Numerical Simulation of Nonlinear and Dispersive Wave Equations using Adaptive Mesh Refinement (AMR)

by

G M Ashikur Rahman

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Applied Mathematics

Waterloo, Ontario, Canada, 2020

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Numerical solution of time dependent Partial Differential Equations plays an important role in different fluid flow modelling problems. Sometimes a little portion of the computational domain needs high grid resolution in order to resolve phenomena such as steep fronts or shocks while use of a very high resolution mesh for the whole computational domain is a waste of computational resources since they are not required all over the domain. An Adaptive Mesh Refinement (AMR) procedure is an efficient and practical method for the numerical solution of Partial Differential Equation problems with regions of large gradients occupying a small subregion of the domain. An AMR algorithm refines grids by placing finer and finer subgrids in the different portions of the computational domain where they are required. For the time dependent problem the refinement is dynamic since the regions requiring refinement change with time and the AMR algorithm adaptively changes that. In this thesis we developed an AMR code for the numerical solution of linear, nonlinear and dispersive wave equations inspired by existing algorithms in the literature. In this work we kept the implementation simple and we use simple refinement criteria although the code allows for the use of more complex refinement criteria. In addition the implementation of the data structure was also kept simple. We have done the refinement in both time and space. In our code we generate finer grids which can also have finer grids using a recursive grid generation procedure. We give a review of some existing work along with the necessary components of our work. Numerical simulations of the linear advection equation, Burger's equation and the Regularized Long Wave (RLW) equation have been run with our AMR code. The results of these simulations are shown to have good agreement with numerical solutions obtained on fine resolution single grids which signify the success of our code. A significant time reduction in all the numerical simulations suggests the good performance of our code.

Acknowledgements

First of all, I would like to express my heartfelt gratitude to Almighty Allah – the most benevolent and most merciful – whose eternal grace gave me the ability to work for the research work presented here.

I would like to thank my supervisor Professor Kevin Lamb for all of his continuous support and sincere advice throughout the time of my graduate studies. I also express my heartfelt gratitude to him for his inspiration, timely suggestion, friendly attitude which have helped me to complete this thesis. I also would like to thank him for his close inspection of my work at every stage. He provided me all the essential support which help me to complete my work.

I would like to thank Professor Marek Stastna and Professor Sander Rhebergen for their time to review my work.

I also profusely thank my colleague Yang, Abdulmajid and Dawei Wang for their friendly support.

I would like to express fervid affection to my family members, especially to my father, mother and wife whose mental fortitude made this piece of work possible.

Dedication

This thesis is dedicated to my parents.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Numerical solution of partial differential equations (PDE) is an essential tool in all fields of science and engineering. It is useful for dealing with problems difficult to solve analytically. Using a uniform computational grid with any numerical method may become computationally expensive when high resolution is required to resolve features of interest that occupy a small part of the domain. An adaptive procedure is one of the best ways to counter this problem. In this thesis a simple adaptive algorithm will be used for solving different nonlinear and dispersive wave equations.

1.1 Introduction

The solution accuracy of numerical methods such as finite-difference or finite-volume methods depends on the mesh resolution. In a fixed domain, the accuracy of the result generally increases as the mesh size is reduced. However, limited computational resources can become a primary obstacle for highly accurate solutions especially when the solution domain is too large. In many problems, a high resolution mesh is only really necessary for a small fraction of the computational domain. In numerical models of the shallow water equations (where the water depth is very small compare to wave length scale) using fixed resolution of grids is not a very smart choice if waves occupy only a small portion of the domain which varies as the waves propagate. Another interesting example is the modelling of problem such as formation of KH instabilities for a stratified shear layer in which case all the action occurs in a neighbourhood of the shear layer which evolves with time. Hence it becomes essential to take special care in a particular region. Using finer grid will help to get accurate numerical results on that specific regions. Unfortunately, this sort of simulation will

be expensive in terms of computational time and memory. This problem may arise even for a simple wave propagation problem in a long domain.

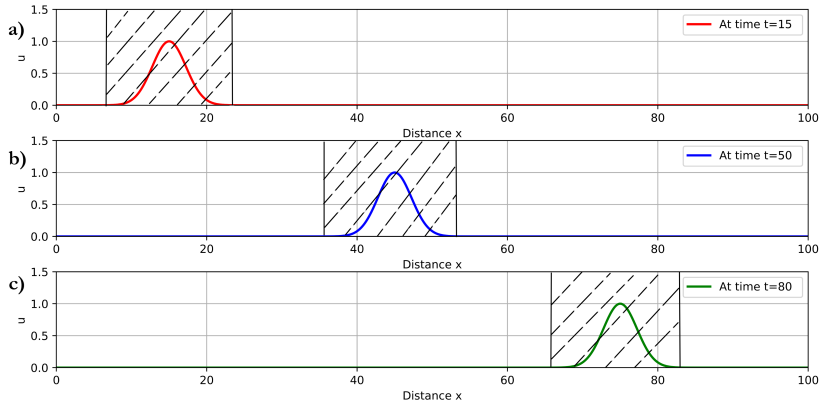


Figure 1.1: Solution of advection equation at time (a) t=15 (b) t=50 (c) t=80.

In figure 1.1 the propagation of a wave is shown at different times. This is the analytic solution of the advection equation. It can be seen that in the long domain only small portion is occupied by the wave. The solution at time $t = 15$ is shown in figure 1.1(a). In figure 1.1(a) large number of grid points are not necessary from $x = 25$ to $x = 100$. Similarly figure 1.1(b) a fine mesh is not necessary for $x < 37$ and $x > 55$. Use of a fine mesh in these regions will unnecessarily increase the cost of the computation. So, high resolution over the whole solution domain is not a smart choice for these cases. So, one available way is to change the mesh size. But in case of a uniform mesh the solution in the whole domain will be affected by the change of the grid size. In figure 1.2 solution curve is plotted at any arbitrary time. Three solution curves have plotted with different numbers of grid points and the curev with 1600 points produces an accurate solution curve. On the other hand the solutions obtained using a lower number of grid points fail to produce a certain number of characteristics of the solution.

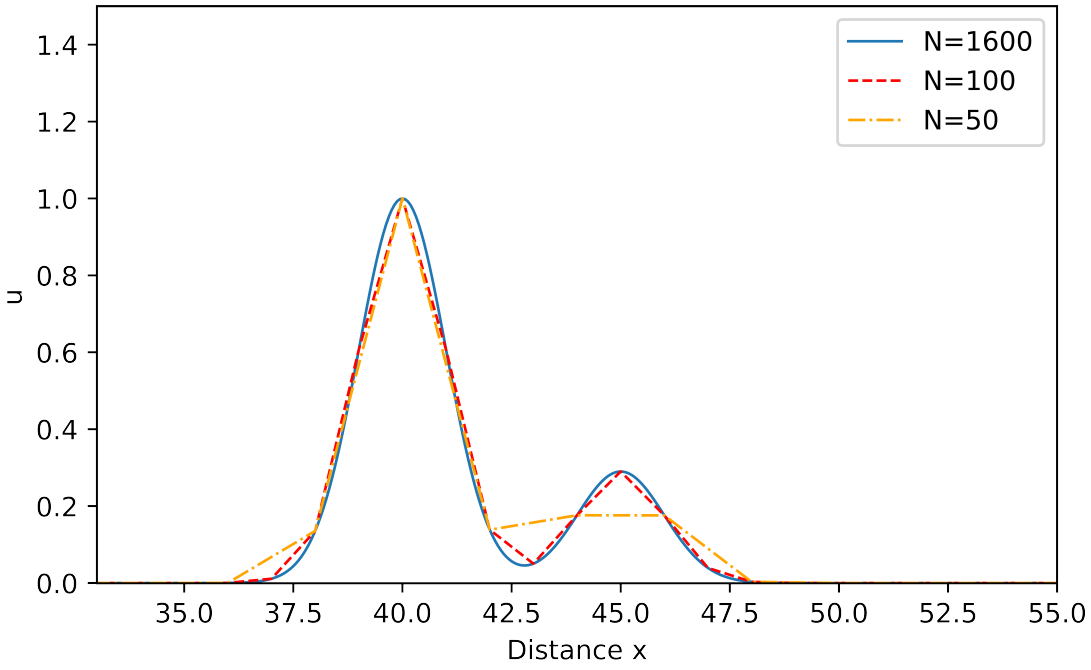


Figure 1.2: Solution with different number of grid points.

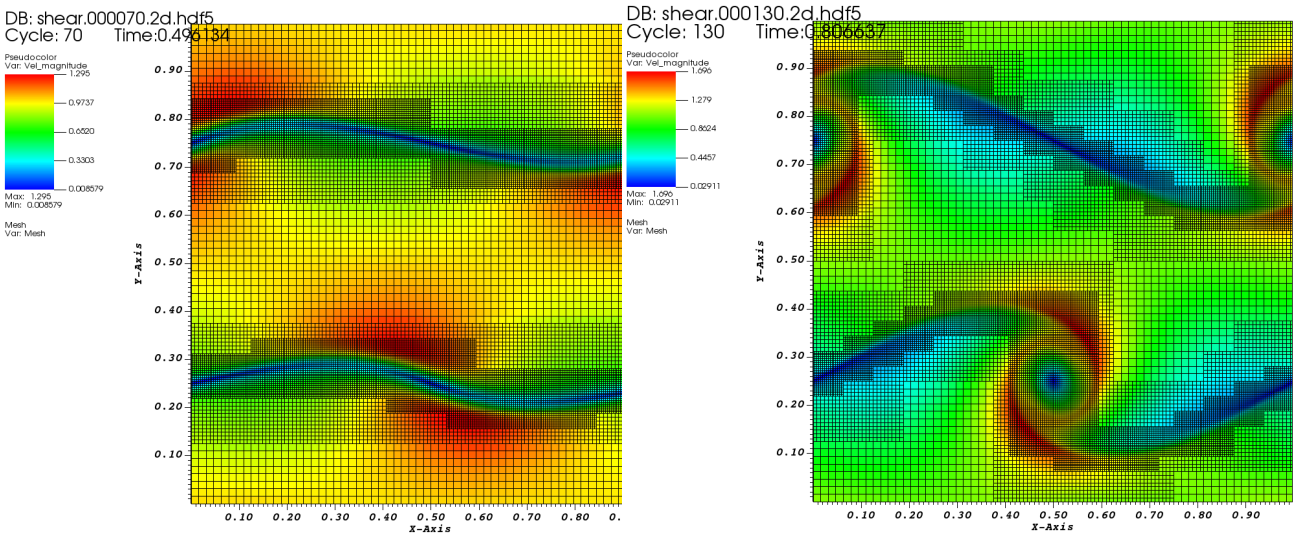
So, in both cases of figure 1.1 and 1.2 we can see using finest resolution over the whole domain is a waste of computational resources. For the implementation of a higher-order scheme even for a simple problem these issues become a primary focus to be taken care of. Adaptive mesh generation algorithms have become a popular way to deal with the unnecessary use of computational resources that can arise when using a fixed grid. The primary objective of the adaptive algorithm for the numerical methods is the optimization of computational effort. In terms of Finite Element terminology, the adaptive procedure can be classified as three types. The first is h-type, where grids are refined or coarsened on the region where more or less resolution is needed. It can be a good choice for the convergence of the problems containing variations on small length scales. If the order of the accuracy is varied in different regions of the computational domain then it is called p-type adaption. It increases the order of the polynomial approximation of the solution locally. In some problem, the combination of p-type and h-type (known as hp-refinement) are used to increase the efficiency. Finally, r-type is when a mesh is moved to follow with dynamic phenomena [14]. So, r-type refinement is more suitable for some time-dependent problems where the grids need to move or relocate to cover some changing characteristics of the solution. Mesh moving and refinement are designed to capture special situations such

as shock waves, boundary layer, shear layer or steep front. Different adaptive algorithms have been used to solve one-dimensional boundary value problems. Most of the algorithms involved some similar components such as solving in a fixed grid, re-grid, solve, averaging etc. At the initial stage of the development of adaptive mesh generation some one dimensional problems were used to build the algorithm. Harten & Hyman (1983) [20] presented a simple algorithm for grid adjustment for the solution of hyperbolic conservation laws. The average of the local characteristic velocities with respect to the amplitude of the signal was taken to determine the grid motion which was an alternative form of Godunov's method. Godunov's and Roe's scheme for Riemannian problem were compared in both fixed and self-adjusting mesh. In this approach the number of grid points was fixed and the grids were moving during the computation. It was found that for both problems the scheme is CFL dependent (restricted by the CFL condition) on a fixed grid. On the other hand, the self-adjusting grid overcomes this issue by adjusting the CFL efficiently. Sanj-Serna & Christie (1986) [33] proposed an alternative adaptive algorithm for solving nonlinear wave problems. This work uses a modification of the algorithm of White [37]. The original procedure is to transform the variable in arclength-like coordinate which introduce a coupling of the spatial variable x and solution u. The modified algorithm of [33] from that of White where the variables are not coupled since they did not use the variable transformation and then grids are created for the adaptive solution of the nonlinear PDE. This work proposed a decoupled procedure for solving the nonlinear Schrödinger equation where the primary focus of the algorithm was to compute a solution that is close to the theoretical solution. Some dedicated work on mesh moving was done by several authors. Huang *et al.*(1994) [23] developed several functions using an equidistribution principle for an Adaptive Mesh Moving procedure which can deal with one-dimensional problems successfully. Huang & Russell (1998) [24] developed a mesh moving strategy for the higher dimensional case as an extension of the previous work of Huang *et al.* (1994) [23]. In their work, they developed their moving mesh algorithm based on the gradient flow equation. Some moving mesh PDE functions were derived to solve the interaction of oblique shock problems, boundary layer problems and moving oblique shock problems. Wouwer *et al.* (2005) [39] implemented the Methods of Line (MOL) in a moving mesh algorithm for an upwind finite differencing scheme and developed a MATLAB library to solve PDE systems using MOL. Arney & Flaherty (1990) [3] developed an adaptive technique for a two-dimensional time-dependent problem which was a combination of mesh moving and local mesh refinement. A local discretization error tracing procedure was used instead of a moving mesh function in the mesh moving algorithm and Richardson's extrapolation method was used to find discretization error which was considered as refinement criteria in the local mesh refinement algorithm. Biswas *et al.* (1993) [14] followed the same procedure for a one-dimensional case such as the Sod shock tube problem which is a popular benchmark problem in gas dynamics.

It was demonstrated that mesh refinement with or without a moving mesh can produces reliable results. Use of a moving mesh method reduce the cost of the computation hence this is a reliable numerical approach. However, implementation of a moving mesh method in higher dimensions is difficult due to its complexity.

The moving mesh algorithm becomes very complicated for two or three dimensional problems. So, Structured Adaptive Mesh Refinement(AMR) algorithms have become popular alternative to the mesh moving algorithm. This algorithm starts with a base coarse grid. Regions requiring finer grid are determined by some refinement criteria. The addition of finer subgrids is a recursive procedure which will continue until a maximum level is reached. Dwyer *et al.* (1980) [16] came up with the idea of AMR for some fluid mechanics and heat transfer problems. Finer grids were placed into the solution domain according to the gradient of the dependent variable. It was demonstrated for both steady and unsteady problems that the solutions were accurate enough but it also failed to specify the error control. However, it was a significant method for the earlier stage of the AMR algorithm development. Berger & Oliger (1984) [13] proposed a standard methodology for mesh refinement. The principal idea was to introduce a recursive procedure where nested grids will be created adaptively in time to reduce the work and achieve more accurate results. A multi-level data structure was used to implement different levels of refinement and the ratio of the time and space steps were kept constant. This procedure used a special kind of multi-component data structure which plays a very important role for the algorithm. This algorithm involves the local truncation error as the basis of the refinement and according to the local truncation error clusters of points are created for the refined solution. Berger & Jameson (1985) [11] introduced another algorithm which was slight extension of their previous work. This time the idea was to determine the accuracy level in the coarse level grid and place a finer grid patch where the solution is not accurate enough. The solutions were computed using a special steady-state integrator which can be modified by the user. The overall algorithm worked nicely for the steady-state transonic flow. It is noticeable that in AMR algorithm data structure and grid clustering play a vital role. Berger (1986) [9] published a paper on the idea of the data structure used in the algorithm of adaptive mesh refinement. In this work she gave a wider view of different aspects of data structures for AMR such as sorting or the manipulation of data. In 1991 Berger & Rigoutsos published a paper on point clustering and grid generation. Berger & Colella (1989) [10] update the previous algorithm of Berger & Oliger (1984) [13] using a more sophisticated data structure to solve time dependent shock hydrodynamics problems. In this work the main idea of mesh refinement was the same but several things were changed to make the algorithm more efficient. The primary concern was to maintain global conservation to compute time dependent flows with shocks. This work is very difficult due to

the implementation of the data structures. Berger and LeVeque (1998) [12] applied the AMR algorithm in a high-resolution wave propagation algorithm in a general framework. The significant point of this work was the removal of the conservation property restriction. This work was implemented in the AMRCLAW software package. Fraga & Morris (1992) [19] proposed a very simple AMR algorithm for nonlinear dispersive wave equations. They refined the mesh based on different cut-off values of the solution. The Korteweg-de Vries equation and nonlinear Schrödinger (NLS) equations were used to demonstrate the result for the soliton solution. The AMR algorithm was faster and produces an accurate solution with minimal stability effect. Zhang *et al.* (2012) [40] implemented a higher order Runge-Kutta method in the AMR algorithm for some simple problems. In 1967 Chorin [15] introduced projection method for solving time-dependent viscous flow problem. Several authors such as [5],[6],[28],[1] developed AMR algorithm for solving fluid flow problem using the projection algorithm.

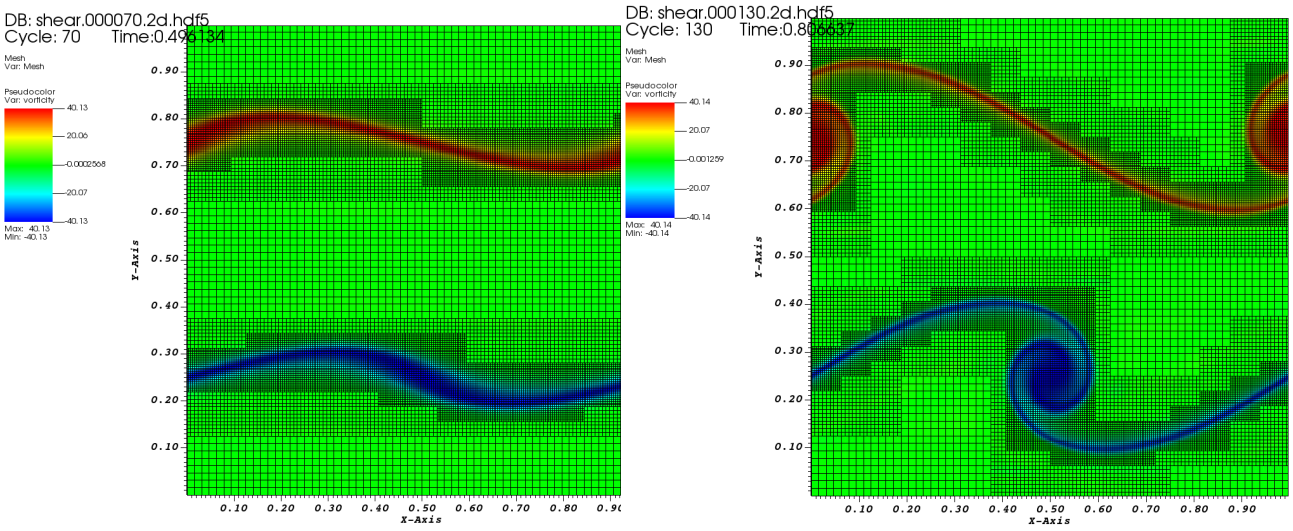


(a) (b)

Figure 1.3: Change of velocity magnitude contour and mesh with time in double shear layer simulation using CHOMBO AMRINS code

The pressure constraint of the Navier-Stokes equation is smoothly managed by the projection method. So, this method is used in some advanced work on AMR especially for the Navier-Stokes simulation problem. The important works of Berger and Oliger and Berger and Colella were used in different AMR software library such as CHOMBO, Boxlib

etc. Combinations of these works along with the projection algorithm CHOMBO AMRINS code was introduced by LBNL to deal with different incompressible Navier-Stokes problems such as vortex simulation, shear layer problems etc. In figure 1.3 and 1.4 results are shown from a simulation using CHOMBO AMRINS code.



(a) (b)

Figure 1.4: Vorticity formulation contour and change of mesh with time in double shear layer simulation using CHOMBO AMRINS code

The velocity magnitude and vorticity fields of a shear layer problem are shown in figure 1.3 and 1.4 respectively. Figure 1.3(a) and 1.3(b) represent solution at time $t = 0.496134$ and $t = 0.806637$ respectively. Three different grids can be seen. It is evident that the grid size is changing with time. Similar situations for vorticity fields are shown by figure 1.4(a) at $t = 0.496134$ and figure 1.4(b) at $t = 0.806637$. The simulation was done by CHOMBO AMRINS code and the maximum level of refinement for this simulation is two. These figures illustrate how the AMR algorithm changes the grid resolution over the whole domain with time. With the advancement of time the finer grid is changing its position. The mesh becomes finer where the computation requires finer grid. In the region where the velocity magnitude is changing (such as blue, red, yellow region) the mesh becomes finer and in the other parts of the domain lower resolution is used to reduce computational effort.

The existing codes and algorithms are efficient. However, the existing codes have a lot

of abstraction which can be very difficult to modify. In this thesis a simplified version of an AMR algorithm will be used solve one-dimensional nonlinear and dispersive wave equations numerically. The simplified version of our AMR algorithm is used to numerically solve three important wave equations. We developed the code from the scratch with the help of some advanced features of C++ programming language. Abstract classes or methods were avoided for the implementation of the algorithm. This work is inspired by [10],[13],[19],[14].

1.1.1 Simple advection equation

It is very important to determine the accuracy of any numerical algorithm. So, it is a very good idea to apply the algorithm to a very simple model. The advection equation

$$u_t + cu_x = 0 \tag{1.1}$$

is used for this purpose. In (1.1) c is the advection velocity. The study of advection equation is very important for different numerical scheme for its simple nature and easy implementation. For example, Molenkamp (1968) [29] compared different finite difference solutions of the advection equation with the analytic solution.

1.1.2 Burger's equation

The one-dimensional viscous Burger's equation is

$$u_t + uu_x = \nu u_{xx} \tag{1.2}$$

here ν is viscosity. If ν=0 the equation will become

$$u_t + uu_x = 0 \tag{1.3}$$

and (1.3) is called inviscid Burger's equation. In the study of fluid flow modelling the Navier-Stokes equation is the key equation and Burgers' equation is a simplified form of the complex and sophisticated form of Navier-Stokes equations. Burger introduce this equation for one dimension with a hope to contribute in the study of turbulence. This equation has a wide range of application including the field of gas dynamics, turbulent flow in a channel or the traffic flow problem. The physical application is not the only reason for the interest Burger's equation. Since this equation is combination of convection and diffusion transport mechanisms the numerical treatment for the solution of Burger's equation is widely used for the study of the behavior and comparison of different numerical

schemes. Different researchers have used different numerical schemes for the solution of Burger's equation. Schofield & Hammerton (2014) [34] showed the asymptotic behavior of nonlinear Burger's equation. The prediction of different asymptotic shock structures were compared with numerical schemes. Zhanlav *et al.* (2015) [41] proposed higher order finite-difference methods for the numerical solution of Burger's equation for highly accurate solutions. The prime objective of the different numerical schemes is to capture phenomena such as shocks with high accuracy.

1.1.3 Regularized Long Wave (RLW) equation

In 1844 naval architect John Scott Russell made a wonderful observation while he was following a wave formed under the middle of the boat and which then took off past the boat entirely. He noticed that the wave was propagating without changing its shape. He named it "the wave of translation" which is known as solitary wave today. However the theory of solitary waves was not established at that time. It was proved that solitary waves can be theoretically possible by the independent research of Boussinesq and Lord Rayleigh [26]. In 1895 Korteweg and Varies made some significant extension of Boussinesq theory and introduced the solitary water wave equation named Kortweg-de Varies(KdV) equation [27]. The KdV equation is

$$u_t + u_x + uu_x + u_{xxx} = 0 \tag{1.4}$$

It is noticeable that in the KdV equation dissipiation is absent which allows the solitary wave to propagate a long distance without changing shape. This equation has some other properties mentioned in [25]. The properties are given below.

- Solitary waves can travel very large distance without changing their shape, their amplitude remain same during the propagation. Solitary waves are stable and do not break.
- The propagation speed of the solitary wave increases with its amplitude.
- Solitary waves do not obey the superposition principle.

This equation was derived for the propagation of shallow water waves. However, it has some other fields of application such as magnetohydrodynamic waves in plasma, an harmonic lattice, longitudinal dispersive waves in elastic rods etc. [17]. Benjamin, Bona and Mahony (1972) proposed an alternative to the KdV equation named the Regularized Long Wave (RLW) equation. The RLW equation is

$$u_t + u_x + uu_x - u_{txx} = 0 \tag{1.5}$$

with the the physical boundary condition $u \to 0$ at $x \to \pm\infty$. Equation (1.5) is also known as BBM equation. In 1966 Pregrine also derived equation (1.5) and used it to study the development of undular bores (in water). For the wave motion he used same order of approximation as the KdV equation [17]. For infinitesimally small waves the KdV equation reduces to the linearized KdV equation

$$u_t + u_x + u_{xxx} = 0$$

which has dispersion relation $\omega = k - k^3 = k(1-k^2)$. The phase speed c and group velocity c_g are

$$c = \frac{\omega}{k} = 1 - k^2$$

$$c_g = \frac{d\omega}{dk} = 1 - 3k^2$$

Both c and c_g go to $-\infty$ like k^2 as $k \to \infty$. This behaviour is completely unphysical for systems that are modelled with the KdV equation. Numerically it causes considerable difficulty because as the grid spacing Δx is reduced the phase speed and group velocity of the shortest resolved waves goes to infinity like $\frac{1}{\Delta x^2}$. The CFL condition then requires the time step to go to zero like Δx^3. Thus the number of time steps is proportional to N^3 where N is the number of grid points. The linearized RLW equation is

$$u_t + u_x - u_{xxt} = 0$$

with dispersion relation $\omega = \frac{k}{1+k^2}$.The phase speed c and group velocity c_g are

$$c = \frac{\omega}{k} = \frac{1}{1+k^2}$$

$$c_g = \frac{d\omega}{dk} = \frac{1-k^2}{(1+k^2)^2}$$

These are now bounded.The phase speed c has a maximum value of 1 and decreases monotonically to 0 as $k \to \infty$. The group velocity has a maximum value of 1. It decreases to 0 at $k = 1$, has a minimum value of -0.125 before increasing, going to zero like $-\frac{1}{k^2}$ as $k \to \infty$. The CFL condition now gives a time step proportional to N. As the KdV equation is a model for long waves it is only valid for small k in which case the KdV and RLW equations have the same behaviour. Benjamin *et al.* (1972) [7] discussed some problems with the KdV equation and argued that the RLW is equivalent to the KdV equation. However, the RLW equation has the same wide range of application as the KdV equation. The numerical solution of KdV equation is much more difficult to achieve since if the resolution is

increased the time step must rapidly decreased. The RLW equation overcomes this problem. Moreover the dispersion relation of RLW equation is more physical than that of the KdV equation. Many solutions of the KdV equation reported in the literature are actually solutions of the RLW equation. The KdV and RLW equation share the same properties for long waves hence for long waves their solutions are very similar. The stability of the solution for different initial conditions and forcings was derived by Benjamin *et al.* [7]. In this thesis we choose RLW equation for the simulation of dispersive equation.

1.2 Thesis overview

The linear, nonlinear and the dispersive wave equations that are considered in this thesis introduced in this chapter. These equations will be used for the simulation using our algorithm. The remainder of this thesis is divided into two major components. In the first portion of chapter 2 numerical methods, data structures, boundary conditions and other requirements for developing AMR algorithm are presented. In the last portion the details of AMR algorithm is discussed. A brief overview is represented. The discretization of different wave equations are given in the same chapter. The AMR code we developed is a simplified version. This code used for the simulations of the equations discussed in chapter 1. The results of the numerical simulation of different wave equations are presented in chapter 3. Comparisons using uniform and AMR grids are discussed which demonstrated the efficiency of the current algorithm and code.

Finally, different aspects of AMR algorithm and the future extensions of the current work are discussed at the end of the thesis.

Chapter 2

Methodology

Different computational tools and components are involved in the development process of AMR algorithms for different systems. **Block Structured AMR** is chosen for this thesis. The work of Berger & Oliger (1984) [13] and Berger & Colella (1989) [10] will be discussed as the basic building block of this current research work. We will use the idea of AMR from these significant works but we will use a different implementation of the algorithm. Berger & Oliger (1984) [13] and Berger & Colella (1989)[10] implemented their algorithm in Fortran. However, modern AMR toolboxes are not written only in Fortran. C++ is a modern high level language with a lot of features which are very helpful for AMR computation. Some existing software framework for solving PDE with AMR such as CHOMBO, GERRIS, BOXLIB use C++ for its advanced features. The LBNL used mixed source programming language with C++ and Fortran for CHOMBO. We have used only C++ for the developement of our AMR code. The ancillary components such as discretization scheme, grid description along with AMR algorithm will be discussed in this chapter.

2.1 Discretization scheme

Discretization is a process by which a closed-form mathematical expression such as a function or differential equation or integral equation involving functions, all having a continuum of values throughout some domain, is approximated by an analogous (but different) expression which prescribe values at a finite number of discrete points or volumes in the domain. In other words, the analytic solution of a PDE expresses the variation of the dependent variables continuously throughout the domain where numerical solutions do that only for

a number of discrete points known as grid points [2]. In CFD computation three widely employed discretization techniques are used. They are known as finite volume, finite difference and finite element method. In this thesis the finite difference and the finite volume method will be used.

2.1.1 Finite Difference Method

The Finite Difference Method (FDM) is one of oldest and most straightforward discretization techniques in the literature of CFD. The application of FDM is very simple for the uniform mesh problem. FDM uses the basic properties of the Taylor series expansion. The key idea is to approximate the partial derivatives of the governing equation by the ratio of algebraic differences at two or more grid points. There are several methods available for the derivation of finite difference approximation. However, for the simplicity we will use the Taylor series expansion for the derivation of the finite difference expression. Let $f(x)$ be a function. Then $f(x + \Delta x)$ can be expanded in a Taylor series about x as,

$$f(x + \Delta x) = f(x) + \Delta x \frac{\partial f}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 f}{\partial x^2} + \frac{(\Delta x)^3}{3!} \frac{\partial^3 f}{\partial x^3} + \dots$$

Solving for $\frac{\partial f}{\partial x}$ we get

$$\frac{\partial f}{\partial x} = \frac{f(x + \Delta x) - f(x)}{\Delta x} - \frac{\Delta x}{2!} \frac{\partial^2 f}{\partial x^2} - \frac{(\Delta x)^2}{3!} \frac{\partial^3 f}{\partial x^3} + \dots$$

summing all the term having Δx and higher and expressing them as $O(\Delta x)$ gives

$$\frac{\partial f}{\partial x} = \frac{f(x + \Delta x) - f(x)}{\Delta x} + O(\Delta x)$$

This is a first order approximation of the partial derivative of f with respect to x. This is the forward difference approximation. Similarly, using $f(x - \Delta x)$ and $f(x)$ we can get the backward approximation. Now consider $f(x + \Delta x)$ and $f(x + 2\Delta x)$ as

$$f(x + \Delta x) = f(x) + \Delta x \frac{\partial f}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 f}{\partial x^2} + \frac{(\Delta x)^3}{3!} \frac{\partial^3 f}{\partial x^3} + \dots \tag{2.1}$$

and

$$f(x + 2\Delta x) = f(x) + (2\Delta x) \frac{\partial f}{\partial x} + \frac{(2\Delta x)^2}{2!} \frac{\partial^2 f}{\partial x^2} + \frac{(2\Delta x)^3}{3!} \frac{\partial^3 f}{\partial x^3} + \dots \tag{2.2}$$

Multiply equation (2.1) by 2 and subtract equation (2.2) then rearrange the results we get

$$\frac{\partial^2 f}{\partial x^2} = \frac{f(x+2\Delta x) - 2f(x+\Delta x) + f(x)}{(\Delta x)^2} + O(\Delta x)$$

Similarly the other expression can be derived which we applied for the discretization of the RLW equation.

2.1.2 Finite Volume Method

In the study of numerical solutions of conservation laws one of the popular and widely used schemes is the finite volume method. It has an extensive use in fluid flow modelling problems or for system of partial differential equations where a balance of one or more quantities is involved. The finite volume method uses the integral form of a conservation law. The principal idea of this scheme is to introduce the idea of control volume for each computational grid. The computational domain is divided into a finite number of grid points called nodes in the finite volume sense. The control volume will be set up in a way that the node can be considered as the center of the control volume.

In the figure 2.1 the control volume set up in one-dimension is shown based on the control volume concept of [36]. Here 5 nodal points between A and B, which are considered as the physical boundaries of the solution domain are shown. If P is any present node then the left neighbouring node is W while E is right neighbouring node. The boundaries of the control volume of the nodal point P are positioned mid-way between the neighbouring nodes. The edge of the domain is positioned such that physical boundaries coincide with it. A general one-dimensional conservation law is

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} = \mathbf{S} \tag{2.3}$$

where $\mathbf{f}$ is the component of the flux vector and $\mathbf{S}$ represents a source term. Now we can introduce a finite volume mesh set up for equation (2.3) by assigning a finite cell for each of the node or mesh point. As mentioned before the 'cell faces' will be taken as the mid points between two adjacent nodes. In figure 2.2 node i has the 'cell faces' at the mid points between two neighbouring node. So, $(i - \frac{1}{2})$ is the left face and $(i + \frac{1}{2})$ is the right face for node i. The conservation equation (2.3) can be discretized via

$$\frac{\partial u_i}{\partial t} + \frac{f_{i+\frac{1}{2}} - f_{i-\frac{1}{2}}}{\Delta x} = S_i \tag{2.4}$$

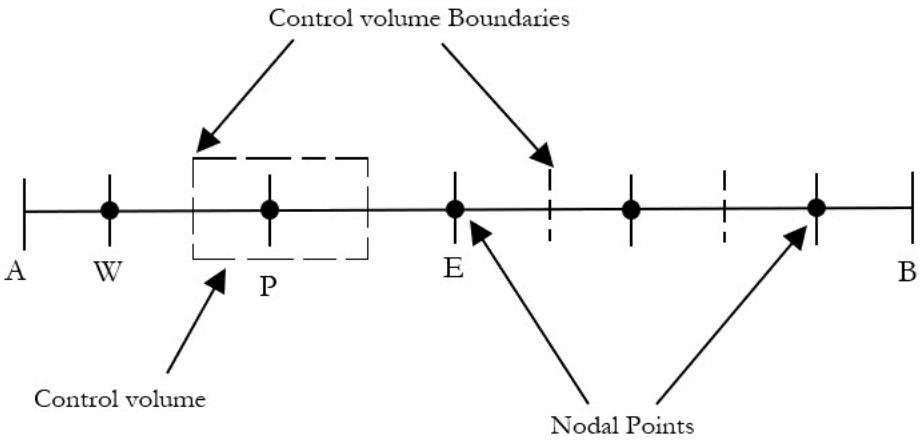


Figure 2.1: Control volume

assuming that the cell size is constant that is $\Delta x_i = \Delta x_{i+\frac{1}{2}} = \Delta x_{i-\frac{1}{2}} = \Delta x$. Different schemes will be used for the approximation of the flux $f_{i+\frac{1}{2}}$ and $f_{i-\frac{1}{2}}$. Next this finite volume scheme will be used for the discretization of the advection equation and the Burger's equation.

2.2 Finite volume discretization of advection equation

The advection equation is

$$u_t + cu_x = 0 \tag{2.5}$$

If we integrate (2.5) from $x_{i-\frac{1}{2}}$ to $x_{i+\frac{1}{2}}$ we will get

$$\frac{d}{dt}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} u dx + c[u]_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} = 0 \tag{2.6}$$

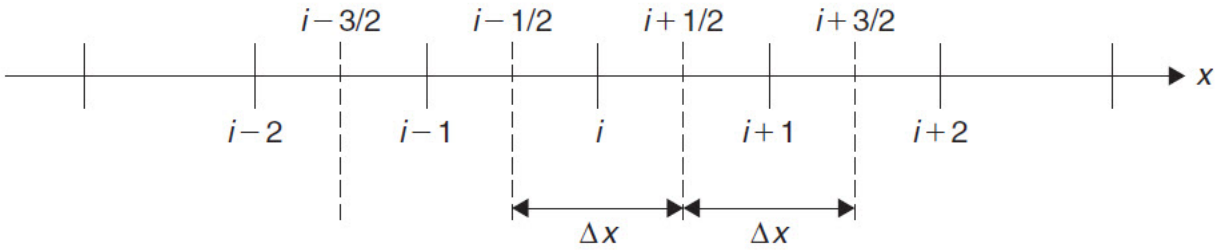


Figure 2.2: Finite volume grid setup

Now define the average value of u in control volume i as

$$u_i = \frac{1}{\Delta x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} u dx$$

Here $\Delta x = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$.
So equation (2.6) can be written as

$$\Delta x \frac{du_i}{dt} + c[u(x_{i+\frac{1}{2}}) - u(x_{i-\frac{1}{2}})] = 0 \tag{2.7}$$

For simple implementation we are going to use first-order time stepping and get

$$\Delta x \frac{u_i^{n+1} - u_i^n}{dt} + c[u(x_{i+\frac{1}{2}}) - u(x_{i-\frac{1}{2}})] = 0 \tag{2.8}$$

Finally the full discrete form can be written as

$$u_i^{n+1} = u_i^n - (\frac{c\Delta t}{\Delta x})[u(x_{i+\frac{1}{2}}) - u(x_{i-\frac{1}{2}})] \tag{2.9}$$

Using the first order upwind scheme we have

$$u(x_{i+\frac{1}{2}}) = \begin{cases} u_i^n & if\ c > 0 \\ u_{i+1}^n & if\ c < 0 \end{cases} \tag{2.10}$$

$$u(x_{i-\frac{1}{2}}) = \begin{cases} u_{i+1}^n & if\ c > 0 \\ u_i^n & if\ c < 0 \end{cases} \tag{2.11}$$

Here $(\frac{c\Delta t}{\Delta x})$ is called Courant-Friedrichs-Lewy (CFL) number which is very important for the stability condition of the advection equation. There are many higher-order schemes available for time discretization but for AMR this is the simplest to implement.

2.3 Finite volume discretaization of Burger's Equation

We can write Burger's equation (1.2) as

$$u_t + [f(u)]_x = \nu u_{xx} \tag{2.12}$$

where $f(u) = \frac{1}{2}\, u^2$. If we integrate (2.12) from $x_{i-\frac{1}{2}}$ to $x_{i+\frac{1}{2}}$ we will get

$$\frac{d}{dt}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} u dx + [f(u)]_{x_i-\frac{1}{2}}^{x_{i+\frac{1}{2}}} = \nu[u_x]_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \tag{2.13}$$

Again define the average value of u in control volume i as

$$u_i = \frac{1}{\Delta x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} u dx$$

Here $\Delta x = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$.
The viscous term can be written as

$$\nu[u_x]_{x_i-\frac{1}{2}}^{x_i+\frac{1}{2}} = \nu[u_x(x_{i+\frac{1}{2}}, t) - u_x(x_{i-\frac{1}{2}}, t)] \approx \nu[\frac{u(x_{i+1}, t) - u(x_i, t)}{\Delta x} - \frac{u(x_i, t) - u(x_{i-1}, t)}{\Delta x}] \tag{2.14}$$

or

$$\nu[u_x]_{x_i-\frac{1}{2}}^{x_i+\frac{1}{2}} \approx \nu[\frac{u(x_{i+1}, t) - 2u(x_i, t) + u(x_{i-1}, t)}{\Delta x}] \tag{2.15}$$

Next,

$$[f(u)]_{x_i-\frac{1}{2}}^{x_i+\frac{1}{2}} = f(u(x_{i+\frac{1}{2}}, t)) - f(u(x_{i-\frac{1}{2}}, t)) \tag{2.16}$$

Finally, if we put all together and divided by Δx we obtain a system of ordinary differential equation.

$$\frac{du_i}{dt} + \frac{f(u_{i+\frac{1}{2}}) - f(u_{i-\frac{1}{2}})}{\Delta x} = \nu[\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}] \tag{2.17}$$

If a first-order forward difference formulation is used for the time derivative for simplicity we have the final explicit discretization of Burger's equation is

$$u_i^{n+1} = u_i^n - \Delta t\Big(\frac{f(u_{i+\frac{1}{2}}^n) - f(u_{i-\frac{1}{2}}^n)}{\Delta x} - \frac{\nu}{\Delta x^2}[u_{i+1}^n - 2u_i^n + u_{i-1}^n]\Big) \tag{2.18}$$

Equation (2.18) will be used for both coarse and fine grid integration. For simplicity we can estimate the cell edges as

$$u(x_{i+\frac{1}{2}}, t) = \frac{u_i^n + u_{i+1}^n}{2} \tag{2.19}$$

and

$$u(x_{i-\frac{1}{2}}, t) = \frac{u_{i-1}^n + u_i^n}{2} \tag{2.20}$$

We can define the dimensionless parameter Péclet number Pe as the relative measurement of convection and diffusion strength as

$$Pe = \frac{Lu}{D} \tag{2.21}$$

Where L is characteristic length and D is diffusion coefficient. For each cell the Péclet number for equation 2.12 is

$$Pe = \frac{u}{\nu/\Delta x}$$

If the ν is very small (weak diffusion problem) then the convection will be strong and for large Δx the solution using (2.19) and (2.20) might produce some oscillations where the $Pe > 2$. An upwind scheme can resolve this issue nicely. If we consider

$$a = \frac{u_i^n + u_{i+1}^n}{2} \tag{2.22}$$

and

$$b = \frac{u_{i-1}^n + u_i^n}{2} \tag{2.23}$$

then we have by first order upwind scheme for the values at the cell edges as

$$u(x_{i+\frac{1}{2}}) = \begin{cases} u_i^n & if\ a > 0 \\ u_{i+1}^n & if\ a < 0 \end{cases} \tag{2.24}$$

and

$$u(x_{i-\frac{1}{2}}) = \begin{cases} u_{i-1}^n & if\ b > 0 \\ u_i^n & if\ b < 0 \end{cases} \tag{2.25}$$

However, in this thesis the simplest estimation using (2.19) and (2.20) will be used to show how AMR algorithm can resolve the problems arising from numerical scheme.

2.4 Finite difference discretization of RLW-equation

The regularized Long Wave (RLW) equation is

$$u_t + u_x + uu_x - u_{xxt} = 0 \tag{2.26}$$

which can be written as

$$(u - u_{xx})_t + (1 + u)u_x = 0 \tag{2.27}$$

Finite difference technique will be used to discretized equation (2.27). We chose finite difference scheme for two reasons. The first reason is we want to demonstrate our code in a simple scheme. The second reason is we want to use an existing scheme with detail stability analysis. We followed the scheme proposed by [17] where the stability analysis of the scheme have described briefly. If the first and second spatial derivatives and the time derivative at time step n are denoted by ${D_x u_i}^n$, ${D_x}^2 {u_i}^n$ and $\Delta_t {u_i}^n$ respectively. These differencing operator can be defined as

$${D_x}^2 {u_i}^n = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{dx^2}$$

$${D_x u_i}^n = \frac{u_{i+1}^n - u_{i-1}^n}{2dx}$$

and finally

$$\Delta_t {u_i}^n = \frac{u_i^{n+1} - u_i^n}{dt}$$

So equation (2.27) can be written as

$$\Delta_t (1 - {D_x}^2){u_i}^n + (1 + {u_i}^n){D_x u_i}^n = 0 \tag{2.28}$$

$$\implies \Delta_t {u_i}^n - \Delta_t {D_x}^2 {u_i}^n = -(1 + {u_i}^n){D_x u_i}^n$$

which can be written as

$$\frac{u_i^{n+1} - u_i^n}{dt} - \Delta_t\left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{dx^2}\right) = -(1 + {u_i}^n)\frac{u_{i+1}^n - u_{i-1}^n}{2dx}$$

$$\implies \frac{u_i^{n+1} - u_i^n}{dt} - \frac{1}{dt\,dx^2}(u_{i+1}^{n+1} - u_{i+1}^n - 2u_i^{n+1} + 2u_i^n + u_{i-1}^{n+1} - u_{i-1}^n) = -(1 + {u_i}^n)\frac{u_{i+1}^n - u_{i-1}^n}{2dx}$$

after some rearrangement we can write

$$\begin{aligned} -\frac{1}{dt\,dx^2}u_{i-1}^{n+1} + (\frac{1}{dt} + \frac{2}{dt\,dx^2})u_i^{n+1} - \frac{1}{dt\,dx^2}u_{i-1}^{n+1} = -\frac{1}{dt\,dx^2}u_{i-1}^{n} \\ +(\frac{1}{dt} + \frac{2}{dt\,dx^2})u_i^n - \frac{1}{dt\,dx^2}u_{i-1}^n - (1+{u_i}^n)\frac{u_{i+1}^n - u_{i-1}^n}{2dx} \end{aligned} \tag{2.29}$$

Multiplying equation(2.29) by $dt\,dx^2$ we get,

$$u_{i-1}^{n+1} - (2+dx^2)u_i^{n+1} + u_{i+1}^{n+1} = u_{i-1}^n - (2+dx^2)u_i^n + u_{i+1}^n + \frac{1}{2}(1+{u_i}^n)(u_{i+1}^n - u_{i-1}^n) \tag{2.30}$$

Equation (2.30) is the final discretized version of RLW equation. This is an implicit equation and can be written as a tridiagonal system of equations.

2.5 Tridiagonal system solver

A $n \times n$ tridiagonal system can be written as

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_3 & b_3 & c_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}$$

which can also be written as

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i \tag{2.31}$$

for $i = 1...., n$ and $a_1 = 0$ & $c_n = 0$. This system can be solved using different algorithms such as Gaussian elemination or LU factorization. However, a tridiagonal system solver is the best solution method due to its computational simplicity. In this work the algorithm described in [17] is followed.

The algorithm is equivalent to the LU factorization method. However, it is a bit more efficient. In this algorithm P_i and q_i are independent of the right hand side vector d_i. So, for the single grid computation they can be computed and stored at the beginning for later use at each time step.

Algorithm 1: Tridiagonal System $Ax = b$ Solver for

Result: Solution vector $x_1, x_2, ..., x_n$;

set $P_1 = b_1$;
set $g_1 = \frac{d_1}{P_1}$;
set $q_1 = -\frac{c_1}{P_1}$;
set $i = 2$;
while $i \leq n$ **do**
    $P_i = b_i + a_i q_{i-1}$;
    $q_i = -\frac{c_i}{P_i}$;
    $g_i = \frac{d_i - a_i g_{i-1}}{P_i}$;
    $i \leftarrow i + 1$
end
set $x_n = g_n$;
set $j = n$;
while $j > 1$ **do**
    $x_j = g_{j-1} + x_j q_{j-1}$;
    $j \leftarrow j - 1$
end

2.6 AMR algorithm

In the computation of the numerical solution of **PDE**s some phenomena such as steep shocks or discontinuities are often arise which are difficult to capture numerically. So, very high resolution is required for the numerical computation. Implementation of some higher order scheme may not be convenient all the time due to extra constraints. We have seen that high resolution of a mesh over the whole domain can make the computation unnecessarily expensive. Improvements can be made by increasing the resolution on some subregions of the computational domain. This is not only reduces the cost of the computation, but also is smart memory management process. In our work we used the **Block Structured Adaptive Mesh Refinement (AMR)** method which is popular and convenient to modify. This algorithm allows the computation to change the resolution over time in different subregions (blocks) of the domain and manage the memory efficiently.

Different types of mesh refinement algorithms are available in the literature. We will focus on the **Block Structured Adaptive Mesh Refinement** approaches inspired by previous work [13],[10],[19],[14]. In this section the AMR algorithm based on the work of

Berger & Oliger (1984) [13] and Berger& Colella (1989) [10] will be discussed since these works give a core framework of the block structured AMR algorithm. These works have been extensively used to develop powerful AMR solvers. However the algorithm of Fraga & Morris (1992) [19] is also very important. This work presented a simplified implementation of one-dimensional problems with simple refinement criteria for the numerical solution.

2.6.1 Grid Description

We begin with a brief overview of the grid generation which is an essential component of the AMR algorithm. The computation of AMR will start from a user specified level 0 base grid denoted as D_0. The mesh spacing of the base grid is Δx_0. This base grid will be fixed during the computation. Successively refined grids D_1, D_2 are generated automatically in

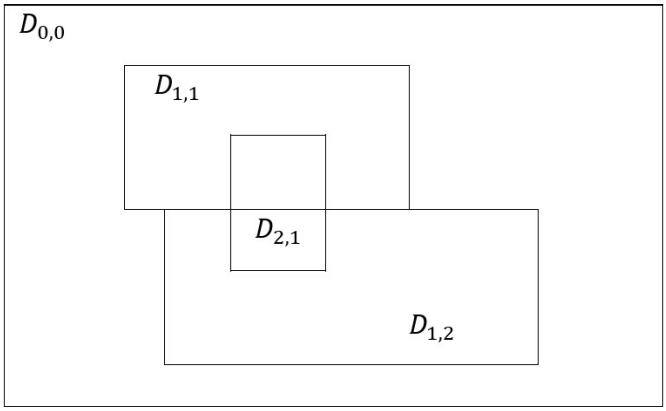


Figure 2.3: Nesting property of AMR grid in 2 dimensional space [10]

an adaptive fashion. The grid spacing at level 1 and level 2 are Δx_1 and Δx_2 respectively. In addition Δt_1 and Δt_2 are the time step size of the level 1 and level 2 grid respectively. The adaptive algorithm creates a time-varying sequence of nested and logically rectangular meshes on which the discretized **PDE** is solved. In this set up a finer subgrid may contain even finer subgrid within their boundaries since subgrid generation is a recursive procedure. Grid D_1 is the level 1 refined grid which contains the subgrids of D_0. This will be applicable for the other levels. In addition each grid level D_i can be composed of several components D_{ij}. Denoting the different levels of refinement by $l = 1, 2, 3..l_{max}$, grid $D_{l,k}$ has mesh width Δx_l. Ultimately the grid can be defined as

$$D_l = \underset{k}{\cup} D_{l,k}$$

The components may overlap that is $D_{l,j} \cap D_{l,k} \neq 0$ for $j \neq k$ though for one dimensional case they will not overlap. So, any point in this grid set up can be contained in different components of one level grids. Finally, the grid must be nested which is described in the figure 2.3 which was taken from [10]. Figure 2.3 shows a sample snapshot of an evolving grid in two spatial dimensions. Initially the base grid D_0 is at level 0 grid in the described grid hierarchy. After any time t a level 1 grid D_1 is created and in the next step the grid is further refined. D_2 is the refined grids of level 1. There is only one grid component in D_2 which is $D_{2,1}$ and every point in $D_{2,1}$ is contained in one of the level 1 grid components.

In AMR, grid refinement is needed in time and space and the **refinement ratio** should be same. The grid spacing and time step at any level l are Δx_l and Δt_l respectively. Then the **refinement ratio** r can be defined as

$$r = \frac{\Delta x_{l-1}}{\Delta x_l} = \frac{\Delta t_{l-1}}{\Delta t_l}$$

This ratio r is same for all level l and hence

$$\frac{\Delta t_l}{\Delta x_l} = \frac{\Delta t_{l-1}}{\Delta x_{l-1}} = \frac{\Delta t_{l-2}}{\Delta x_{l-2}} = ... = \frac{\Delta t_0}{\Delta x_0}$$

2.6.2 Grid Generation

For the generation of a higher level grid a refinement criterion has to be introduced. Berger & Oliger (1984) [13] and Berger & Colella (1989) [10] used error estimation to determine the region where the refinement is needed. They estimated the error at all grid points in a grid level and flagged the grid points where refinement is necessary. The new level grid is created in such a way that the flagged points will be the interior of the finer grid. So, the grid generation procedure follows a series of steps. First, based on the error estimation (the refinement criteria) the grid points at level l will be identified where the finer grid will be placed in the next level $l + 1$ and those points will be flagged. The next and most difficult step is separating the flagged points. The separated flagged points will be used to make clusters. A buffer zone is added around the flagged grid points to prevent the propagation of discontinuities or high error from the finer region to coarser region at the next regridding time. In our case we keep the 1 buffer point at the leftmost and rightmost point of a cluster. Adding a buffer zone may increase the integration cost but this cost is small because of the small number of extra grid points. If flagged points are separated by a good number unflagged points then two or more clusters at level $l + 1$ may be formed. In our code we considered that two clusters will be separated by at least 3 unrefined cells.

So, in this way two or more subgrids can be found. The number of minimum unflagged points is predefined. Berger & Oliger (1984) [13] considered flagged points which are closer together than twice the size of the buffer zone should be in same grid refinement. The work of this thesis followed the same grid generation procedure.

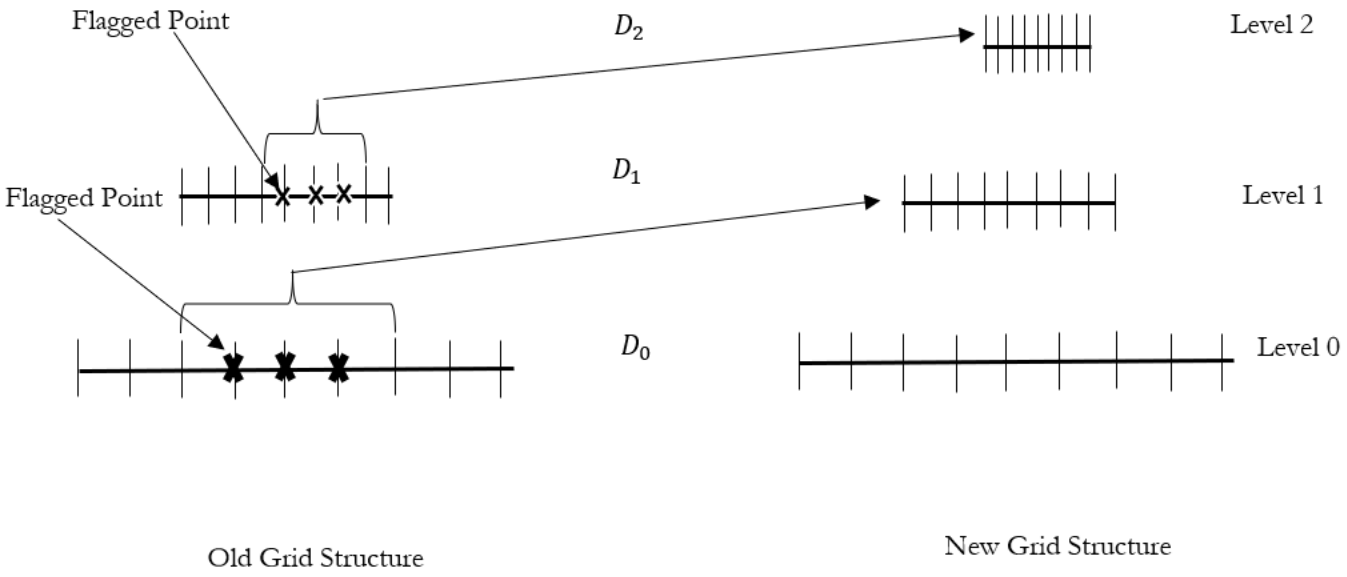


Figure 2.4: 1-D grid generation [13]

Figure 2.4 from [13] describes the regridding procedure. In the left side the old grid structure is shown and on the right side the new grid is shown. Here $\times$ indicates the flagged points. Here the grid structures are shown step-wise and Figure 2.5 shows the superimposed grid structure.

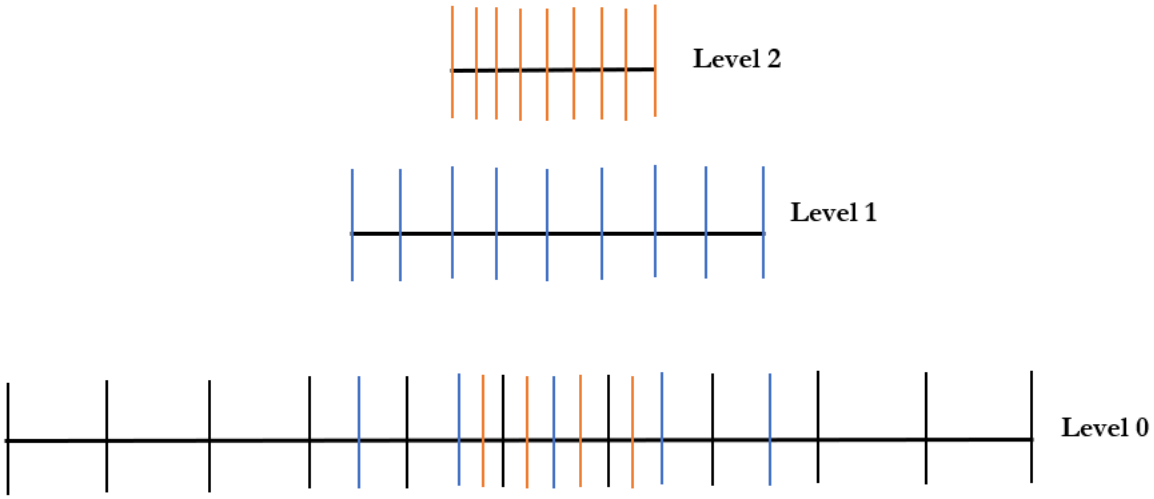


Figure 2.5: Composite grid Structure [13]

2.6.3 Refinement criteria

One of the crucial point of the AMR algorithm is to locate the regions where refinement is needed and to cover them a finer mesh which is a recursive procedure. Refinement criteria can vary for different types of problems, e.g. a refinement criterion based on gradient can be a good refinement criterion where steep shocks can be found. It also should be noted that in some cases the refinement criteria can increase the cost of the computation. However, several factors can be related to the cost estimation of this such as solution algorithm or norm of the solution error (in case where error estimation uses as refinement criteria). In the algorithm of Berger & Oliger (1984) [13] and Berger & Colella (1989) [10] error estimation was the basis of their refinement criteria. Fraga & Morris (1992) [19] used the geometrical property that is the solution cut-off value for the AMR computation for single and double soliton problems. They predefined a minimum cut-off value which helped the algorithm to find the solitons in the domain. Even in the case of a moving grid algorithm an equidistribution principle is used as a parameter where the grid will be concentrated since in a moving mesh algorithm the grid concentration changes over the domain dynam-

ically over time.
However, in this work different kinds of refinement criteria are used for different problems. First of all, for a very simple advection type problem a combination of gradient and curvature is set for locating the area where the refinement is required. So, any specific part of the computational domain having value greater than or equal to that minimum summation value of gradient and curvature will be considered for the refinement at any level. However, this changes for Burger's equation a little where steep shock formed over time. Now grid refinement must be based in part on the gradients of the solution. Different types of refinement criteria were selected for different levels. Higher level grids were put at the region of steep shock. If the gradient at level l is defined by ${G_x}^{(l)}$. If any level l has flux ${\psi_i}^{(l)}$ and ${\psi_{i+1}}^{(l)}$ at i and $i+1$ respectively then we have

$$ {G_x}^{(l)}\psi_i = \frac{{\psi_{i+1}}^{(l)} - {\psi_i}^{(l)}}{\Delta x^{(l)}} $$

For Burger's equation a combination of a cut-off value and gradient was used as the refinement criteria. Finally, for the nonlinear and dispersive equation we set several refinement criteria for two different problems. In the first level we use cut-off value for the refinement while at the next level the combination of the cut-off value, gradient and second derivative was used. In all the simulation of this work the refinement criteria are kept simple. The main goal was to demonstrate that the code worked with simple refinement criteria.

2.6.4 Data structure and implementation

The data structure used by AMR algorithms can be sophisticated and sometimes the implementation becomes very difficult for the general users. In this section the data structure for one dimensional case will be discussed. This is going to be an extension of the discussions about the grid in the grid description section where a level $l+1$ grid must be entirely contained within the level l grid which we defined as the nesting property. In the AMR algorithm of Bereger & Oliger (1984) [13] and Berger & Colella (1989) [10] a special kind of tree data structure known as right child left sibling tree data structure was used. To represent this as a tree data structure every grid is considered as a node. So the coarser parent grid will act as a parent of higher level grid. Subgrids can be considered as a child of its parent grid. Siblings are subgrids within the same grid level. Figure 2.6 taken from [13] demonstrates this structure. In this figure the bidirectional arrows indicates the connection between parent and their child grid. On the other hand the unidirectional arrows symbolises the one way connection between the siblings in the same grid level. Two levels of refinement are represented by figure 2.6. The base grid is denoted by $D_{0,1}$. The base

grid has three children which are denoted by $D_{1,1}$, $D_{1,2}$ and $D_{1,3}$ respectively and they are the subgrids of level 1 grid. Similarly, $D_{2,1}$ is the children of $D_{1,1}$ and $D_{2,2}$ is the children of $D_{1,2}$. Finally, $D_{1,1}$, $D_{1,2}$, $D_{1,3}$ are siblings of each other in the level 1 grid and $D_{2,1}$ and $D_{2,2}$ are siblings of each other in the level 2 grid.

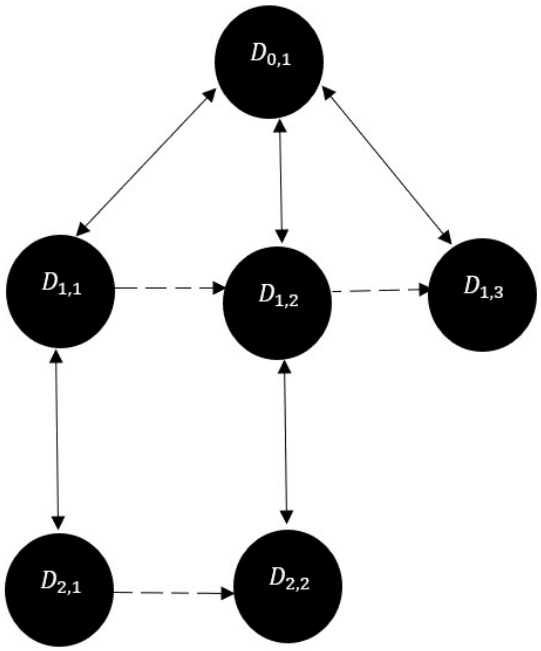


Figure 2.6: 1-D grid Data Structure

So, there is a sequential order where each node can have multiple childs. The representation clearly allows us to have multiple finer subgrids of any grid. The internal operation for the implementation of AMR such as fine grid boundary value set up, updating the value from fine to coarse grid have an information flow which follows path links in the tree. The algorithm has a one way link between subgrids at the same level which are called neighbours and this simplifies the implementation of operations. Figure 2.7 illustrates the AMR grid structure described by 2.6 for a one-dimensional grid.

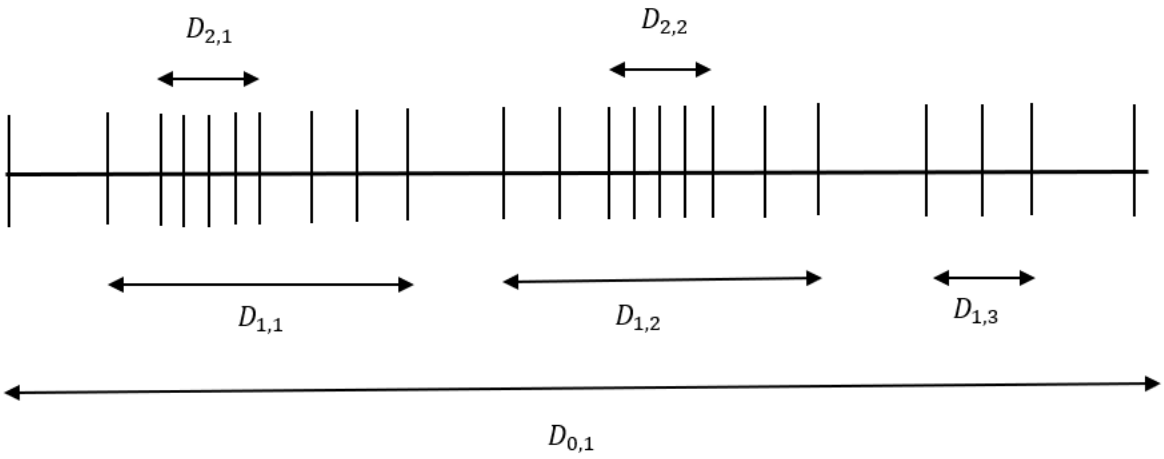


Figure 2.7: 1-D grid Data Structure (linear view) [13]

In figure (2.7) $D_{0,1}$ is the base grid. The base grid contain $D_{1,1}$, $D_{1,2}$, $D_{1,3}$ level 1 subgrids. Similarly, $D_{1,1}$ is containing level 2 grid $D_{2,1}$ and $D_{1,2}$ is containing level 2 grid $D_{2,2}$.

Now in the algorithm of Berger & Oliger (1984) [13] it is evident that the tree structure can grow or shrink dynamically which needs storage allocation to be dynamic. The actual code was written in Fortran where external memory allocation provided by the linked list of free nodes. These were assigned to new grids and recovered when the subgrid is not necessary and removed.

In our work the implementation of the data structure and the storage allocation is different. We choose C++ programming language. C++ is an object oriented language. Primarily we used the object oriented feature to create the same operational flow in a tree data structure used in the AMR algorithm. It is very convenient to create a class for the grids. We consider the user defined base grid as an object. This object is considered as parent for the other object. We can either derive the finer level grids from the base class object by restricting the access of the base grid integration algorithm or we can also handle this using the array of the grid object. In the second case we need two different integration algorithm under the same class. This is a simple approach and we implemented this approach for our code.

Two kinds of memory management systems are available. One of them is stack another

one is heap. Usually stack is a special kind of memory where the temporary variables are created by the functions. When variables are stored in the stack the compiler knows their size and the allocation is automatic. So, allocation and deallocation of variables in stack memory is automatic. When the size of the variables is unknown or the storage is dynamic it is not useful to use stack memory and also due to the size limitation it is not always a convenient choice. On the other hand heap memory is a place where we don't have to think about the size of the memory. The allocation happens on contiguous blocks of memory in stack where it is totally different in case of the allocation in heap. C++ allows us to allocate and deallocate memory as necessary. The only issue we need to take care of is that we have deallocate properly after allocation. If we fail to do so it will create memory leakage. However, memory leakage can be traced during the debugging stage. This advanced feature of C++ allow us to use a simple implementation of our AMR code. The memory allocation of heap and stack are shown in figure 2.8. On the left side of figure 2.8 the contiguous arrangement of the memory in stack is shown by the same rectangular boxes and the right side is demonstrating the variable size memory allocation for different variable. So, in case of the usage of the heap memory the allocation and deallocation should be handled carefully for the successful computational step.

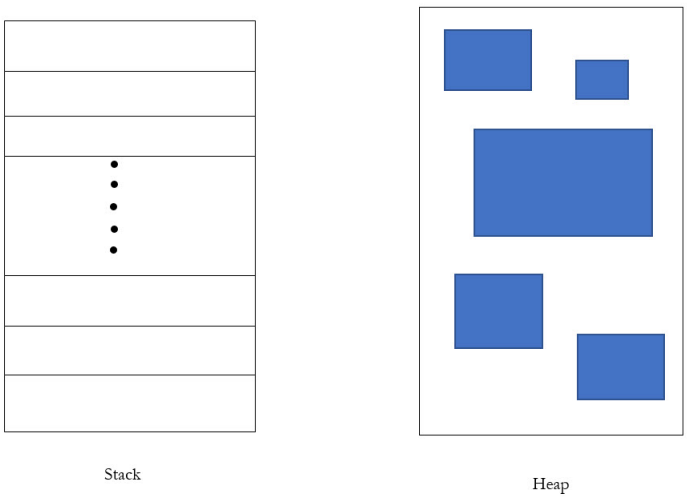


Figure 2.8: Memory allocation in Stack and Heap

2.6.5 Time integration

For the computation we need a time integration scheme. We derived an explicit integration scheme using a finite volume method for the advection equation and Burger's equation. For the RLW equation an implicit finite difference scheme was used. In the discretized equations (2.9) and (2.18) we denoted the cell average quantity in cell i at time step n by u_i^n. In equation (2.30) the cell value at cell i at time step n defined by u_i^n. The solution on each grid is advanced using supplied boundary conditions. The integration algorithm is the same for all grid levels. So, the solution vector for each grid level is independent and they will advance with their own time step. However, boundary conditions are required and these conditions may vary from grid to grid. In [10] the cell values were modified in two cases, the first case is when the coarse grid is covered by a finer grid. The values for the coarse grid will be modified by using the conservative average of the solution on the next finest level grid. This is a simple strategy to save memory space, because extra storage will be required if coarse fluxes need to be redefined around overlayed coarse grid. In this thesis explicit and implicit schemes are used. For both cases, the solution on a single grid is advanced from time t to $t + \Delta t$ with the supplied physical boundary conditions. After the base grid integration all the finer level grids advance sequentially by the order of level with their own time steps to reach $t + \Delta t$. This is very helpful for the implicit scheme which is used for the RLW equation. In the case of an implicit integration step the value from the coarse level data at the advanced time is needed for the boundary conditions. In our code the single grid advance from t to $t + \Delta t$ but the finer grids need more steps to reach from t to $t + \Delta t$. So if the single grid time step is Δt then for level 1 finer grid the time step size would be $\frac{\Delta t}{r}$ similarly for the level 2 the time step would be $\frac{\Delta t}{r^2}$ and finally the time step for level l grid would be $\frac{\Delta t}{r^l}$. So, it can be seen that single grid need just one time integration where level 1, level 2 need integration for r and r^2 times. If the maximum level is l then the number would be r^l. In [10] and [13] the finer level solution synchronized with the coarse solution. The synchronization is demonstrated by figure 2.9 which is created based on [13]. The refinement ratio in this case is 2. The level 2 solution synchronized twice and the level 1 solution is synchronized with level 0 solution just once. In our work the synchronization is done by averaging. If the solution at any level can be defined by ψ then the average can be written as

$$\psi_i^{Coarse} \leftarrow \frac{1}{r} \sum_{p=0}^{r-1} \psi_{i+p}^{fine} \tag{2.32}$$

This is a very important part of the algorithm which keeps the solution stable at the coarse level at different times.

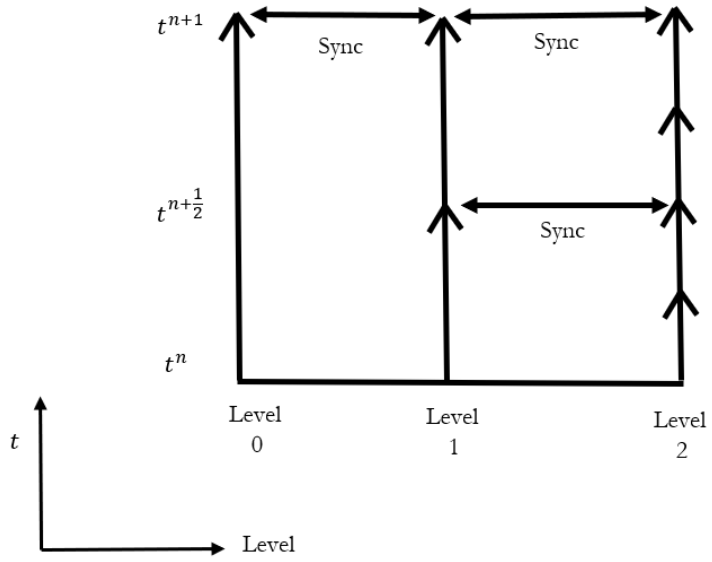


Figure 2.9: Berger-Oliger time stepping

2.6.6 Boundary conditions

The boundary conditions are an essential part of the integration algorithm. The boundary values for the finer level grid can be obtained form the coarse level grid. Usually different interpolation methods are applied to get the boundary values. In [10] a bilinear interpolation technique was used to find the boundary values at any grid level l from the level $l-1$ solution if those boundary values are not available from the adjacent level. However, the interpolation was linear in space and time (if that is required). The steps followed in [13] and [10] are slightly complicated. At any level l they divided the border cells into one or more rectangular patches and then for each rectangular patch they followed the steps given below

i. Solution values will be found from a level $l-1$ grids on a slightly larger rectangular region including border cells.

ii. get the border cell values using linear interpolation.

iii. at any level l the linearly interpolated value will be overwritten if the border cell values can be obtained from the same level grid.

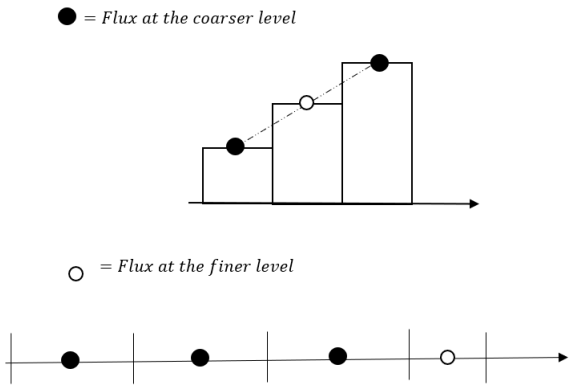


Figure 2.10: Coarse fine grid interpolation

In this work we have used the coarse-fine interpolation procedure for the boundary values but for our one dimensional case the implementation was not similar to [13] and [10]. The level 0 grid boundary values are supplied by the users. The boundary conditions for the higher level grids are supplied from coarse grid by the predefined code. In the explicit solver polynomial fitting is used for the coarse-fine interpolation. Our code interpolates the boundary values from the previous level grid. So, for the integration on level l boundary conditions will be obtained from the coarse grid level $l-1$. The idea of getting the boundary values is shown in figure 2.10. Here the black dots are coarse grid values and the white dots are the values of the finer grid. The finer grid fluxes for any point are obtained by interpolation from the neighbouring coarse grid value. In figure 2.10 the first plot shows the linear interpolation of a cell from two points. The next plot shows the finer level value obtained using the 3 points of the coarse grid level. In the case of implicit solver the code is similar for the space interpolation. However, boundary conditions are also required at the advanced time steps. In the previous section it was mentioned that the integration is done in the different levels sequentially. So, the boundary fluxs at time t and time $t + \Delta t$ are available. Simple linear interpolation is used to obtain boundary conditions at the intermediate times on the finer level grids.

2.6.7 Grid creation operations and regridding

With all the components the AMR algorithm needs to create a grid hierarchy and also maintain the regridding procedure. The base grid will remain fixed all the time during the computation. The operation starts with the refinement criteria. For creating each grid

structure for the upper level the algorithm calls the user specified refinement criteria. If the algorithm starts with the base grid it will call the refinement criteria (such as cut off value or gradient). The next thing it will do is to flag the points where refinement criteria is met. The algorithm will flag the points where refinement criteria are met. These flagged points are used to create the clusters in the upper grid level. A minimum distance is maintained between the clusters of the same level grid. We used a minimum two points distance between two clusters. The procedure of determination of the minimum distance is different in [13] and [10]. Finer grid will be added to all the clusters. In our work after every integration step we go through the regridding step. This regridding is a recursive procedure. In the regridding step we again look for new flagged points and make necessary adjustment to the upper level clusters. In our algorithm, the computational cost for this is little even regridding has done after every step. We minimize the cost by tracking the previous steps information. Our code perform the regridding only in a very small portion of a domain every step if that is required. It should also be noted that that regridding procedure will continue only if it is necessary. This recursive procedure will continue until the maximum level is reached. Once max level is reached the finer grid will be put in the the flagged regions. After the integration steps the setting might get changed. New grid points or cluster might be added or removed.

2.6.8 Combination of all algorithm

All the components and strategies were discussed in the previous sections for the better representation of our algorithm. All the components are connected sequentially. However, one point should me mentioned that the descritization schemes are different but the procedure is similar. It is also should be noted that the algorithm the RLW solver is an implicit solver. So, some components are different here also. All the required algorithms are presented here.

Algorithm 2: Main function (common for all solvers)

```
Build initial grids;
Compute the initial profile u_0 at t = 0;
for l ← 1 to l_max do
    for i ← 1 to r^{l-1} * n do
        fl_i^l =flag(u_{i-1}^l,u_i^l,u_{i+1}^l,Δx/r^l,l);
        i ← i + 1;
    end
    cldata=cluster(fl^l,Δx/r^l,Δt/r^l,p * n);
    initialize(l,u^{l-1},cldata);
    l ← l + 1;
end
while t ≤ t_final do
    advancebase(u_0,u_p,n,Δx,Δt);
    for l ← 1 to l_max do
        for j ← 1 to Number of cluster in level l do
            for k ← 0 to r^l − 1 do
                integratec(cldata,Δx/r^l,Δt/r^l,r);
            end
        end
    end
    set l = l_max;
    while l ≥ 1 do
        averageandupdate(u^l,u^{l-1},r);
        l ← l − 1;
    end
    for l ← 1 to l_max do
        for i ← 1 to r^{l-1} * n do
            fl_i^l =flag(u_{i-1}^l,u_i^l,u_{i+1}^l,Δx/r^l,l);
            i ← i + 1;
        end
        cldata=updatecluster(fl^l,Δx/r^l,Δt/r^l);
        regrid(cldata,u^l,u^{l-1});
        l ← l + 1;
    end
    t ← t + Δt;
end
```

Algorithm 3: Flag function

```
function flag(u^l_{i-1},u^l_i,u^l_{i+1},Δx/r^l,l);
calculate refval1 and refval2 using u^l_{i-1},u^l_i,u^l_{i+1};
if l = 1 then
    if refval1 ≥ level 1 refinement criterion then
        return 1 ;
    else
        return 0 ;
    end
else
    if refval2 ≥ level 2 refinement criterion then
        return 1 ;
    else
        return 0 ;
    end
end
```

Algorithm 4: Clustering

```
function cluster(fl^l,Δx/r^l,Δt/r^l,p * n);
set tempcount=0;
set clustercount=0;
for i ← 0 to p * n do
    if fl^l_i = 1 and tempcount=0 then
        tempcount = i;
    if fl^l_i = 0 at neighbouring and tempcount ≠ 0 then
        if minimum cell distance between cluster > 2 then
            clustercount ← clustercount + 1;
            setval(Δx/r^l,Δt/r^l,tempcount-1,i+1);
        else
            continue;
        end
        tempcount=0;
end
```

Algorithm 5: Initialization on higher level grid

```
function initialize(l,u^{l-1},cldata,r);
set k = 1;
while k ≤ number of clusters in level l do
    for i=cluster starting point to i≤ cluster ending point do
        for j=1 to j ≤r do
            u_l^{r*(i-1)+j} =polynomial interpolation from level (l − 1) using points
              i, i − 1, i + 1;
            j ← j + 1;
        end
        i ← i + 1;
    end
    k ← k + 1;
end
```

Algorithm 6: Integration on base grid (for advection and Burger's equation)

```
function advancebase(u_0,n,Δx,Δt);
set the boundary conditions;
set i = 1;
while i < n do
    compute u_i^0 using 2.9 or 2.18 with u_0;
    u_i^0 = u_{0,i};
    i ← i + 1;
end
```

Algorithm 7: Integration on base grid (for RLW equation)

function advancebase(u_0,n,Δx,Δt);
crate vector s for right hand side of $A\mathbf{x} = \mathbf{b}$;
create vector p for the solution vector of $A\mathbf{x} = \mathbf{b}$;
set the boundary conditions;
$s_0 = u_{0,0} - (2 + \Delta x^2)u_{0,0} + u_{0,1} + (1 + u_{0,0})(u_{0,1} - u_{0,0})\Delta x \Delta t$;
for $i = 1$ *to* $i \leq n - 1$ **do**
    $s_i = u_{0,i-1} - (2 + \Delta x^2)u_{0,i} + u_{0,i+1} + \frac{1}{2}(1 + u_{0,i})(u_{0,i+1} - u_{0,i+1})\Delta x \Delta t$;
    $i \leftarrow i + 1$;
end
call tridiagonalsolver(s,p,n,Δx);
for $i = 0$ *to* $i \leq n - 1$ **do**
    $u_i = p_i$;
    $i \leftarrow i + 1$;
end
for $i = 0$ *to* $i \leq n - 1$ **do**
    $u_{0,i} = u_i$;
end

Algorithm 8: Integration on higher level grid (for advection and Burger's equation)

function integratec (cldata,$\Delta x/R^l$,$\Delta t/r^l$,r);
set a = starting point of the cluster;
set b = ending point of the cluster;
boundary value at a= polynomial interpolation from $a-1$,$a-1$,a from coarse grid value;
boundary value at b= polynomial interpolation from b,$b+1$,$b+2$ from coarse grid value;
set $i = a$;
while $i < b$ **do**
    compute u_i^l using 2.9 or 2.18 with u_0^l;
    $u_{0,i}^l = u_i^l$;
    $i \leftarrow i + 1$;
end

Algorithm 9: Integration on higher level grid(For RLW equation)

function integratec(cldata,$\Delta x/R^l$,$\Delta t/r^l$,r);
set a = r*starting point of the cluster;
set b = r*ending point of the cluster;
set $n = (b - a)$;
a1= linear time interpolation from t and $t + \Delta t$ at right boundary of cluster;
a2= linear time interpolation from t and $t + \Delta t$ at left boundary of cluster;
crate vector s for right hand side of $A\mathbf{x} = \mathbf{b}$;
create vector p for the solution vector of $A\mathbf{x} = \mathbf{b}$;
set the boundary conditions;
$s_0 = u_{0,a} - (2 + \Delta x^2)u_{0,a} + u_{0,a+1} + (1 + u_{0,a})(u_{0,a+1} - u_{0,a})\Delta x\Delta t$-a1;
for $i = 1$ *to* $i \leq n - 1$ **do**
    $s_i = u_{0,a+i-1} - (2+\Delta x^2)u_{0,a+i} + u_{0,a+i+1} + \frac{1}{2}(1+u_{0,a+i})(u_{0,a+i+1} - u_{0,a+i+1})\Delta x\Delta t$;
    $i \leftarrow i + 1$;
end
$s_{n-1} = u_{0,a+n-2} - (2+\Delta x^2)u_{0,a+n-1} + u_{0,a+n} + \frac{1}{2}(1+u_{0,a+n-1})(u_{0,a+n} - u_{0,a+n-2})\Delta x\Delta t$-a2;

call tridiagonalsolver(s,p,n,Δx);
for $i = 0$ *to* $i \leq n - 1$ **do**
    $u_i^l = p_i$;
    $i \leftarrow i + 1$;
end
u_{a+n}^l =polynomial interpolation from previous neighbouing points at same level;
for $i = 0$ *to* $i \leq n - 1$ **do**
    $u_{0,i} = u_i^l$
end

Algorithm 10: Synchronization (average and update the coarse grid value)

```
1  function averageandupdate(u^l,u^{l-1},r);
2  set l = l_max;
3  while l > 0 do
4      for cl = 1 to cl ≤ number of cluster in that level do
5          for i = start point of the cluster+1 to i ≤ endpoint of the cluster−1 do
6              set sum = 0.0;
7              for k = 0 to k < r do
8                  sum← sum+u^l_{r_{*i+k}};
9                  k ← k + 1;
10             end
11             u_i^{(l-1)} = sum/r;
12             i ← i + 1;
13         end
14         cl ← cl + 1;
15     end
16     l ← l − 1;
17 end
```

Algorithm 11: Regridding procedure

```
1  function regrid(cldata,u^l,u^(l−1));
2  set l = 1;
3  while l ≤ l_max do
4      call flag(u^l_{i−1},u^l_i,u^l_{i+1},Δx/r^l,l);
5      call cluster(fl^l,Δx/r^l,Δt/r^l,p ∗ n);
6      set k = 1;
7      while k ≤ number of cluster in level l do
8          for i= first point of the domain where data is not available to i≤ new
             cluster ending point do
9              for j=1 to j ≤r do
10                 u_l^{r∗(i−1)+j} =polynomial interpolation from level (l − 1) using points
                     i, i − 1, i + 1;
11                 j ← j + 1;
12             end
13             i ← i + 1;
14         end
15         k ← k + 1;
16     end
17     l ← l + 1
18 end
```

Chapter 3

Results and Discussion

In the previous chapter, the details of different components of the AMR algorithm have been discussed briefly. We developed our code based on those algorithms. For the implementation of the algorithm C++ was used. A mixed source code is not used for simplicity. Code written in FORTRAN can be added easily to the current thesis work. For all the numerical computation we used an Intel core-i3 processor with 8 gigabytes of RAM. It is essential to validate the computational results by making comparisons with the exact solution. So, to validate the code we started with a very simple linear problem. In this work some simple discretization schemes are used from different works in the literature such as [17] for the RLW solver to make the comparisons easier. In this chapter results from a couple of numerical experiments will be presented to provide a clear view of how our algorithm works. The numerical simulations were done on a uniform grid the results of which were used for the comparison to test the performance of AMR code.

3.1 Advection equation solver

In this section, we are going to start with a simple problem as mentioned earlier. This is a simple demonstration of the efficacy of the AMR algorithm and we do not strive for a highly accurate solution. The advection equation will be solved using (2.9). A simple Gaussian curve is taken as the initial condition. So the problem is

$$u_t + u_x = 0$$

where $0 \leq x \leq 2.0$ and $0 \leq t \leq 1.0$ and the initial condition is

$$u(x, 0) = e^{-100.0(x-0.3)^2}$$

The analytic solution is

$$u(x,t) = e^{-100.0(x-0.3-t)^2}$$

The solution is a wave of permanent form propagating rightward with speed 1. The exact solution is shown in figure 3.1 at time $t = 0.10$, $t = 0.30$ and $t = 0.90$. The solution wave

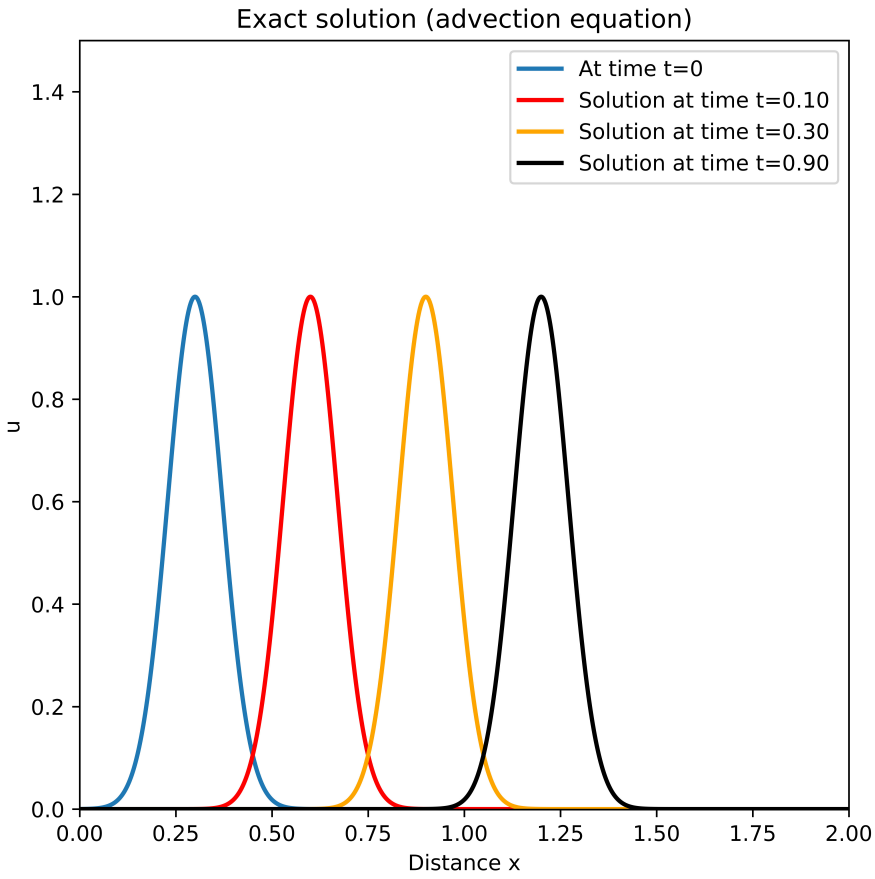


Figure 3.1: Advection equation exact solution

propagates without changing shape.

3.1.1 Solution of advection equation on uniform grid

The behavior of the numerical solution using 200 grid cells and 1000 time steps is shown in figure 3.2. The solution demonstrates a visible deviation from the analytic solution. The wave gets smaller and wider. We used (2.9) for the simulation which is first order approximation in both space and time. There is dissipation but no evidence of dispersion. Numerical dissipation is expected for the numerical solution of this hyperbolic type PDE when only 200 grid cells. Initially the maximum is 1 at $x = 0.3$ but by $t = 0.90$ the maximum has reduced to 0.6. The wave form has also increased in width.

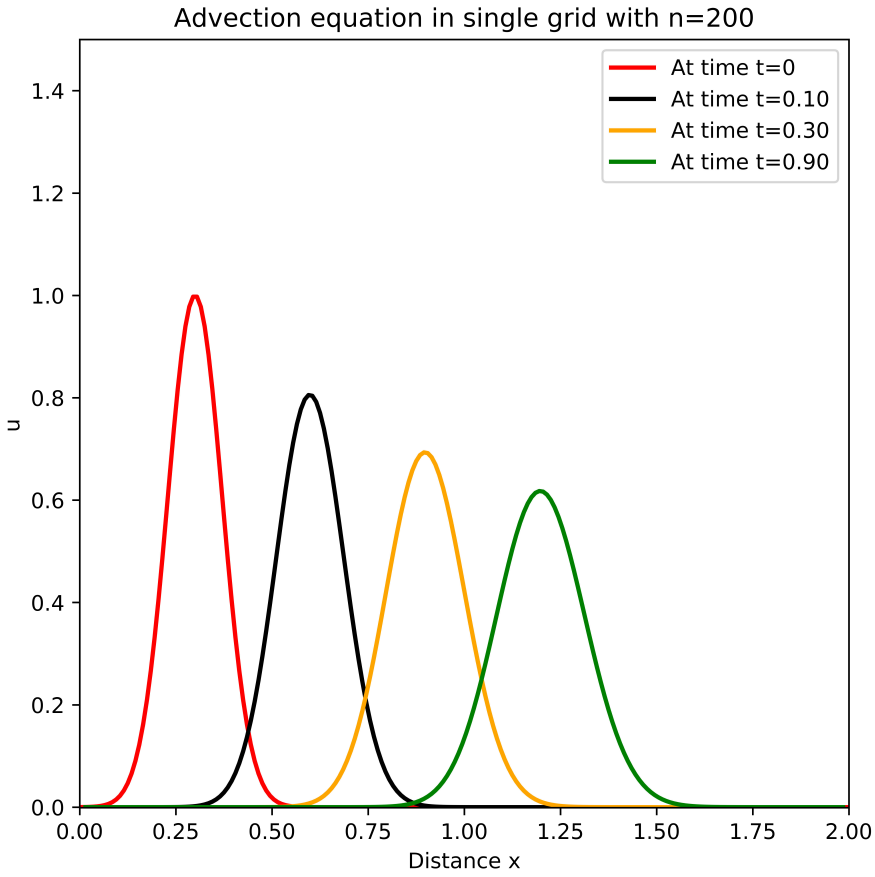


Figure 3.2: Single grid solution of advection equation with 200 grid cells

For any simple straightforward numerical scheme increasing the resolution can reduce the numerical dissipation. So, in the next step we ran the simulation on a uniform grid with

an increased resolution with more time steps. The space interval Δx and time interval Δt are taken in a specific way for the solution such that their ratio between two consecutive grids can be a constant. In this case, the ratio is 4. For the first simulation, we took 200 grid cells and 1000 time steps. To ensure numerical stability we take 1000 time steps to make the time interval Δt sufficiently small. Increase the number of grid cells and times steps by a factor of 4 we will take 800 grid cells and 4000 time steps for the next simulation. In figure 3.3 and 3.4 the simulations are shown with 800 and 3200 grid cells respectively. In figure 3.3 the numerical solution with 800 grid cells produces better results but dissipation is still significant. In the following figure 3.4 the behaviour of the solution is even better since the numerical dissipation has reduced significantly.

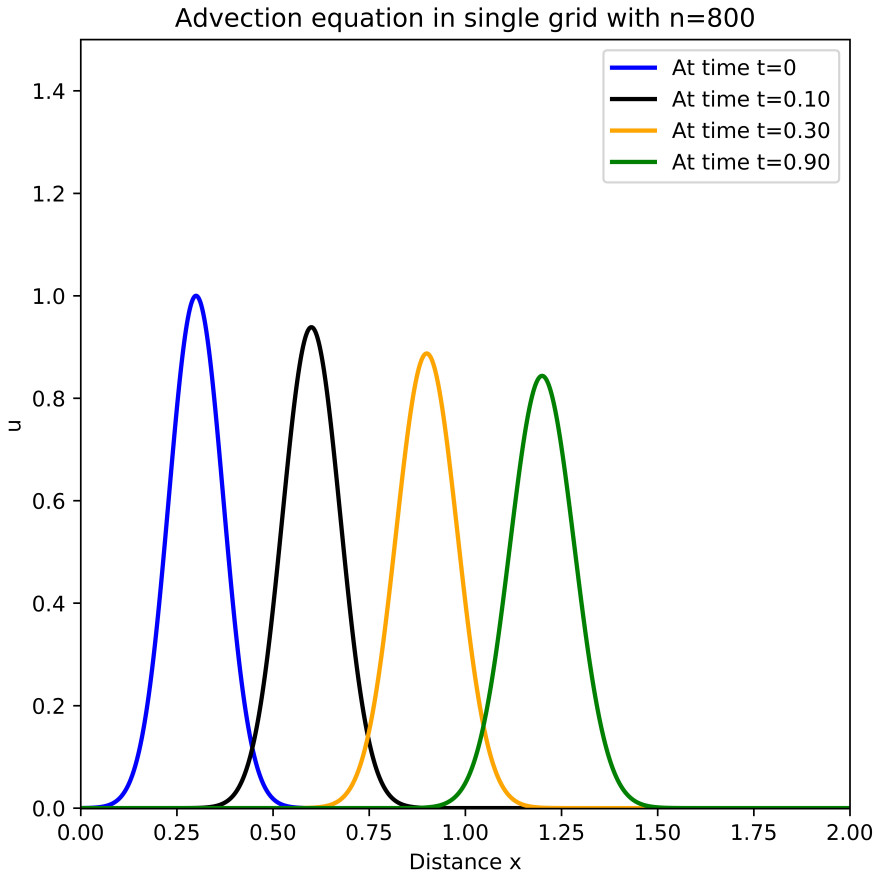


Figure 3.3: Single grid solution of advection equation with 800 grid cells

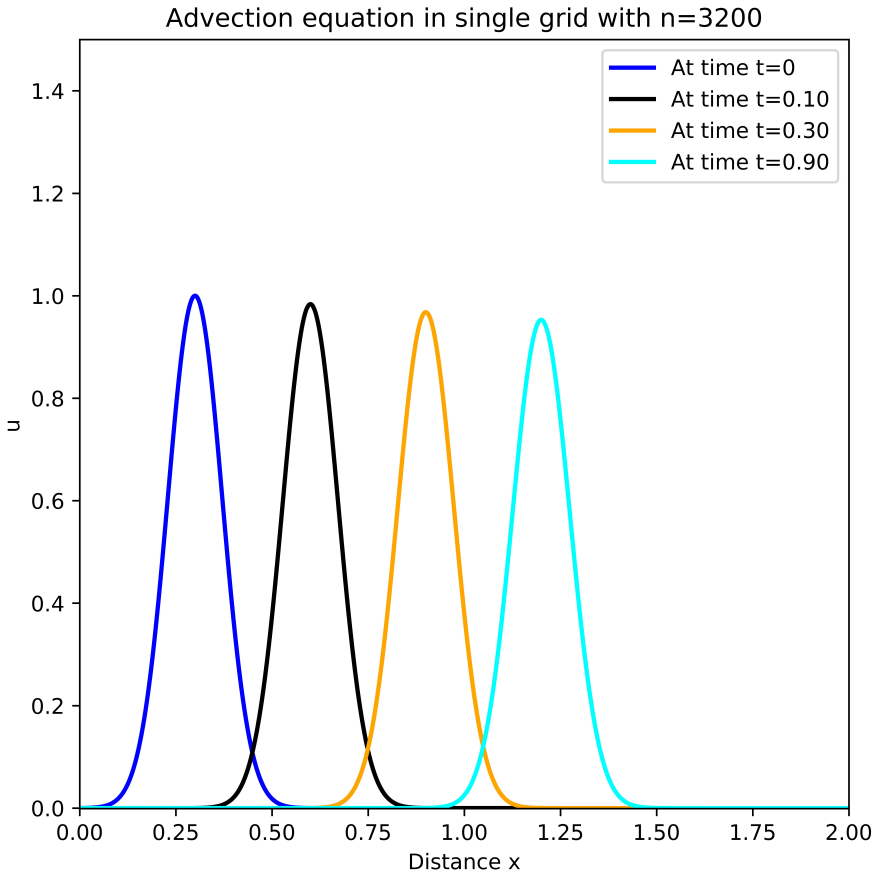


Figure 3.4: Single grid solution of advection equation with 3200 grid cell

In figure 3.5 the error at time $t = 1.0$ is shown. The error is calculate via

$$error = |u_{approximate} - u_{analytic}| \tag{3.1}$$

The error of the solution with 200, 800 and 3200 grid points are shown at the final time $t = 1.0$. We can see from figure 3.5 that the error has reduced significantly with increased resolution. If we define the error norm by

$$||Error||_\infty = \max_i |e_i| \tag{3.2}$$

In (3.2) e_i is the error at each grid point x_i. The error norms for different uniform grid solution approximated by the above formula at time $t = 1.0$ are given in table 3.1. From

this table it can be seen that the error norm reduced more than 2 times with 800 grid points and with 3200 grid points the value reduced more than 7.5 times than the error norm with 200 grid points.

Solution Level	**Number of grid points (n)**	$\|\|Error\|\|_\infty$
0	200	0.400574
1	800	0.169467
2	3200	0.051905

Table 3.1: Advection equation error norm for different uniform grid solution

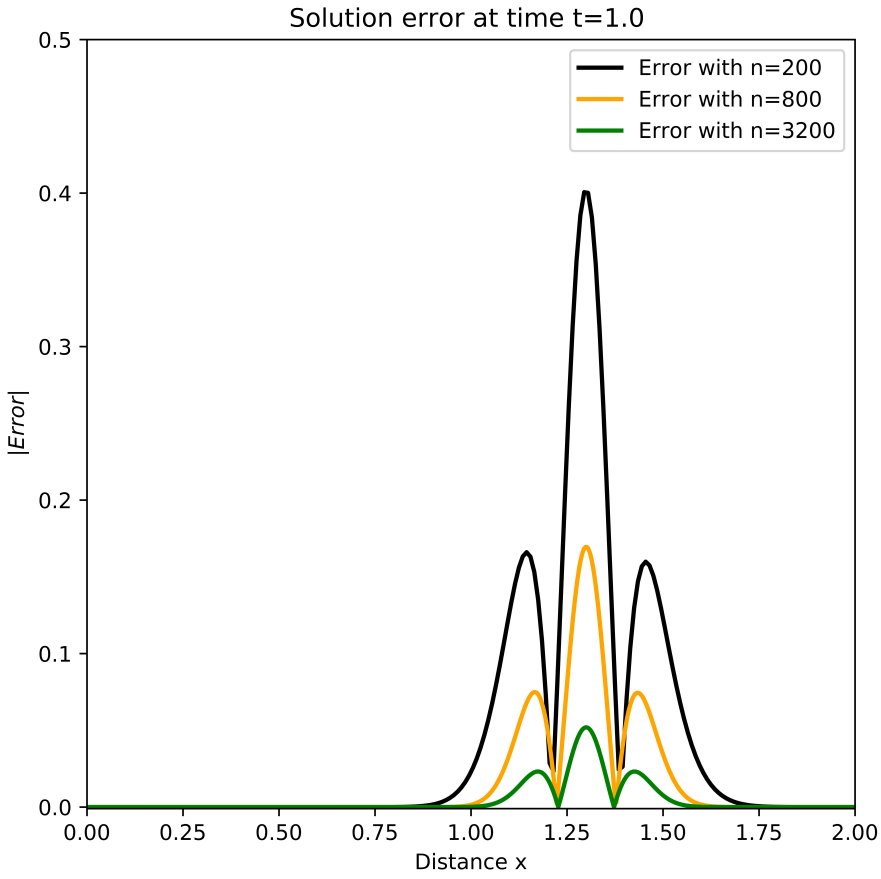


Figure 3.5: Advection equation solution error at time $t = 1.0$ for different uniform grid

Figure 3.6 plots the absolute value of the gradient ($|u_x|$) of the solutions with 800 grid cells. From figure 3.6 we can see that the gradient decreases with time. It is zero at the wave crest. At time $t = 0.1$ the maximum gradient is around 7.0 while the maximum gradients at time $t = 0.30$ and $t = 0.90$ are around 6.0 and 5.0 respectively. However, the pattern of the gradient curves is similar. We want to see the change of the absolute value of the curvature ($|u_{xx}|$) for the solution where the second derivative corresponds to the curvature. In Figure 3.7 the curvature changing is shown by the values of the second derivative. In figure 3.7 the value is large at the peak.

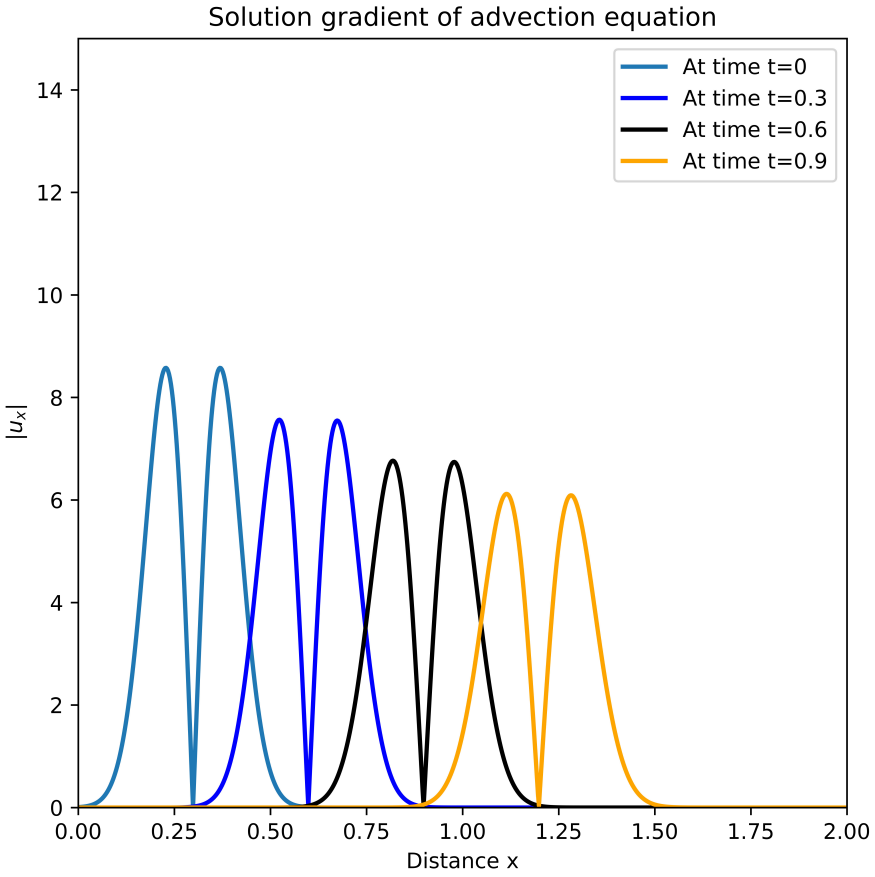


Figure 3.6: Absolute value of the gradient of the solution of advection equation

The maximum values are also decreasing like the gradient. At time $t = 0.1$ the maximum value of the second derivative is around 174 and over time the maximum value of the

second derivative decreased. Finally at time $t = 0.90$ the maximum value of the second derivative was less than 125. Here we can see two inflection points at each time step. It also can be noticed that the gradient is very large when the curvature is very close to zero. On the other hand the curvature reaches its peak when the gradient is very close to zero.

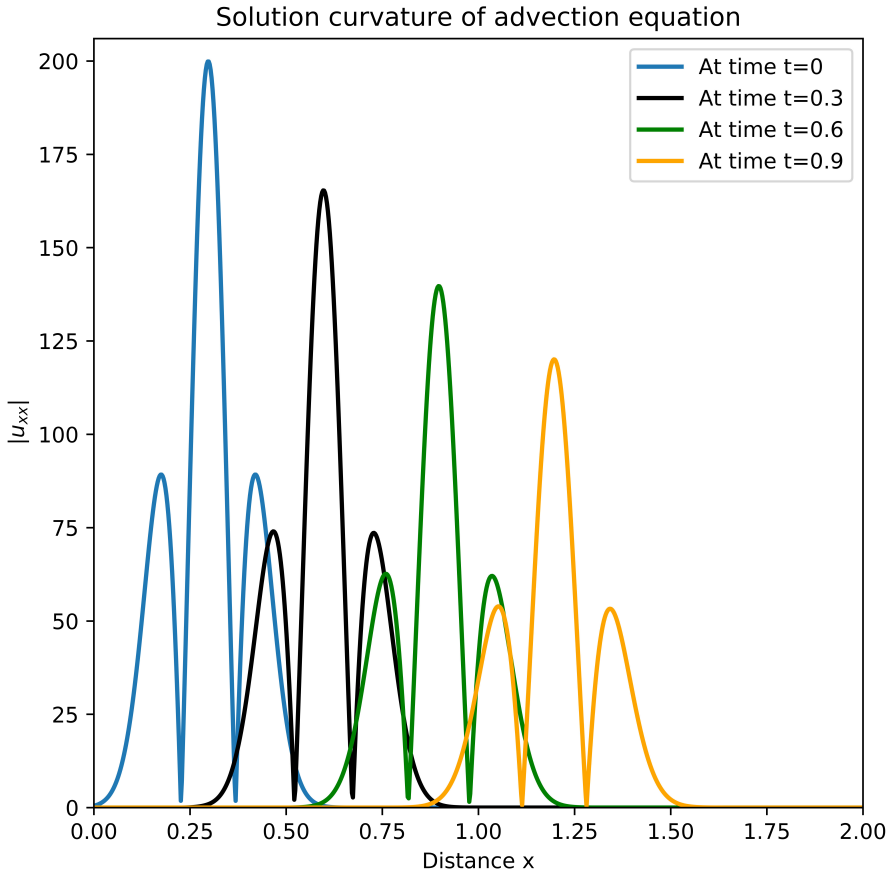


Figure 3.7: Advection equation solution curvature

As the numerical scheme for the simulation is of first-order it is expected that the solution accuracy will be improved by the increased resolution. From figures 3.2 and 3.4 the improvement can be seen.

However, the required time for the computation also increases. The required time for computation increased not only due to the higher resolution but also for the increased number of time steps. Figure 3.8 plots the required time for different numbers of grid

Number of grid Points	Time steps	Δx	Δt	Required time(s)
200	1000	0.01	.001	0.033
800	4000	0.0025	0.00025	0.241
3200	16000	0.000625	0.0000625	3.44

Table 3.2: Required computational time for different grid resolution for advection equation

cells. The computational time is getting significantly large for the highest resolution for our problem. Table 3.2 represents the similar information for three specific computation of the figure 3.8 in tabular form.

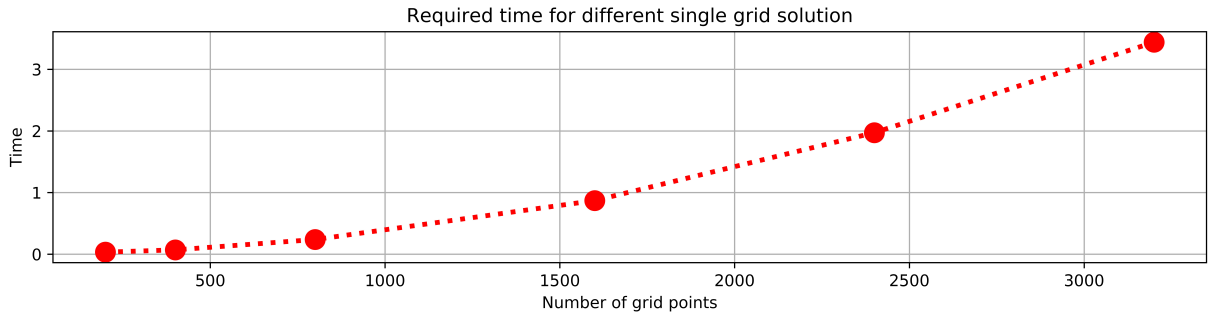


Figure 3.8: Computational time for different single grid solution (Advection equation)

From table 3.2 we can see that the second simulation with 800 grid cells and 4000 time steps has taken 7 times more computational time and the scenario is worse using 3200 grid cells with 16000 time steps where the computation is 104 times slower than the first computation.

3.1.2 Solution of advection equation with AMR

For the AMR computation we are going to use 200 grid cells as our level 0 grid which is also called the single grid. 1000 time steps have taken for this single grid. The maximum level of refinement is fixed at 2 and the refinement ratio between levels is fixed at 4. In figure 3.9 the AMR result is shown at time $t = 0.10$, $t = 0.30$ and $t = 0.90$. We choose summation of $|u_x|$ and $|u_{xx}|$ as the refinement criterion for both level 1 and level 2. The level 1 refinement criterion uses a very small value since we want to cover the wave in the computational domain. The algorithm does not cover the parts of the domain where

the solution characteristics (such as gradient or curvature) are close to zero. For level 2 we increase the value of the refinement criterion. Figure 3.9 is shows the solution with $|u_x| + |u_{xx}| \geq 1.0$ (level 1 criterion) and $|u_x| + |u_{xx}| \geq 5.2$ (level 2 criterion) and figure 3.10 shows the solution with $|u_x| + |u_{xx}| \geq 1.0$ (level 1 criterion) and $|u_x| + |u_{xx}| \geq 85.0$ (level 2 criterion). The location of the different level grids is also indicated. The summary of the AMR computations of the simulations is shown in table 3.3 and table 3.4.

Number of grid points in Level 0	200
Number of time steps in Level 0	1000
Refinement ratio	4
Level 0 grid space width	0.01
Level 0 grid time interval	0.001
Maximum level of refinement	2

Table 3.3: Summary of AMR computation of advection equation

Level 1 refinement criterion	**Level 2 refinement criterion**	**Required time(s)**
$\|u_x\| + \|u_{xx}\| \geq 1.0$	$\|u_x\| + \|u_{xx}\| \geq 5.2$	0.239
$\|u_x\| + \|u_{xx}\| \geq 1.0$	$\|u_x\| + \|u_{xx}\| \geq 85.0$	0.1095

Table 3.4: Advection equation AMR solutions required time

So, the computation starts with 200 grid cells and 1000 time steps and it refines portion of the domain in time and space with the factor of 4 where the level 1 refinement criteria are fulfilled. So, again if the level 2 refinement criteria is met then the algorithm computes the solution with $\Delta x = 0.000625$ and $\Delta t = .0000625$ for that region. In figure 3.9 the black curve is the initial profile. The red curve is the solution at time $t = 0.10$. The horizontal red line is indicating the location of level 0, level 1, and level 2 solution exactly at that time. It can be seen that at time $t = 0.10$ from $x = 0$ to $x = 0.25$ (approximately) and from $x = 0.78$ (approximately) to $x = 2.0$ level 0 region exists. Then the level 1 solution exists from approximately $x = 0.25$ to $x = 0.77$. In figure 3.9 at time $t = 0.10$ the level 2 region exists around $x = 0.3$ to $x = 0.73$ while in figure 3.10 the level 2 region exists around $x = 0.55$ to $x = 0.65$ at time $t = 0.10$. Also the grid hierarchy can be seen from the AMR solution according to our algorithm. In figure 3.9 and 3.10 it can be seen that level 1 grid is contained by the level 0 grid and level 2 grid is contained by level 1 grid. The solution behaves like the high resolution single grid solution. In figure 3.13 and 3.14 both the AMR and highest resolution solutions are shown and a good agreement can be

observed. However, at time $t = 0.30$ and $t = 0.90$ a little bit difference can be found at the bottom of the solution in figure 3.14. The principal reason behind this is that those portion of the solution is computed with the level 1 solution.

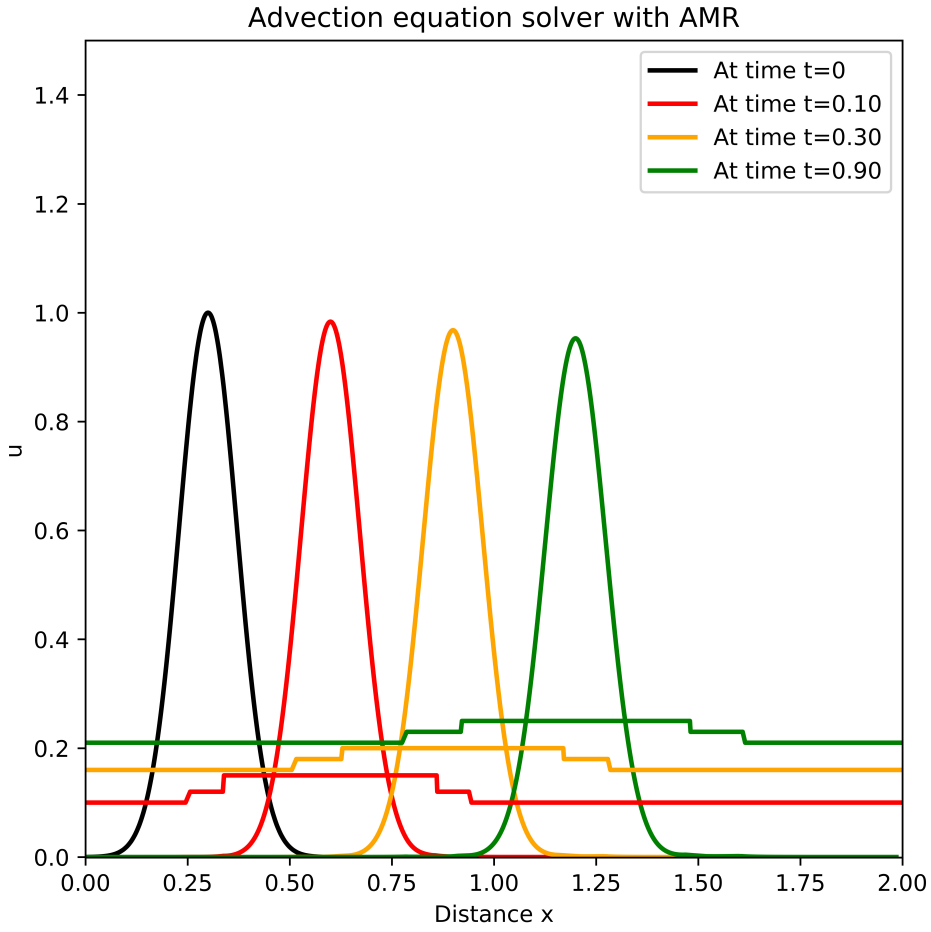


Figure 3.9: AMR solution for the advection equation with refinement criteria $|u_x| + |u_{xx}| \geq 1.0$ for level 1 and $|u_x| + |u_{xx}| \geq 5.2$ for level 2. The step functions indicate the locations of the different level grids with matching colours

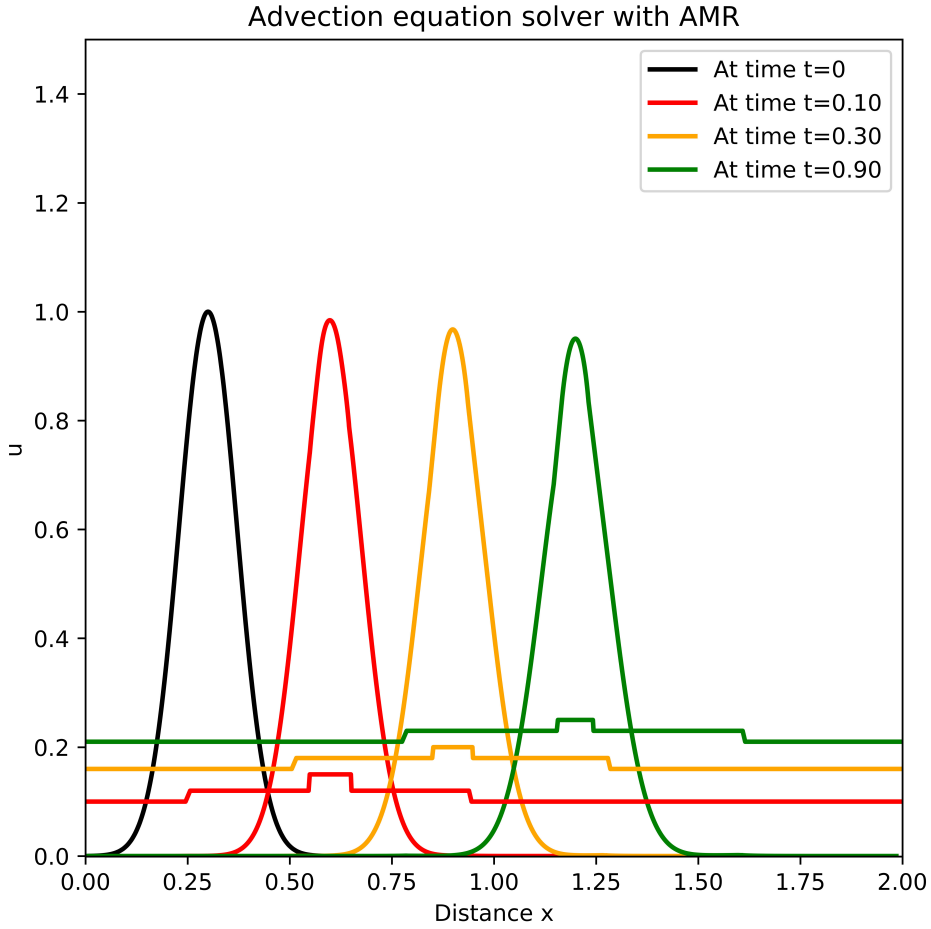


Figure 3.10: AMR solution for the advection equation with refinement criteria $|u_x| + |u_{xx}| \geq 1.0$ for level 1 and $|u_x| + |u_{xx}| \geq 85$ for level 2. The step functions indicate the locations of the different level grids with matching colours

In figure 3.11 the error of the AMR solution at different levels is shown at the final time $t = 1.0$. In figure 3.11 the refinement criterion for level 1 is $|u_x| + |u_{xx}| \geq 1.0$ and $|u_x| + |u_{xx}| \geq 5.2$ is the level 2 criterion. In figure 3.12 level 1 criterion is the same but the level 2 refinement criterion is $|u_x| + |u_{xx}| \geq 85.0$. The summary of these figures is shown in tables 3.5 and 3.6 respectively. From these tables we can see that the maximum error at time $t = 1.0$ at level 1 and level 2 is different. In figure 3.12 the level 2 refinement criterion is very high so the scenario is different here. The level 1 grid covers a significant portion of the wave so the values of the maximum error have changed for both level 1 and level 2

grid. In figure 3.14 we can see that at time $t = .90$ the AMR solution at the bottom does not match exactly with the finest resolution solution since the level 2 grid exits only for a small region which results in the error increment in the other parts of the solution.

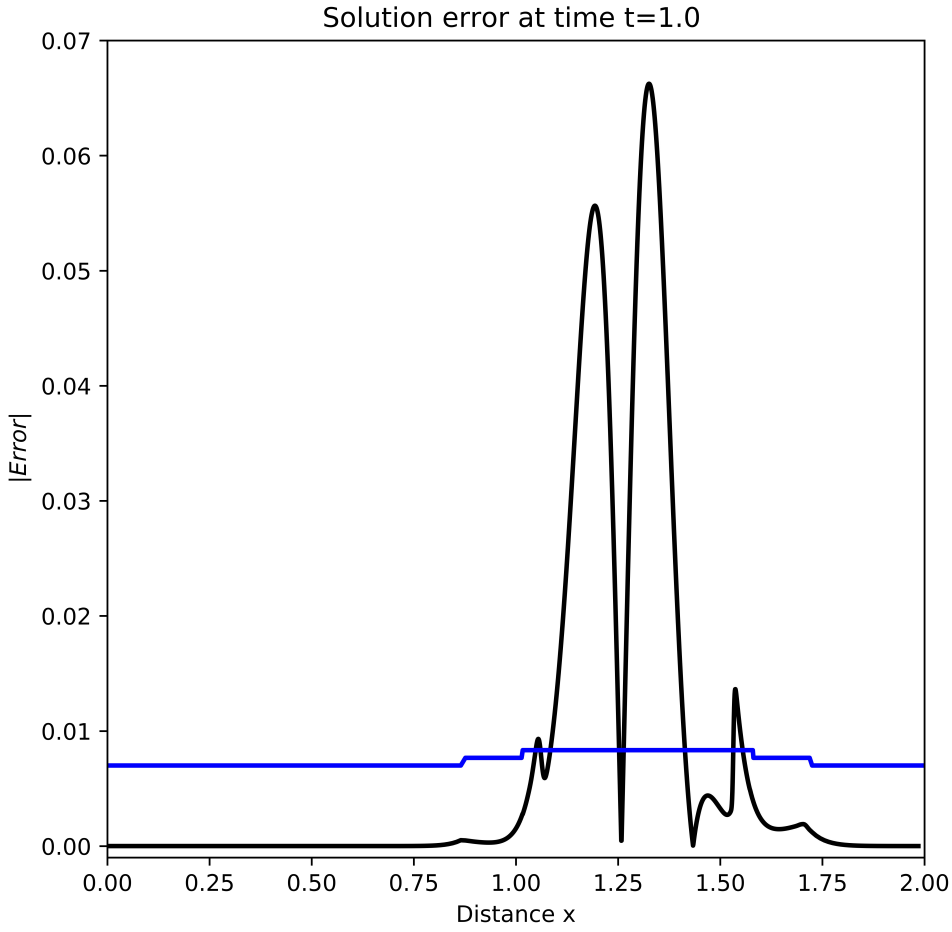


Figure 3.11: Error at different levels of the AMR solution at time $t = 1.0$ for the advection equation with $|u_x| + |u_{xx}| \geq 1.0$ as the level 1 refinement criterion and $|u_x| + |u_{xx}| \geq 5.2$ as the level 2 refinement criterion. The horizontal step function indicate the locations of the different level grids

Level	Δx	Δt	**Refinement criterion**	$\|\|Error\|\|_\infty$
0	0.01	0.001		0.00119917
1	0.0025	0.00025	$\|u_x\| + \|u_{xx}\| \geq 1.0$	0.00382384
2	0.000025	0.0000625	$\|u_x\| + \|u_{xx}\| \geq 5.2$	0.0662431

Table 3.5: Summary of error of the AMR solution of advection equation in figure 3.11

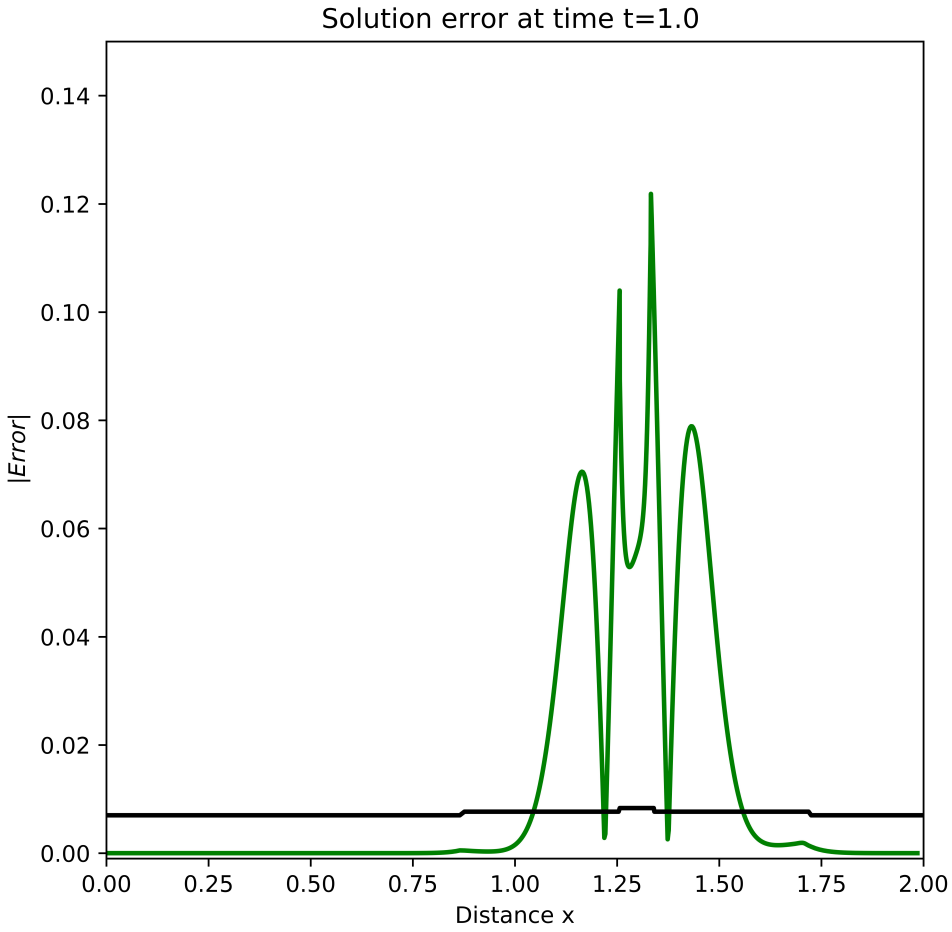


Figure 3.12: Error at different levels of the AMR solution same as figure 3.12 with level 2 refinement criterion changed to $|u_x| + |u_{xx}| \geq 85$

Level	Δx	Δt	**Refinement criterion**	$\|\|Error\|\|_\infty$
0	0.01	0.001		0.00119917
1	0.0025	0.00025	$\|u_x\| + \|u_{xx}\| \geq 1.0$	0.103968
2	0.000025	0.0000625	$\|u_x\| + \|u_{xx}\| \geq 85$	0.121839

Table 3.6: Summary of error of the AMR solution of advection equation in figure 3.12

It is noticeable from table 3.4 that the computational time for AMR solution is only 0.239 second and 0.1095 second for the solution shown in figure 3.9 and 3.10 respectively. This is very good compared to the single grid solutions. The first solution is slightly faster than the level 1 solution (800 grid cells and 4000 time steps) but the second one is almost 2 times faster than the level 1 single grid solution the numerical dissipation is reduced significantly. The first AMR solution in figure 3.9 is more than 14 times faster than the level 2 single grid solution with 1600 grid cells and 16000 time steps and the second solution of figure 3.10 is 31 times faster than level 2 solution. In figure 3.13 and 3.14 both the solutions are compared with the highest resolution single grid solution. The AMR solutions agree with the highest resolution single grid solutions. Although a little bit difference can be found from the highest resolution solution if the refinement criterion is very high ($|u_x| + |u_{xx}| \geq 85.0$ for level 2) but the overall performance is satisfactory. We will apply similar procedure to the other nonlinear and dispersive wave problems with different refinement criteria.

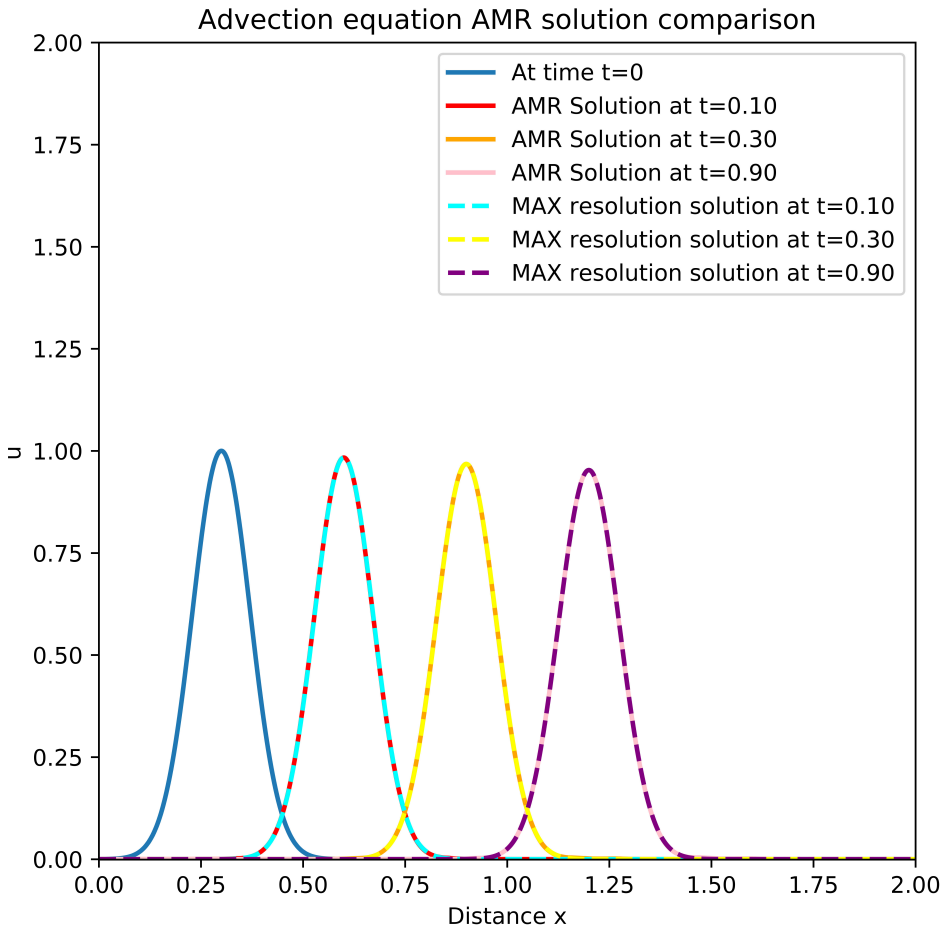


Figure 3.13: AMR solution (level 1 $|u_x| + |u_{xx}| \geq 1.0$ and level 2 $|u_x| + |u_{xx}| \geq 5.2$) comparison for advection equation with highest resolution solution

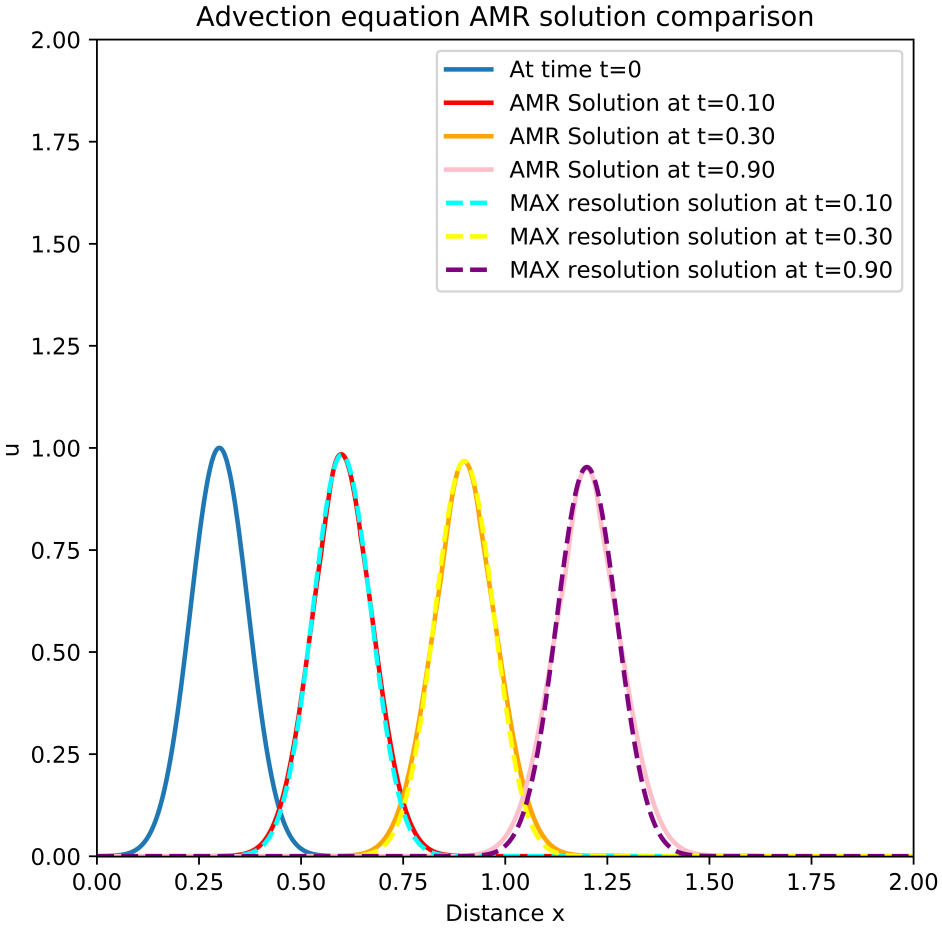


Figure 3.14: AMR solution (level 1 $|u_x| + |u_{xx}| \geq 1.0$ and level 2 $|u_x| + |u_{xx}| \geq 85$) comparison for advection equation with highest resolution solution

3.2 Burger's equation solver

In the previous section, the performance of our AMR code was shown for a simple linear problem and it worked well. In this section we are going to consider Burger's equation. This problem is more difficult because it includes nonlinear steeping and shock formation making it more difficult to implement AMR clustering. Burger's equation is

$$u_t + uu_x = \nu u_{xx}$$

with the initial condition

$$u(x, 0) = u_0(x)$$

and boundary conditions are

$$u(0, t) = a$$

$$u(l, t) = b$$

where $0 \leq x \leq l$ and $0 \leq t \leq t_{final}$. We will consider two cases using our code. The first problem will be a single crest problem and the second problem is a double crest problem.

3.2.1 Single crest problem

For this problem the initial condition is

$$u(x, 0) = e^{-100.0(x-0.3)^2}$$

where $0 \leq x \leq 1.0$ and $0 \leq t \leq 1.0$ and the boundary conditions are $u(0, t) = 0 = u(1, t)$ and $\nu = 0.001$.

3.2.1.1 Burger's equation (single crest problem) solution on uniform grid

With the initial setup described above the numerical solution of Burger's equation with 100 grid cells and 1000 time steps is shown in figure 3.15. We have used finite volume discretization which we derived in the previous chapter. Strong dispersion and damping can be observed in the solution. It can be observed that the grid resolution should be small enough to maintain smooth and accurate results. From figure 3.15 we can see that the solution is not smooth and breaks down shortly after $t = 0.125$. It starts to break down at the top where large gradients form. Small ν results in the formation of a shock. The steep shock encountered by the solutions can not be resolved with the coarse grid. It also should

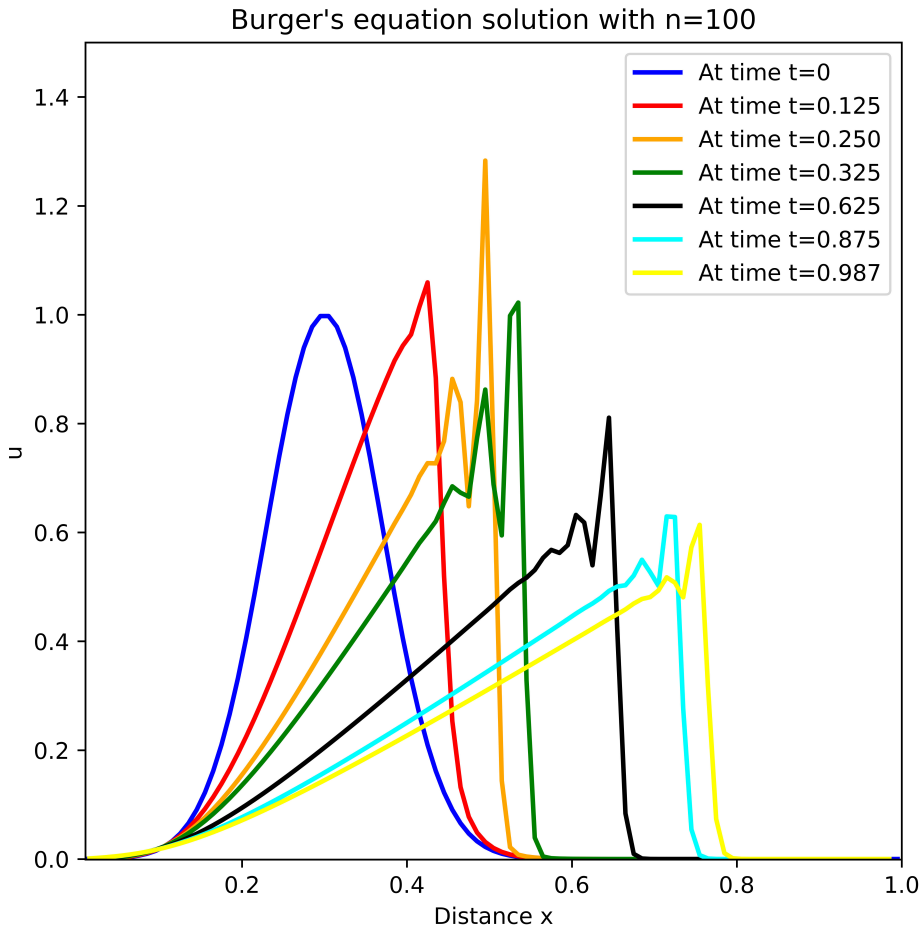


Figure 3.15: Burger's equation single grid solution with 100 grid cells (single crest problem)

be noted that we have used 2.19t and **??** for the scheme 2.18. This problem is a convection dominant problem since the diffusion is very weak. So, for our scheme the Péclet number (Pe) should be less than 2. However, with only 100 grid points the Pe reaches up to 10 which is far away from 2 and this results in the overshooting and undershooting in the figure 3.15. In figure 3.16 this problem was resolved by using the upwind scheme. In our case we will stick on scheme 2.18 to show that our AMR algorithm can deal with the limitations of the scheme. In figure 3.16 we can see that there is no breaking down of the solution at the top although the solution is not accurate enough.

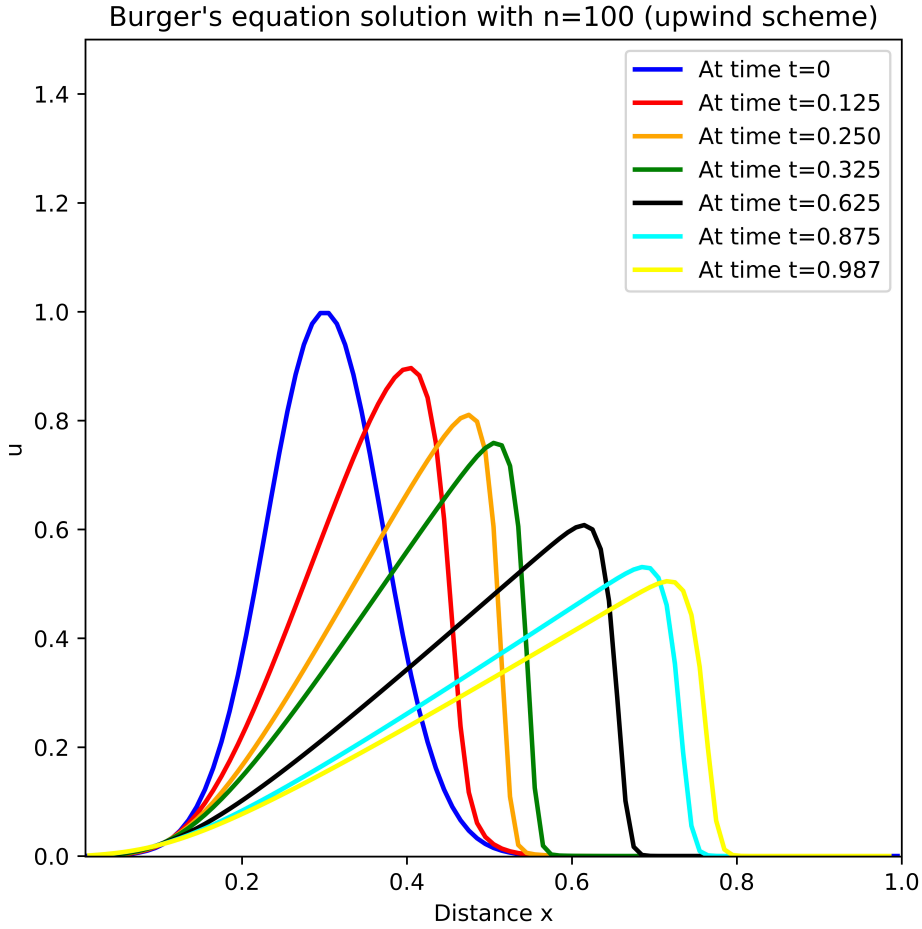


Figure 3.16: Burger's equation single grid upwind solution with 100 grid cells (single crest problem)

In figure 3.17 the evolution of the absolute value of the gradient over time can be found. After starting the computation the gradient rapidly becomes very large at the front of the wave which fills a small portion of the computational domain. At time $t = 0.125$ the magnitude of the absolute value of the gradient is close to 50 around $x = 0.45$. At time $t = 0.250$ the value reaches to its peak which close to 100 around $x = 0.55$. In later times this value declines. It has reduced to below 40 near $x = 0.75$ at $t = 0.875$ and $t = 0.985$. The gradient evolves in time and large gradients occur at different portions of the computational spaces. It is evident that the region of high gradient requires special attention during the computation to ensure solution accuracy. So, the idea of putting

the finest grid over the high gradient region to achieve an accurate and smooth solution suggests using the gradient as our refinement criterion.

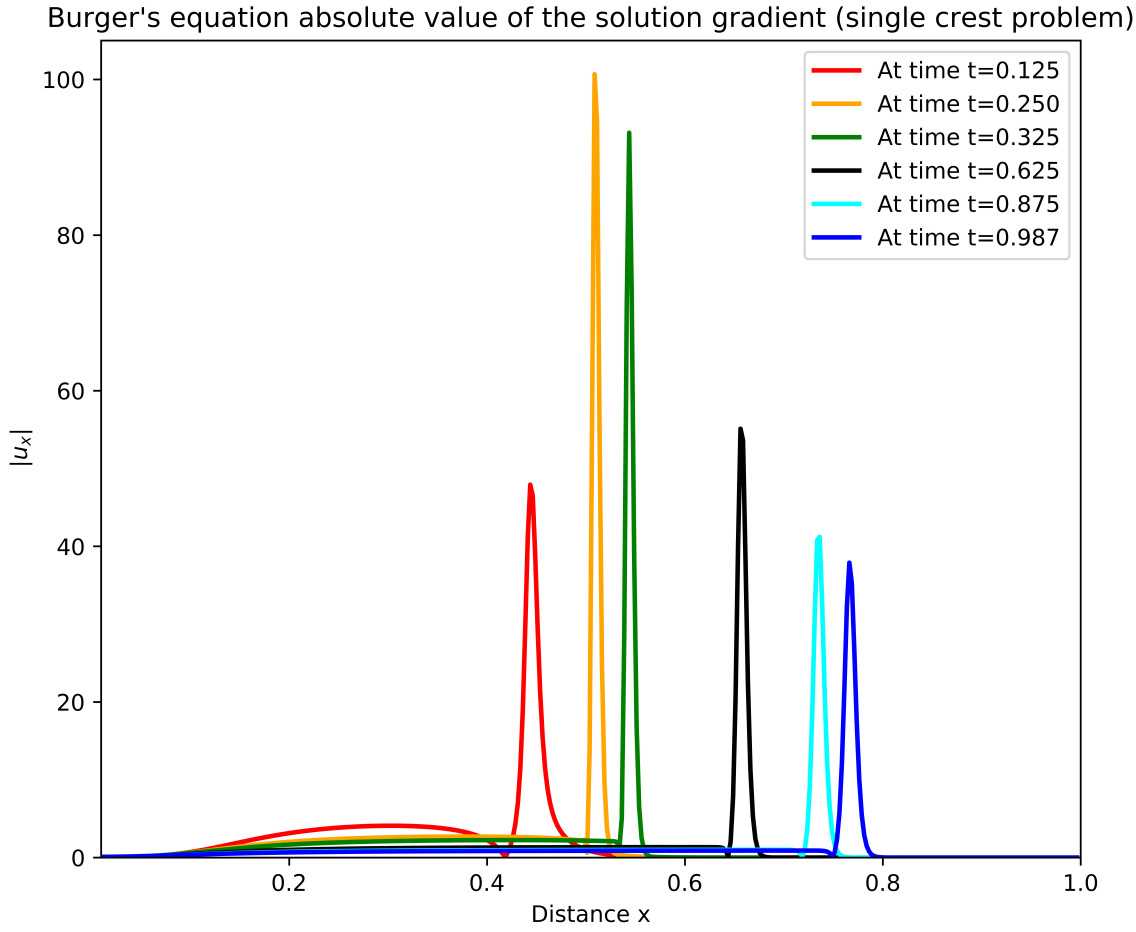


Figure 3.17: Absolute value of the solution gradient at different time (Burger's equation single crest problem)

The numerical solution with 400 grid cells using 4000 time steps is plotted in figure 3.18. The solution is better in these figures. In figure 3.18 a little bit noise can be found at the top at time $t = 0.250$ and $t = 0.325$ where we have seen the high gradient values in figure 3.17. So, the highest level of accuracy has not yet been achieved. However, the solution at the later times remains smooth since the gradient is not that high for a large portion of the domain.

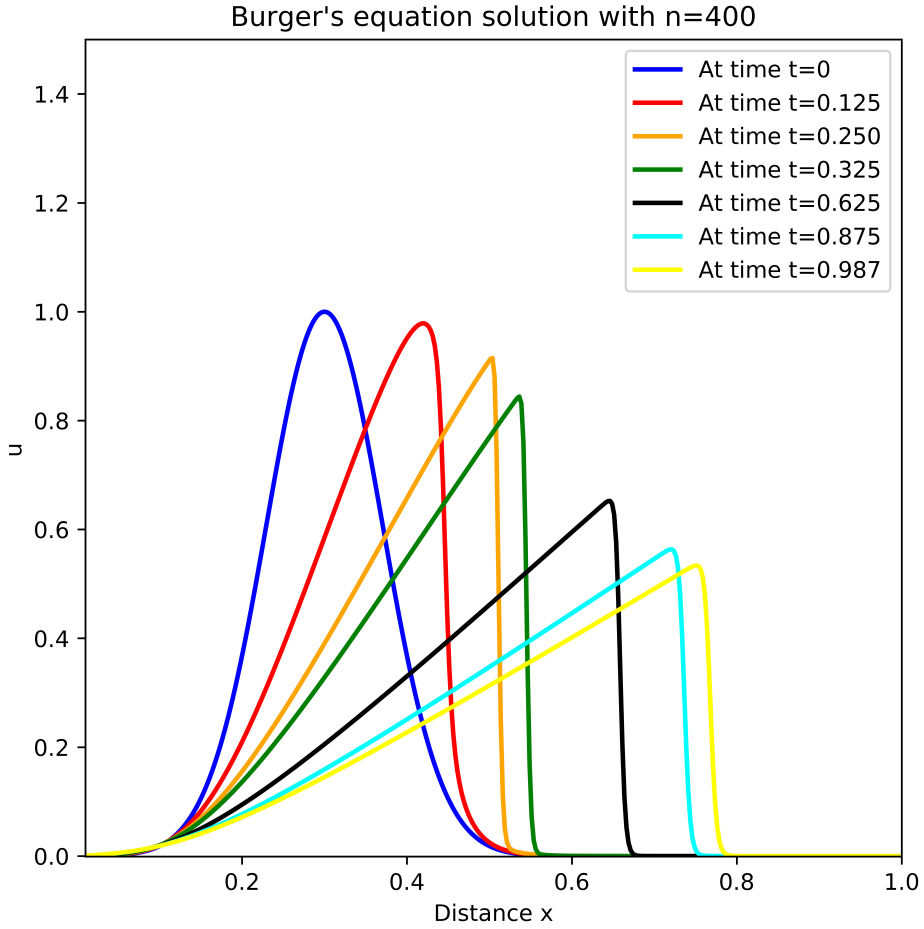


Figure 3.18: Burger's equation single grid solution with 400 grid cells (single crest problem)

In figure 3.19 the solution obtained using 1600 spatial grid cells and 16000 time steps is shown. The problems arising in the previous simulations due to the strong non-linearity and week diffusion have been resolved by the high resolution grid. The solution is smooth and is not breaking due to the steep gradient.

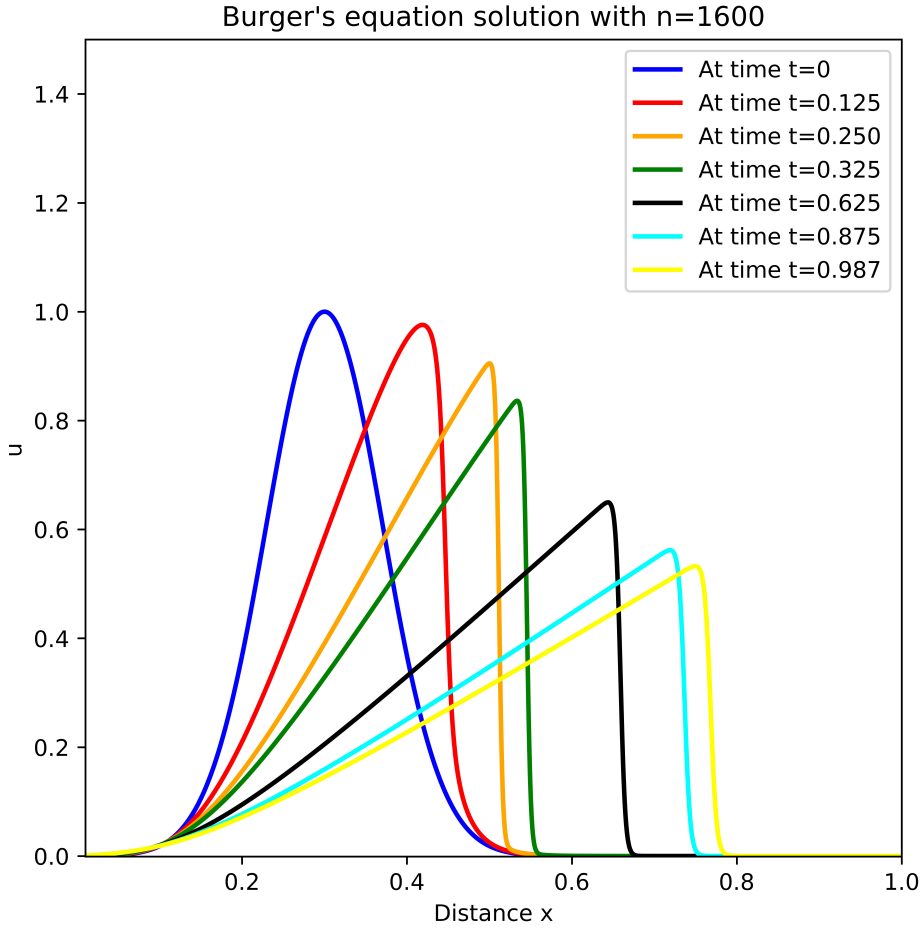


Figure 3.19: Burger's equation single grid solution with 1600 grid cells (single crest problem)

In figure 3.20 shows how the computational time varies with the number of grid points for the uniform grid cases. It is expected that the computational time will increase by the factor we multiply the grid. However, it can be seen from the figure that the time increment is agreed with our expectation initially but with large number of grids the increment is much higher than the expected value. One of the probable reason might be memory management during the computation which also consumes some time. Table 3.7 shows the detailed time information required for three of the numerical solutions.

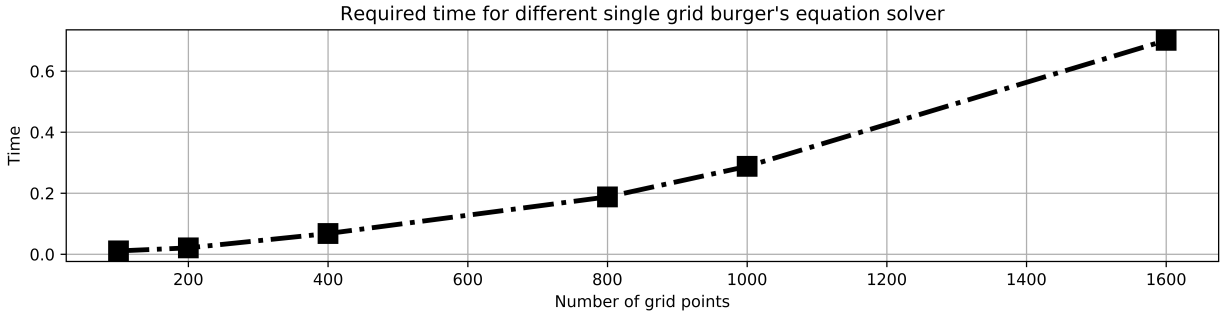


Figure 3.20: Required time for different grid cells (Burger's equation single crest problem)

Number of grid Points	**Time steps**	Δx	Δt	**Required time(s)**
100	1000	0.01	0.001	0.016
400	4000	0.0025	0.00025	0.068
1600	16000	0.000625	6.25×10^{-5}	0.701

Table 3.7: Required computational time for different grid resolution for Burger's equation (single crest problem)

3.2.1.2 Burger's equation (single crest problem) solution with AMR

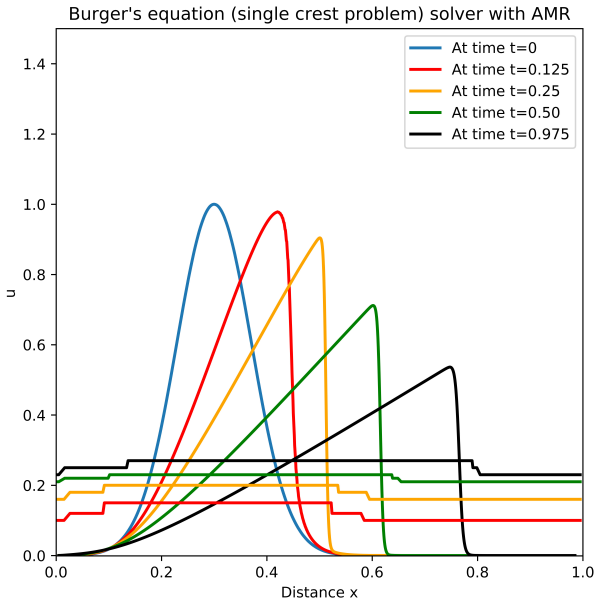
We now present numerical solutions of Burger's equation obtained with the AMR code. We use 100 grid cells in our level 0 grid with the step size is $\Delta t = 0.001$. For the first level the refinement we choose the geometric property of the solution which is the cut-off value of the solution in this case. By taking the cut-off value small (such as $|u| \geq .001$) we cover the wave in the computational domain. We will not include the cells in level 1 which are very close to zero. For level 2 refinement we have taken the absolute value of the gradient as the refinement criterion. From figure 3.17 we can see the change of the value of the absolute values of the gradient and we consider the value greater than or equal 0.50 as our starting point for level 2 refinement criterion. Next we have run a couple of simulations with different values of refinement criteria. A summary of the simulations is shown in table 3.8.

Figure 3.21 shows the AMR solution for two different level 1 refinement criterion ($|u| \geq 0.001$ and $|u| \geq 0.007$) with the level 2 refinement criterion being $|u_x| \geq 0.50$ in both cases. Again we have kept the level 1 refinement criterion constant at $|u| \geq 0.007$ and run the simulations with $|u_x| \geq 2.5$ and $|u_x| \geq 4.5$ as the level 2 refinement criterion which

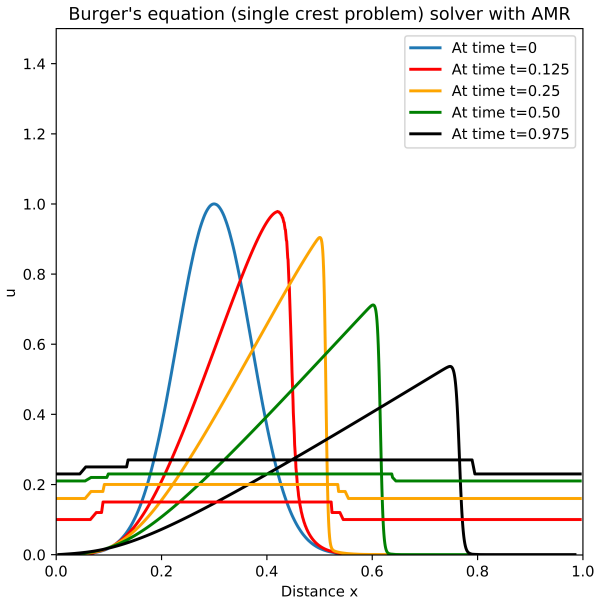
Level 1 refinement criterion	**Level 2 refinement criterion**	**Required time(s)**
$\|u\| \geq 0.001$	$\|u_x\| \geq 0.50$	0.376
$\|u\| \geq 0.007$	$\|u_x\| \geq 0.50$	0.361
$\|u\| \geq 0.007$	$\|u_x\| \geq 2.5$	0.140
$\|u\| \geq 0.007$	$\|u_x\| \geq 4.5$	0.093

Table 3.8: Required time for different refinement criteria (Burger's equation single crest problem)

are shown by figure 3.22. However, reducing the level 1 criterion does not make a great difference. On the other hand, changing the gradient value from 0.5 to 2.5 reduces the time remarkably but when the gradient value is 4.5 the time reduction is also noticeable but the solution remains similar as expected. So, we choose the simulation with $|u| \geq 0.007$ as the level 1 and $|u_x| \geq 5.0$ as the level 2 refinement criteria for the discussion and the results are shown in figure 3.23 . The summary of AMR computations of Burger's equation for the single crest case is given in table 3.9.

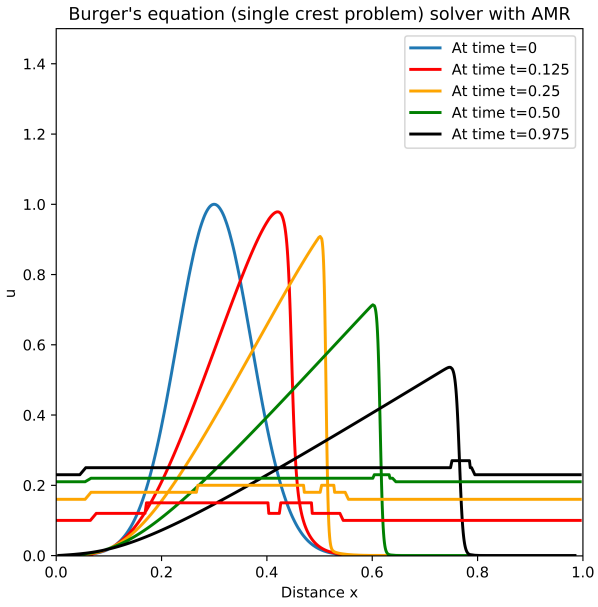


(a)

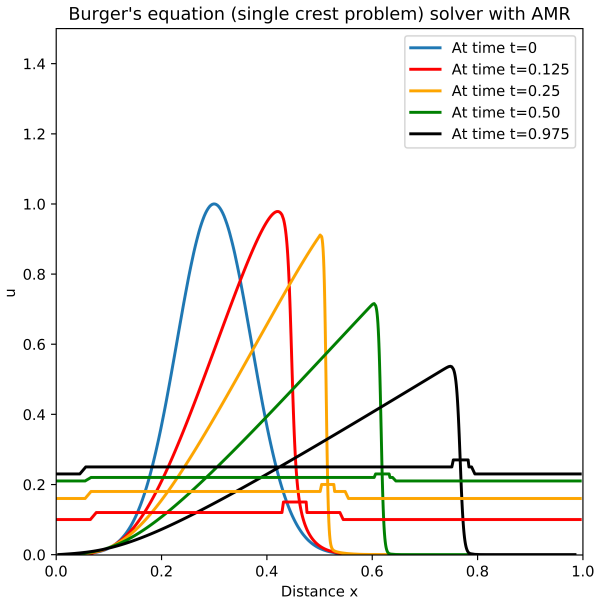


(b)

Figure 3.21: AMR solution for Burger's equation(single crest problem) with level 1 refinement criterion (a) $|u| \geq 0.001$ and (b) $|u| \geq 0.007$ and $|u_x| \geq 0.5$ as the level 2 refinement criterion for both cases. The step functions indicate the locations of the different level grids with matching colours



(a)



(b)

Figure 3.22: AMR solution for Burger's equation(single crest problem) with level 2 refinement criterion (a) $|u_x| \geq 2.5$ and (b) $|u_x| \geq 4.5$ and $|u| \geq 0.007$ as the level 1 refinement criterion for both cases. The step functions indicate the locations of the different level grids with matching colours

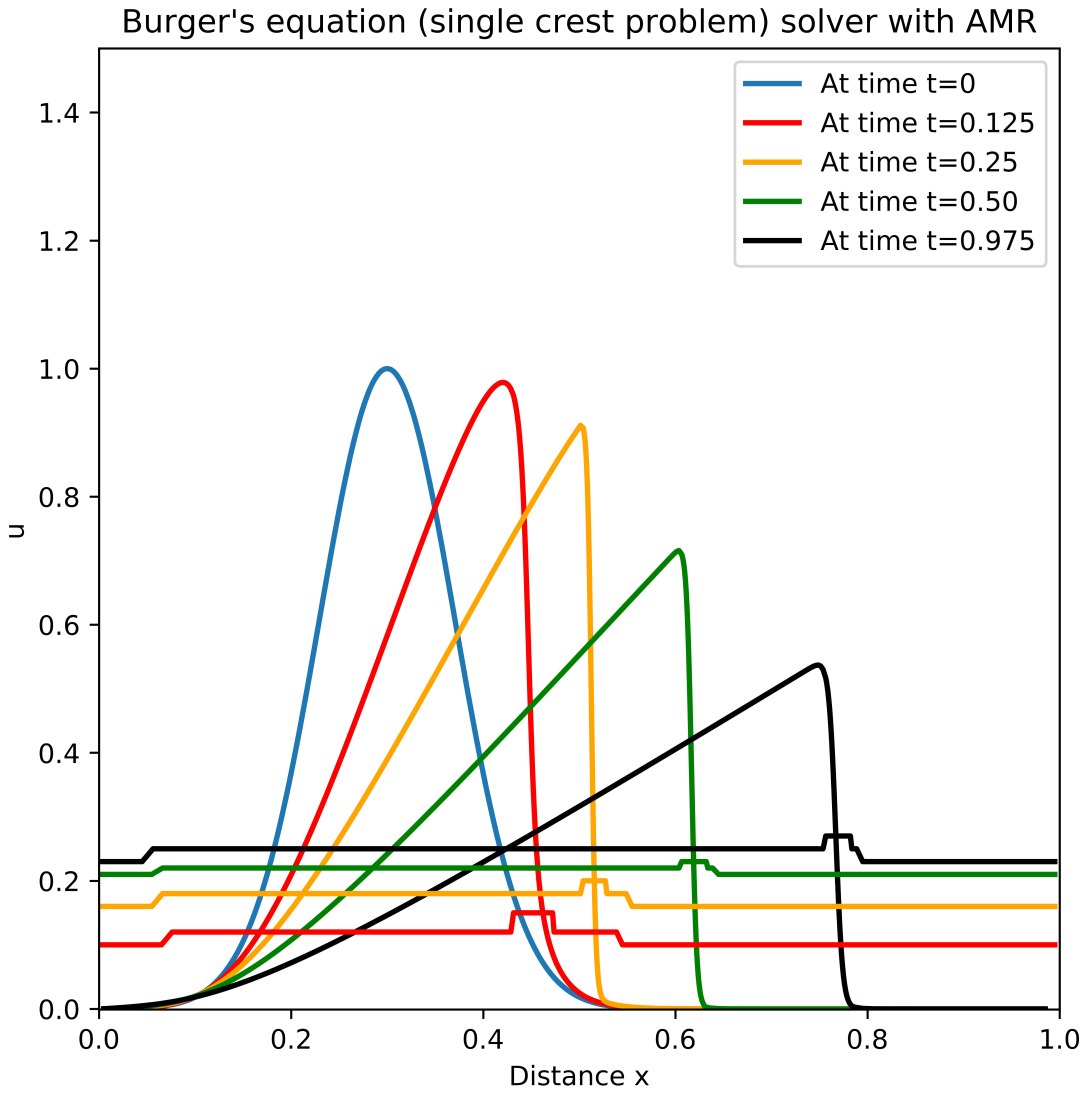


Figure 3.23: Burger's equation (single crest problem) AMR solution with $|u_x| \geq .007$ for level 1 and $|u_x| \geq 5.0$ for level 2 refinement criteria. The step functions indicate the locations of the different level grids with matching colours

Number of grid points in level 0 grid	100
Number of time steps in level 0	1000
Refinement ratio	4
Level 0 grid space width	.01
Level 0 grid time interval	0.001
Maximum level of refinement	2
Level 1 refinement criteria	$\|u\| \geq 0.007$
Level 2 refinement criteria	$\|u_x\| \geq 5.0$
Required computational time	0.078s

Table 3.9: Summary of AMR computation for Burger's equation (single crest problem)

At the time $t = 0.125$ a small level 2 clusters can be seen covering the high gradient area of the solution. The scenario is similar at $t = 0.25$, $t = 0.50$ and $t = 0.975$. However, the level 2 refinement area is very small at time $t = 0.25$, $t = 0.50$ and $t = 0.975$ because high gradient regions are very small at these times which are also shown by figure 3.17. In figure 3.22(a) two level two clusters can be found for small level 2 refinement criterion. The level 1 and level 2 grids are changing dynamically with respect to time over the computational domain and these dynamic changes of the different level grid make the code performance faster and efficient than the uniform grid solution.

In figure 3.24 the AMR solution is compared with the highest resolution single grid solution. In this figure both the highest resolution solution and the solutions achieved by AMR code are plotted at the same time. The highest resolution solution matches with the AMR solution nicely. This validates the claim that AMR solution can produce high resolution solution.

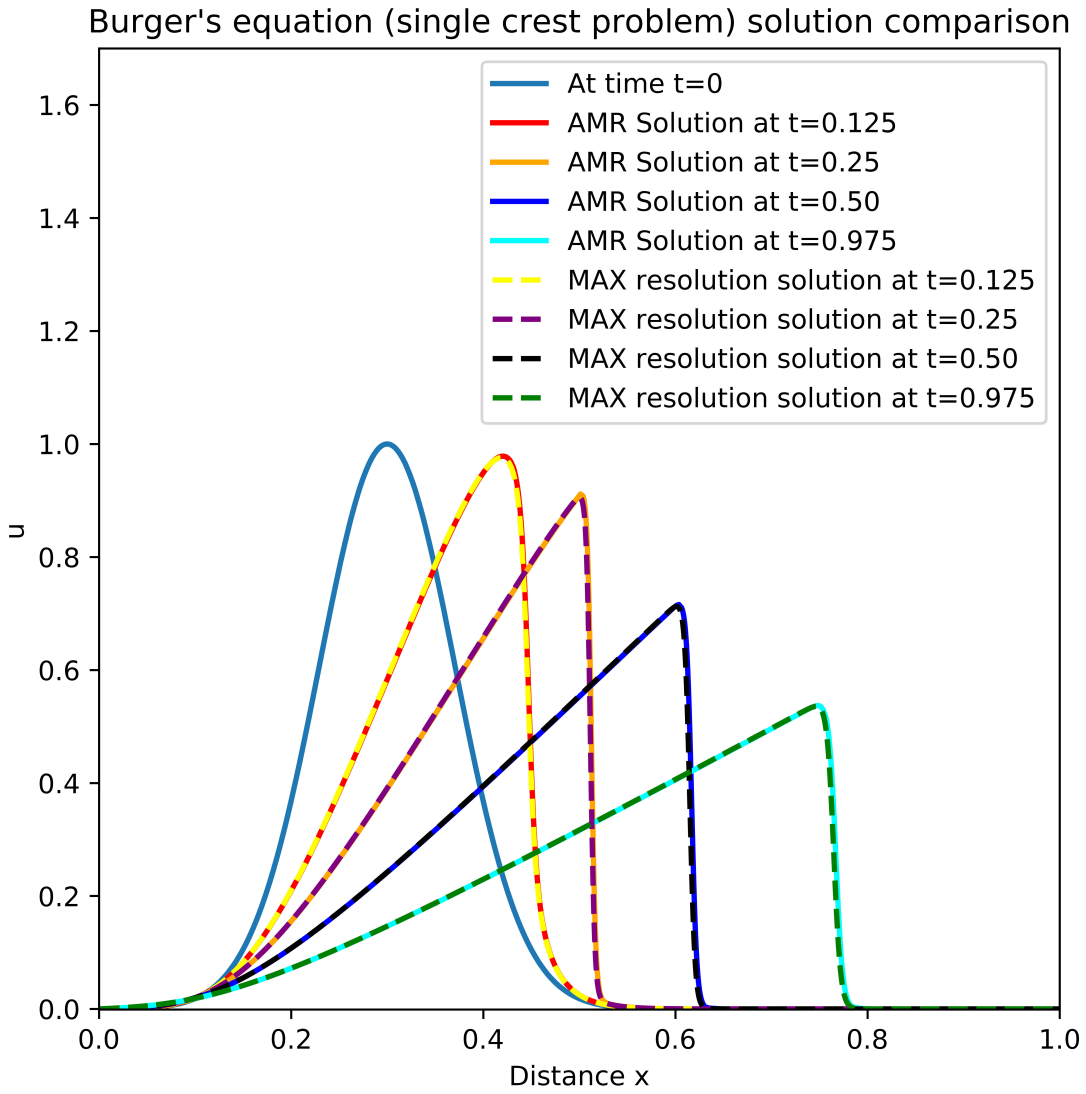


Figure 3.24: AMR solution comparison with highest resolution single grid solution(Burger's equation single crest problem)

It can be seen from table 3.9 that the required time for the AMR computation is 0.078 second which is the average time obtained from 10 simulations. The required time for the highest resolution solution is 0.701 second. So, AMR computation is not as costly as the highest resolution solution. In this case the AMR solution is not faster than the level 0 grid solution or even the level 1 single grid solution (400 grid cells and 4000 time steps).

The solution is more than 8 times faster than the highest resolution single grid solution and produces the finest resolution (1600 grid cells and 16000 time steps) solution. So, still the performance of the AMR code is satisfactory. Finally, the AMR code performance is satisfactory since it is producing highly accurate numerical results with less computational effort. One more point should be noted that the solution will perform better in case of long domain problems. In this experimental case the computational domain is not very long. However, still optimization can be possible to reduce the computational cost. The refinement ratio can be taken small or the value taken as refinement indicator for each level can be increased to get more time efficient solution.

3.2.2 Double crest problem

In this section we are going to demonstrate the performance of our code for double crest problem. This problem is slightly different than the previous problem since the starts with two crests one of them is large and another is small. Steep shock can be seen for both the crest but with time the solution behaves like single crest problem. We are going to investigate these characteristics with our AMR code. For this problem the initial condition is

$$u(x,0) = e^{-120.0(x-0.3)^2} + 0.3e^{-50.0(x-0.6)^2}$$

where $0 \leq x \leq 1$ and $0 \leq t \leq 1$ and the single grid boundary conditions are $u(0,t) = 0 = u(1,t)$ and $\nu = 0.001$ as before.

3.2.2.1 Double crest problem solution on uniform grid

Now our computation starts with two crests. The problem is not too much different than the previous single crest problem. In figure 3.25 the numerical solution with 100 grid cells and 1000 time steps is shown. The discretization scheme is similar to the previous problem. Some expected behaviour of the solution can be observed which we have seen in the previous problem also. From figure 3.25 we can see that the solution is not smooth and it is breaking at the top shortly after time $t = 0.125$. The magnitude of ν is small similar to the previous problem. So, strong nonlinear effects can be visible in the solution. The solution contains steep shocks that can not be resolved by the coarse grid.

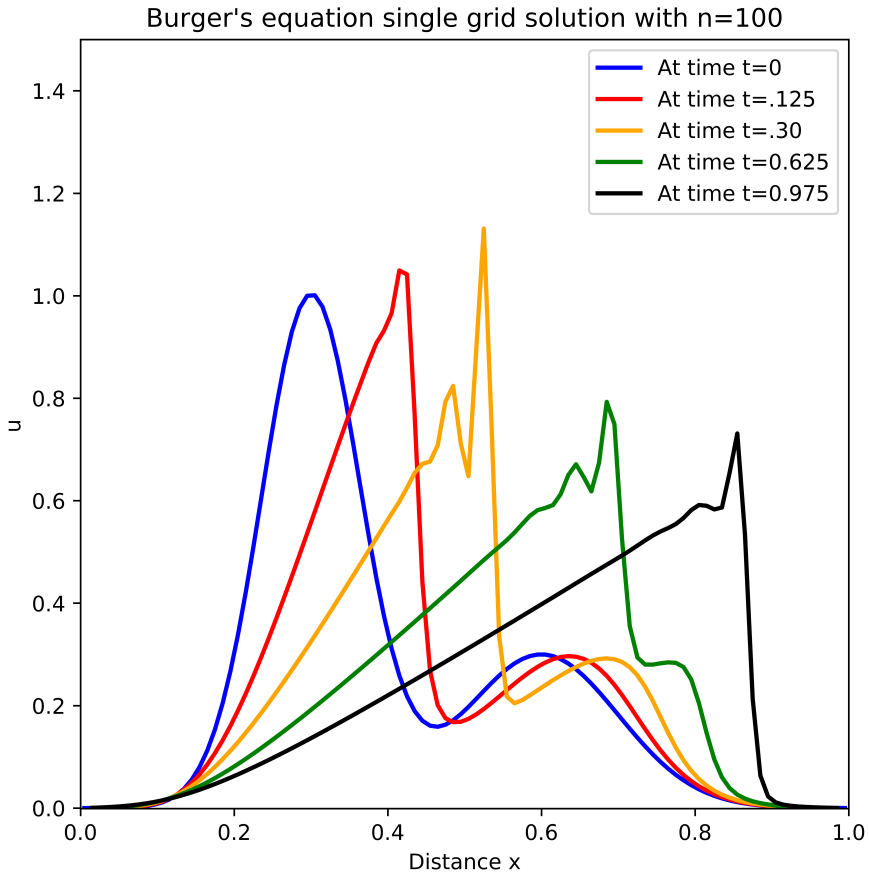


Figure 3.25: Burger's equation single grid solution with n=100 (double crest problem)

The gradient at different portion at different time is shown in figure 3.26. From this figure it can be observed that the maximum magnitude of the absolute value of the gradient is not static. At time $t = 0.125$ the magnitude was around 50 which becomes greater than 60 at time $t = 0.3$ which again reduces around 30 at $t = 0.625$. However, at $t = 0.975$ it rises again. So, to cover the area where the gradient is changing over time in different portions of the computational space we are going to put the finest resolution in the high gradient areas.

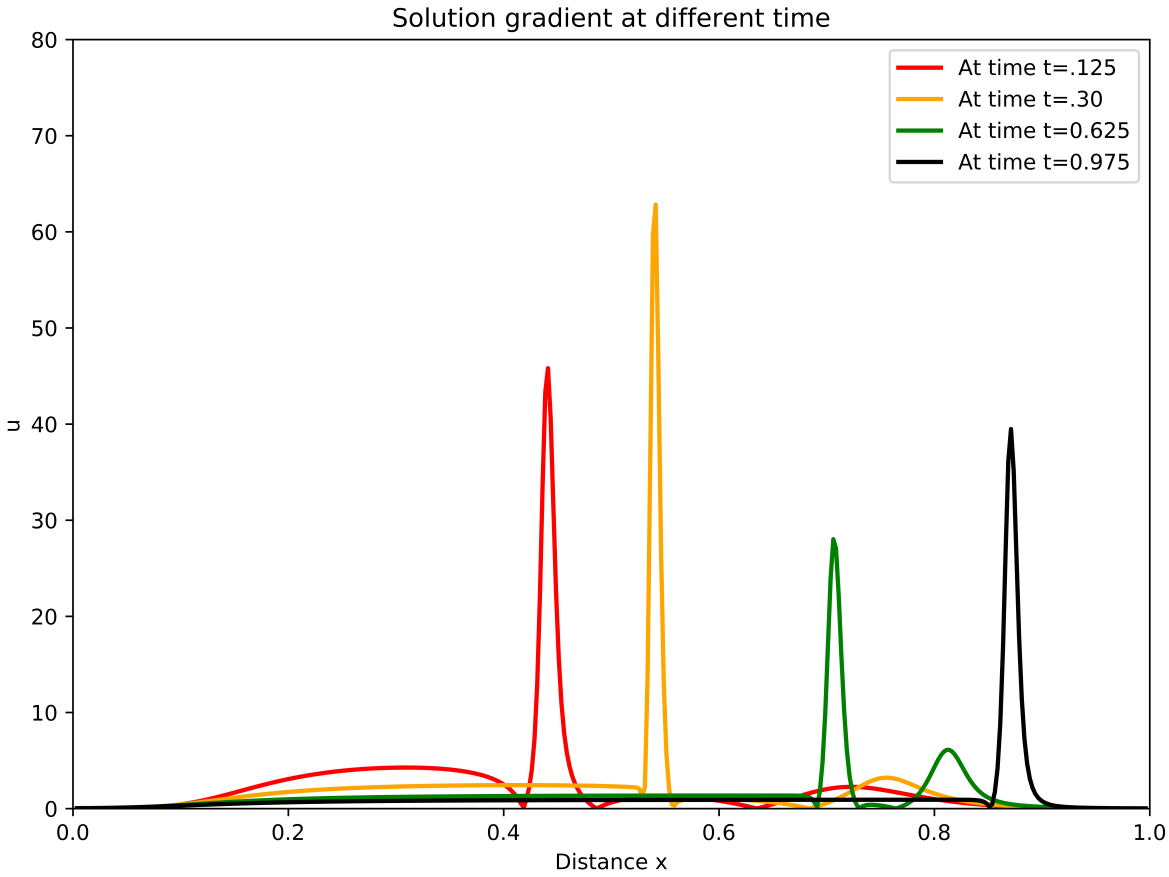


Figure 3.26: Gradient at different time (Burger's equation double crest problem)

We also ran our code with 400 grid cells with 4000 time steps and 1600 grid cells with 16000 time steps. The solutions have plotted in the figure 3.27 and figure 3.28 respectively. The accuracy of the solution increased a lot. The solution with the finest resolution is very smooth as expected.

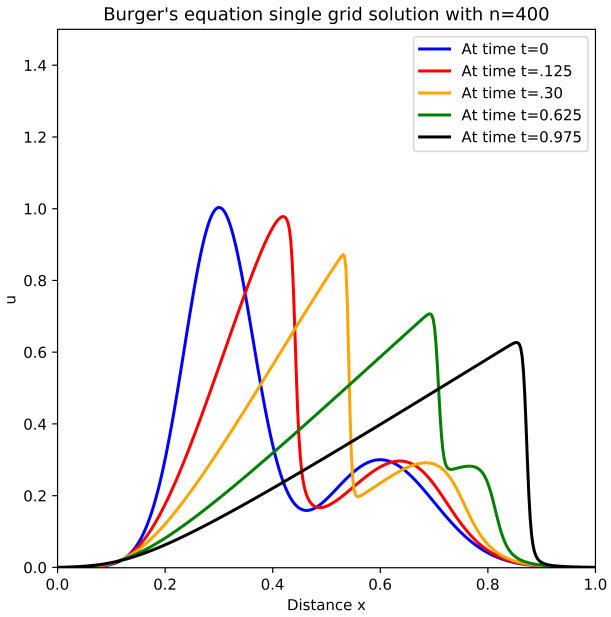


Figure 3.27: Burger's equation single grid solution with n=400 (double crest problem)

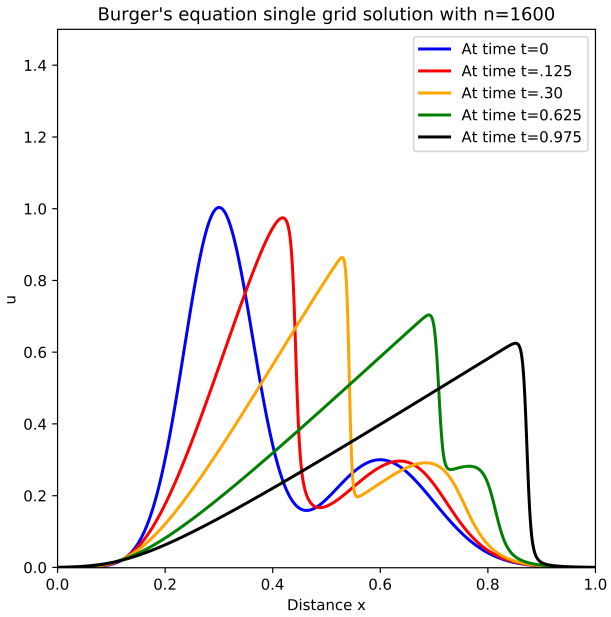


Figure 3.28: Burger's equation single grid solution with n=1600 (double crest problem)

The highly accurate solution is costly. The time changing is shown in figure 3.29. We can see that it follows a similar pattern as for the single crest problem. The details of three of the simulations are given on table 3.10.

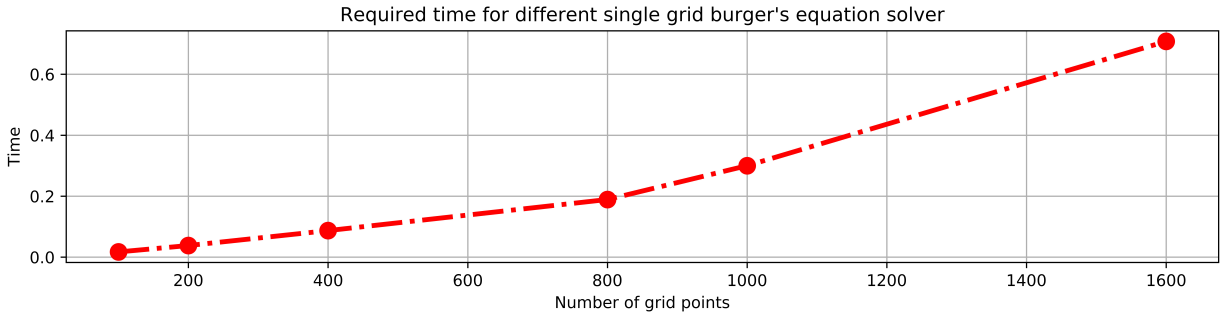


Figure 3.29: Required time for different single grid solution (Burger's equation double crest problem)

Number of grid Points	**Time steps**	Δx	Δt	**Required time(s)**
100	1000	0.01	.001	0.017
400	4000	0.0025	0.00025	0.087
1600	16000	0.000625	6.25×10^{-5}	0.708

Table 3.10: Required computational time for different grid resolution for Burger's equation(double crest problem)

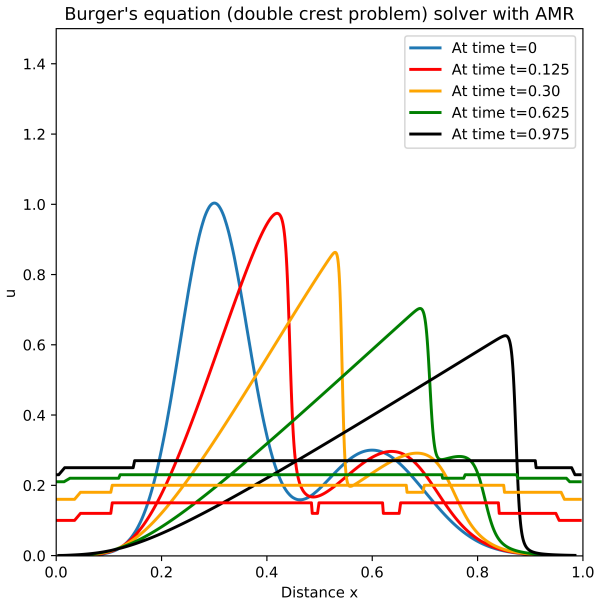
3.2.2.2 Burger's equation (double crest) solution with AMR

Figure 3.32 depicts the AMR solution for the double crest problem. The solution is smooth and visually accurate. We take 100 grid cells and 1000 time steps for the level 0 computation. The refinement ratio is 4. The refinement criteria for level 1 is the same as the single crest problem. We have done some similar analysis to choose the refinement criteria. The idea for level 1 refinement criterion is similar to the other problems. The gradient is also very high for this problem and we have two crests in this case. So for getting the desired results we need to consider the change for both the crests. By observing figure 3.26 we choose to keep the gradient value greater than 0.5 for level 2 refinement criterion to start our simulations. The summary of the simulation run with different refinement criteria is shown in table 3.11.

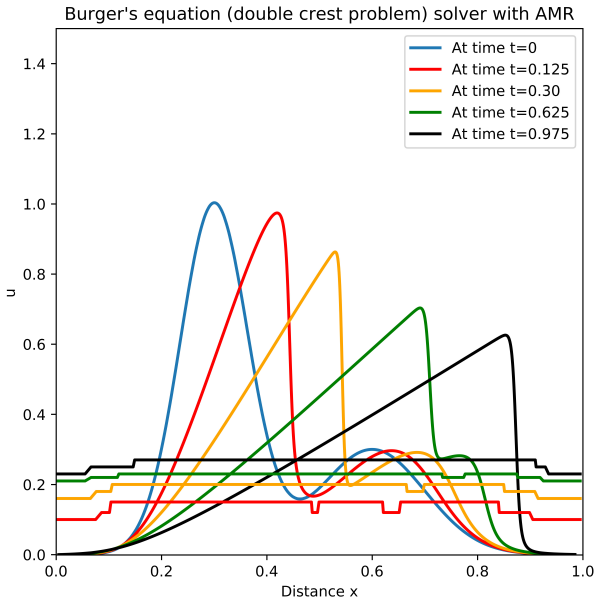
Level 1 refinement criterion	Level 2 refinement criterion	Required time(s)
$\|u\| \geq 0.001$	$\|u_x\| \geq 0.5$	0.472
$\|u\| \geq 0.005$	$\|u_x\| \geq 0.5$	0.460
$\|u\| \geq 0.005$	$\|u_x\| \geq 3.5$	0.125
$\|u\| \geq 0.005$	$\|u_x\| \geq 4.5$	0.111

Table 3.11: Required time for different refinement criteria (Burger's equation double crest problem)

From table 3.11 it can be seen that the slight increase of level 1 refinement is not very time effective while increasing the value of the gradient criterion makes a noticeable difference in the required time. The solution with $|u| \geq 0.001$ as the level 1 refinement criterion and $|u| \geq 0.005$ as the level 1 refinement criterion are shown in figure 3.30(a) and figure 3.30(b) respectively where the level 2 gradient refinement criterion ($|u_x| \geq 0.5$) is same for both figures. Figure 3.31 shows the AMR solution for two different level 2 criterion ($|u_x| \geq 3.5$ and $|u_x| \geq 4.5$) with the level 1 refinement criterion fixed at $|u| \geq 0.005$. The results are similar and for the comparison and further discussion we choose $|u| \geq 0.005$ and $|u_x| \geq 5.0$ as our level 1 and level 2 refinement criterion respectively for further discussion and comparison and the AMR solution with this set up is shown in figure 3.32. The summary of the AMR solution is given in table 3.12. It can be seen from figure 3.32 that at time $t = 0.125$ there is only 1 cluster in level 2 contained in level 1. The number of cluster is remain same at time $t = 0.30$ and $t = 0.975$. It is also noticeable that the size of the level 2 cluster is very small compared to the solution with large level 2 refinement criterion. At time $t = 0.625$ we can see two level 2 cluster. The AMR solutions with different values of refinement criteria are shown in figure 3.30 and 3.31 where we can see multiple large level 2 clusters at different times. In the end the two crests disappear and we get a similar shape which we achieved for the single crest problem and it can be seen that a very small portion of the solution is covered by the level 2 computation while the level 1 grid covers a large portion of the computational domain.

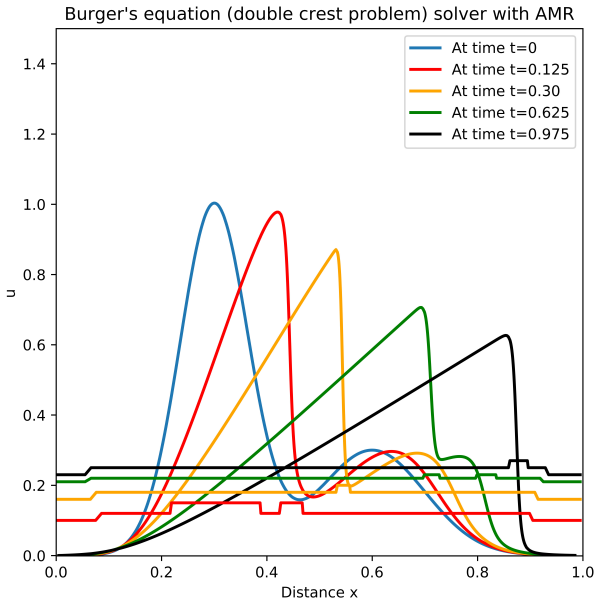


(a)

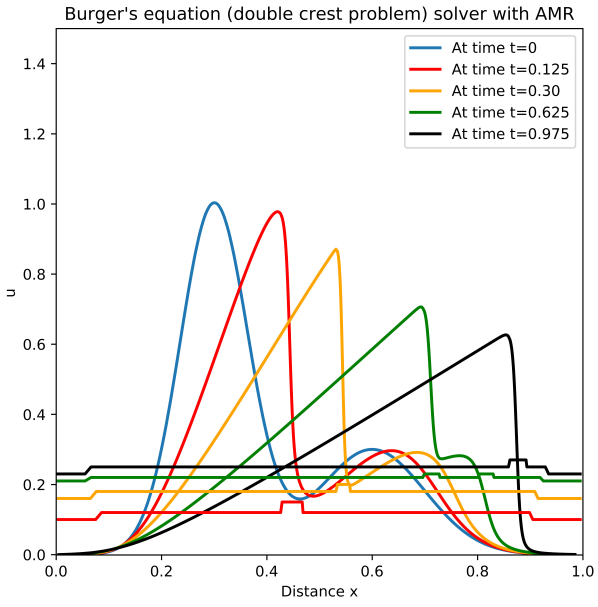


(b)

Figure 3.30: AMR solution for Burger's equation(double crest problem) with (a) $|u| \geq 0.001$ and (b) $|u| \geq 0.005$ as the level 1 refinement criterion and $|u_x| \geq 0.5$ as the level 2 refinement criterion for both cases. The step functions indicate the locations of the different level grids with matching colours



(a)



(b)

Figure 3.31: AMR solution for Burger's equation(double crest problem) with (a) $|u_x| \geq 3.5$ and (b) $|u| \geq 4.5$ as the level 2 refinement criterion and $|u| \geq 0.005$ as the level 1 refinement criterion for both cases. The step functions indicate the locations of the different level grids with matching colours

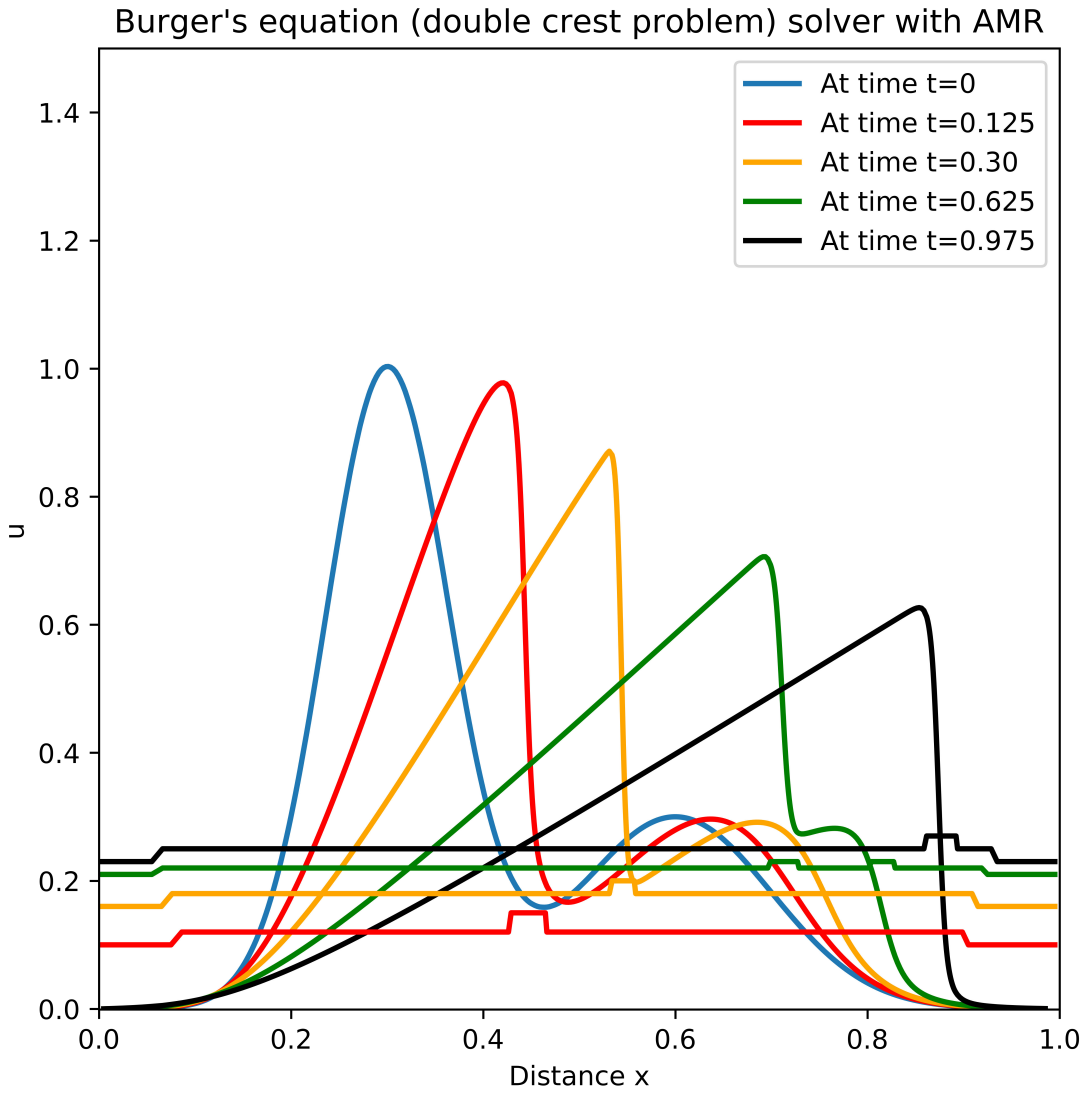


Figure 3.32: AMR solution for Burger's equation (double crest problem) and with $|u| \geq 0.005$ as the level 1 and $|u_x| \geq 5.0$ as the level 2 refinement criterion. The step functions indicate the locations of the different level grids with matching colours

Number of grid cells in level 0 grid	100
Number of time steps in level 0	1000
Refinement ratio	4
Level 0 grid space width	.01
Level 0 grid time interval	0.001
Maximum level of refinement	2
Level 1 refinement criteria	$\|u\| \geq 0.005$
Level 2 refinement criteria	$\|u_x\| \geq 5.0$
Required computational time	0.109s

Table 3.12: Summary of AMR computation for burger's Equation (double crest problem)

The highest resolution uniform grid and AMR solution (with $|u| \geq 0.005$ as the level 1 and $|u_x| \geq 5.0$ as the level 2 refinement criteria) are compared in figure 3.33. The highest resolution solutions agrees with the AMR solution. We can see that the computational cost reduced a lot here since the required time 6 times faster than the highest resolution single grid solution. The performance of the AMR code for this problem is similar to the single crest problem. The performance here can also be increased by modifying the refinement criteria or the refinement ratio. But the overall performance of the code is satisfactory for solving Burger's equation.

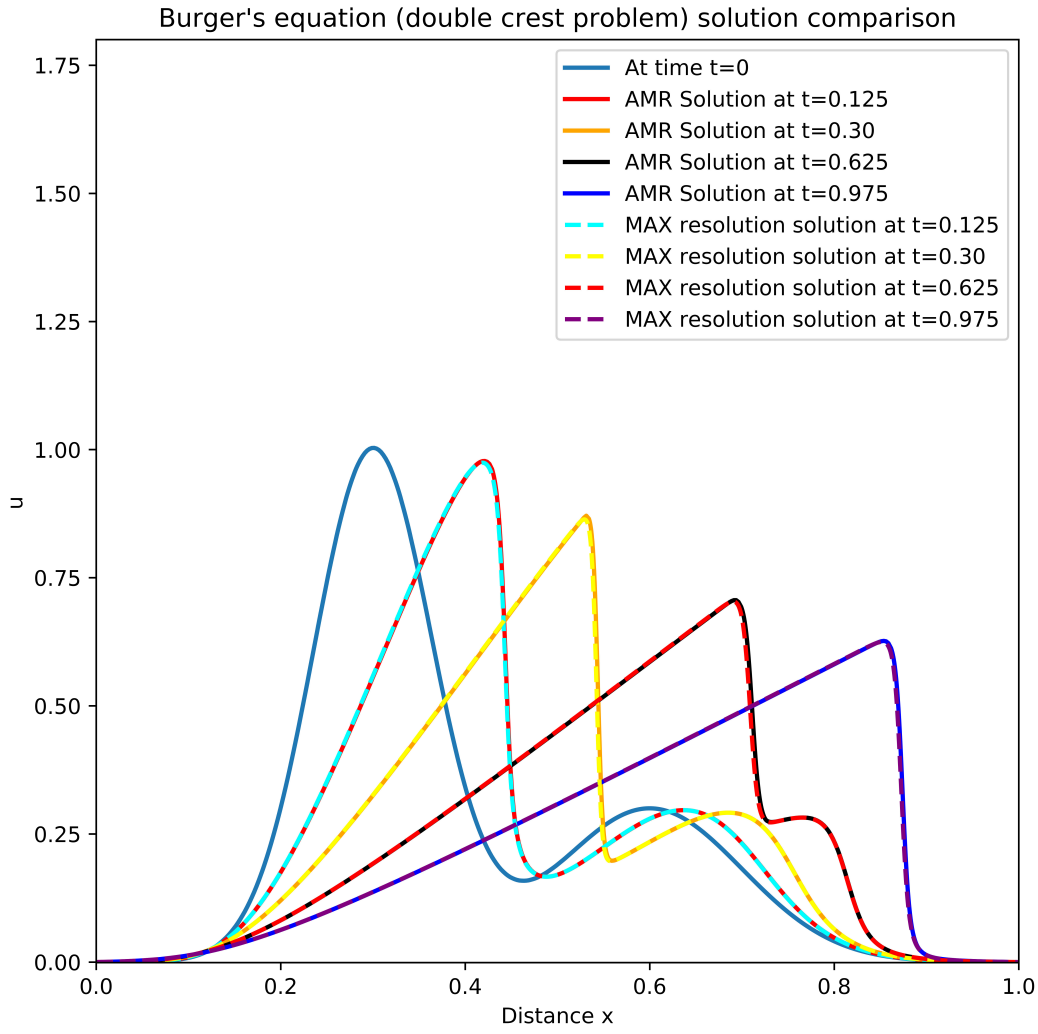


Figure 3.33: AMR solution comparison with highest resolution solution (Burger's equation double crest problem)

3.3 Solution of Regularized Long Wave (RLW) equation

In this section numerical simulations of the interesting Regularized Long Wave (RLW) equation using AMR code will be discussed. The RLW equation is an alternative form of the KdV equation. In the study of nonlinear and dispersive wave equation it is used to describe some important phenomena such as shallow water waves or ion-acoustic plasma waves. However, an analytic solution of the RLW equation can be achieved only for restricted initial and boundary conditions. So, numerical solutions are often required. The RLW equation is

$$u_t + u_x + uu_x - u_{txx} = 0$$

If x is the spatial variable then the for $x_a \leq x \leq x_b$ the boundary conditions are

$$u(x_a, t) = 0$$

$$u(x_b, t) = 0$$

for $0 \leq t \leq T_{final}$. So, we will get some solitary wave solutions and the AMR code will be used for the simulation of RLW equation. Two cases will be investigated in this section. For the first case we will simulate the propagation of the solitary wave.

3.3.1 Propagation of single solitary wave

The theoretical solitary wave solutions of the RLW equation have the form

$$u(x,t) = 3C \operatorname{sech}^2\big(k(x - x_0 - vt)\big) \tag{3.3}$$

where $v = 1 + C$ is the propagation speed of the solitary wave, $3C$ is its amplitude and x_0 is the location of the peak at $t = 0$. The width of the wave is determined by $k = \frac{1}{2}\sqrt{\frac{C}{1+C}}$. We use this as our initial wave at $t = 0$. As we mentioned earlier about the properties of the solitary-wave and this wave will propagate from left to right with a constant phase speed over time in the solution interval $[x_a, x_b]$. For this problem the solution domain is $-30.0 \leq x \leq 100.0$ for the time period $0 \leq t \leq 40$ with parameter $C = \frac{1}{3}$. In figure 3.34 the theoretical solitary wave solution is shown.

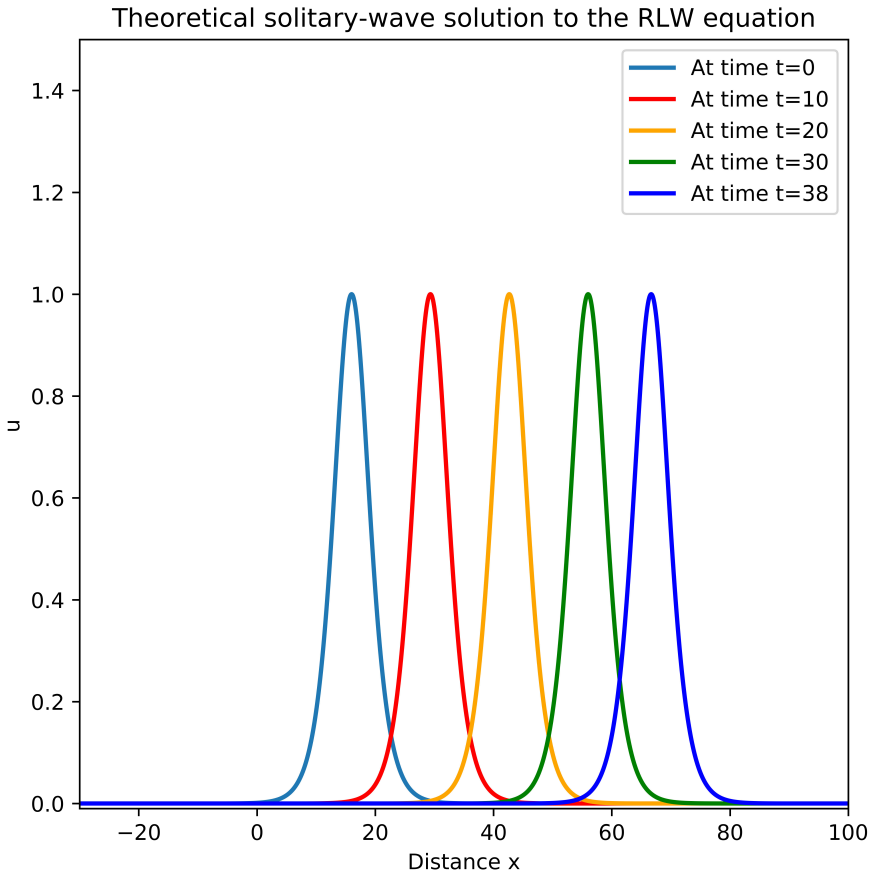


Figure 3.34: Theoretical solution of solitary wave solution

3.3.1.1 Solitary wave solution on uniform grid

In figure 3.34 the solutions are plotted at several times. It can be seen that the wave is propagating without changing shape or amplitude over time. The amplitude is 1.0 for all time. This property was observed in the case advection equation however in this case the propagation speed v depends on the wave amplitude. The balance between the nonlinearity and dispersion makes it possible for the wave to maintain its shape. Similar behaviour is expected from the numerical solution of the RLW equation with our initial condition. The discretization scheme derived earlier is used for all the numerical simulations with or without AMR. It was similar to the approach of [17].

In figure 3.35 the solution with 200 grid points and 2000 time steps is shown. The

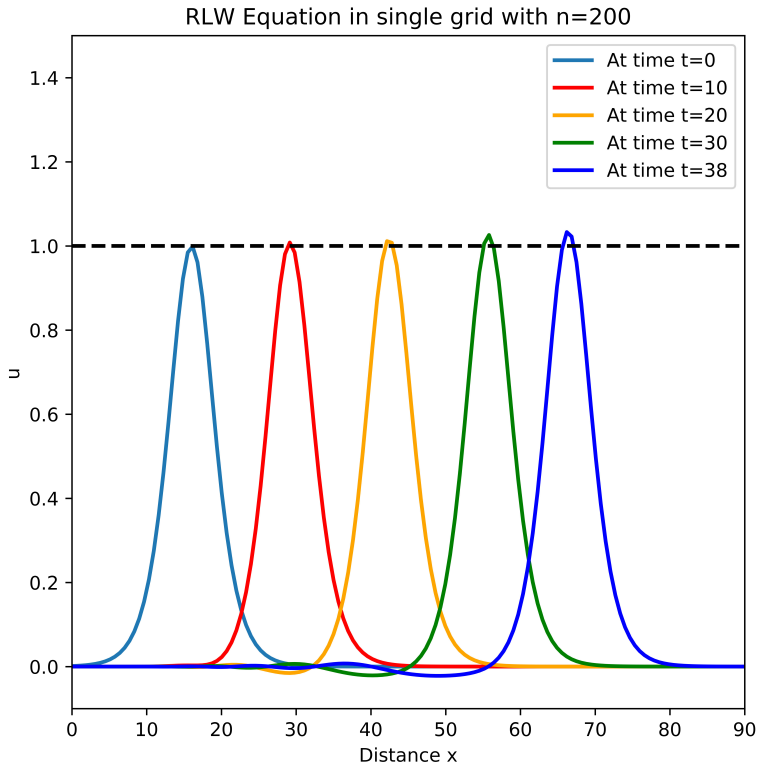


Figure 3.35: Numerical solution of RLW equation (solitary wave propagation) with n=200

amplitude increases with time and to the left of the wave the values of u become negative as can be seen at times $t = 30$ and $t = 38$. In figure 3.36 the error plot is shown. The formula mentioned below is used for the calculation of the errors.

$$error = \mid u_{analytic} - u_{approximate} \mid$$

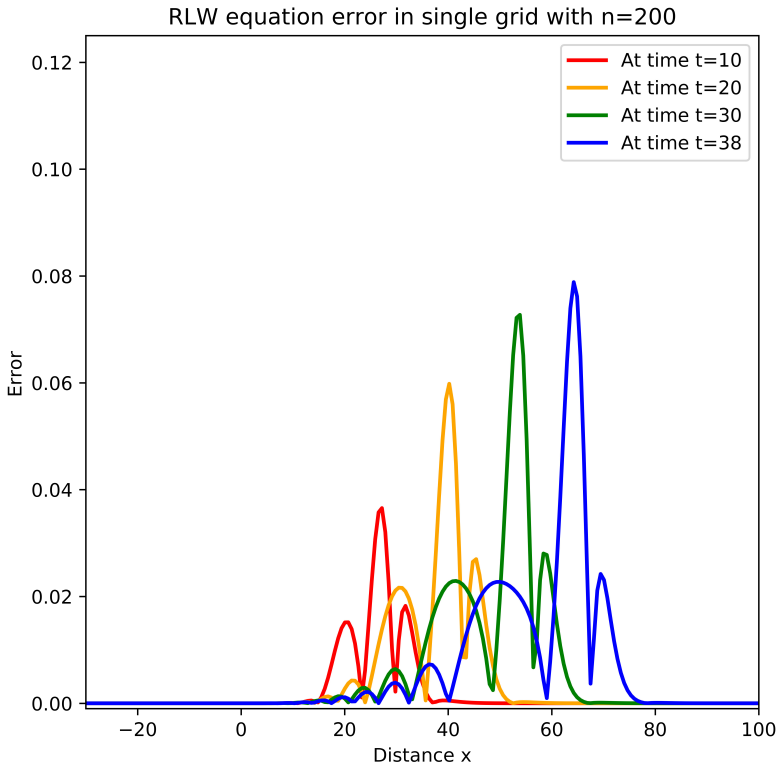
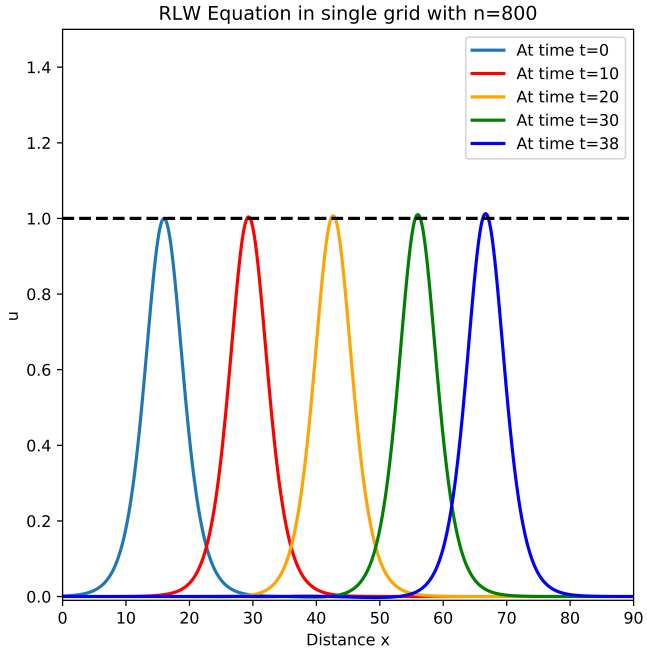
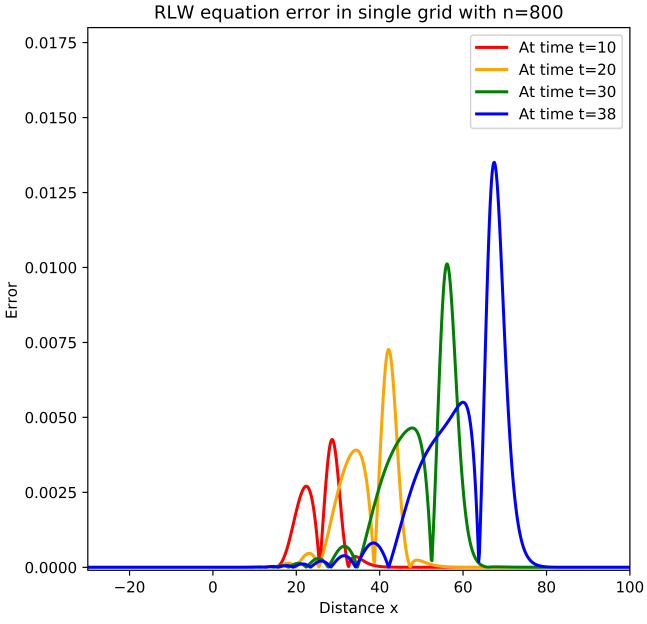


Figure 3.36: RLW equation (solitary wave propagation) solution error in single grid with n=200

From figure 3.36 an error pattern can be seen. The error increases with time and the pattern does not remain exactly the same for all the time. The error pattern at $t = 10$ and $t = 38$ are different. In the next two simulations the resolution is refined. In figure 3.37 the solution and error with 800 grid points and 8000 time steps are shown. Similarly, in figure 3.38 the solution and error with 3200 grid points and 32000 time steps are shown.

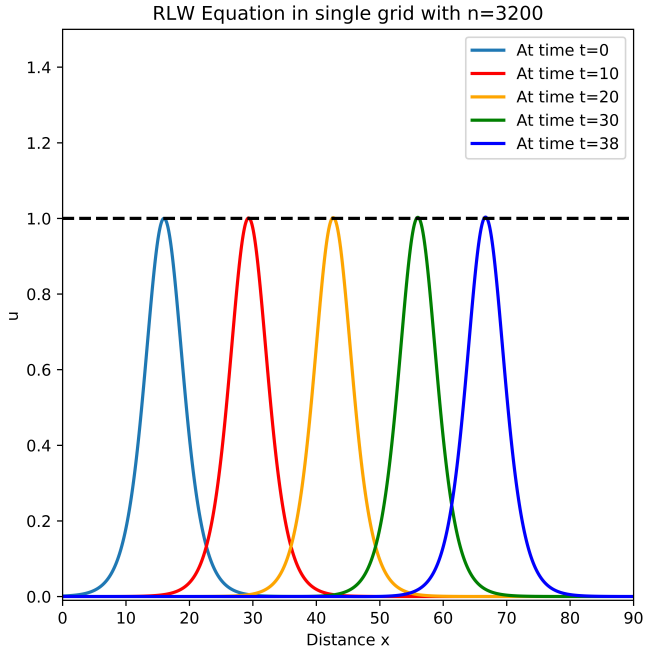


(a)

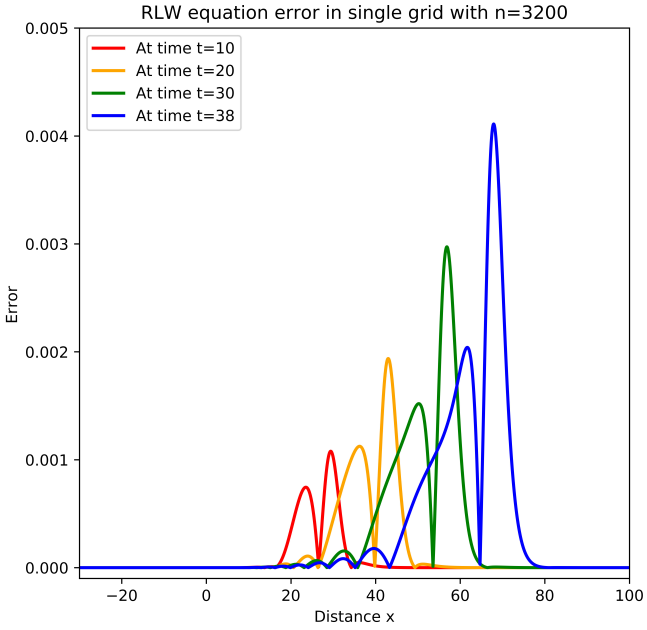


(b)

Figure 3.37: Numerical solution of RLW equation (solitary wave propagation) with (a) 800 grid points (b) Solution error in single grid with n=800



(a)



(b)

Figure 3.38: Numerical solution of RLW equation (solitary wave propagation) with (a) 3200 grid points (b) Solution error in single grid with n=3200

In figure 3.37 it can be clearly observed the improvement of the quality of solution due to the error reduction. The error pattern remain the same. However, the maximum magnitude of error at time $t = 38$ is below 0.02 (the scale is different in error plots 3.36-3.38). The solution has improved significantly with 3200 grid points and 32000 time steps. The maximum error at time $t = 38$ is below 0.005 so the numerical solution is very close to the theoretical solution. In figure 3.35 the amplitude has crossed the line $y = 1$ at time $t = 38$ and this scenario has improved with high resolution solution. So, we have a good improvement with increased resolution. This solution improvement has increased computational time. The required time as a function of the number of grid shown in figure 3.39. However, required times have increased significantly with finer resolution and the increment is expected with little bit fluctuation. The time is increasing between two consecutive grid is proportional to their ratio. Detailed information is presented for three simulations in table 3.13.

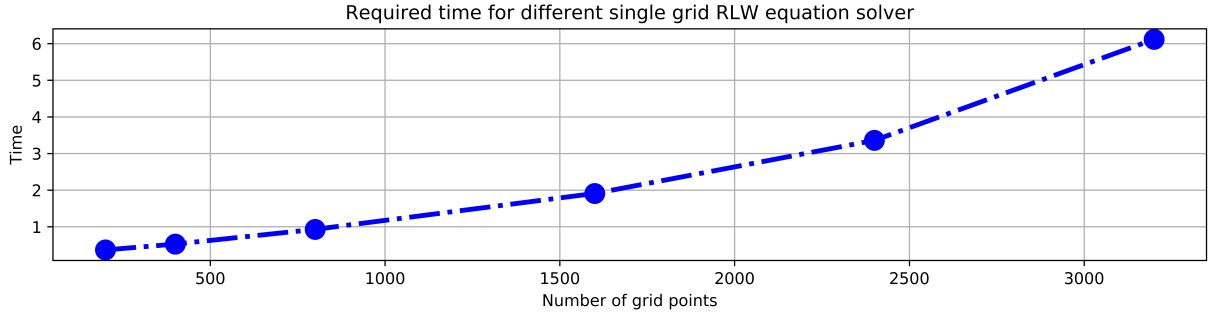


Figure 3.39: Required time for the different single grid solution (solitary wave propagation)

Number of grid Points	**Time steps**	Δx	Δt	**Required time(s)**
200	2000	0.65	0.02	0.368
800	8000	0.1625	0.005	0.930
3200	32000	0.040625	0.00125	6.117

Table 3.13: Required computational time for different grid resolution for RLW equation (solitary wave propagation)

3.3.1.2 Single solitary wave solution with AMR

In this section we will present the AMR solutions. 200 grid points and 2000 time steps were used on the level 0 grid. The refinement ratio is 4 and as before the refinement is done for both time and space. The geometric property (solution cut-off value) is taken as refinement criteria. Some computations were run using different values of refinement criteria. In figure 3.40 the AMR solution with $|u| \geq 0.005$ as the level 1 refinement criterion and $|u| \geq 0.40$ as the level 2 refinement criterion is shown. The results with the same level 2 refinement criterion but increasing the level 1 refinement criterion to $|u| \geq 0.01$ is shown in figure 3.43. The results are similar and the time reduction is not very significant. Now keeping level 1 refinement criterion $|u| \geq 0.01$ the AMR solutions using $|u| \geq 0.45$ and $|u| \geq 0.50$ as the level 2 refinement criteria are shown in figures 3.41 and 3.42 respectively. However, the required time has not been reduced significantly. So, for this problem taking $|u| \geq 0.01$ for level 1 refinement criterion and $|u| \geq 0.4$ for level 2 refinement criterion for further discussion and comparison since we have obtained better performance than on the level 1 and level 2 single grids with this setup. Table 3.14 represents the summary of the computations with different criteria.

Level 1 refinement criterion	**Level 2 refinement criterion**	**Required time(s)**
$\|u\| \geq 0.005$	$\|u\| \geq 0.40$	0.786
$\|u\| \geq 0.01$	$\|u\| \geq 0.40$	0.769
$\|u\| \geq 0.01$	$\|u\| \geq 0.45$	0.713
$\|u\| \geq 0.01$	$\|u\| \geq 0.50$	0.688

Table 3.14: Required time for different refinement criteria (RLW equation solitary wave propagation problem)

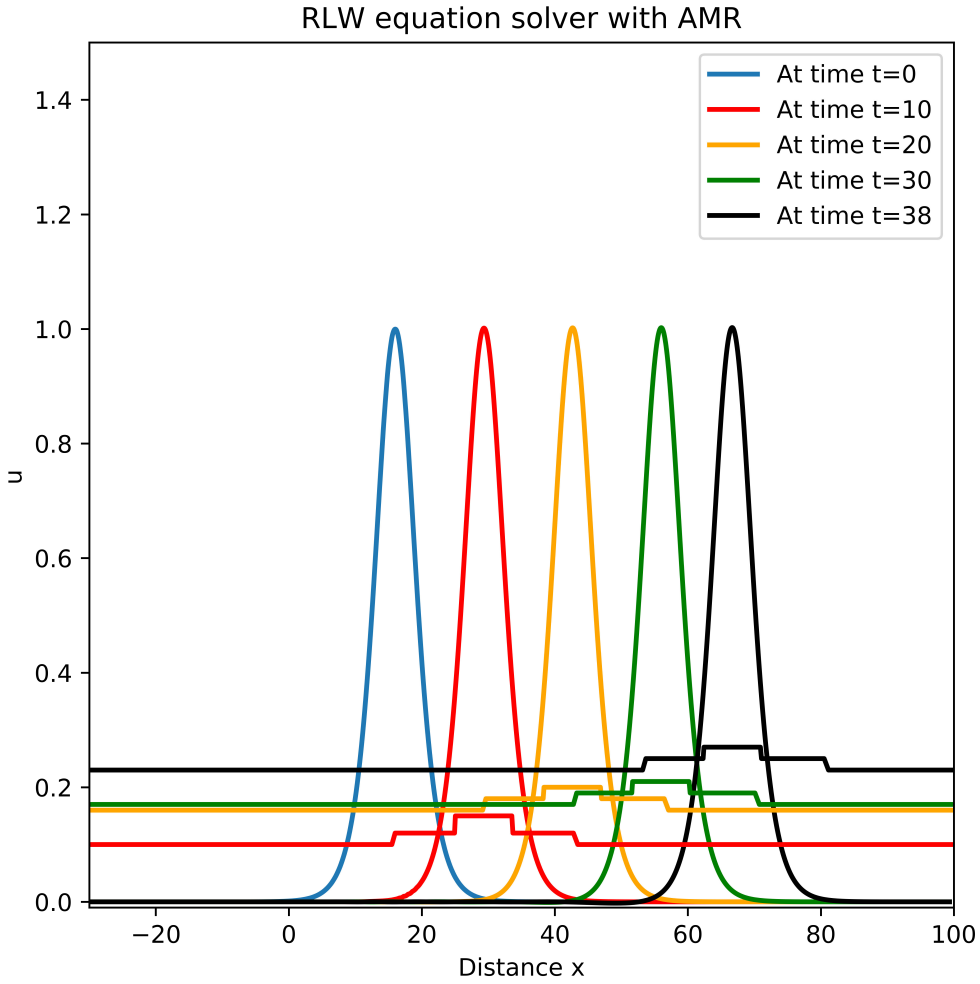


Figure 3.40: AMR solution for RLW equation for the propagation of a single solitary wave with $|u| \geq .005$ as the level 1 refinement criterion and $|u| \geq 0.40$ as the level 2 refinement criterion. The step functions indicate the locations of the different level grids with matching colours

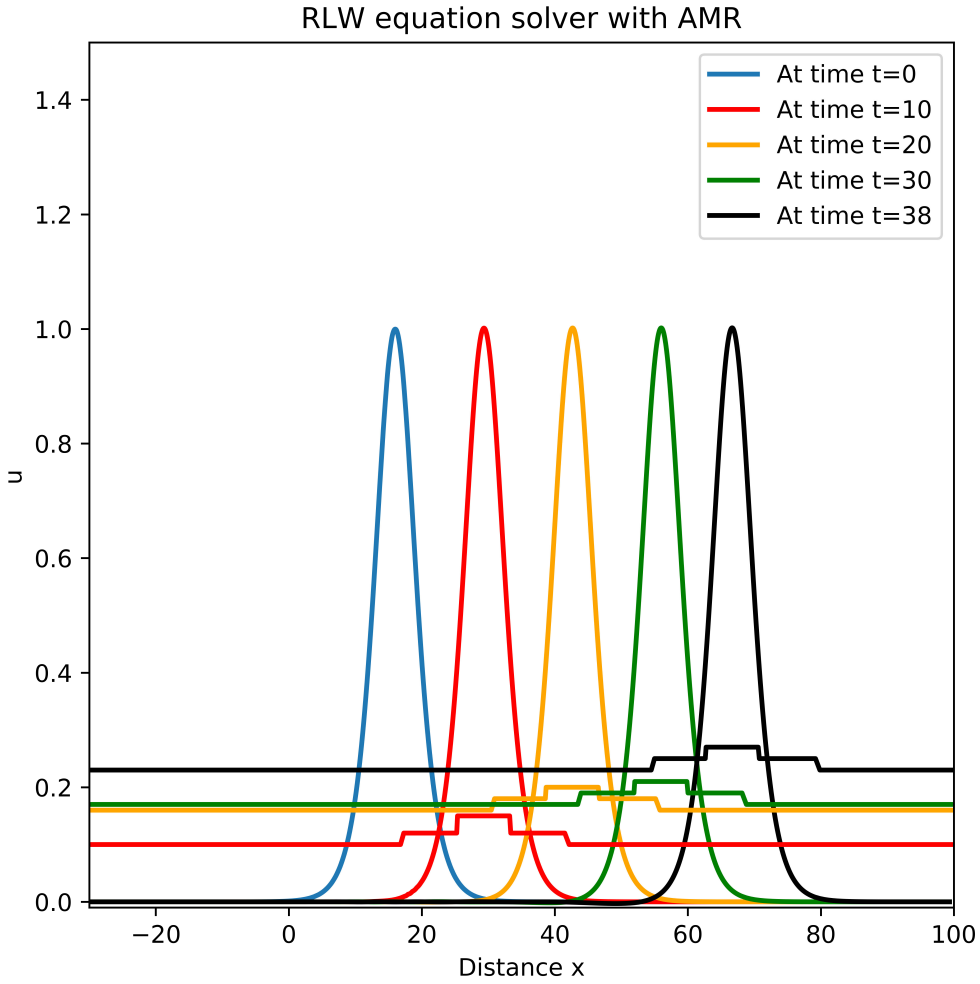


Figure 3.41: AMR solution for RLW equation for the propagation of a single solitary wave with $|u| \geq .01$ as the level 1 refinement criterion and $|u| \geq 0.45$ as the level 2 refinement criterion. The step functions indicate the locations of the different level grids with matching colours

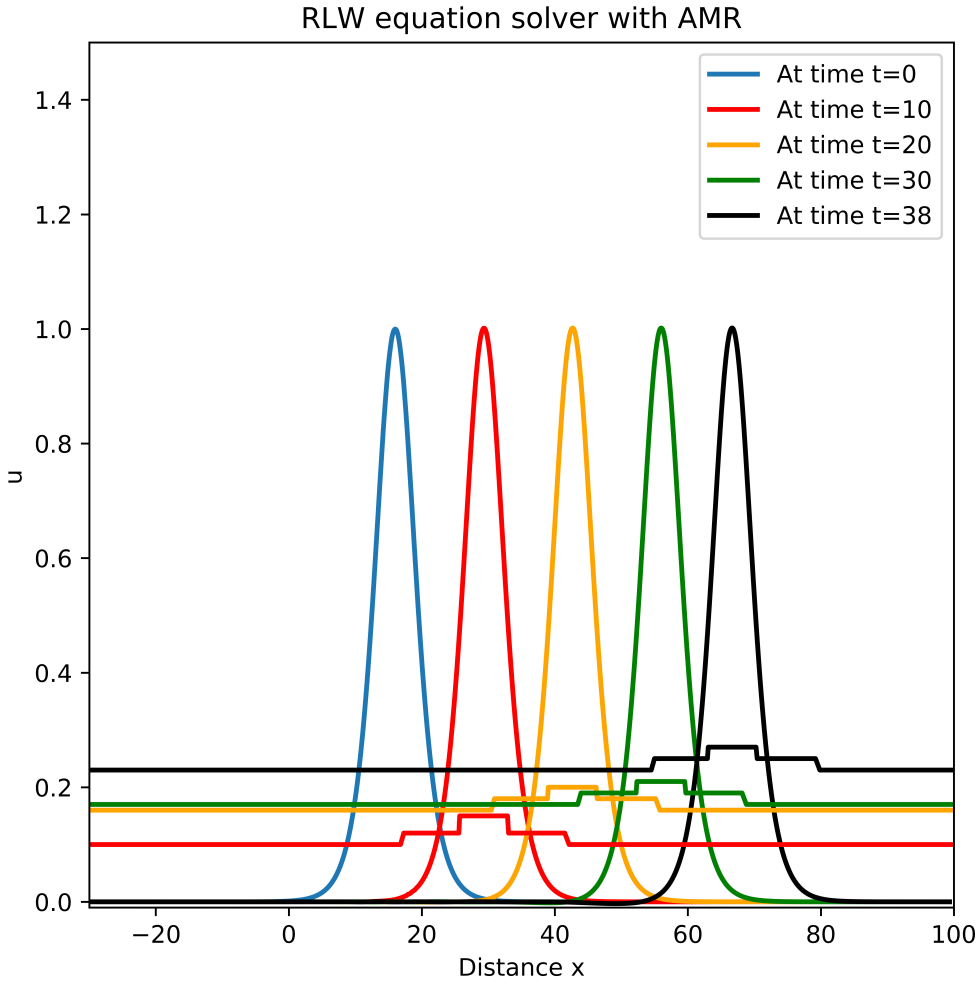


Figure 3.42: AMR solution for RLW equation for the propagation of a single solitary wave with $|u| \geq .01$ as the level 1 refinement criterion and $|u| \geq 0.5$ as the level 2 refinement criterion. The step functions indicate the locations of the different level grids with matching colours

The summary of the AMR computation is shown in table 3.15.

Number of grid points in Level 0	200
Number of time steps in Level 0	2000
Refinement ratio	4
Level 0 grid space width	0.45
Level 0 grid time interval	0.02
Maximum level of refinement	2
Level 1 refinement criteria	$\|u\| \geq 0.01$
Level 2 refinement criteria	$\|u\| \geq 0.40$
Required computational time	0.769s

Table 3.15: Summary of AMR computation for RLW equation (solitary wave propagation problem)

In figure 3.43 the AMR solutions are plotted at times $t = 10$, $t = 20$, $t = 30$ and $t = 38$ as we did for the single grid solution. The level 1 refinement criterion is taken very small ($|u| \geq 0.01$). So, the computation starts with 200 grids and 2000 time steps and refinement is made in time and space with a factor of 4 where the level 1 refinement criterion is met. So, where the level 1 refinement criterion is met the algorithm computes the solution with $\Delta x = 0.1125$ and $\Delta t = 0.005$ for that region. After completing the computation in level 1 the solution algorithm moves to level 2 and again if the refinement criteria is met then the algorithm computes the solution with $\Delta x = 0.028125$ and $\Delta t = 0.00125$ for that region. The grid hierarchy can be found in the AMR solution described in the algorithm. In figure 3.43 it has shown that the level 0 grid solution contains level 1 grid solution and similarly all level 2 solution is contained by the level 1 solution. So, the grid hierarchy maintained properly.

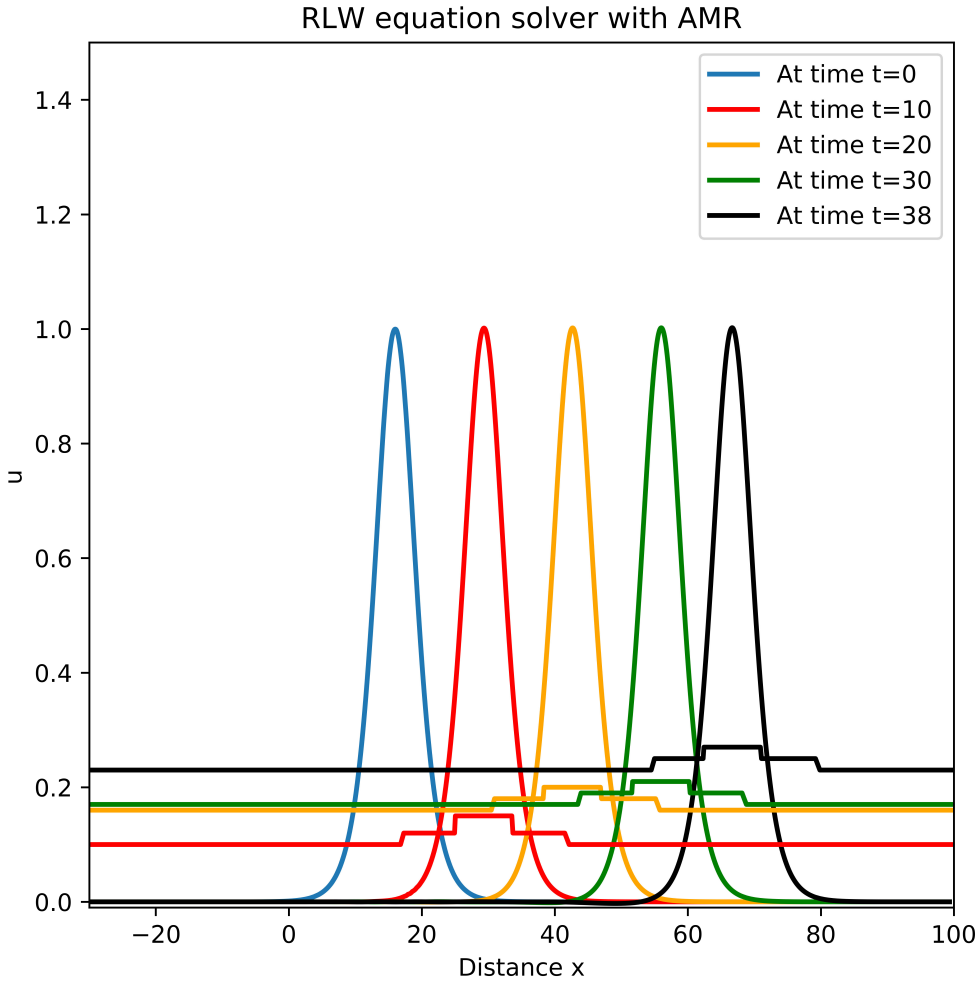


Figure 3.43: AMR solution for RLW equation for the propagation of a single solitary wave with $|u| \geq .01$ as the level 1 refinement criterion and $|u| \geq 0.4$ as the level 2 refinement criterion. The step functions indicate the locations of the different level grids with matching colours

There is no significant change of amplitude in the propagation of the solitary wave solution. In figure 3.44 it can be seen that both the AMR solution and the theoretical solution have agrees. So, the desired level of accuracy has been obtained by the AMR code.

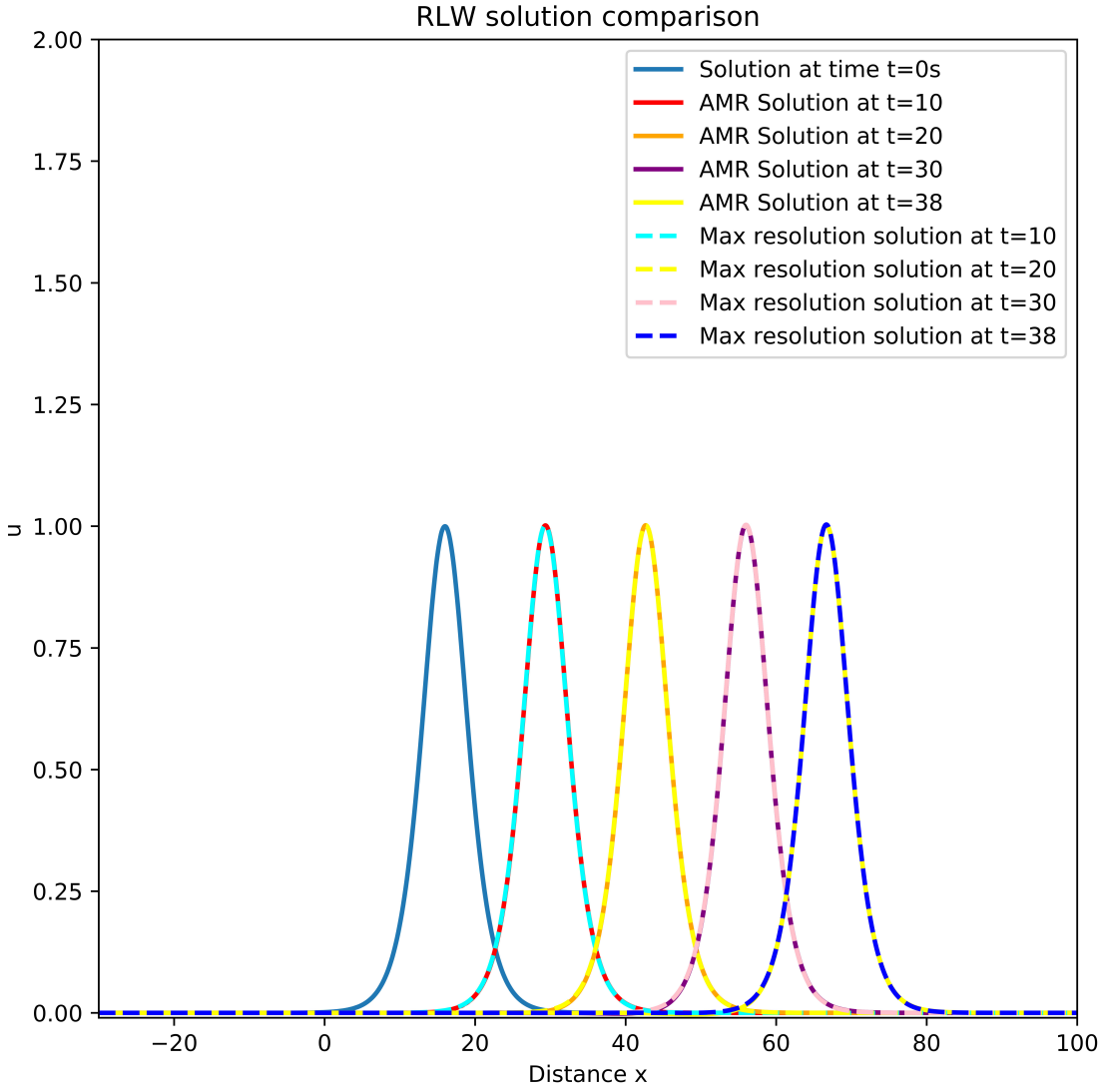


Figure 3.44: RLW single solitary wave propagation problem AMR and finest resolution solution comparison

In table 3.15 it is mentioned that the computational time for AMR solution is only 0.769 second which is very good compared to the highest resolution single grid solutions. The AMR solution is more than 7 times faster than the single grid solution with 3200 grid points and 32000 time steps indicating that the highest level of accuracy was obtained with less computational effort. The most interesting fact that the solution is faster than

the level 1 single grid solution. The Level 1 single grid AMR solution took 0.930 seconds which slightly higher than the time required for AMR solution. So, for the solitary wave problem the AMR code produces faster and better solution which is significant success of the work of this thesis.

3.3.2 Solitary wave fissioning problem

In this section a different behaviour of a single solitary wave solution will be demonstrated. In the previous section the propagation of the solitary wave is shown and we have seen that the solitary wave propagates without changing its shape. However, this situation changes under some minor changes in the parameters of the initial profile. First of all, consider the modified initial profile for the RLW equation is

$$u(x,t) = 20C\,\mathrm{sech}^2\big(k(x-x_0)\big) \tag{3.4}$$

where the definitions of v,k are the same. $20C$ is the amplitude of the wave. Now, for this test case we take the solution domain to be $0 \leq x \leq 250$ for the time period $0 \leq t \leq 80$ with parameter $C = \frac{1}{30}$ which is now smaller than the previous example. This initial profile is a little bit different than the initial condition (3.3). The initial amplitude is less than 1 here but the main difference is that the initial profile is wider than a solitary wave of the specified amplitude.

3.3.2.1 Solution using uniform grid

Figure 3.45, 3.46 and 3.47 show the numerical solutions with 200 grid points and 2000 time steps, 800 grid points and 8000 time steps and 3200 grid points and 32000 time steps respectively. It is a little bit difficult to differentiate the solution behaviour at a later time. So, The solutions at $t = 76$ with 200,800, 3200 grid points are shown in figure 3.48. The difference between the solution with 200 grid points and other solution curve is quite noticeable. The difference between the solution with 800 grid points and 3200 grid points is not that distinguishable but this difference is very important especially the solution difference near $x = 140$.

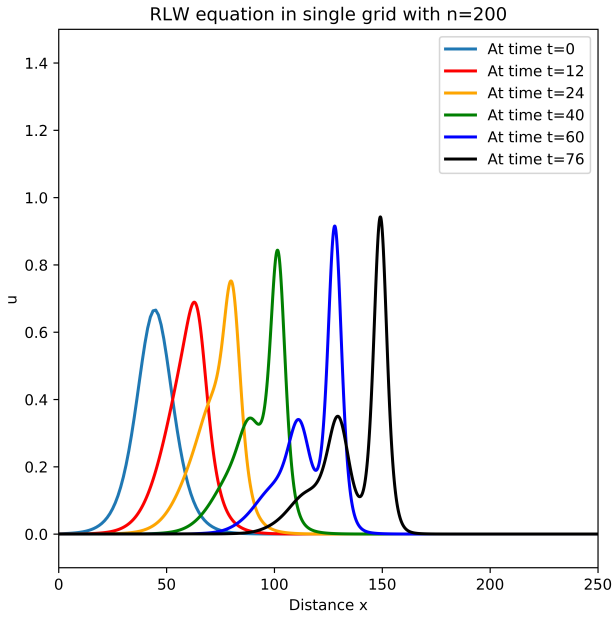


Figure 3.45: RLW equation (wave fissioning problem) solution with 200 grid points

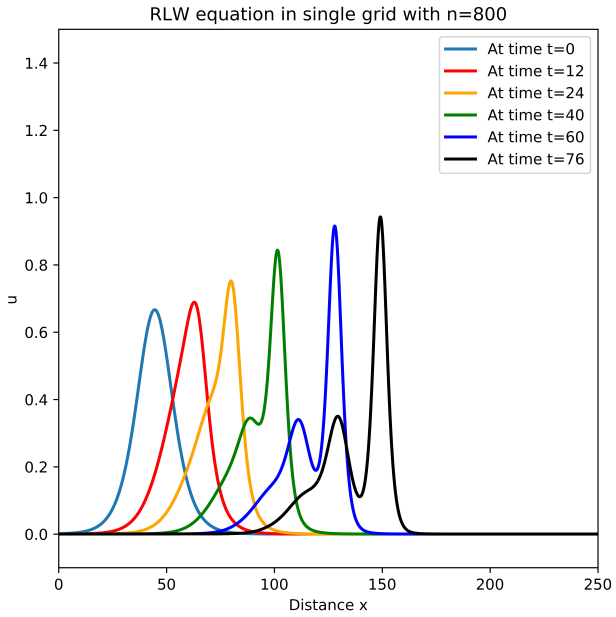


Figure 3.46: RLW equation (wave fissioning problem) solution with 800 grid points

From these figures it can be seen that the wave front steepens at shortly after $t = 24$ another wave starts to emerge. In figure 3.47 at $t = 76$ a second solitary wave is emerging.

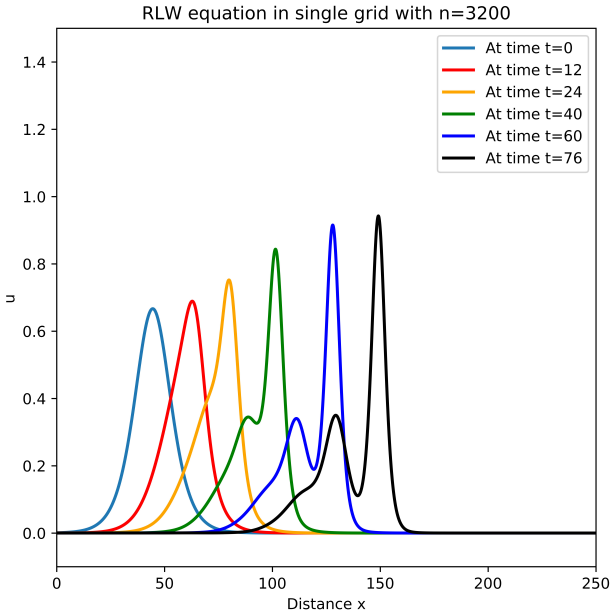


Figure 3.47: RLW equation (wave fissioning problem) solution with 3200 grid points

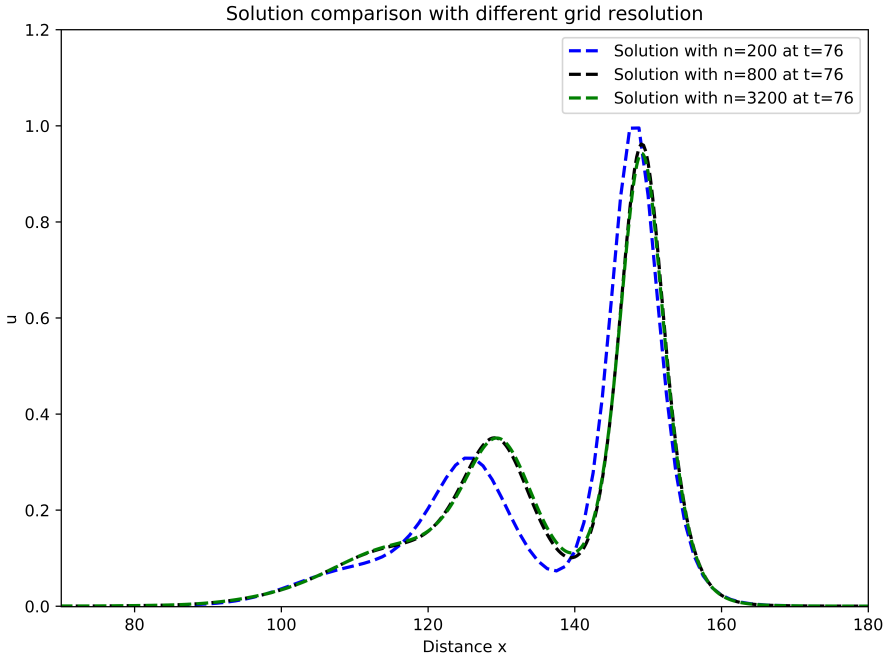


Figure 3.48: RLW equation (wave fissioning problem) solution with different grid points at time $t = 76$

The behaviour is similar in figure 3.46 and 3.45. However, the solution with 3200 grid points with 32000 time steps produces more accurate results due to the numerical scheme which relies on the space and time width. Over time the increment of the magnitude of the amplitude is a little bit high for the low resolution solution. It can also be noted that after a long time the solution requires finer resolution since in the low resolution solution the amplitude becomes too large. The solution gets steep when the gradient is a little bit high and dispersive characteristic responsible for splitting up the solution. Figure 3.49 shows the change of gradient and second derivative at different points in the solution domain.

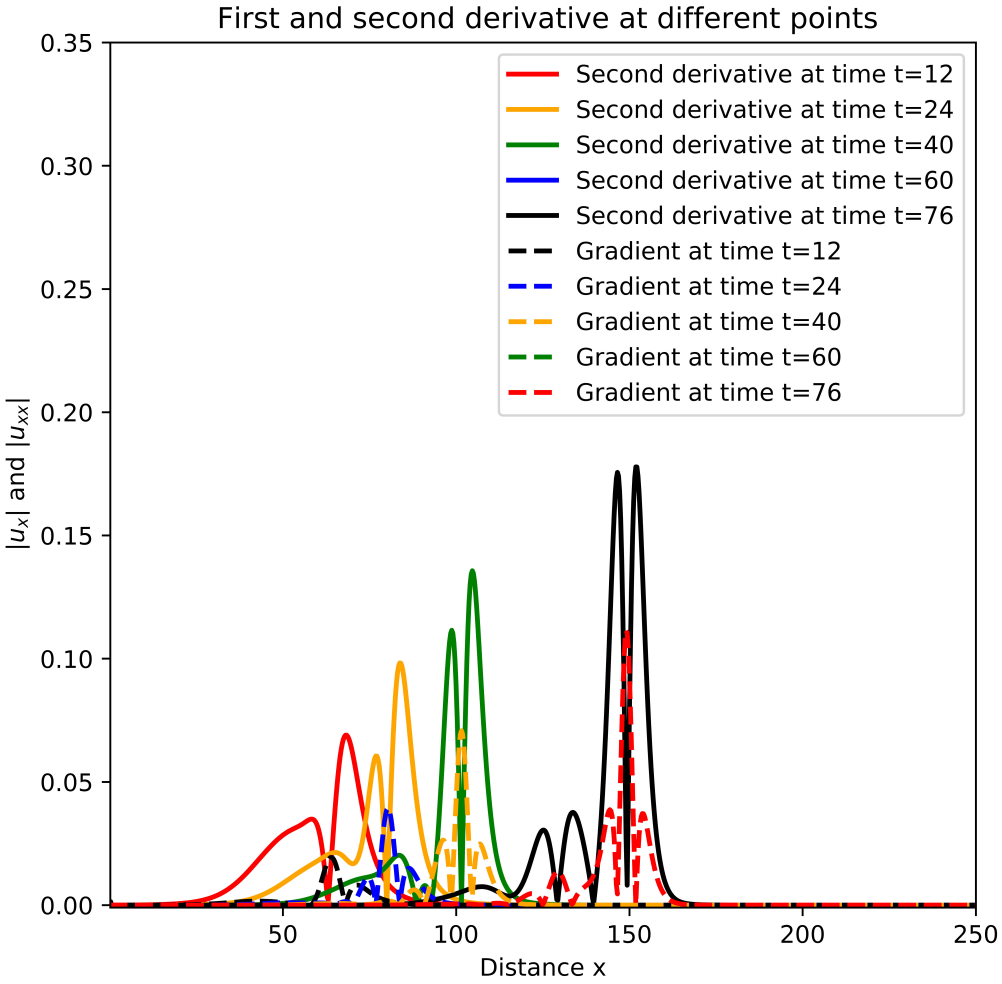


Figure 3.49: Gradient and second derivative at different points (wave fissioning problem)

In figure 3.49 it is evident that both the magnitude of the gradient and second derivative increase with time. However, at the point where the gradient is high, the magnitude of the second derivative is low and vice versa. So, for this problem we will use a combination of the gradient and second derivative for the level 2 refinement criterion. The required time for the different single grid solutions is plotted in figure 3.50. Details of the three different solutions is provided in table 3.16.

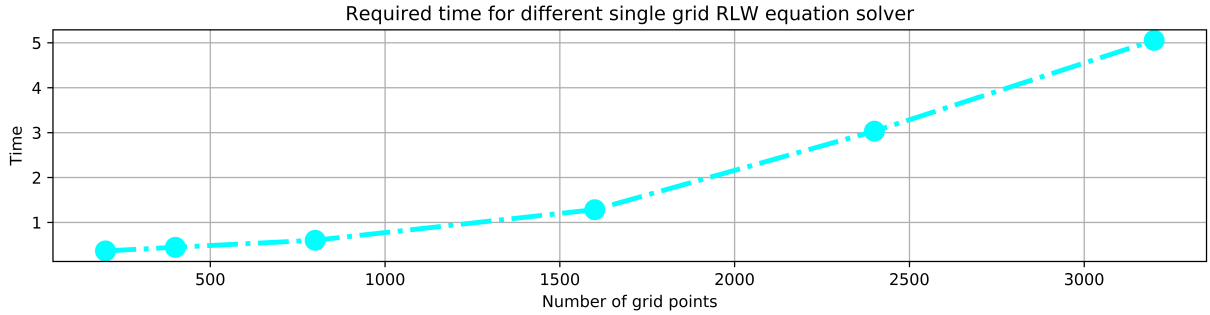


Figure 3.50: Required time for different single grid solution (wave fissioning problem)

Number of grid Points	**Time steps**	Δx	Δt	**Required time(s)**
200	2000	1.25	0.04	0.366
800	8000	0.3125	0.01	1.285
3200	32000	0.078125	0.0025	5.053

Table 3.16: Required computational time for different grid resolution for RLW equation(solitary wave fissioning problem) solution

3.3.2.2 Solution with AMR

For this problem AMR simulation was run with taking 200 grid points as level 0 grid. Small solution cut-off value is set for the level 1 refinement criterion. Level 2 refinement criterion selection is challenging and work on finding the appropriate refinement criterion is underway. However In figure 3.49 we can see the change of gradient and second derivative at different points. We set the combination of gradient, second derivative, and cut-off values as the refinement criterion. If the square root of the sum of the square of these three quantities is greater than or equal 0.1 at any region than the point will be marked for the level 2 refinement. However, this is not the best refinement criterion for this type of problem. This criterion is chosen by the observation of some numerical experiments. The summary of 4 simulations are shown in table 3.17. From this table it can be seen that the level 1 cut-off value increment reduced the required computational time a little. Similarly, this trend is also applicable for level 2 refinement criterion value increment. Figure 3.51 and 3.52 demonstrate the AMR solution with level 1 refinement criterion $|u| \geq .001$ and $|u| \geq 0.01$ respectively. For both the case the level 2 refinement criterion was $\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.05$. In Figure 3.53 and 3.54 are shown the AMR results with level

2 refinement criterion $\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.07$ and $\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.1$ respectively and level 1 refinement criterion is $|u| \geq .01$ for both cases. For the further comparison and discussion We choose $|u| \geq .01$ and $\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.1$ as our level 1 and level 2 refinement criterion respectively. The summary of AMR computation is given in table 3.18.

Level 1 refinement criterion	**Level 2 refinement criterion**	**Required time(s)**
$\|u\| \geq 0.001$	$\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.05$	1.152
$\|u\| \geq 0.01$	$\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.05$	1.095
$\|u\| \geq 0.01$	$\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.07$	0.991
$\|u\| \geq 0.01$	$\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.1$	0.898

Table 3.17: Required time for different refinement criteria (RLW equation wave fissioning problem)

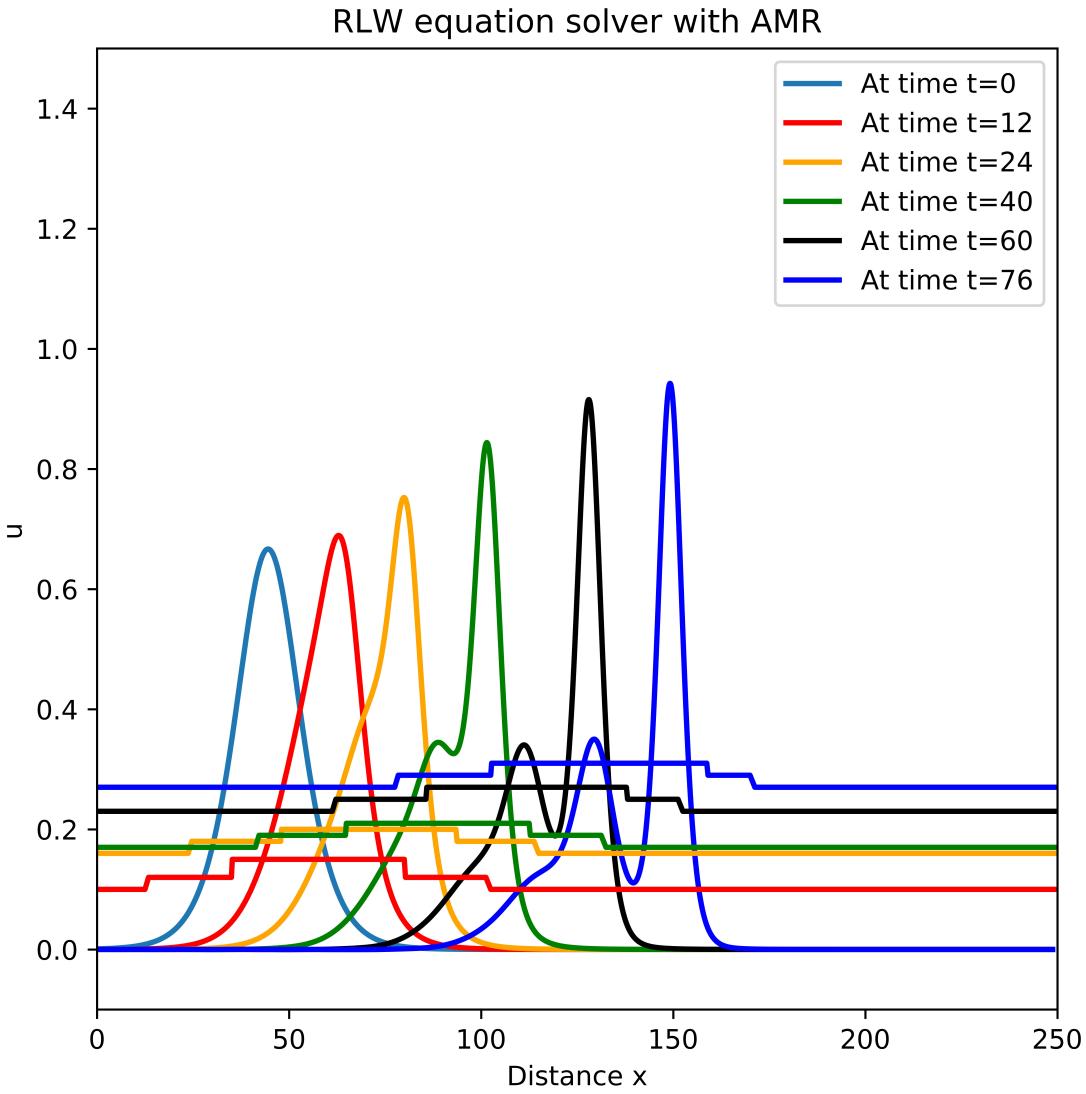


Figure 3.51: AMR solution for RLW equation (solitary wave fissioning problem) with $|u| \geq .001$ as the level 1 refinement criterion and $\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.05$ as the level 2 refinement criterion. The step functions indicate the locations of the different level grids with matching colours

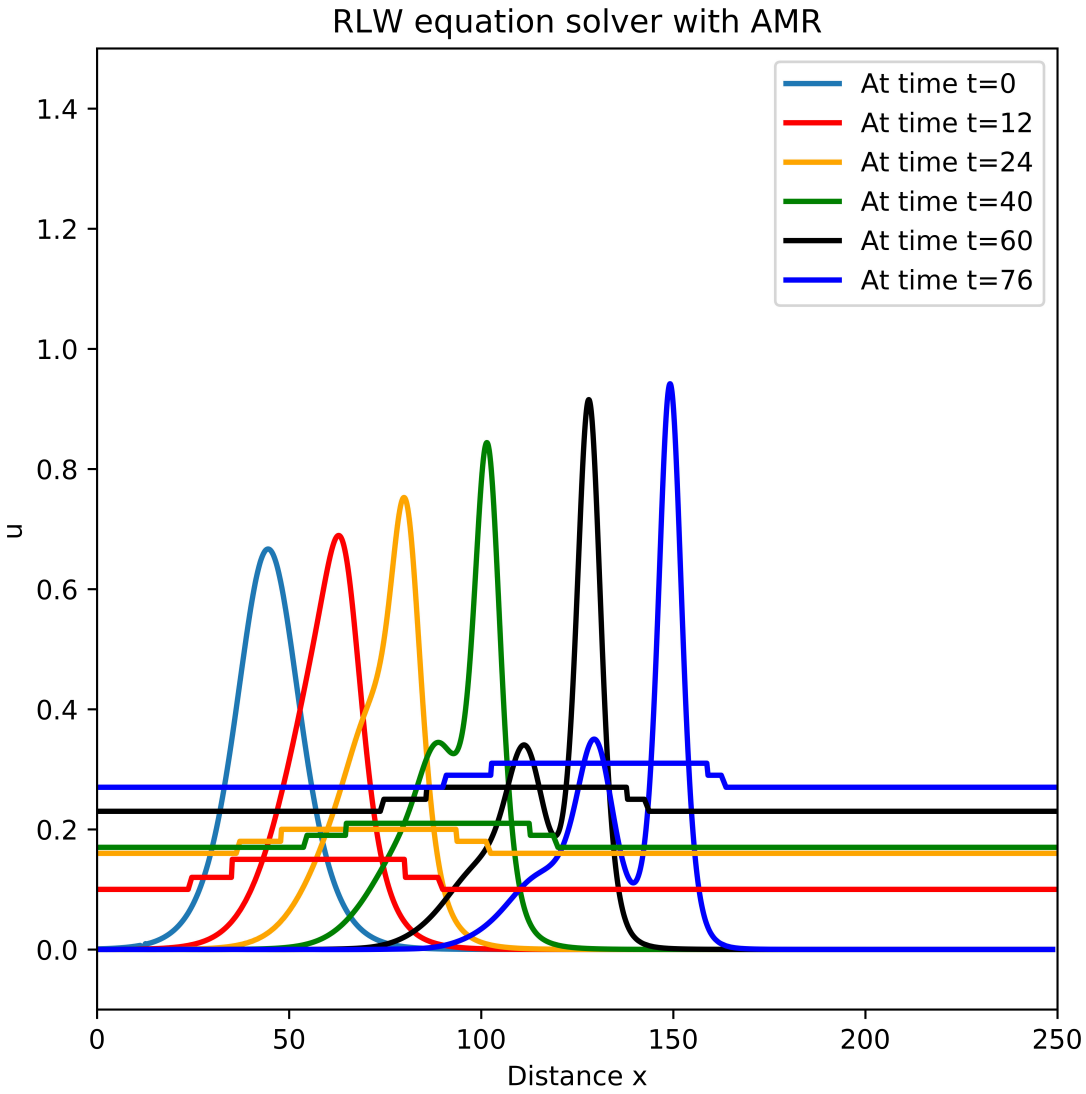


Figure 3.52: AMR solution for RLW equation (solitary wave fissioning problem) with $|u| \geq .01$ as the level 1 refinement criterion and $\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.05$ as the level 2 refinement criterion. The step functions indicate the locations of the different level grids with matching colours

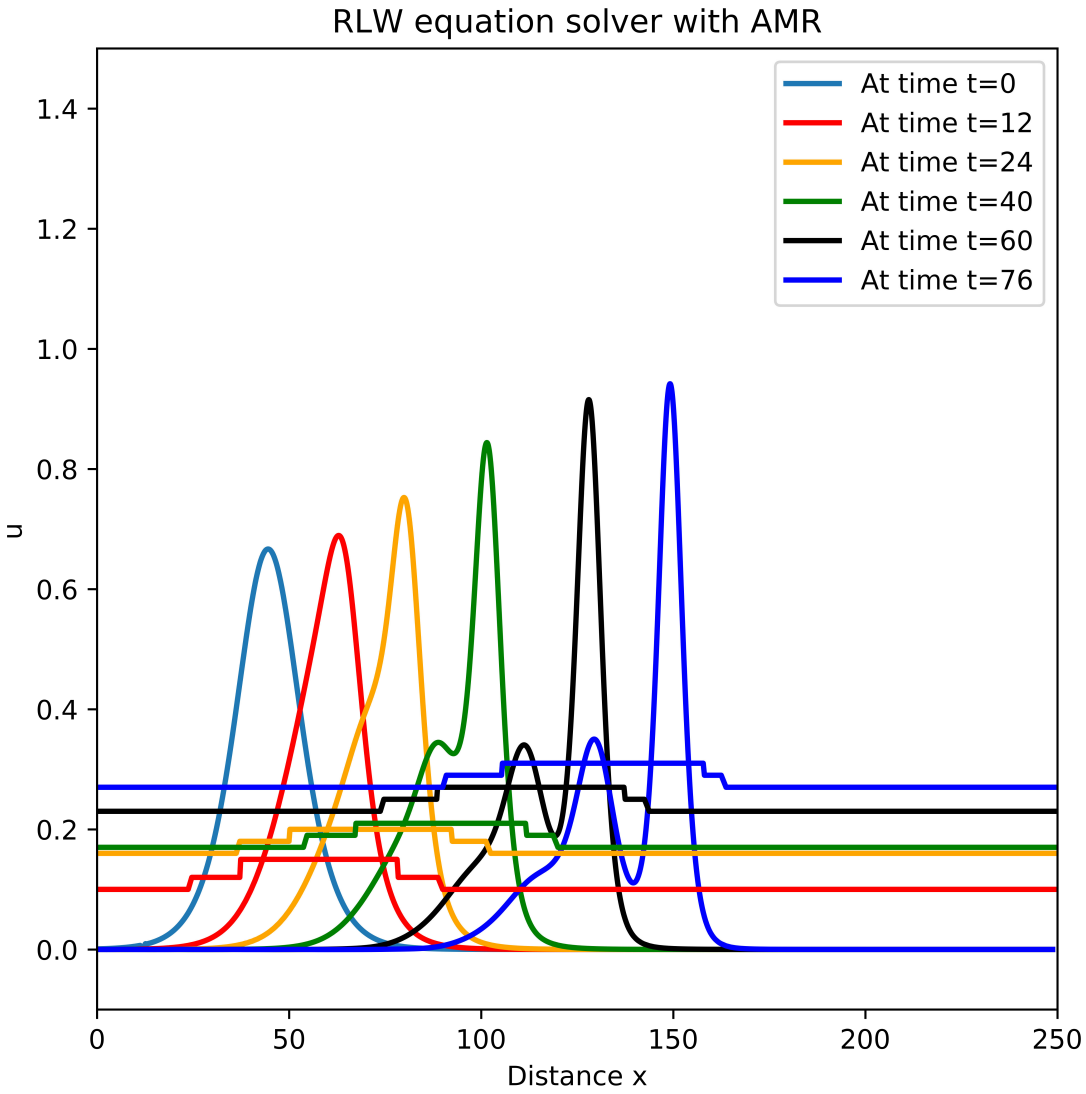


Figure 3.53: AMR solution for RLW equation (solitary wave fissioning problem) with $u \geq .01$ as the level 1 refinement criterion and $\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.07$ as the level 2 refinement criterion. The step functions indicate the locations of the different level grids with matching colours

Number of grid points in Level 0	200
Number of time steps in Level 0	2000
Refinement ratio	4
Level 0 grid space width	0.45
Level 0 grid time interval	0.02
Maximum level of refinement	2
Level 1 refinement	$u \geq 0.01$
Level 2 refinement	$\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.1$
Required computational time	0.898s

Table 3.18: Summary of AMR computation for RLW equation (wave fissioning problem)

Form the AMR solutions the grid hierarchy can be observed. We can found one level 2 cluster is contained by the level 1 grid which is also contained by level 0 grid.

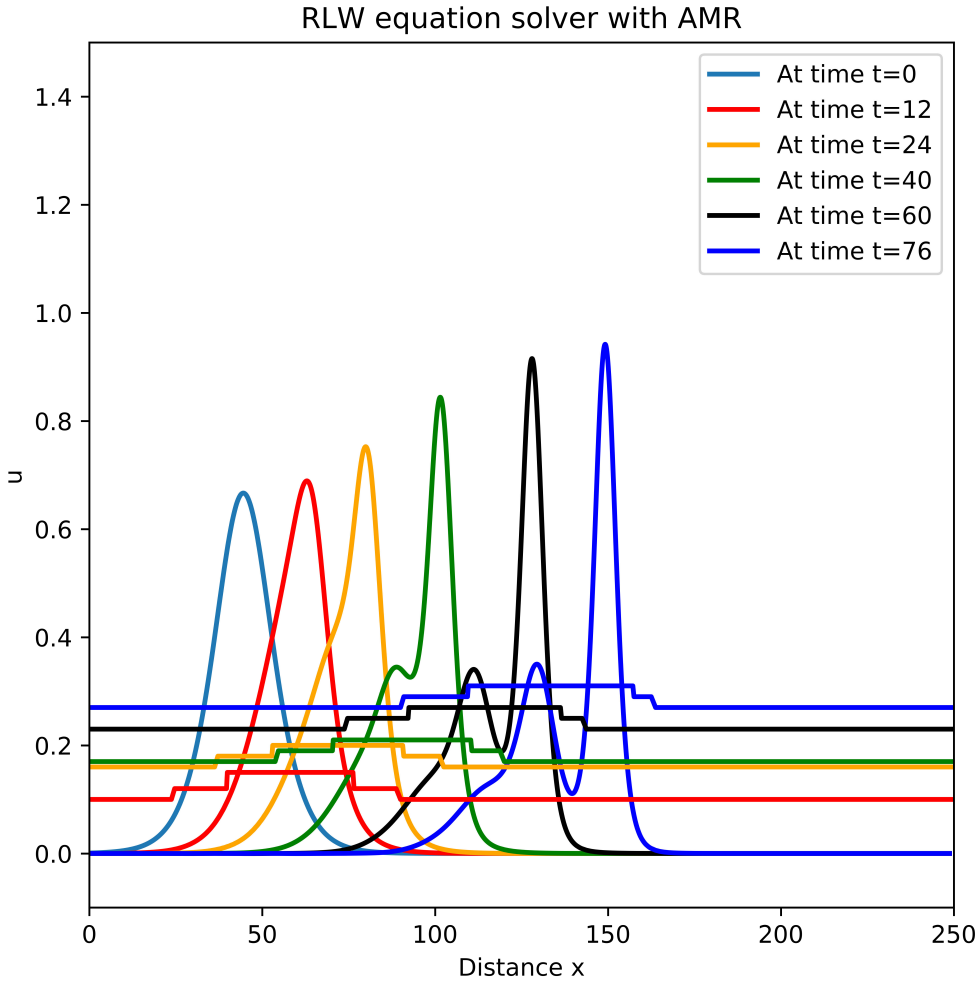


Figure 3.54: AMR solution for RLW equation (solitary wave fissioning problem) with $|u| \geq .01$ as the level 1 refinement criterion and $\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.1$ as the level 2 refinement criterion. The step functions indicate the locations of the different level grids with matching colours

From figure 3.55 the agreement between the AMR and highest resolution uniform grid solution can be seen. So, we achieved the desired results from this.

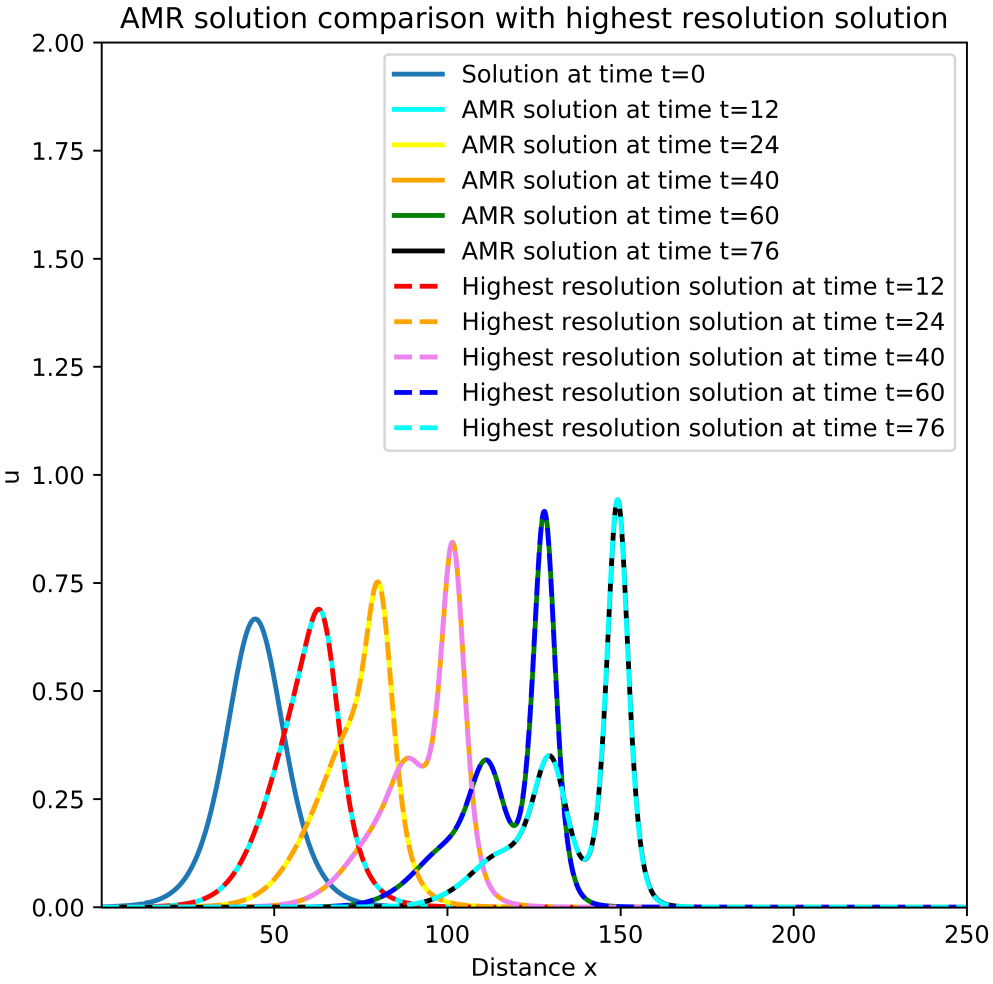


Figure 3.55: AMR solution (with $|u| \geq .01$ as the level 1 refinement criterion and $\sqrt{u^2 + {u_x}^2 + {u_{xx}}^2} \geq 0.1$ as the level 2 refinement criterion) comparison with finest resolution single grid solution

The required computational time for the AMR solution is 0.898 second which is better than the level 1 (800 grid points and 8000 time steps) uniform grid solution. However, the AMR code performs more than 5.5 times faster than the level 2 (3200 grid points and 32000 time steps) uniform grid solution. The objective was to capture the scenario of nonlinearity and dispersion imbalance using multiple refinement levels which is successfully demonstrated. In spite of having some restriction in refinement criteria the performance of the code in this case was satisfactory.

Chapter 4

Conclusions

A finite volume based explicit AMR code has been developed for the numerical computation of the linear advection equation and Burger's equation. Using a similar approach a finite difference based implicit AMR code has also been implemented for the numerical solution of the RLW equation. We demonstrated the ability of our code to numerically solve the wave equations using AMR. The implementation of the data structure is very simple in our code which worked nicely. We have done several numerical experiments to analyze the performance of the code which demonstrated that our AMR code produces results with desired accuracy equivalent to the conventional highest resolution uniform grid solution with a less computational cost.

In the case of linear advection equation or RLW equation we used very simple refinement criteria for the simplification of our implementation. More sophisticated refinement criteria can be used. The performance of AMR code for all the simulations including Burger's equation is satisfactory. A good agreement can be observed between AMR solution and the finest resolution single grid solutions. In addition, a significant time reduction compared to the single grid solutions proves that the code is time efficient. However, the performance will be much more noticeable in the case of longer domain. Simulation of the RLW equation is such a case where the domain was longer than the other problems. The performance was much better than the level 1 uniform grid. Our code successfully implemented the multi level block structured refinement algorithm. We demonstrated that our code can use multiple levels of refinement with any appropriate refinement criteria during the computation. Finally, both explicit and implicit solver has met our expectation.

There are several important extensions of this research that need to be mentioned. We have implemented our code for the one-dimensional case only which can be extended to

two or three dimensions. However, the higher dimensional extension of this code might be a bit challenging. The clustering algorithm might need some modification for the higher dimension. The memory allocation and deallocation should also be taken care of carefully. The refinement criteria we used are very simple and a more complete investigation of possible refinement criteria is needed. Using the error extrapolation from different level grid solutions might be an excellent choice and that work is currently underway. Implementation of higher order Finite Volume Method or Finite Difference Method can possibly be an extension for more accuracy. The interaction of two or more solitary waves is also a great extension to be explored. Finally, this code can be used for some important physical models such as the simulations of undular bores which are quite dispersive in nature.

References

[1] Ann S Almgren, John B Bell, Phillip Colella, Louis H Howell, and Michael L Welcome. A conservative adaptive projection method for the variable density incompressible navier–stokes equations. *Journal of computational Physics*, 142(1):1–46, 1998.

[2] Jhon D Anderson. Jr.(1995). *Computational Fluid Dynamics*, 1995.

[3] David C. Arney and Joseph E. Flaherty. An adaptive mesh-moving and local refinement method for time-dependent partial differential equations. *ACM Trans. Math. Softw.*, 16(1):48–71, March 1990.

[4] David C Arney and Joseph E Flaherty. An adaptive mesh-moving and local refinement method for time-dependent partial differential equations. *ACM Transactions on Mathematical Software (TOMS)*, 16(1):48–71, 1990.

[5] John B Bell, Phillip Colella, and Harland M Glaz. A second-order projection method for the incompressible navier-stokes equations. *Journal of Computational Physics*, 85(2):257–283, 1989.

[6] John B Bell and Daniel L Marcus. A second-order projection method for variable-density flows. *Journal of Computational Physics*, 101(2):334–348, 1992.

[7] Thomas Brooke Benjamin, Jerry Lloyd Bona, and John Joseph Mahony. Model equations for long waves in nonlinear dispersive systems. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 272(1220):47–78, 1972.

[8] Marsha Berger and Isidore Rigoutsos. An algorithm for point clustering and grid generation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1278–1286, 1991.

[9] Marsha J Berger. Data structures for adaptive grid generation. *SIAM Journal on Scientific and Statistical Computing*, 7(3):904–916, 1986.

[10] Marsha J Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of computational Physics*, 82(1):64–84, 1989.

[11] Marsha J Berger and Antony Jameson. Automatic adaptive grid refinement for the euler equations. *AIAA journal*, 23(4):561–568, 1985.

[12] Marsha J Berger and Randall J LeVeque. Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems. *SIAM Journal on Numerical Analysis*, 35(6):2298–2316, 1998.

[13] Marsha J Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of computational Physics*, 53(3):484–512, 1984.

[14] Rupak Biswas, Joseph E Flaherty, and David C Arney. An adaptive mesh-moving and refinement procedure for one-dimensional conservation laws. *Applied numerical mathematics*, 11(4):259–282, 1993.

[15] Alexandre Joel Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of computational physics*, 2(1):12–26, 1967.

[16] HA Dwyer, RJ Kee, and BR Sanders. Adaptive grid method for problems in fluid mechanics and heat transfer. *AIAA Journal*, 18(10):1205–1212, 1980.

[17] JC Eilbeck and GR McGuire. Numerical study of the regularized long-wave equation i: numerical methods. *Journal of Computational Physics*, 19(1):43–57, 1975.

[18] JC Eilbeck and GR McGuire. Numerical study of the regularized long-wave equation. ii: Interaction of solitary waves. *Journal of Computational Physics*, 23(1):63–73, 1977.

[19] Eric S Fraga and John Ll Morris. An adaptive mesh refinement method for nonlinear dispersive wave equations. *Journal of Computational Physics*, 101(1):94–103, 1992.

[20] Ami Harten and James M Hyman. Self adjusting grid methods for one-dimensional hyperbolic conservation laws. *Journal of computational Physics*, 50(2):235–269, 1983.

[21] Charles Hirsch. *Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics*. Elsevier, 2007.

[22] Klaus A Hoffmann and Steve T Chiang. Computational fluid dynamics volume i. *Engineering Education System*, 2000.

[23] Weizhang Huang, Yuhe Ren, and Robert D Russell. Moving mesh methods based on moving mesh partial differential equations. *Journal of Computational Physics*, 113(2):279–290, 1994.

[24] Weizhang Huang and Robert D Russell. A high dimensional moving mesh strategy. *Applied Numerical Mathematics*, 26(1-2):63–76, 1998.

[25] D Irk. Solitary wave solutions for the regularized long-wave equation. *Physics of Wave Phenomena*, 20(3):174–183, 2012.

[26] William L Kath. Making waves: solitons and their optical applications. *Siam News*, 1998.

[27] Diederik Johannes Korteweg and Gustav De Vries. Xli. on the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 39(240):422–443, 1895.

[28] Daniel F Martin, Phillip Colella, and Daniel Graves. A cell-centered adaptive projection method for the incompressible navier–stokes equations in three dimensions. *Journal of Computational Physics*, 227(3):1863–1886, 2008.

[29] Charles R Molenkamp. Accuracy of finite-difference methods applied to the advection equation. *Journal of Applied Meteorology*, 7(2):160–167, 1968.

[30] Peter K Moore and Joseph E Flaherty. A local refinement finite-element method for one-dimensional parabolic systems. *SIAM Journal on Numerical Analysis*, 27(6):1422–1444, 1990.

[31] T Nirmala and K Ganesan. Modified crout's method for an lu decomposition of an interval matrix. In *Journal of Physics: Conference Series*, volume 1000, page 012134. IOP Publishing, 2018.

[32] DH Peregrine. Calculations of the development of an undular bore. *Journal of Fluid Mechanics*, 25(2):321–330, 1966.

[33] J M_ Sanz-Serna and I Christie. A simple adaptive technique for nonlinear wave problems. *Journal of Computational Physics*, 67(2):348–360, 1986.

[34] JM Schofield and PW Hammerton. Numerical and asymptotic solutions of generalised burgers' equation. *Wave Motion*, 51(6):919–934, 2014.

[35] CE Seyler and DL Fenstermacher. A symmetric regularized-long-wave equation. *The Physics of fluids*, 27(1):4–7, 1984.

[36] HK Versteeg and W Malalasekera. Computational fluid dynamics. *The finite volume method*, 1995.

[37] Andrew B White, Jr. On selection of equidistributing meshes for two-point boundary-value problems. *SIAM Journal on Numerical Analysis*, 16(3):472–502, 1979.

[38] Gerald Beresford Whitham. *Linear and nonlinear waves*, volume 42. John Wiley & Sons, 2011.

[39] A Vande Wouwer, P Saucez, WE Schiesser, and S Thompson. A matlab implementation of upwind finite differences and adaptive grids in the method of lines. *Journal of computational and applied mathematics*, 183(2):245–258, 2005.

[40] Qinghai Zhang, Hans Johansen, and Phillip Colella. A fourth-order accurate finite-volume method with structured adaptive mesh refinement for solving the advection-diffusion equation. *SIAM Journal on Scientific Computing*, 34(2):B179–B201, 2012.

[41] T Zhanlav, Ochbadrakh Chuluunbaatar, and V Ulziibayar. Higher-order accurate numerical solution of unsteady burgers' equation. *Applied Mathematics and Computation*, 250:701–707, 2015.

[42] OC Zienkiewicz and JZ Zhu. Adaptivity and mesh generation. *International Journal for Numerical Methods in Engineering*, 32(4):783–810, 1991.