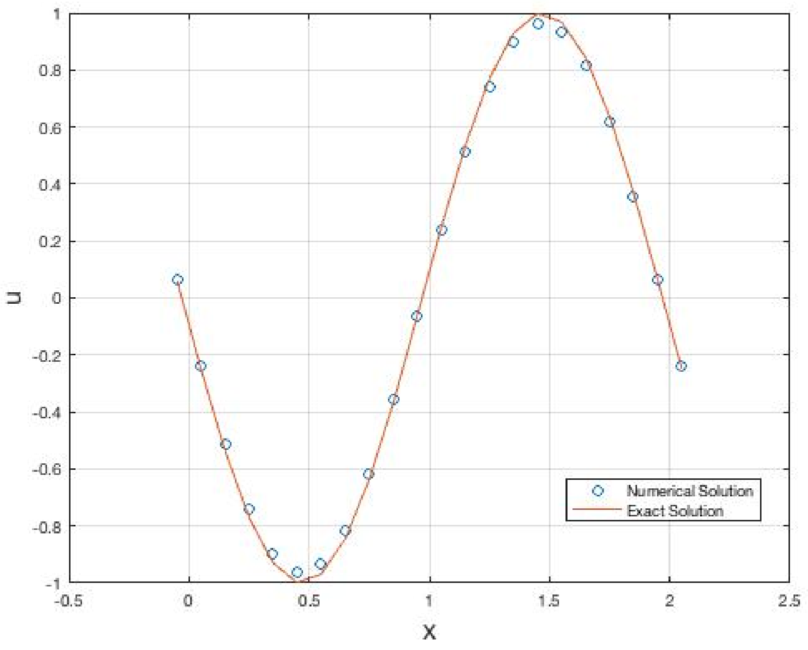


Figure 3.4: velocity at *t=100* for *20* cells

Table 3.1: Order Of accuracy Of 1D Linear Acoustic Simple wave

Level	N	$L_2(\overline{p})$	order	$L_2(\overline{u})$	order
1	20	$1.2563e-04$	–	$2.5127e-04$	–
2	40	$1.5420e-05$	3.02	$3.0840e-05$	3.02
3	80	$1.9143e-06$	3.00	$3.8286e-06$	3.00
4	160	$2.3842e-07$	3.00	$4.7682e-07$	3.00
5	320	$2.9750e-08$	3.00	$5.9500e-08$	3.00
6	640	$3.7155e-09$	3.00	$7.4311e-09$	3.00
7	1280	$4.6424e-10$	3.00	$9.2848e-10$	3.00
8	2560	$5.8017e-11$	3.00	$1.1603e-10$	3.00

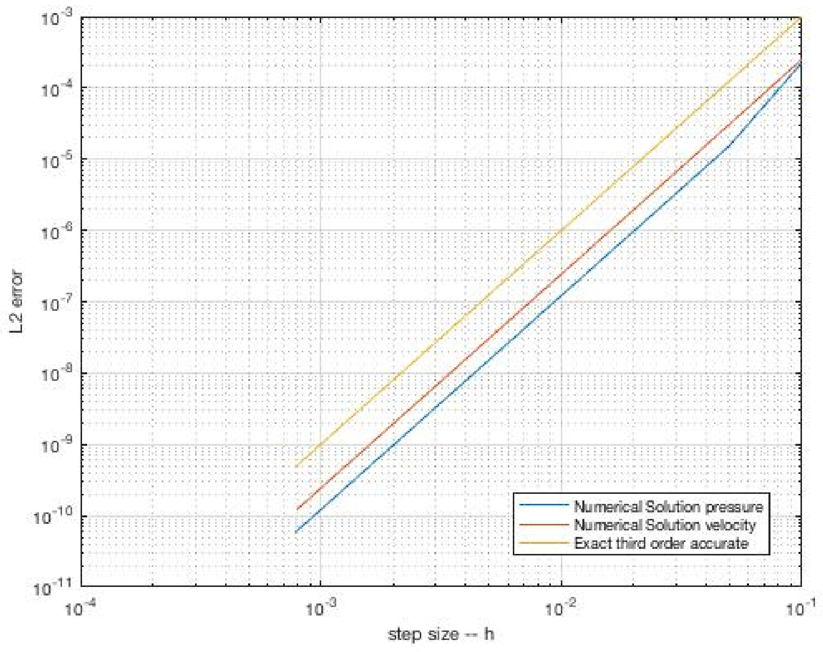


Figure 3.5: L_2 norm errors for simple wave

Experiments on Non-Simple Wave

For non-simple wave the initial condition is

$$p(x,0) = \frac{1}{4} + \frac{1}{80}\sin 2\pi x \tag{3.16}$$

$$u(x,0) = \frac{1}{4} - \frac{1}{10}\sin \pi x \tag{3.17}$$

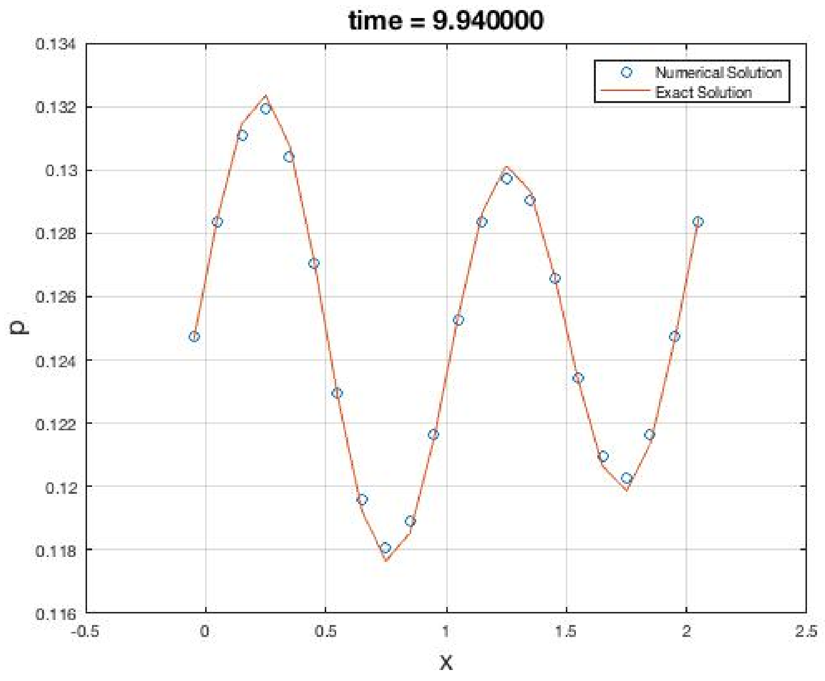


Figure 3.6: Pressure at *t=10* for *20* cells

For checking the order of accuracy the results are tabulated in table 3.2.

Table 3.2: Order Of accuracy Of 1D Linear Acoustic Non-Simple wave

Level	N	$L_2(\overline{p})$	order	$L_2(\overline{u})$	order
1	20	$7.5173e-06$	–	$2.5320e-05$	–
2	40	$8.5604e-07$	3.13	$3.0853e-06$	3.03
3	80	$1.0118e-07$	3.08	$3.8293e-07$	3.01
4	160	$1.2266e-08$	3.04	$4.7685e-08$	3.00
5	320	$1.5091e-09$	3.02	$5.9500e-09$	3.00
6	640	$1.8713e-10$	3.01	$7.4311e-10$	3.00
7	1280	$2.3297e-11$	3.00	$9.2848e-10$	3.00
8	2560	$2.9062e-12$	3.00	$1.1603e-11$	3.00

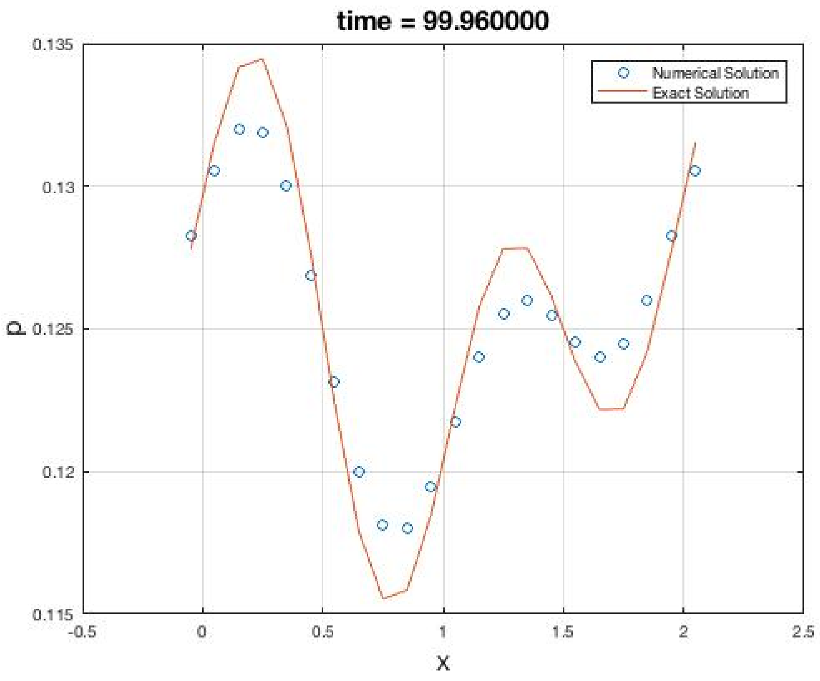


Figure 3.7: Pressure at *t=100* for *20* cells

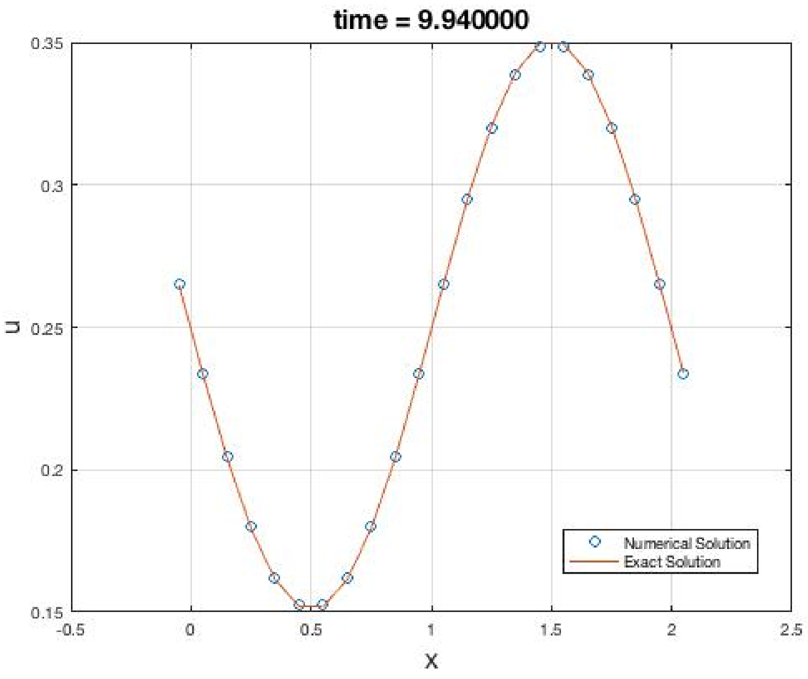


Figure 3.8: velocity at *t=10* for *20* cells

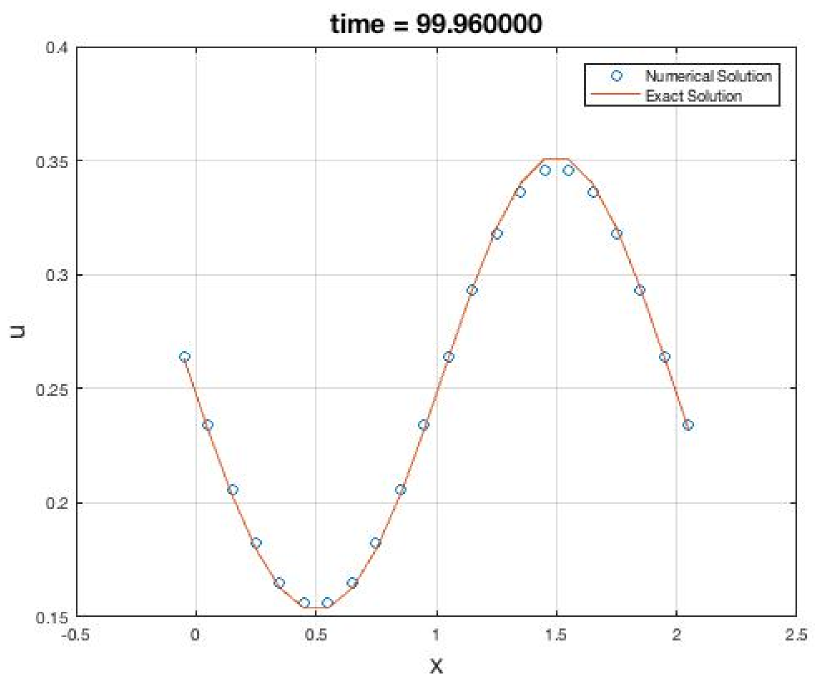


Figure 3.9: velocity at *t=100* for *20* cells

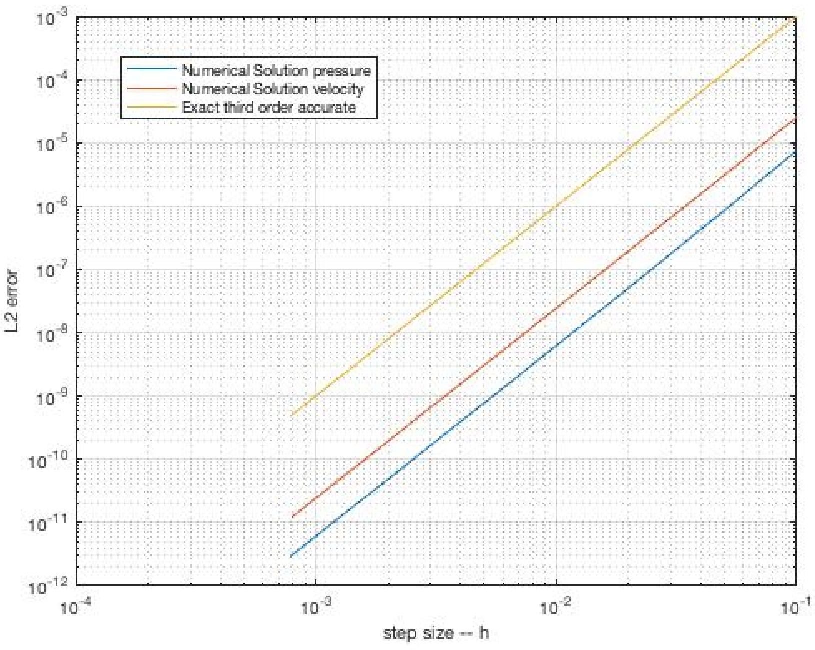


Figure 3.10: L_2 norm errors for non simple wave

4 Coding Structure For Two Dimensional Problems in Cartesian Grid

In this chapter, we present a unique discussion about the coding structure employed by the authors of this text to solve the scalar and system of hyperbolic equations on the rectangular cartesian grid. These coding structures are just to motivate the reader about one of the ways the problem can be solved and in no way imply that there is no other way to simulate. The work has been carried out in an $x - y$ plane but we believe similar work can be thought of in cylindrical polar coordinates and spherical coordinates, which may simplify and ease up the use of CFD related equations.

4.1 Introduction to the Cartesian Grid

First of all, a proper explanation of the cartesian grid is must. The $x - y$ plane is discretised into computational cell of length Δx and width Δy. If,

Number of cells placed horizontally = n
Number of cells placed vertically = m
Left domain limit = xa
Right domain limit = xb

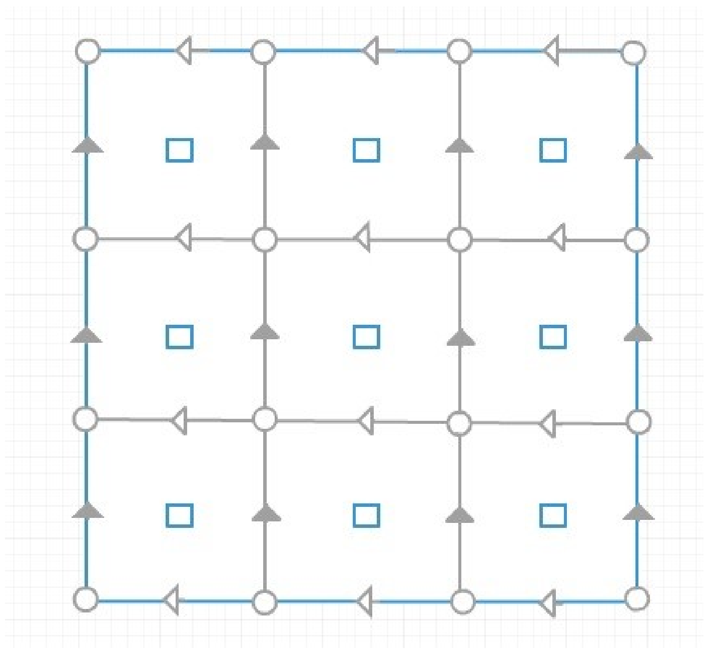


Figure 4.1: 2D Grid

Maximum domain limit along y = yb

Minimum domain limit along y = ya

Then,

$$\Delta x = \frac{(xb - xa)}{n} \tag{4.1}$$

$$\Delta y = \frac{(yb - ya)}{m} \tag{4.2}$$

For all the 2-D numerical experiments carried out in this text, the domain is chosen symmetrical to the origin so that $xa = ya$ and $xb = yb$. Also, a number of cells horizontally placed is taken equal to the number of cells placed vertically i.e, $n = m$. The convections listed above are not mandatory and AF does not require the domain to be symmetric about the origin. It has just been chosen to facilitate the ease in simulating. After the grid has been formed, we need to look at the data storage locations in the 2D grid, which is presented in the next section.

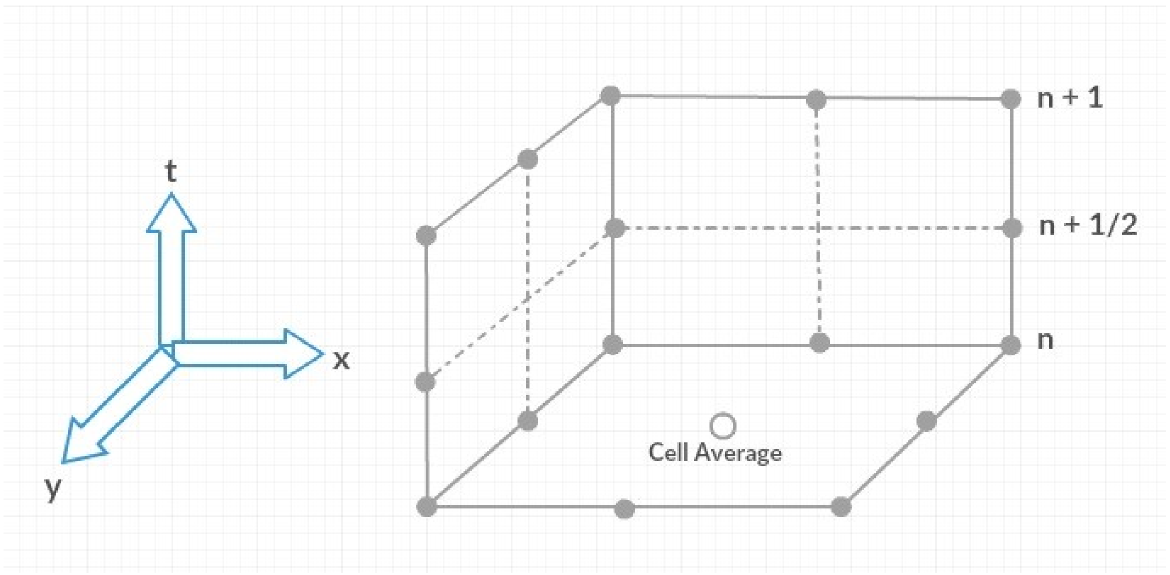


Figure 4.2: Data Storage Locations in 2D AF

4.2 Data Storage Locations in 2D Active Flux

The Figure 4.1 illustrates the 2D grid.It is evident that there are four kinds of data storage points in a grid namely,

1. The square – at the center of each cell representing the conserved quantity cell average of the cell.
2. Circles – at the four corner points of the cell.
3. Shaded triangles – which are at the vertical edge of the cell.
4. Blank triangles – which are at the horizontal edges of the cell.

Also, Figure 4.2 shows the number of data storage points in one cell. It is evident that for regular Finite volume strategies, the cell average is the only DOF per cell. Presently furthermore to the cell average, there are 8 edge and vertex degrees of opportunity disseminated along the cell boundary. They are anyway likewise halfway shared by neighboring cells: There are four nodes esteem, each mutual by four cells. Additionally, there are four edges esteems, which are shared by two cells each. In this manner per cell one is left with

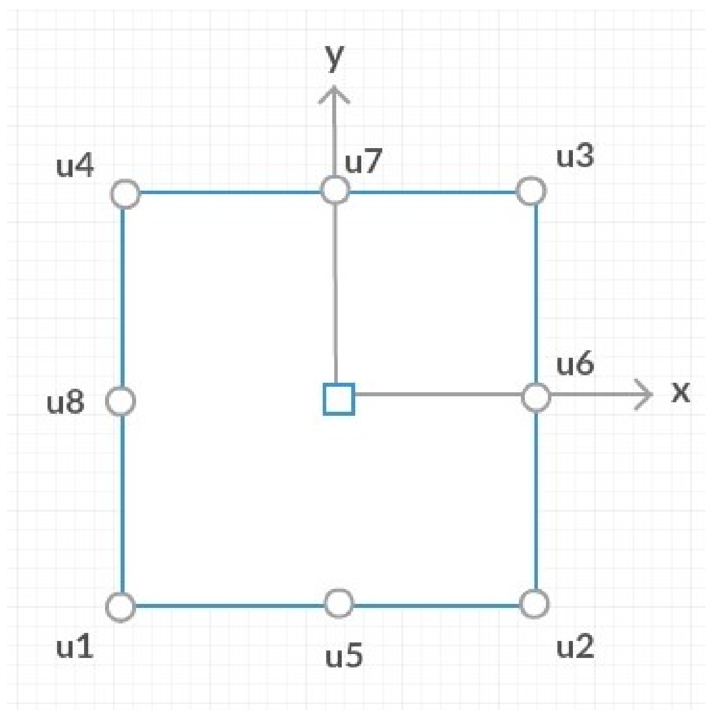


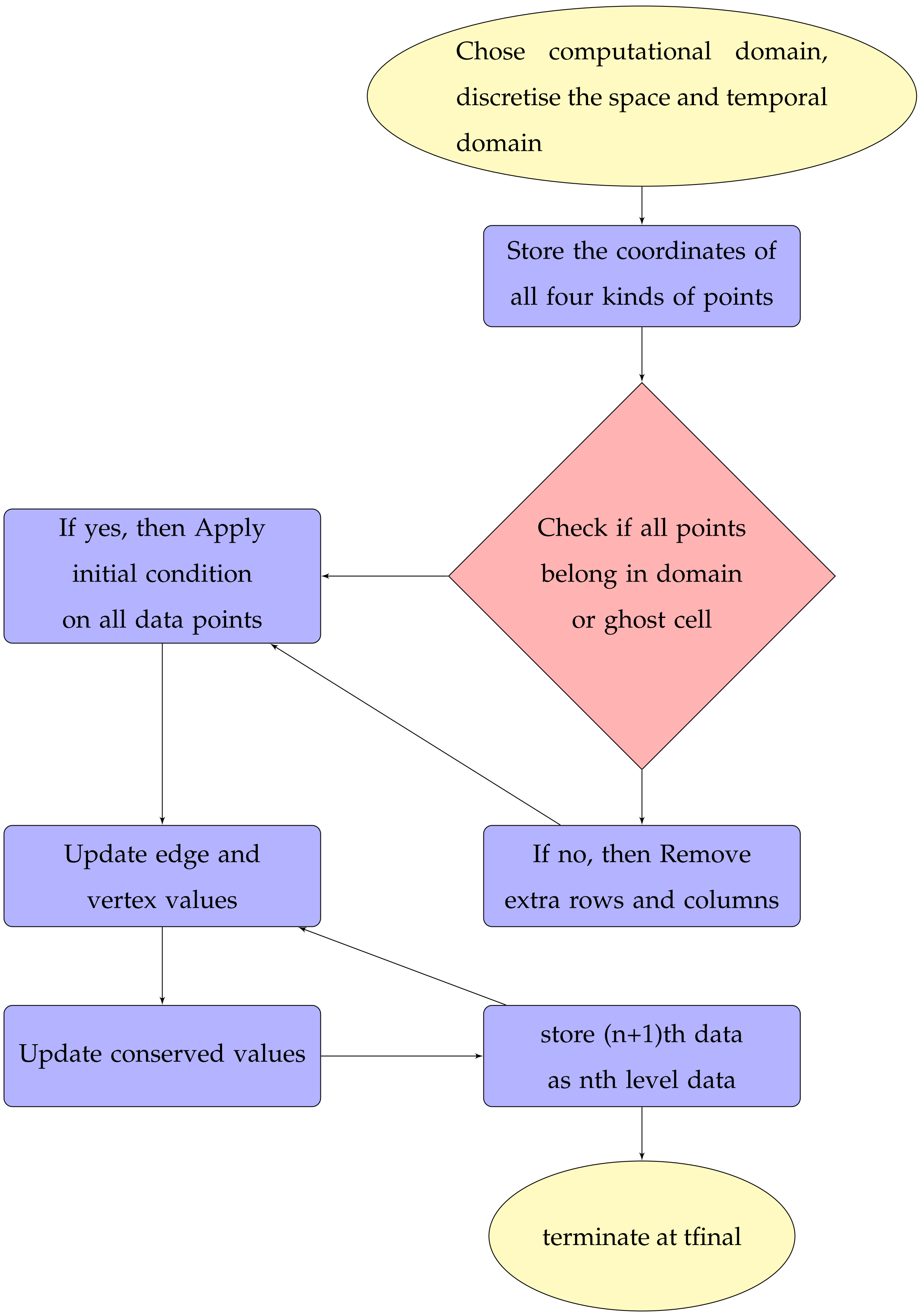
Figure 4.3: A computational Cell in Cartesian Grid

- one cell average
- one node value
- two edge values (horizontal and vertical)

In Figure 4.2 these DOF are shown by changed geometry. In fact, every one of the frames a grid with spacings Δx and Δy in x and y direction, individually. When tackling linear acoustics with the AF technique, there are subsequently 12 DOF for every cell, for example, 4 degrees of freedom for each cell with 3 variables for every level of opportunity. In the accompanying, the classification from Figure [] is utilized to separate between the various types of degrees of freedom.

The reason why the distinction between all four kinds of points has been made can be understood by first trying to set up the problem in any mathematical software like MATLAB [17]. Then it will be clear that to simulate the exact formulas of the 2D AF scheme this is necessary. the general algorithm of AF in the 2D grid is presented in the next subsection.

A General Algorithm For 2D Problems



Chose computational domain, discretise the space and temporal domain
Store the coordinates of all four kinds of points
Check if all points belong in domain or ghost cell
If yes, then Apply initial condition on all data points
If no, then Remove extra rows and columns
Update edge and vertex values
Update conserved values
store (n+1)th data as nth level data
terminate at tfinal

The algorithm of a general solution of the 2D conservation equation in the 2d grid is shown on the previous page. Choosing and discretizing the domain has been discussed earlier. On all four kinds of points, the initial condition is stored, because we need them to update the edge and vertex values. Actually, the blank triangles and shaded triangles are the midpoints of the edges of the computational cells, so in finding flux through each side, these quantities contribute in the n^{th} level. similarly, after updating the $(n+\frac{1}{2})^{th}$ level and $(n+1)^{th}$ level values of the edge and the vertex of the triangles and the circles contribute to the flux. After flux calculation, square – the conserved quantity is updated. This step is repeated until the final time is attained.

4.3 Updating the Values in Cartesian Grid

Basically, two kinds of update take place.

1. The update, or the more sophisticated way to say, finding of edge and vertex values at $(n+\frac{1}{2})^{th}$ level and $(n+1)^{th}$ level.
2. Update of conserved quantities at the $(n+1)^{th}$ level.

For Half Time Step

At half time step, only the edge and vertex values are to be calculated. As we know the values at all four types of data points in the computational cell at the n^{th} level, applying the physics of the problem intelligently taking directions of propagation into account, $(n+\frac{1}{2})^{th}$ level calculations can be taken care of. Specifics about how to perform $(n+\frac{1}{2})^{th}$ level calculations in test problems of advection and acoustics have been discussed in respectively.

For Full Time Step

At $(n+1)^{th}$ level both the conserved quantities and the edge, vertex values need to be calculated. After calculating $(n+1)^{th}$ level value of edge and vertex using the similar techniques employed in to find at the half time step values, flux through each face is calculated. Using update equation the conserved quantities are updated at $(n+1)^{th}$ level. These values are then stored as n^{th} level values for running the loop through time, till the final time where solutions are required.

5 Two Dimensional Linear Advection Equation

AF method on two dimensional linear advection problem is discussed in this chapter. This chapter is a straightforward extension of chapter 2 The sections deal with developing the theory about reconstruction and updates. The chapter ends with a numerical experiment and its explanation regarding it.

5.1 The Equation

Two dimensional linear advection equation is given by

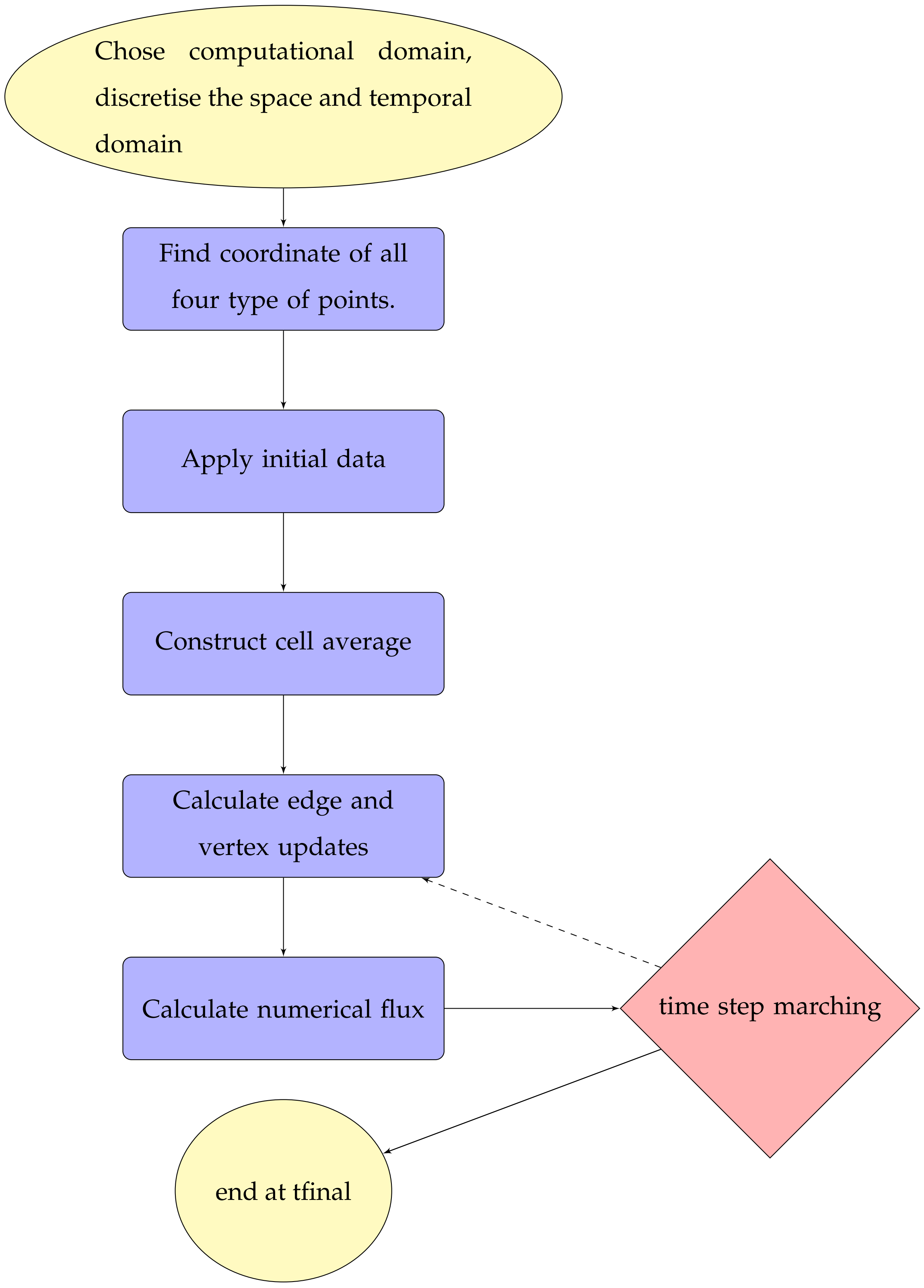
$$u_t + au_x + bu_y = 0 \tag{5.1}$$

with initial condition

$$u(x, y, 0) = u_0(x, y) \tag{5.2}$$

supplied together with suitable boundary conditions, that are needed to be applied on the four boundaries of the Cartesian domain, depending upon the problem. Similarly, as in one dimensional linear advection the theory of AF here also depend upon the back tracing of characteristics in space.

The Algorithm


Chose computational domain, discretise the space and temporal domain
Find coordinate of all four type of points.
Apply initial data
Construct cell average
Calculate edge and vertex updates
Calculate numerical flux
time step marching
end at tfinal

The algorithm is pretty similar to the algorithm applied to the one-dimensional advection equation. When programming the two-dimensional code. It sometimes becomes tedious to take care of all four kinds of points and to substitute the values in all the four kinds of point. This is the only difference in the case of 2D otherwise constructing the cell averages and updating the edge and vertex values and the numerical flux are all steps the same.

One important step is to give attention to which points are updated from which cells. This decision is taken based on the direction of the advection vector. For the $(1,1)$ advection vector the points in east, north, and north-east are updated from the cell only and the rest of the five boundary points are updated from neighboring cells.

5.2 Reference Element and Mapping

In this section two dimensional mapping from physical space to reference space is discussed. A linear mapping Jacobian is introduced to map between the physical and the reference element space.

$$\boldsymbol{x} = \boldsymbol{x_c} + \boldsymbol{J}\boldsymbol{\xi} \tag{5.3}$$

where, $\boldsymbol{x_c}$ is the centroid coordinate in the physical space. This mapping applies to any convex arbitrary quadrilateral.

$$\boldsymbol{J} = \frac{\partial x_i}{\partial \xi_j} \tag{5.4}$$

The solution of two dimensional linear advection is also constant along the characteristics. This theory will be used primarily to solve by AF scheme.

5.3 Internal Reconstructions

Same as in the one dimensional case, standard finite element construction is done inside every cell. Additionally to maintain conservation cubic bubble function is considered

$$u(\xi) = \sum_{i=1}^{9} c_i \phi_i(\xi, \eta) \tag{5.5}$$

Table 5.1 lists all reconstruction coefficients and basis functions. Two dimensional reconstruction is simply constructed by sum of Lagrange basis function and bubble function. A bi-quadratic polynomial is used, this is due to the additional interface nodes introduced in the geometry. The unknowns in the above equation can be found out in the same manner as done in 2.4. The unknowns are tabulated as follows

Table 5.1: Basis function and coefficient for 2D reconstruction

Index	c_i	ϕ_i
1	u_1	$0.25(\xi-1)(\eta-1)\xi\eta$
2	u_2	$0.25(\xi+1)(\eta-1)\xi\eta$
3	u_3	$0.25(\xi+1)(\eta+1)\xi\eta$
4	u_4	$0.25(\xi-1)(\eta+1)\xi\eta$
5	u_5	$0.5(1-\xi^2)(\eta-1)\eta$
6	u_6	$0.5(1-\eta^2)(\xi+1)\xi$
7	u_7	$0.5(1-\xi^2)(\eta+1)\eta$
8	u_8	$0.5(1-\eta^2)(\xi-1)\xi$
9	u_9	$(1-\xi^2)(1-\eta^2)$

5.4 Updating the Edge and Vertex Values

The time advanced interface solution is evaluated at the characteristics origin $\boldsymbol{\xi_F}$.

Now, the requirement is to find correct expression for $\boldsymbol{\xi_F}$. This is fulfilled by the theory of characteristics back tracing. The upwind direction can be determined by tracing the characteristics vector which is just the constant velocity vector in linear advection.

The two dimensional characteristic origin $\boldsymbol{\xi_F}$ is defined as follows

$$\boldsymbol{\xi_F} = \boldsymbol{\xi_I} - \Delta t \boldsymbol{J^{-1}} \boldsymbol{a} \tag{5.6}$$

Where $\boldsymbol{J^{-1}}$ is the inverse coordinate mapping Jacobian matrix to transform the vector $\boldsymbol{a}$ in physical space to reference space. The full-time step value at the interface $\boldsymbol{\xi_I}$ is evaluated by interpolating the characteristics origin corresponding to full time.

$$u^{n+1}(\boldsymbol{\xi_I}) = u^n(\boldsymbol{\xi_F}) \tag{5.7}$$

The half time step value at the same interface $\boldsymbol{\xi_I}$ is also easily evaluated by interpolating at the characteristics origin corresponding to half time.

$$u^{n+1/2}(\boldsymbol{\xi_I}) = u^n(\boldsymbol{\xi_F}) \tag{5.8}$$

Where $\boldsymbol{\xi_F}$ for half time step is defined as

$$\boldsymbol{\xi_F} = \boldsymbol{\xi_I} - \frac{\Delta t}{2} \boldsymbol{J^{-1}} \boldsymbol{a} \tag{5.9}$$

Once we have all these four expressions calculated numerically, we are sufficient to find the numerical flux.

5.5 Calculating Numerical Flux and Time Step Marching

Having calculated the edge and vertex updates we are only left with calculating the updates for the conserved quantity, cell averages, these are calculated with the help of flux functions. The two dimensional linear advection flux vector is given by

$$\boldsymbol{f} = (au; bu)$$

The flux function help in finding the flux through all four sides of the computational cell. The fluxes are multiplied with the normal vector and the length of each side of the cell. The conserved cell average formula is given by

$$\overline{u}_j^{n+1} = \overline{u}_j^n - \frac{\Delta t}{\Omega_j} \sum_{i=1}^{4} (\boldsymbol{f}.\boldsymbol{n}) l \tag{5.10}$$

where $\boldsymbol{n}$ is normal vector through the interface. Which are basically the unit normal vectors along the x and y axis in case of cartesian grid. The only unknown in the above formula is interface flux. which is calculated by Simpson's composite rule.

$$f = \frac{1}{9}\left[\frac{1}{4}(f_L^n + f_R^n + f_L^{n+1} + f_R^{n+1}) + (f_M^n + f_L^{n+1/2} + f_R^{n+1/2} + f_M^{n+1}) + 4 f_M^{n+1/2}\right] \tag{5.11}$$

where for $i = L, M, R$

$f_i^n = (au_i^n; bu_i^n)$
$f_i^{n+1/2} = (au_i^{n+1/2}; bu_i^{n+1/2})$
$f_i^{n+1} = (au_i^{n+1}; bu_i^{n+1})$

5.6 Numerical Illustration

It is elementary to define the step size or step length in 1D problems, but without defining them properly in 2D computations, study of order of convergence cannot be done. In AF degree of freedom for a quadrilateral is defined as follows

$$DOF = N_{Cell} + \frac{1}{2}N_{Face} + \frac{1}{4}N_{Vertex} \tag{5.12}$$

and reference length h as follows

$$h = DOF^{-1/2} \tag{5.13}$$

As in 1D calculations, $L_2(\overline{u})$ norm error is used to calculate the order of accuracy. A Gaussian initial condition is considered for numerical accuracy test with periodic boundary condition. With advection speed $a = 1$ and for spatial domain $x \in [-5, 5]$ and $y \in [-5, 5]$. Numerical tests are presented at time $t = 12$ and 30 for 2500 and 10000 cells. For checking order of accuracy the results are tabulated in 5.2. Solution plot at time 1, 5 and 7.5 has been shown to inspect the behaviour at the boundary.

$$u_0(x, y) = e^{-0.5(x^2+y^2)} \tag{5.14}$$

Table 5.2: Order Of accuracy Of 2D Linear Advection

DOF	$L_2(\overline{u})$	order
$1.5842e+03$	0.0069	–
$5.7542e+03$	$9.5520e-04$	1.53
$2.1894e+04$	$4.5232e-05$	2.28
$8.5374e+04$	$1.7176e-05$	2.35
$3.3713e+05$	$6.2374e-06$	2.42
$1.3399e+06$	$4.6080e-07$	2.50
$5.3421e+06$	$1.6440e-07$	2.50

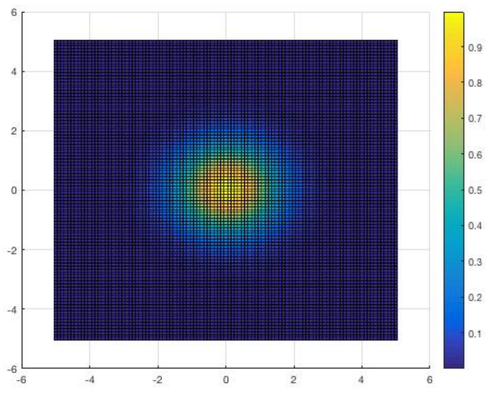


Figure 5.1: Initial Condition for *10000* cells

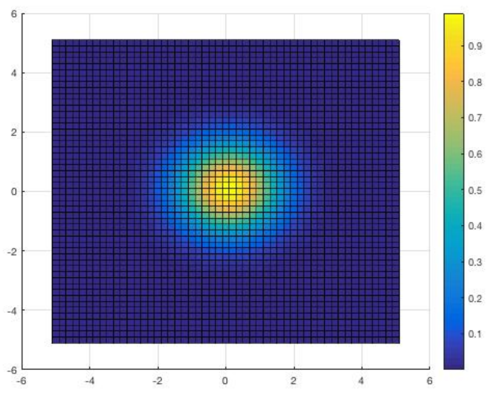


Figure 5.2: Initial Condition for *2500* cells

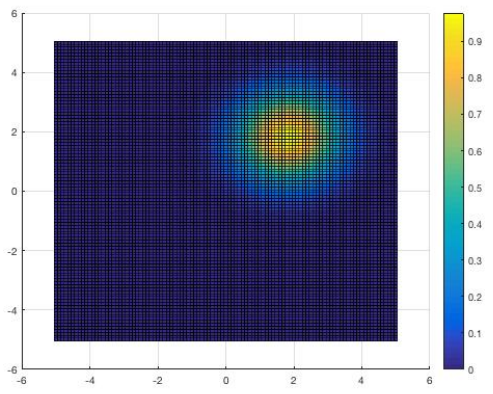


Figure 5.3: Solution at time *t=12* for *10000* cells

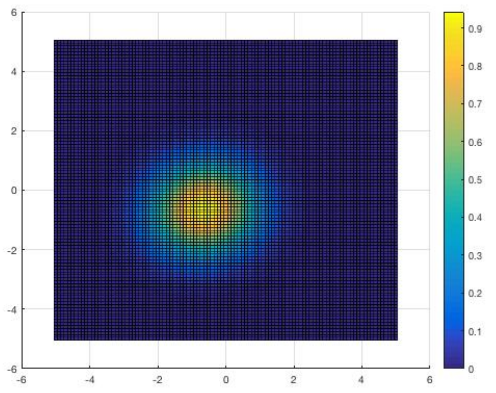


Figure 5.4: Solution at time *t=30* for *10000* cells

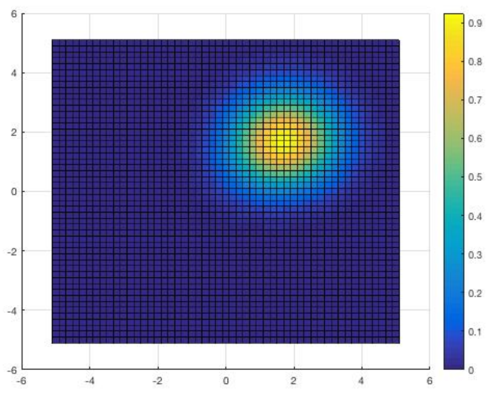


Figure 5.5: Solution at time *t=12* for *2500* cells

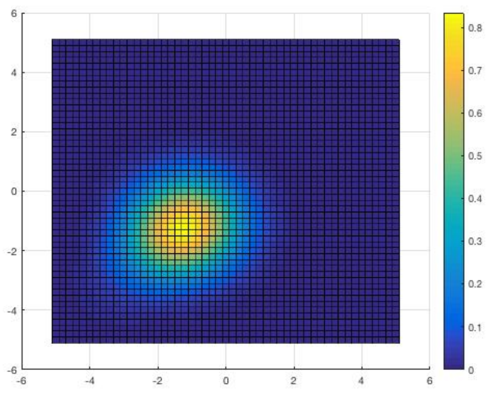


Figure 5.6: Solution at time *t=30* for *2500* cells

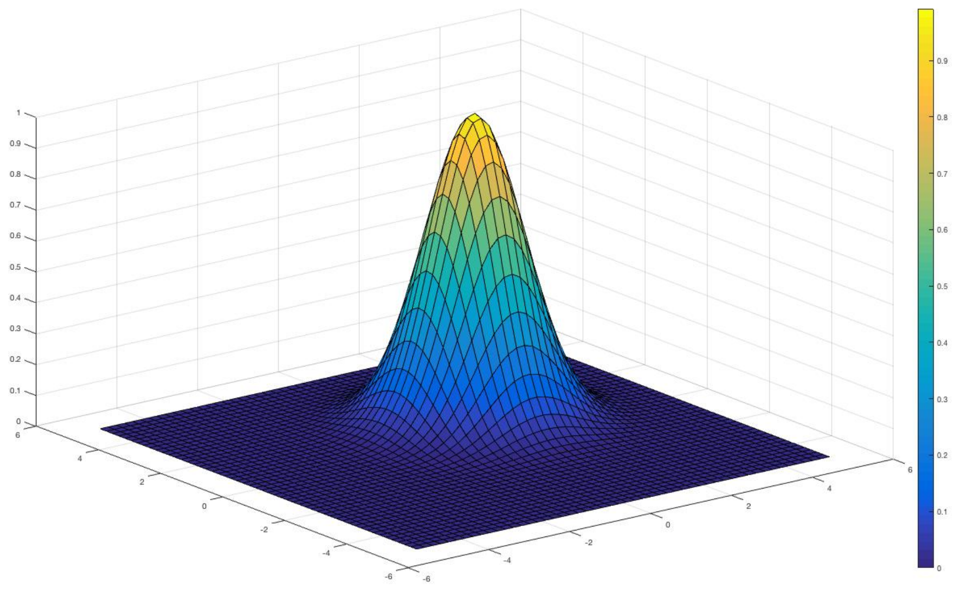


Figure 5.7: Solution at time *t=1* for *2500* cells

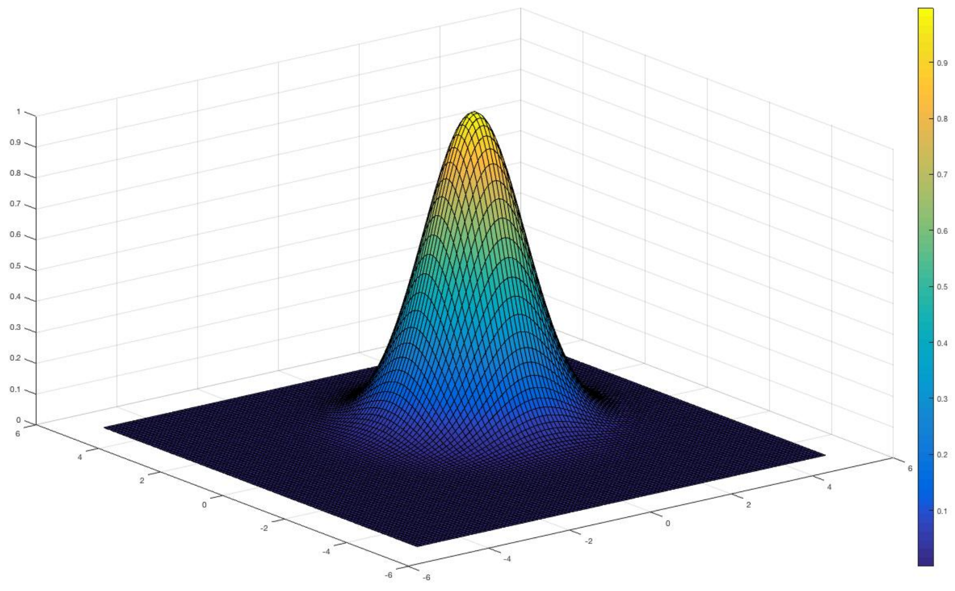


Figure 5.8: Solution at time *t=1* for *10000* cells

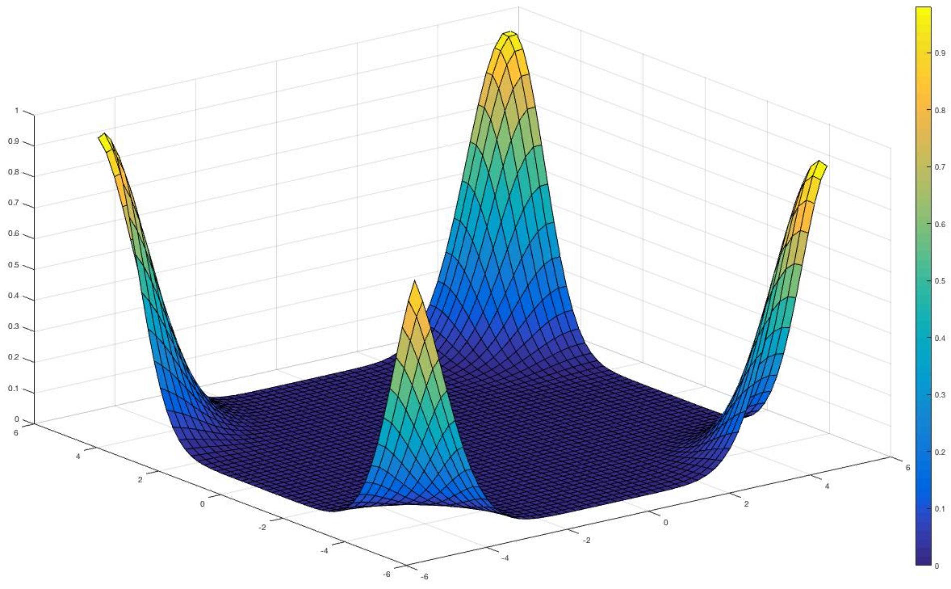


Figure 5.9: Solution at time *t=5* for *2500* cells

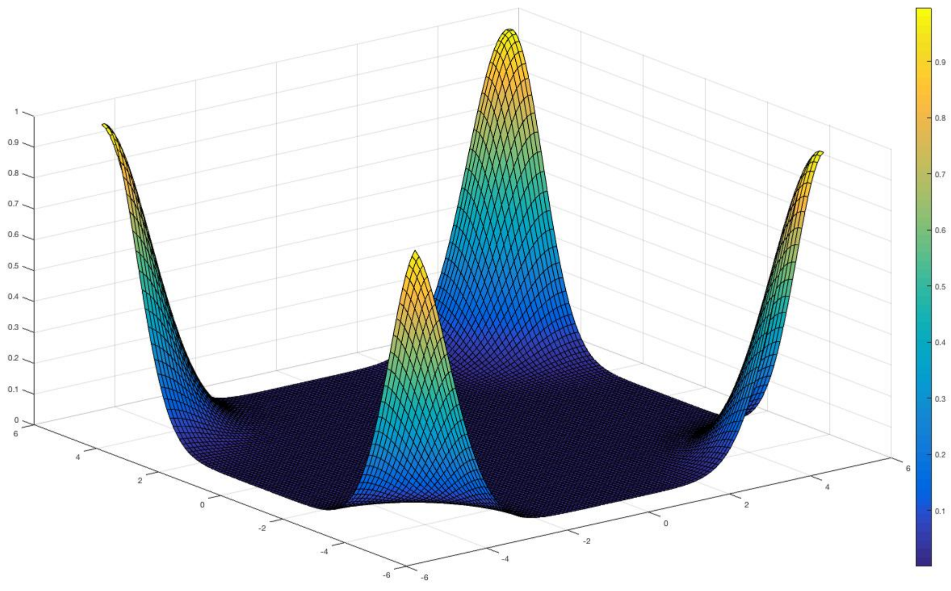


Figure 5.10: Solution at time *t=5* for *10000* cells

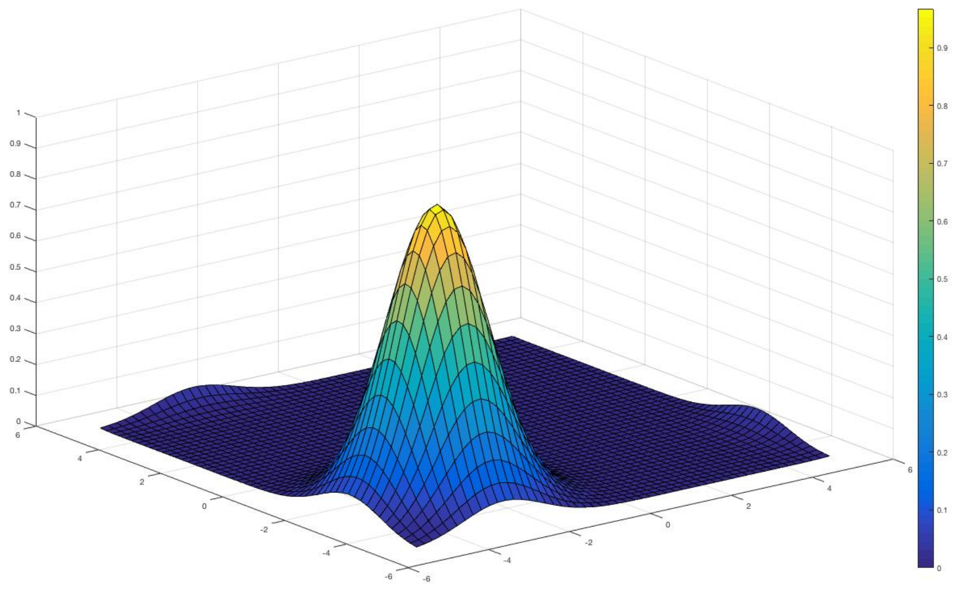


Figure 5.11: Solution at time *t=7.5* for *2500* cells

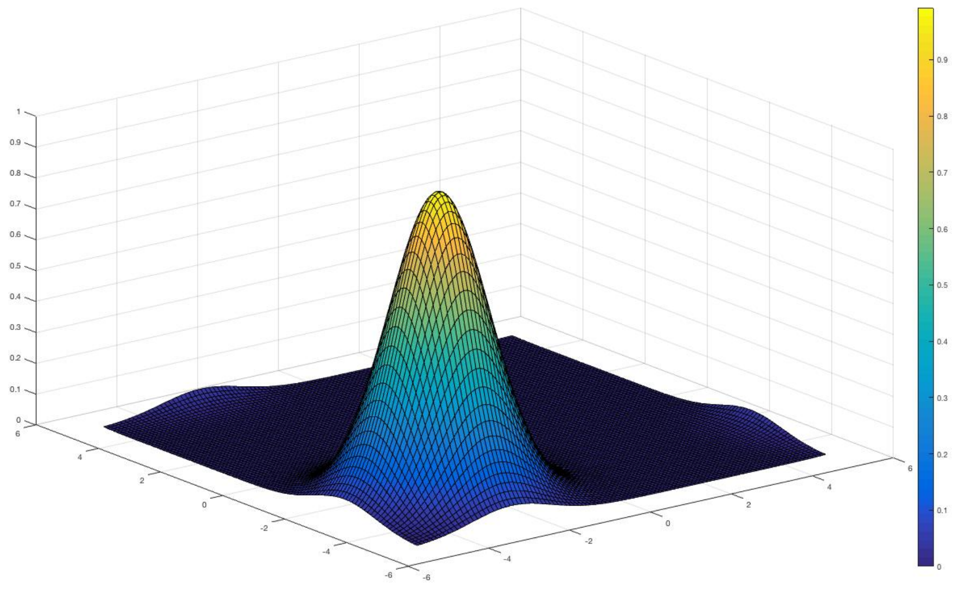


Figure 5.12: Solution at time *t=7.5* for *10000* cells

5.7 Explanations For the Preposterous Order Of Accuracy

It is evident that the order of accuracy obtained is not three, which was expected. However, this problem has been discussed in [11] thoroughly. The order of accuracy of two-dimensional linear advection is not equal to three because of the mesh alignment problem in multidimensional advection. In Maeng's thesis remedies about the problem are also suggested which can increase the order of accuracy to 2.6 to 2.7 but still the order is less than three. without stating the remedies or trying them we move to implement the 2D acoustic equations considering linear advection as a special case.

6 System of Two Dimensional Linear Acoustic Equations

6.1 The Equations

As discussed earlier, acoustic equations is a system which shows relationship between pressure and velocity. Now in 2D, three equations comprise the system which are coupled in nature. The system is given by

$$\begin{cases} p_t + a_0 u_x + a_0 v_y = 0 \quad ; \\ u_t + a_0 p_x = 0 \quad ; \\ v_t + a_0 p_y = 0 \end{cases} \tag{6.1}$$

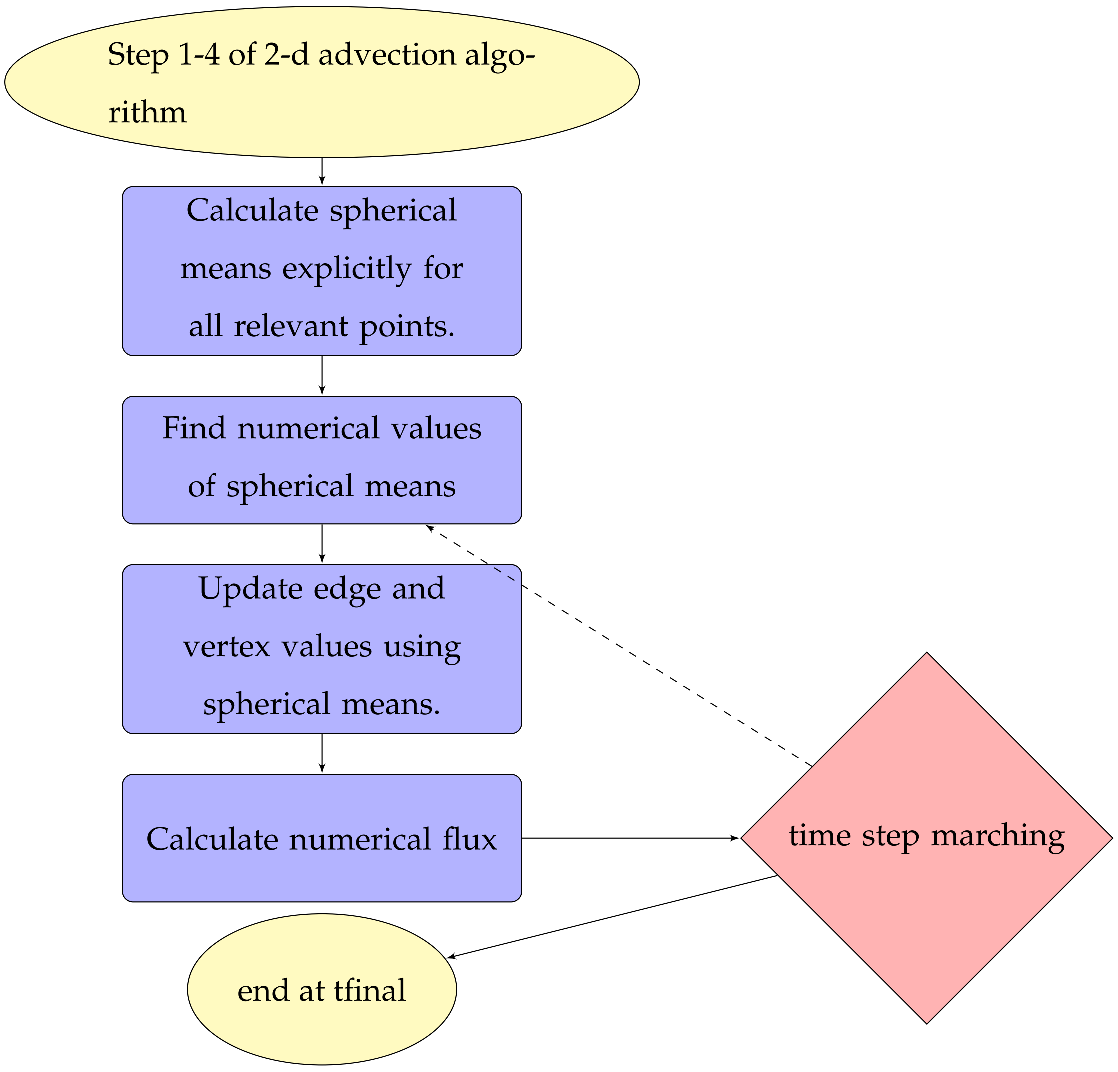
The equation must be complemented with an initial condition, like given by

$$\begin{cases} p(x, y, 0) = p_0(x, y) \quad ; \\ u(x, y, 0) = u_0(x, y) \quad ; \\ v(x, y, 0) = v_0(x, y) \end{cases} \tag{6.2}$$

Also a proper boundary condition is to be supplied accordingly. Decoupling strategy used to deal with the 1D acoustic case can not be utilised here, as the equations can not be decoupled in that manner. Hence a different approach known as *spherical means* is chosen for edge and vertex updates. First we have a look at the algorithm of 2D acoustic equations by AF scheme.

The Algorithm

6.2 Spherical Means-The Strategy to Solve Wave Equation



Step 1-4 of 2-d advection algorithm
Calculate spherical means explicitly for all relevant points.
Find numerical values of spherical means
Update edge and vertex values using spherical means.
Calculate numerical flux
time step marching
end at tfinal

The algorithm as expected is very similar to the 2D advection problem. However, the edge and vertex updates are now taken care of by the spherical means around disks. All rest steps are exactly the same.

Internal Reconstructions

Internal reconstruction is the same as described for the case of linear advection in two dimensions. just the difference lies in the fact that now three physical quantities have to be reconstructed inside every cell, rather than just one.

6.3 Edge and Vertex Update Using the Spherical Means

This section is the crust of the whole chapter. To update the edge and vertex solutions, exact solutions of spherical means are used. Which were derived in [20]. Actually, spherical means this name is misguiding. In Practice, this is nothing but a formula, which is of the exact solution to the initial-value problem for the scalar wave equation at an arbitrary point in space and time.

In layman language, we can understand this as we already have the exact solution formulas for the boundary DOF but we will rather use them in an interesting way to generate our numerical scheme for edges and vertices.

The formula for spherical mean for a function "M" in 2D is given by :

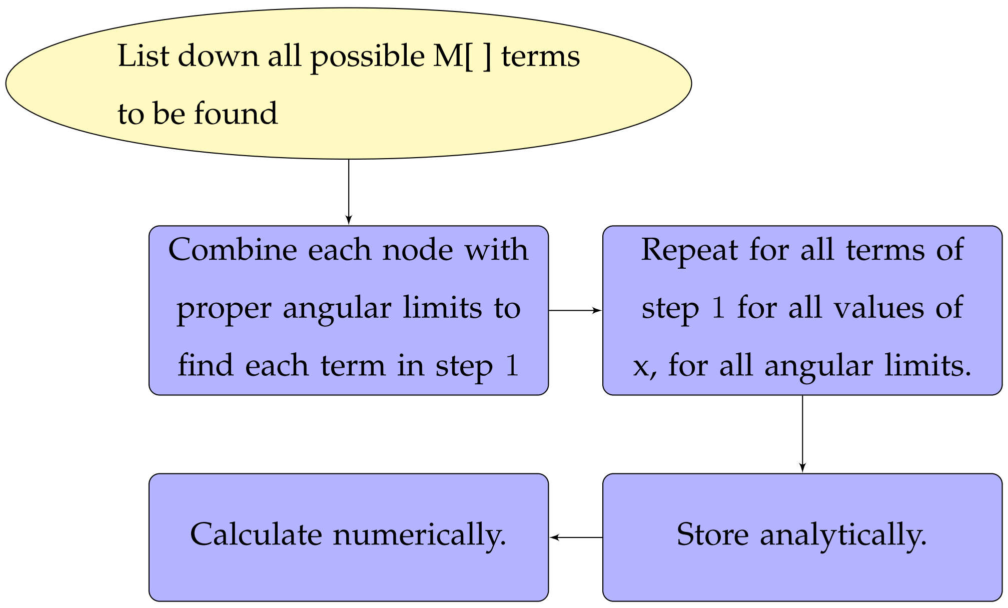
$$M[f](x, y, r) = \frac{1}{2\pi r} \int_0^{2\pi} \int_0^r f(x + s\cos(\phi), y + s\sin(\phi)) \frac{s}{\sqrt{r^2 - s^2}} \, ds \, d\phi. \tag{6.3}$$

When dealing with the continuous problem, the boundary elements of the cell are updated using the formula :

$$\begin{cases} p(t) = M_R[p] + R[\partial_R M_R[p] - M_R(div[\boldsymbol{u}])] \\ \boldsymbol{u}(t) = M_R[\boldsymbol{u}] + R[\partial_R M_R[\boldsymbol{u}] - M_R(\nabla p)] \end{cases} \tag{6.4}$$

When we have these values ready we can go onto update the cell average at next level.

But, the difficult task is to find all M[x] terms correctly. One way to do this is do has been written in the form of an algorithm in the flow chart. There can be several other ways that can be thought of to simplify the process and make the computation cheap.


List down all possible M[ ] terms to be found
Combine each node with proper angular limits to find each term in step 1
Repeat for all terms of step 1 for all values of x, for all angular limits.
Store analytically.
Calculate numerically.

6.4 Calculating Numerical Fluxes and Time Step Marching

Once the edge and vertex values have been calculated by the help of spherical means, the calculation of numerical flux and the update procedure of the physical quantities are pretty simple. They are a replica of what has been done for the advection in 2D. All the formulas of flux calculation and the final update formula remain the same.

6.5 Numerical Example

The order of accuracy of the acoustic method is confirmed by using the problem proposed by Morton [19]. With $a_0 = 1$, for $x \in [-2, 2]$ and $y \in [-2, 2]$. Exact solutions are implemented at the boundary. Numerical tests are presented at the time $1/8$, for 50 and 100 cells. Courant number is set to be 0.2.

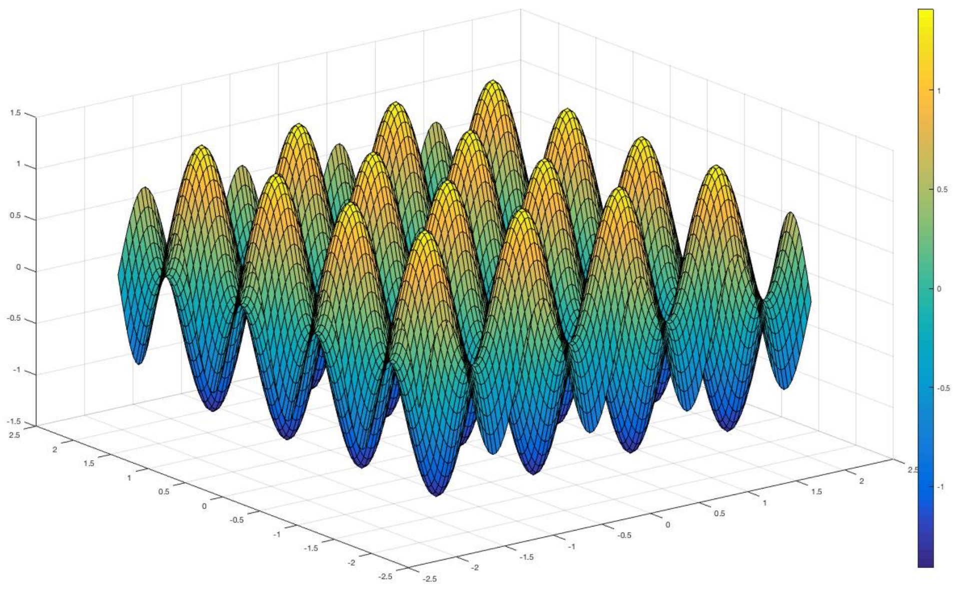


Figure 6.1: Pressure for *100* cells

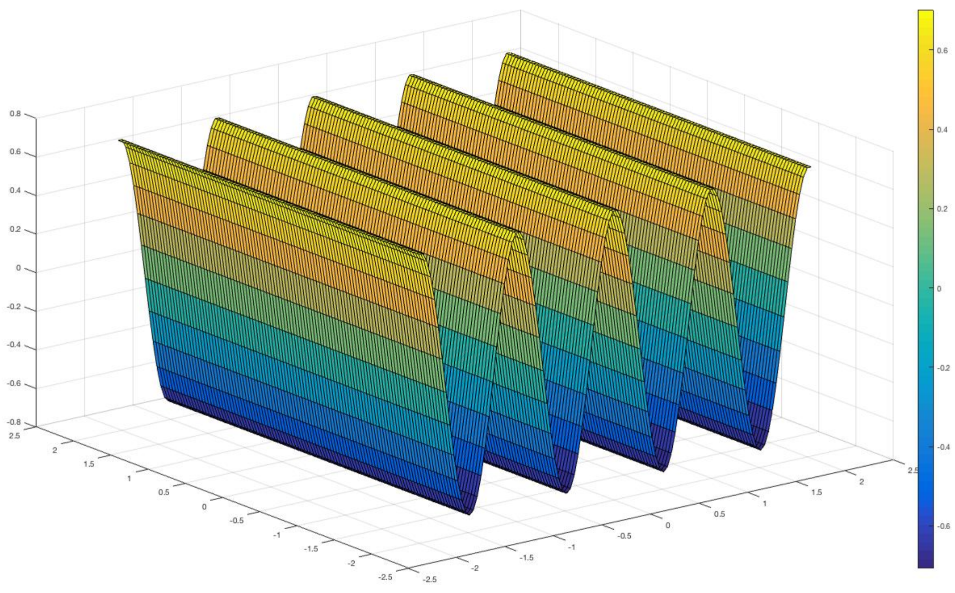


Figure 6.2: velocity *1* for *100* cells

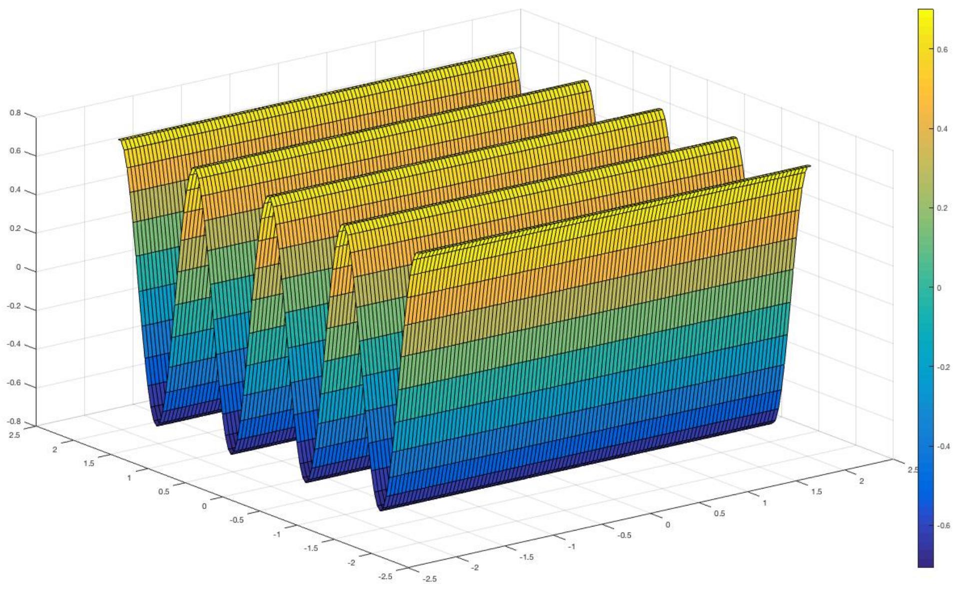


Figure 6.3: velocity *2* for *100* cells

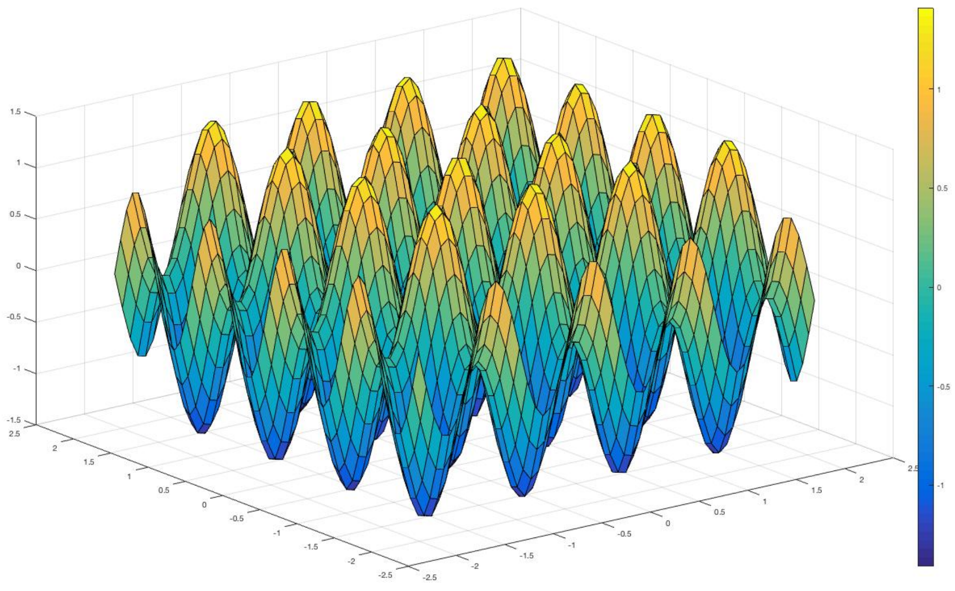


Figure 6.4: Pressure for *50* cells

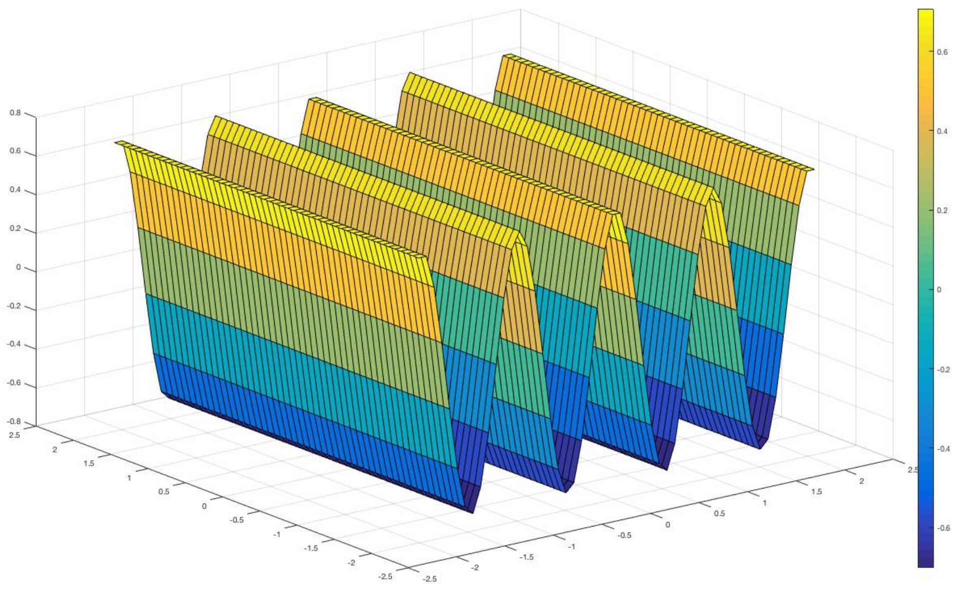


Figure 6.5: velocity *1* for *50* cells

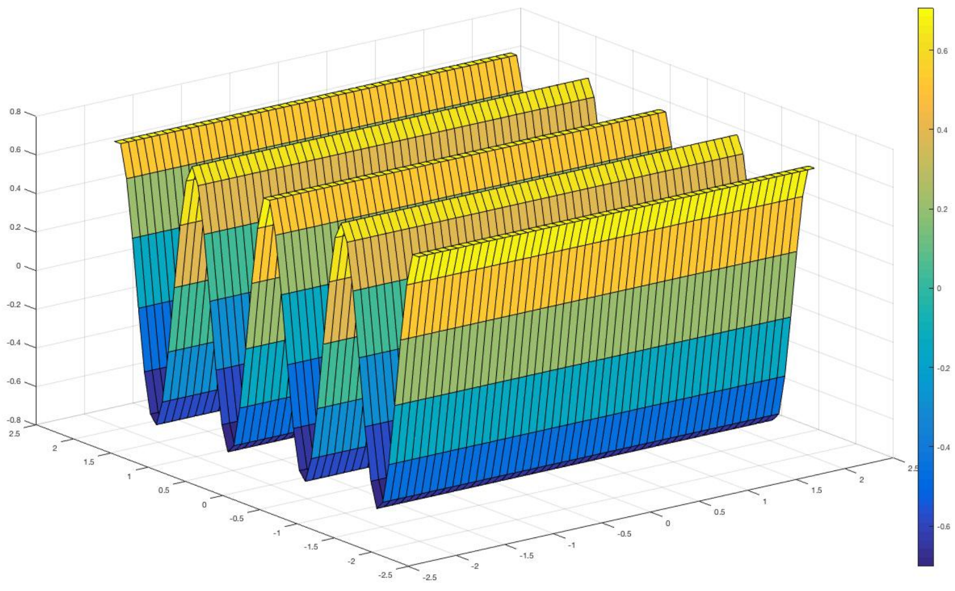


Figure 6.6: velocity *2* for *50* cells

7 Conclusions

7.1 Summary

This dissertation is principally concerned about the investigation of the headway of AF Schemes, a significant new class of techniques for explaining hyperbolic conservation laws, which are intently connected with the FV methods. In AF the edge and vertex values are developed free from the conserved cell quantities. Afterward, the interface values are utilized to compute the fluxes that moderately update the cell averages.

The AF scheme utilizes constant continuous internal reconstruction with a Lagrange basis function subject to each Cartesian cell with a cubic bubble function to update conservation in two dimensions which guarantees third order spatial exactness. Each reconstruction is subject to an individual component, which makes AF compact, conservative and unmistakably increasingly appropriate for irregular, unstructured lattices.

This project displays the usage of AF on the one dimension and the two-dimensional Cartesian grids for linear advection and linear acoustics equations, where the extension to three dimensions is straightforward. Test problems for each case have likewise been executed on Cartesian lattices. The AF method effectively accomplishes third order precision for every condition, for all courant number running in the range of zero and one. Moreover, the coding structure has been talked about in detail for the 2D usage of AF plans. Exhaustive points of interest in the utilization of Spherical means has been looked into and is enhanced with the Matlab and Mathematica codes.

7.2 Contributions and Conclusions

The significant contributions of this dissertation are from:

1. Extension of Eymann's one-dimensional Scheme [7] into a general multidimensional framework [8] applicable to linear conservation laws.
2. Using the idea presented in [13] to correctly find the order of accuracy in acoustics case for the continuous initial condition.
3. Use of spherical means to obtain evolution equations for physical systems that obey the scalar wave equation.

chapter 2 and chapter 3 demonstrate that the AF method accomplishes third order exactness utilizing two DOF per component in one dimension and three DOF in two measurements. AF is equivalent to a DG1 scheme, which utilizes a linear reconstruction. On the off chance that the DG method superconvergence, the work expected to figure the answer for the AF and DG techniques are around equivalent. The DG technique requires six DOF per component for third-order exactness, while the AF technique needs around three. The expanded number of DOF speaks to a substantial outstanding burden increment for non-super converging DG strategies contrasted with the third-request AF arrangement.

chapter 4 establishes a framework of coding structure required for further investigations.

chapter 5 and chapter 6 show the upside of non-conservative edge update. This element makes the novel limiting methodology conceivable and encourages the utilization of spherical means to ascertain edge and vertex values. The flexibility of the approach is additionally featured by appearing how trivial it is to stretch out the technique to the third measurement. The multidimensional treatment of the acoustics system allows the AF method to preserve excellent symmetry properties on the cartesian grid. Both the acoustics and advection results demonstrate that the AF scheme is stable provided that the the physical domain of dependence is contained within the nearest-neighbors of the point to be updated, giving the scheme the robustness and accuracy combination required of production-level codes.

7.3 Future Work

This project is intended to show the value of the AF technique and give instances of its execution. There is an abundance of research roads that may continue from this work.

- **Computational efficiency:** There are numerous chances to improve the effectiveness of the method. One example is numerically integrating the spherical mean formulas instead of utilizing the explicit structure. Likewise, any term in the point-update formulas that is $O(t^3)$ or higher can be ignored without losing third order exactness.
- **Three-dimensional work**: The majority of the center thoughts exhibited will reach out to three dimensions, be that as it may, new articulations for the bubble function and acoustic integrals should be determined.
- **Alternate applications**: This thesis represents the AF strategy as connected to the acoustics equations, however, the inference holds for any physical the framework that can be portrayed by the wave condition, for example, electromagnetics and linear elasticity equations
- **Multidimensional limiters**: limiters can be applied to both one dimensional

and two-dimensional cases. Throughout the discussion in this text, only continuous problems have been illustrated, to extend the ideas to discontinuous questions, limiters have to be applied to get solutions at boundary correctly.

- **Nonlinear equations**: Nonlinear equations such as Burger′s equations have not been dealt with in this text. Although Burger′s equation in one dimension has been carried out in but for two-dimensional case, a thorough study of shape-preserving limiters must be done to avoid spurious oscillations.

APPENDICES

APPENDIX A

A.1 MATLAB Coding

Coding of Plots Presented in chapter 2

```
% implementation of 1D linear Advection problem by Active Flux Scheme
% order of Accuracy is three
% formulas for scheme have been taken from eymann's thesis
% initial condition and problem has been taken from maeng's thesis
% wave speed is positive

function lahtek1
clear all;                                                % clearing workspace
clc;                                                      % clearing workspace
format short;                                             % for better accuracy
hold off;                                        % to not repeat previous curve
xa = 0.0;                                  % left limit of computational domain
xb = 1.0;                                 % right limit of computational domain
t = 0.0;                                                      % starting time
tend = 1.0;                                     % time at which soln is reqrd
No_Cell = 160;                                                 % number of cells
Delta_x = (xb-xa)/No_Cell;                                                % dx
Cell_Centers = xa-Delta_x/2:Delta_x:xb+Delta_x/2;               % cell centers
Cell_Interfaces = xa-Delta_x:Delta_x:xb+Delta_x;             % cell interfaces
n = size(Cell_Centers,2);                             % number of cell centers
m = size(Cell_Interfaces,2);                       % number of cell interfeces
u0 = (1/(2*pi))*sin(2*pi*Cell_Centers);                  % initial condition
v0 = (1/(2*pi))*sin(2*pi*Cell_Interfaces);               % initial condition
```

```
w0 = ( v0(1:m-1) + 4*u0(1:n)  + v0(2:m) )/6;     % constructing cell average
v1 = v0; w1=w0; v=0.7;                   % pre allocating v1 and w1 cfl =0.7

while(t<tend)
    dt = v * Delta_x;                                        % time step size
    dt = min(tend-dt,dt);                          % so that final time is tend
    %cell inrerface values update
    v1(3:m) = v*(3*v-2)*v0(2:m-1) + 6*v*(1-v)*w0(2:n)  + (1-v)*(1-3*v)*v0(3:m);
    v1(1) = v1(m-2); v1(2) = v1(m-1);        % periodic boundary conditions
    %cell average update
    w1(2:n-1) = v^2*(v-1)*v0(1:m-3) + v^2*(3-2*v)*w0(1:n-2) + v*(1-v)*v0(2:m-2) +
         (1-v)^2*(1+2*v)*w0(2:n-1) - v*(1-v)^2*v0(3:m-1);
    w1(1) = w1(n-1); w1(n) = w1(2);          % periodic boundary conditions
    v0 = v1; w0 = w1;                 % for next time step, to continue loop

    exact = (1/(2*pi))*sin(2*pi*(Cell_Centers-t-dt));

    %ploting of solution and exact solution;
    hold off;
    plot(Cell_Centers,w1,'o')
    grid on;
    hold on;
    plot(Cell_Centers,exact);
    ylabel('u','fontsize', 16)
    xlabel('x','fontsize', 16)
    title(sprintf('time = %f',t),'fontsize',16)
    drawnow;
    t=t+dt;
end
L2=sqrt((1/((xb-xa+2*Delta_x)*(dt))).*sum(abs(w1*Delta_x*dt-exact*Delta_x*dt).^2)
    )
```

Coding of Plots Presented in chapter 3

```
% solving the 1d acoustics equation for simple wave
% problem is 1d acoustic with a0 = 1; rho = 1/2;
% ic is p'(x,0) = sin(pi*x)/2 ; u'(x,0) = -sin(pi*x)

% implementation of 1D linear Advection problem by Active Flux Scheme
% order of Accuracy is three
% formulas for scheme have been taken from eymann's thesis
% initial condition and problem has been taken from maeng's thesis
% wave speed is negative

function one_d_acoustic_1
clear all;                                                      % clearing workspace
clc;                                                            % clearing workspace
format long;                                                   % for better accuracy
hold off;                                             % to not repeat previous curve
xa = 0.0;                                       % left limit of computational domain
xb = 2.0;                                      % right limit of computational domain
t = 0.0;                                                             % starting time
tend = 10.0;                                          % time at which soln is reqrd
No_Cell = 50;                                                      % number of cells
Delta_x = (xb-xa)/No_Cell;                                                      % dx
Cell_Centers = xa-Delta_x/2:Delta_x:xb+Delta_x/2;                     % cell centers
Cell_Interfaces = xa-Delta_x:Delta_x:xb+Delta_x;                   % cell interfaces
n = size(Cell_Centers,2);                                   % number of cell centers
m = size(Cell_Interfaces,2);                             % number of cell interfeces
u0 = -sqrt(2)*sin(pi*Cell_Centers);                          % initial condition
v0 = -sqrt(2)*sin(pi*Cell_Interfaces);                       % initial condition
w0 = ( v0(1:m-1) + 4*u0(1:n)  + v0(2:m) )/6;     % constructing cell average

U0 = 0*Cell_Centers;                   % initial condition
V0 = 0*Cell_Interfaces;                % initial condition
W0 = ( V0(1:m-1) + 4*U0(1:n)  + V0(2:m) )/6;     % constructing cell average

v1 = v0; w1=w0; v=0.7;                      % pre allocating v1 and w1 cfl =0.7
V1 = V0; W1=W0;

while(t<tend)
    dt = v * Delta_x;                                               % time step size
    dt = min(tend-dt,dt);                              % so that final time is tend
    % cell inrerface values update
    v1(1:m-2) = (1-v)*(1-3*v)*v0(1:m-2) + 6*v*(1-v)*w0(1:n-1)  + v*(3*v-2)*v0(2:m
        -1);
    v1(m) = v1(3); v1(m-1) = v1(2);         % periodic boundary conditions
    % cell average update
    w1(2:n-1) = -v*(1-v)^2*v0(2:m-2) + (1-v)^2*(1+2*v)*w0(2:n-1) + v*(1-v)*v0(3:m
         -1) + v^2*(3-2*v)*w0(3:n) + v^2*(v-1)*v0(4:m);
    w1(1) = w1(n-1); w1(n) = w1(2);          % periodic boundary conditions
    v0 = v1; w0 = w1;                % for next time step, to continue loop



    % cell inrerface values update
    V1(3:m) = v*(3*v-2)*V0(2:m-1) + 6*v*(1-v)*W0(2:n)  + (1-v)*(1-3*v)*V0(3:m);
    V1(1) = V1(m-2); V1(2) = V1(m-1);        % periodic boundary conditions
    % cell average update
    W1(2:n-1) = v^2*(v-1)*V0(1:m-3) + v^2*(3-2*v)*W0(1:n-2) + v*(1-v)*V0(2:m-2) +
         (1-v)^2*(1+2*v)*W0(2:n-1) - v*(1-v)^2*V0(3:m-1);
    W1(1) = W1(n-1); W1(n) = W1(2);          % periodic boundary conditions
    V0 = V1; W0 = W1;                % for next time step, to continue loop

    exact_neg = -sqrt(2)*sin(pi*(Cell_Centers+t+dt));
    exact_pos = U0;

    pressure = (-w1 + W1)/(2*sqrt(2));
    velocity = (w1 + W1)/sqrt(2);

    exact_pressure = (-exact_neg + exact_pos)/(2*sqrt(2));
    exact_velocity = (exact_neg + exact_pos)/sqrt(2);

    %ploting of solution and exact solution;
    figure(1)
    hold off;
    plot(Cell_Centers,pressure,'o')
    grid on;
    hold on;
    plot(Cell_Centers,exact_pressure);
    ylabel('p','fontsize', 16)
    xlabel('x','fontsize', 16)
    title(sprintf('time = %f',t),'fontsize',16)
    drawnow;

    figure(2)
    hold off;
    plot(Cell_Centers,velocity,'o')
    grid on;
```

```
    hold on;
    plot(Cell_Centers,exact_velocity);
    ylabel('u','fontsize', 16)
    xlabel('x','fontsize', 16)
    title(sprintf('time = %f',t),'fontsize',16)
    drawnow;
    t=t+dt;
end
L2_pressure=sqrt((1/((xb-xa+2*Delta_x)*(dt))).*sum(abs(pressure*Delta_x*dt-
    exact_pressure*Delta_x*dt).^2))
L2_velocity=sqrt((1/((xb-xa+2*Delta_x)*(dt))).*sum(abs(velocity*Delta_x*dt-
    exact_velocity*Delta_x*dt).^2))
```

Coding of Plots Presented in chapter 5

```
% two dimensional advection code solved by active flux method
% desired order of accuracy is between 2.5 and 3
% solving u_t + u_x + u_y = 0
% ic is gaussian type u(x,y,0) = exp(-0.5*(x^2+y^2))
% bc depending on the advection is applied
% cartesian mesh is used

clear all;                                                      % clearing workspace
clc;                                                            % clearing workspace
format long                                                    % for better accuracy
xa = -5;                                                      % left limit of domain
xb = 5;                                                      % right limit of domain
ya = -5;                                                     % below limit of domain
yb = 5;                                                        % top limit of domain
No_Cell_x = 50;                                               % no of cells along x
No_Cell_y = 50;                                               % no of cells along y
Delta_x = (xb-xa)/No_Cell_x;                                             % step size
Delta_y = (yb-ya)/No_Cell_y;                                             % step size
v = 0.7;                                                                % cfl number
t = 0;                                                                  % time start
tend = 1;                                                                 % time end
dt = (v*Delta_x*Delta_y)/(Delta_x+Delta_y);                         % cfl condition
lambda = [1;1];                                                         % wave speed

dfbx = xa - Delta_x/2 : Delta_x : xb + Delta_x/2;   % descritization for black
```

```
    dot
dfby = ya - Delta_y/2 : Delta_y : yb + Delta_y/2;   % descritization for black
    dot

dfcx = xa - Delta_x : Delta_x : xb + Delta_x;    % descritization for cross
dfcy = ya - Delta_y : Delta_y : yb + Delta_y;    % descritization for cross

dfsx = xa - 3*Delta_x/2 : Delta_x : xb + Delta_x/2;% descritization for star
dfsy = ya - Delta_y : Delta_y : yb + Delta_y;      % descritization for star

dfwx = xa - Delta_x : Delta_x : xb + Delta_x;       % descritization for white
    dot
dfwy = ya - 3*Delta_y/2 : Delta_y : yb + Delta_y/2; % descritization for white
    dot

[X1,Y1] = meshgrid(dfbx,dfby);                              % creating grid
[X2,Y2] = meshgrid(dfcx,dfcy);                              % creating grid
[X3,Y3] = meshgrid(dfsx,dfsy);                              % creating grid
[X4,Y4] = meshgrid(dfwx,dfwy);                              % creating grid

lx1 = length(X1);
lx2 = length(X2);
lx3 = length(X3);
lx4 = length(X4);
%========================= work for IC (verified)=========================
u0 = zeros(lx1,lx1);                                       % pre allocating
v0 = zeros(lx2,lx2);                                       % pre allocating
w0 = zeros(lx3,lx3);                                       % pre allocating
z0 = zeros(lx4,lx4);                                       % pre allocating
exact = zeros(lx1,lx1);                                    % pre allocating

sizeu = size(u0,1)*size(u0,2);
sizev = size(v0,1)*size(v0,2);
sizew = size(w0,1)*size(w0,2);
sizez = size(z0,1)*size(z0,2);

for i=1:sizeu
    u0(i) = exp(-0.5*(X1(i)^2 + Y1(i)^2)); % initial condition on black dot(u)
end
```

```
for i=1:sizev
    v0(i) = exp(-0.5*(X2(i)^2 + Y2(i)^2)); % initial condition on cross(v)
end

for i=1:sizew
    w0(i) = exp(-0.5*(X3(i)^2 + Y3(i)^2)); % initial condition on star(w)
end

for i=1:sizez
    z0(i) = exp(-0.5*(X4(i)^2 + Y4(i)^2)); % initial condition on white dot(z)
end

w0 = w0(:,2:end);
z0 = z0(2:end,:);

%initial condition plot
figure(1)
surf(X1,Y1,u0)
%view(2)
colorbar
drawnow

%===========================================================================

% finally stored only the grid points with IC not anything outside

%=================== length of each face (verified)========================
No_of_edges = 4*sizeu;                   % number of edges with repetition
l = Delta_x;                                           % length of one edge
length_of_face = l*ones(4*sizeu,1);           % storing all length together
%===========================================================================

%========================= work for normals (verified)=====================
n1 = zeros(4*sizeu,1);          % calculating first column of normal vectors
n1(1:4:end) = 0;
n1(2:4:end) = 1;
n1(3:4:end) = 0;
n1(4:4:end) = -1;
```

```
n2 = zeros(4*sizeu,1);        % calculating second column of normal vectors
n2(1:4:end) = -1;
n2(2:4:end) = 0;
n2(3:4:end) = 1;
n2(4:4:end) = 0;
normals = [n1 n2];                                % normal vector through faces
%============================================================================


%========================= work for area (verified)========================
ar = Delta_x*Delta_y;                                       % area of each grid
area = ar*ones(sizeu,1);                    % storing all grid areas together
%============================================================================

jac_inv = [2/Delta_x 0;0 2/Delta_y];          % inverse jacobian calculation
minus_part = dt*jac_inv*lambda;                         % minus part for xi_f

% pre allocating xi_i follows

xi_i_6_x_half = 1;
xi_i_6_y_half = 0;

xi_i_3_x_half = 1;
xi_i_3_y_half = 1;

xi_i_7_x_half = 0;
xi_i_7_y_half = 1;

xi_i_6_x_full = 1;
xi_i_6_y_full = 0;

xi_i_3_x_full = 1;
xi_i_3_y_full = 1;

xi_i_7_x_full = 0;
xi_i_7_y_full = 1;

% calculation of xi_f follows
```

```
xi_f_6_x_half = xi_i_6_x_half - minus_part(1)/2;
xi_f_6_y_half = xi_i_6_y_half - minus_part(2)/2;

xi_f_3_x_half = xi_i_3_x_half - minus_part(1)/2;
xi_f_3_y_half = xi_i_3_y_half - minus_part(2)/2;

xi_f_7_x_half = xi_i_7_x_half - minus_part(1)/2;
xi_f_7_y_half = xi_i_7_y_half - minus_part(2)/2;

xi_f_6_x_full = xi_i_6_x_full - minus_part(1);
xi_f_6_y_full = xi_i_6_y_full - minus_part(2);

xi_f_3_x_full = xi_i_3_x_full - minus_part(1);
xi_f_3_y_full = xi_i_3_y_full - minus_part(2);

xi_f_7_x_full = xi_i_7_x_full - minus_part(1);
xi_f_7_y_full = xi_i_7_y_full - minus_part(2);

% allocating values on all four type of points
u0 = u0;
u1 = v0(1:end-1,1:end-1);
u2 = v0(1:end-1,2:end);
u3 = v0(2:end,2:end);
u4 = v0(2:end,1:end-1);
u5 = w0(1:end-1,:);
u6 = z0(:,2:end);
u7 = w0(2:end,:);
u8 = z0(:,1:end-1);

% calculating cell average
u9 = ( u1+u2+u3+u4 + 4*(u5+u6+u7+u8) +16*u0 )/36;

% pre allocating the values at half time step
u1_1 = u1;
u2_1 = u2;
u3_1 = u3;
u4_1 = u4;
u5_1 = u5;
```

```
u6_1 = u6;
u7_1 = u7;
u8_1 = u8;

% pre allocating vales at full time step
u1_2 = u1;
u2_2 = u2;
u3_2 = u3;
u4_2 = u4;
u5_2 = u5;
u6_2 = u6;
u7_2 = u7;
u8_2 = u8;
u9_2 = u9;

% loop starts
for ii = t:dt:tend
    % calculating the half time interface update of pt. 6
    u6_1 = 0.25*(xi_f_6_x_half-1)*(xi_f_6_y_half-1)*(xi_f_6_x_half)*(
        xi_f_6_y_half)*u1 + 0.25*(xi_f_6_x_half+1)*(xi_f_6_y_half-1)*(
        xi_f_6_x_half)*(xi_f_6_y_half)*u2 + 0.25*(xi_f_6_x_half+1)*(xi_f_6_y_half
        +1)*(xi_f_6_x_half)*(xi_f_6_y_half)*u3 + 0.25*(xi_f_6_x_half-1)*(
        xi_f_6_y_half+1)*(xi_f_6_x_half)*(xi_f_6_y_half)*u4 + 0.5*(1-
        xi_f_6_x_half^2)*(xi_f_6_y_half-1)*(xi_f_6_y_half)*u5 + 0.5*(1-
        xi_f_6_y_half^2)*(xi_f_6_x_half+1)*(xi_f_6_x_half)*u6 + 0.5*(1-
        xi_f_6_x_half^2)*(xi_f_6_y_half+1)*(xi_f_6_y_half)*u7 + 0.5*(1-
        xi_f_6_y_half^2)*(xi_f_6_x_half-1)*(xi_f_6_x_half)*u8 + (1-xi_f_6_x_half
        ^2)*(1-xi_f_6_y_half^2)*u9;
    % calculating the full time interface update of pt. 6
    u6_2 = 0.25*(xi_f_6_x_full-1)*(xi_f_6_y_full-1)*(xi_f_6_x_full)*(
        xi_f_6_y_full)*u1 + 0.25*(xi_f_6_x_full+1)*(xi_f_6_y_full-1)*(
        xi_f_6_x_full)*(xi_f_6_y_full)*u2 + 0.25*(xi_f_6_x_full+1)*(xi_f_6_y_full
        +1)*(xi_f_6_x_full)*(xi_f_6_y_full)*u3 + 0.25*(xi_f_6_x_full-1)*(
        xi_f_6_y_full+1)*(xi_f_6_x_full)*(xi_f_6_y_full)*u4 + 0.5*(1-
        xi_f_6_x_full^2)*(xi_f_6_y_full-1)*(xi_f_6_y_full)*u5 + 0.5*(1-
        xi_f_6_y_full^2)*(xi_f_6_x_full+1)*(xi_f_6_x_full)*u6 + 0.5*(1-
        xi_f_6_x_full^2)*(xi_f_6_y_full+1)*(xi_f_6_y_full)*u7 + 0.5*(1-
        xi_f_6_y_full^2)*(xi_f_6_x_full-1)*(xi_f_6_x_full)*u8 + (1-xi_f_6_x_full
        ^2)*(1-xi_f_6_y_full^2)*u9;
```

```
% calculating the half time interface update of pt. 3
u3_1 = 0.25*(xi_f_3_x_half-1)*(xi_f_3_y_half-1)*(xi_f_3_x_half)*(
    xi_f_3_y_half)*u1 + 0.25*(xi_f_3_x_half+1)*(xi_f_3_y_half-1)*(
    xi_f_3_x_half)*(xi_f_3_y_half)*u2 + 0.25*(xi_f_3_x_half+1)*(xi_f_3_y_half
    +1)*(xi_f_3_x_half)*(xi_f_3_y_half)*u3 + 0.25*(xi_f_3_x_half-1)*(
    xi_f_3_y_half+1)*(xi_f_3_x_half)*(xi_f_3_y_half)*u4 + 0.5*(1-
    xi_f_3_x_half^2)*(xi_f_3_y_half-1)*(xi_f_3_y_half)*u5 + 0.5*(1-
    xi_f_3_y_half^2)*(xi_f_3_x_half+1)*(xi_f_3_x_half)*u6 + 0.5*(1-
    xi_f_3_x_half^2)*(xi_f_3_y_half+1)*(xi_f_3_y_half)*u7 + 0.5*(1-
    xi_f_3_y_half^2)*(xi_f_3_x_half-1)*(xi_f_3_x_half)*u8 + (1-xi_f_3_x_half
    ^2)*(1-xi_f_3_y_half^2)*u9;
% calculating the full time interface update of pt. 3
u3_2 = 0.25*(xi_f_3_x_full-1)*(xi_f_3_y_full-1)*(xi_f_3_x_full)*(
    xi_f_3_y_full)*u1 + 0.25*(xi_f_3_x_full+1)*(xi_f_3_y_full-1)*(
    xi_f_3_x_full)*(xi_f_3_y_full)*u2 + 0.25*(xi_f_3_x_full+1)*(xi_f_3_y_full
    +1)*(xi_f_3_x_full)*(xi_f_3_y_full)*u3 + 0.25*(xi_f_3_x_full-1)*(
    xi_f_3_y_full+1)*(xi_f_3_x_full)*(xi_f_3_y_full)*u4 + 0.5*(1-
    xi_f_3_x_full^2)*(xi_f_3_y_full-1)*(xi_f_3_y_full)*u5 + 0.5*(1-
    xi_f_3_y_full^2)*(xi_f_3_x_full+1)*(xi_f_3_x_full)*u6 + 0.5*(1-
    xi_f_3_x_full^2)*(xi_f_3_y_full+1)*(xi_f_3_y_full)*u7 + 0.5*(1-
    xi_f_3_y_full^2)*(xi_f_3_x_full-1)*(xi_f_3_x_full)*u8 + (1-xi_f_3_x_full
    ^2)*(1-xi_f_3_y_full^2)*u9;

% calculating the half time interface update of pt. 7
u7_1 = 0.25*(xi_f_7_x_half-1)*(xi_f_7_y_half-1)*(xi_f_7_x_half)*(
    xi_f_7_y_half)*u1 + 0.25*(xi_f_7_x_half+1)*(xi_f_7_y_half-1)*(
    xi_f_7_x_half)*(xi_f_7_y_half)*u2 + 0.25*(xi_f_7_x_half+1)*(xi_f_7_y_half
    +1)*(xi_f_7_x_half)*(xi_f_7_y_half)*u3 + 0.25*(xi_f_7_x_half-1)*(
    xi_f_7_y_half+1)*(xi_f_7_x_half)*(xi_f_7_y_half)*u4 + 0.5*(1-
    xi_f_7_x_half^2)*(xi_f_7_y_half-1)*(xi_f_7_y_half)*u5 + 0.5*(1-
    xi_f_7_y_half^2)*(xi_f_7_x_half+1)*(xi_f_7_x_half)*u6 + 0.5*(1-
    xi_f_7_x_half^2)*(xi_f_7_y_half+1)*(xi_f_7_y_half)*u7 + 0.5*(1-
    xi_f_7_y_half^2)*(xi_f_7_x_half-1)*(xi_f_7_x_half)*u8 + (1-xi_f_7_x_half
    ^2)*(1-xi_f_7_y_half^2)*u9;
% calculating the full time interface update of pt. 7
u7_2 = 0.25*(xi_f_7_x_full-1)*(xi_f_7_y_full-1)*(xi_f_7_x_full)*(
    xi_f_7_y_full)*u1 + 0.25*(xi_f_7_x_full+1)*(xi_f_7_y_full-1)*(
    xi_f_7_x_full)*(xi_f_7_y_full)*u2 + 0.25*(xi_f_7_x_full+1)*(xi_f_7_y_full
```

```
        +1)*(xi_f_7_x_full)*(xi_f_7_y_full)*u3 + 0.25*(xi_f_7_x_full-1)*(
        xi_f_7_y_full+1)*(xi_f_7_x_full)*(xi_f_7_y_full)*u4 + 0.5*(1-
        xi_f_7_x_full^2)*(xi_f_7_y_full-1)*(xi_f_7_y_full)*u5 + 0.5*(1-
        xi_f_7_y_full^2)*(xi_f_7_x_full+1)*(xi_f_7_x_full)*u6 + 0.5*(1-
        xi_f_7_x_full^2)*(xi_f_7_y_full+1)*(xi_f_7_y_full)*u7 + 0.5*(1-
        xi_f_7_y_full^2)*(xi_f_7_x_full-1)*(xi_f_7_x_full)*u8 + (1-xi_f_7_x_full
        ^2)*(1-xi_f_7_y_full^2)*u9;

    % updating pt. 1 half time step
    u1_1(2:end,2:end) = u3_1(1:end-1,1:end-1);
    u1_1(1,:) = u3_1(:,end)'; % bc
    u1_1(:,1) = u3_1(end,:)'; % bc

    % updating pt. 1 full time step
    u1_2(2:end,2:end) = u3_2(1:end-1,1:end-1);
    u1_2(1,:) = u3_2(:,end)'; % bc
    u1_2(:,1) = u3_2(end,:)'; % bc

    % updating pt. 5 half time step
    u5_1(2:end,:) = u7_1(1:end-1,:);
    u5_1(1,:) = u7_1(:,end)'; % bc

    % updating pt. 5 full time step
    u5_2(2:end,:) = u7_2(1:end-1,:);
    u5_2(1,:) = u7_2(:,end)'; % bc

    % updating pt. 2 half time step
    u2_1(2:end,:) = u3_1(1:end-1,:);
    u2_1(1,:) = u3_1(:,end)'; % bc

    % updating pt. 2 full time step
    u2_2(2:end,:) = u3_2(1:end-1,:);
    u2_2(1,:) = u3_2(:,end)'; % bc

    % updating pt. 4 half time step
    u4_1(:,2:end) = u3_1(:,1:end-1);
    u4_1(:,1) = u3_1(end,:)'; % bc

    % updating pt. 4 full time step
    u4_2(:,2:end) = u3_2(:,1:end-1);
    u4_2(:,1) = u3_2(end,:)'; % bc

    % updating pt. 8 half time step
    u8_1(:,2:end) = u6_1(:,1:end-1);
    u8_1(:,1) = u6_1(end,:)'; % bc

    % updating pt. 8 full time step
    u8_2(:,2:end) = u6_2(:,1:end-1);
    u8_2(:,1) = u6_2(end,:)'; % bc

    % parameters for flux through face one

    uln1 = u1;
    umn1 = u5;
    urn1 = u2;
    ulnplushalf1 = u1_1;
    umnplushalf1 = u5_1;
    urnplushalf1 = u2_1;
    ulnplusone1 = u1_2;
    umnplusone1 = u5_2;
    urnplusone1 = u2_2;

    % parameters for flux through face three

    uln3 = u2;
    umn3 = u6;
    urn3 = u3;
    ulnplushalf3 = u2_1;
    umnplushalf3 = u6_1;
    urnplushalf3 = u3_1;
    ulnplusone3 = u2_2;
    umnplusone3 = u6_2;
    urnplusone3 = u3_2;

    % parameters for flux through face two

    uln2 = u3;
    umn2 = u7;
    urn2 = u4;
    ulnplushalf2 = u3_1;
    umnplushalf2 = u7_1;
    urnplushalf2 = u4_1;
    ulnplusone2 = u3_2;
    umnplusone2 = u7_2;
    urnplusone2 = u4_2;

    % parameters for flux through face four

    uln4 = u4;
    umn4 = u8;
    urn4 = u1;
    ulnplushalf4 = u4_1;
    umnplushalf4 = u8_1;
    urnplushalf4 = u1_1;
    ulnplusone4 = u4_2;
    umnplusone4 = u8_2;
    urnplusone4 = u1_2;

    % flux through face 1
    flux1 = (( uln1 + urn1 + ulnplusone1 + urnplusone1 ) + 4*( umn1 + umnplusone1
         + ulnplushalf1 + urnplushalf1 ) + 16*umnplushalf1 )/36;

    % flux through face 2
    flux2 = (( uln2 + urn2 + ulnplusone2 + urnplusone2 ) + 4*( umn2 + umnplusone2
         + ulnplushalf2 + urnplushalf2 ) + 16*umnplushalf2 )/36;

    % flux through face 3
    flux3 = (( uln3 + urn3 + ulnplusone3 + urnplusone3 ) + 4*( umn3 + umnplusone3
         + ulnplushalf3 + urnplushalf3 ) + 16*umnplushalf3 )/36;

    % flux through face 4
    flux4 = (( uln4 + urn4 + ulnplusone4 + urnplusone4 ) + 4*( umn4 + umnplusone4
         + ulnplushalf4 + urnplushalf4 ) + 16*umnplushalf4 )/36;

    %flux1(1,:) = flux2(:,end);
    %flux4(:,1) = flux3(end,:);
```

```
sz = size(flux1,1)*size(flux1,2);
flux = zeros(4*sz,1);

flux(1:4:end) = flux1;
flux(2:4:end) = flux2;
flux(3:4:end) = flux3;
flux(4:4:end) = flux4;
final_flux = [flux flux];

flux_dot_normals = final_flux.*normals;
final_dot = flux_dot_normals(:,1) + flux_dot_normals(:,2);
under_summation = final_dot.*length_of_face;

for i=1:sizeu
    total(i) = under_summation(4*i-3) + under_summation(4*i-2) +
        under_summation(4*i-1) + under_summation(4*i);
end

for i=1:sizeu
    u9_2(i) = u9(i) -(dt/area(i))*total(i);
end

%u9_2(1,:) = u9_2(:,end)';
%u9_2(1,:) = u9_2(end,:)';

u9 = u9_2;
u1 = u1_2;
u2 = u2_2;
u3 = u3_2;
u4 = u4_2;
u5 = u5_2;
u6 = u6_2;
u7 = u7_2;
u8 = u8_2;

hold off;
figure(2)
surf(X1,Y1,u9_2)
%view(2)
```

```
    colorbar
    drawnow
end

for i=1:sizeu
    exact(i) = exp(-0.5*((X1(i)-1)^2 + (Y1(i)-1)^2)); % initial condition on
        black dot(u)
end
u9_2;
exact;
forl2 = Delta_x*Delta_y*dt*ones(lx1,lx1);
L2=sqrt((1/((xb-xa+2*Delta_x)*(yb-ya+2*Delta_y)*(dt))).*sum(sum(abs(u9_2.*forl2-
    exact.*forl2).^2)))
ncell = sizeu;
nface = No_of_edges;
nvertex = sizev;
DOF = ncell + nface/2 + nvertex/4
h = 1/sqrt(DOF)
```

Coding of Plots Presented in chapter 6

```
% System of two dimensional Linear Acoustic Equations solved by Active Flux
% Scheme
% desired order of accuracy is 3
% solving p_t + div(v) = 0; v_t + grad(p) = 0;
% IC is p(x,y,0) = [sin(2*pi*x) + sin(2*pi*y)]/c ; u(x,y,0) = 0 = v(x,y,0)
% exact solution is implemented on boundary
% cartesian mesh is used

clear all;                                                  % clearing workspace
clc;                                                        % clearing workspace
format short                                               % for better accuracy
xa = -1;                                                  % left limit of domain
xb = 1;                                                  % right limit of domain
ya = -1;                                                 % below limit of domain
yb = 1;                                                    % top limit of domain
No_Cell_x = 160;                                            % no of cells along x
No_Cell_y = 160;                                            % no of cells along y
Delta_x = (xb-xa)/No_Cell_x;                                         % step size
Delta_y = (yb-ya)/No_Cell_y;                                         % step size
```

```
v = 0.5;                                                              % cfl number
dt = (v*Delta_x*Delta_y)/(Delta_x+Delta_y);                        % cfl condition
t = dt;                                                                %time start
tend = 0.1;                                                            % time end
g = Delta_x;            % to substitute in values obtained from mathematica
r = dt;                 % to substitute in values obtained from mathematica

dfbx = xa - Delta_x/2 : Delta_x : xb + Delta_x/2;   % descritization for black
    dot
dfby = ya - Delta_y/2 : Delta_y : yb + Delta_y/2;   % descritization for black
    dot

dfcx = xa - Delta_x : Delta_x : xb + Delta_x;    % descritization for cross
dfcy = ya - Delta_y : Delta_y : yb + Delta_y;    % descritization for cross

dfsx = xa - 3*Delta_x/2 : Delta_x : xb + Delta_x/2; % descritization for star
dfsy = ya - Delta_y : Delta_y : yb + Delta_y;       % descritization for star

dfwx = xa - Delta_x : Delta_x : xb + Delta_x;       % descritization for white
    dot
dfwy = ya - 3*Delta_y/2 : Delta_y : yb + Delta_y/2; % descritization for white
    dot

[X1,Y1] = meshgrid(dfbx,dfby);                              % creating grid
[X2,Y2] = meshgrid(dfcx,dfcy);                              % creating grid
[X3,Y3] = meshgrid(dfsx,dfsy);                              % creating grid
[X4,Y4] = meshgrid(dfwx,dfwy);                              % creating grid

lx1 = length(X1);
lx2 = length(X2);
lx3 = length(X3);
lx4 = length(X4);
%========================= work for IC (verified)=========================
u10 = zeros(lx1,lx1);                                       % pre allocating
    pressure
v10 = zeros(lx2,lx2);                                       % pre allocating
    pressure
w10 = zeros(lx3,lx3);                                       % pre allocating
    pressure
```

```
z10 = zeros(lx4,lx4);                                     % pre allocating
    pressure

u20 = zeros(lx1,lx1);                                     % pre allocating vel1
v20 = zeros(lx2,lx2);                                     % pre allocating vel1
w20 = zeros(lx3,lx3);                                     % pre allocating vel1
z20 = zeros(lx4,lx4);                                     % pre allocating vel1

u30 = zeros(lx1,lx1);                                     % pre allocating vel2
v30 = zeros(lx2,lx2);                                     % pre allocating vel2
w30 = zeros(lx3,lx3);                                     % pre allocating vel2
z30 = zeros(lx4,lx4);                                     % pre allocating vel2

exactp = zeros(lx1,lx1);                                  % pre allocating
    exact pressure
exactvel1 = zeros(lx1,lx1);                               % pre allocating
    exact vel1
exactvel2 = zeros(lx1,lx1);                               % pre allocating
    exact vel2

sizeu = size(u10,1)*size(u10,2);
sizev = size(v10,1)*size(v10,2);
sizew = size(w10,1)*size(w10,2);
sizez = size(z10,1)*size(z10,2);

% initial condition for pressure

for i=1:sizeu
    u10(i) = sin(2*pi*X1(i)) + sin(2*pi*Y1(i)); % initial condition on black dot(
        u) pressure
end

for i=1:sizev
    v10(i) = sin(2*pi*X2(i)) + sin(2*pi*Y2(i)); % initial condition on cross(v)
        pressure
end

for i=1:sizew
    w10(i) = sin(2*pi*X3(i)) + sin(2*pi*Y3(i)); % initial condition on star(w)
```

```
        pressure
end

for i=1:sizez
    z10(i) = sin(2*pi*X4(i)) + sin(2*pi*Y4(i)); % initial condition on white dot(
        z) pressure
end

w10 = w10(:,2:end);
z10 = z10(2:end,:);

% initial condition for velocity 1

for i=1:sizeu
    u20(i) = 0; % initial condition on black dot(u) vel1
end

for i=1:sizev
    v20(i) = 0; % initial condition on cross(v) vel1
end

for i=1:sizew
    w20(i) = 0; % initial condition on star(w) vel1
end

for i=1:sizez
    z20(i) = 0; % initial condition on white dot(z) vel1
end

w20 = w20(:,2:end);
z20 = z20(2:end,:);

% initial condition for velocity 2

for i=1:sizeu
    u30(i) = 0; % initial condition on black dot(u) vel2
end

for i=1:sizev
```

```
    v30(i) = 0; % initial condition on cross(v) vel2
end

for i=1:sizew
    w30(i) = 0; % initial condition on star(w) vel2
end

for i=1:sizez
    z30(i) = 0; % initial condition on white dot(z) vel2
end

w30 = w30(:,2:end);
z30 = z30(2:end,:);

for i=1:sizeu
    exactp(i) = cos(2*pi*tend)*[sin(2*pi*X1(i)) + sin(2*pi*Y1(i))];
end

for i=1:sizeu
    exactvel1(i) = sin(2*pi*tend)*cos(2*pi*X1(i));
end

for i=1:sizeu
    exactvel2(i) = sin(2*pi*tend)*cos(2*pi*Y1(i));
end
%{
%initial condition plot

figure(1)          % pressure plot
surf(X1,Y1,u10)
view(2)
colorbar

figure(2)          % velocity 1 plot
surf(X1,Y1,u20)
view(2)
colorbar

figure(3)          % velocity 2 plot
surf(X1,Y1,u30)
view(2)
colorbar

%Exact solution plot
%}
figure(4)          % pressure plot
surf(X1,Y1,exactp)
%view(2)
colorbar

figure(5)          % velocity 1 plot
surf(X1,Y1,exactvel1)
%view(2)
colorbar

figure(6)          % velocity 2 plot
surf(X1,Y1,exactvel2)
%view(2)
colorbar

%===========================================================================

% finally stored only the grid points with IC not anything outside

%=================== length of each face (verified)========================
No_of_edges = 4*sizeu;                    % number of edges with repetition
length_of_face = Delta_x;                          % storing length of face
%===========================================================================

%========================= work for area (verified)========================
ar = Delta_x*Delta_y;                                   % area of each grid
area = ar*ones(sizeu,1);                  % storing all grid areas together
%===========================================================================

% allocating values on all four type of points for pressure
press0 = u10;
press1 = v10(1:end-1,1:end-1);
press2 = w10(1:end-1,:);
```

```
press3 = v10(1:end-1,2:end);
press4 = z10(:,2:end);
press5 = v10(2:end,2:end);
press6 = w10(2:end,:);
press7 = v10(2:end,1:end-1);
press8 = z10(:,1:end-1);

% calculating cell average for pressure
press9 = 9*( 4*press0 + ( press1 + press3 + press5 + press7 -4*press2 -4*press4
    -4*press6 -4*press8)/3 )/16 ;

% allocating values on all four type of points for velocity 1
vel10 = u20;
vel11 = v20(1:end-1,1:end-1);
vel12 = w20(1:end-1,:);
vel13 = v20(1:end-1,2:end);
vel14 = z20(:,2:end);
vel15 = v20(2:end,2:end);
vel16 = w20(2:end,:);
vel17 = v20(2:end,1:end-1);
vel18 = z20(:,1:end-1);

% calculating cell average for velocity 1
vel19 = 9*( 4*vel10 + ( vel11 + vel13 + vel15 + vel17 -4*vel12 -4*vel14 -4*vel16
    -4*vel18)/3 )/16 ;

% allocating values on all four type of points for velocity 2
vel20 = u30;
vel21 = v30(1:end-1,1:end-1);
vel22 = w30(1:end-1,:);
vel23 = v30(1:end-1,2:end);
vel24 = z30(:,2:end);
vel25 = v30(2:end,2:end);
vel26 = w30(2:end,:);
vel27 = v30(2:end,1:end-1);
vel28 = z30(:,1:end-1);

% calculating cell average for velocty 2
vel29 = 9*( 4*vel20 + ( vel21 + vel23 + vel25 + vel27 -4*vel22 -4*vel24 -4*vel26
    -4*vel28)/3 )/16 ;

% pre allocating the values at half time step for pressure
press1_1 = press1;
press2_1 = press2;
press3_1 = press3;
press4_1 = press4;
press5_1 = press5;
press6_1 = press6;
press7_1 = press7;
press8_1 = press8;

% pre allocating vales at full time step for pressure
press1_2 = press1;
press2_2 = press2;
press3_2 = press3;
press4_2 = press4;
press5_2 = press5;
press6_2 = press6;
press7_2 = press7;
press8_2 = press8;
press9_2 = press9;

% pre allocating the values at half time step for velocity 1
vel11_1 = vel11;
vel12_1 = vel12;
vel13_1 = vel13;
vel14_1 = vel14;
vel15_1 = vel15;
vel16_1 = vel16;
vel17_1 = vel17;
vel18_1 = vel18;

% pre allocating vales at full time step for velocity 1
vel11_2 = vel11;
vel12_2 = vel12;
vel13_2 = vel13;
vel14_2 = vel14;
vel15_2 = vel15;
```

```
vel16_2 = vel16;
vel17_2 = vel17;
vel18_2 = vel18;
vel19_2 = vel19;

% pre allocating the values at half time step for velocity 2
vel21_1 = vel21;
vel22_1 = vel22;
vel23_1 = vel23;
vel24_1 = vel24;
vel25_1 = vel25;
vel26_1 = vel26;
vel27_1 = vel27;
vel28_1 = vel28;

% pre allocating vales at full time step for velocity 2
vel21_2 = vel21;
vel22_2 = vel22;
vel23_2 = vel23;
vel24_2 = vel24;
vel25_2 = vel25;
vel26_2 = vel26;
vel27_2 = vel27;
vel28_2 = vel28;
vel29_2 = vel29;

% putting initial values in all 15 type of points %%%%%%%%%%%%%%%%%%%%%

t1_press_iv = press1(2:end,2:end);
t1_vel1_iv = vel11(2:end,2:end);
t1_vel2_iv = vel21(2:end,2:end);

t2_press_iv = press1(1,2:end);
t2_vel1_iv = vel11(1,2:end);
t2_vel2_iv = vel21(1,2:end);

t3_press_iv = press7(end,2:end);
t3_vel1_iv = vel17(end,2:end);
t3_vel2_iv = vel27(end,2:end);
```

```
t4_press_iv = press1(2:end,1);
t4_vel1_iv = vel11(2:end,1);
t4_vel2_iv = vel21(2:end,1);

t5_press_iv = press3(2:end,end);
t5_vel1_iv = vel13(2:end,end);
t5_vel2_iv = vel23(2:end,end);

t6_press_iv = press1(1,1);
t6_vel1_iv = vel11(1,1);
t6_vel2_iv = vel21(1,1);

t7_press_iv = press3(1,end);
t7_vel1_iv = vel13(1,end);
t7_vel2_iv = vel23(1,end);

t8_press_iv = press7(end,1);
t8_vel1_iv = vel17(end,1);
t8_vel2_iv = vel27(end,1);

t9_press_iv = press5(end,end);
t9_vel1_iv = vel15(end,end);
t9_vel2_iv = vel25(end,end);

t10_press_iv = press2(2:end,:);
t10_vel1_iv = vel12(2:end,:);
t10_vel2_iv = vel22(2:end,:);

t11_press_iv = press2(1,:);
t11_vel1_iv = vel12(1,:);
t11_vel2_iv = vel22(1,:);

t12_press_iv = press6(end,:);
t12_vel1_iv = vel16(end,:);
t12_vel2_iv = vel26(end,:);

t13_press_iv = press4(:,1:end-1);
t13_vel1_iv = vel14(:,1:end-1);
```

```
t13_vel2_iv = vel24(:,1:end-1);

t14_press_iv = press8(:,1);
t14_vel1_iv = vel18(:,1);
t14_vel2_iv = vel28(:,1);

t15_press_iv = press4(:,end);
t15_vel1_iv = vel14(:,end);
t15_vel2_iv = vel24(:,end);

% putting initial values in all 15 type of points ends %%%%%%%%%%%%%%

% Spherical means obtained from mathematica starts

% full time steps

I1f = -((r^2*(4*g+pi*r))/(8*g^3*pi));
I2f = (r*(-g^2+r^2))/(4*g^3);
I3f = I1f;
I4f = I2f;
I5f = I1f;
I6f = I2f;
I7f = I1f;
I8f = I2f;

I1xf = -((r*(9*g*pi + 4*(4+pi)* r))/(24*g^3*pi));
I2xf = 0;
I3xf = -I1xf;
I4xf = ((pi*r)/g - (4*pi*r^3)/(3*g^3))/(2*pi*r);
I5xf = -I1xf;
I6xf = 0;
I7xf = I1xf;
I8xf = (-3*g^2 + 4*r^2)/(6*g^3);

I1yf = -((r*(9*g*pi + 4*(4+pi)*r))/(24*g^3*pi));
I2yf = (-3*g^2 + 4*r^2)/(6*g^3);
I3yf = I1yf;
I4yf = 0;
I5yf = -I1yf;
```

```
I6yf = ((pi*r)/g - (4*pi*r^3)/(3*g^3))/(2*pi*r);
I7yf = -I1yf;
I8yf = 0;

rI1xf = r*(-((r*(9*g*pi + 4*(4+pi)* r))/(24*g^3*pi)));
rI2xf = 0;
rI3xf = -rI1xf;
rI4xf = r*((pi*r)/g - (4*pi*r^3)/(3*g^3))/(2*pi*r);
rI5xf = -rI1xf;
rI6xf = 0;
rI7xf = rI1xf;
rI8xf = r*(-3*g^2 + 4*r^2)/(6*g^3);

rI1yf = r*(-((r*(9*g*pi + 4*(4+pi)*r))/(24*g^3*pi)));
rI2yf = r*(-3*g^2 + 4*r^2)/(6*g^3);
rI3yf = rI1yf;
rI4yf = 0;
rI5yf = -rI1yf;
rI6yf = r*((pi*r)/g - (4*pi*r^3)/(3*g^3))/(2*pi*r);
rI7yf = -rI1yf;
rI8yf = 0;

rDI1f = r*((-r^2)/(8*g^3)-r*(4*g+pi*r)/(4*g^3*pi));
rDI2f = r*((r^2)/(2*g^3)+(r^2-g^2)/(4*g^3));
rDI3f = rDI1f;
rDI4f = rDI2f;
rDI5f = rDI1f;
rDI6f = rDI2f;
rDI7f = rDI1f;
rDI8f = rDI2f;

% half time steps

I1h = -(((r/2)^2*(4*g+pi*(r/2)))/(8*g^3*pi));
I2h = ((r/2)*(-g^2+(r/2)^2))/(4*g^3);
I3h = I1h;
I4h = I2h;
I5h = I1h;
I6h = I2h;
```

```
I7h = I1h;
I8h = I2h;

I1xh = -(((r/2)*(9*g*pi + 4*(4+pi)* (r/2)))/(24*g^3*pi));
I2xh = 0;
I3xh = -I1xh;
I4xh = ((pi*(r/2))/g - (4*pi*(r/2)^3)/(3*g^3))/(2*pi*(r/2));
I5xh = -I1xh;
I6xh = 0;
I7xh = I1xh;
I8xh = (-3*g^2 + 4*(r/2)^2)/(6*g^3);

I1yh = -(((r/2)*(9*g*pi + 4*(4+pi)*(r/2)))/(24*g^3*pi));
I2yh = (-3*g^2 + 4*(r/2)^2)/(6*g^3);
I3yh = I1yh;
I4yh = 0;
I5yh = -I1yh;
I6yh = ((pi*(r/2))/g - (4*pi*(r/2)^3)/(3*g^3))/(2*pi*(r/2));
I7yh = -I1yh;
I8yh = 0;

rI1xh = (r/2)*(-(((r/2)*(9*g*pi + 4*(4+pi)* (r/2)))/(24*g^3*pi)));
rI2xh = 0;
rI3xh = -rI1xh;
rI4xh = (r/2)*((pi*(r/2))/g - (4*pi*(r/2)^3)/(3*g^3))/(2*pi*(r/2));
rI5xh = -rI1xh;
rI6xh = 0;
rI7xh = rI1xh;
rI8xh = (r/2)*(-3*g^2 + 4*(r/2)^2)/(6*g^3);

rI1yh = (r/2)*(-(((r/2)*(9*g*pi + 4*(4+pi)*(r/2)))/(24*g^3*pi)));
rI2yh = (r/2)*(-3*g^2 + 4*(r/2)^2)/(6*g^3);
rI3yh = rI1yh;
rI4yh = 0;
rI5yh = -rI1yh;
rI6yh = (r/2)*((pi*(r/2))/g - (4*pi*(r/2)^3)/(3*g^3))/(2*pi*(r/2));
rI7yh = -rI1yh;
rI8yh = 0;
```

```
rDI1h = (r/2)*((-(r/2)^2)/(8*g^3)-(r/2)*(4*g+pi*(r/2))/(4*g^3*pi));
rDI2h = (r/2)*(((r/2)^2)/(2*g^3)+((r/2)^2-g^2)/(4*g^3));
rDI3h = rDI1h;
rDI4h = rDI2h;
rDI5h = rDI1h;
rDI6h = rDI2h;
rDI7h = rDI1h;
rDI8h = rDI2h;

% Spherical means obtained from mathematica ends

x11vf = I1f;
x12vf = I2f;
x13vf = I3f;
x14vf = I4f;
x15vf = I5f;
x16vf = I6f;
x17vf = I7f;
x18vf = I8f;

x11vh = I1h;
x12vh = I2h;
x13vh = I3h;
x14vh = I4h;
x15vh = I5h;
x16vh = I6h;
x17vh = I7h;
x18vh = I8h;


x21vf = rDI1f;
x22vf = rDI2f;
x23vf = rDI3f;
x24vf = rDI4f;
x25vf = rDI5f;
x26vf = rDI6f;
x27vf = rDI7f;
x28vf = rDI8f;
```

```
x21vh = rDI1h;
x22vh = rDI2h;
x23vh = rDI3h;
x24vh = rDI4h;
x25vh = rDI5h;
x26vh = rDI6h;
x27vh = rDI7h;
x28vh = rDI8h;


x31vf = rI1xf;
x32vf = rI2xf;
x33vf = rI3xf;
x34vf = rI4xf;
x35vf = rI5xf;
x36vf = rI6xf;
x37vf = rI7xf;
x38vf = rI8xf;

x31vh = rI1xh;
x32vh = rI2xh;
x33vh = rI3xh;
x34vh = rI4xh;
x35vh = rI5xh;
x36vh = rI6xh;
x37vh = rI7xh;
x38vh = rI8xh;


x41vf = rI1yf;
x42vf = rI2yf;
x43vf = rI3yf;
x44vf = rI4yf;
x45vf = rI5yf;
x46vf = rI6yf;
x47vf = rI7yf;
x48vf = rI8yf;

x41vh = rI1yh;
```

```
x42vh = rI2yh;
x43vh = rI3yh;
x44vh = rI4yh;
x45vh = rI5yh;
x46vh = rI6yh;
x47vh = rI7yh;
x48vh = rI8yh;


y11vf = x11vf;
y12vf = x12vf;
y13vf = x13vf;
y14vf = x14vf;
y15vf = x15vf;
y16vf = x16vf;
y17vf = x17vf;
y18vf = x18vf;

y11vh = x11vh;
y12vh = x12vh;
y13vh = x13vh;
y14vh = x14vh;
y15vh = x15vh;
y16vh = x16vh;
y17vh = x17vh;
y18vh = x18vh;


y21vf = x21vf;
y22vf = x22vf;
y23vf = x23vf;
y24vf = x24vf;
y25vf = x25vf;
y26vf = x26vf;
y27vf = x27vf;
y28vf = x28vf;

y21vh = x21vh;
y22vh = x22vh;
```

```
y23vh = x23vh;
y24vh = x24vh;
y25vh = x25vh;
y26vh = x26vh;
y27vh = x27vh;
y28vh = x28vh;


y31vf = x31vf;
y32vf = x32vf;
y33vf = x33vf;
y34vf = x34vf;
y35vf = x35vf;
y36vf = x36vf;
y37vf = x37vf;
y38vf = x38vf;

y31vh = x31vh;
y32vh = x32vh;
y33vh = x33vh;
y34vh = x34vh;
y35vh = x35vh;
y36vh = x36vh;
y37vh = x37vh;
y38vh = x38vh;


z11vf = x11vf;
z12vf = x12vf;
z13vf = x13vf;
z14vf = x14vf;
z15vf = x15vf;
z16vf = x16vf;
z17vf = x17vf;
z18vf = x18vf;

z11vh = x11vh;
z12vh = x12vh;
z13vh = x13vh;
```

```
z14vh = x14vh;
z15vh = x15vh;
z16vh = x16vh;
z17vh = x17vh;
z18vh = x18vh;


z21vf = x21vf;
z22vf = x22vf;
z23vf = x23vf;
z24vf = x24vf;
z25vf = x25vf;
z26vf = x26vf;
z27vf = x27vf;
z28vf = x28vf;

z21vh = x21vh;
z22vh = x22vh;
z23vh = x23vh;
z24vh = x24vh;
z25vh = x25vh;
z26vh = x26vh;
z27vh = x27vh;
z28vh = x28vh;


z31vf = x31vf;
z32vf = x32vf;
z33vf = x33vf;
z34vf = x34vf;
z35vf = x35vf;
z36vf = x36vf;
z37vf = x37vf;
z38vf = x38vf;

z31vh = x31vh;
z32vh = x32vh;
z33vh = x33vh;
z34vh = x34vh;
```

```
z35vh = x35vh;
z36vh = x36vh;
z37vh = x37vh;
z38vh = x38vh;

% preparing spherical means for all 15 type of points for all three
% pressure, vel1 and vel2 at half and full time steps

t1phx1 = x11vh + x13vh + x15vh + x17vh;
t1phx2 = x21vh + x23vh + x25vh + x27vh;
t1phx3 = x31vh + x33vh + x35vh + x37vh;
t1phx4 = x41vh + x43vh + x45vh + x47vh;

t1pfx1 = x11vf + x13vf + x15vf + x17vf;
t1pfx2 = x21vf + x23vf + x25vf + x27vf;
t1pfx3 = x31vf + x33vf + x35vf + x37vf;
t1pfx4 = x41vf + x43vf + x45vf + x47vf;

t1uhy1 = y11vh + y13vh + y15vh + y17vh;
t1uhy2 = y21vh + y23vh + y25vh + y27vh;
t1uhy3 = y31vh + y33vh + y35vh + y37vh;

t1ufy1 = y11vf + y13vf + y15vf + y17vf;
t1ufy2 = y21vf + y23vf + y25vf + y27vf;
t1ufy3 = y31vf + y33vf + y35vf + y37vf;

t1vhz1 = z11vh + z13vh + z15vh + z17vh;
t1vhz2 = z21vh + z23vh + z25vh + z27vh;
t1vhz3 = z31vh + z33vh + z35vh + z37vh;

t1vfz1 = z11vf + z13vf + z15vf + z17vf;
t1vfz2 = z21vf + z23vf + z25vf + z27vf;
t1vfz3 = z31vf + z33vf + z35vf + z37vf;


t2phx1 = x11vh + x13vh;
t2phx2 = x21vh + x23vh;
t2phx3 = x31vh + x33vh;
t2phx4 = x41vh + x43vh;
```

```
t2pfx1 = x11vf + x13vf;
t2pfx2 = x21vf + x23vf;
t2pfx3 = x31vf + x33vf;
t2pfx4 = x41vf + x43vf;

t2uhy1 = y11vh + y13vh;
t2uhy2 = y21vh + y23vh;
t2uhy3 = y31vh + y33vh;

t2ufy1 = y11vf + y13vf;
t2ufy2 = y21vf + y23vf;
t2ufy3 = y31vf + y33vf;

t2vhz1 = z11vh + z13vh;
t2vhz2 = z21vh + z23vh;
t2vhz3 = z31vh + z33vh;

t2vfz1 = z11vf + z13vf;
t2vfz2 = z21vf + z23vf;
t2vfz3 = z31vf + z33vf;



t3phx1 = x15vh + x17vh;
t3phx2 = x25vh + x27vh;
t3phx3 = x35vh + x37vh;
t3phx4 = x45vh + x47vh;

t3pfx1 = x15vf + x17vf;
t3pfx2 = x25vf + x27vf;
t3pfx3 = x35vf + x37vf;
t3pfx4 = x45vf + x47vf;

t3uhy1 = y15vh + y17vh;
t3uhy2 = y25vh + y27vh;
t3uhy3 = y35vh + y37vh;

t3ufy1 = y15vf + y17vf;
```

```
t3ufy2 = y25vf + y27vf;
t3ufy3 = y35vf + y37vf;

t3vhz1 = z15vh + z17vh;
t3vhz2 = z25vh + z27vh;
t3vhz3 = z35vh + z37vh;

t3vfz1 = z15vf + z17vf;
t3vfz2 = z25vf + z27vf;
t3vfz3 = z35vf + z37vf;


t4phx1 = x11vh + x17vh;
t4phx2 = x21vh + x27vh;
t4phx3 = x31vh + x37vh;
t4phx4 = x41vh + x47vh;

t4pfx1 = x11vf + x17vf;
t4pfx2 = x21vf + x27vf;
t4pfx3 = x31vf + x37vf;
t4pfx4 = x41vf + x47vf;

t4uhy1 = y11vh + y17vh;
t4uhy2 = y21vh + y27vh;
t4uhy3 = y31vh + y37vh;

t4ufy1 = y11vf + y17vf;
t4ufy2 = y21vf + y27vf;
t4ufy3 = y31vf + y37vf;

t4vhz1 = z11vh + z17vh;
t4vhz2 = z21vh + z27vh;
t4vhz3 = z31vh + z37vh;

t4vfz1 = z11vf + z17vf;
t4vfz2 = z21vf + z27vf;
t4vfz3 = z31vf + z37vf;
```

```
t5phx1 = x15vh + x13vh;
t5phx2 = x25vh + x23vh;
t5phx3 = x35vh + x33vh;
t5phx4 = x45vh + x43vh;

t5pfx1 = x15vf + x13vf;
t5pfx2 = x25vf + x23vf;
t5pfx3 = x35vf + x33vf;
t5pfx4 = x45vf + x43vf;

t5uhy1 = y15vh + y13vh;
t5uhy2 = y25vh + y23vh;
t5uhy3 = y35vh + y33vh;

t5ufy1 = y15vf + y13vf;
t5ufy2 = y25vf + y23vf;
t5ufy3 = y35vf + y33vf;

t5vhz1 = z15vh + z13vh;
t5vhz2 = z25vh + z23vh;
t5vhz3 = z35vh + z33vh;

t5vfz1 = z15vf + z13vf;
t5vfz2 = z25vf + z23vf;
t5vfz3 = z35vf + z33vf;



t6phx1 = x11vh;
t6phx2 = x21vh;
t6phx3 = x31vh;
t6phx4 = x41vh;

t6pfx1 = x11vf;
t6pfx2 = x21vf;
t6pfx3 = x31vf;
t6pfx4 = x41vf;
```

```
t6uhy1 = y11vh;
t6uhy2 = y21vh;
t6uhy3 = y31vh;

t6ufy1 = y11vf;
t6ufy2 = y21vf;
t6ufy3 = y31vf;

t6vhz1 = z11vh;
t6vhz2 = z21vh;
t6vhz3 = z31vh;

t6vfz1 = z11vf;
t6vfz2 = z21vf;
t6vfz3 = z31vf;



t7phx1 = x13vh;
t7phx2 = x23vh;
t7phx3 = x33vh;
t7phx4 = x43vh;

t7pfx1 = x13vf;
t7pfx2 = x23vf;
t7pfx3 = x33vf;
t7pfx4 = x43vf;

t7uhy1 = y13vh;
t7uhy2 = y23vh;
t7uhy3 = y33vh;

t7ufy1 = y13vf;
t7ufy2 = y23vf;
t7ufy3 = y33vf;

t7vhz1 = z13vh;
t7vhz2 = z23vh;
t7vhz3 = z33vh;
```

```
t7vfz1 = z13vf;
t7vfz2 = z23vf;
t7vfz3 = z33vf;



t8phx1 = x17vh;
t8phx2 = x27vh;
t8phx3 = x37vh;
t8phx4 = x47vh;

t8pfx1 = x17vf;
t8pfx2 = x27vf;
t8pfx3 = x37vf;
t8pfx4 = x47vf;

t8uhy1 = y17vh;
t8uhy2 = y27vh;
t8uhy3 = y37vh;

t8ufy1 = y17vf;
t8ufy2 = y27vf;
t8ufy3 = y37vf;

t8vhz1 = z17vh;
t8vhz2 = z27vh;
t8vhz3 = z37vh;

t8vfz1 = z17vf;
t8vfz2 = z27vf;
t8vfz3 = z37vf;



t9phx1 = x15vh;
t9phx2 = x25vh;
t9phx3 = x35vh;
t9phx4 = x45vh;

t9pfx1 = x15vf;
```

```
t9pfx2 = x25vf;
t9pfx3 = x35vf;
t9pfx4 = x45vf;

t9uhy1 = y15vh;
t9uhy2 = y25vh;
t9uhy3 = y35vh;

t9ufy1 = y15vf;
t9ufy2 = y25vf;
t9ufy3 = y35vf;

t9vhz1 = z15vh;
t9vhz2 = z25vh;
t9vhz3 = z35vh;

t9vfz1 = z15vf;
t9vfz2 = z25vf;
t9vfz3 = z35vf;



t10phx1 = x12vh + x16vh;
t10phx2 = x22vh + x26vh;
t10phx3 = x32vh + x36vh;
t10phx4 = x42vh + x46vh;

t10pfx1 = x12vf + x16vf;
t10pfx2 = x22vf + x26vf;
t10pfx3 = x32vf + x36vf;
t10pfx4 = x42vf + x46vf;

t10uhy1 = y12vh + y16vh;
t10uhy2 = y22vh + y26vh;
t10uhy3 = y32vh + y36vh;

t10ufy1 = y12vf + y16vf;
t10ufy2 = y22vf + y26vf;
t10ufy3 = y32vf + y36vf;
```

```
t10vhz1 = z12vh + z16vh;
t10vhz2 = z22vh + z26vh;
t10vhz3 = z32vh + z36vh;

t10vfz1 = z12vf + z16vf;
t10vfz2 = z22vf + z26vf;
t10vfz3 = z32vf + z36vf;


t11phx1 = x12vh;
t11phx2 = x22vh;
t11phx3 = x32vh;
t11phx4 = x42vh;

t11pfx1 = x12vf;
t11pfx2 = x22vf;
t11pfx3 = x32vf;
t11pfx4 = x42vf;

t11uhy1 = y12vh;
t11uhy2 = y22vh;
t11uhy3 = y32vh;

t11ufy1 = y12vf;
t11ufy2 = y22vf;
t11ufy3 = y32vf;


t11vhz1 = z12vh;
t11vhz2 = z22vh;
t11vhz3 = z32vh;

t11vfz1 = z12vf;
t11vfz2 = z22vf;
t11vfz3 = z32vf;
```

```
t12phx1 = x16vh;
t12phx2 = x26vh;
t12phx3 = x36vh;
t12phx4 = x46vh;

t12pfx1 = x16vf;
t12pfx2 = x26vf;
t12pfx3 = x36vf;
t12pfx4 = x46vf;

t12uhy1 = y16vh;
t12uhy2 = y26vh;
t12uhy3 = y36vh;

t12ufy1 = y16vf;
t12ufy2 = y26vf;
t12ufy3 = y36vf;



t12vhz1 = z16vh;
t12vhz2 = z26vh;
t12vhz3 = z36vh;

t12vfz1 = z16vf;
t12vfz2 = z26vf;
t12vfz3 = z36vf;



t13phx1 = x14vh + x18vh;
t13phx2 = x24vh + x28vh;
t13phx3 = x34vh + x38vh;
t13phx4 = x44vh + x48vh;

t13pfx1 = x14vf + x18vf;
t13pfx2 = x24vf + x28vf;
t13pfx3 = x34vf + x38vf;
t13pfx4 = x44vf + x48vf;

t13uhy1 = y14vh + y18vh;
```

```
t13uhy2 = y24vh + y28vh;
t13uhy3 = y34vh + y38vh;

t13ufy1 = y14vf + y18vf;
t13ufy2 = y24vf + y28vf;
t13ufy3 = y34vf + y38vf;

t13vhz1 = z14vh + z18vh;
t13vhz2 = z24vh + z28vh;
t13vhz3 = z34vh + z38vh;

t13vfz1 = z14vf + z18vf;
t13vfz2 = z24vf + z28vf;
t13vfz3 = z34vf + z38vf;


t14phx1 = x18vh;
t14phx2 = x28vh;
t14phx3 = x38vh;
t14phx4 = x48vh;

t14pfx1 = x18vf;
t14pfx2 = x28vf;
t14pfx3 = x38vf;
t14pfx4 = x48vf;

t14uhy1 = y18vh;
t14uhy2 = y28vh;
t14uhy3 = y38vh;

t14ufy1 = y18vf;
t14ufy2 = y28vf;
t14ufy3 = y38vf;

t14vhz1 = z18vh;
t14vhz2 = z28vh;
t14vhz3 = z38vh;

t14vfz1 = z18vf;
```

```
t14vfz2 = z28vf;
t14vfz3 = z38vf;



t15phx1 = x14vh;
t15phx2 = x24vh;
t15phx3 = x34vh;
t15phx4 = x44vh;

t15pfx1 = x14vf;
t15pfx2 = x24vf;
t15pfx3 = x34vf;
t15pfx4 = x44vf;

t15uhy1 = y14vh;
t15uhy2 = y24vh;
t15uhy3 = y34vh;

t15ufy1 = y14vf;
t15ufy2 = y24vf;
t15ufy3 = y34vf;

t15vhz1 = z14vh;
t15vhz2 = z24vh;
t15vhz3 = z34vh;

t15vfz1 = z14vf;
t15vfz2 = z24vf;
t15vfz3 = z34vf;

% allocating spherical means on 15 type of points ends

kk = dt;
ll = dt/2;
% loop starts
for ii = t:dt:tend

    t1_press_h = (t1_press_iv * t1phx1) + (t1_press_iv * t1phx2) - (t1_vel1_iv *
        t1phx3) - (t1_vel2_iv * t1phx4) ;
    t1_vel1_h = (t1_vel1_iv * t1uhy1) + (t1_vel1_iv * t1uhy2) - (t1_press_iv *
        t1uhy3) ;
    t1_vel2_h = (t1_vel2_iv * t1vhz1) + (t1_vel2_iv * t1vhz2) - (t1_press_iv *
        t1vhz3) ;

    t1_press_f = (t1_press_iv * t1pfx1) + (t1_press_iv * t1pfx2) - (t1_vel1_iv *
        t1pfx3) - (t1_vel2_iv * t1pfx4) ;
    t1_vel1_f = (t1_vel1_iv * t1ufy1) + (t1_vel1_iv * t1ufy2) - (t1_press_iv *
        t1ufy3) ;
    t1_vel2_f = (t1_vel2_iv * t1vfz1) + (t1_vel2_iv * t1vfz2) - (t1_press_iv *
        t1vfz3) ;



    t2_press_h = (t2_press_iv * t2phx1) + (t2_press_iv * t2phx2) - (t2_vel1_iv *
        t2phx3) - (t2_vel2_iv * t2phx4) ;
    t2_vel1_h = (t2_vel1_iv * t2uhy1) + (t2_vel1_iv * t2uhy2) - (t2_press_iv *
        t2uhy3) ;
    t2_vel2_h = (t2_vel2_iv * t2vhz1) + (t2_vel2_iv * t2vhz2) - (t2_press_iv *
        t2vhz3) ;

    t2_press_f = (t2_press_iv * t2pfx1) + (t2_press_iv * t2pfx2) - (t2_vel1_iv *
        t2pfx3) - (t2_vel2_iv * t2pfx4) ;
    t2_vel1_f = (t2_vel1_iv * t2ufy1) + (t2_vel1_iv * t2ufy2) - (t2_press_iv *
        t2ufy3) ;
    t2_vel2_f = (t2_vel2_iv * t2vfz1) + (t2_vel2_iv * t2vfz2) - (t2_press_iv *
        t2vfz3) ;




    t3_press_h = (t3_press_iv * t3phx1) + (t3_press_iv * t3phx2) - (t3_vel1_iv *
        t3phx3) - (t3_vel2_iv * t3phx4) ;
    t3_vel1_h = (t3_vel1_iv * t3uhy1) + (t3_vel1_iv * t3uhy2) - (t3_press_iv *
        t3uhy3) ;
    t3_vel2_h = (t3_vel2_iv * t3vhz1) + (t3_vel2_iv * t3vhz2) - (t3_press_iv *
        t3vhz3) ;

    t3_press_f = (t3_press_iv * t3pfx1) + (t3_press_iv * t3pfx2) - (t3_vel1_iv *
        t3pfx3) - (t3_vel2_iv * t3pfx4) ;
    t3_vel1_f = (t3_vel1_iv * t3ufy1) + (t3_vel1_iv * t3ufy2) - (t3_press_iv *
        t3ufy3) ;
    t3_vel2_f = (t3_vel2_iv * t3vfz1) + (t3_vel2_iv * t3vfz2) - (t3_press_iv *
        t3vfz3) ;



    t4_press_h = (t4_press_iv * t4phx1) + (t4_press_iv * t4phx2) - (t4_vel1_iv *
        t4phx3) - (t4_vel2_iv * t4phx4) ;
    t4_vel1_h = (t4_vel1_iv * t4uhy1) + (t4_vel1_iv * t4uhy2) - (t4_press_iv *
        t4uhy3) ;
    t4_vel2_h = (t4_vel2_iv * t4vhz1) + (t4_vel2_iv * t4vhz2) - (t4_press_iv *
        t4vhz3) ;

    t4_press_f = (t4_press_iv * t4pfx1) + (t4_press_iv * t4pfx2) - (t4_vel1_iv *
        t4pfx3) - (t4_vel2_iv * t4pfx4) ;
    t4_vel1_f = (t4_vel1_iv * t4ufy1) + (t4_vel1_iv * t4ufy2) - (t4_press_iv *
        t4ufy3) ;
    t4_vel2_f = (t4_vel2_iv * t4vfz1) + (t4_vel2_iv * t4vfz2) - (t4_press_iv *
        t4vfz3) ;



    t5_press_h = (t5_press_iv * t5phx1) + (t5_press_iv * t5phx2) - (t5_vel1_iv *
        t5phx3) - (t5_vel2_iv * t5phx4) ;
    t5_vel1_h = (t5_vel1_iv * t5uhy1) + (t5_vel1_iv * t5uhy2) - (t5_press_iv *
        t5uhy3) ;
    t5_vel2_h = (t5_vel2_iv * t5vhz1) + (t5_vel2_iv * t5vhz2) - (t5_press_iv *
        t5vhz3) ;

    t5_press_f = (t5_press_iv * t5pfx1) + (t5_press_iv * t5pfx2) - (t5_vel1_iv *
        t5pfx3) - (t5_vel2_iv * t5pfx4) ;
    t5_vel1_f = (t5_vel1_iv * t5ufy1) + (t5_vel1_iv * t5ufy2) - (t5_press_iv *
        t5ufy3) ;
    t5_vel2_f = (t5_vel2_iv * t5vfz1) + (t5_vel2_iv * t5vfz2) - (t5_press_iv *
        t5vfz3) ;

    t6_press_h = (t6_press_iv * t6phx1) + (t6_press_iv * t6phx2) - (t6_vel1_iv *
        t6phx3) - (t6_vel2_iv * t6phx4) ;
    t6_vel1_h = (t6_vel1_iv * t6uhy1) + (t6_vel1_iv * t6uhy2) - (t6_press_iv *
        t6uhy3) ;
    t6_vel2_h = (t6_vel2_iv * t6vhz1) + (t6_vel2_iv * t6vhz2) - (t6_press_iv *
        t6vhz3) ;

    t6_press_f = (t6_press_iv * t6pfx1) + (t6_press_iv * t6pfx2) - (t6_vel1_iv *
        t6pfx3) - (t6_vel2_iv * t6pfx4) ;
    t6_vel1_f = (t6_vel1_iv * t6ufy1) + (t6_vel1_iv * t6ufy2) - (t6_press_iv *
        t6ufy3) ;
    t6_vel2_f = (t6_vel2_iv * t6vfz1) + (t6_vel2_iv * t6vfz2) - (t6_press_iv *
        t6vfz3) ;


    t7_press_h = (t7_press_iv * t7phx1) + (t7_press_iv * t7phx2) - (t7_vel1_iv *
        t7phx3) - (t7_vel2_iv * t7phx4) ;
    t7_vel1_h = (t7_vel1_iv * t7uhy1) + (t7_vel1_iv * t7uhy2) - (t7_press_iv *
        t7uhy3) ;
    t7_vel2_h = (t7_vel2_iv * t7vhz1) + (t7_vel2_iv * t7vhz2) - (t7_press_iv *
        t7vhz3) ;

    t7_press_f = (t7_press_iv * t7pfx1) + (t7_press_iv * t7pfx2) - (t7_vel1_iv *
        t7pfx3) - (t7_vel2_iv * t7pfx4) ;
    t7_vel1_f = (t7_vel1_iv * t7ufy1) + (t7_vel1_iv * t7ufy2) - (t7_press_iv *
        t7ufy3) ;
    t7_vel2_f = (t7_vel2_iv * t7vfz1) + (t7_vel2_iv * t7vfz2) - (t7_press_iv *
        t7vfz3) ;


    t8_press_h = (t8_press_iv * t8phx1) + (t8_press_iv * t8phx2) - (t8_vel1_iv *
        t8phx3) - (t8_vel2_iv * t8phx4) ;
    t8_vel1_h = (t8_vel1_iv * t8uhy1) + (t8_vel1_iv * t8uhy2) - (t8_press_iv *
        t8uhy3) ;
    t8_vel2_h = (t8_vel2_iv * t8vhz1) + (t8_vel2_iv * t8vhz2) - (t8_press_iv *
        t8vhz3) ;

    t8_press_f = (t8_press_iv * t8pfx1) + (t8_press_iv * t8pfx2) - (t8_vel1_iv *
        t8pfx3) - (t8_vel2_iv * t8pfx4) ;
    t8_vel1_f = (t8_vel1_iv * t8ufy1) + (t8_vel1_iv * t8ufy2) - (t8_press_iv *
        t8ufy3) ;
    t8_vel2_f = (t8_vel2_iv * t8vfz1) + (t8_vel2_iv * t8vfz2) - (t8_press_iv *
        t8vfz3) ;
```

```
t9_press_h = (t9_press_iv * t9phx1) + (t9_press_iv * t9phx2) - (t9_vel1_iv *
    t9phx3) - (t9_vel2_iv * t9phx4) ;
t9_vel1_h = (t9_vel1_iv * t9uhy1) + (t9_vel1_iv * t9uhy2) - (t9_press_iv *
    t9uhy3) ;
t9_vel2_h = (t9_vel2_iv * t9vhz1) + (t9_vel2_iv * t9vhz2) - (t9_press_iv *
    t9vhz3) ;

t9_press_f = (t9_press_iv * t9pfx1) + (t9_press_iv * t9pfx2) - (t9_vel1_iv *
    t9pfx3) - (t9_vel2_iv * t9pfx4) ;
t9_vel1_f = (t9_vel1_iv * t9ufy1) + (t9_vel1_iv * t9ufy2) - (t9_press_iv *
    t9ufy3) ;
t9_vel2_f = (t9_vel2_iv * t9vfz1) + (t9_vel2_iv * t9vfz2) - (t9_press_iv *
    t9vfz3) ;



t10_press_h = (t10_press_iv * t10phx1) + (t10_press_iv * t10phx2) - (
    t10_vel1_iv * t10phx3) - (t10_vel2_iv * t10phx4) ;
t10_vel1_h = (t10_vel1_iv * t10uhy1) + (t10_vel1_iv * t10uhy2) - (
    t10_press_iv * t10uhy3) ;
t10_vel2_h = (t10_vel2_iv * t10vhz1) + (t10_vel2_iv * t10vhz2) - (
    t10_press_iv * t10vhz3) ;

t10_press_f = (t10_press_iv * t10pfx1) + (t10_press_iv * t10pfx2) - (
    t10_vel1_iv * t10pfx3) - (t10_vel2_iv * t10pfx4) ;
t10_vel1_f = (t10_vel1_iv * t10ufy1) + (t10_vel1_iv * t10ufy2) - (
    t10_press_iv * t10ufy3) ;
t10_vel2_f = (t10_vel2_iv * t10vfz1) + (t10_vel2_iv * t10vfz2) - (
    t10_press_iv * t10vfz3) ;



t11_press_h = (t11_press_iv * t11phx1) + (t11_press_iv * t11phx2) - (
    t11_vel1_iv * t11phx3) - (t11_vel2_iv * t11phx4) ;
t11_vel1_h = (t11_vel1_iv * t11uhy1) + (t11_vel1_iv * t11uhy2) - (
    t11_press_iv * t11uhy3) ;
t11_vel2_h = (t11_vel2_iv * t11vhz1) + (t11_vel2_iv * t11vhz2) - (
    t11_press_iv * t11vhz3) ;

t11_press_f = (t11_press_iv * t11pfx1) + (t11_press_iv * t11pfx2) - (
    t11_vel1_iv * t11pfx3) - (t11_vel2_iv * t11pfx4) ;
t11_vel1_f = (t11_vel1_iv * t11ufy1) + (t11_vel1_iv * t11ufy2) - (
    t11_press_iv * t11ufy3) ;
t11_vel2_f = (t11_vel2_iv * t11vfz1) + (t11_vel2_iv * t11vfz2) - (
    t11_press_iv * t11vfz3) ;



t12_press_h = (t12_press_iv * t12phx1) + (t12_press_iv * t12phx2) - (
    t12_vel1_iv * t12phx3) - (t12_vel2_iv * t12phx4) ;
t12_vel1_h = (t12_vel1_iv * t12uhy1) + (t12_vel1_iv * t12uhy2) - (
    t12_press_iv * t12uhy3) ;
t12_vel2_h = (t12_vel2_iv * t12vhz1) + (t12_vel2_iv * t12vhz2) - (
    t12_press_iv * t12vhz3) ;

t12_press_f = (t12_press_iv * t12pfx1) + (t12_press_iv * t12pfx2) - (
    t12_vel1_iv * t12pfx3) - (t12_vel2_iv * t12pfx4) ;
t12_vel1_f = (t12_vel1_iv * t12ufy1) + (t12_vel1_iv * t12ufy2) - (
    t12_press_iv * t12ufy3) ;
t12_vel2_f = (t12_vel2_iv * t12vfz1) + (t12_vel2_iv * t12vfz2) - (
    t12_press_iv * t12vfz3) ;




t13_press_h = (t13_press_iv * t13phx1) + (t13_press_iv * t13phx2) - (
    t13_vel1_iv * t13phx3) - (t13_vel2_iv * t13phx4) ;
t13_vel1_h = (t13_vel1_iv * t13uhy1) + (t13_vel1_iv * t13uhy2) - (
    t13_press_iv * t13uhy3) ;
t13_vel2_h = (t13_vel2_iv * t13vhz1) + (t13_vel2_iv * t13vhz2) - (
    t13_press_iv * t13vhz3) ;

t13_press_f = (t13_press_iv * t13pfx1) + (t13_press_iv * t13pfx2) - (
    t13_vel1_iv * t13pfx3) - (t13_vel2_iv * t13pfx4) ;
t13_vel1_f = (t13_vel1_iv * t13ufy1) + (t13_vel1_iv * t13ufy2) - (
    t13_press_iv * t13ufy3) ;
t13_vel2_f = (t13_vel2_iv * t13vfz1) + (t13_vel2_iv * t13vfz2) - (
    t13_press_iv * t13vfz3) ;
```

```
t14_press_h = (t14_press_iv * t14phx1) + (t14_press_iv * t14phx2) - (
    t14_vel1_iv * t14phx3) - (t14_vel2_iv * t14phx4) ;
t14_vel1_h = (t14_vel1_iv * t14uhy1) + (t14_vel1_iv * t14uhy2) - (
    t14_press_iv * t14uhy3) ;
t14_vel2_h = (t14_vel2_iv * t14vhz1) + (t14_vel2_iv * t14vhz2) - (
    t14_press_iv * t14vhz3) ;

t14_press_f = (t14_press_iv * t14pfx1) + (t14_press_iv * t14pfx2) - (
    t14_vel1_iv * t14pfx3) - (t14_vel2_iv * t14pfx4) ;
t14_vel1_f = (t14_vel1_iv * t14ufy1) + (t14_vel1_iv * t14ufy2) - (
    t14_press_iv * t14ufy3) ;
t14_vel2_f = (t14_vel2_iv * t14vfz1) + (t14_vel2_iv * t14vfz2) - (
    t14_press_iv * t14vfz3) ;



t15_press_h = (t15_press_iv * t15phx1) + (t15_press_iv * t15phx2) - (
    t15_vel1_iv * t15phx3) - (t15_vel2_iv * t15phx4) ;
t15_vel1_h = (t15_vel1_iv * t15uhy1) + (t15_vel1_iv * t15uhy2) - (
    t15_press_iv * t15uhy3) ;
t15_vel2_h = (t15_vel2_iv * t15vhz1) + (t15_vel2_iv * t15vhz2) - (
    t15_press_iv * t15vhz3) ;

t15_press_f = (t15_press_iv * t15pfx1) + (t15_press_iv * t15pfx2) - (
    t15_vel1_iv * t15pfx3) - (t15_vel2_iv * t15pfx4) ;
t15_vel1_f = (t15_vel1_iv * t15ufy1) + (t15_vel1_iv * t15ufy2) - (
    t15_press_iv * t15ufy3) ;
t15_vel2_f = (t15_vel2_iv * t15vfz1) + (t15_vel2_iv * t15vfz2) - (
    t15_press_iv * t15vfz3) ;



% calculating for 8 points on a cell

% for pressure

% point 1
press1_1(2:end,2:end) = t1_press_h;
press1_1(1,2:end) = t2_press_h;
```

```
    press1_1(2:end,1) = t4_press_h;
    press1_1(1,1) = t6_press_h;

    press1_2(2:end,2:end) = t1_press_f;
    press1_2(1,2:end) = t2_press_f;
    press1_2(2:end,1) = t4_press_f;
    press1_2(1,1) = t6_press_f;

    % point 2

    press2_1(2:end,:) = t10_press_h;
    press2_1(1,:) = t11_press_h;

    press2_2(2:end,:) = t10_press_f;
    press2_2(1,:) = t11_press_f;

    % point 3

    press3_1(2:end,1:end-1) = t1_press_h;
    press3_1(1,1:end-1) = t2_press_h;
    press3_1(2:end,end) = t5_press_h;
    press3_1(1,end) = t7_press_h;

    press3_2(2:end,1:end-1) = t1_press_f;
    press3_2(1,1:end-1) = t2_press_f;
    press3_2(2:end,end) = t5_press_f;
    press3_2(1,end) = t7_press_f;

    % point 4

    press4_1(:,1:end-1) = t13_press_h;
    press4_1(:,end) = t15_press_h;

    press4_2(:,1:end-1) = t13_press_f;
    press4_2(:,end) = t15_press_f;

    % point 5

    press5_1(1:end-1,1:end-1) = t1_press_h;
```

```
press5_1(end,1:end-1) = t3_press_h;
press5_1(1:end-1,end) = t5_press_h;
press5_1(end,end) = t9_press_h;

press5_2(1:end-1,1:end-1) = t1_press_f;
press5_2(end,1:end-1) = t3_press_f;
press5_2(1:end-1,end) = t5_press_f;
press5_2(end,end) = t9_press_f;

% point 6

press6_1(1:end-1,:) = t10_press_h;
press6_1(end,:) = t12_press_h;

press6_2(1:end-1,:) = t10_press_f;
press6_2(end,:) = t12_press_f;

% point 7

press7_1(1:end-1,2:end) = t1_press_h;
press7_1(1:end-1,1) = t4_press_h;
press7_1(end,2:end) = t3_press_h;
press7_1(end,1) = t8_press_h;

press7_2(1:end-1,2:end) = t1_press_f;
press7_2(1:end-1,1) = t4_press_f;
press7_2(end,2:end) = t3_press_f;
press7_2(end,1) = t8_press_f;

% point 8

press8_1(:,2:end) = t13_press_h;
press8_1(:,1) = t14_press_h;

press8_2(:,2:end) = t13_press_f;
press8_2(:,1) = t14_press_f;
```

```
% for velocity 1

% point 1
vel11_1(2:end,2:end) = t1_vel1_h;
vel11_1(1,2:end) = t2_vel1_h;
vel11_1(2:end,1) = t4_vel1_h;
vel11_1(1,1) = t6_vel1_h;

vel11_2(2:end,2:end) = t1_vel1_f;
vel11_2(1,2:end) = t2_vel1_f;
vel11_2(2:end,1) = t4_vel1_f;
vel11_2(1,1) = t6_vel1_f;

% point 2

vel12_1(2:end,:) = t10_vel1_h;
vel12_1(1,:) = t11_vel1_h;

vel12_2(2:end,:) = t10_vel1_f;
vel12_2(1,:) = t11_vel1_f;

% point 3

vel13_1(2:end,1:end-1) = t1_vel1_h;
vel13_1(1,1:end-1) = t2_vel1_h;
vel13_1(2:end,end) = t5_vel1_h;
vel13_1(1,end) = t7_vel1_h;

vel13_2(2:end,1:end-1) = t1_vel1_f;
vel13_2(1,1:end-1) = t2_vel1_f;
vel13_2(2:end,end) = t5_vel1_f;
vel13_2(1,end) = t7_vel1_f;

% point 4

vel14_1(:,1:end-1) = t13_vel1_h;
vel14_1(:,end) = t15_vel1_h;

vel14_2(:,1:end-1) = t13_vel1_f;
```

```
    vel14_2(:,end) = t15_vel1_f;

    % point 5

    vel15_1(1:end-1,1:end-1) = t1_vel1_h;
    vel15_1(end,1:end-1) = t3_vel1_h;
    vel15_1(1:end-1,end) = t5_vel1_h;
    vel15_1(end,end) = t9_vel1_h;

    vel15_2(1:end-1,1:end-1) = t1_vel1_f;
    vel15_2(end,1:end-1) = t3_vel1_f;
    vel15_2(1:end-1,end) = t5_vel1_f;
    vel15_2(end,end) = t9_vel1_f;

    % point 6

    vel16_1(1:end-1,:) = t10_vel1_h;
    vel16_1(end,:) = t12_vel1_h;

    vel16_2(1:end-1,:) = t10_vel1_f;
    vel16_2(end,:) = t12_vel1_f;

    % point 7

    vel17_1(1:end-1,2:end) = t1_vel1_h;
    vel17_1(1:end-1,1) = t4_vel1_h;
    vel17_1(end,2:end) = t3_vel1_h;
    vel17_1(end,1) = t8_vel1_h;

    vel17_2(1:end-1,2:end) = t1_vel1_f;
    vel17_2(1:end-1,1) = t4_vel1_f;
    vel17_2(end,2:end) = t3_vel1_f;
    vel17_2(end,1) = t8_vel1_f;

    % point 8

    vel18_1(:,2:end) = t13_vel1_h;
    vel18_1(:,1) = t14_vel1_h;
```

```
vel18_2(:,2:end) = t13_vel1_f;
vel18_2(:,1) = t14_vel1_f;



% for velocity 2

% point 1
vel21_1(2:end,2:end) = t1_vel2_h;
vel21_1(1,2:end) = t2_vel2_h;
vel21_1(2:end,1) = t4_vel2_h;
vel21_1(1,1) = t6_vel2_h;

vel21_2(2:end,2:end) = t1_vel2_f;
vel21_2(1,2:end) = t2_vel2_f;
vel21_2(2:end,1) = t4_vel2_f;
vel21_2(1,1) = t6_vel2_f;

% point 2

vel22_1(2:end,:) = t10_vel2_h;
vel22_1(1,:) = t11_vel2_h;

vel22_2(2:end,:) = t10_vel2_f;
vel22_2(1,:) = t11_vel2_f;

% point 3

vel23_1(2:end,1:end-1) = t1_vel2_h;
vel23_1(1,1:end-1) = t2_vel2_h;
vel23_1(2:end,end) = t5_vel2_h;
vel23_1(1,end) = t7_vel2_h;

vel23_2(2:end,1:end-1) = t1_vel2_f;
vel23_2(1,1:end-1) = t2_vel2_f;
vel23_2(2:end,end) = t5_vel2_f;
vel23_2(1,end) = t7_vel2_f;

% point 4
```

```
    vel24_1(:,1:end-1) = t13_vel2_h;
    vel24_1(:,end) = t15_vel2_h;

    vel24_2(:,1:end-1) = t13_vel2_f;
    vel24_2(:,end) = t15_vel2_f;

    % point 5

    vel25_1(1:end-1,1:end-1) = t1_vel2_h;
    vel25_1(end,1:end-1) = t3_vel2_h;
    vel25_1(1:end-1,end) = t5_vel2_h;
    vel25_1(end,end) = t9_vel2_h;

    vel25_2(1:end-1,1:end-1) = t1_vel2_f;
    vel25_2(end,1:end-1) = t3_vel2_f;
    vel25_2(1:end-1,end) = t5_vel2_f;
    vel25_2(end,end) = t9_vel2_f;

    % point 6

    vel26_1(1:end-1,:) = t10_vel2_h;
    vel26_1(end,:) = t12_vel2_h;

    vel26_2(1:end-1,:) = t10_vel2_f;
    vel26_2(end,:) = t12_vel2_f;

    % point 7

    vel27_1(1:end-1,2:end) = t1_vel2_h;
    vel27_1(1:end-1,1) = t4_vel2_h;
    vel27_1(end,2:end) = t3_vel2_h;
    vel27_1(end,1) = t8_vel2_h;

    vel27_2(1:end-1,2:end) = t1_vel2_f;
    vel27_2(1:end-1,1) = t4_vel2_f;
    vel27_2(end,2:end) = t3_vel2_f;
    vel27_2(end,1) = t8_vel2_f;

    % point 8
```

```
vel28_1(:,2:end) = t13_vel2_h;
vel28_1(:,1) = t14_vel2_h;

vel28_2(:,2:end) = t13_vel2_f;
vel28_2(:,1) = t14_vel2_f;




% boundary conditions
%{
press1_1(1,:) = cos(2*pi*ll) * ( sin(2*pi*X2(1,1:end-1)) + sin(2*pi*Y2(1,1:
    end-1)) );
press3_1(:,end) = cos(2*pi*ll) * ( sin(2*pi*X2(1:end-1,end)) + sin(2*pi*Y2(1:
    end-1,end)) );
press7_1(:,1) = cos(2*pi*ll) * ( sin(2*pi*X2(2:end,1)) + sin(2*pi*Y2(2:end,1)
    ) );
press5_1(end,:) = cos(2*pi*ll) * ( sin(2*pi*X2(end,2:end)) + sin(2*pi*Y2(end
    ,2:end)) );
press2_1(1,:) = cos(2*pi*ll) * ( sin(2*pi*X3(1,2:end)) + sin(2*pi*Y3(1,2:end)
    ) );
press6_1(end,:) = cos(2*pi*ll) * ( sin(2*pi*X3(end,2:end)) + sin(2*pi*Y3(end
    ,2:end)) );
press4_1(:,end) = cos(2*pi*ll) * ( sin(2*pi*X4(2:end,end)) + sin(2*pi*Y4(2:
    end,end)) );
press8_1(:,1) = cos(2*pi*ll) * ( sin(2*pi*X4(2:end,1)) + sin(2*pi*Y4(2:end,1)
    ) );

press1_2(1,:) = cos(2*pi*kk) * ( sin(2*pi*X2(1,1:end-1)) + sin(2*pi*Y2(1,1:
    end-1)) );
press3_2(:,end) = cos(2*pi*kk) * ( sin(2*pi*X2(1:end-1,end)) + sin(2*pi*Y2(1:
    end-1,end)) );
press7_2(:,1) = cos(2*pi*kk) * ( sin(2*pi*X2(2:end,1)) + sin(2*pi*Y2(2:end,1)
    ) );
press5_2(end,:) = cos(2*pi*kk) * ( sin(2*pi*X2(end,2:end)) + sin(2*pi*Y2(end
    ,2:end)) );
press2_2(1,:) = cos(2*pi*kk) * ( sin(2*pi*X3(1,2:end)) + sin(2*pi*Y3(1,2:end)
    ) );
press6_2(end,:) = cos(2*pi*kk) * ( sin(2*pi*X3(end,2:end)) + sin(2*pi*Y3(end
    ,2:end)) );
press4_2(:,end) = cos(2*pi*kk) * ( sin(2*pi*X4(2:end,end)) + sin(2*pi*Y4(2:
    end,end)) );
press8_2(:,1) = cos(2*pi*kk) * ( sin(2*pi*X4(2:end,1)) + sin(2*pi*Y4(2:end,1)
    ) );

vel11_1(1,:) = sin(2*pi*ll) * cos(2*pi*X2(1,1:end-1));
vel13_1(:,end) = sin(2*pi*ll) * cos(2*pi*X2(1:end-1,end));
vel17_1(:,1) = sin(2*pi*ll) * cos(2*pi*X2(2:end,1));
vel15_1(end,:) = sin(2*pi*ll) * cos(2*pi*X2(end,2:end));
vel12_1(1,:) = sin(2*pi*ll) * cos(2*pi*X3(1,2:end));
vel16_1(end,:) = sin(2*pi*ll) * cos(2*pi*X3(end,2:end));
vel14_1(:,end) = sin(2*pi*ll) * cos(2*pi*X4(2:end,end));
vel18_1(:,1) = sin(2*pi*ll) * cos(2*pi*X4(2:end,1));

vel11_2(1,:) = sin(2*pi*kk) * cos(2*pi*X2(1,1:end-1));
vel13_2(:,end) = sin(2*pi*kk) * cos(2*pi*X2(1:end-1,end));
vel17_2(:,1) = sin(2*pi*kk) * cos(2*pi*X2(2:end,1));
vel15_2(end,:) = sin(2*pi*kk) * cos(2*pi*X2(end,2:end));
vel12_2(1,:) = sin(2*pi*kk) * cos(2*pi*X3(1,2:end));
vel16_2(end,:) = sin(2*pi*kk) * cos(2*pi*X3(end,2:end));
vel14_2(:,end) = sin(2*pi*kk) * cos(2*pi*X4(2:end,end));
vel18_2(:,1) = sin(2*pi*kk) * cos(2*pi*X4(2:end,1));

vel21_1(1,:) = sin(2*pi*ll) * cos(2*pi*Y2(1,1:end-1));
vel23_1(:,end) = sin(2*pi*ll) * cos(2*pi*Y2(1:end-1,end));
vel27_1(:,1) = sin(2*pi*ll) * cos(2*pi*Y2(2:end,1));
vel25_1(end,:) = sin(2*pi*ll) * cos(2*pi*Y2(end,2:end));
vel22_1(1,:) = sin(2*pi*ll) * cos(2*pi*Y3(1,2:end));
vel26_1(end,:) = sin(2*pi*ll) * cos(2*pi*Y3(end,2:end));
vel24_1(:,end) = sin(2*pi*ll) * cos(2*pi*Y4(2:end,end));
vel28_1(:,1) = sin(2*pi*ll) * cos(2*pi*Y4(2:end,1));

vel21_2(1,:) = sin(2*pi*kk) * cos(2*pi*Y2(1,1:end-1));
vel23_2(:,end) = sin(2*pi*kk) * cos(2*pi*Y2(1:end-1,end));
vel27_2(:,1) = sin(2*pi*kk) * cos(2*pi*Y2(2:end,1));
vel25_2(end,:) = sin(2*pi*kk) * cos(2*pi*Y2(end,2:end));
vel22_2(1,:) = sin(2*pi*kk) * cos(2*pi*Y3(1,2:end));
vel26_2(end,:) = sin(2*pi*kk) * cos(2*pi*Y3(end,2:end));
```

```
vel24_2(:,end) = sin(2*pi*kk) * cos(2*pi*Y4(2:end,end));
vel28_2(:,1) = sin(2*pi*kk) * cos(2*pi*Y4(2:end,1));
%}
% bc ends

% parameters for flux through face one

pln1 = press1;
pmn1 = press2;
prn1 = press3;
plnplushalf1 = press1_1;
pmnplushalf1 = press2_1;
prnplushalf1 = press3_1;
plnplusone1 = press1_2;
pmnplusone1 = press2_2;
prnplusone1 = press3_2;

uln1 = vel11;
umn1 = vel12;
urn1 = vel13;
ulnplushalf1 = vel11_1;
umnplushalf1 = vel12_1;
urnplushalf1 = vel13_1;
ulnplusone1 = vel11_2;
umnplusone1 = vel12_2;
urnplusone1 = vel13_2;

vln1 = vel21;
vmn1 = vel22;
vrn1 = vel23;
vlnplushalf1 = vel21_1;
vmnplushalf1 = vel22_1;
vrnplushalf1 = vel23_1;
vlnplusone1 = vel21_2;
vmnplusone1 = vel22_2;
vrnplusone1 = vel23_2;

% parameters for flux through face two
```

```
    pln2 = press3;
    pmn2 = press4;
    prn2 = press5;
    plnplushalf2 = press3_1;
    pmnplushalf2 = press4_1;
    prnplushalf2 = press5_1;
    plnplusone2 = press3_2;
    pmnplusone2 = press4_2;
    prnplusone2 = press5_2;

    uln2 = vel13;
    umn2 = vel14;
    urn2 = vel15;
    ulnplushalf2 = vel13_1;
    umnplushalf2 = vel14_1;
    urnplushalf2 = vel15_1;
    ulnplusone2 = vel13_2;
    umnplusone2 = vel14_2;
    urnplusone2 = vel15_2;

    vln2 = vel23;
    vmn2 = vel24;
    vrn2 = vel25;
    vlnplushalf2 = vel23_1;
    vmnplushalf2 = vel24_1;
    vrnplushalf2 = vel25_1;
    vlnplusone2 = vel23_2;
    vmnplusone2 = vel24_2;
    vrnplusone2 = vel25_2;

    % parameters for flux through face three

    pln3 = press5;
    pmn3 = press6;
    prn3 = press7;
    plnplushalf3 = press5_1;
    pmnplushalf3 = press6_1;
    prnplushalf3 = press7_1;
    plnplusone3 = press5_2;
    pmnplusone3 = press6_2;
    prnplusone3 = press7_2;

    uln3 = vel15;
    umn3 = vel16;
    urn3 = vel17;
    ulnplushalf3 = vel15_1;
    umnplushalf3 = vel16_1;
    urnplushalf3 = vel17_1;
    ulnplusone3 = vel15_2;
    umnplusone3 = vel16_2;
    urnplusone3 = vel17_2;

    vln3 = vel25;
    vmn3 = vel26;
    vrn3 = vel27;
    vlnplushalf3 = vel25_1;
    vmnplushalf3 = vel26_1;
    vrnplushalf3 = vel27_1;
    vlnplusone3 = vel25_2;
    vmnplusone3 = vel26_2;
    vrnplusone3 = vel27_2;

    % parameters for flux through face four

    pln4 = press7;
    pmn4 = press8;
    prn4 = press1;
    plnplushalf4 = press7_1;
    pmnplushalf4 = press8_1;
    prnplushalf4 = press1_1;
    plnplusone4 = press7_2;
    pmnplusone4 = press8_2;
    prnplusone4 = press1_2;

    uln4 = vel17;
    umn4 = vel18;
    urn4 = vel11;
    ulnplushalf4 = vel17_1;
    umnplushalf4 = vel18_1;
    urnplushalf4 = vel11_1;
    ulnplusone4 = vel17_2;
    umnplusone4 = vel18_2;
    urnplusone4 = vel11_2;

    vln4 = vel27;
    vmn4 = vel28;
    vrn4 = vel21;
    vlnplushalf4 = vel27_1;
    vmnplushalf4 = vel28_1;
    vrnplushalf4 = vel21_1;
    vlnplusone4 = vel27_2;
    vmnplusone4 = vel28_2;
    vrnplusone4 = vel21_2;

    % calculating R1, R2 and R3

    R1 = -(( vln1 + vrn1 + vlnplusone1 + vrnplusone1 ) + 4*( vmn1 + vmnplusone1 +
         vlnplushalf1 + vrnplushalf1 ) + 16*vmnplushalf1 )/36 + (( uln2 + urn2 +
        ulnplusone2 + urnplusone2 ) + 4*( umn2 + umnplusone2 + ulnplushalf2 +
        urnplushalf2 ) + 16*umnplushalf2 )/36 + (( vln3 + vrn3 + vlnplusone3 +
        vrnplusone3 ) + 4*( vmn3 + vmnplusone3 + vlnplushalf3 + vrnplushalf3 ) +
        16*vmnplushalf3 )/36 - (( uln4 + urn4 + ulnplusone4 + urnplusone4 ) + 4*(
         umn4 + umnplusone4 + ulnplushalf4 + urnplushalf4 ) + 16*umnplushalf4 )
        /36;

    R2 = (( pln2 + prn2 + plnplusone2 + prnplusone2 ) + 4*( pmn2 + pmnplusone2 +
        plnplushalf2 + prnplushalf2 ) + 16*pmnplushalf2 )/36 - (( pln4 + prn4 +
        plnplusone4 + prnplusone4 ) + 4*( pmn4 + pmnplusone4 + plnplushalf4 +
        prnplushalf4 ) + 16*pmnplushalf4 )/36;

    R3 = -(( pln1 + prn1 + plnplusone1 + prnplusone1 ) + 4*( pmn1 + pmnplusone1 +
         plnplushalf1 + prnplushalf1 ) + 16*pmnplushalf1 )/36 + (( pln3 + prn3 +
        plnplusone3 + prnplusone3 ) + 4*( pmn3 + pmnplusone3 + plnplushalf3 +
        prnplushalf3 ) + 16*pmnplushalf3 )/36;

    for i=1:sizeu
        press9_2(i) = press9(i) -(dt/area(i))*R1(i);
    end

    for i=1:sizeu
        vel19_2(i) = vel19(i) -(dt/area(i))*R2(i);
    end

    for i=1:sizeu
        vel29_2(i) = vel29(i) -(dt/area(i))*R3(i);
    end



    t1_press_iv = press1_2(2:end,2:end);
    t1_vel1_iv = vel11_2(2:end,2:end);
    t1_vel2_iv = vel21_2(2:end,2:end);

    t2_press_iv = press1_2(1,2:end);
    t2_vel1_iv = vel11_2(1,2:end);
    t2_vel2_iv = vel21_2(1,2:end);

    t3_press_iv = press7_2(end,2:end);
    t3_vel1_iv = vel17_2(end,2:end);
    t3_vel2_iv = vel27_2(end,2:end);

    t4_press_iv = press1_2(2:end,1);
    t4_vel1_iv = vel11_2(2:end,1);
    t4_vel2_iv = vel21_2(2:end,1);

    t5_press_iv = press3_2(2:end,end);
    t5_vel1_iv = vel13_2(2:end,end);
    t5_vel2_iv = vel23_2(2:end,end);

    t6_press_iv = press1_2(1,1);
    t6_vel1_iv = vel11_2(1,1);
    t6_vel2_iv = vel21_2(1,1);

    t7_press_iv = press3_2(1,end);
    t7_vel1_iv = vel13_2(1,end);
    t7_vel2_iv = vel23_2(1,end);
    t8_press_iv = press7_2(end,1);
    t8_vel1_iv = vel17_2(end,1);
    t8_vel2_iv = vel27_2(end,1);

    t9_press_iv = press5_2(end,end);
    t9_vel1_iv = vel15_2(end,end);
    t9_vel2_iv = vel25_2(end,end);

    t10_press_iv = press2_2(2:end,:);
    t10_vel1_iv = vel12_2(2:end,:);
    t10_vel2_iv = vel22_2(2:end,:);

    t11_press_iv = press2_2(1,:);
    t11_vel1_iv = vel12_2(1,:);
    t11_vel2_iv = vel22_2(1,:);

    t12_press_iv = press6_2(end,:);
    t12_vel1_iv = vel16_2(end,:);
    t12_vel2_iv = vel26_2(end,:);

    t13_press_iv = press4_2(:,1:end-1);
    t13_vel1_iv = vel14_2(:,1:end-1);
    t13_vel2_iv = vel24_2(:,1:end-1);

    t14_press_iv = press8_2(:,1);
    t14_vel1_iv = vel18_2(:,1);
    t14_vel2_iv = vel28_2(:,1);

    t15_press_iv = press4_2(:,end);
    t15_vel1_iv = vel14_2(:,end);
    t15_vel2_iv = vel24_2(:,end);

    press9 = press9_2;
    press1 = press1_2;
    press2 = press2_2;
    press3 = press3_2;
    press4 = press4_2;
    press5 = press5_2;
    press6 = press6_2;
    press7 = press7_2;
    press8 = press8_2;

    vel19 = vel19_2;
    vel11 = vel11_2;
    vel12 = vel12_2;
    vel13 = vel13_2;
    vel14 = vel14_2;
    vel15 = vel15_2;
    vel16 = vel16_2;
    vel17 = vel17_2;
    vel18 = vel18_2;

    vel29 = vel29_2;
    vel21 = vel21_2;
    vel22 = vel22_2;
    vel23 = vel23_2;
    vel24 = vel24_2;
    vel25 = vel25_2;
    vel26 = vel26_2;
    vel27 = vel27_2;
    vel28 = vel28_2;

    figure(7)
    surf(X1(2:end-1,2:end-1),Y1(2:end-1,2:end-1),press9_2(2:end-1,2:end-1))
    %view(2)
    colorbar
    drawnow

    figure(8)
    surf(X1(2:end-1,2:end-1),Y1(2:end-1,2:end-1),vel19_2(2:end-1,2:end-1))
    %view(2)
    colorbar
    drawnow

    figure(9)
    surf(X1(2:end-1,2:end-1),Y1(2:end-1,2:end-1),vel29_2(2:end-1,2:end-1))
    %view(2)
    colorbar
    drawnow
    kk = kk + dt;
    ll = ll + dt;
end

forl2 = Delta_x*Delta_y*dt*ones(lx1,lx1);

L2_press=sqrt((1/((xb-xa+2*Delta_x)*(yb-ya+2*Delta_y)*(dt))).*sum(sum(abs(
    press9_2.*forl2-exactp.*forl2).^2)))

L2_vel1=sqrt((1/((xb-xa+2*Delta_x)*(yb-ya+2*Delta_y)*(dt))).*sum(sum(abs(vel19_2
    .*forl2-exactvel1.*forl2).^2)))

L2_vel2=sqrt((1/((xb-xa+2*Delta_x)*(yb-ya+2*Delta_y)*(dt))).*sum(sum(abs(vel29_2
    .*forl2-exactvel2.*forl2).^2)))

ncell = sizeu;
nface = No_of_edges;
nvertex = sizev;
DOF = ncell + nface/2 + nvertex/4
h = 1/sqrt(DOF)
```

$\mathcal{B}$

APPENDIX B

B.1 Mathematica Coding

b1 $= -x*y*(2*x+2*y+3*g)/g\hat{}3$

$-\frac{xy(3g+2x+2y)}{g^3}$

b2 $= y*(2*x-g)*(2*x+g)/g\hat{}3$

$\frac{(-g+2x)(g+2x)y}{g^3}$

b3 $= x*y*(2*y-2*x+3*g)/g\hat{}3$

$\frac{xy(3g-2x+2y)}{g^3}$

b4 $= -x*(2*y-g)*(2*y+g)/g\hat{}3$

$-\frac{x(-g+2y)(g+2y)}{g^3}$

b5 $= x*y*(2*x+2*y-3*g)/g\hat{}3$

$\frac{xy(-3g+2x+2y)}{g^3}$

b6 $= -y*(2*x-g)(2*x+g)/g\hat{}3$

$-\frac{(-g+2x)(g+2x)y}{g^3}$

b7 $= x*y*(2*x+3*g-2*y)/g\hat{}3$

$\frac{x(3g+2x-2y)y}{g^3}$

b8 $= x*(2*y-g)*(2*y+g)/g\hat{}3$

$\frac{x(-g+2y)(g+2y)}{g^3}$

b1x $= D[$**b1**$, x]$

$-\frac{2xy}{g^3} - \frac{y(3g+2x+2y)}{g^3}$

b2x $= D[$**b2**$, x]$

$\frac{2(-g+2x)y}{g^3} + \frac{2(g+2x)y}{g^3}$

b3x $= D[$**b3**$, x]$

$-\frac{2xy}{g^3} + \frac{y(3g-2x+2y)}{g^3}$

b4x $= D[$**b4**$, x]$

$-\frac{(-g+2y)(g+2y)}{g^3}$

b5x $= D[$**b5**$, x]$

$\frac{2xy}{g^3} + \frac{y(-3g+2x+2y)}{g^3}$

b6x $= D[$**b6**$, x]$

$-\frac{2(-g+2x)y}{g^3} - \frac{2(g+2x)y}{g^3}$

b7x $= D[$**b7**$, x]$

$\frac{2xy}{g^3} + \frac{(3g+2x-2y)y}{g^3}$

b8x $= D[$**b8**$, x]$

$\frac{(-g+2y)(g+2y)}{g^3}$

b1y $= D[$**b1**$, y]$

$-\frac{2xy}{g^3} - \frac{x(3g+2x+2y)}{g^3}$

b2y $= D[\mathbf{b2}, y]$

$$\frac{(-g+2x)(g+2x)}{g^3}$$

b3y $= D[\mathbf{b3}, y]$

$$\frac{2xy}{g^3}+\frac{x(3g-2x+2y)}{g^3}$$

b4y $= D[\mathbf{b4}, y]$

$$-\frac{2x(-g+2y)}{g^3}-\frac{2x(g+2y)}{g^3}$$

b5y $= D[\mathbf{b5}, y]$

$$\frac{2xy}{g^3}+\frac{x(-3g+2x+2y)}{g^3}$$

b6y $= D[\mathbf{b6}, y]$

$$-\frac{(-g+2x)(g+2x)}{g^3}$$

b7y $= D[\mathbf{b7}, y]$

$$\frac{x(3g+2x-2y)}{g^3}-\frac{2xy}{g^3}$$

b8y $= D[\mathbf{b8}, y]$

$$\frac{2x(-g+2y)}{g^3}+\frac{2x(g+2y)}{g^3}$$

c1 $=$ **b1**$/.x \to \{s * \mathrm{Cos}[\varphi]\}$

$$\left\{-\frac{sy\mathrm{Cos}[\varphi](3g+2y+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

d1 $=$ **c1**$/.y \to \{s * \mathrm{Sin}[\varphi]\}$

$$\left\{\left\{-\frac{s^2\mathrm{Cos}[\varphi]\mathrm{Sin}[\varphi](3g+2s\mathrm{Cos}[\varphi]+2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

c2 $=$ **b2**$/.x \to \{s * \mathrm{Cos}[\varphi]\}$

$$\left\{\frac{y(-g+2s\mathrm{Cos}[\varphi])(g+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

d2 $=$ **c2**$/.y \to \{s * \mathrm{Sin}[\varphi]\}$

$$\left\{\left\{\frac{s(-g+2s\text{Cos}[\varphi])(g+2s\text{Cos}[\varphi])\text{Sin}[\varphi]}{g^3}\right\}\right\}$$

c3 = b3/.$x \to \{s * \text{Cos}[\varphi]\}$

$$\left\{\frac{sy\text{Cos}[\varphi](3g+2y-2s\text{Cos}[\varphi])}{g^3}\right\}$$

d3 = c3/.$y \to \{s * \text{Sin}[\varphi]\}$

$$\left\{\left\{\frac{s^2\text{Cos}[\varphi]\text{Sin}[\varphi](3g-2s\text{Cos}[\varphi]+2s\text{Sin}[\varphi])}{g^3}\right\}\right\}$$

c4 = b4/.$x \to \{s * \text{Cos}[\varphi]\}$

$$\left\{-\frac{s(-g+2y)(g+2y)\text{Cos}[\varphi]}{g^3}\right\}$$

c5 = b5/.$x \to \{s * \text{Cos}[\varphi]\}$

$$\left\{\frac{sy\text{Cos}[\varphi](-3g+2y+2s\text{Cos}[\varphi])}{g^3}\right\}$$

c6 = b6/.$x \to \{s * \text{Cos}[\varphi]\}$

$$\left\{-\frac{y(-g+2s\text{Cos}[\varphi])(g+2s\text{Cos}[\varphi])}{g^3}\right\}$$

c7 = b7/.$x \to \{s * \text{Cos}[\varphi]\}$

$$\left\{\frac{sy\text{Cos}[\varphi](3g-2y+2s\text{Cos}[\varphi])}{g^3}\right\}$$

c8 = b8/.$x \to \{s * \text{Cos}[\varphi]\}$

$$\left\{\frac{s(-g+2y)(g+2y)\text{Cos}[\varphi]}{g^3}\right\}$$

d1

$$\left\{\left\{-\frac{s^2\text{Cos}[\varphi]\text{Sin}[\varphi](3g+2s\text{Cos}[\varphi]+2s\text{Sin}[\varphi])}{g^3}\right\}\right\}$$

d2

$$\left\{\left\{\frac{s(-g+2s\text{Cos}[\varphi])(g+2s\text{Cos}[\varphi])\text{Sin}[\varphi]}{g^3}\right\}\right\}$$

d3

$$\left\{\left\{\frac{s^2\text{Cos}[\varphi]\text{Sin}[\varphi](3g-2s\text{Cos}[\varphi]+2s\text{Sin}[\varphi])}{g^3}\right\}\right\}$$

$\mathbf{d4 = c4/.y \to \{s * Sin[\varphi]\}}$

$$\left\{\left\{-\frac{s\text{Cos}[\varphi](-g+2s\text{Sin}[\varphi])(g+2s\text{Sin}[\varphi])}{g^3}\right\}\right\}$$

$\mathbf{d5 = c5/.y \to \{s * Sin[\varphi]\}}$

$$\left\{\left\{\frac{s^2\text{Cos}[\varphi]\text{Sin}[\varphi](-3g+2s\text{Cos}[\varphi]+2s\text{Sin}[\varphi])}{g^3}\right\}\right\}$$

$\mathbf{d6 = c6/.y \to \{s * Sin[\varphi]\}}$

$$\left\{\left\{-\frac{s(-g+2s\text{Cos}[\varphi])(g+2s\text{Cos}[\varphi])\text{Sin}[\varphi]}{g^3}\right\}\right\}$$

$\mathbf{d7 = c7/.y \to \{s * Sin[\varphi]\}}$

$$\left\{\left\{\frac{s^2\text{Cos}[\varphi]\text{Sin}[\varphi](3g+2s\text{Cos}[\varphi]-2s\text{Sin}[\varphi])}{g^3}\right\}\right\}$$

$\mathbf{d8 = c8/.y \to \{s * Sin[\varphi]\}}$

$$\left\{\left\{\frac{s\text{Cos}[\varphi](-g+2s\text{Sin}[\varphi])(g+2s\text{Sin}[\varphi])}{g^3}\right\}\right\}$$

$\mathbf{c1x = b1x/.x \to \{s * Cos[\varphi]\}}$

$$\left\{-\frac{2sy\text{Cos}[\varphi]}{g^3}-\frac{y(3g+2y+2s\text{Cos}[\varphi])}{g^3}\right\}$$

$\mathbf{c2x = b2x/.x \to \{s * Cos[\varphi]\}}$

$$\left\{\frac{2y(-g+2s\text{Cos}[\varphi])}{g^3}+\frac{2y(g+2s\text{Cos}[\varphi])}{g^3}\right\}$$

$\mathbf{c3x = b3x/.x \to \{s * Cos[\varphi]\}}$

$$\left\{-\frac{2sy\text{Cos}[\varphi]}{g^3}+\frac{y(3g+2y-2s\text{Cos}[\varphi])}{g^3}\right\}$$

$\mathbf{c4x = b4x/.x \to \{s * Cos[\varphi]\}}$

$$-\frac{(-g+2y)(g+2y)}{g^3}$$

$\mathbf{c5x = b5x/.x \to \{s * Cos[\varphi]\}}$

$$\left\{\frac{2sy\text{Cos}[\varphi]}{g^3}+\frac{y(-3g+2y+2s\text{Cos}[\varphi])}{g^3}\right\}$$

$\mathbf{c6x = b6x/.x \to \{s * Cos[\varphi]\}}$

$$\left\{-\frac{2y(-g+2s\mathrm{Cos}[\varphi])}{g^3}-\frac{2y(g+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

$\mathbf{c7x = b7x/.}x \to \{s * \mathrm{Cos}[\varphi]\}$

$$\left\{\frac{2sy\mathrm{Cos}[\varphi]}{g^3}+\frac{y(3g-2y+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

$\mathbf{c8x = b8x/.}x \to \{s * \mathrm{Cos}[\varphi]\}$

$$\frac{(-g+2y)(g+2y)}{g^3}$$

$\mathbf{d1x = c1x/.}y \to \{s * \mathrm{Sin}[\varphi]\}$

$$\left\{\left\{-\frac{2s^2\mathrm{Cos}[\varphi]\mathrm{Sin}[\varphi]}{g^3}-\frac{s\mathrm{Sin}[\varphi](3g+2s\mathrm{Cos}[\varphi]+2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

$\mathbf{d2x = c2x/.}y \to \{s * \mathrm{Sin}[\varphi]\}$

$$\left\{\left\{\frac{2s(-g+2s\mathrm{Cos}[\varphi])\mathrm{Sin}[\varphi]}{g^3}+\frac{2s(g+2s\mathrm{Cos}[\varphi])\mathrm{Sin}[\varphi]}{g^3}\right\}\right\}$$

$\mathbf{d3x = c3x/.}y \to \{s * \mathrm{Sin}[\varphi]\}$

$$\left\{\left\{-\frac{2s^2\mathrm{Cos}[\varphi]\mathrm{Sin}[\varphi]}{g^3}+\frac{s\mathrm{Sin}[\varphi](3g-2s\mathrm{Cos}[\varphi]+2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

$\mathbf{d4x = c4x/.}y \to \{s * \mathrm{Sin}[\varphi]\}$

$$\left\{-\frac{(-g+2s\mathrm{Sin}[\varphi])(g+2s\mathrm{Sin}[\varphi])}{g^3}\right\}$$

$\mathbf{d5x = c5x/.}y \to \{s * \mathrm{Sin}[\varphi]\}$

$$\left\{\left\{\frac{2s^2\mathrm{Cos}[\varphi]\mathrm{Sin}[\varphi]}{g^3}+\frac{s\mathrm{Sin}[\varphi](-3g+2s\mathrm{Cos}[\varphi]+2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

$\mathbf{d6x = c6x/.}y \to \{s * \mathrm{Sin}[\varphi]\}$

$$\left\{\left\{-\frac{2s(-g+2s\mathrm{Cos}[\varphi])\mathrm{Sin}[\varphi]}{g^3}-\frac{2s(g+2s\mathrm{Cos}[\varphi])\mathrm{Sin}[\varphi]}{g^3}\right\}\right\}$$

$\mathbf{d7x = c7x/.}y \to \{s * \mathrm{Sin}[\varphi]\}$

$$\left\{\left\{\frac{2s^2\mathrm{Cos}[\varphi]\mathrm{Sin}[\varphi]}{g^3}+\frac{s\mathrm{Sin}[\varphi](3g+2s\mathrm{Cos}[\varphi]-2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

$\mathbf{d8x = c8x/.}y \to \{s * \mathrm{Sin}[\varphi]\}$

$$\left\{\frac{(-g+2s\mathrm{Sin}[\varphi])(g+2s\mathrm{Sin}[\varphi])}{g^3}\right\}$$

$$\mathbf{c1y} = \mathbf{b1y}/.x \to \{s * \mathrm{Cos}[\varphi]\}$$

$$\left\{-\frac{2sy\mathrm{Cos}[\varphi]}{g^3} - \frac{s\mathrm{Cos}[\varphi](3g+2y+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

$$\mathbf{c2y} = \mathbf{b2y}/.x \to \{s * \mathrm{Cos}[\varphi]\}$$

$$\left\{\frac{(-g+2s\mathrm{Cos}[\varphi])(g+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

$$\mathbf{c3y} = \mathbf{b3y}/.x \to \{s * \mathrm{Cos}[\varphi]\}$$

$$\left\{\frac{2sy\mathrm{Cos}[\varphi]}{g^3} + \frac{s\mathrm{Cos}[\varphi](3g+2y-2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

$$\mathbf{c4y} = \mathbf{b4y}/.x \to \{s * \mathrm{Cos}[\varphi]\}$$

$$\left\{-\frac{2s(-g+2y)\mathrm{Cos}[\varphi]}{g^3} - \frac{2s(g+2y)\mathrm{Cos}[\varphi]}{g^3}\right\}$$

$$\mathbf{c5y} = \mathbf{b5y}/.x \to \{s * \mathrm{Cos}[\varphi]\}$$

$$\left\{\frac{2sy\mathrm{Cos}[\varphi]}{g^3} + \frac{s\mathrm{Cos}[\varphi](-3g+2y+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

$$\mathbf{c6y} = \mathbf{b6y}/.x \to \{s * \mathrm{Cos}[\varphi]\}$$

$$\left\{-\frac{(-g+2s\mathrm{Cos}[\varphi])(g+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

$$\mathbf{c7y} = \mathbf{b7y}/.x \to \{s * \mathrm{Cos}[\varphi]\}$$

$$\left\{-\frac{2sy\mathrm{Cos}[\varphi]}{g^3} + \frac{s\mathrm{Cos}[\varphi](3g-2y+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

$$\mathbf{c8y} = \mathbf{b8y}/.x \to \{s * \mathrm{Cos}[\varphi]\}$$

$$\left\{\frac{2s(-g+2y)\mathrm{Cos}[\varphi]}{g^3} + \frac{2s(g+2y)\mathrm{Cos}[\varphi]}{g^3}\right\}$$

$$\mathbf{d1y} = \mathbf{c1y}/.y \to \{s * \mathrm{Sin}[\varphi]\}$$

$$\left\{\left\{-\frac{2s^2\mathrm{Cos}[\varphi]\mathrm{Sin}[\varphi]}{g^3} - \frac{s\mathrm{Cos}[\varphi](3g+2s\mathrm{Cos}[\varphi]+2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

$$\mathbf{d2y} = \mathbf{c2y}/.y \to \{s * \mathrm{Sin}[\varphi]\}$$

$$\left\{\frac{(-g+2s\mathrm{Cos}[\varphi])(g+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

$$\mathbf{d3y} = \mathbf{c3y}/.y \to \{s * \mathrm{Sin}[\varphi]\}$$

$$\left\{\left\{\frac{2s^2\mathrm{Cos}[\varphi]\mathrm{Sin}[\varphi]}{g^3}+\frac{s\mathrm{Cos}[\varphi](3g-2s\mathrm{Cos}[\varphi]+2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

$$\mathbf{d4y = c4y/.}y \to \{s * \mathrm{Sin}[\varphi]\}$$

$$\left\{\left\{-\frac{2s\mathrm{Cos}[\varphi](-g+2s\mathrm{Sin}[\varphi])}{g^3}-\frac{2s\mathrm{Cos}[\varphi](g+2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

$$\mathbf{d5y = c5y/.}y \to \{s * \mathrm{Sin}[\varphi]\}$$

$$\left\{\left\{\frac{2s^2\mathrm{Cos}[\varphi]\mathrm{Sin}[\varphi]}{g^3}+\frac{s\mathrm{Cos}[\varphi](-3g+2s\mathrm{Cos}[\varphi]+2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

$$\mathbf{d6y = c6y/.}y \to \{s * \mathrm{Sin}[\varphi]\}$$

$$\left\{-\frac{(-g+2s\mathrm{Cos}[\varphi])(g+2s\mathrm{Cos}[\varphi])}{g^3}\right\}$$

$$\mathbf{d7y = c7y/.}y \to \{s * \mathrm{Sin}[\varphi]\}$$

$$\left\{\left\{-\frac{2s^2\mathrm{Cos}[\varphi]\mathrm{Sin}[\varphi]}{g^3}+\frac{s\mathrm{Cos}[\varphi](3g+2s\mathrm{Cos}[\varphi]-2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

$$\mathbf{d8y = c8y/.}y \to \{s * \mathrm{Sin}[\varphi]\}$$

$$\left\{\left\{\frac{2s\mathrm{Cos}[\varphi](-g+2s\mathrm{Sin}[\varphi])}{g^3}+\frac{2s\mathrm{Cos}[\varphi](g+2s\mathrm{Sin}[\varphi])}{g^3}\right\}\right\}$$

$$\mathbf{I1} = \frac{1}{2*\pi*r}\left(\mathbf{Integrate}\left[\mathbf{d1} * \frac{s}{\mathbf{Sqrt}[r^2-s^2]}, \{s,0,r\}, \{\varphi,0,\pi/2\}\right]\right)$$

$$\left\{\left\{\mathrm{ConditionalExpression}\left[-\frac{r^2(4g+\pi r)}{8g^3\pi}, \mathrm{Re}[r]>0\&\&\mathrm{Im}[r]==0\right]\right\}\right\}$$

$$\mathbf{I2} = \frac{1}{2*\pi*r}\left(\mathbf{Integrate}\left[\mathbf{d2} * \frac{s}{\mathbf{Sqrt}[r^2-s^2]}, \{s,0,r\}, \{\varphi,0,\pi\}\right]\right)$$

$$\left\{\left\{\mathrm{ConditionalExpression}\left[\frac{r\left(-g^2+r^2\right)}{4g^3}, \mathrm{Re}[r]>0\&\&\mathrm{Im}[r]==0\right]\right\}\right\}$$

$$\mathbf{I3} = \frac{1}{2*\pi*r}\left(\mathbf{Integrate}\left[\mathbf{d3} * \frac{s}{\mathbf{Sqrt}[r^2-s^2]}, \{s,0,r\}, \{\varphi,\pi/2,\pi\}\right]\right)$$

$$\left\{\left\{\mathrm{ConditionalExpression}\left[-\frac{r^2(4g+\pi r)}{8g^3\pi}, \mathrm{Re}[r]>0\&\&\mathrm{Im}[r]==0\right]\right\}\right\}$$

$$\mathbf{I4} = \frac{1}{2*\pi*r}\left(\mathbf{Integrate}\left[\mathbf{d4} * \frac{s}{\mathbf{Sqrt}[r^2-s^2]}, \{s,0,r\}, \{\varphi,\pi/2,3*\pi/2\}\right]\right)$$

$$\left\{\left\{\mathrm{ConditionalExpression}\left[\frac{r\left(-g^2+r^2\right)}{4g^3}, \mathrm{Re}[r]>0\&\&\mathrm{Im}[r]==0\right]\right\}\right\}$$

$$\mathbf{I5} = \frac{1}{2*\pi*r}\left(\mathbf{Integrate}\left[\mathbf{d5} * \frac{s}{\mathbf{Sqrt}[r^2-s^2]}, \{s,0,r\}, \{\varphi,\pi,3*\pi/2\}\right]\right)$$

$\left\{\left\{\text{ConditionalExpression}\left[-\frac{r^2(4g+\pi r)}{8g^3\pi}, \text{Re}[r] > 0\&\&\text{Im}[r] == 0\right]\right\}\right\}$

$\mathbf{I6 = \frac{1}{2*\pi*r}\left(\text{Integrate}\left[d6 * \frac{s}{\text{Sqrt}[r^2-s^2]}, \{s, 0, r\}, \{\varphi, \pi, 2*\pi\}\right]\right)}$

$\left\{\left\{\text{ConditionalExpression}\left[\frac{r\left(-g^2+r^2\right)}{4g^3}, \text{Re}[r] > 0\&\&\text{Im}[r] == 0\right]\right\}\right\}$

$\mathbf{I7 = \frac{1}{2*\pi*r}\left(\text{Integrate}\left[d7 * \frac{s}{\text{Sqrt}[r^2-s^2]}, \{s, 0, r\}, \{\varphi, -\pi/2, 0\}\right]\right)}$

$\left\{\left\{\text{ConditionalExpression}\left[-\frac{r^2(4g+\pi r)}{8g^3\pi}, \text{Re}[r] > 0\&\&\text{Im}[r] == 0\right]\right\}\right\}$

$\mathbf{I8 = \frac{1}{2*\pi*r}\left(\text{Integrate}\left[d8 * \frac{s}{\text{Sqrt}[r^2-s^2]}, \{s, 0, r\}, \{\varphi, -\pi/2, \pi/2\}\right]\right)}$

$\left\{\left\{\text{ConditionalExpression}\left[\frac{r\left(-g^2+r^2\right)}{4g^3}, \text{Re}[r] > 0\&\&\text{Im}[r] == 0\right]\right\}\right\}$

$\mathbf{I1x = \frac{1}{2*\pi*r}\left(\text{Integrate}\left[d1x * \frac{s}{\text{Sqrt}[r^2-s^2]}, \{s, 0, r\}, \{\varphi, 0, \pi/2\}\right]\right)}$

$\mathbf{\left\{\left\{\text{ConditionalExpression}\left[-\frac{r(9g\pi+4(4+\pi)r)}{24g^3\pi}, \text{Re}[r] > 0\&\&\text{Im}[r] == 0\right]\right\}\right\}}$

$\mathbf{I1y = \frac{1}{2*\pi*r}\left(\text{Integrate}\left[d1y * \frac{s}{\text{Sqrt}[r^2-s^2]}, \{s, 0, r\}, \{\varphi, 0, \pi/2\}\right]\right)}$

$\left\{\left\{\text{ConditionalExpression}\left[-\frac{r(9g\pi+4(4+\pi)r)}{24g^3\pi}, \text{Re}[r] > 0\&\&\text{Im}[r] == 0\right]\right\}\right\}$

$\mathbf{I2x = \frac{1}{2*\pi*r}\left(\text{Integrate}\left[d2x * \frac{s}{\text{Sqrt}[r^2-s^2]}, \{s, 0, r\}, \{\varphi, 0, \pi\}\right]\right)}$

$\{\{0\}\}$

$\mathbf{I2y = \frac{1}{2*\pi*r}\left(\text{Integrate}\left[d2y * \frac{s}{\text{Sqrt}[r^2-s^2]}, \{s, 0, r\}, \{\varphi, 0, \pi\}\right]\right)}$

$\left\{\text{ConditionalExpression}\left[\frac{-3g^2+4r^2}{6g^3}, \text{Re}[r] > 0\&\&\text{Im}[r] == 0\right]\right\}$

$\mathbf{I3x = \frac{1}{2*\pi*r}\left(\text{Integrate}\left[d3x * \frac{s}{\text{Sqrt}[r^2-s^2]}, \{s, 0, r\}, \{\varphi, \pi/2, \pi\}\right]\right)}$

$\left\{\left\{\text{ConditionalExpression}\left[\frac{r(9g\pi+4(4+\pi)r)}{24g^3\pi}, \text{Re}[r] > 0\&\&\text{Im}[r] == 0\right]\right\}\right\}$

$\mathbf{I3y = \frac{1}{2*\pi*r}\left(\text{Integrate}\left[d3y * \frac{s}{\text{Sqrt}[r^2-s^2]}, \{s, 0, r\}, \{\varphi, \pi/2, \pi\}\right]\right)}$

$\left\{\left\{\text{ConditionalExpression}\left[-\frac{r(9g\pi+4(4+\pi)r)}{24g^3\pi}, \text{Re}[r] > 0\&\&\text{Im}[r] == 0\right]\right\}\right\}$

$\mathbf{I4x = \frac{1}{2*\pi*r}\left(\text{Integrate}\left[d4x * \frac{s}{\text{Sqrt}[r^2-s^2]}, \{s, 0, r\}, \{\varphi, \pi/2, 3*\pi/2\}\right]\right)}$

$$\left\{\text{ConditionalExpression}\left[\frac{\frac{\pi r}{g}-\frac{4\pi r^3}{3g^3}}{2\pi r},\text{Re}[r]>0\&\&\text{Im}[r]==0\right]\right\}$$

$$\mathbf{I4y}=\frac{1}{2*\pi*r}\left(\text{Integrate}\left[\text{d4y}*\frac{s}{\text{Sqrt}[r^2-s^2]},\{s,0,r\},\{\varphi,\pi/2,3*\pi/2\}\right]\right)$$

$$\{\{0\}\}$$

$$\mathbf{I5x}=\frac{1}{2*\pi*r}\left(\text{Integrate}\left[\text{d5x}*\frac{s}{\text{Sqrt}[r^2-s^2]},\{s,0,r\},\{\varphi,\pi,3*\pi/2\}\right]\right)$$

$$\left\{\left\{\text{ConditionalExpression}\left[\frac{r(9g\pi+4(4+\pi)r)}{24g^3\pi},\text{Re}[r]>0\&\&\text{Im}[r]==0\right]\right\}\right\}$$

$$\mathbf{I5y}=\frac{1}{2*\pi*r}\left(\text{Integrate}\left[\text{d5y}*\frac{s}{\text{Sqrt}[r^2-s^2]},\{s,0,r\},\{\varphi,\pi,3*\pi/2\}\right]\right)$$

$$\left\{\left\{\text{ConditionalExpression}\left[\frac{r(9g\pi+4(4+\pi)r)}{24g^3\pi},\text{Re}[r]>0\&\&\text{Im}[r]==0\right]\right\}\right\}$$

$$\mathbf{I6x}=\frac{1}{2*\pi*r}\left(\text{Integrate}\left[\text{d6x}*\frac{s}{\text{Sqrt}[r^2-s^2]},\{s,0,r\},\{\varphi,\pi,2*\pi\}\right]\right)$$

$$\{\{0\}\}$$

$$\mathbf{I6y}=\frac{1}{2*\pi*r}\left(\text{Integrate}\left[\text{d6y}*\frac{s}{\text{Sqrt}[r^2-s^2]},\{s,0,r\},\{\varphi,\pi,2*\pi\}\right]\right)$$

$$\left\{\text{ConditionalExpression}\left[\frac{\frac{\pi r}{g}-\frac{4\pi r^3}{3g^3}}{2\pi r},\text{Re}[r]>0\&\&\text{Im}[r]==0\right]\right\}$$

$$\mathbf{I7x}=\frac{1}{2*\pi*r}\left(\text{Integrate}\left[\text{d7x}*\frac{s}{\text{Sqrt}[r^2-s^2]},\{s,0,r\},\{\varphi,-\pi/2,0\}\right]\right)$$

$$\left\{\left\{\text{ConditionalExpression}\left[-\frac{r(9g\pi+4(4+\pi)r)}{24g^3\pi},\text{Re}[r]>0\&\&\text{Im}[r]==0\right]\right\}\right\}$$

$$\mathbf{I7y}=\frac{1}{2*\pi*r}\left(\text{Integrate}\left[\text{d7y}*\frac{s}{\text{Sqrt}[r^2-s^2]},\{s,0,r\},\{\varphi,-\pi/2,0\}\right]\right)$$

$$\left\{\left\{\text{ConditionalExpression}\left[\frac{r(9g\pi+4(4+\pi)r)}{24g^3\pi},\text{Re}[r]>0\&\&\text{Im}[r]==0\right]\right\}\right\}$$

$$\mathbf{I8x}=\frac{1}{2*\pi*r}\left(\text{Integrate}\left[\text{d8x}*\frac{s}{\text{Sqrt}[r^2-s^2]},\{s,0,r\},\{\varphi,-\pi/2,\pi/2\}\right]\right)$$

$$\left\{\text{ConditionalExpression}\left[\frac{-3g^2+4r^2}{6g^3},\text{Re}[r]>0\&\&\text{Im}[r]==0\right]\right\}$$

$$\mathbf{I8y}=\frac{1}{2*\pi*r}\left(\text{Integrate}\left[\text{d8y}*\frac{s}{\text{Sqrt}[r^2-s^2]},\{s,0,r\},\{\varphi,-\pi/2,\pi/2\}\right]\right)$$

$$\{\{0\}\}$$

BIBLIOGRAPHY

[0]

BIBLIOGRAPHY

[1] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[2] L. C. Evans, *PARTIAL DIFFERENTIAL EQUATIONS (GRADUATE STUDIES IN MATHEMATICS)*. American Mathematical Society, 2014. 2

[3] U. S. F. Siddhartha Mishra and R. Abgrall, "Numerical methods for conservation laws and related equations." ETH Zurich. 2, 5

[4] R. J. LeVeque and R. J. Leveque, *Numerical methods for conservation laws*, vol. 132. Springer, 1992. 5, 18

[5] R. J. LeVeque, *Finite volume methods for hyperbolic problems*, vol. 31. Cambridge university press, 2002. 5

[6] E. Godlewski and P.-A. Raviart, *Numerical approximation of hyperbolic systems of conservation laws*, vol. 118. Springer Science & Business Media, 2013. 5

[7] T. Eymann and P. Roe, "Active flux schemes," in *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, p. 382, 2011. 6, 9, 12, 64

[8] T. A. Eymann and P. L. Roe, "Multidimensional active flux schemes," in *21st AIAA computational fluid dynamics conference*, p. 2940, 2013. 9, 24, 64

[9] T. Eymann and P. Roe, "Active flux schemes for systems," in *20th AIAA computational fluid dynamics conference*, p. 3840, 2011. 9

[10] T. A. Eymann, *Active Flux Schemes.* PhD thesis, University of Michigan, 2013. 9

[11] J. B. Maeng, *On the Advective Component of Active Flux Schemes for Nonlinear Hyperbolic Conservation Laws*. PhD thesis, University of Michigan, 2017. 9, 12, 55

[12] D. Fan, *On the Acoustic Component of Active Flux Schemes for Nonlinear Hyperbolic Conservation Laws*. PhD thesis, University of Michigan, 2017. 9, 24

[13] W. Barsukow, J. Hohm, C. Klingenberg, and P. L. Roe, "The active flux scheme on cartesian grids and its low mach number limit," *arXiv preprint arXiv:1812.01612*, 2018. 9, 64

[14] G. D. Smith, *Numerical solution of partial differential equations: finite difference methods*. Oxford university press, 1985. 12

[15] B. Van Leer, "Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov's method," *Journal of computational Physics*, vol. 32, no. 1, pp. 101–136, 1979. 15

[16] J. Kunnen, "Numerical study of adaptive mesh refinement applied to a third order minimum truncation error active flux method," Master's thesis, Delft University of Technology, 2018. 16

[17] H. L. Eric Darve, "Matlab workbook." Leland Stanford Junior University. 39

[18] W. R. Inc., "Mathematica, Version 12.0." Champaign, IL, 2019.

[19] M. Lukáčová-Medvid'ová, K. Morton, and G. Warnecke, "Evolution galerkin methods for hyperbolic systems in two space dimensions," *Mathematics of Computation of the American Mathematical Society*, vol. 69, no. 232, pp. 1355–1384, 2000. 59

[20] C. K. WASILIJ BARSUKOW, “Exact solution and a truly multidimensional godunov scheme for the acoustic equations,” *identity*, 2017. 58