A NEW H-R HYBRID MOVING MESH - LEVEL SET METHOD

by

Benjamin Ong

B.Sc., Simon Fraser University, 2000

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
Doctor of Philosophy
in the Department
of
Mathematics

© Benjamin Ong 2007
SIMON FRASER UNIVERSITY
Summer 2007

All rights reserved. This work may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.
APPROVAL

Name: Benjamin Ong
Degree: Doctor of Philosophy
Title of thesis: A New $h$-$r$ Hybrid Moving Mesh - Level Set Method

Examinining Committee: Dr. Veselin Jungic
Chair

_______________________________
Dr. Steven Ruuth
Senior Supervisor

_______________________________
Dr. Robert Russell
Supervisor

_______________________________
Dr. James Verner
Committee Member

_______________________________
Dr. Paul Muir
External Examiner

_______________________________
Dr. J.F. Williams
Internal Examiner

Date Approved: August 1, 2007
To my Lord and Savior, Jesus Christ, who faithfully guides and watches over me. To Him be all praise, honor and glory.
“... live a life worthy of the calling you have received. ”

— Ephesians 4:1
Acknowledgments

I thank my supervisors, Dr. S. Ruuth and Dr. R. Russell for their constant encouragement, insight and generosity. Special thanks also to Dr. J. Verner, Dr. J.F. Williams and C.B. Macdonald for useful discussions on several problems.

I owe special thanks to my family for their love and support during my entire time at university. I also thank all my friends and colleagues who have made my time at SFU meaningful and rewarding.

The research behind this thesis was supported by NSERC and the C.D. Nelson Memorial Scholarship.
Contents

Approval  ii
Abstract    iii
Dedication  iv
Quotation   v
Acknowledgments  vi
Contents     vii
List of Figures xi

1 Introduction  1

2 Moving Mesh  4
  2.1 Adaptive Mesh Generation  4
     2.1.1 Monitor Functions  5
     2.1.2 Equi-distribution  6
     2.1.3 De Boor’s Algorithm  7
     2.1.4 Simplistic Moving Mesh Method  9
  2.2 Moving Mesh PDE (MMPDE)  10
     2.2.1 Derivation  10
     2.2.2 Boundary Conditions  14
     2.2.3 Discussion  15
  2.3 Geometric Conservation Law (GCL)  15
     2.3.1 Derivation  15


<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.2</td>
<td>Boundary Conditions</td>
<td>19</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Discussion</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>Other Moving Mesh Methods</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>Coupling with Physical Equations</td>
<td>20</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Inter-weaving Approach</td>
<td>20</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Lagrangian Correction</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Level Set Framework</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>Level Set Representation of a Static Equi-distributed 1D Mesh</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>Level Set Representation of a Static Higher Dimensional Mesh</td>
<td>28</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Level Contours in Higher Dimensions</td>
<td>28</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Intersection of Level Contours in $\mathbb{R}^2$</td>
<td>30</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Intersection of Level Contours in $\mathbb{R}^3$</td>
<td>34</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Level Set Function Initialization for $\mathbb{R}^2$</td>
<td>34</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Level Set Function Initialization for $\mathbb{R}^3$</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>Level Set Evolution Equation</td>
<td>40</td>
</tr>
<tr>
<td>3.4</td>
<td>Further Discussion</td>
<td>42</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Reinitialization</td>
<td>42</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Representing Arbitrary Structured Meshes</td>
<td>42</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Connectivity</td>
<td>43</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Implicit versus Explicit Representation</td>
<td>44</td>
</tr>
<tr>
<td>3.4.5</td>
<td>Adaptive Contour Levels</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>New $h-r$ Hybrid Method</td>
<td>46</td>
</tr>
<tr>
<td>4.1</td>
<td>Hybrid Moving Mesh - Level Set Method</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Initialization of Level Set Functions in $\mathbb{R}^2$</td>
<td>48</td>
</tr>
<tr>
<td>4.3</td>
<td>Coupling with a Physical Equation in $\mathbb{R}^1$</td>
<td>49</td>
</tr>
<tr>
<td>4.4</td>
<td>New Boundary Conditions in $\mathbb{R}^1$</td>
<td>50</td>
</tr>
<tr>
<td>4.5</td>
<td>Interpretation of New Boundary Conditions</td>
<td>52</td>
</tr>
<tr>
<td>4.6</td>
<td>Further Discussion</td>
<td>52</td>
</tr>
<tr>
<td>4.6.1</td>
<td>New Boundary Conditions for Moving Domains</td>
<td>52</td>
</tr>
<tr>
<td>4.6.2</td>
<td>New Boundary Conditions in $\mathbb{R}^2$</td>
<td>54</td>
</tr>
<tr>
<td>4.6.3</td>
<td>New Boundary Conditions without the Level Set Framework</td>
<td>56</td>
</tr>
</tbody>
</table>
### 5 Numerical Discretization

5.1 Moving Mesh Equations ................................. 58
  5.1.1 MMPDE4 ........................................... 58
  5.1.2 MMPDE6 ........................................... 59
  5.1.3 GCL ............................................... 60
5.2 Level Set Equation ........................................ 61
5.3 Physical PDEs ............................................ 62
  5.3.1 Advection-Diffusion Equation ....................... 62
  5.3.2 Burgers’ Equation .................................. 63
5.4 Temporal Discretization ................................... 64
  5.4.1 Forward Euler ...................................... 64
  5.4.2 Multi-Step and Multi-Stage Schemes ................. 64
5.5 Coupled Hybrid Algorithm in $\mathbb{R}^1$ .................. 64
5.6 Software Development .................................... 68

### 6 Numerical Examples in $\mathbb{R}^1$

6.1 Example 1: an Equi-spaced Mesh where $L(t)$ is Increasing ........ 69
6.2 Example 2: a Non-uniform Mesh where $L(t)$ is Increasing .......... 74
6.3 Example 3: a Non-uniform Mesh where $L(t)$ is Decreasing .......... 77
  6.3.1 $\psi(\eta)$ Monotonically Increasing in $\eta$ .................... 77
  6.3.2 $\psi(\eta)$ Monotonically Decreasing in $\eta$ .................... 80
  6.3.3 $\psi(\eta)$ Non Monotonic in $\eta$ ............................. 83
6.4 Example 4: A Solution with Finite Time Blow-up .................. 86
  6.4.1 Adding Nodes at the Domain Boundary ....................... 86
  6.4.2 Adding Nodes in the Interior of the Domain ................. 89
6.5 Example 5: Viscous Burgers’ Equation .......................... 94
6.6 Example 6: Advection-Diffusion Equation .......................... 103

### 7 Numerical Examples in $\mathbb{R}^2$

7.1 Example 8: Resolving a Circle .................................. 112
7.2 Example 9: Travelling Circle .................................. 119
7.3 Example 10: Oscillating Membrane .............................. 120
7.4 Example 11: A Rotating Circle .................................. 120
7.5 Discussion .................................................. 125
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 1.1</td>
<td>Mesh Crossing Example</td>
<td>3</td>
</tr>
<tr>
<td>Fig. 2.1</td>
<td>Coordinate Mappings</td>
<td>7</td>
</tr>
<tr>
<td>Fig. 2.2</td>
<td>GCL: Transformation Mapping</td>
<td>16</td>
</tr>
<tr>
<td>Fig. 2.3</td>
<td>Lagrangian Correction as a Linear Interpolation</td>
<td>23</td>
</tr>
<tr>
<td>Fig. 3.1</td>
<td>Level Set Function Initialized as the Integral of a Monitor Function</td>
<td>27</td>
</tr>
<tr>
<td>Fig. 3.2</td>
<td>Non-monotonic Level Set Function</td>
<td>28</td>
</tr>
<tr>
<td>Fig. 3.3</td>
<td>Multiple Level Set Representations of a Mesh</td>
<td>29</td>
</tr>
<tr>
<td>Fig. 3.4</td>
<td>Level Contours of a 2D Function</td>
<td>30</td>
</tr>
<tr>
<td>Fig. 3.5</td>
<td>Level Set Representation of a 2D Uniform Mesh</td>
<td>32</td>
</tr>
<tr>
<td>Fig. 3.6</td>
<td>Level Set Representation of a 2D Adaptive Mesh</td>
<td>33</td>
</tr>
<tr>
<td>Fig. 3.7</td>
<td>Incorrect Level Set Initialization of a 2D Mesh</td>
<td>36</td>
</tr>
<tr>
<td>Fig. 3.8</td>
<td>Normalized Level Set Initialization of a 2D Mesh</td>
<td>38</td>
</tr>
<tr>
<td>Fig. 3.9</td>
<td>Problems with Level Set Initialization Along Lines in 2D</td>
<td>39</td>
</tr>
<tr>
<td>Fig. 4.1</td>
<td>Boundary Segments for $\psi_1(x,y)$</td>
<td>55</td>
</tr>
<tr>
<td>Fig. 4.2</td>
<td>Boundary Segments for $\psi_2(x,y)$</td>
<td>56</td>
</tr>
<tr>
<td>Fig. 6.1</td>
<td>Example 1: Natural Creation of Grid Points</td>
<td>72</td>
</tr>
<tr>
<td>Fig. 6.2</td>
<td>Example 1: Mesh Trajectories</td>
<td>73</td>
</tr>
<tr>
<td>Fig. 6.3</td>
<td>Example 2: Traditional Boundary Condition Fixes Number of Nodes</td>
<td>75</td>
</tr>
<tr>
<td>Fig. 6.4</td>
<td>Example 2: Mesh Trajectories Using Traditional Boundary Conditions</td>
<td>76</td>
</tr>
<tr>
<td>Fig. 6.5</td>
<td>Example 2: New Boundary Conditions Allows for Creation of Nodes</td>
<td>78</td>
</tr>
<tr>
<td>Fig. 6.6</td>
<td>Example 2: Mesh Trajectories Using New Boundary Conditions</td>
<td>79</td>
</tr>
<tr>
<td>Fig. 6.7</td>
<td>Example 3: Monotonically Increasing $\psi$</td>
<td>81</td>
</tr>
<tr>
<td>Fig. 6.8</td>
<td>Example 3: Mesh Trajectory with a Monotonically Increasing $\psi$</td>
<td>82</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

To solve time-dependent partial differential equations (PDEs) numerically, a method of lines approach is typically implemented; the PDE is discretized in space with an appropriate spatial mesh, and the resulting system of differential algebraic equations (DAEs) are integrated in time using a suitable DAE time-integrator.

It is often advantageous to employ an adaptive spatial mesh if fine-scale structures develop, propagate or disappear as the solution evolves. Examples of such behavior can be seen in combustion simulations [Cao et al., 1999a, Yuan & Tang, 2007], shock formation and propagation in compressible fluid flow and shear flows [Thompson et. al., 1999]. Ideally, the mesh adaptation scheme clusters a higher density of grid points in a local, possibly time-dependent, region of interest.

There are three main types of methods to perform grid, or mesh, adaptation: (i) $h$-refinement methods, which coarsen or refine a mesh locally, (ii) $p$-refinement methods, which take higher or lower order polynomial approximations locally, and (iii) $r$-refinement methods, or moving mesh methods, which redistribute a fixed number of grid nodes, concentrating a higher density of grid nodes in desired regions. Much work has been done on $h$-refinement methods, $p$-refinement methods, as well as $h$-$p$ hybrid methods. We refer the interested reader to [Arney & Flaherty, 1990, Berger & Oliger, 1984]. In the literature, $h$-refinement is synonymous with Adaptive Mesh Refinement (AMR). In this thesis, we focus our discussion on $r$-refinement methods (referred to interchangeably as moving mesh methods), and introduce our new $h$-$r$ hybrid method. Previous work on $h$-$r$ hybrid methods include [Capon & Jimack, 1996], which utilizes $r$-adaptivity to move a finite element mesh, refining or coarsening the mesh using $h$-adaptivity as necessary and [Wang & Tang, 2007], which
utilizes $r$-adaptivity to move a coarse mesh, and refines the coarse mesh using $h$-adaptivity.

The development of our new $h$-$r$ hybrid method is motivated in two ways. The first goal is to develop a method that contains desired qualities of both $h$- and $r$-refinement methods. Specifically, $h$-refinement methods coarsen or refine a mesh naturally, while $r$-refinement methods give a "smooth" evolution of the mesh. We wish to develop a new $h$-$r$ hybrid method that gives a smooth evolution of the mesh, even with the addition or removal of mesh nodes. Mathematically, we seek a moving mesh, i.e., a set of mesh trajectories $x_k(t), k = 0, \ldots, N(t)$, where $N(t)$ denotes the number of mesh nodes, and $t$ is the independent time variable. The second motivation addresses a potential hurdle for using moving mesh methods, commonly known as mesh crossing. In one spatial dimension, mesh crossing occurs when two neighboring nodes, $x_k(t) < x_{k+1}(t)$, switch orderings at a later time, $t + \Delta t$, i.e. $x_k(t + \Delta t) > x_{k+1}(t + \Delta t)$. In Figure 1.1, the two nodes $x_{10}$ and $x_{11}$ have switched orderings.

Our new hybrid moving mesh method employs level set technology to represent and move the grid nodes. Specifically, we embed the location of the mesh nodes in an implicit function representation, whose "level sets" or "level contours" give the location of the mesh. This is a novel approach on two levels: (i) it employs ideas from both moving mesh methods and level set methods, and (ii) the number of grid points is allowed to vary according to the total integral of a specified monitor function, but the mesh is moved smoothly in a fashion similar to adaptive $r$-refinement. Through the formulation, this hybrid method is guaranteed to circumvent mesh crossing events.

The thesis begins with two introductory chapters. Chapter 2 introduces basic principles about moving mesh methods, and compares the so-called MMPDE and GCL methods, while Chapter 3 addresses how we use ideas from the level set framework to represent and evolve the mesh. In Chapter 4, we derive and describe our new $h$-$r$ hybrid method. Chapter 5 gives the numerical discretizations and practical implementation details, while Chapter 6 and Chapter 7 present numerical examples in one and two dimensions to validate our new hybrid method.
Figure 1.1: The above figure is an illustration of mesh crossing. Notice that at this snapshot of time, the mesh nodes, $x_{10}$ and $x_{11}$ have switched orientations, $x_{10} > x_{11}$. 
Chapter 2

Moving Mesh

When solving time-dependent PDES using the method of lines, it may be advantageous to redistribute spatial mesh nodes, concentrating a higher density of nodes in desired regions. These nodes are moved into desired regions by solving suitable equations for mesh velocities. A classic example of moving mesh methods are the moving mesh partial differential equations (MMPDEs) derived in [Huang et al., 1994b]. MMPDEs in one spatial dimension are derived by differentiating perturbations of the so-called equi-distribution relation with respect to time, resulting in equations for mesh velocities. These mesh velocities preserve the equi-distribution principle as the solution and grid evolve.

In Section 2.1, we discuss the basic principles of equi-distribution, static grid generation and related terminology. We then derive the MMPDEs in one spatial dimension, generalizing earlier work by Huang, Ren and Russell to allow for a changing number of grid points. We also present a derivation of the Geometric Conservation Law (GCL), a newer moving mesh scheme.

2.1 Adaptive Mesh Generation

The problem is simple: given a function $\rho(x) > 0$, how do we use a finite number of grid points, $\{x_j\}$ to best approximate $\rho(x)$, i.e., how can $\{x_j\}$ be chosen adaptively? This section begins to answer this question by providing some basic principles of adaptive mesh generation, beginning with a discussion on monitor functions and principles of equi-distribution.
CHAPTER 2. MOVING MESH

2.1.1 Monitor Functions

Adaptive meshes are particularly useful for resolving problems that have boundary layers, shocks or blow-up regions. In order to quantify how the solution \( u(x, t) \) behaves spatially, one prescribes a monitor function, \( \rho(u(x, t)) \). Learning the art of selecting a good monitor function takes practice, and often an understanding of the underlying physics and mathematics. One might be interested in the curvature of the solution (e.g., resolving a corner type solution), the gradient of the solution (e.g., some measure of the arc-length), some error estimate, or the solution itself.

In this section, we present commonly used monitor functions employed in this thesis, as well as other monitor functions of interest, not considered in this thesis. In Section 2.1.2, we will discuss how the notion of equi-distributing a monitor function can be used for grid generation. The first monitor function of interest is

\[ \rho(u(x, t)) = 1. \]

Observe that \( \rho(u(x, t)) = 1 \) is independent of \( x \) and \( t \). One expects (as we will show in the next section) that equi-distributing this monitor function produces a static, uniformly spaced mesh.

We also employ a modified arc-length monitor function,

\[ \rho(u(x, t)) = \sqrt{1 + \alpha u_x^2(x, t)}, \]

which utilizes the first derivative of \( u \) to describe how quickly the solution \( u(x, t) \) is increasing or decreasing spatially and temporally. Here, \( \alpha > 0 \) is a tuning parameter to scale the derivative term in the arc-length monitor function. Note that with \( \alpha = 1 \), the integral of \( \rho \) over the domain gives the arc-length of \( u(x, t) \).

Lastly, the monitor function

\[ \rho(u(x, t)) = (u(x, t))^p, \]

where \( p > 1 \) is a problem specific parameter, is used to quantify the behavior of \( u(x, t) \) in a problem containing a blow-up feature. [Budd et al., 1996a] show that scaling invariance is obtained when this monitor function is used in certain problems.

There are other monitor functions of interest, not considered in this thesis. Error based monitor functions, such as the norm of the squared relative error or some interpolation error,
provide a means of controlling spatial error (see [Huang & Sun, 2003]). For example, the
monitor function
\[
\rho(\tilde{f}(x)) = \sqrt{|\tilde{f}(x) - f(x)|^2},
\]
where \(\tilde{f}(x)\) is some interpolated approximation to \(f(x)\), gives a quantitative measure of,
and means of controlling, interpolation error. One can also use the monitor function
\[
\rho(u(x,t)) = \sqrt{|u_{h}(x,t) - u_{h/2}(x,t)|^2},
\]
to perform spatial error control. Here, \(u_{h}(x,t)\) is a discrete approximation to \(u(x,t)\) on a
coarse mesh and \(u_{h/2}(x,t)\) is a discrete approximation to \(u(x,t)\) on a finer mesh.

In this thesis, we assume that either (i) a monitor function is given explicitly as a function
of \(x\) and/or \(t\), i.e., \(\rho(x,t)\) or \(\rho(x)\), or (ii) the monitor function can be computed to quantify
features of the solution, \(u(x)\) or \(u(x,t)\), i.e., \(\rho(u(x))\) or \(\rho(u(x,t))\). In the remainder of this
chapter and in parts of this thesis, we adopt the notation \(\rho(u(x)) = \rho(x)\) or \(\rho(u(x,t)) = \rho(x,t)\).

### 2.1.2 Equi-distribution

The idea behind equi-distributing a prescribed monitor function, \(\rho(x)\), is to find a set of
grid points \(\{x_j\}, j = 0,\ldots,N\), such that
\[
\int_{x_{j-1}}^{x_j} \rho(x) \, dx = \frac{\int_{x_0}^{x_N} \rho(x) \, dx}{N}, \quad \forall j = 1,\ldots,N, \quad a \leq x \leq b. \tag{2.1}
\]
In other words, we find the set of grid points \(\{x_j\}\) such that the integral of the monitor
function in each subinterval \([x_{j-1}, x_j]\) is the same value. We will refer to an equi-distributed mesh
as a mesh which equi-distributes a monitor function \(\rho(u(x))\). In Figure 2.1, we show
an example of a mesh which equi-distributes \(\rho(u(x))\) with respect to arc-length. In the left
plot of Figure 2.1, we show the function \(u(x)\) sampled at the equi-distributed mesh points
\(\{x_j\}\), i.e., we plot \((x_j, u(x_j))\). In the right plot, we let \(\xi_j = j/N\) and plot \((\xi_j, u(x_j))\), where
\(\xi_j\) is a scaled version of the enumeration \(j = 0,\ldots,N\), and is commonly referred to as the
computational variable or logical variable. In fact, the left plot of \((x_j, u(x_j))\) is referred to
as a plot of \(u(x)\) in the physical domain, whereas the right plot of \((\xi_j, u(x_j))\) is referred to
as a plot of \(u(\xi)\) in the computational, or logical domain. In the computational domain, the
plot of \(u(\xi)\) is much smoother than the plot of \(u(x)\) in the physical domain. This is not a
coincidence, since an equi-distributed mesh clusters more mesh nodes around areas of large variation of \( u(x, t) \) in the physical domain. Since for each \( j \) there is a unique mesh node, \( x_j = x(j) \) can be interpreted as a function of the enumeration variable. Similarly, we can interpret \( x(\xi_j) = x_{\xi_jN} \) as a function, or a mapping of the normalized enumeration variable, \( \xi_j = j/N \).

![Equi-Distributed Mesh, physical domain](image1)

![Equi-Distributed Mesh, computational domain](image2)

Figure 2.1: The left plot shows the graph of a function \( u(x) \) in the physical domain, while the right plot shows the graph of the function \( u(\xi) \) in the computational domain. Notice that the plot of the function \( u \) is much smoother in the computational domain (i.e., the slope is less steep).

We return to the previous discussion in Section 2.1.1 and illustrate how equi-distributing \( \rho(u(x, t)) = 1 \) can be used for grid generation. The set of mesh nodes \( \{x_j\} \) which satisfies

\[
\int_{x_{j-1}}^{x_j} dx = \frac{1}{N} \int_a^b dx, \quad j = 1, \ldots, N, \quad x \in [a, b],
\]

is the uniformly spaced mesh, \( x_j = a + j\Delta x \), where

\[
\Delta x = \frac{b - a}{N}.
\]

### 2.1.3 De Boor's Algorithm

In one dimension, there are several robust algorithms to generate static meshes which equi-distribute a given monitor function. We give a brief description of de Boors algorithm, one such simple, yet reliable algorithm.
CHAPTER 2. MOVING MESH

Suppose that the monitor function \( p \) is given on some prescribed, non-optimal mesh \( \{x_k\}, k = 0, \ldots, M \). Here, we mean non-optimal in the sense that the monitor function \( p \) is not equi-distributed within each subinterval. The goal is to find an optimal mesh, \( \{x_j\}, j = 0, \ldots, N \), i.e., one which equi-distributes \( p \). Here, \( M \) is the number of mesh nodes initially prescribed, and \( N \) is the number of mesh nodes desired. Denote \( p_k = p(x_k) \). The idea behind de Boor’s algorithm is to approximate \( p(x) \) by a piecewise constant polynomial, \( \tilde{p}(x) \), (i.e., step functions) on the prescribed mesh,

\[
\tilde{p}(x) = \frac{1}{2}(\rho_{k-1} + \rho_k), \quad \text{for } x \in (x_{k-1}, x_k), \quad k = 1, \ldots, M, \quad (2.2)
\]

and then find the equi-distributing mesh for this piecewise-constant polynomial, \( \tilde{p}(x) \). Denoting

\[
I(\xi) = \int_a^\xi \tilde{p}(x) \, dx,
\]

and observing that \( I(x) \) is a piecewise-linear polynomial with \( I(x_0) = a = 0 \), and

\[
I(\xi_k) = \sum_{r=1}^{k} (\xi_r - \xi_{r-1}) \left( \frac{\rho_{r-1} + \rho_r}{2} \right), \quad k = 1, \ldots, M,
\]

the goal now is finding \( \{x_j\}, j = 0, \ldots, N \), such that

\[
I(x_j) = \frac{j}{N} I(b).
\]

Setting \( x_0 = a \) and \( x_N = b \), we find the remaining mesh nodes, \( \{x_j\}, j = 1, \ldots, N - 1 \) by letting \( s \) be an integer such that

\[
I(\xi_{s-1}) < \frac{j}{N} I(b) < I(\xi_s).
\]

Since \( I(x) \) is piecewise linear, \( \{x_j\} \) can be calculated using

\[
(x_j - \xi_{s-1}) \left( \frac{\rho_{s-1} + \rho_s}{2} \right) = \frac{j}{N} I(b) - I(\xi_{s-1}). \quad (2.3)
\]

Observe that this algorithm only generates an equi-distributed mesh to the piecewise constant polynomial defined in equation (2.2).

To obtain a good approximation to the equi-distributing mesh for a more general monitor function \( \rho(x) \), an iterative scheme needs to be implemented. One such iterative scheme given by [Huang & Russell, 2003] uses the following iterative algorithm. (N.B., we adopt a
MATLAB style notation to describe vectors. For example, \([0 : 0.25 : 1]\) represents the vector \([0.0, 0.25, 0.5, 0.75, 1]\). The problem is as follows: Given \(\rho(x)\) on the domain \(a \leq x \leq b\), find the equi-distributed mesh, \(\{x_j\}, j = 0, \ldots, N\).

1. Set \(\{y_j\}\) to be a uniform mesh, 
\[ \bar{y} = \left[ a : \frac{b - a}{N} : b \right]. \]
2. Set \(\rho_j = \rho(y_j)\).
3. Define \(\bar{\rho}\) to be a piecewise constant polynomial, given by
\[ \bar{\rho}(x) = \frac{1}{2} (\rho_{j-1} + \rho_j) \quad \text{for} \ x \in (y_{j-1}, y_j), \ j = 1, \ldots, N. \]
4. Find \(I(y) = \int_a^b \bar{\rho}(x) \, dx\), a piecewise linear function. Specifically, \(I(y_0) = 0\), and 
\[ I(y_j) = \sum_{r=1}^{j} (y_r - y_{r-1}) \left( \frac{\rho_{r-1} + \rho_r}{2} \right) \quad j = 1, \ldots, N. \]
5. Using linear interpolation (2.3), find \(\{z_j\}\) such that 
\[ I(z_j) = \frac{j}{N} I(b). \]
6. If \(|\bar{z} - \bar{y}| < TOL\), set \(\bar{y} = \bar{z}\) and STOP, else, set \(\bar{y} = \bar{z}\) and return to Step 2.

### 2.1.4 Simplistic Moving Mesh Method

Solving a PDE whose physical solution changes both spatially and temporally, typically results in a monitor function, \(\rho(x, t)\), that also changes spatially and temporally. Building on the idea of (2.1), a moving mesh algorithm finds a set of mesh points, \(\{x_j(t)\}, j = 0, \ldots, N\), such that 
\[ \int_{x_{j-1}(t)}^{x_j(t)} \rho(x, t) \, dx = \frac{\int_{x_0(t)}^{x_N(t)=b(t)} \rho(x, t) \, dx}{N}, \quad j = 1, \ldots, N, \quad \forall t. \quad (2.4) \]

In most moving mesh literature, the endpoints are held fixed, \(a(t) = a\) and \(b(t) = b\), and a normalized monitor function, \(\bar{\rho}\), is defined,
\[ \bar{\rho}(x, t) = \frac{\rho(x, t)}{\int_a^b \rho(x, t) \, dx} = \frac{\rho(x, t)}{L(t)}. \quad (2.5) \]
CHAPTER 2. MOVING MESH

The term \( L(t) \), a time-dependent function, is used to denote the integral of the monitor function on the entire domain. Using the normalized monitor function, the equi-distribution problem simplifies to finding mesh trajectories \( x_j(t) \) such that

\[
\int_{a(t)}^{x_j(t)} \tilde{\rho}(x, t) \, dx = \frac{j}{N}, \quad j = 0, \ldots, N, \quad \forall t.
\]

Numerically, a simple way to generate a moving mesh in one dimension is to implement de Boor's algorithm at each discrete time level (i.e., we regenerate a new equi-distributed mesh at each time integration step). There are several advantages and disadvantages to using such an implementation. While the algorithm is generally successful at generating a sequence of equi-distributed grid points \( \{x_j(t^n)\} \), there is no control over how smoothly the mesh changes from time level \( t^n \) to time level \( t^{n+1} \). Also, an interpolation of the solution on the old mesh to the solution on the new mesh, \( \tilde{u}_j^{n+1} \approx u(x_j(t^n), t^{n+1}) \rightarrow u_j^{n+1} \approx u(x_j(t^{n+1}), t^{n+1}) \), is required at each step.

To avoid this interpolation step, a seminal paper [Huang et al., 1994b], describes moving mesh partial differential equations (MMPDE) based on the equi-distribution principle. The idea is to find some mesh velocity, \( v \), which will update the mesh, and solve the physical PDE with a corresponding Lagrangian correction term (see Section 2.5.2). We follow their outline, generalizing to allow the total number of grid nodes to vary.

2.2 Moving Mesh PDE (MMPDE)

In this section, we present a derivation of the MMPDEs, a set of equations whose solution gives mesh velocities \( v(x, t) \), which move a mesh towards equi-distribution.

2.2.1 Derivation

In MMPDE algorithms, there is a computational grid, \( \xi \) (which is normally equi-spaced), and an associated mapping \( x(\xi, t) \) which gives the location of the grid points at time \( t \). We refer the reader back to Section 2.1.2 for a discussion and illustration of the computational versus the logical grid. In this section, we use ideas from equi-distribution to derive some
of the MMPDE algorithms. The equi-distribution equation (2.4) can be expressed as

$$\int_{a(t)}^{x_j(t)} \rho(\tilde{x}, t) \, d\tilde{x} = \frac{j}{N(t)} \int_{a(t)}^{b(t)} \rho(\tilde{x}, t) \, d\tilde{x}, \quad j = 1, \ldots, N(t). \tag{2.6}$$

Here, $N(t)$ dictates the number of sub-intervals and unlike [Huang et al., 1994b], $N(t)$ is not constrained to be an integer. In the case where $N(t)$ is not an integer, this means physically that one does not have a grid point on both domain boundaries, $a(t)$ and $b(t)$, simultaneously. We define the integral of $\rho(x, t)$ over the domain as

$$L(t) = \int_{a(t)}^{b(t)} \rho(x, t) \, dx,$$

and define $\xi$ to be the piecewise linear extension of

$$\xi_j = \frac{j}{N(t)}, \quad k = 0, \ldots, \text{floor}(N(t)).$$

Generalizing equation (2.6), we obtain the expression

$$\int_{a(t)}^{x_j(t)} \rho(\tilde{x}, t) \, d\tilde{x} = \xi L(t). \tag{2.7}$$

Observe that

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

and if $N(t)$ is an integer,

$$x(1, t) = b(t).$$

We differentiate equation (2.7), the integral formulation of the equi-distribution principle, with respect to $\xi$ to obtain

$$\rho(x(\xi, t), t) \frac{\partial}{\partial \xi} x(\xi, t) = L(t),$$

and differentiate again with respect to $\xi$ to obtain a differential form of the equi-distribution equation,

$$\frac{\partial}{\partial \xi} \left\{ \rho(x(\xi, t), t) \frac{\partial}{\partial \xi} x(\xi, t) \right\} = 0. \tag{2.8}$$

Notice that the differential form of the equi-distribution equation (2.8) does not depend on $L(t)$ explicitly.
CHAPTER 2. MOVING MESH

Suppose that a mesh satisfies the differential form of the equi-distribution principle (2.8). We also want the mesh to satisfy the equi-distribution equation (2.8) at some later time, \( t + \tau \), where \( 0 < \tau \ll 1 \). Mathematically we want

\[
\frac{\partial}{\partial \xi} \left\{ \rho(x(\xi, t + \tau), t + \tau) \frac{\partial}{\partial \xi} x(\xi, t + \tau) \right\} = 0. \tag{2.9}
\]

We substitute Taylor expansions for \( x(\xi, t + \tau) \) and \( \rho(x(\xi, t + \tau), t + \tau) \),

\[
\frac{\partial}{\partial \xi} x(\xi, t + \tau) = \frac{\partial}{\partial \xi} x(\xi, t) + \tau \frac{\partial}{\partial \xi} x_t(\xi, t) + O(\tau^2),
\]

\[
\rho(x(\xi, t + \tau), t + \tau) = \rho(x(\xi, t), t) + \tau x_t(\xi, t) \frac{\partial}{\partial x} \rho(x(\xi, t), t) + \tau \frac{\partial}{\partial t} \rho(x(\xi, t), t) + O(\tau^2),
\]

into the perturbed equi-distribution equation (2.9). Dropping higher order terms, we obtain an expression for the mesh velocities, \( v = x_t \), also known in the literature as MMPDE2,

\[
\frac{\partial}{\partial \xi} \left( \rho \frac{\partial v}{\partial \xi} \right) + \frac{\partial}{\partial \xi} \left( \rho \frac{\partial v}{\partial \xi} \right) = -\frac{\partial}{\partial \xi} \left( \frac{\partial \rho}{\partial t} \xi \xi \right) - \frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right). \tag{2.10}
\]

If \( \tau \) is small, the mesh velocities, \( v \), move the mesh towards one which equi-distributes \( \rho(x, t) \). In practice, \( \frac{\partial \rho}{\partial t} \) is a term that is difficult to calculate accurately; an estimate of \( \rho \) is needed at various time levels to approximate \( \frac{\partial \rho}{\partial t} \). Also observe that when \( x(\xi, t) \) is not equi-distributed, (2.10) moves the mesh closer to equi-distribution, even when \( \rho(x, t) \) is independent of \( t \). [Huang et al., 1994b] argue to drop the term \( \frac{\partial \rho}{\partial \xi} \frac{\partial v}{\partial \xi} \), or both \( \frac{\partial \rho}{\partial \xi} \frac{\partial v}{\partial \xi} \) and \( \frac{\partial \rho}{\partial \xi} \). Respectively, these choice give simplified equations for the mesh velocities, \( v \),

\[
\frac{\partial^2}{\partial \xi^2} (\rho v) = -\frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right), \tag{2.11}
\]

and

\[
\frac{\partial}{\partial \xi} \left( \rho \frac{\partial v}{\partial \xi} \right) = -\frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right). \tag{2.12}
\]

In the literature, equation (2.11) is known as MMPDE3 and equation (2.12) is known as MMPDE4 [Huang et al., 1994a]. We utilize MMPDE4 in most of our simulations in Chapter 6. It is instructive to look at a centered-difference approximation of (2.12). Assuming that the spacing in the variable \( \xi \) is uniform, the discretized version of MMPDE4 is

\[
\rho_{i+\frac{1}{2}} (v_{i+1} - v_i) - \rho_{i-\frac{1}{2}} (v_i - v_{i-1}) = -\frac{1}{\tau} \left( \rho_{i+\frac{1}{2}} (x_{i+1} - x_i) - \rho_{i-\frac{1}{2}} (x_i - x_{i-1}) \right). \tag{2.13}
\]
If we consider \( \rho_{i+\frac{1}{2}}(x_{i+1} - x_i) \) as the midpoint approximation to \( \int_{x_{i+1}}^{x_i} \rho \, dx \), the right hand side of equation (2.13) can be interpreted as \( \frac{1}{\tau} \) times the difference of the integrals of \( \rho \) in the interval \([x_{i-1}, x_i]\) and in the interval \([x_i, x_{i+1}]\). When the integral of \( \rho \) in the right interval, \([x_i, x_{i+1}]\), is larger than the integral in the left interval, \([x_{i-1}, x_i]\), the right hand side of (2.13) is negative (if \( \rho > 0 \)). If the width of the interval \([x_i, x_{i+1}]\) is held fixed, i.e. \( v_{i+1} = v_i \), then the interval \([x_{i-1}, x_i]\) must increase in width to increase \( \int_{x_{i-1}}^{x_i} \rho \). Similar logic is consistent when the integral in the left interval is larger than the integral in the right region. This is the so-called attraction and repulsion force felt between mesh nodes, as described in [Huang et al., 1994a].

Alternatively, an equation for mesh velocities can also be derived by considering deviations in the integral of \( \rho \) from some average value. [Adjerid & Flaherty, 1986] follow a similar derivation where the integral of the monitor function is a measure of truncation error. Denoting the integral of the monitor function in each subinterval as \( d_j = \int_{x_{j+1}}^{x_j} \rho(x, t) \, dx \), we use an attraction and repulsion argument as described above to justify an expression,

\[
v_{j+1} - v_j = -\lambda (d_j - \bar{d}), \tag{2.14}
\]

for mesh velocities \( v \). Here, \( \bar{d} \) is some average of \{\( d_j \)\}, and \( \lambda \) is some positive parameter. We eliminate \( \bar{d} \) from equation (2.14) by differencing two consecutive intervals to obtain

\[
v_{j+1} - 2v_j + v_{j-1} = -\lambda (d_j - d_{j-1}). \tag{2.15}
\]

If we take the discrete approximation to the integral of the monitor function in each subinterval,

\[
d_j \approx \rho_{j+1/2}(x_{j+1} - x_j),
\]

we obtain

\[
v_{j+1} - 2v_j + v_{j-1} = -\lambda \left( \rho_{j+1/2}(x_{j+1} - x_j) - \rho_{j-1/2}(x_j - x_{j-1}) \right). \tag{2.16}
\]

Denoting \( \lambda \) by \( \frac{1}{\tau} \), equation (2.16) is a centered difference approximation to

\[
\frac{\partial^2 \tilde{x}}{\partial \xi^2} = \frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial \tilde{x}}{\partial \xi} \right), \tag{2.17}
\]

which is known as MMPDE6 in the literature. It has been shown in [Budd et al., 1996b] that (2.12) does not evolve the mesh quickly enough to resolve a peak in a blow-up problem.
if $\tau$ is fixed. By using (2.17) with a suitable monitor function, some scaling invariance in the Frank-Kamenetskii blow-up problem can be preserved. We utilize MMPDE6 for a test blow-up problem in Chapter 6.

We note that generally, the MMPDEs affect the mesh globally, i.e., when $\rho$ changes in a compact region, the entire mapping $x(\xi, t)$ is affected.

### 2.2.2 Boundary Conditions

Dirichlet boundary conditions for the mesh velocities,

\begin{align*}
v(a(t)) &= c_1(t), \\
v(b(t)) &= c_2(t),
\end{align*}

are often imposed for the solution of MMPDE4 (2.12); this choice has the advantage that it gives a unique solution for mesh velocities. These Dirichlet boundary conditions are reasonable under two assumptions: (i) the evolution of the domain boundary is known explicitly, and (ii) a mesh node remains on a domain boundary for all time if it starts on a domain boundary,

\[ x_0(t) = a(t), \quad x_N(t) = b(t). \]

We note that there are many problems which satisfy the above two assumptions. For example, solutions to the porous medium equation have expressions for the velocity of nodes on the boundary domain. This second condition also includes particularly common scenarios where the domain interval remains fixed, i.e.,

\[ a(t) = a, \quad b(t) = b. \]

The choice $c_1 = c_2 = 0$ in (2.18) keeps the grid nodes fixed on the boundary domain, i.e. $x_0 = a, x_N = b$.

This approach of determining Dirichlet boundary conditions has an additional consequence. By enforcing that grid nodes remain on the domain boundary throughout the computation, it doesn’t allow for the natural addition or removal of grid nodes through the boundary. This restriction will be more apparent in Chapter 4 when the new hybrid method and boundary conditions are introduced.
2.2.3 Discussion

In practice, the MMPDE approach has some stability issues. These issues stem from the use of the MMPDEs as a "velocity-based" method instead of a "location-based" method. Specifically, we solve for mesh velocities, \( v \), and use those mesh velocities to update the mesh, \( x_t = v \), whereas the method is primarily interested in the mapping transformation to give the location of the grid nodes. Stability problems often first appear as oscillations in the mesh velocities field, i.e., the plot of the mesh velocities versus the spatial variable exhibits oscillations. Several techniques are commonly used to add stability to the system.

[Huang & Russell, 1997] analyze the process of smoothing the monitor function to make the transformation mapping smooth, while [Bank & Smith, 1997] discusses algorithms to smooth the mesh. The new hybrid method that we propose in Chapter 4 essentially performs a smoothing of the mesh velocities, accomplished by interpolation of mesh velocities from the physical to the computational grid.

Using smoothing to add stability to the MMPDEs is reasonable. Firstly, it is more important that there is a clustering of mesh nodes in the desired area; the exact configuration of the mesh is less important in many problems. Secondly, the MMPDEs will drive the mesh towards equi-distribution, even if it has strayed from an equi-distributed state.

We finish our discussion by noting that the above derivation of the MMPDEs does not extend easily to higher dimensions. In [Huang & Russell, 1998], the authors give a higher dimensional generalization of the MMPDE strategy. It involves a higher dimensional version of the adaptation function, studied in [Cao et al., 1999b], a gradient flow equation, and a variational approach to derive the MMPDE in higher dimensions. We instead switch to a discussion of another moving mesh method which uses conservation of geometric quantities as its basic framework. This moving mesh method, known as the Geometric Conservation Law (GCL), is easily derived and solved for one, two, and three spatial dimensions.

2.3 Geometric Conservation Law (GCL)

2.3.1 Derivation

The engineering community has utilized the idea of volume conservation in their derivation of so-called finite volume schemes for many years. [Cao et al., 2002] utilize this argument of volume preservation to devise a moving mesh scheme, referred to as the GCL. We perform
this derivation in two dimensions, though the argument is easily carried out in one and three dimensions.

Let $A_c$ be an arbitrary, fixed cell in the computational domain with boundary $\partial A_c$ and let $A(t)$ be the corresponding cell in the physical domain under the time-dependent coordinate transform $x(\xi, t)$,

$$A(t) = \{ x \mid x = x(\xi, t), \ \forall \xi \in A_c \}.$$  

The GCL imposes that the change in the area of $A(t)$ equals the total flux through the surface $\partial A(t)$,

$$\frac{d}{dt} \int_{A(t)} dx = \int_{\partial A(t)} x_t \cdot dS.$$  \hspace{1cm} (2.19)

In Figure 2.2, we illustrate a mapping of a cell in the physical domain at time $t_1$ under the action of the mesh velocity, $v$.

Figure 2.2: GCL: The change in area of $A(t)$ equals the total flux through the surface $\partial A(t)$.

Changing variables, the left-hand side of equation (2.19) can be rewritten as

$$\frac{d}{dt} \int_{A(t)} dx = \frac{d}{dt} \int_{A_c(t)} J(\xi, t) d\xi,$$

$$= \int_{A_c} \frac{D}{Dt} J(\xi, t) d\xi,$$  \hspace{1cm} (2.20)

where $J(\xi, t)$ is the corresponding determinant of the Jacobian, and $\frac{D[\cdot]}{Dt}$ is the material time derivative,

$$\frac{D[\cdot]}{Dt} = \frac{\partial [\cdot]}{\partial t} + \dot{x} \cdot \nabla [\cdot].$$

Using the divergence theorem and changing variables, the right-hand side of equation (2.19)
can be rewritten for the determinant of the Jacobian,
\[
\int_{\partial A(t)} \hat{x} \cdot dS = \int_{A(t)} \nabla \cdot \hat{x} \, dx,
\]
\[
= \int_{A_c} (\nabla \cdot \hat{x}) \, J(\xi, t) \, d\xi.
\] (2.21)

Combining (2.20) and (2.21), and observing that \( A_c \) can be arbitrarily selected in the computational domain, we arrive at the differential form of the GCL,
\[
\nabla \cdot \hat{x} = \frac{1}{J} \frac{D}{Dt} J.
\] (2.22)

In one dimension, we can find an expression for the GCL in terms of our monitor function by differentiating the integral form of the equi-distribution equation (2.6) with respect to \( x \), and obtaining an expression for the determinant of our Jacobian,
\[
\rho(x, t) = \frac{\partial x}{\partial \xi} L(t), \quad \implies J = \frac{\partial x}{\partial \xi} = \frac{L(t)}{\rho(x, t)}.
\]
If we use this expression for the determinant of our Jacobian in higher dimensions,
\[
J = \frac{L(t)}{\rho(x, t)},
\]
the differential form of the GCL, (2.22) becomes
\[
\nabla \cdot \hat{x} = \frac{1}{J} \frac{D}{Dt} J,
\]
\[
= \left( \frac{\rho(x, t)}{L(t)} \right) \left( \frac{D}{Dt} \left( \frac{L(t)}{\rho(x, t)} \right) \right),
\]
\[
= \frac{\rho(x, t)}{L(t)} \left( \frac{\partial}{\partial t} \left( \frac{L(t)}{\rho(x, t)} \right) + \hat{x} \cdot \nabla \left( \frac{L(t)}{\rho(x, t)} \right) \right),
\]
\[
= \frac{\rho(x, t)}{L(t)} \left( \frac{1}{\rho} \frac{\partial L}{\partial t} - \frac{L(t)}{\rho^2(x, t)} \frac{\partial \rho}{\partial t} - \frac{L(t)}{\rho^2(x, t)} \hat{x} \cdot \nabla \rho(x, t) \right).
\]
Thus,
\[
\rho(x, t) \nabla \cdot \hat{x} = \frac{\rho(x, t)}{L(t)} \frac{\partial L}{\partial t} - \frac{\partial \rho(x, t)}{\partial t} - \hat{x} \cdot \nabla \rho(x, t),
\] (2.23)
\[
\implies \nabla \cdot (\rho(x, t) \hat{x}) + \frac{\partial \rho}{\partial t} = \frac{\rho(x, t)}{L(t)} \frac{\partial L}{\partial t}.
\]
Normalizing \( \rho(x, t) \) results in the GCL equation derived in [Cao et al., 2002]. Specifically, recall the normalized monitor function (2.5),
\[
\hat{\rho}(x, t) = \frac{\rho(x, t)}{L(t)}.
\]
Substituting \( \rho = \tilde{\rho} L \) into (2.23) gives

\[
L \nabla \cdot (\tilde{\rho} \dot{x}) + \frac{\partial}{\partial t}(\tilde{\rho} L) = \tilde{\rho} \frac{\partial L}{\partial t},
\]

Since

\[
\frac{\partial}{\partial t}(\tilde{\rho} L) = L \frac{\partial \tilde{\rho}}{\partial t} + \tilde{\rho} \frac{\partial L}{\partial t},
\]

we obtain

\[
\nabla \cdot (\rho(x,t)\dot{x}) + \frac{\partial \rho}{\partial t} = 0,
\]

(2.24)

where we drop the tildes for ease of notation. This moving mesh equation is more “local” in the sense that it relies on \( \frac{\partial \rho}{\partial t} \) to calculate the appropriate mesh velocities. Unlike MMPDE4 however, there is no built in structure to bring the mesh closer to an equi-distributed state at every time step. We note that GCL equation (2.24) holds in higher dimensions if \( J = c/\rho \).

Also, the Helmholtz decomposition theorem tells us that a continuous and differentiable vector field can be decomposed into two orthogonal components: the gradient of a scalar field and the curl of a vector field. Since the GCL equation (2.24) only specifies a condition on the divergence of the mesh velocity, an additional condition on the curl of the velocity field must be imposed. We choose a curl-free condition

\[
\nabla \times w(v - u) = 0,
\]

(2.25)

where \( w \) is a weight function and \( u \) is a background velocity field. One typically makes the choice \( u = 0 \) unless there is some inherent background velocity in the problem. Authors in [Cao et al., 2002] claims that \( w = 1 \) is a superior choice compared to \( w = \rho \), the deformation method used in [Liao et al., 2002]. In either case, the above curl condition implies that there is a potential function, \( \phi \), whose gradient gives the mesh velocities,

\[
\dot{x} = v = u + \frac{1}{w} \nabla \phi.
\]

(2.26)

Substituting the expression for \( \dot{x} \) into the GCL equation (2.24) results in an elliptic system for \( \phi \),

\[
\nabla \cdot \left( \frac{\rho(x,t)}{w} \nabla \phi \right) + \frac{\partial \rho}{\partial t} = 0,
\]

(2.27)

where the velocity is computed afterwards by calculating by \( v = \nabla \phi \). A detailed discussion can be found in [Cao et al., 2002].
2.3.2 Boundary Conditions

Boundary conditions for the GCL method are slightly tricky. The difficulty arises from the introduction of the potential function $\phi$ in equation (2.26), which is defined such that the velocity of the mesh nodes is the gradient of the potential function,

$$v = \nabla \phi,$$

provided the background velocity $u = 0$, and the weight function $w = 1$. So, even though we solve for the potential function $\phi$ in the elliptic equation (2.27), the quantity we are most interested in is the gradient of $\phi$, which gives the mesh velocities. Thus, provided that $\phi$ is computed up to an arbitrary constant, we still obtain the same velocities for the mesh speed:

$$v = \nabla \phi = \nabla (\phi + \alpha),$$

where $\alpha$ is some arbitrary constant. How does this “non-uniqueness” of $\phi$ play out in terms of the boundary condition? Recall that Dirichlet boundary conditions provide a unique solution for the MMPDE approach,

$$v(a(t)) = c_1(t),$$

$$v(b(t)) = c_2(t).$$

Since mesh velocities are given as the gradient of the potential function, $\phi$, Neumann conditions seem to be reasonable boundary conditions for the GCL approach,

$$\phi_x(a(t), t) = v(a(t)) = c_1(t),$$

$$\phi_x(b(t), t) = v(b(t)) = c_2(t).$$

To specify a unique $\phi$, i.e. solve a well-posed problem, we specify $\phi$ somewhere in the domain as well, for example,

$$\phi(x_j) = 0.$$

As before, the assumptions for these Neumann conditions are that (i) the evolution of the boundary is known explicitly, and (ii) a grid node remains on a boundary for all time if it starts on the boundary. In the special case that the domain remains fixed, we obtain the Neumann conditions

$$\phi_x(a, t) = \phi_x(b, t) = 0.$$
In Chapter 4, we develop a new boundary condition which allows for the natural creation and removal of grid nodes.

### 2.3.3 Discussion

If a mesh is equi-distributed, the GCL method uses equation (2.27) to derive a set of mesh velocities to keep the mesh equi-distributed at a later time. Unlike the MMPDE, however, the GCL method doesn't move a mesh towards equi-distribution if the mesh has strayed away from its equi-distributed state. The current formulation lacks the "stabilizing" effect of the MMPDE, which was obtained by solving a perturbed version of the equi-distribution equation (2.9). Finding a way to stabilize the GCL method seems like a plausible endeavor, but is beyond the scope of this thesis.

### 2.4 Other Moving Mesh Methods

There are many other moving mesh methods which have been proposed for solving time dependent partial differential equations. We point the reader to [Adjerid & Flaherty, 1986] for more information about moving finite element methods (MFE Methods), [Liao et al., 2002] for details about the deformation method, [Li et al., 2001] for a Harmonic Mapping approach, and variational approaches in [Ceniceros & Hou, 2001].

### 2.5 Coupling with Physical Equations

Moving mesh methods have proven to be quite successful in solving 1-D PDEs. We describe two main approaches to coupling with the physical equation: (i) an inter-weaving approach, and (ii) a Lagrangian correction to the physical PDE.

#### 2.5.1 Inter-weaving Approach

The inter-weaving approach is probably the most familiar and intuitive way of implementing a moving mesh framework. The general idea is to freeze the mesh while solving the physical PDE, update the mesh when necessary, and interpolate the physical solution from the old mesh to the new mesh. A simple implementation by which the mesh is updated after every
time-step is listed below:

| Step 1: Solve the physical PDE on a frozen mesh for $\tilde{u}_j^{n+1} \approx u(x_j^n, t^{n+1})$. |
| Step 2: Solve for the mesh velocities, either using the MMPDE or the GCL. |
| Step 3: Solve for the new mesh, $x_j^{n+1}$. |
| Step 4: Interpolate $\tilde{u}_j^{n+1} \approx u(x_j^n, t_{n+1})$ from the old mesh to the new mesh, $u_j^{n+1} \approx u(x_j^{n+1}, t_{n+1})$. |
| Step 5: Set $n = n + 1$, and return to Step 1. |

A drawback of this algorithm is stability, which presents itself as a "lagging" effect. Specifically, important features of the physical solution may be missed if the mesh is frozen in Step 1, resulting in stability issues while solving the physical PDE. There are various solutions to correct this phenomenon, such as an iterative approach to updating the mesh and physical solution simultaneously. The Lagrangian correction approach listed below addresses this problem, and additionally circumvents the need for interpolation.

### 2.5.2 Lagrangian Correction

The inter-weaving approach, while intuitive, requires an interpolation step, which could potentially introduce instability (lagging effect) into the numerical computation. A different approach is to reformulate a physical PDE to account for the moving frame of reference. Take for example Burgers' equation,

$$u_t + uu_x = \epsilon u_{xx}.$$  

In a moving framework, Burgers' equation becomes

$$\frac{d}{dt} u(x(t), t) + u_x = \epsilon u_{xx}.$$  

Expanding the derivatives, Burgers' equation with the so called Lagrangian correction becomes

$$u_t + v u_x + uu_x = \epsilon u_{xx},$$

where $v = x_t$ are the velocities of the mesh nodes.
CHAPTER 2. MOVING MESH

To solve a physical PDE on a moving mesh, the idea is to solve the physical PDE with the Lagrangian correction and update the mesh simultaneously. For example, given an initial mapping \( x(\xi, 0) \) and an initial solution \( u(x(\xi, 0), 0) \) to Burgers' equation, we solve the physical PDE on a moving mesh by solving the following systems of equations:

\[
\begin{align*}
  u_t + v u_x + uu_x &= \xi u_{xx}, \\
  \frac{\partial}{\partial \xi} \left( \rho \frac{\partial v}{\partial \xi} \right) &= -\frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right), \\
  x_t &= v.
\end{align*}
\]

The first equation (2.28a), is Burgers' equation with the Lagrangian correction term, and (2.28b) is a moving mesh PDE. The third equation (2.28c) evolves the transformational mapping. (A discretized version of (2.28c) corresponds to a system of ODEs used to explicitly update the mesh location.) In (2.28a), the Lagrangian correction term circumvents the need for interpolation. In fact, one can show that \((v\Delta t)u_x\) is really a linear interpolation in a forward Euler discretization scheme, i.e.,

\[
u(x_j(t^{n+1}), t^{n+1}) \approx u(x_j(t^n), t^{n+1}) + (v\Delta t)u_x(x_j(t^n), t^{n+1}).\]

We illustrate this graphically in Figure 2.3.

It may be possible to generalize this connection between the Lagrangian correction and interpolation for different time stepping schemes, but it is beyond the scope of this thesis. It should also be observed that this Lagrangian correction is only meaningful if the number of mesh points remains the same in the problem, and \( x(\xi, t) \) is truly a mapping from the computational grid to the physical grid. If the number of grid points changes, there is no avoiding the interpolation step. We discuss some possible improvements in Chapter 4.
If a forward Euler discretization is used, the Lagrangian correction is, to first approximation, a linear interpolation of the solution at an old mesh to a solution at the new mesh.
Chapter 3

Level Set Framework

In an explicit representation of a mesh, one knows exactly the location of the grid points \{x_i\}. One alternative is an *implicit* representation where grid points are represented by iso-contours of a level set function, or the intersection of iso-contours of several level set functions. What does this mean exactly?

Readers are perhaps more familiar with the idea of a level set function being an implicit interface representation. In particular, the zero iso-contour normally separates a domain into an “interior” and “exterior” region. For example, in \( \mathbb{R}^1 \), the zero iso-contour of \( \psi(x) = x^2 - 1 \) is the set of all points for which \( \psi(x) = 0 \), namely, \{−1, 1\}. The zero iso-contour separates \( \psi \) into two regions,

\[
\begin{align*}
\psi(x) < 0 & \quad \text{when } x \in (-1, 1), \\
\psi(x) \geq 0 & \quad \text{when } x \notin (-1, 1).
\end{align*}
\]

If we now view the different iso-contours of a level set function as separating “regions” or “sub-intervals” in \( \mathbb{R}^1 \), then a discrete set of iso-contours gives the location of the mesh points! In Section 3.1, we discuss the level set representation of a static mesh that equi-distributes some monitor function. Then in Section 3.2, we discuss how to represent a mesh using level set functions in higher dimensions. We follow the discussion of representing grids by deriving an evolution equation that evolves the level set function to implicitly move the represented grid.

First, we provide a brief history outlining the development of level set functions and their applications to grid generation. The use of level set functions to implicitly represent the evolution of surfaces was first introduced in [Osher & Sethian, 1988]. In [Sethian, 1994],
level set functions are used to generate a static grid, while in [Liao et al., 2000], some pre-
liminary work is shown on how to use level set functions to represent and move the grids. While Liao et al.’s approach of using a deformation mesh movement strategy to evolve a level set representation of the grid was novel, there was no discussion of the advantages, or motivation for their new implementation.

In Section 3.4, we discuss the advantages of using level set functions to represent a mesh, as well as subtleties and potential challenges in using a level set representation of the mesh.

3.1 Level Set Representation of a Static Equi-distributed 1D Mesh

Recall that for a given monitor function $\rho(x)$, finding an equi-distributed mesh in 1D means finding a set of mesh nodes, ${x_j}$, such that

$$\int_a^{x_j} \rho(\tilde{x}) \, d\tilde{x} = \frac{j}{N} \int_a^b \rho(\tilde{x}) \, d\tilde{x}, \quad j = 0, \ldots, N.$$ 

The continuous extension is to find the mapping $x(\xi)$ such that

$$\int_a^{x(\xi)} \rho(\tilde{x}) \, d\tilde{x} = \xi L,$$

where $\xi$ is the piecewise linear extension of $\frac{j}{N}$ and

$$L = \int_a^b \rho(\tilde{x}) \, d\tilde{x}.$$

If we define our level set function as

$$\psi(\eta) = \int_a^\eta \rho(x) \, dx,$$  \hspace{1cm} (3.1)

and initialize a discrete set of iso-contour levels as

$$c_j = \frac{j}{N} \int_a^b \rho(x) \, dx, \quad j = 0, \ldots, N,$$  \hspace{1cm} (3.2)

then finding the iso-contours of $\psi(\eta)$,

$$\{x_j\} = \left\{ x_j \Big| \psi(x_j) = c_j \quad \text{where} \; c_j = \frac{j}{N} L \right\}, \quad j = 0, \ldots, N,$$  \hspace{1cm} (3.3)
is equivalent to finding the equi-distributed mesh given \( \rho(x) \), since we are trying to find \( \{x_j\} \) such that

\[
\psi(x_j) = c_j, \\
\int_a^{x_j} \rho(x) \, dx = \frac{j}{N} L.
\]

In Figure 3.1, we illustrate a level set representation of a mesh which equi-distributes the arc-length of a prescribed function \( u(x) \). In the top left plot, the graph of \( u(x) \), a Gaussian peak centered around \( x = 0.5 \), is shown. In the top right plot, we graph the corresponding arc-length monitor function, \( \rho(u(x)) = \sqrt{1 + u(x)^2} \). In the bottom left plot, we set the level set function using the initialization formula (3.1). The contour levels of \( \psi \) are drawn as horizontal dotted lines in the plot. The grid nodes corresponding to the iso-contours of \( \psi \) are plotted as circular markers. In the bottom right plot, the graph of \( u(x) \) is overlaid with the equi-distributed mesh. One can see visually that the arc-length in each sub-interval is constant.

In the above construction (3.1), \( \psi(\eta) \) is a monotonically increasing function, so each level contour of the level set function is a single point provided \( \rho > 0 \). An alternative construction of a level set function to represent a static, equi-distributed 1-D mesh is

\[
\psi(\eta) = \int_a^{\eta} -\rho(x) \, dx.
\]

The function \( \psi(\eta) \) is now a monotonically decreasing function, so each level contour of the level set function once again corresponds to a single point, provided \( \rho > 0 \).

In fact, a non-monotonic level set function can be used to represent a mesh. In Figure 3.2, we illustrate a non-monotonic level set function giving rise to several grid points for some level contours, and re-iterate that the level set representation of a mesh is not unique. Figure 3.3 gives a specific example, showing two different level set functions representing a uniform mesh: the equi-spaced contours of the two level set functions, \( \psi_a(x) = x + 1 \), \( \psi_b(x) = |x| \), give rise to the same mesh on \([-1, 1]\) if \( c = [0 : 0.1 : 2] \).

Choosing a suitable level set function to represent an equi-distributed 1D mesh is still an open research problem. In Chapter 6, we show that a monotonic level set representation of a
CHAPTER 3. LEVEL SET FRAMEWORK

Figure 3.1: In (a), the graph of a function $u(x)$ is plotted. The function $u(x)$ is chosen to be a Gaussian peak centered around $x = 0.5$. In (b), the corresponding arc-length monitor function associated with $u(x)$ is plotted. Notice that it looks double-peaked because the graph of $u(x)$ is flat near the $x = 0.5$. In (c), the level set function is defined to be the integral of the monitor function. Notice that $\psi$ changes rapidly when $\rho$ is large. The contour levels are shown as dotted lines. Where these dotted contour lines intersect the level set function, we find the equi-distributed mesh nodes. Finally, in (d) the resulting equi-distributed mesh overlays the function $u(x)$.
CHAPTER 3. LEVEL SET FRAMEWORK

3.1 Non-Monotonic Level Set Functions

A non-monotonic level set function may give rise to multiple grid points for some iso-contour levels, e.g., \( \{ x | \psi(x) = 0 \} = \{ 0.5, 0.8, 1.9 \} \).

Figure 3.2: A non-monotonic level set function may give rise to multiple grid points for some iso-contour levels, e.g., \( \{ x | \psi(x) = 0 \} = \{ 0.5, 0.8, 1.9 \} \).

3.2 Level Set Representation of a Static Higher Dimensional Mesh

In \( \mathbb{R}^1 \), the mesh nodes are given by the level contours of a level set function, \( \psi(\eta) \). In higher dimensions, we use the level contours of multiple level set functions to generate our mesh. Specifically, for a 2D mesh, we use the intersection of level contours from two level set functions, \( \psi_1(\eta) \) and \( \psi_2(\eta) \), to define the mesh. For a 3D mesh, the intersection of level contours from three level set functions, \( \psi_1(\eta) \), \( \psi_2(\eta) \) and \( \psi_3(\eta) \), define the mesh. We first discuss the concept of iso-surfaces, before explaining how to represent a mesh using level set functions in higher dimensions.

3.2.1 Level Contours in Higher Dimensions

We limit our discussion to co-dimension one objects, namely, the cases where the iso-surfaces are one dimension “less” than the spatial dimension. We use the terminology iso-surface, iso-contour or level contour interchangeably for the rest of the thesis. In \( \mathbb{R}^1 \) (\( n = 1 \)), our iso-surfaces are \( n - 1 = 0 \) dimensional objects, or discrete points. In \( \mathbb{R}^2 \) (\( n = 2 \)), our iso-surfaces are one dimensional objects, or curves. In \( \mathbb{R}^3 \) (\( n = 3 \)), our iso-surfaces are two dimensional objects, or surfaces. Mathematically, the \( k^{th} \) level set, or the \( k^{th} \) iso-contour of
Figure 3.3: Here, two different level set functions represent the same mesh. In the left plot, $\psi_a(\eta) = \eta + 1$. In the right plot, $\psi_b(\eta) = |\eta|$. Both functions have level contours which result in the same equi-spaced grid (cross markers) on [-1,1] if $\eta = [0 : 0.1 : 2]$. 
a function $\psi(\vec{n})$ is defined as \( \{ \vec{n} | \psi(\vec{n}) = c_k \} \) where \( c_k \) is the \( k \text{th} \) component of \( \vec{c} \), a vector with constant entries. We illustrate the level contours of \( \psi(x, y) = \sqrt{x^2 + y^2} \) in Figure 3.4. Note that we have expanded \( \vec{n} = (x, y) \) for clarity. Since level contours of \( \psi(x, y) \) satisfy

\[
\psi(x, y) = \sqrt{x^2 + y^2} = c,
\]

or equivalently,

\[
x^2 + y^2 = c^2,
\]

the level contours of \( \psi(x, y) = \sqrt{x^2 + y^2} \) are concentric circles centered at \((0, 0)\).

![Figure 3.4: Level set contours of $\psi(x, y) = \sqrt{x^2 + y^2}$ are concentric circles centered about $(0, 0)$.](image)

### 3.2.2 Intersection of Level Contours in $\mathbb{R}^2$

Given two level set functions, \( \psi_1(\vec{n}) \) and \( \psi_2(\vec{n}) \), we define the location of mesh nodes as the intersection of a discrete set of \( \psi_1(\vec{n}) \) iso-contours with a discrete set of \( \psi_2(\vec{n}) \) iso-contours. Specifically, we initialize the iso-contours of \( \psi_1(\vec{n}) \) as

\[
c_j = \frac{j}{N_1} \Delta \psi_1, \quad j = 0, \ldots, N_1,
\]
where $\Delta \psi_1$ is the spacing between the iso-contour levels of $\psi_1(\vec{r})$, and $N_1$ is the number of $\psi_1(\vec{r})$ iso-contour levels of interest. Similarly, we initialize the iso-contours of $\psi_2(\vec{r})$ as

$$d_k = \frac{k}{N_2} \Delta \psi_2, \quad k = 0, \ldots, N_2,$$

where $\Delta \psi_2$ is the spacing between the iso-contour levels of $\psi_2(\vec{r})$, and $N_2$ is the number of $\psi_2(\vec{r})$ contour levels of interest. The location of mesh nodes $\{x_p\}$ is the intersection of the discrete set of $\psi_1(\vec{r})$ iso-contours with the discrete set of $\psi_2(\vec{r})$ iso-contours, namely,

$$\vec{x}_p = \{\vec{x} \mid \psi_1(\vec{x}) = c_j, \quad \psi_2(\vec{x}) = d_k\}, \quad j = 0, \ldots, N_1, \quad k = 0, \ldots, N_2.$$

Note that we have chosen the notation $\{\vec{x}_p\}$ rather than $\{\vec{x}_{jk}\}$ since

$$\{\vec{x}_{jk}\} = \{\vec{x} \mid \psi_1(\vec{x}) = c_j, \quad \psi_2(\vec{x}) = d_k\},$$

may not exist, i.e. $\vec{x}_{jk} = \emptyset$, if the iso-contour $\psi_1(\vec{r}) = c_j$ doesn't intersect with the iso-contour $\psi_2(\vec{r}) = d_k$. $\{\vec{x}_{jk}\}$ may also represent multiple points if the iso-contour of $\psi_1(\vec{r}) = c_j$ intersects with the iso-contour $\psi_2(\vec{r}) = d_k$ multiple times.

Graphically, the contour plots for each level set function are created, and then overlaid. For example, Figure 3.5 shows the contour plots of

$$\psi_1(x, y) = x, \quad \text{and} \quad \psi_2(x, y) = y,$$

as well as the overlaid contour plots. The iso-contours of $\psi_1(x, y) = x$ are vertical lines, and the iso-contours of $\psi_2(x, y) = y$ are horizontal lines. The intersection of the contour lines result in a set of uniformly spaced mesh nodes.

Figure 3.6 illustrates a more interesting mesh, obtained as part of a numerical experiment in Chapter 7. In this example, the intersection of iso-contours from $\psi_1(\vec{r})$ and $\psi_2(\vec{r})$ clusters more grid nodes in the vicinity of a sinusoidal curve in the $\mathbb{R}^2$. Specifically, the iso-contours of $\psi_2(\vec{r})$ are more densely packed around the sinusoidal curve of interest, while the iso-contours of $\psi_1(\vec{r})$ are perturbed slightly from straight vertical lines. Details of how $\psi_1(\vec{r})$ and $\psi_2(\vec{r})$ were computed are given in Chapter 7. Such a mesh might be useful for resolving membrane dynamics in $\mathbb{R}^2$, or wave motions in $\mathbb{R}^3$. The mesh shown is a snapshot of a numerical example in Section 7.3.
(a) Level contours of $\psi_1(x, y) = x$

(b) Level contours of $\psi_2(x, y) = y$

(c) The level contours of $\psi_1(x, y)$ and $\psi_2(x, y)$ overlaid to form the mesh.

Figure 3.5: In (a), the contour plot of $\psi_1(x, y) = x$ results in a set of vertical contour lines, while in (b) the contour plot of $\psi_2(x, y) = y$ results in a set of horizontal contour lines. The intersection of the level contours gives a set of mesh nodes, plotted in the overlaid contour plot (c) as dots.
Figure 3.6: In (a), the contour plot of $\psi_1(\eta)$ results in contour lines that essentially run up/down, while in (b) the contour plot of $\psi_2(\eta)$ results in a set of contour lines that run left/right. In (c), the intersection of the level contours gives a set of mesh nodes, shown in the overlaid contour plot, which clusters more grid nodes about the sinusoidal curve. Details of how $\psi_1(\eta)$ and $\psi_2(\eta)$ are obtained are presented in Chapter 7.
3.2.3 Intersection of Level Contours in $\mathbb{R}^3$

Given three level set functions, $\psi_1(\vec{\eta})$, $\psi_2(\vec{\eta})$, and $\psi_3(\vec{\eta})$, we define the location of mesh nodes as the intersection of a discrete set of $\psi_1(\vec{\eta})$, $\psi_2(\vec{\eta})$, and $\psi_3(\vec{\eta})$ iso-contours. Specifically, we initialize the iso-contours of $\psi_1(\vec{\eta})$, $\psi_2(\vec{\eta})$ and $\psi_3(\vec{\eta})$ respectively as

$$c_j = \frac{j}{N_1} \Delta \psi_1, \quad j = 0, \ldots, N_1,$$

$$d_k = \frac{k}{N_2} \Delta \psi_2, \quad k = 0, \ldots, N_2,$$

$$f_l = \frac{l}{N_3} \Delta \psi_3, \quad l = 0, \ldots, N_3,$$

where $\Delta \psi_*$ is the spacing between the iso-contour levels of $\psi_*(\vec{\eta})$, and $N_*$ is the number of $\psi_*(\vec{\eta})$ iso-contour levels of interest. The location of mesh nodes $\{x_p\}$ is specified by the intersection of the discrete set of $\psi_1(\vec{\eta})$, $\psi_2(\vec{\eta})$ and $\psi_3(\vec{\eta})$ iso-contours, namely,

$$\{\vec{x}_p\} = \{\vec{x} \mid \psi_1(\vec{x}) = c_j, \quad \psi_2(\vec{x}) = d_k, \quad \psi_3(\vec{x}) = f_l\},$$

$$j = 0, \ldots, N_1, \quad k = 0, \ldots, N_2, \quad l = 0, \ldots, N_3.$$

3.2.4 Level Set Function Initialization for $\mathbb{R}^2$

In this thesis, we explore three approaches for initializing a level set representation of an adaptive mesh in $\mathbb{R}^n$, $n > 1$. The first two approaches are extensions of the 1D initialization, with a few caveats. The third method requires more tools, and will be introduced in Section 4.2.

Recall that in $\mathbb{R}^1$, the level set initialization was

$$\psi(\eta) = \int_a^\eta \rho(x) \, dx.$$

A natural extension for $\mathbb{R}^2$ is to initialize the two level set functions as follows. Once again, we expand $\vec{\eta} = (x, y)$ for clarity. If the domain is a rectangular region,

$$a \leq x \leq b, \quad c \leq y \leq d,$$

a possible initialization is to decouple the $x$ and $y$ directions by integrating the monitor function along lines $x = \text{const}$ and $y = \text{const}$. This approach leads to the initialization of
two level set functions

\[
\psi_1(x, y) = \int_a^x \rho(\tilde{x}, y)\, d\tilde{x},
\]

\[
\psi_2(x, y) = \int_c^y \rho(x, \tilde{y})\, d\tilde{y}.
\]

Meshes generated by (3.4) are potentially skewed because of variations in the magnitudes of \(\psi_1(b, y)\) and \(\psi_2(x, d)\). The level contours of \(\psi_1(x, y)\) and \(\psi_2(x, y)\) may also intersect multiple times, resulting in a mesh that doesn’t equi-distribute \(\rho(x, y)\), [Huang & Sloan, 1994]. This is best illustrated through an example. Suppose we specify the monitor function

\[
\rho(x, y) = 1 + A \exp \left\{ -B \left[ \left( x - \frac{1}{2} \right)^2 + \left( y - \frac{1}{2} \right)^2 - \left( \frac{1}{4} \right) \right] \right\},
\]

\((x, y) \in [a, b] \times [c, d] = [0, 1] \times [0, 1],\)

where \(A > 0\) and \(B > 0\) are constants. Notice that when

\[
\left( x - \frac{1}{2} \right)^2 + \left( y - \frac{1}{2} \right)^2 - \left( \frac{1}{4} \right) = 0,
\]

the monitor function \(\rho(x, y)\) attains its supremum value, \(\rho = 1 + A\), and it decays exponentially to \(\rho = 1\) elsewhere. In other words, along a circle centered at \(\left( \frac{1}{2}, \frac{1}{2} \right)\) of radius \(\frac{1}{2}\), the monitor function attains its supremum \(\rho = 1 + A\), and decays to \(\rho = 1\) elsewhere. An ideal (adaptive) mesh should cluster a higher density of grid nodes around the circle defined by (3.6). Taking \(A = 5\), and \(B = 20\), we show in Figure 3.7 the inability of the initialization (3.4) to produce a good mesh to resolve \(\rho\), as well as the skewness that arises from the variation in \(\psi_1(b, y)\) and \(\psi_2(x, d)\).

An alternative initialization of the level set functions is a normalized version of (3.4).

\[
\psi_1(x, y) = \frac{\int_a^x \rho(\tilde{x}, y)\, d\tilde{x}}{\int_a^b \rho(\tilde{x}, y)\, d\tilde{x}},
\]

\[
\psi_2(x, y) = \frac{\int_c^y \rho(x, \tilde{y})\, d\tilde{y}}{\int_c^d \rho(x, \tilde{y})\, d\tilde{y}}.
\]

Notice that the level set functions are now scaled so that \(\psi_1(b, y) = 1\) is constant, and \(\psi_2(x, d) = 1\) is also constant. Scaling \(\psi_1(x, y)\) enforces that the \(y = c\) and \(y = d\) are iso-contours of \(\psi_1(x, y)\), while scaling \(\psi_2(x, y)\) enforces that \(x = a\) and \(x = b\) are iso-contours of \(\psi_2(x, y)\). Scaling provides some control of the iso-contours of \(\psi_1(x, y)\) and \(\psi_2(x, y)\) by
Figure 3.7: In (a), we show the contour plot of $\psi_1(x, y)$ using (3.4) where $\rho$ is defined in (3.5) with $A = 5$ and $B = 20$. In (b), we show the contour plot of $\psi_2(x, y)$, while in (c) the intersection of the level contours are shown. Note that the resulting mesh fails to equi-distribute $\rho(x, y)$ in the sense that we expect closely packed nodes on the circle defined by (3.6).
controlling the range of $\psi_1$ and $\psi_2$. We illustrate this second initialization approach by successfully generating an adaptive mesh using (3.5) with $A = 5$, and $B = 20$. Figure 3.8 shows that there is a higher density of grid nodes around the circle of radius 0.25, centered about (0.5,0.5).

Using (3.7) to initialize the level set functions in $\mathbb{R}^2$ is not robust. The first challenge is a generic consequence of using some form of equi-distribution in higher dimensions. If the given monitor function is rotated about some axis, e.g., we let $\theta \rightarrow \theta + \Delta \theta$ in some polar co-ordinate frame of reference, and $(x - x^*, y - y^*) = (r \cos \theta, r \sin \theta)$, then the resulting mesh obtained from the intersection of level contours could be completely different, since this initialization technique, (3.7), essentially decouples the $x$ and the $y$ directions. In the adaptive mesh community, this problem is often described as a “non-uniqueness” property of trying to satisfy the equi-distribution principle in higher dimensions [Huang, 2005].

The second challenge is unique to using level set functions to represent a higher dimensional mesh. The problem arises when level contours of $\psi_1(x, y)$ and level contours of $\psi_2(x, y)$ run almost parallel to each other, or when they intersect multiple times. This is illustrated best through an example.

In Figure 3.9, the level set function is initialized using the initialization equation (3.7), where $\rho(x, y)$ is specified as before using equation (3.5). This time, we use $A = 20$, and $B = 50$, which clusters an even higher density of grid points around a circle of radius 0.25, centered about (0.5,0.5). The level contours of $\psi_1(x, y)$ and $\psi_2(x, y)$ indeed cluster more points around the circle as expected, but the iso-contours of $\psi_1$ run almost parallel to the iso-contours of $\psi_2$, and the iso-contours intersect multiple times, resulting in a spurious clustering of grid points. Also, the grid is not axis-symmetric about (0.5,0.5) whereas the monitor function is.

We present a third method for initialization in Section 4.2. This third method circumvents most of concerns arising from grid effects and contours intersecting multiple times. There is a potential concern that this third initialization technique does not allow for a very high concentration of grid points in a localized area. Still, it is worth mentioning though that adaptivity in higher dimensions is always a compromise between getting a high concentration of grid points in a localized region, and creating elements which are not too skewed. We extend the initialization concept briefly to $\mathbb{R}^3$, and then discuss how level set functions are used to represent a moving mesh before elaborating on how to combine moving mesh and level set ideas in Chapter 4.
CHAPTER 3. LEVEL SET FRAMEWORK

Figure 3.8: These plots are obtained by initializing the level set function using (3.7) with a monitor function specified by (3.5), where the parameters are $A = 5$ and $B = 20$. In (a), the contour plot of $\psi_1(x, y)$ results in contour lines that essentially run left/right, while in (b) the contour plot of $\psi_2(x, y)$ results in a set of contour lines that run up/down. (c) The intersection of the level contours gives a set of mesh nodes, shown in the overlaid contour plot.
Figure 3.9: These plots are obtained by initializing the level set function using (3.7) with a monitor function specified by (3.5), where the parameters are $A = 20$ and $B = 50$. In (a), the contour plot of $\psi_1(x, y)$ results in contour lines that essentially run left/right, while in (b) the contour plot of $\psi_2(x, y)$ results in a set of contour lines that run up/down. (c) The intersection of the level contours overlaid in the plot. (d) the resulting set of mesh nodes is not an ideal mesh which equi-distributes $\rho(x, y)$ defined in (3.5). The presence of grid effects is very noticeable.
3.2.5 Level Set Function Initialization for $\mathbb{R}^3$

We briefly mention extensions of the previously mentioned initializations to $\mathbb{R}^3$ for completeness. In practice, the initialization discussed later in Section 4.2 is preferred.

If the domain is a cubic region,

$$a \leq x \leq b, \quad c \leq y \leq d, \quad f \leq z \leq g,$$

a possible initialization is to decouple the $x$, $y$ and $z$ directions by integrating the monitor function along lines $x = \text{const}$, $y = \text{const}$ and $z = \text{const}$. This approach leads to the initialization of three level set functions

\[
\psi_1(x, y, z) = \int_a^b \rho(\tilde{x}, y, z) \, d\tilde{x},
\]

\[
\psi_2(x, y, z) = \int_c^d \rho(x, \tilde{y}, z) \, d\tilde{y},
\]

\[
\psi_3(x, y, z) = \int_f^g \rho(x, y, \tilde{z}) \, d\tilde{z}.
\]

A scaled variant of the initialized level set functions can also be introduced to minimize skewness of a mesh due to large variations in $\psi_1(b, y, z)$, $\psi_2(x, d, z)$ and $\psi_3(x, y, g)$. The resulting level set functions are given by

\[
\psi_1(x, y, z) = \frac{1}{\int_a^b \rho(\tilde{x}, y, z) \, d\tilde{x}} \int_a^b \rho(\tilde{x}, y, z) \, d\tilde{x},
\]

\[
\psi_2(x, y, z) = \frac{1}{\int_c^d \rho(x, \tilde{y}, z) \, d\tilde{y}} \int_c^d \rho(x, \tilde{y}, z) \, d\tilde{y},
\]

\[
\psi_3(x, y, z) = \frac{1}{\int_f^g \rho(x, y, \tilde{z}) \, d\tilde{z}} \int_f^g \rho(x, y, \tilde{z}) \, d\tilde{z}.
\]

3.3 Level Set Evolution Equation

Thus far, we have discussed how to use level set functions to represent a static mesh. Now, we explore how to implicitly move the mesh by updating the level set function. Recall that by construction, mesh nodes are located on level contours of a level set function (or in the case of higher dimensions, the level contours of multiple level set functions). Suppose that we now wish to implicitly move a particular mesh node $\tilde{x}_p(t)$ according to some velocity field $(\tilde{x}_p)_t = \vec{v}_p(t)$. Ideally, we require that a mesh node initially located on the $k^{th}$ level
contour of \( \psi \) remains on the \( k^{th} \) level contour of \( \psi \) at a later time. Mathematically, the mesh node \( \bar{x}_p(t) \) located on the contour level \( c_k \) satisfies

\[
\psi (\bar{x}_p(t), t) = c_k.
\]

Taking the derivative of both sides with respect to \( t \), and expanding the derivatives using the chain rule, we obtain

\[
\psi_t (\bar{x}_p(t), t) + \nabla \psi (\bar{x}_p(t), t) \cdot \bar{v}_p(t) = 0,
\]

or equivalently,

\[
\psi_t (\bar{x}_p(t), t) + \bar{v}_p(t) \cdot \nabla \psi (\bar{x}_p(t), t) = 0,
\]

where \( \bar{v}_p(t) = \bar{x}_p'(t) \) is the velocity of the mesh node on the \( k^{th} \) iso-contour of \( \psi(x, t) \).

If we require that all mesh nodes located on their respective iso-contours of \( \psi(x) \) remain on their initial iso-contours at a later time, then

\[
\psi_t (\bar{x}_p(t), t) + \bar{v}_p(t) \cdot \nabla \psi (\bar{x}_p(t), t) = 0, \quad p = 0, \ldots, N,
\]

where \( N \) is the total number of mesh nodes. If we extend (by interpolating) the velocity of the mesh nodes \( \{ \bar{v}_p(t) \} \) to the computational grid to obtain a velocity field \( v(\eta, t) \), we obtain the level set equation,

\[
\psi_t (\eta, t) + \bar{v}(\eta, t) \cdot \nabla \psi (\eta(t), t) = 0. \tag{3.8}
\]

Since \( \nabla \psi \) gives the normal vector to \( \psi \), the level set equation is often written as

\[
\psi_t (\eta, t) + \bar{v}_n(\eta, t) |\nabla \psi (\eta(t), t)| = 0,
\]

where \( \bar{v}_n = \bar{v} \cdot \bar{n} \) is the velocity component normal to \( \psi \).

The level set equation falls into the general class of Hamilton–Jacobi equations, where there is a wealth of schemes (see [Zhang & Shu, 2003, Jiang & Peng, 2000]) to solve them numerically.

Moreover, in \( \mathbb{R}^2 \) or \( \mathbb{R}^3 \), the level set equation (3.8) holds for all the level set functions used to represent the mesh, i.e., given the velocity of the mesh nodes and extending the velocities to the computational grid, \( \bar{v} \), we would update all the level set functions representing the mesh. In \( \mathbb{R}^2 \), we implicitly move the mesh by solving

\[
\begin{cases}
\psi_{1t}(\eta, t) + \bar{v}(\eta, t) \cdot \nabla \psi_1(\eta, t) = 0, \\
\psi_{2t}(\eta, t) + \bar{v}(\eta, t) \cdot \nabla \psi_2(\eta, t) = 0.
\end{cases}
\]
In $\mathbb{R}^3$, we implicitly move the mesh by solving

$$
\begin{align*}
\psi_1(\vec{r}, t) + \vec{v}(\vec{r}, t) \cdot \nabla \psi_1(\vec{r}, t) &= 0, \\
\psi_2(\vec{r}, t) + \vec{v}(\vec{r}, t) \cdot \nabla \psi_2(\vec{r}, t) &= 0, \\
\psi_3(\vec{r}, t) + \vec{v}(\vec{r}, t) \cdot \nabla \psi_3(\vec{r}, t) &= 0.
\end{align*}
$$

### 3.4 Further Discussion

There are several other issues related to a level set mesh representation that require further discussion.

#### 3.4.1 Reinitialization

In much of the level set literature, e.g. [Chen et al., 1997], a reinitialization of the level set function is introduced periodically to keep the level set function close to a signed distance function. A signed distance function, $f$, satisfies the Eikonal equation [Sethian, 1999],

$$
|\nabla f| = 1.
$$

If $\psi$ is close to a signed distance function, one is able to obtain more accurate approximations to important quantities, such as the normal vector, $\hat{n}$, to the interface. In this level set representation of a mesh, all the iso-contours provide useful information, and a standard reinitialization procedure would unnecessarily move most iso-contours, giving an incorrect location of the desired mesh. It is possible that a well-designed re-initialization step could be implemented to "smooth" the mesh in a controlled fashion, but this reinitialization idea has not been formulated.

#### 3.4.2 Representing Arbitrary Structured Meshes

In this chapter, we described how to initialize level set functions to generate static grids $\{x_k\}$ which equi-distribute some monitor function, $\rho(\vec{x})$. Another useful scenario is to generate a level set function (or level set functions) to properly represent a prescribed set of grid nodes. In $\mathbb{R}^1$, this can be done trivially by assigning linearly increasing values of $\psi$ to each grid node, e.g. given a set of grid nodes $\{x_k\}, k = 0, \ldots, N$, set

$$
\psi(x_k) = \frac{k}{N}.
$$
CHAPTER 3. LEVEL SET FRAMEWORK

and interpolate these values of $\psi$ from the given set of grid nodes to the computational grid. The quality of this representation is dependent on the resolution of the computational grid.

In higher dimensions, representing arbitrary structured meshes is an open problem. In order to properly represent the set of grid nodes by level set functions, there must be some measure of connectivity. How one uses information about nearest neighbors to construct an implicit representation of the mesh is an open research question.

3.4.3 Connectivity

In addition to implicitly storing the location of the mesh nodes, a level set representation of a mesh also contains useful information about how the mesh nodes are connected. In $\mathbb{R}^1$, neighboring nodes are separated by two consecutive iso-contour levels of the level set function, $\psi$, if $\psi(\eta)$ is a monotonically increasing or decreasing function in $\eta$. Specifically, two nodes $x$ and $y$, are adjacent if

$$\psi(x) - \psi(y) = p\Delta\psi,$$

where $p \in \{-1, 1\}$ and $\Delta\psi$ is the spacing of interest between iso-contours of $\psi$. If $\psi(\eta)$ is a function that is non-monotonic in $\eta$, then it is possible for two adjacent nodes to be on the same iso-contour of $\psi$. For example, if $\psi(\eta) = |\eta| - 0.1$ and $c = [0, 1, 2, \ldots]$, then the two nodes resulting from the zero iso-contour, $\{-0.1, 0.1\}$, are adjacent, and $\psi(-0.1) - \psi(0.1) = 0$.

In $\mathbb{R}^2$, provided the level contours of $\psi_1(\eta)$ and $\psi_2(\eta)$ do not intersect multiple times, we can define two nodes, $\vec{x}$ and $\vec{y}$, as adjacent if the following two equations hold,

$$\psi_1(\vec{x}) - \psi_1(\vec{y}) = p\Delta\psi_1,$$
$$\psi_2(\vec{x}) - \psi_2(\vec{y}) = q\Delta\psi_2,$$

where $p \in \{-1, 1\}$, $q \in \{-1, 1\}$, and $\Delta\psi_k$ is the spacing between iso-contours of the respective level set functions. In $\mathbb{R}^3$, two nodes $\vec{x}$ and $\vec{y}$ are adjacent if all three equations hold,

$$\psi_1(\vec{x}) - \psi_1(\vec{y}) = p\Delta\psi_1,$$
$$\psi_2(\vec{x}) - \psi_2(\vec{y}) = q\Delta\psi_2,$$
$$\psi_3(\vec{x}) - \psi_3(\vec{y}) = r\Delta\psi_3,$$
where \( r \in \{-1, 1\} \) and \( \Delta \psi_3 \) is the spacing between iso-contours of \( \psi_3(\mathbf{x}) \).

Knowing connectivity information is particularly useful in higher dimensions, especially if we wish to create quadrilateral elements in \( \mathbb{R}^2 \), or the equivalent six-faced object in \( \mathbb{R}^3 \). It is unclear if the connectivity information allows the formation of structured triangulated or tetrahedral meshes.

### 3.4.4 Implicit versus Explicit Representation

In \( \mathbb{R}^1 \), the advantages of using an implicit (level set) representation of a mesh are (i) the circumvention of mesh crossing, and (ii) the availability of connectivity information. Considering an implicit representation of the mesh was a necessary step to discovering shortcomings in the current moving mesh technology. The level set representation of a mesh motivates the derivation of a new set of boundary conditions in Chapter 4, which allows for the natural addition and removal of grid nodes within a moving grid framework.

In higher dimensions, circumventing mesh crossing and easy access to connectivity information is certainly a major incentive to use a level set representation of mesh nodes; detecting and correcting for mesh crossing events is an expensive operation. We anticipate that using level set functions to represent the mesh nodes will also provide a systematic approach to adding and removing grid points, depending on which iso contours are present.

An implicit representation of the mesh, nevertheless, means that we never have the exact location of the mesh nodes. Specifically, since the mesh nodes are obtained by using interpolation to find iso-contours of level set functions, the location of the mesh is accurate only up to the resolution of the computational mesh. In [Babuška & Rheinboldt, 1979], authors argue that the location of a mesh is not needed to high accuracy.

There is one concern associated with using a level set representation of the mesh, the presence of “multiple grids”. Recall that the discrete set of iso-contours of a level set function (or level set functions) give the desired moving grid. The level set functions themselves, however, are defined on a computational (normally uniformly spaced) grid. Computations are performed on both grids, (i) on the moving grid to update the physical PDE and (ii) on the computational grid to update the level set function, implicitly moving the mesh. There is a necessary interpolation step in this hybrid method, to be discussed in Chapter 4.
3.4.5 Adaptive Contour Levels

In our discussions so far, mesh nodes have been located on iso-contours of $\psi$,

$$\psi(\vec{x}) = c_k,$$

where $\vec{c}$ is a vector normally given as

$$\vec{c} = \min \psi : \Delta \psi : \max \psi.$$

We are not restricted, however, to such a definition for $\vec{c}$. Depending on the actual problem, one may choose $\Delta \psi$ to be time dependent. In fact, $\vec{c}$ can be chosen adaptively, although we currently see no obvious advantages for doing so.
Chapter 4

New $h$-$r$ Hybrid Method

[Sethian, 1994] first discussed the use of level set functions for static grid generation. He proposed a front propagating idea, which is used to generate a set of "body-fitted" coordinates. [Liao et al., 2000] expand on this idea of using level set functions to represent equi-distributed meshes. They considered dynamic problems where a solution feature is changing, and the spatial mesh needs to evolve in order to continue equi-distributing a given monitor function. They use a deformation mapping argument to determine how far a mesh strays from an equi-distributed state, and then compute a set of mesh velocities to update the level set function, implicitly updating the mesh. Their resulting "$r$-refinement hybrid moving mesh - level set" method, has no apparent advantages over traditional $r$-refinement schemes.

In this chapter, we introduce a new $h$-$r$ hybrid method. The key feature of this new hybrid method is the creation or deletion of grid nodes in a smooth, natural fashion. This contrasts with a traditional $h$-refinement method which locally adds or removes points abruptly, or a traditional $r$-refinement method which is constrained to keep the number of grid points fixed, regardless of how a solution feature is changing.

In Section 4.1, we describe a hybrid moving mesh - level set framework which uses velocities from a moving mesh method to update a level set representation of a mesh. We utilize this mesh movement framework in Section 4.2, to initialize level set functions $\psi_1(\vec{y})$ and $\psi_2(\vec{y})$ in $\mathbb{R}^2$, such that the iso-contours give a mesh which equi-distributes a defined monitor function. In Section 4.3, we discuss how to incorporate the hybrid moving mesh - level set framework into the solution of a physical PDE of interest. In Section 4.4, we derive a new set of boundary conditions in $\mathbb{R}^1$, motivated by the level set representation of the
mesh, that allows for the natural creation or deletion of grid points. Finally, we discuss the implication of these new boundary conditions in Section 4.5.

4.1 Hybrid Moving Mesh - Level Set Method

In Section 3.3, we derived the level set evolution equation, which showed how to implicitly move the mesh by updating the level set function appropriately. In this section, we discuss a hybrid moving mesh method - level set method, introduced by Liao et al., which solves a moving mesh equation for mesh velocities, and uses those mesh velocities to update the level set function, implicitly moving the mesh. We discuss this hybrid method in \( \mathbb{R}^1 \) for a fixed domain, \( \Omega = [a, b] \), before generalizing to \( \mathbb{R}^n \). The idea is as follows.

Let \( \psi(\eta, t) \) denote the level set function, where \( \eta \) is the computational grid upon which the level set function is defined, and let \( x(t) \) denote the mesh trajectories. Given some monitor function \( \rho(x, t) \), we initialize a level set function \( \psi(\eta, 0) \), whose iso-contours give the location of the initial mesh, \( x(0) \). We then solve equations associated with a moving mesh method, such as an MMPDE (2.12) or the GCL (2.24), to find a set of mesh velocities, \( v(x, t) \). These velocities are interpolated to the computational grid, and the interpolated velocities are used to update the level set function (3.8). In \( \mathbb{R}^1 \), given \( \rho(x, t) \), we initialize \( \psi(\eta, 0) \) using

\[
\psi(\eta, 0) = \int_a^\eta \rho(x, 0) \, dx.
\]

If MMPDE4 is chosen as the moving mesh method to find mesh velocities, we then move the mesh by solving

\[
\psi_t + \hat{v}\psi_\eta = 0,
\]

where \( \hat{v} \) is an interpolated velocity field, derived from solving MMPDE4 for mesh velocities \( v \),

\[
\frac{d}{d\xi} \left( \rho \frac{dv}{d\xi} \right) = -\frac{1}{\tau} \left( \rho \frac{dx}{d\xi} \right),
\]

\[v(a) = v(b) = 0.\]

See Chapter 5 for implementation details.

Extensions to higher dimensions is straightforward. Given \( \rho(\vec{x}, t) \) in \( \mathbb{R}^2 \), we initialize two level set functions, \( \psi_1(\vec{\eta}, 0) \) and \( \psi_2(\vec{\eta}, 0) \), so that the intersection of a discrete set of
\psi_1 iso-contours with a discrete set of \psi_2 iso-contours results in an initial mesh, \bar{x}(0) which
equi-distributes \rho(\bar{x}, 0). If we choose to solve the GCL (2.24) to find mesh velocities, we
move the mesh by solving
\begin{align*}
\psi_{1t} + \tilde{v} \cdot \nabla \psi_1 &= 0, \\
\psi_{2t} + \tilde{v} \cdot \nabla \psi_2 &= 0,
\end{align*}
where \tilde{v} is the interpolated velocity field. This interpolated velocity field is obtained by first
solving
\begin{align*}
\nabla \cdot \left( \frac{\rho}{w} \nabla \phi \right) + \frac{\partial \rho}{\partial t} &= 0, \\
\nabla \phi|_{\partial D} &= 0,
\end{align*}
for \phi, then differentiating \phi to obtain mesh velocities,
\[ \tilde{v} = \frac{1}{w} \nabla \phi, \]
and then interpolating to the computational grid to obtain \tilde{v}. As described in Section 2.3, the choice of \( w \), a weight function, gives different moving mesh schemes.

4.2 Initialization of Level Set Functions in \( \mathbb{R}^2 \)

In Section 3.2.4, we presented two algorithms for initializing a level set representation of
a mesh in higher dimensions. Here, we provide an alternative method for initializing the
level set representation of a mesh. The idea is to evolve a spatially uniform mesh towards
an equi-distributed mesh. Specifically, if we desire an adaptive mesh which equi-distributes \( \rho(x, y) \), we first initialize our level set functions as
\[ \psi_1(x, y, T_\alpha) = x, \quad \psi_2(x, y, T_\alpha) = y, \]
where \( T_\alpha \) is some artificial time, and is chosen along with \( \Delta t \) and \( T_\gamma \) to give a gradually
changing \( \bar{\rho}(x, y, t) \) defined below in (4.1). Notice that the level contours of \( \psi_1 \) intersect with
the level contours of \( \psi_2 \) to give an initially uniform mesh. We then evolve both level set
functions
\begin{align*}
\psi_{1t} + v \nabla \psi_1 &= 0, \\
\psi_{2t} + v \nabla \psi_2 &= 0,
\end{align*}
from time $t = [T_\alpha : \Delta t : T_\gamma]$. The mesh velocities $v$ are computed by solving a moving mesh method of choice, with a dynamically changing monitor function

$$\bar{p}(x, y, t) = \beta + \frac{T_\alpha - t}{T_\alpha - T_\gamma}(\rho(x, y) - \beta),$$

where

$$\beta = \frac{\iint \rho(x, y) \, dx \, dy}{\iint dx \, dy},$$

is constant in time. Note that $\bar{p}(x, y, T_\alpha) \equiv \beta$, which is best represented by a uniform mesh, while $\bar{p}(x, y, T_\gamma) = \rho(x, y)$, the desired monitor function.

### 4.3 Coupling with a Physical Equation in $\mathbb{R}^1$

We illustrate how to incorporate the hybrid moving mesh framework into the numerical solution of a physical equation by returning to our previous example of Burgers' equation in Section 2.5.2. Now, instead of using an explicit representation of the mesh, $\{x_k\}$, we use a level set function $\psi(\eta, t)$ to represent the mesh. Instead of updating the mesh directly (i.e. $x_t = v$), we update the level set function. Starting from a defined monitor function $\rho(u(x, t))$, chosen to give some quantitative measure of $u(x(t), t)$, our proposed system of equations which incorporates the level set technology is

$$u_t + vu_x + uu_x = \epsilon u_{xx}, \quad (4.2a)$$

$$\psi_t + \tilde{v}\psi_x = 0, \quad (4.2b)$$

$$\frac{d}{d\xi} \left( \frac{du}{\rho_{d\xi}} \right) = -\frac{1}{\tau} \frac{d}{d\xi} \left( \rho \frac{dx}{d\xi} \right), \quad (4.2c)$$

$$v(a) = v(b) = 0.$$
4.4 New Boundary Conditions in $\mathbb{R}^1$

Recall from Section 3.1 that we initialize the level set function as an integral of a given monitor function. On a fixed 1D domain, we set

$$\psi(\eta, 0) = \int_a^\eta \rho(u(x, 0)) \, dx,$$

$$c_j = \frac{2}{N} \int_a^b \rho(u(x, 0)) \, dx.$$

Finding level contours of this level set function is then equivalent to finding the mesh nodes which equi-distribute the monitor function. A natural generalization is to require that the level set function represents the integral of the monitor function for all time,

$$\psi(\eta, t) = \int_a^\eta \rho(u(x, t)) \, dx. \quad (4.3)$$

If we update the level set function using computed velocities from solving an MMPDE with boundary conditions $v(a) = v(b) = 0$, then the level set function does not represent the integral of the monitor function for all time. In fact, solving

$$\psi_t + vv_x = 0,$$

$$v(a) = v(b) = 0,$$

implies

$$\psi_t(a, t) = 0,$$

$$\psi_t(b, t) = 0,$$

so that

$$\psi(a, t) = \psi(a, 0) = 0,$$

$$\psi(b, t) = \psi(b, 0), \quad \forall t,$$

instead of

$$\psi(b, t) = \int_a^b \rho(u(x, t)) \, dx.$$

In order for $\psi(b, t)$ to equal $\int_a^b \rho(u(x, t)) \, dx$ (4.3), we derive a new boundary condition. First, define

$$L(t) = \int_a^b \rho(u(x, t)) \, dt. \quad (4.4)$$
Next, we want \( \psi(b, t) = \int_a^b \rho(u(x, t)) \, dx \), so we set

\[
\psi_t(b, t) = L_t(t),
\]

so that

\[
\psi(b, t) = L(t), \quad \forall t.
\]

In terms of the evolution equation, (4.5) means that

\[
v(b, t) |\psi_x(b, t)| = -L_t(t),
\]

or equivalently,

\[
v(b, t) = \frac{-L_t(t)}{|\psi_x(b, t)|}.
\]

We have taken the absolute value of \( \psi_x \) so that \( v(b, t) \) is not tied to the normal direction of \( \psi \), i.e., \( v(b, t) < 0 \) corresponds to mesh movement to the left, and \( v(b, t) > 0 \) corresponds to mesh movement to the right.

This is a key result in the thesis! For a fixed 1D domain \( x \in [a, b] \), solving a moving mesh equation with a new Dirichlet condition at the right boundary \( v(b, t) = -L_t(t) \) (replacing \( v(b, t) = 0 \)), results in a set of mesh velocities that evolve the level set equation with the following properties.

1. The implicitly represented mesh \( \{x_p(t)\} \) equi-distributes \( \rho(u(x, t)) \); computed mesh velocities \( v(x, t) \) continue to move the mesh towards one that equi-distributes \( \rho(u(x, t)) \).

2. The level set function thus represents the integral of the monitor function for all time,

\[
\psi(\eta, t) = \int_a^\eta \rho(u(x, t)) \, dx.
\]

3. Grid nodes are naturally added or removed, depending on the number of sub-intervals specified by \( \psi(x, t) \). In particular, if we keep the spacing between iso-contours constant,

\[
\Delta \psi = \frac{1}{N(0)} \int_a^b \rho(x, 0) \, dx,
\]

where \( N(0) \) is the initial number of sub-intervals, then the number of sub-intervals is given by

\[
N(t) = \frac{\psi(b, t) - \psi(a, t)}{\Delta \psi},
\]

where \( N(t) \) is no longer constrained to be an integer.
4.5 Interpretation of New Boundary Conditions

So, what do these new boundary conditions really mean in terms of finding mesh nodes to equi-distribute a given monitor function? The main difference is, instead of trying to find a fixed number of mesh nodes to equi-distribute some normalized monitor function, we now try to find a set of mesh nodes that keeps the piecewise integral of the monitor function in each region fixed. If $L_t(t) > 0$ increases, then more mesh points are considered for equi-distribution. If $L_t(t) < 0$, less mesh points are needed for the equi-distribution. There is an interesting interpretation if the monitor function is some measure of the error. The integral of the monitor function, $L(t)$, gives the total error in the region. Adding or removing mesh points depending on $L_t(t)$ corresponds to a type of error control, as commonly done in the $h$-adaptivity community.

4.6 Further Discussion

There are interesting research directions that were not tackled since they were beyond the scope of the thesis. We briefly discuss some of these topics.

4.6.1 New Boundary Conditions for Moving Domains

Consider the problem of equi-distribution with a moving boundary, $x \in [a(t), b(t)]$. Ideally, we want

$$
\psi(\eta, t) = \frac{\rho(x, t)}{\eta} \, dx, \quad \eta \in [a(t), b(t)].
$$

There are two grids which need to be updated: the computational and the implicitly represented grid.

First, the computational grid must be updated so that the level set function (i.e., the implicit representation of the mesh) is defined in the entire domain. A “moving mesh” implementation of this would solve

$$
\eta_t = v_1(\eta, t),
$$

where

$$
v_1(\eta, t) = a_t(t) + \frac{(\eta - a(t))}{(b(t) - a(t))}b_t(t),
$$
CHAPTER 4. NEW H-R HYBRID METHOD

is a spatially linear function, defined so that

\[ v_1(a(t), t) = a(t), \]
\[ v_1(b(t), t) = b(t). \]

The level set function can be interpolated as the computational grid changes, or it can be smoothly updated by solving

\[ \psi_t + v_1(\eta, t) \psi_\eta = 0, \]

a Lagrangian style update introduced in Section 2.5.2.

Secondly, the implicitly represented grid, (i.e., the level set function) needs to be updated to handle the problem of equi-distribution. We follow the derivation in Section 4.4, solving a moving mesh equation with the new Dirichlet boundary conditions, i.e.

\[ \frac{d}{d\xi} \left( \rho \frac{dv_2}{d\xi} \right) = -\frac{1}{\tau} \left( \rho \frac{dx}{d\xi} \right), \]
\[ v_2(a(t), t) = 0, \]
\[ v_2(b(t), t) = \frac{-L_t(t)}{|\psi_\eta(b(t))|}. \]

After interpolation, this results in another set of mesh velocities, \( v_2(\eta, t) \) to evolve the level set equation.

Combining these ideas, we propose that a reasonable algorithm to construct a moving mesh for a problem with a moving boundary, \( x \in [a(t), b(t)] \), is as follows.

Let \( \psi(\eta(t), t) \) denote the level set function, where \( \eta(t) \) is the computational grid upon which the level set function is defined, and let \( x(t) \) denote the mesh trajectories. Given \( \rho(x, t) \), we initialize \( \psi(\eta(0), 0) \) using

\[ \psi(\eta, 0) = \int_a^b \rho(x, 0) \, dx. \]

If MMPDE4 is chosen as the moving mesh method to find mesh velocities, we then move both meshes by solving

\[ \psi_t + f(v_1, v_2) \psi_\eta = 0, \]
\[ \eta_t = v_1(\eta, t), \]
\[ \psi_t + f(v_1, v_2) \psi_\eta = 0, \]
\[ \eta_t = v_1(\eta, t), \]
where

\[ v_1(\eta, t) = a(t) + \frac{(\eta - a(t))}{(b(t) - a(t))} b(t), \]

and \( v_2 \) are obtained by solving MMPDE4 (2.12)

\[
\frac{d}{d\xi} \left( \rho \frac{dv}{d\xi} \right) = -\frac{1}{\tau} \left( \rho \frac{dx}{d\xi} \right),
\]

\[ v(a, t) = 0, \]
\[ v(b, t) = \frac{-L(t)}{|\psi_x(b, t)|}, \]

and

\[ f(v_1, v_2) = v_1 + v_2, \]

where \( \bar{v} \) is an interpolation of \( v \) to the computational mesh. This proposed mesh movement scheme has not been implemented.

### 4.6.2 New Boundary Conditions in \( \mathbb{R}^2 \)

To derive new boundary conditions in \( \mathbb{R}^2 \), it is useful to consider normalized monitor functions. Suppose we are working on a rectangular computational domain

\[ (x, y) \in \Omega = [a, b] \times [c, d]. \]

Letting

\[ g(y) = \int_a^b \rho(x, y, 0) \, dx, \]
\[ f(x) = \int_c^d \rho(x, y, 0) \, dy, \]

and defining normalized monitor functions

\[ \rho_1(x, y, t) = \frac{\rho(x, y, t)}{g(y)}, \]
\[ \rho_2(x, y, t) = \frac{\rho(x, y, t)}{f(x)}, \]

we initialize our level set functions as

\[ \psi_1(x, y, 0) = \int_a^x \rho_1(x, y, 0) \, dx, \]
\[ \psi_2(x, y, 0) = \int_c^y \rho_2(x, y, 0) \, dy. \]
Observe that using normalized monitor functions corresponds to normalizing the level set functions. We also define the integral of the monitor along coordinate lines as

\[ L_1(y, t) = \int_a^b \rho_1(x, y, t) \, dx, \]
\[ L_2(x, y) = \int_c^d \rho_2(x, y, t) \, dy. \]

Following the derivation of new boundary conditions in \( \mathbb{R}^1 \), we require

\[ \psi_1(x, y, t) = \int_a^x \rho_1(\tilde{x}, y, t) \, d\tilde{x}, \]
\[ \psi_2(x, y, t) = \int_c^y \rho_2(x, \tilde{y}, t) \, d\tilde{y}. \]

Recall the level set evolution equation,

\[ \psi_{1t} + \vec{v}_1 \cdot \nabla \psi_1 = 0, \]
\[ \psi_{2t} + \vec{v}_2 \cdot \nabla \psi_2 = 0, \]

where we allow each level set function to evolve with different velocity fields. First, we define boundary segments for \( \psi_1(x, y) \) and \( \psi_2(x, y) \) in Figure 4.1 and Figure 4.2 respectively.

![Figure 4.1: Boundary Segments for \( \psi_1(x, y) \)](image)

If we fix \( v_1 = 0 \) on \( \partial \Omega_{1a} \), i.e., \( \psi_1(a, y, t) = \psi_1(a, y, 0) \), then we require

\[ \psi_1(b, y, t) = \int_a^b \rho_1(x, y, t) \, dx = L_1(y, t). \]

Since

\[ \nabla \psi_1 = \psi_{1x} \hat{x} \text{ on } \partial \Omega_{1b}, \]
the level set evolution equation for $\psi_1$ on $\partial \Omega_{1b}$ gives
\[ \psi_{11}(b, y, t) + v_1(b, y, t)\psi_{1x} = 0, \]
\[ \implies v_1(b, y, t) = -\frac{L_{11}(y, t)}{\psi_{1x}(b, y, t)}. \]
Similarly, if we fix $v_2 = 0$ on $\partial \Omega_{2a}$, i.e., $\psi_2(x, c, t) = \psi_2(x, c, 0)$, then we require
\[ \psi_2(x, d, t) = \int_c^d \rho_1(x, y, t) dy = L_2(x, t). \]
Since
\[ \nabla \psi_2 = \psi_{2y} \mathbf{y} \text{ on } \partial \Omega_{2b}, \]
the level set evolution equation for $\psi_2$ on $\partial \Omega_{2b}$ gives
\[ \psi_{21}(x, d, t) + v_2(x, d, t)\psi_{2y} = 0, \]
\[ \implies v_2(x, d, t) = -\frac{L_{22}(x, t)}{\psi_{2y}(x, d, t)}. \]
Observe that in the current formulation, the boundary conditions for $v_1(x, y, t)$ and $v_2(x, y, t)$ are different. This necessitates solving the GCL equation (2.24) twice for $v_1$ and $v_2$ separately. In Chapter 7, we handle only the special case $v = 0$ on $\partial \Omega$, i.e., mesh nodes that start on the boundary remain on the boundary. This requires us to solve the GCL equation once for a common velocity field $v$ to update both level set functions.

4.6.3 New Boundary Conditions without the Level Set Framework

We return to the discussion in Section 4.5. On a fixed domain $x \in [a, b]$, $L_t(t) > 0$ resulted in $v(b, t) = \frac{-L_t(t)}{\psi_{1x}(b, t)} < 0$, corresponding to the node on the right boundary moving to the
left to allow for the creation of new grid nodes at the right boundary. More precisely, the implicitly represented mesh node at the right boundary is moved in such a way that the integral of $\rho$ between the left boundary and $x_N(t)$ is constant,

$$\int_a^{x_N(t)} \rho(x, t) \, dx = \int_a^{x_N(0)} \rho(x, t) \, dx.$$ We can incorporate this idea directly into the moving mesh methods in Section 2.2, dropping the level set framework. For example, we can solve MMPDE4 with the following boundary conditions,

$$\frac{\partial}{\partial \xi} \left( \rho \frac{\partial v}{\partial \xi} \right) = -\frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right),$$

$$v(a) = 0,$$

$$v(b) = -L(t),$$

adding nodes by extrapolation as necessary, or removing nodes that move outside of $[a, b]$.

In fact, this highlights that the level set framework provides a natural way to remove grid nodes, or add grid nodes through extrapolation.

It is unclear how to extend the new boundary conditions for $\mathbb{R}^2$ (derived about in Section 4.6.2) into a framework that doesn’t employ level set functions.
Chapter 5

Numerical Discretization

The general approach here utilizes a method of lines approach to solve a given physical PDE and the level set evolution equation. An essentially non-oscillatory (ENO) scheme [Shu, 1998] is used to discretize the level set equation spatially, while an appropriate spatial discretization (e.g. finite difference/volume scheme) is used to discretize the given physical PDE. The resulting ODE system is solved using a forward Euler time integrator. A cubic Hermite interpolation is used to find the explicit locations of mesh nodes from $\psi(x, t)$ in $\mathbb{R}^1$, while a contour zoning (bi-linear interpolation) is used to find the explicit location of the mesh nodes from $\psi_k(\vec{x}, t)$ in $\mathbb{R}^2$ as necessary. To solve MMPDEs or the GCL equation, a centered finite difference stencil is used. A Hermite cubic interpolation scheme is used to interpolate between the physical mesh (explicit location of the mesh) and the computational mesh (implicit representation of the mesh).

5.1 Moving Mesh Equations

We present the discretization schemes for solving MMPDE4 (2.12) and MMPDE6 (2.17) in $\mathbb{R}^1$. In $\mathbb{R}^2$, we show the discretization schemes for solving the GCL equation (2.27).

5.1.1 MMPDE4

Given the location of a mesh, $\{x_j\}, j = 0, \ldots, N$, as well as a monitor function $\rho(x)$ we solve MMPDE4 (2.12) to find a set of mesh velocities $\{v_j\}, j = 0, \ldots, N$, that move the mesh towards a new mesh that equi-distributes $\rho(x)$.
Recall MMPDE4,
\[
\frac{\partial}{\partial \xi} \left( \rho \frac{\partial v}{\partial \xi} \right) = -\frac{1}{\tau \frac{\partial}{\partial \xi}} \left( \rho \frac{\partial x}{\partial \xi} \right).
\]
If \( \xi \) is our computational grid with \( N \) subintervals, then we can take a centered difference approximation to
\[
\frac{\partial x}{\partial \xi} \approx \frac{x_{j+\frac{1}{2}} - x_{j-\frac{1}{2}}}{\frac{1}{N}}.
\]
Taking another centered difference,
\[
-\frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right) \approx -\frac{1}{\tau} \frac{\rho_{j+\frac{1}{2}} (x_{j+1} - x_j) - \rho_{j-\frac{1}{2}} (x_j - x_{j-1})}{\left( \frac{1}{N} \right)^2},
\]
where we denote \( \rho_j = \rho(x_j) \). This is a discrete approximation for the right hand side of MMPDE4. We can follow the same approximations for the left hand side of MMPDE4 to obtain
\[
\frac{\partial}{\partial \xi} \left( \rho \frac{\partial v}{\partial \xi} \right) \approx \frac{\rho_{j+\frac{1}{2}} (v_{j+1} - v_j) - \rho_{j-\frac{1}{2}} (v_j - v_{j-1})}{\left( \frac{1}{N} \right)^2}.
\]
For Dirichlet boundary conditions, i.e. \( v_0 \) and \( v_N \) are given, equating the finite difference approximations gives the discretized version of MMPDE4, i.e., the system of equations for \( \{v_j\}, j = 1, \ldots, N-1, \)
\[
\rho_{j+\frac{1}{2}} (v_{j+1} - v_j) - \rho_{j-\frac{1}{2}} (v_j - v_{j-1}) = -\frac{1}{\tau} \left[ \rho_{j+\frac{1}{2}} (x_{j+1} - x_j) - \rho_{j-\frac{1}{2}} (x_j - x_{j-1}) \right].
\]

### 5.1.2 MMPDE6

We also use a centered difference approximation to discretize MMPDE6,
\[
\frac{\partial^2 v}{\partial \xi^2} = -\frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right),
\]
resulting in a system of equations whose solution give \( \{v_j\}, \)
\[
(v_{j+1} - 2v_j + v_{j-1}) = -\frac{1}{\tau} \left[ \rho_{j+\frac{1}{2}} (x_{j+1} - x_j) - \rho_{j-\frac{1}{2}} (x_j - x_{j-1}) \right],
\]
for \( j = 1, \ldots, N-1 \), where \( v_0 \) and \( v_N \) are given Dirichlet boundary conditions.
5.1.3 GCL

We utilize the GCL equation to move the mesh in $\mathbb{R}^2$ due to its simplicity for computing mesh velocities. As mentioned in Chapter 2, the MMPDEs are preferred in $\mathbb{R}^1$ because they appear to be more stable than the GCL formulation; here, we mean that MMPDEs are stable in the sense that they move a mesh towards one that equi-distributes some monitor function, $\rho$. The GCL formulation, however, is only able to move a mesh towards one that preserves the equi-distribution of a monitor function, i.e., if a mesh has strayed away from equi-distributing $\rho$, the GCL formulation is unable to move the mesh to equi-distribute $\rho$. The MMPDEs framework is difficult to extend to $\mathbb{R}^2$, so we implement the GCL mesh movement strategy in $\mathbb{R}^2$. Recall the GCL equation,

$$\nabla \cdot (\rho \nabla \phi_t) + \rho_t = 0, \quad \text{in } \Omega = [0, 1] \times [0, 1],$$

$$\nabla \phi = 0, \quad \text{on } \partial \Omega.$$

Expanding the gradient operator,

$$\nabla \cdot \begin{pmatrix} \rho \phi_x \\ \rho \phi_y \end{pmatrix} + \rho_t = 0,$$

and the divergence operator, we obtain

$$(\rho \phi_x)_x + (\rho \phi_y)_y + \rho_t = 0.$$

Using a centered finite difference approximation, the discretized version of the GCL equation is

$$\frac{\rho^n_{j+\frac{1}{2},k} (\phi^n_{j+1,k} - \phi^n_{j,k}) - \rho^n_{j-\frac{1}{2},k} (\phi^n_{j,k} - \phi^n_{j-1,k})}{\Delta \xi^2} +$$

$$\frac{\rho^n_{j,k+\frac{1}{2}} (\phi^n_{j,k+1} - \phi^n_{j,k}) - \rho^n_{j,k-\frac{1}{2}} (\phi^n_{j,k} - \phi^n_{j,k-1})}{\Delta \eta^2} + (\rho_t)_j^n = 0,$$

for $j = 1, \ldots, N_1 - 1, \quad k = 1, \ldots, N_2 - 1$. 


with associated Neumann boundary conditions for $\phi$,

\[
\frac{\phi_{1,k}^n - \phi_{0,k}^n}{\Delta\xi} = 0, \quad k = 0, \ldots, N_2,
\]

\[
\frac{\phi_{N_1,k}^n - \phi_{N_1-1,k}^n}{\Delta\xi} = 0, \quad k = 0, \ldots, N_2,
\]

\[
\frac{\phi_{j,1}^n - \phi_{j,0}^n}{\Delta\eta} = 0, \quad j = 0, \ldots, N_1,
\]

\[
\frac{\phi_{j,N_2}^n - \phi_{j,N_2-1}^n}{\Delta\eta} = 0, \quad j = 0, \ldots, N_1.
\]

Our computational spatial grid, $[\xi, \eta] \in [0,1] \times [0,1]$, is an equally spaced grid with cell dimensions $(\Delta\xi, \Delta\eta)$. Here, $\rho_{jk}^n = \rho(j\Delta\xi, k\Delta\eta, t^n)$, and $(\rho_t)_{jk}^n = \rho_t(j\Delta\xi, k\Delta\eta, t^n)$.

There are various options for discretizing $\rho_t$. We use the monitor function from the previous time level to approximate $\rho_t$,

\[
\frac{\partial \rho}{\partial t} \approx \frac{\rho_{jk}^n - \rho_{jk}^{n-1}}{t^n - t^{n-1}}.
\]

### 5.2 Level Set Equation

Since the level set evolution equation is a special class of Hamilton-Jacobi equations, we discretize the level set equation spatially using ENO (Essentially Non-Oscillatory) schemes.

Recall the level set equation (3.8)

\[
\psi_t + \vec{V} \cdot \nabla \psi = 0.
\]

In one dimension,

\[
(\psi_t)_j + v_j(\psi_x)_j = 0.
\]

If $v_j > 0$, the values of $\psi$ are moving from left to right, and the method of characteristics tells us to look to the left of $x_j$ to determine $\psi_x$. Similarly, if $v_j < 0$, the values of $\psi$ are moving from right to left, and the method of characteristics tells us to look to the right of $x_j$ to determine $\psi_x$. This tells us that an accurate estimation of $\psi_x$ is an upwinding (a first order ENO scheme) scheme, viz.,

\[
(\psi_x^+) = \frac{\psi_{j+1}^n - \psi_j^n}{x_{j+1} - x_j}, \quad (\psi_x^-) = \frac{\psi_j^n - \psi_{j-1}^n}{x_j - x_{j-1}}.
\]
\[ \psi_x = \begin{cases} (\psi_x^+)_j & \text{if } v_j < 0, \\ (\psi_x^-)_j & \text{otherwise}. \end{cases} \]

This first-order accurate upwind scheme can be improved upon by using a more accurate approximation for \( \psi_x^+ \) and \( \psi_x^- \). [Jiang & Peng, 2000, Osher & Fedkiw, 2003] give a more thorough explanation of higher order ENO schemes.

## 5.3 Physical PDEs

We solve two 1D PDEs in Chapter 6. In this section, we discuss the spatial discretizations used to approximate the spatial derivatives.

### 5.3.1 Advection-Diffusion Equation

The advection-diffusion equation is given by

\[ u_t = f(t, u), \]

\[ u_t = -a(x) u_x + \epsilon u_{xx}. \]

To handle the advection term, one has to satisfy the necessary condition for stability: the numerical domain of dependence must contain the true domain of dependence of the PDE. A simple stability analysis shows that we need up-winded differences to handle the advection term. A centered difference scheme is used to handle the diffusion term. The advection-diffusion equation is thus discretized using

\[
(u_t)_j = -a_j [u_x^+]_j + \epsilon u_{j-1} \left( \frac{1}{x_{j-1} - x_j} \right) \left( \frac{1}{x_{j-1} - x_{j+1}} \right) + \epsilon u_j \left( \frac{1}{x_j - x_{j-1}} \right) \left( \frac{1}{x_j - x_{j+1}} \right) + \epsilon u_{j+1} \left( \frac{1}{x_{j+1} - x_j} \right) \left( \frac{1}{x_{j+1} - x_{j-1}} \right),
\]

where

\[
(u_x^+)_j = \frac{u_{j+1} - u_j}{x_{j+1} - x_j}, \quad (u_x^-)_j = \frac{u_j - u_{j-1}}{x_j - x_{j-1}}.
\]

is chosen based on \( sgn(a) \), i.e.,

\[
[u_x^+]_j = \begin{cases} (u_x^+)_j & \text{if } a_j < 0, \\ (u_x^-)_j & \text{if } a_j > 0. \end{cases}
\]
5.3.2 Burgers’ Equation

To ensure that shocks and other steep gradients are captured by our numerical scheme (i.e., these shocks move at the right speed), we solve Burgers’ equation in a discrete conservative form; the rate of change of conserved quantities is equal to a difference of fluxes. In particular,

\[ u_t + (f(u))_x = \epsilon u_{xx}, \quad f(u) = \frac{1}{2} u^2, \]

is discretized using

\[
[u_j]_t + \frac{F(x_{j+1/2}) - F(x_{j-1/2})}{x_{j+1/2} - x_{j-1/2}} = \epsilon u_{j-1} \left( \frac{1}{x_{j-1} - x_j} \right) \left( \frac{1}{x_{j-1} - x_{j+1}} \right) \\
+ \epsilon u_j \left( \frac{1}{x_j - x_{j-1}} \right) \left( \frac{1}{x_j - x_{j+1}} \right) \\
+ \epsilon u_{j+1} \left( \frac{1}{x_{j+1} - x_j} \right) \left( \frac{1}{x_{j+1} - x_{j-1}} \right),
\]

where \( F(x) \) is the numerical flux function, which satisfies

\[ f(u_j)_x = \frac{F_{j+1/2} - F_{j-1/2}}{\Delta x}. \]

To compute this numerical flux function, we define \( h(x) \) implicitly through the equation

\[ f(u(x)) = \frac{1}{\Delta x} \int_{x-\Delta x/2}^{x+\Delta x/2} h(y) \, dy. \]

Taking the derivative on both sides of the equation yields

\[ f(u(x))_x = \frac{h(x + \Delta x/2) - h(x - \Delta x/2)}{\Delta x}, \]

which shows that the numerical flux function at the cell wall is identical to \( h \), i.e.,

\[ F_{j\pm1/2} = h(x_{j\pm1/2}), \forall j. \]

We find \( h \) by computing its primitive using polynomial interpolation,

\[ H(x) = \int_{x-1/2}^{x} h(y) \, dy, \]

and then taking a derivative to get \( h \).
CHAPTER 5. NUMERICAL DISCRETIZATION

5.4 Temporal Discretization

5.4.1 Forward Euler

For all the simulations presented in Chapter 6 and Chapter 7, a simple forward Euler time stepping scheme is employed. The equation

\[ u_t = f(t, u) \]

is discretized as

\[ \frac{u^{n+1} - u^n}{\Delta t} = f(t^n, u^n). \]

While a forward Euler discretization is limited in the sense that the stability region does not include the imaginary axis, it is used in our simulations because it is a one step integrator, in contrast to multi-step or multi-stage schemes.

5.4.2 Multi-Step and Multi-Stage Schemes

A subtlety of using the method of lines approach on a moving grid was revealed when we tried to implement the h-r hybrid method with a multi-stage Runge-Kutta scheme to solve the physical PDE and the level set equation. What do we do when the number of mesh points changes within the intermediate stages or steps in a multi-stage or multi-step scheme? Two reasonable approaches are (i) implement interpolation between stages or steps, or (ii) freeze the mesh between stages or steps. Neither of these two approaches were implemented in this thesis.

5.5 Coupled Hybrid Algorithm in \( \mathbb{R}^1 \)

We give a more detailed discussion of how to numerically incorporate the moving mesh framework into the solution of a physical PDE, described in Section 4.3. In this algorithm, we use a simple forward Euler time-stepping scheme. The physical PDE is solved in the physical space, that is, the physical PDE is solved on the moving grid implicitly represented by the evolving level set function. For simplicity, we use the inter-weaving approach, mentioned in Section 2.5.1, choosing to interpolate the solution to the new mesh at each time step, rather than adding the Lagrangian term. Mesh velocities are computed on the physical grid using a moving mesh method of choice, and then interpolated to the computational mesh to
update the level set function, implicitly moving the mesh. Ideas to improve the algorithm and remove some of the interpolation requirements are discussed at the end of the section.

To solve a given PDE,

\[ u_t = f(t, u), \quad x \in [a, b], \quad t \in [T_0, T_f], \]

\[ u(x, T_0) = g(x), \]

in a moving framework, the following algorithm is used.

1. We decide on some preferred way to quantify how the solution \( u \) is changing, both spatially and temporally, by defining a monitor function, \( \rho(u(x, t)) \).

2. Fix an underlying computational mesh,

\[ \eta_r = a + r \Delta \eta, \quad r = 0, \ldots, M, \]

where

\[ \Delta \eta = \frac{(b - a)}{M}, \]

and \( M \) is the number of equally-spaced subintervals in the computational domain.

3. Set \( n = 0, \ t^0 = T_0. \)

4. Initialize the level set function, \( \psi_0 \), approximating the integral using the trapezoidal rule. Specifically, set \( \psi_0^0 = 0 \), and

\[ \psi_r^0 = \int_a^{\eta_r} \rho(u(x, t^0)) \, dx, \]

\[ \approx \sum_{s=1}^{r} \left( \rho(u(\eta_s, t^0)) + \rho(u(\eta_{s-1}, t^0)) \right) \frac{\Delta \eta}{2}, \quad r = 1, \ldots, M. \]

We also initialize the spacing between iso-contours of interest,

\[ \Delta \psi = \frac{\psi_M^0}{N(t^0)}, \]

where \( N(t^0) \), an integer, is the initial number of grid nodes chosen in the physical grid.

Finally, we define the discrete contour levels of interest \( (3.2) \),

\[ c_j^0 = j \Delta \psi, \quad j = 0, \ldots, N(t^0). \]
5. Find $\{x_j^0\}$, (3.3) such that
\[
\{x_j^0\} = \left\{ x \mid \tilde{\psi}(x) = c_j^0 \right\}, \quad j = 0, \ldots, N(t^0),
\]
where $\tilde{\psi}(x)$ is a piecewise cubic Hermite polynomial with $\tilde{\psi}(\eta_r) = \psi_r^0$, $r = 0, \ldots, M$.

6. Initialize $u_j^0 = g(x_j^0)$, $j = 0, \ldots, N(t^0)$.

For each $n = 0, 1, \ldots$, we compute the following.

7. Calculate $\rho_j^n = \rho(u_j^n)$, $j = 0, \ldots, N(t^n)$.

8. Set $t^{n+1} = t^n + \Delta t$, where $\Delta t$ is the forward Euler time-step, and find $\tilde{u}_j^{n+1} \approx u(x_j^n, t^{n+1})$ by solving
\[
\tilde{u}_j^{n+1} = u_j^n + \Delta t f(t^n, u^n), \quad j = 0, \ldots, N(t^n),
\]
where $f$ is some appropriate spatial discretization of the physical PDE.

9. Compute $\rho_j^{n+1} = \rho(\tilde{u}_j^{n+1})$, $j = 0, \ldots, N(t^n)$.

10. Compute an approximation to $L_t(t^n)$, (4.4),
\[
L_t(t^n) = \frac{\partial}{\partial t} \int_a^b \rho(x, t) \, dx \bigg|_{t=t^n},
\]
\[
\approx \frac{L(t^{n+1}) - L(t^n)}{\Delta t},
\]
\[
\approx \left( \sum_{s=1}^{N(t^n)} \left( \frac{\rho_{s+1}^n + \rho_s^n}{2} \right) (x_s^n - x_{s-1}^n) \right) - \left( \sum_{s=1}^{N(t^n)} \left( \frac{\rho_{s+1}^n + \rho_s^n}{2} \right) (x_s^n - x_{s-1}^n) \right),
\]
\[
\equiv \beta.
\]

11. Compute mesh velocities, $v_j^n$, $j = 0, \ldots, N(t^n)$, by either solving MMPDE4 or MMPDE6. The discretized version of MMPDE4 is the system of equations for $\{v_j^n\}$ given as
\[
\begin{align*}
v_0^n &= 0, \\
v_N(t^n) &= \frac{\beta}{v_M - v_{M-1}} (\text{or } v_{N(t^n)} = 0), \\
\end{align*}
\]
for $j = 1, \ldots, N(t^n) - 1,$
\[
\left( \frac{\rho_{j+1}^n + \rho_j^n}{2} \right) (v_{j+1}^n - v_j^n) - \left( \frac{\rho_{j+1}^{n+1} + \rho_j^{n+1}}{2} \right) (v_j^n - v_{j-1}^n) \\
&= -\frac{1}{\tau} \left( \frac{\rho_{j+1}^{n+1} + \rho_j^{n+1}}{2} \right) (x_{j+1}^n - x_j^n) - \frac{1}{\tau} \left( \frac{\rho_{j+1}^n + \rho_j^n}{2} \right) (x_j^n - x_{j-1}^n).
\]
The discretized version of MMPDE6 is the system of equations for \( \{v^n_k\} \) given as

\[
\begin{align*}
  v^n_0 &= 0, \\
  v^n_{N(t^n)} &= \frac{-1}{v^n_{M} - v^n_{M-1}} \quad \text{(or } v^n_{N(t^n)} = 0), \\
  v^n_j &= \frac{1}{2} \left( \frac{v^{n+1}_{j+1} + v^{n+1}_{j-1}}{2} \right) (x^n_{j+1} - x^n_{j-1}) - \frac{1}{2} \left( \frac{v^n_{j+1} + v^n_{j-1}}{2} \right) (x^n_j - x^n_{j-1}),
\end{align*}
\]

for \( j = 1, \ldots, N(t^n) - 1 \).

12. Interpolate \( \psi^n_j \) to the computational grid, that is, set \( \tilde{\psi}(x, t^n) \) to be the piecewise cubic Hermite polynomials with 

\[
\tilde{\psi}(x_j, t^n) = v^n_j, \quad j = 0, \ldots, N(t^n).
\]

Then set

\[
\psi^n_r = \tilde{\psi}(\eta_r, t^n), \quad r = 0, \ldots, M.
\]

13. Update the level set function (3.8),

\[
\psi^{n+1}_r = \psi^n_r - \Delta t \frac{\partial \psi^n_r}{\partial x}, \quad r = 1, \ldots, M - 1,
\]

where \( (\psi^n_r)_x \) is discretized as the up-wind difference approximations to \( \psi_x \), chosen based on \( \text{sgn}(\psi^n_r) \), i.e.,

\[
(\psi^n_r)_x = \begin{cases} 
(\psi^n_r)^+ & \text{if } \psi^n_r < 0, \\
(\psi^n_r)^- & \text{if } \psi^n_r > 0,
\end{cases}
\]

and

\[
(\psi^n_r)^+ = \frac{\psi^n_{r+1} - \psi^n_r}{\Delta \eta}, \quad (\psi^n_r)^- = \frac{\psi^n_r - \psi^n_{r-1}}{\Delta \eta}.
\]

The corresponding boundary conditions are

\[
\psi^{n+1}_0 = \psi^n_0, \quad \psi^{n+1}_M = \psi^n_M - \Delta t \frac{\psi^n_M - \psi^n_{M-1}}{\Delta \eta}.
\]

14. Redefine the iso-contour levels of interest, by first computing

\[
N(t^{n+1}) = \text{floor} \left( \psi^{n+1}_M - \psi^{n+1}_0 \right),
\]
where \( \text{floor}(x) \) is a function that truncates a real number \( x \) to the largest integer \( n \) such that \( n \leq x \). Then, we set

\[
c_j^{n+1} = j\Delta \psi, \quad j = 0, \ldots, N(t^{n+1}).
\]

15. Find the new grid points, \( \{x_j^{n+1}\} \). Define \( \tilde{\psi}(x, t^{n+1}) \) to be a piecewise cubic Hermite polynomials with \( \tilde{\psi}(\eta_r, t^{n+1}) = \psi_r^{n+1}, \ r = 0, \ldots, M \). Then, find the level contours of \( \tilde{\psi}(x, t^{n+1}) \),

\[
\{x_j^{n+1}\} = \left\{ x \mid \tilde{\psi}(x) = c_j^{n+1} \right\}, \quad j = 0, \ldots, N(t^{n+1}),
\]

16. Update the solution \( \tilde{u}_j^{n+1} \) to the new mesh, \( u_j^{n+1} \). Specifically, set \( \tilde{u}(x, t^{n+1}) \) to be a piecewise cubic Hermite polynomial with \( \tilde{u}(x_j^n, t^{n+1}) = \tilde{u}_j^{n+1}, \ j = 0, \ldots, N(t^{n+1}) \). Then,

\[
u_j^{n+1} = \tilde{u}(x_j^{n+1}, t^{n+1}).
\]

17. If \( t^{n+1} > T_f \), STOP, else, set \( n = n + 1 \) and loop back to Step 7.

As mentioned earlier, this algorithm is not optimized in the sense that there are calculations on both the physical and computational grid, resulting in interpolation requirements. Further investigations are necessary to determine if we can perform the calculations on either the computational mesh or the physical mesh (but not both!).

5.6 Software Development

The above algorithm was implemented in \texttt{MATLAB}. In steps 13, 15 and 16 in the above algorithm, the \texttt{interp1} algorithm was used to investigate the effects of using various interpolation schemes. In particular, linear interpolation schemes, piecewise cubic spline interpolation and piecewise cubic Hermite polynomials were tested. To generate the plot of mesh trajectories, the built in \texttt{contour} algorithm was used to plot level contours of \( \psi(\eta, t) \).
Chapter 6

Numerical Examples in $\mathbb{R}^1$

This chapter illustrates how the algorithms developed in Chapter 4 lead to the evolution of an implicitly represented, equi-distributed mesh. In the first four examples, we avoid solving physical PDEs, and show instead the evolution of the implicitly represented mesh for known functions $u(x,t)$ and selected monitor functions $\rho(u(x,t))$. The new boundary conditions discussed in Section 4.4 are implemented and contrasted with the old boundary conditions in Section 2.2.2. Then, we introduce physical PDEs and show the solution of the PDE together with the evolution of the level set function $\psi(\eta, t)$, and the evolution of the mesh nodes.

6.1 Example 1: an Equi-spaced Mesh where $L(t)$ is Increasing

This first example demonstrates that a level set function can represent a generalized integral of a specified monitor function,

$$\psi(\eta, t) = \int_a^b \rho(u(x,t)) \, dx, \quad \eta \in [a, b],$$

provided (i) a moving mesh method which generates mesh velocities to equi-distribute the given monitor function is employed, and (ii) the correct boundary conditions are used for the solution of mesh velocities. We show for an explicitly specified monitor function, that using velocities computed from MMPDE4 (with appropriate boundary conditions) results in the level set function evolving such that (i) the iso-contours generate a moving mesh that properly equi-distributes the specified $\rho(u)$, and (ii) the level set function represents the
CHAPTER 6. NUMERICAL EXAMPLES IN $\mathbb{R}^1$

generalized integral of the specified $\rho(u)$. By representing the generalized integral of $\rho(u)$, we are able to naturally add nodes as $L(t) = \int_a^b \rho(u) \, dx$ increases.

Let

$$u(x, t) = t \left(1 - \left|\frac{x}{a}\right|\right), \quad -a \leq x \leq a, \quad t \geq 0, \quad (6.1)$$

and set the monitor function, $\rho(u)$ to be the arc-length monitor function, specified by

$$\rho(u(x, t)) = \sqrt{1 + \frac{u_x^2}{a^2}},$$

$$= \sqrt{1 + \left(\frac{t}{a}\right)^2}, \quad x \in [-a, a] \setminus \{0\}.$$

The function $u(x, t)$ is a piecewise linear function of $x$, symmetric about $x = 0$. The arc-length monitor function, $\rho(u)$, depends on time, but doesn't vary spatially. The ideal mesh to represent $u(x, t)$ is always an equi-spaced mesh, since $\rho$ doesn't vary spatially. Recall that an equi-distributed mesh satisfies $\frac{\partial \rho}{\partial \xi} = c$, where $c$ is a constant. Since $\rho$ doesn't vary spatially, $\rho_{\xi}^{\partial \xi} = \text{const}$, and the right hand side of MMPDE4 (2.12) is identically zero,

$$\frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right) = 0,$$

Thus, MMPDE4 simplifies to

$$\frac{d^2 v}{d \xi^2} = 0.$$

If we use zero Dirichlet boundary conditions for the mesh velocities (see Section 2.2.2), i.e., $v(-a) = v(a) = 0$, then $v \equiv 0$. This is reasonable since:

- An initial grid which equi-distributes $\rho = \text{const}$ is an equally spaced grid.

- If the number of mesh nodes is kept constant, then regardless of how $L(t) = \int_a^b \rho \, dx$ is changing, an equally spaced grid equi-distributes $\rho$, since $\rho$ doesn't vary spatially.

- Since the mesh is equally spaced for all time, and the number of mesh nodes is held fixed for all time, we really have a static mesh, which is consistent with the discussion above.

Alternatively, we can impose the following “new” Dirichlet boundary conditions discussed in Section 4.4. An analytic form for $L(t)$ is easily derived,

$$L(t) = 2\sqrt{a^2 + t^2}, \quad \Rightarrow \quad L(t) = \frac{2t}{\sqrt{a^2 + t^2}} > 0,$$
and the new boundary conditions suggested are

\[ v(-a,t) = 0, \quad v(a,t)\psi_x(a, t) = -L(t), \]

or equivalently,

\[ v(-a,t) = 0, \quad v(a,t) = -\frac{L(t)}{\psi_x(a, t)}. \]

Solving

\[ \frac{d^2v}{d\xi^2} = 0, \]
\[ v(-a) = 0, \]
\[ v(a) = -\left( \frac{2t}{\sqrt{a^2 + t^2}} \right) \frac{1}{\psi_x(a, t)}, \]

for the mesh velocities, \( v \) gives

\[ v(\xi) = -\left( \frac{1}{\psi_x(a, t)} \right) \left( \frac{t}{a\sqrt{a^2 + t^2}} \right) (\xi + a). \quad (6.2) \]

In the right column of Figure 6.1, we initialize the level set representation of the mesh as a linear function \( \psi(x,0) = x + 1 \). We plot the evolution of the level set function, using the velocity field, (6.2), updating the level set evolution equation, (3.8).

As the simulation progresses (top to bottom), the level set function remains linear as expected, but the slope of \( \psi \) and the value of \( \psi(b, t) \) both increases. Since the mesh is given by the contour levels

\[ \tilde{\xi} = \min \psi : \Delta \psi : \max \psi, \]

of \( \psi \), where \( \Delta \psi \) is kept fixed, the number of mesh points increases as \( (\max \psi - \min \psi) \) increases. The left column shows a plot of the \( u(x, t) \) from (6.1) with the adaptive mesh given by the contour levels of the level set function. The key feature to observe is that the arc-length between any two points is constant for all time. We also plot the mesh trajectories in Figure 6.2. Notice in this figure that:

- Mesh nodes are being created from the right boundary.
- Grid nodes are always moving towards the left domain boundary.
- An equally spaced mesh is created for each time level (a horizontal cross section of the mesh trajectory plot will show that the distance between grid nodes are approximately equal).
Figure 6.1: Example 1: We evolve the level set function using the exact mesh velocities. The mesh remains equi-spaced throughout the computation, while grid points are smoothly added as the total integral of the monitor function increases.
Figure 6.2: Example 1: The mesh trajectory from the simulation. Notice that grid points are smoothly added from the right boundary as the total integral of the monitor function increases. A horizontal cross-section will show that an equi-spaced mesh is generated at each time level, consistent with the monitor function $\rho(u)$ which is not varying spatially.
6.2 Example 2: a Non-uniform Mesh where $L(t)$ is Increasing

We now evolve a mesh to equi-distribute the arc-length of a more interesting $u(x, t)$, which is quadratic in $x$ for each $t$. The monitor function thus varies spatially, and $L(t)$, the integral of the monitor function, is increasing.

Letting

$$u(x, t) = t(1 - x^2), \quad -1 \leq x \leq 1, \quad t > 0,$$

the arc-length monitor function gives

$$\rho(u(x, t)) = \sqrt{1 + u_x^2} = \sqrt{1 + 4t^2x^2}.$$ 

Since $\rho(u(x, 0))$ doesn't vary spatially, the mesh is initially equi-spaced; we set $\psi(x, 0) = x + 1$ to represent the initially equi-spaced mesh. As $t$ increases, $u(x, t)$ becomes an increasingly steeper inverted parabola. Since $L(t) = \int_a^b \rho(u(x, t)) \, dx$ is an increasing function (for increasing $t$), we desire the addition of grid points to resolve $u(x, t)$ as we move the mesh forward in time. We anticipate a higher clustering of grid points near the domain boundary, the steepest part of $u(x, t)$ for $t > 0$. First, we use the functional form of $\rho(u(x, t))$, and solve MMPDE4 (2.12) with zero velocity Dirichlet boundary conditions,

$$\frac{\partial}{\partial \xi} \left( \rho \frac{\partial \nu}{\partial \xi} \right) = -\frac{1}{r} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right),$$

$$v(-1) = v(1) = 0.$$

Figure 6.3 shows the evolution of $u$ and $\psi$ with these boundary conditions. As expected, $\psi(\eta, t) \neq \int_a^\eta \rho(u(x, t)) \, dx$. Specifically, $\psi(1, t) = \psi(1, 0)$ instead of $\psi(1, t) = \int_{-1}^1 \rho(u(x, t)) \, dx$. Consequently, the number of grid points,

$$N(t) = \frac{\psi(b, t) - \psi(a, t)}{\Delta \psi},$$

$$= \frac{\psi(b, 0) - \psi(a, 0)}{\Delta \psi},$$

is held constant. The left column of Figure 6.3 displays the evolution of the $u$ while the right column displays the evolution of the level set function. MMPDE4 (2.12) is solved for mesh velocities (given $\rho(u(x, t)))$, and the level set function is updated using the level set evolution equation (3.8) and computed velocities. Note that the plots in the left column shows the resulting adaptive mesh and the exact solution, $(x_k, u(x_k))$. The arc-length measure between
Figure 6.3: Example 2: boundary condition \( v(-1) = v(1) = 0 \) fixes the number of mesh points in this problem. The left graphs show the solution along with the equi-distributed mesh, the graphs on the right show the associated level set function. Notice that \( \psi(\eta, t) \neq \int_{-1}^{1} \rho(u(x, t)) \, dx \). At each time level, the mesh appears to distribute an equal amount of arc-length in each subinterval.
Figure 6.4: Example 2: mesh trajectories using the traditional boundary conditions. The number of mesh nodes are held constant.
two successive grid nodes increases as \( t \) increases. Additionally, we plot the mesh trajectories in Figure 6.4, which confirms that the number of mesh nodes remains constant.

We also solve MMPDE4 with the “new” boundary conditions:

\[
\frac{\partial}{\partial \xi} \left( \rho \frac{\partial v}{\partial \xi} \right) = -\frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right),
\]

\( v(0) = 0, \quad v(1) = \frac{L_t(t)}{|\psi_x(1, t)|}, \)

which allows for the creation of grid points. In Figure 6.5, the right column shows the evolution of the level set function when MMPDE4 (2.12) is solved with the right boundary velocity, \( v(1) \propto L_t(t) \). Mesh points are added naturally while keeping the mesh equi-distributed. The arc-length between two successive points is constant for all time. Analytic values

\[
L(t) = \int_{-1}^{1} \sqrt{1 + 4t^2x^2} \, dx,
\]

\[
= \sqrt{1 + 4t^2} + \frac{1}{2t} \ln \left(2t + \sqrt{1 + 4t^2}\right),
\]

\[
L_t(t) = \left(L(t)\right)_t = \frac{\sqrt{4t^2 + 1}}{t} - \frac{1}{2t^2} \ln \left(2t + \sqrt{1 + 4t^2}\right),
\]

were used to compute the right boundary velocity. The mesh trajectory plot in Figure 6.6 clearly shows the creation of grid nodes which enter at the right domain boundary.

### 6.3 Example 3: a Non-uniform Mesh where \( L(t) \) is Decreasing

#### 6.3.1 \( \psi(\eta) \) Monotonically Increasing in \( \eta \)

We now try to evolve a mesh to equi-distribute the arc-length of a specified \( u(x, t) \) where the integral of the monitor function is decreasing. We use a variant of the previous example by letting

\[ u(x, t) = (1 - t)(1 - x^2), \quad -1 \leq x \leq 1, \quad 0 < t < 1, \]

\[
\rho(u(x, t)) = \sqrt{1 + u_x^2},
\]

\[
= \sqrt{1 + 4(1 - t)^2x^2}. \]
Figure 6.5: Example 2: We use the right boundary condition $v(1) \propto L(t)$. This allows the number of mesh points to change, while equi-distributing the mesh points. The level set function, $\psi(y, t) \approx \int_{-1}^{1} \rho(u(x, t)) \, dx$. 
Figure 6.6: Example 2: mesh trajectories using the new boundary conditions. The number of mesh nodes increases as the total integral of the monitor function increases.
Since \( u \) is constant when \( t = 1 \), the ideal moving mesh should evolve to an equi-spaced mesh at \( t = 1 \). We initialize the level set function,

\[
\psi(\eta, 0) = \int_{-1}^{\eta} \rho(u(x, 0)) \, dx.
\]

Then, we solve MMPDE4 for mesh velocities \( v \), with the new boundary conditions

\[
\frac{\partial}{\partial \xi} \left( \rho \frac{\partial v}{\partial \xi} \right) = -\frac{1}{\tau} \frac{\partial}{\partial \xi} \left( \rho \frac{\partial x}{\partial \xi} \right),
\]

\[
v(-1) = 0, \quad v(1) = -\frac{L(t)}{|\psi_x(1, t)|},
\]

where the analytic values

\[
L(t) = \int_{-1}^{1} \sqrt{1 + 4(1-t)^2} x^2 \, dx,
\]

\[
= \sqrt{1 + 4(1-t)^2} + \frac{1}{2(1-t)} \ln \left(2(1-t) + \sqrt{1 + 4(1-t)^2}\right),
\]

\[
L_t(t) = \frac{-4(1-t)}{\sqrt{1 + 4(1-t)^2}} - \frac{1}{(1-t)\sqrt{1 + 4(1-t)^2}} - \frac{1}{2(1-t)^2} \ln \left(2(1-t) + \sqrt{1 + 4(1-t)^2}\right),
\]

are used.

In the right column of Figure 6.7, the level set function is evolved using the computed mesh velocities \( v \). Qualitatively, the level set function evolves towards a straight line, implicitly representing a mesh that evolves towards an equally spaced grid. As \( (\max \psi - \min \psi) \) decreases, grid nodes are removed, while keeping the arc-length in each sub-interval constant throughout the simulation. We also plot a graph of mesh trajectories in Figure 6.8, showing grid points leaving the domain at the right boundary.

### 6.3.2 \( \psi(\eta) \) Monotonically Decreasing in \( \eta \)

As discussed in Section 3.1, the level set initialization for a mesh is not unique. We continue with this example to explore different level set representations of the grids. In Figure 6.9, we initialize the level set function as a monotonically decreasing function of \( \eta \),

\[
\psi(\eta, 0) = \int_{-1}^{1} \rho(u(x, 0)) \, dx, \eta \in [-1, 1].
\]

Using the new boundary conditions, \( v(a) \propto L_t(t) \), and the functional form of \( \rho(u(x, t)) \), we solve for the mesh velocities using MMPDE4 (2.12), and update the level set function using the level set evolution equation (3.8) with the computed
Figure 6.7: Example 3: \( v(1) \propto L(t) \), allows number of mesh points to change, while equi-distributing the mesh points. The level set function, \( \psi(\eta,t) = \int_0^1 \rho(u(x,t)) \, dx \). The level set function used to represent the mesh is a monotonically increasing function of \( \eta \).
Figure 6.8: Example 3: Mesh trajectories using the level set function $\psi(\eta)$, which is monotonically increasing in $\eta$. The new boundary conditions allow grid points to leave the domain boundary.
velocities. In the right column, the level set function $\psi$ evolves to a straight line (representing an equi-spaced mesh) as anticipated. Also, $(\max \psi - \min \psi)$ decreases, resulting in a removal of grid nodes. The adaptive mesh and how it resolves $u$ are shown in the left column. Observe that $(\max \psi(t) - \min \psi(t)) = L(t)$. In fact,

$$\psi(\eta, t) = \int_{\eta}^{1} \rho(u(x, t)) \, dx + \beta(t), \quad \eta \in [-1, 1],$$

where

$$\beta(t) = L(t) - L(0).$$

The graph of mesh trajectories is shown in Figure 6.10. The graph clearly shows mesh points leaving the right domain boundary for this monotonically decreasing level set representation of the mesh.

### 6.3.3 $\psi(\eta)$ Non Monotonic in $\eta$

In Figure 6.11, we initialize the level set function as a function which is non-monotonic in $\eta$, but whose level contours give a mesh which equi-distributes $\rho(u(x, 0))$. Using the new boundary conditions, $u(a) \propto L(t)$, and the functional form of $\rho(u(x, t))$, we once again solve for the mesh velocities using MMPDE4, and update the level set function using the level set evolution equation (3.8) and computed mesh velocities. In this simulation however, we lose mesh points in the interior of the domain. Specifically, we lose mesh points at the “peak” of the level set function, in addition to the removal of mesh nodes at the right boundary. Intuitively, this happens at the “peak” because a mesh node is removed as soon as its level set function value drops below the threshold contour level, likely due to the discretization approximation to $\psi_x$.

Note that since $\psi$ is not monotonic in $\eta$, we need a different way to quantify how $\psi(\eta, t)$ corresponds as a function of $t$ to the generalized integral of the monitor function. We propose a total variation formula to quantify how $\psi(x, t)$ tracks $L(t)$. Discretely, we define the total variation, $q(t)$ to take the form

$$q(t) = \sum_{\tau=0}^{M-1} |\psi_{\tau+1}^n - \psi_{\tau}^n|,$$

(6.4)
Figure 6.9: Example 3: $v(1) \propto L_t(t)$, allows number of mesh points to change, while equi-distributing the mesh points. The level set function, $\psi(\eta, t) \approx \int_0^\eta \rho(x, t) \, dx + \beta(t)$, where $\beta(t) = L(t) - L(0)$. The initialized level set function, $\psi(\eta, 0)$, a monotonically decreasing function of $\eta$. 
Figure 6.10: Example 3: Mesh trajectories using a level set function $\psi(\eta)$, which is monotonically decreasing in $\eta$. The new boundary conditions allow grid points to leave the domain boundary.
where $\psi^n_r \approx \psi(\eta_r, t^n)$. Observe that if $\psi(x, t)$ is a monotonically increasing function of $\eta$,

$$
q(t) = \sum_{r=0}^{M-1} |\psi^n_{r+1} - \psi^n_r|,
$$

$$
= \sum_{r=0}^{M-1} (\psi^n_{r+1} - \psi^n_r),
$$

$$
= \psi^n_M - \psi^n_0,
$$

$$
= L(t).
$$

For this level set initialization that was non-monotonic in $\eta$, the computations lead to an equi-spaced mesh as desired, but $q(1) < L(1)$, resulting in the loss of grid points. More work needs to be done to understand why the evolution of non-monotonic level set functions (in $\eta$) results in $q(t) \neq L(t)$ with the new boundary conditions. We suggest that losing grid points at the “peak” causes the numerical code some difficulty, as mesh points need to race back to preserve the equi-distribution of the mesh. Notice in Figure 6.12 that at time $t \approx 0.05$, the level set representation loses a mesh node within the domain, and the mesh nodes need to rush back to equi-distribute the physical solution (with fewer mesh points). There is also instability in the mesh trajectories as $t$ approaches 1.

In Section 6.4.2, we discuss a special case of adding grid nodes in the interior of the domain by exploiting known symmetry in the problem.

6.4 Example 4: A Solution with Finite Time Blow-up

6.4.1 Adding Nodes at the Domain Boundary

Suppose we want a mesh to represent a solution which is exhibiting blowup in finite time,

$$
u(x, t) = e^{\frac{x}{T-t}}, \quad t \in [0, T), \quad x \in [0, 1], \quad (6.5)$$

where $T > 0$, a constant, is the blowup time, since for $x > 0$, $u \to \infty$ as $t \to T$. It was shown in [Budd et al., 1996a] that the blowup solution to the Frank-Kamenetskii equation,

$$
u_t = \nu_{xx} + \nu^p,$$

$$
u(x, 0) = 2,$$

$$
u(-1, t) = \nu(1, t) = 2,$$
Figure 6.11: Example 3: $v(1) \propto L(t)$, allows number of mesh points to change, while equi-distributing the mesh points. The initialized level set function, $\psi(\eta,0)$, is a function, non-monotonic in $\eta$. Notice that the total variation $q(t)$ does not equal $L(t)$, i.e., $q(1) = \psi(1) = 1.5$ since $\psi$ is monotonically increasing at $t = 1$, but $L(1) = 2$ (see (6.3)).
Figure 6.12: Example 3: Mesh trajectories using the non-monotonic $\psi$ and the new boundary conditions. Grid points leave the domain boundary as expected, but they also leave from inside the computational domain.
where \( p \) is an integer, exhibits a blowup structure similar to \((6.5)\). They recommended using a monitor function

\[
\rho(u) = u^{p-1},
\]

to preserve similarity solutions. They also noted that MMPDE4 did not move the mesh nodes into the blowup region rapidly enough if \( \tau \) is fixed, and that in contrast, MMPDE6 performed better for blowup problems. In this example, we set

\[
\rho(u(x, t)) = u(x, t) = e^{x^2}, \quad t \in [0, T), x \in [-1, 1],
\]

and show some results obtained using the hybrid moving mesh - level set method. MMPDE6 is used to solve for mesh velocities, and the level set function is updated using the computed mesh velocities. A plot of \( \max u(x, t) \) vs \( t \) is shown in Figure 6.13, illustrating that \( u \) is unbounded as \( t \to T \). We show in Figure 6.14 that our hybrid \( h-r \) algorithm adds grid nodes to resolve the blow up solution. In fact, these grid points are added mostly to the blowup region, as shown in the mesh trajectory plot in Figure 6.15. It is hard to quantify how many points are in the blowup region by examining the plot. A more useful plot is the density of grid nodes in the domain, where we approximate the density, \( p(x) \), as

\[
p(x_{j+1/2}) = \frac{1}{x_{j+1} - x_j}, \quad j = 0, \ldots, N - 1.
\]

We plot the density of mesh nodes at \( t = 0.85 \) in Figure 6.16, which shows that points are being added to the blowup region as desired.

### 6.4.2 Adding Nodes in the Interior of the Domain

In Section 6.3.3, we showed that a level set function, \( \psi(\eta, t) \), initialized to be non monotonic in \( \eta \), lost grid nodes (in an uncontrolled manner) in the interior of the domain and did not preserve the total variation. Here, we reexamine the blow up problem, exploiting symmetry to add grid nodes in the interior of the domain. Suppose that

\[
u(x, t) = e^{\frac{1-x^2}{T-t}}, \quad x \in [-1, 1], \quad t \in [0, T).
\]

As \( t \to T \), \( u \to \infty \) as \( x \to \pm \). Setting the monitor function \( \rho(u) = u \), we initialize the level set function as

\[
\psi(\eta, 0) = \left\{ \begin{array}{ll}
\int_{-1}^{\eta} \rho(u(x, 0)) \, dx & \text{if } \eta \in [-1, 0], \\
\int_{-1}^{-\eta} \rho(u(x, 0)) \, dx & \text{if } \eta \in (0, 1].
\end{array} \right.
\]
Figure 6.13: Example 4: in this plot of $\max u(x,t)$ vs. $t$, we show that the solution $u$ is unbounded as $t \to T$. Here, $T = 1$, and the simulation was run to $t = 0.85$. 
Figure 6.14: Example 4: we show our hybrid moving mesh - level set framework increasing the number of grid nodes gradually as $t \to T$. We can only run our simulation to $t = 0.89$. Otherwise, there are too many grid nodes added to the blowup region. Perhaps a scaling of the monitor function, or an adaptively chosen set of contour levels (on a logarithm scale) might alleviate this problem.
Figure 6.15: Example 4: the mesh trajectories for this blowup problem is plotted above. Notice that grid points are added from the right boundary domain as the solution (and $L(t)$) increases.
Figure 6.16: Example 4: we quantify the number of points in a sub region by plotting the density of grid nodes, $1/(\text{cell spacing})$, and show that mesh nodes are being added to the blowup region, as desired.
Observe that the level set function $\psi(\eta, 0)$ is symmetric about $\eta = 0$. If the level set function retains its symmetry throughout the computation, then $\frac{\partial}{\partial \eta} \psi(0, t) = 0$. Consequently, adding grid points at the blowup location, $\eta = 0$, requires a few modifications. We solve MMPDE6 with the following boundary conditions,
\[
\begin{cases}
  v(-1, t) = 0, \\
  v(0, t) = \frac{-q(t)}{2|\psi_\eta(x, t)|}, \\
  v(1, t) = 0,
\end{cases}
\]
where $q(t)$ is the total variation (6.4), and the appropriate one-sided differencing is used to approximate $\psi_\eta(\epsilon, t)$. The computed velocities are used to update the level set function. In Figure 6.17, we show the resulting mesh trajectories, showing the addition of grid nodes at $\eta = 0$. In the computation, we chose $T = 0.2$, and picked $\epsilon$ to be the grid node closest to $\eta = 0$. A few interesting features can be observed.

- Grid nodes are added in pairs at $\eta = 0$, due to the symmetry in $\psi$. It is difficult to observe this directly from the figure due to resolution limitations.

- There is some interesting behavior at $\eta = 0$ for $t > 0.07$, namely, the level set function goes from “single peaked” to “twin peaked”, likely because the time step was not sufficiently refined. This requires further investigations.

### 6.5 Example 5: Viscous Burgers’ Equation

Now that we have seen the application of the $h$-$r$ hybrid moving mesh - level set method (through the evolution of implicitly represented grids for which $u(x, t)$ is specified), we now consider the application of this approach to solving physical PDEs for $u(x, t)$. We solve a standard test problem for most numerical algorithms, namely the viscous Burgers’ equation,
\[
u_t + \frac{1}{2} (u^2)_x = \epsilon u_{xx},
\]
simultaneously with the moving mesh/level set PDE for the moving mesh. The initial condition
\[
u(x, 0) = 1 - \tanh \left( \frac{x - x_\epsilon}{2\epsilon} \right),
\]
Figure 6.17: Example 4: Mesh trajectories showing the addition of grid nodes in the interior of the domain. Here, we have exploited known symmetry in the problem to construct a level set function that is non monotonic in $\eta$. 
and boundary conditions
\[ u(0, t) = 2, \quad u_x(2, t) = \frac{1}{2\epsilon} \left( 1 - \tanh^2 \left( \frac{2 - x_c - t}{2\epsilon} \right) \right), \]
result in a travelling inner solution. The exact solution for the travelling wave is
\[ u(x, t) = 1 - \tanh \left( \frac{x - x_c - t}{2\epsilon} \right). \tag{6.7} \]
Knowing the exact solution allows us to check rates of convergence for our hybrid moving mesh - level set method.

In Figure 6.18, we show the results of using the algorithm in Section 5.5. Briefly, a level set function, \( \psi(x, t) \) is initialized to represent a mesh, equi-distributed with respect to the arc-length monitor function, \( \rho(x, t) \). Specifically,
\[ \psi(\eta, 0) = \int_0^\eta \rho(u(x, 0)) \, dx, \]
where
\[ \rho(u(x, 0)) = \sqrt{1 + u_x(x, 0)^2}. \]
The level contours of this initialized level set function give a set of mesh nodes \( \{x_j(0)\} \). The initial condition for Burgers' equation (6.6) is initialized at these mesh nodes
\[ u_j^0 = u(x_j(0), 0) = 1 - \tanh \left( \frac{x_j(0) - x_c}{2\epsilon} \right). \]
The physical solution, \( u_j^n \approx u(x_j(t^n), t^n) \) is updated using a finite volume scheme to give \( u_j^{n+1} \approx u(x_j(t^n), t^{n+1}) \). This updated solution on the frozen mesh, \( u_j^{n+1} \), is used to find a set of mesh velocities \( v_j^n = u(x_j(t^n), t^n) \) to update the level set function, implicitly updating the mesh, \( x_j(t^{n+1}) \). The physical solution is interpolated to the new mesh, \( u_j^{n+1} \approx u(x_j(t^{n+1}), t^{n+1}) \), and the process is repeated.

In Figure 6.18, the right column shows the level set representation of the mesh, while the left column shows the physical solution \( u \) and the resulting mesh. We plot the exact solution using a dotted line for comparison. Qualitatively, the level set function seems to generate an adaptive mesh that resolves the traveling wave; the level set function is linear (corresponding to an equi-spaced mesh) away from the shock, and the level set function is steep (corresponding to a clustering of mesh points) at the shock. In Figure 6.19, we show a plot of the mesh trajectories, giving another representation of how the mesh
points track the travelling shock. Qualitatively, the physical solution \( u(x,t) \) gives a good approximation to the exact solution. The solution is less accurate “behind” the shock. In [Arney & Flaherty, 1990, Beckett & Mackenzie, 2000, Beckett et al., 2001], authors show that this observation is a result of using the popular arc-length monitor function. They show in their analysis that the so-called BM monitor function

\[
\rho(u) = \alpha \left| \frac{\partial u}{\partial x}(x,t) \right|^m,
\]

gives a better proportion of grid points located in the steep area. The BM monitor function was not implemented in this thesis.

Quantitatively, we can check for convergence of our hybrid moving mesh method by calculating the error for different spatial resolutions, namely, the error as a function of the number of initial grid points. If \( u_h \) is the discretized numerical solution and \( u \) is the exact solution, we define the weighted \( p \)-norm error and infinity norm error as

\[
||u_h - u||_p^p = \left( \sum_{j=1}^{N} |u(x_{j-1/2},t) - u_h(x_{j-1/2},t)|^p \right)^{\frac{1}{p}}, \quad p \geq 1,
\]

\[
||u_h - u||_\infty = \max_{0 \leq j \leq N} |u(x_j,t) - u_h(x_j,t)|.
\]

We show the convergence results in Figure 6.20, Figure 6.21 and Figure 6.22 for the \( || \cdot ||_1 \), \( || \cdot ||_2 \) and \( || \cdot ||_\infty \) norms, respectively. The errors arising from the static uniform mesh and the moving mesh scheme (with various interpolation implementations) exhibit linear behavior in a loglog plot as the number of grid points is increased. The linear behavior in a loglog plot means that the error satisfies

\[
\text{error} = c_N N^{p_e},
\]

where \( N \) is the number of grid points, and \( p_e \) is the rate of convergence, and \( c_N \) is the error coefficient. The expected rate of convergence is \( O(\Delta x) = O(N^{-1}) \) since the discretizations for solving the PDE and updating the mesh in Section 5.5 are \( O(\Delta x) \). We present the \( || \cdot ||_1 \) error convergence plots in Figure 6.20. The order of convergence for a static uniform mesh is \( p_e \approx -0.6 \), slightly lower than expected. In contrast, the order of convergence for a moving mesh implementation, using linear interpolation to compute the explicit location of the mesh nodes and interpolating between the computational and physical grid, is slightly improved with \( p_{m1} \approx -1 \). The interesting result is the order of convergence for the moving
CHAPTER 6. NUMERICAL EXAMPLES IN $\mathbb{R}^1$

Figure 6.18: Example 5: Solving the coupled physical PDE (Burgers' equation) and level set PDE (along with MMPDE4). The tanh initial condition (6.6) results in a travelling wave solution (6.7). Note that $\psi(\eta, t) \approx \int_0^\eta \rho(u(x, t)) \, dx$. 
Figure 6.19: Example 5: a plot of the mesh trajectories, giving another viewpoint of how mesh nodes are clustered around the traveling shock.
mesh implementation with either cubic splines or Hermite cubic splines. The order of convergence is \( p_{\text{cs}} \approx -1.5 \), even though the overall scheme to solve the physical PDE is only first order. This is likely due to the non-linearity of Burgers' equation, or possibly the denseness of the mesh nodes at the sharp travelling shock. Note that there are three

![L_1 Error](image)

Figure 6.20: Example 5: We plot the \( ||\cdot||_1 \) error as a function of the number of grid points to show convergence to the exact solution. In this plot, we compare the rate of convergence for a static uniform mesh, and moving mesh schemes with varying interpolation implementations. While each scheme converges as the number of mesh nodes is increased, the moving mesh scheme with cubic spline interpolation implementations converge at a higher rate (slope \( \approx -1.5 \)) compared to a moving mesh scheme with linear interpolation (slope \( \approx -1 \)) and a static uniform mesh (slope \( \approx -0.6 \)). The same observation can made in the \( ||\cdot||_2 \) convergence plots and \( ||\cdot||_\infty \) convergence plots on the following pages.

interpolation steps per update in our algorithm, discussed in Section 5.5: (i) Step 12, where the velocity is interpolated to the computational grid, (ii) Step 15, where the new grid nodes are found from the level set function, and (iii) Step 16, where the solution is updated from the old mesh to the new mesh. Implementing a cubic spline interpolation in all three of the above steps may be unnecessary, and possibly introduces oscillations in the solution. In our simulations, it appears that the improvement in error convergence occurs when the cubic
Figure 6.21: Example 5: We plot the $|| \cdot ||_2$ error as a function of the number of grid points to show convergence to the exact solution. The same behavior is observed and discussed in Figure 6.20.
Figure 6.22: Example 5: We plot the $\| \cdot \|_{\infty}$ error as a function of the number of grid points to show convergence to the exact solution. The same behavior is observed and discussed in Figure 6.20.
interpolation is used to interpret the solution from the old mesh to the new mesh (i.e., using linear interpolation for Steps 12 and 15, and cubic spline interpolation for Step 16 yielded equivalent results to using cubic spline interpolations everywhere).

We also simulate the shock creation case which arises from the initial and boundary conditions

\[ u(x, 0) = \sin \pi x, \quad 0 < x < 2, \]
\[ u(0) = u(2) = 0. \]

The results are shown in Figure 6.23. Qualitatively, the mesh behaves as expected by clustering more grid points at \( x = 1 \), the location of the shock creation. The mesh trajectories are shown in Figure 6.24.

### 6.6 Example 6: Advection-Diffusion Equation

The advection-diffusion equation takes the form

\[ u_t + u_x = Du_{xx}, \tag{6.8} \]

where \( D \) is the diffusion constant. This equation can be used to model a variety of groundwater problems, including advection dominated chemical transport and reaction, nonlinear infiltration in soil, and the coupling of density dependent flow and transport. See [Huang et al., 2002] for a more thorough description of such models.

If (6.8) is solved on \( 0 < x < 1 \) using the initial condition

\[ u(x, t_0) = \frac{1}{2} \text{erfc} \left( \frac{x - t_0}{\sqrt{4Dt_0}} \right) + \frac{1}{2} \exp \left( \frac{x}{D} \right) \text{erfc} \left( \frac{x + t_0}{\sqrt{4Dt_0}} \right), \]

and Dirichlet boundary conditions

\[ u(0, t) = \frac{1}{2} \text{erfc} \left( \frac{t}{\sqrt{4Dt}} \right) + \frac{1}{2} \text{erfc} \left( \frac{t}{\sqrt{4Dt}} \right), \]
\[ u(1, t) = \frac{1}{2} \text{erfc} \left( \frac{1 - t}{\sqrt{4Dt}} \right) + \frac{1}{2} \exp \left( \frac{1}{D} \right) \text{erfc} \left( \frac{1 + t}{\sqrt{4Dt}} \right), \]

then there is an exact solution

\[ u(x, t) = \frac{1}{2} \text{erfc} \left( \frac{x - t}{\sqrt{4Dt}} \right) + \frac{1}{2} \exp \left( \frac{x}{D} \right) \text{erfc} \left( \frac{x + t}{\sqrt{4Dt}} \right), \quad t > t_0. \]
Figure 6.23: Example 5: Solving the coupled physical PDE (Burgers' equation) and level set PDE (along with MMPDE#4 (2.12)). The sinusoidal initial condition results a shock developing at $x = 1$. 

\[ u(t, x) = 0.0005, \#\ points = 92 \] 

\[ u(t, 0.1), \#\ points = 92 \] 

\[ u(t, 0.2), \#\ points = 92 \] 

\[ u(t, 0.2995), \#\ points = 92 \]
Figure 6.24: Example 5: solving the coupled physical PDE (Burgers’ equation) and level set PDE (along with MMPDE4 (2.12)). The sinusoidal initial condition results in a shock developing at $x = 1$. The mesh trajectories confirm that there is a clustering of grid points at $x = 1$ as the physical PDE is solved, and the mesh is updated.
We solve (6.8) and associated initial and boundary conditions with $D = 10^{-3}$, $t_0 = 10^{-4}$. The arc-length monitor function and MMPDE4 (2.12) are used to move the mesh in the hybrid moving mesh - level set framework. Snapshots of the simulation are shown in Figure 6.25. One can clearly see the effects of numerical diffusion (flattening of the shock) in the discretization scheme used in Section 5.3.1. Recall the upwind difference scheme to approximate $u_x$,

$$(u_x^+)_j = \frac{u_{j+1} - u_j}{x_{j+1} - x_j}, \quad (u_x^-)_j = \frac{u_j - u_{j-1}}{x_j - x_{j-1}}.$$ 

Taking the Taylor series expansion

$$u_{j+1} = u_j + (u_x)_j(x_{j+1} - x_j) + \frac{1}{2}(u_{xx})_j(x_{j+1} - x_j)^2 + O(x_{j+1} - x_j)^3,$$

we see that

$$\frac{u_{j+1} - u_j}{x_{j+1} - x_j} = (u_x)_j + \frac{1}{2}(u_{xx})_j(x_{j+1} - x_j) + O(x_{j+1} - x_j)^2.$$

The term

$$\frac{1}{2}(u_{xx})_j(x_{j+1} - x_j),$$

is the source of the numerical diffusion. In Figure 6.26, Figure 6.27, and Figure 6.28, we show the convergence of the hybrid level set - moving mesh framework to the exact solution by checking the error as a function of the number of grid points, and compare the results with a static uniform grid. In Figure 6.26, the rate of convergence for MMPDE4 with cubic spline interpolations is $\approx O(N^{-1})$, while the rate of convergence for linear interpolation is $\approx O(N^{-0.8})$, while the rate of convergence for the static framework is $\approx O(N^{-0.6})$. Graphs of mesh trajectories are shown in Figure 6.29. Qualitatively, mesh nodes cluster around the travelling shock. Some interesting behavior is observed in this mesh trajectory plot; mesh nodes are initially removed as the wave the travelling wave flattens. As the wave approaches the right boundary, mesh nodes are added through the right boundary to resolve the shock. This is a result of the velocities near the right boundary being negative (causing grid points to move to the left), in turn resulting in the creation of mesh nodes near the right boundary due to the level set representation of the mesh. Notice that the addition of mesh nodes at $t \approx 0.8$ affects the global location of mesh nodes (i.e., all the nodes have to shift to accommodate the creation of the mesh nodes). As the shock exits through the right boundary, mesh nodes are once again removed.
Figure 6.25: Example 6: Time evolution of the advection-diffusion equation. The effects of diffusion are quite noticeable, even in a moving mesh framework.
Figure 6.26: Example 6: We present an $L_1$ error comparison of our moving mesh results to that of a static uniform mesh. The rate of convergence for MMPDE4 with cubic spline interpolation is $\approx O(N^{-1})$, the rate of convergence for MMPDE4 with linear interpolation is $\approx O(N^{-0.8})$, and the rate of convergence for the static framework is $\approx O(N^{-0.6})$. 
Figure 6.27: Example 6: We present an $L_2$ error comparison of our moving mesh results to that of a static uniform mesh. The rate of convergence for MMPDE4 with cubic spline interpolation is $\approx O(N^{-0.9})$, the rate of convergence for MMPDE4 with linear interpolation is $\approx O(N^{-0.7})$, and the rate of convergence for the static framework is $\approx O(N^{-0.5})$. 
Figure 6.28: Example 6: We present an $L_\infty$ error comparison of our moving mesh results to that of a static uniform mesh. The results are not as sharp as the previous example involving Burgers’ equation. The rate of convergence for MMPDE4 with cubic spline interpolation is $\approx O(N^{-0.8})$, the rate of convergence for MMPDE4 with linear interpolation is $\approx O(N^{-0.6})$, and the rate of convergence for the static framework is $\approx O(N^{-0.4})$. 
Figure 6.29: Example 6: The resulting mesh trajectories when using the hybrid level set-moving mesh method. Notice that the mesh nodes cluster near the travelling shock. There is also some interesting "action" at the right boundary with respect to the removal, addition, and removal of mesh points.
Chapter 7

Numerical Examples in $\mathbb{R}^2$

In this chapter, we show some preliminary results using our moving mesh - level set framework to create moving meshes resolving a specified $\rho(x, y, t)$ in $\mathbb{R}^2$. We extend on work previously done in [Liao et al., 2000] by implementing a different moving mesh method for finding mesh velocities, and comparing the results with an explicit representation of the mesh. Specifically, the GCL equation is solved for mesh velocities, with zero Neumann boundary conditions, $\frac{\partial \phi}{\partial n} = 0$ (corresponding to zero Dirichlet boundary velocities, $v|_{\partial \Omega} = 0$). The computed velocities are used to evolve the level set representation of the mesh, implicitly moving the embedded mesh. The presented problems are taken from [Cao et al., 2002] to allow for qualitative comparison.

We note that examples in this chapter do not involve any addition or removal of mesh nodes. Implementation of the new boundary conditions derived in Section 4.6.2 is beyond the scope of this thesis.

7.1 Example 8: Resolving a Circle

Following in the footsteps of [Cao et al., 2002], we generate an adaptive mesh for the density function,

$$d(x, y) = 1 + Ae^{-50[(x-0.5)^2+(y-0.5)^2-0.25^2]}, \quad (x, y) \in \Omega = [0, 1] \times [0, 1].$$

Here, $A$ is a parameter that controls the ratio of the largest cell size to the smallest one, and $\Omega$ is a rectangular domain of interest. We will use this density function to define a monitor function in the following lines. However, observe that the density function attains
its maximum value, \( d(x, y) = 1 + A \), on a circle specified by

\[
(x - 0.5)^2 + (y - 0.5)^2 - 0.25^2 = 0,
\]

that is, a circle centered at \((0.5, 0.5)\) with radius 0.25. The monitor function decays exponentially to \( \rho = 1 \) outside of the circle. In [Cao et al., 2002], authors state that an ideal mesh should be about \( 1 + A \) times denser around the circle specified by (7.1).

To generate this adaptive mesh, we first define the time-dependent monitor function

\[
\rho(x, y, t) = \frac{1 + t A e^{-50[(x-0.5)^2+(y-0.5)^2-0.25^2]}}{\int 1 + t A e^{-50[(x-0.5)^2+(y-0.5)^2-0.25^2]} dS}, \quad 0 \leq t \leq 1, \quad \Omega \in [0, 1] \times [0, 1].
\]

Notice that this time dependent monitor function, \( \rho(x, y, t) \) changes gradually from a constant, \( \rho(x, y, 0) = 1 \), to the desired density function, \( \rho(x, y, 1) = d(x, y) \). We initialize the two level set functions,

\[
\psi_1(x, y, 0) = x, \quad \psi_2(x, y, 0) = y,
\]

and use the GCL to solve for mesh velocities, \( v(x, y, t) \). The time-dependent monitor function, \( \rho(x, y, t) \), along with Neumann boundary conditions,

\[
\frac{\partial \phi}{\partial n}|_{\partial \Omega} = v|_{\partial \Omega} = 0,
\]

are used. The computed velocities are then used to advect the level set functions, implicitly moving the grid. Two sets of results are shown. The left column of Figure 7.1 shows the evolution of the overlaid contour plots of \( \psi_1(x, y, t) \) and \( \psi_2(x, y, t) \) with \( A = 5 \) and weight function \( w = 1 \). The choice of weight functions \( w \) (introduced in Section 2.3) corresponds to different moving the mesh schemes. In [Cao et al., 2002], the authors argue that \( w = 1 \) is a superior choice to \( w = \rho \), the deformation method. The right column shows the evolution of the overlaid contour plots of using the deformation method, \( A = 5 \) and \( w = \rho \). Comparing the bottom plots of Figure 7.1, there are no observable differences between using the GCL and the deformation mapping method. The computational grid is \( 40 \times 40 \), \( \Delta t = 0.1 \), and the iso contours,

\[
\bar{c} = [0 : 0.02 : 1],
\]

are used.
Figure 7.1: Example 8: Using different moving mesh schemes to generate an adaptive mesh for resolving a circle. The case $A = 5$, $w = 1$ is on the left column, $A = 5$ and $w = \rho$ on the right column. There is no noticeable difference between using the weight function $w = 1$ and $w = \rho$. 
It is interesting that the meshes obtained using this hybrid method do not cluster the expected density of points around the circle defined by (7.1). How can we quantify this? In \( \mathbb{R}^1 \), recall that a reasonable measure of density is

\[
\text{density} = \frac{1}{\text{spacing between grid nodes}}.
\]

In \( \mathbb{R}^2 \), a reasonable measure of density is

\[
\text{density} = \frac{1}{\text{cell areas}}.
\]

This is consistent with the measure of density from \( \mathbb{R}^1 \), since for cells with smaller cell area, a larger number can fit into a region of fixed size, and hence, a smaller cell area will lead to a higher density of cells. The same argument also works for larger cell areas, lower density. Instead of calculating cell areas, we note that the gradient of the level set functions gives a good quantitative measure of the number of iso-contours locally. The number of iso contours in a region is “equivalent” to the number of sub-regions defined by the level set function,

\[
\text{number of sub-regions} \propto |\nabla \psi|.
\]

A measure of the cell dimension is in turn inversely proportional to the number of sub-regions, hence,

\[
\text{cell dimension} \propto \frac{1}{|\nabla \psi|}.
\]

A plot of mesh densities, \( p(x, y) \) can thus be expressed as

\[
p(x, y) = \alpha|\nabla \psi_1||\nabla \psi_2|,
\]

where \( \alpha \) is some proportionality constant. In Figure 7.2, we show a plot of the mesh at time \( t = 1 \), and the associated mesh density, \( p(x, y) \), for the GCL method with \( A = 5 \) and \( w = 1 \) and the deformation method, \( A = 5 \) and \( w = \rho \). The proportionality constant is set as \( \alpha = 1 \). Recall that in [Cao et al., 2002], the authors state that an ideal mesh should be about \( 1 + A \) times denser around the circle specified by (7.1). In our simulations, we were only able to obtain a mesh which had a density two to three times as much in the circle defined by (7.1) compared to the rest of the domain. If we increase to \( A = 20 \), we get a higher density of mesh nodes around the circle as expected, but nowhere near the predicted
Figure 7.2: In the left plot, we show the final mesh obtained by evolving the level set function using the GCL method with \( A = 5 \), and \( w = 1 \). In the right plot, we show the final mesh obtained using the Deformation method with \( A = 5 \), and \( w = \rho \). In both plots, the mesh density (7.2) near the circle defined by (7.1) is between two to three times as dense as the rest of the region.
value in [Cao et al., 2002]. The meshes and densities obtained with $A = 20$ are shown in Figure 7.3. We use the same computational grid, $40 \times 40$, and the same contour levels,

$$\bar{c} = [0 : 0.02 : 1],$$

but reduce the time step to $\Delta t = 0.01$, to allow for a more gradual change in the time dependent monitor function, $\rho(x, y, t)$.

![Figure 7.3](image)

Figure 7.3: In the left plot, we show the final mesh obtained by evolving the level set function using the GCL method with $A = 20$, and $w = 1$. In the right plot, we show the final mesh obtained using the Deformation method with $A = 20$, and $w = \rho$. In both plots, the mesh density (7.2) near the circle defined by (7.1) is between five to six times as dense as the rest of the region.

There have been suggestions that a level set representation of a mesh cannot be used implicitly to represent very fine meshes. We show in the Figure 7.4 that if we scale the velocities obtained from solving the GCL is increased by a factor of two, we can indeed
represent a very fine mesh using the level set representation of a mesh. In fact, the density of mesh nodes around the circle defined by (7.1) is as much as 10 times as dense as the rest of the domain. It is unclear if a more stable moving mesh method, say the MMPDE in higher dimensions, would help cluster the desired density of mesh nodes around the circle defined by (7.1). The parameters used are the same used as those to generate the last set of plots.

![Image of mesh densities](image)

Figure 7.4: In the left plot, we show the final mesh obtained by evolving the level set function using an amplified velocity from the GCL method with \( A = 20 \), and \( w = 1 \). In the right plot, we show the final mesh obtained using an amplified velocity from the Deformation method with \( A = 20 \), and \( w = \rho \). In both plots, the mesh density (7.2) near the circle defined by (7.1) is between five to six times as dense as the rest of the region.
CHAPTER 7. NUMERICAL EXAMPLES IN $\mathbb{R}^2$

7.2 Example 9: Travelling Circle

A moving mesh is generated for the time-dependent monitor function specified by

$$\rho(x, y, t) = \frac{d(x, y, t)}{\int d(x, y, t) dS},$$

(7.3)

where

$$d(x, y, t) = \begin{cases} 
1 + 100(t + 0.1)e^{-50[(x-0.5)^2+(y-0.5)^2-0.25^2]}, & \text{for } -0.1 < t < 0, \\
1 + 10e^{-50[(x-0.5-t)^2+(y-0.5)^2-0.25^2]}, & \text{for } t \geq 0.
\end{cases}$$

Notice that we have utilized the homotopy argument described in (4.1). This is similar to a problem in [Cao et al., 2002]. Notice that $d(x, y, -0.1) = 1$, which corresponds to an equally spaced mesh. In the interval $t \in [-0.1, 0]$, the monitor function changes in a similar fashion to Example 8, clustering more grid points around a circle defined by

$$(x - 0.5)^2 + (y - 0.5)^2 - 0.25^2 = 0.$$ 

For $t > 0$, the monitor function simulates a circular peak that travels to the right with speed 1, and eventually leaves the domain while maintaining its shape. Once again, we initialize the level set functions,

$$\psi_1(x, y, 0) = x, \quad \psi_2(x, y, 0) = y,$$

and use the GCL to solve for mesh velocities, $v(x, y, t)$. Neumann boundary conditions,

$$\left. \frac{\partial \phi}{\partial n} \right|_{\partial \Omega} = v|_{\partial \Omega} = 0,$$

are used. The computed velocities are then used to advect the level set functions, implicitly moving the grid. Two sets of results are shown. In Figure 7.5, we overlay the contour levels of $\psi_1(x, y, t)$ with $\psi_2(x, y, t)$ to give the resulting moving mesh. We use the weight function $w = 1$. In Figure 7.6, we plot the corresponding results obtained using the deformation mapping ($w = \rho$). In [Cao et al., 2002], implementation of the GCL leads to a “non-smooth” mesh, whereas using the hybrid moving mesh - level set algorithm leads to a smoother mesh. While this is an improvement, there is still a problem, namely, the final mesh shows an image of the circular object at the end of the simulation. This image does not disappear even if the spatial and time discretization is refined. A computational mesh of $40 \times 40$ is used, and the iso-contours

$$\mathcal{C} = [0 : 0.02 : 1],$$
are used. Time steps, $\Delta t = 0.005$ are used for the initialization, and $\Delta t = 0.01$ is used to evolve the level set function for $t > 0$.

### 7.3 Example 10: Oscillating Membrane

This third example in [Cao et al., 2002] was chosen to demonstrate the effect of discretization error and the non smoothness of meshes generated by the GCL method. Qualitatively, using the level set representation of mesh has removed some of the non-smoothness observed in [Cao et al., 2002]. A quantitative study of how smooth the mesh is, and how well it is adapting to $\rho$ could be done by following recent suggestions from [Huang, 2005].

The density function is given as

$$d(x, y, t) = \begin{cases} 1 + 50(t + 0.1)e^{-50|y-0.5|} & \text{for } -0.1 < t < 0, \\ 1 + 5e^{-50[y-0.5-0.25\sin(2\pi t)\sin(2\pi t)]} & \text{for } t \geq 0, \end{cases}$$

and the monitor function is obtained using equation (7.3).

Once again the density/monitor function is defined in two time phases. For $t < 0$, it serves to create an adaptive mesh for $t = 0$ starting from a uniform mesh at $t = -0.1$. The evolution of the level set functions for the initialization process,

$$\psi_1(x, y, 0) = x, \quad \psi_2(x, y, 0) = y,$$

are shown in Figure 7.7. For $t > 0$, the monitor function simulates the motion of a periodic sine wave. We use $w = 1$ to generate the overlaid contour plots in Figure 7.8. The computational grid is $40 \times 40$, $\Delta t = 0.005$ for the initialization process ($t < 0$), and $\Delta t = 0.01$ for $t > 0$. The iso-contours used to generate the plots are

$$\vec{c} = [0 : 0.02 : 1].$$

### 7.4 Example 11: A Rotating Circle

The last example models a small circle which rotates around the point $(0.5, 0.5)$. This is a difficult test problem for many moving mesh methods as the mesh can easily become skewed, and eventually singular. The density function is given by

$$d(x, y, t) = \begin{cases} 1 + 50(t + 0.1)e^{50[(x-0.75)^2+(y-0.5)^2+0.01]} & \text{for } -0.1 < t < 0, \\ 1 + 5e^{50[(x-0.75-0.25\cos(2\pi t))^2+(y-0.5-0.25\sin(2\pi t))^2+0.01]} & \text{for } t \geq 0, \end{cases}$$
Figure 7.5: Example 9: A moving mesh simulating a circle moving to the right with speed 1, weight function $w = 1$. 
Figure 7.6: Example 9: A moving mesh simulating a circle moving to the right with speed 1, deformation mapping ($w = \rho$). Notice the “ghost” of the initial circle at the end of the simulation, and the unexpected distribution of nodes at the right boundary.
Figure 7.7: Example 10: Initializing an adaptive mesh clustering more points at \( y = 0.5 \).
Figure 7.8: Example 10: A moving mesh simulating an oscillating sine wave, $w = 1$. The plots are generated by overlaying the iso contours of $\psi_1(x, y, t)$ and $\psi_2(x, y, t)$. The results agree qualitatively with [Cao et al., 2002].
and the monitor function is obtained using equation (7.3). Figure 7.9 shows the initialization of the mesh from a uniformly spaced mesh to an adaptive mesh resolving a circle of radius 0.1, centered at (0.75, 0.5). In Figure 7.10, the moving mesh simulates a circle traveling around the origin. The results agree qualitatively with results from [Cao et al., 2002].

7.5 Discussion

In the previous four examples, we have successfully initialized level set functions $\psi_1(x, y, 0)$ and $\psi_2(x, y, 0)$ to resolve a specified $\rho(x, y, 0)$. The initialization technique in Section 4.2 was implemented, which results in a smoothed version of the normalized integral initialization in Section 3.2.4, equation (3.7).

A moving mesh scheme (GCL) was used to generate mesh velocities to evolve the implicitly represented mesh. The resulting grids cluster grid points in desired regions, though the density of the clustered points is less than expected. We conjecture that a moving mesh method (such as an MMPDE) which moves a mesh towards equi-distribution, rather than a moving mesh method which keeps a mesh equi-distributed (such as the GCL), would cluster the expected density of mesh points in desired regions. Our resulting grids however display a desired property of smoothness, compared to results in [Cao et al., 2002]. This re-emphasizes the discussion in Section 3.2.4 that adaptivity in higher dimensions is always a compromise between getting a high concentration of grid points in a localized region, and creating elements which are not too skewed.

In $\mathbb{R}^2$, it is desirable to implement new boundary conditions that allow for the natural addition and removal of grid nodes.
Figure 7.9: Example 11: Initializing an adaptive mesh resolving a circle of radius 0.1, centered $(0.75, 0.5)$. 
Figure 7.10: Example 11: A moving mesh simulating a circle traveling around (0.5,0.5). The results agree qualitatively with [Cao et al., 2002].
Chapter 8

Conclusions and Future Work

In this thesis, we successfully construct a new $h$-$r$ hybrid moving mesh method. This moving mesh method redistributes a changing number of mesh nodes, adding or removing grid nodes as solution features change.

In Chapter 2, we briefly review two moving mesh methods, MMPDEs and GCL, both based on the equi-distribution principle. We are primarily interested in utilizing both moving mesh methods as “velocity-based” methods, i.e., mesh velocities are computed to update a mesh. While it is difficult to make general conclusions about the relative merits of each method, some comments are appropriate. The GCL framework extends naturally to $\mathbb{R}^2$ and $\mathbb{R}^3$, but appears to be less stable than the MMPDE framework. Here, we consider stability in the sense that the MMPDEs move a mesh towards equi-distribution, whereas the GCL can only attempt to keep a mesh equi-distributed. An interesting consequence not documented in this thesis, is that the GCL acts as a “local” mesh movement method, utilizing $\frac{\partial g}{\partial t}$ in the calculation of mesh velocities, $v$, whereas the MMPDEs are a more “global” movement strategy, designed to redistribute mesh nodes based on the global behavior of $\rho$. The advantages of using a “local” versus “global” mesh movement strategy to update an implicit representation of a mesh is an open and interesting research question.

In Chapter 3, we introduce implicit representations of meshes. In $\mathbb{R}^1$, a level set function, $\psi(\eta)$, is used to represent a mesh. Specifically, a discrete set of $\psi(\eta)$ iso-contours give the explicit location of the mesh. In $\mathbb{R}^2$, the intersection of a discrete set of $\psi_1(\eta)$ iso-contours with a discrete set of $\psi_2(\eta)$ iso-contours give the mesh location. In $\mathbb{R}^3$, the intersection of a discrete set of $\psi_1(\eta)$, $\psi_2(\eta)$ and $\psi_3(\eta)$, give the explicit location of the mesh. In $\mathbb{R}^1$, we use
ideas from equi-distribution to generate the level set representation of the mesh,

\[ \psi(\eta) = \int_{a}^{b} \rho(x) \, dx. \]

In \( \mathbb{R}^2 \) and \( \mathbb{R}^3 \), we show that a similar strategy of equi-distributing along lines (i.e., decoupling the \( x \) and \( y \) directions, or \( x, y \), and \( z \) directions), gives a poor initialization for the implicitly represented mesh. Before presenting an alternative initialization in Chapter 4, we derive the level set evolution equation for implicitly moving the mesh, and discuss advantages for using a level set representation of a mesh. Two interesting open problems are (i) the smoothing of an implicitly defined mesh by reinitializing the level set function, and (ii) the exploitation of connectivity information embedded in the level set representation of a mesh.

In Chapter 4, we discuss the moving mesh - level set method, first introduced in [Liao et al., 2000], and utilize the framework for initializing the level set representation of a mesh in higher dimensions. A new set of boundary conditions are derived to allow for a more general interpretation of the level set function in \( \mathbb{R}^1 \). Specifically, we derive boundary conditions so that

\[ \psi(\eta, t) = \int_{a}^{b} \rho(x, t) \, dx. \]

This more general interpretation allows for the natural creation or removal of grid nodes, since mesh nodes, \( \{ x_j(t) \} \), are given as the discrete set of \( \psi(\eta, t) \) iso-contours,

\[ \{ x_j(t) \} = \{ x(t) \mid \psi(x(t), t) = c_j, \quad c_j = j \Delta \psi \}, \quad j = 0, \ldots, \text{floor} \left( \frac{L(t)}{\Delta \psi} \right), \]

where

\[ L(t) = \int_{a}^{b} \rho(x, t) \, dx. \]

If \( L_1(t) > 0 \), a larger number of grid nodes may be utilized to equi-distribute \( \rho(x, t) \), whereas if \( L_1(t) < 0 \), a smaller number of grid nodes may be utilized to equi-distribute \( \rho(x, t) \). A challenge associated with using the hybrid moving mesh - level set framework is the presence of “multiple grids”: a computational grid, \( \eta \), for the level set function \( \psi(\eta) \), and a physical grid, \( \{ x_j \} \), the iso-contour levels of \( \psi(\eta) \) (in \( \mathbb{R}^1 \)). Three open challenges that surface are (i) find a reformulation of the physical PDE, or perhaps the level set PDE to eliminate the presence of multiple grids, (ii) find new boundary conditions that allow for the addition of grid points within the domain instead of through the boundary, and (iii) implementation
and study of new boundary conditions for $\mathbb{R}^2$ and $\mathbb{R}^3$. Also of great interest is now “de-coupling” the moving mesh - level set framework, i.e., as in Section 4.6.3, can we implement new boundary conditions for moving mesh methods without employing level set functions?

Chapter 5 gives implementation details, emphasizing the presence of multiple grids described above. In Chapter 6, we present examples in $\mathbb{R}^1$ to show the evolution of the implicitly represented grids for known functions $u(x,t)$ and selected monitor functions $\rho(u(x,t))$, and contrast the differences between using the old and new boundary conditions. Then, we solve Burgers’ equation and an advection-diffusion equation, and show convergence of our hybrid moving mesh - level set scheme to the exact solution. It appears that the choice of interpolation schemes affects the rate of convergence of the numerical scheme. This warrants a more careful study of the effect of interpolation in this hybrid moving mesh - level set framework.

Lastly, we show some preliminary numerical results for $\mathbb{R}^2$ in Chapter 7. The GCL equation is solved for mesh velocities, and the computed velocities are used to evolve the level set representation of the mesh, implicitly moving the embedded mesh. It is interesting that the density of clustered nodes obtained using the hybrid GCL - level set method, is less than expected in regions of interest. The meshes obtained, however, display a desired property of smoothness, reemphasizing that adaptivity in higher dimensions is a compromise between getting a high concentration of grid nodes in a localized region, and creating elements that are not to skewed. Open research questions in $\mathbb{R}^2$ are numerous. As previously mentioned, do the proposed new boundary conditions for our moving mesh - level set framework in $\mathbb{R}^2$ obtain the desired effect? If an iso-contour of $\psi_1(\tilde{\eta})$ intersects with an iso-contour of $\psi_2(\tilde{\eta})$ more than once, how does that affect the equi-distribution of $\rho(\tilde{x})$? Would a different moving mesh scheme (other than the GCL) couple with the hybrid level set framework to cluster the desired density of grid nodes in a region? These are just some of the questions that are worthy of further study.
Bibliography


BIBLIOGRAPHY


