Characteristics Mapping Methods and Application to Adaptive Moving Mesh

Xi Yuan Yin

Master of Science

Department of Mathematics and Statistics

McGill University

Montreal,Quebec

August 2016

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Science in Mathematics and Statistics

ACKNOWLEDGEMENTS

I would like to thank my supervisors Prof. Jean-Christophe Nave and Prof. Linan Chen for their guidance and support. I also thank the Natural Sciences and Engineering Research Council of Canada for its financial support.

The new ideas in Chapters 3 and 4 were developed with Prof. Nave. The mesh management method in Chapter 5 was developed with Prof. Nave and Prof. Chen.

ABSTRACT

The characteristics mapping method has many applications in numerical mathematics, notably in the simulation of moving surfaces. There are several versions of this method, each adapted to different problems. In this thesis, we present characterisitcs mapping methods used to evolve implicit and parametric surfaces. We also attempt to implement a method designed for the cases where these two types of surfaces are simulated at the same time. Finally, we propose an application of the characteristics mapping methods to the mesh adaptivity problem. This is used to control the quality of the simulations for parametric surfaces.

ABRÉGÉ

La méthode de l'application des caractéristiques a plusieurs utilitées en mathématiques numériques, notamment en simulation de surfaces mobiles. Il existe plusieurs versions de cette méthode, chacune adaptée à de différentes problèmes. Dans cette thèse, nous présentons des méthodes de l'application des caractéristiques utilisées pour l'évolution des surfaces implicites et paramétriques. Nous tentons aussi d'implémenter une méthode conçue pour les cas où ces deux types de surfaces évoluent en même temps. Finalement, nous proposons une utilisation des méthodes de l'application des caractéristiques pour résoudre le problème du maillage adaptatif. Celle-ci a pour but de contrôler la qualité des simulations des surfaces paramétriques.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

Simulations of moving surfaces are useful in many real life applications. For instance, problems in computer graphics often involve finding the position and shape of a surface that changes under the influence of a time dependent velocity field. Images and videos of objects such as cloth can be generated this way by computer. In fluid dynamics, the interface between two non-mixing fluids, such as oil and water, is a surface that evolves with the fluid flows. Solving for the movement of the interface is essential for simulating the behavior of these fluids.

The behaviors of these physical systems are very difficult to simulate directly: we simply do not have the computing power and memory to keep track of every molecule in play. Instead, the macroscopic features of these surfaces are captured and approximated by mathematical equations, notably partial differential equations (PDEs). These equations model the change of interesting quantities such as position, velocity, curvature etc. throughout the simulation. Once formulated mathematically, these equations are then discretized and solved numerically on a computer, providing the information we need to reproduce the simulated surfaces at different times.

There are many algorithms used for this task. Depending on the surfaces we are given, some methods work better than others. Here we will mainly focus on Characteristics Mapping, a method that performs surface evolution by integrating the characteristic ODEs of an advection equation. Furthermore, we will propose a

particle management method based on Characteristics Mapping. Particle management will allow us to control the way information is stored on the surface, thus to improve the efficiency of the simulations.

In this thesis, we will first review in Chapter 2 some preliminary mathematical concepts. Chapter 3 discusses the theory behind Gradient Augmented Level Set and Characteristics Mapping methods and provides an analysis of convergence. We then apply these methods to surface evolution and carry out numerical experiments in Chapter 4. In Chapter 5, we design a particle management method applied to our problem. Finally, Chapter 6 contains some final remarks and propose potential future research on the subject of surface advection.

CHAPTER 2
Preliminaries

In this chapter, we will introduce a few mathematical concepts that we will use throughout the paper. We will give a quick overview of finite difference approximation for derivatives, polynomial interpolation, numerical ODE solvers, numerical integration via Gaussian quadrature, implicit and explicit definitions for surfaces, and characteristic equations for PDEs.

Finite Difference Schemes

Finite difference schemes are used to approximate derivatives of continuously differentiable functions. We can obtain these by writing out the Taylor expansion of a function f at a given point x_0. For a $k+1$ times continuously differentiable (C^{k+1}) function $f : \mathbb{R} \to \mathbb{R}$, we have the following expansion:

$$f(x) = \sum_{i=0}^{k} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i + R_k(x) \tag{2.1}$$

where $R_k(x)$ is the remainder term and is of order $O((x - x_0)^{k+1})$.

We can approximate derivatives of f at x_0 by evaluating the function at points $x_j = x_0 + \alpha_j \Delta x$ for some chosen α_j's and Δx small. Using these values, we can set up formulæ or "finite difference schemes" of the form $f^{(m)}(x_0) \approx \sum_{j=1}^{s} \beta_j f(x_j)$ to approximate the derivatives. One can solve for the appropriate coefficients β_j by expanding $f(x_j)$ around x_0 as in (2.1). For instance, $f'(x_0)$ can be approximated by a centered difference with $\alpha_1 = -1$, $\beta_1 = \frac{-1}{2\Delta x}$, $\alpha_2 = 1$ and $\beta_2 = \frac{1}{2\Delta x}$:

$$f'(x_0) = \frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x} + O(\Delta x^2) \tag{2.2}$$

This scheme can be generalized to higher dimensions to approximate the gradient vector of a given function.

Also, for the second derivative, we can use

$$f''(x_0) = \frac{f(x_0 + \Delta x) - 2f(x_0) + f(x_0 - \Delta x)}{\Delta x^2} + O(\Delta x^2) \tag{2.3}$$

this scheme can be used to approximate the Laplacian operator $\Delta = \sum_{i=1}^{d} \partial_{x_i}^2$ in d-dimensional space.

In some cases we work on a rectangular domain equipped with a mesh grid, for instance, the square $[0, 1]^2$ with grid points $(n/N, m/M)$ for $n = 0, 1, \ldots, N$ and $m = 0, 1, \ldots, M$. Oftentimes, the function values of f will be provided at the grid points. Then we can find derivatives of f using "cell-based finite difference". We will choose $\Delta x, \Delta y$ to be the distance between adjacent grid points. The derivatives at a grid point will be approximated by a finite difference of the function values at neighboring grid points.

In 2 dimensions, we will often use the 4-points stencil to approximate the gradient of a function f. Let $\vec{x}_{i,j} = (x_i, y_j)$ be the grid points of a regular grid. We have uniform spacing in each dimension, that is, $x_{i+1} - x_i = \Delta x \, \forall i$ and $y_{j+1} - y_j = \Delta y \, \forall j$. We denote $f_{i,j} = f(\vec{x}_{i,j})$. Then, we have that the gradient and the second order

mixed partial are approximated by:

$$f_x(\vec{x}_{i,j}) = \frac{f_{i+1,j+1} + f_{i+1,j-1} - f_{i-1,j+1} - f_{i-1,j-1}}{4\Delta x} + O(\Delta x^2)$$
$$f_y(\vec{x}_{i,j}) = \frac{f_{i+1,j+1} - f_{i+1,j-1} + f_{i-1,j+1} - f_{i-1,j-1}}{4\Delta y} + O(\Delta y^2)$$
$$f_{xy}(\vec{x}_{i,j}) = \frac{f_{i+1,j+1} - f_{i+1,j-1} - f_{i-1,j+1} + f_{i-1,j-1}}{4\Delta x \Delta y} + O(\Delta x \Delta y)$$

Polynomial Interpolation

The main purpose of polynomial interpolation is to provide a polynomial approximation to a regular enough function f everywhere over some given interval I. The usual construction of the interpolating polynomial, called interpolant, involves evaluating the function and/or its derivatives at given distinct sample points $\{x_i\}_{i=0}^n \in I$. For instance, the Lagrange interpolant is given by $L_f(x) = \sum_{i=0}^n f(x_i) l_i(x)$. This interpolation uses degree n basis polynomials $l_i(x) = \prod_{j=0}^n \frac{x-x_j}{x_i-x_j}$. These satisfy $l_i(x_j) = \delta_{i,j}$, where $\delta_{i,j}$ denotes the Kronecker delta; it is equal to 1 iff $i = j$ and 0 otherwise. It can be shown that the Lagrange polynomial is the unique polynomial of degree n that matches the value of f at every sample point. However, this method suffers from some accuracy issues, notably the Runge effect where higher order polynomials exhibit large oscillations between sample points.

In this paper, we work with $d-$dimensional rectangular domains obtained from Cartesian product of intervals. These are equipped with mesh grids of uniform spacing in each dimension. Since we have a large number of evenly spaced grid points, we opt for piecewise polynomial interpolations instead of high degree Lagrange polynomials. These interpolants are piecewise defined on an interval $I = [a, b]$ with

sample points $a = x_0 < x_1 < \ldots < x_n = b$; they are polynomial in each subinterval $[x_i, x_{i+1}]$. The interpolants will match the function values and derivatives of the interpolated function at the grid points, and may be chosen to satisfy some continuity and differentiability conditions.

For instance, the interpolation method we will use in most of this paper is the Hermite Cubic spline. It is piecewise cubic. In 1 dimension, it matches the function and its derivative at the grid points. In higher dimensions, it matches the gradient and all partial derivatives with at most one differentiation in each variable. The resulting interpolant is therefore C^1 on the whole domain.

The following definitions and notation for the Hermite Cubic interpolant are taken from [8]. We will use this notation in Chapter 3.

Definition 2.0.1. The four Hermite basis function w_0^0, w_0^1, w_1^0 and w_1^1 are cubic polynomials $[0, 1] \to \mathbb{R}$ defined as follows:

$$\begin{aligned} w_0^0(x) &= 1 - 3x^2 + 2x^3 & w_0^1(x) &= 3x^2 - 2x^3 \\ w_1^0(x) &= x^3 - 2x^2 + x & w_1^1(x) &= x^3 - x^2 \end{aligned} \tag{2.4}$$

These basis functions have the property that $\partial^\beta w_\alpha^v(u) = \delta_{\alpha\beta}\delta_{uv}$, for α, β, u and $v \in \{0, 1\}$. That is, each basis function only contributes a value of 1 to either function or derivative value at exactly one of the two endpoints.

We can scale this interpolant from $[0, 1]$ to an interval $[x_i, x_{i+1}]$ with length Δx_i. $x \in [x_i, x_{i+1}]$ can be mapped back to the unit interval by $x \mapsto \frac{x - x_i}{\Delta x_i}$. Therefore, given

a C^1 function $f(x)$, we have that

$$p(x) = f(x_i)w_0^0\left(\frac{x-x_i}{\Delta x_i}\right) + f(x_{i+1})w_0^1\left(\frac{x-x_i}{\Delta x_i}\right) + f'(x_i)w_1^0\left(\frac{x-x_i}{\Delta x_i}\right)\Delta x_i + f'(x_{i+1})w_1^1\left(\frac{x-x_i}{\Delta x_i}\right)\Delta x_i \qquad \text{for } x \in [x_i, x_{i+1}] \quad (2.5)$$

gives us a piecewise polynomial interpolation that matches f and its derivative at all grid points.

This interpolation can be generalized in d dimensions using a tensor product of the basis functions.

Definition 2.0.2. The $d-$cubic Hermite basis functions are polynomials $[0,1]^d \subset \mathbb{R}^d \to \mathbb{R}$ obtained from a tensor product of d Hermite basis defined in (2.4). We denote by $\vec{x}$ the $d-$dimensional vector $\vec{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(d)})$ where $x^{(k)}$ is the value of the k^{th} coordinate.

$$W_{\vec{\alpha}}^{\vec{v}}(\vec{x}) = \prod_{k=1}^{d} w_{\alpha^{(k)}}^{v^{(k)}}(x^{(k)}) \quad (2.6)$$

We write $\partial^{\vec{\beta}} = \partial_1^{\beta^{(1)}}\partial_2^{\beta^{(2)}}\ldots\partial_d^{\beta^{(d)}}$ for the mixed partial derivative given by the vector $\vec{\beta} \in \{0,1\}^d$. As before, $\vec{u}, \vec{v} \in \{0,1\}^d$, that is $\vec{u}$ is a corner of the hypercube.

Then, from the properties of Hermite basis functions, we have that

$$\partial^{\vec{\beta}} W_{\vec{\alpha}}^{\vec{v}}(\vec{u}) = \prod_{k=1}^{d} \delta_{\alpha^{(k)}\beta^{(k)}}\delta_{u^{(k)}v^{(k)}} = \delta_{\vec{\alpha}\vec{\beta}}\delta_{\vec{u}\vec{v}} \quad (2.7)$$

where $\delta_{\vec{a}\vec{b}} = 1$ if $a^{(k)} = b^{(k)}$ for every $k = 1, 2, \ldots d$ and 0 otherwise.

This interpolant is defined on the unit hypercube $[0,1]^d$ and can also be scaled to an arbitrary hypercube of side lengths h.

The domains we work with are equipped with a $d-$dimensional uniform mesh grid. We will use the vector $\vec{i}$ to index the grid points. These are given by $\vec{x}_{\vec{i}} = (x_{i_1}^{(1)}, x_{i_2}^{(2)}, \ldots, x_{i_d}^{(d)})$, $x_{i_k}^{(k)}$ is the $i_k{}^{th}$ grid point in the k^{th} dimension. We say that a point $\vec{x}$ lies in the $\vec{i}^{th}$ cell if $x^{(k)} \in [x_{i_k}^{(k)}, x_{i_k+1}^{(k)}]$ for $k = 1, 2, \ldots, d$. We will use the following notation to refer to the corners of the cell which contains a point $\vec{x}$. Let $\vec{v} \in \{0,1\}^d$. For $\vec{x}$ a point in the cell $\vec{i}$, we denote by $\vec{x}_{[\vec{v}]}$ the grid point given by $x_{[\vec{v}]}^{(k)} = x_{i_k+v_k}^{(k)}$ or in short, $\vec{x}_{[\vec{v}]} = \vec{x}_{\vec{i}+\vec{v}}$. For instance, $\vec{x}_{[\vec{0}]}$ is the corner which is lowest in every dimension, i.e. the "cell floor" for $\vec{x}$, and $\vec{x}_{[\vec{1}]}$ is the highest corner, the "cell ceiling". In 2 dimensions, they correspond respectively to the lower left and upper right corners of the cell containing $\vec{x}$.

Definition 2.0.3. The $d-$cubic Hermite interpolant for a function f at position $\vec{x}$ is given as follows:

$$H[\vec{x}, \hat{f}] = \sum_{\vec{v},\vec{\alpha} \in \{0,1\}^d} f_{\vec{\alpha}}^{\vec{v}} h^{|\vec{\alpha}|} W_{\vec{\alpha}}^{\vec{v}} \left(\frac{\vec{x} - \vec{x}_{[\vec{0}]}}{h} \right) \tag{2.8}$$

Here, $f_{\vec{\alpha}}^{\vec{v}}$ refers to the $\partial^{\vec{\alpha}}$ mixed partial derivative of the function f evaluated at the cell corner $\vec{x}_{[\vec{v}]}$. We denote by $\hat{f}$ the array of function and derivative values $f_{\vec{\alpha}}^{\vec{v}}$ at grid points for all $\vec{v}, \vec{\alpha} \in \{0,1\}^d$. Also, $|\vec{\alpha}| = \sum \alpha_i$ and $\vec{x} - \vec{x}_{[\vec{0}]}$ is the position of $\vec{x}$ relative to the $\vec{x}_{[\vec{0}]}$ grid point of the cell containing $\vec{x}$.

Two immediate consequences of this definition is that H is linear in $\hat{f}$, and it follows from (2.7) that

$$\partial^{\vec{\alpha}} H[\vec{x}_{[\vec{v}]}, \hat{f}] = f_{\vec{\alpha}}^{\vec{v}} = \partial^{\vec{\alpha}} f(\vec{x}_{[\vec{v}]}) \tag{2.9}$$

It is known that for f a C^2 function, this interpolant can approximate $f(\vec{x})$ to order $O(h^4)$ and the gradient $\nabla f(\vec{x})$ to $O(h^3)$ everywhere in the domain. Further

details on this interpolant and its application to the advection problem can be found in [8].

Numerical Solutions to ODEs

The ordinary differential equations (ODEs) we deal with in this paper are simple first order equations of the form

$$\begin{aligned} \frac{d}{dt}Y(t) &= v(Y,t) \\ Y(0) &= Y_0 \end{aligned} \tag{2.10}$$

We want to approximate the solution $Y(t)$ at discrete times steps Δt apart. That is, we want to find $Y(n\Delta t)$ for n an integer. Integrating both sides of the equation with respect to t, we get that the solution must satisfy

$$Y(t+\Delta t) = Y(t) + \int_t^{t+\Delta t} v(Y(\tau),\tau)d\tau \tag{2.11}$$

Hence, approximating the integral on the right gives us a way to find $Y(n\Delta t)$ by iteratively applying (2.11) starting from $Y(0)$. Different schemes used to approximate the integral give rise to a variety of ODE solvers.

For instance, the left endpoint approximation $\int_t^{t+\Delta t} v(Y(\tau),\tau)d\tau = v(Y(t),t)\Delta t + O(\Delta t^2)$ gives the forward Euler method $y(t+\Delta t) = y(t) + v(y(t),t)\Delta t$, which has a local truncation error (LTE) of $O(\Delta t^2)$. Here, the lowercase y denotes the numerical approximation. Similarly, the right endpoint rule gives the implicit Euler method $y(t+\Delta t) = y(t) + v(y(t+\Delta t), t+\Delta t)\Delta t$. This method also has an LTE of $O(\Delta t^2)$. However, implicit methods are slower and more complicated numerically: since the unknown appears on both sides of the equality, an implicit equation needs to be

solved. In both cases, if we want the solution $Y(T)$ at some fixed time T, the number of times step we take is $T/\Delta t$. It can then be shown that the global truncation error (GTE) $|Y(T) - y(T)|$ is of order $O(\Delta t)$ for both methods.

The trapezoidal rule offers better accuracy on the time integration, however, like implicit Euler, it is an implicit method and is slower to solve on a computer. Instead, we approximate the right endpoint by first using a forward Euler step. This gives us Heun's method (also known as improved Euler):

$$y(t + \Delta t) = y(t) + \frac{1}{2}\big(v(y(t), t) + v(\tilde{y}, t + \Delta t)\big)\Delta t \tag{2.12}$$

where $\tilde{y} = y(t) + v(y(t), t)\Delta t$. This method has global accuracy of order $O(\Delta t^2)$.

In this paper, we will use a family of ODE solvers known as Runge-Kutta methods. The Heun's method above is one of the second order explicit methods in the Runge-Kutta family. We will also use a third and a fourth order explicit methods given below. These are abbreviated as RK3 and RK4.

$$\begin{aligned} y(t + \Delta t) &= y(t) + \Delta t \sum_{i=1}^{s} b_i k_i \\ k_i &= v\left(y(t) + \sum_{j=1}^{i-1} a_{i,j} k_j \Delta t, t + \sum_{j=1}^{i-1} a_{i,j} \Delta t\right) \end{aligned} \tag{2.13}$$

Equation (2.13) is the general form of an explicit Runge-Kutta method. For RK3, we selected $b_1 = 1/6$, $b_2 = 2/3$, $b_3 = 1/6$, $a_{2,1} = 1/2$, $a_{3,1} = -1$ and $a_{3,2} = 2$. For RK4, we chose $b_1 = 1/6$, $b_2 = 1/3$, $b_3 = 1/3$, $b_4 = 1/6$, $a_{2,1} = 1/2$, $a_{3,1} = 0$, $a_{3,2} = 1/2$, $a_{4,1} = a_{4,2} = 0$ and $a_{4,3} = 1$.

Gaussian Quadrature

Gaussian quadrature rules are a family of numerical methods used to accurately approximate a definite integral. The general n-points quadrature rule estimates the integral of $f(x)$ over $[-1, 1]$ using n function evaluations. The approximation has the form:

$$\int_{-1}^{1} f(x) \approx \sum_{i=1}^{n} w_i f(x_i) \qquad x_i \in [-1, 1] \tag{2.14}$$

The quadrature rules we use in this paper are the Gauss-Legrendre quadratures. The weights w_i and sample points x_i are chosen such that all polynomials of degree $2n-1$ or less are integrated exactly by formula (2.14). This means that scaling the quadrature rule to the interval $[x - \Delta x, x + \Delta x]$, the method achieves $O(\Delta x^{2n+1})$ accuracy for $C^{2n}([x - \Delta x, x + \Delta x])$ functions:

$$\int_{x-\Delta x}^{x+\Delta x} f(x)dx = \Delta x \sum_{i=1}^{n} w_i f(\tilde{x}_i) + O(\Delta x^{2n+1}) \quad \text{for } \tilde{x}_i = x + x_i \Delta x \tag{2.15}$$

This can be generalized to 2 dimensions where the sample points are coordinate-wise the same as in the 1 dimensional case. The algorithm in Chapter 5 was implemented with a 4 points quadrature rule which corresponds to a 2 points quadrature in 1 dimension. We used the sample points $(x_i, y_j) = \left(\pm\frac{1}{\sqrt{3}}, \pm\frac{1}{\sqrt{3}}\right)$ with equal weights $w_{i,j} = 1$. For general quadrilaterals, the sample points are mapped from the $[-1, 1]^2$ square to the quadrilateral using a linear map, and the weights are scaled by area.

Implicit and Explicit Definitions for Surfaces

In this paper, we use two ways to define surfaces: by explicit parametrization and by implicit level set functions.

A parametric surface S is represented as the image of a subset of $\mathbb{R}^2$ by a parametrization function $\vec{P} : U \subset \mathbb{R}^2 \to S \subset \mathbb{R}^3$. $\vec{P}$ is given by: $x = P_1(u, v)$, $y = P_2(u, v)$, $z = P_3(u, v)$ where $(u, v) \in U \subset \mathbb{R}^2$ are parameters. The parametrization function $\vec{P}$ not only provides the location of the surface, it also contains some useful information about the surface geometry. The Jacobian of the parametrization, denoted $\nabla\vec{P}$ gives us a pushforward of the tangent space at (u_0, v_0) in U to $\vec{P}(u_0, v_0)$ on S; it can be used to map a vector in the parametric space to a vector in the tangent space of S and vice-versa. The first fundamental form of the surface with respect to this parametrization is also relevant. We will mainly use the coefficients $E = \vec{P}_u \cdot \vec{P}_u$, $G = \vec{P}_v \cdot \vec{P}_v$ and $F = \vec{P}_u \cdot \vec{P}_v$ to compute the area element $A = \sqrt{EG - F^2}$. This is used to find the surface area corresponding to the image of a subset $R \subset U$. We have that the area of $\vec{P}(R)$ is given by:

$$Area(\vec{P}(R)) = \int_R \sqrt{EG - F^2}\, dudv \tag{2.16}$$

This will be useful when defining a mesh density for redistributing sample points on a surface in Chapter 5.

We can also define a surface implicitly without relying on a parametric space. Indeed, we can define the surface as the level set of some function $\Phi : \mathbb{R}^3 \to \mathbb{R}$. For instance, the zero level set of $\Phi_1(x, y, z) = \sqrt{x^2 + y^2 + z^2} - 1$ is the unit sphere in $\mathbb{R}^3$. Note that for a same surface, these level set functions are not unique: we can also define the unit sphere as the zero level set of $\Phi_2(x, y, z) = (\sqrt{x^2 + y^2 + z^2} - 1)^2$. However, numerically speaking, Φ_1 is a much better choice. Indeed, a small perturbation or error in Φ_2 can result in drastic changes for the surface represented. We

see that even for small $\epsilon > 0$, the zero level set of $\Phi_2 + \epsilon$ is the empty set whereas the level set of $\Phi_1 + \epsilon$ is a sphere of radius $1 - \epsilon$. Similarly, $\Phi_2 - \epsilon$ gives two spheres of radius $1 \pm \sqrt{\epsilon}$, $\Phi_1 - \epsilon$ gives a sphere of radius $1 + \epsilon$.

For the reasons above, we require that the gradient of the level set function be nonzero at any point on the level set. Furthermore, it can be shown via Taylor expansion that an ϵ perturbation in the function value causes an error on the order of $\epsilon/|\nabla\Phi|$ on the location of the level set. Hence, we would like $|\nabla\Phi|$ to be nearly constant on the surface so that numerical error doesn't produce drastic changes in the shape of the surface, as the example below will demonstrate. Another reason why we require $|\nabla\Phi|$ to be constant is that in some applications such as fluid dynamics, we need to compute an ϵ neighborhood or "tube" around the surface. A constant $|\nabla\Phi|$ is necessary to ensure that this tube has uniform width.

Here we demonstrate a case where $|\nabla\Phi|$ is not constant along the surface in a 2 dimensional case: consider the function $\Phi(x, y) = (x^2 + x + 0.275)(\sqrt{x^2 + y^2} - 1)$ whose zero level set is the unit circle. Figure 2–1 shows the level set $\{\vec{x} : \Phi(\vec{x}) = c\}$ at $c = -0.01, 0$ and 0.01. Clearly, this is not a good level set function to represent the unit circle since a slight perturbation results in very inaccurate curves.

As we can see, the ideal level set function for representing a surface S should satisfy the following:

$$\begin{aligned} \Phi(\vec{x}) = 0 &\iff \vec{x} \in S \\ |\nabla\Phi(\vec{x})| = 1 &\quad \forall \vec{x} \in S \end{aligned} \tag{2.17}$$

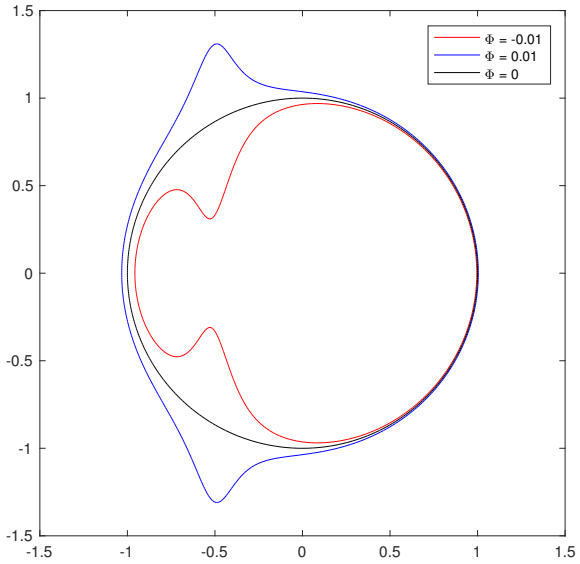


Figure 2–1: Effect of Gradient for Level Set Functions

One way of finding a good level set function is by solving an Eikonal equation, i.e. extending the gradient constraint in equation (2.17) to the whole space. This gives us the signed distance function Φ, with $|\Phi(\vec{x})| = d(\vec{x}, S)$. Φ is set to be negative on one side of the surface and positive on the other.

These requirements make finding a good level set function difficult if not impossible in some cases. For instance, non-orientable surfaces cannot be represented this way since there is no continuous choice of normal vectors on the surface. Suppose Φ is a continuously differentiable level set function and $|\nabla\Phi|$ is non-zero and constant on the surface, then we have that the unit normal is $\vec{n} = \nabla\Phi/|\nabla\Phi|$. However, this choice of normal vector is continuous, hence we have a contradiction.

Characteristic Equations for PDEs

A more complete and rigorous treatment of characteristic equations can be found in Chapter 3 of Evans [5]. The following is a summary:

The method of characteristics is a technique used to solve a first order PDE by transforming it into a family of ODEs which we can solve.

Let the PDE be given by $F(Du, u, \xi) = 0$, where F is the left-hand side of the equation involving Du, u and ξ. $\xi \in \Omega$ and $u : \Omega \to \mathbb{R}$ is the solution. Here Du denotes the vector of all first order partial derivatives (gradient) including the time derivative if applicable. We can see F as a function from $\mathbb{R}^d \times \mathbb{R} \times \Omega$ to $\mathbb{R}$ where d is the dimension of Ω (hence also the dimension of the space containing Du).

The method of characteristics seeks to find families of parametrized curves in the space $\mathbb{R}^d \times \mathbb{R} \times \Omega$ such that each point on the curve solves $F = 0$. For this purpose, we let s be the parameter and write $\xi = \xi(s)$, $z(s) = u(\xi(s))$ and $p(s) = Du(\xi(s))$. We look for curves $(p(s), z(s), \xi(s))$ that satisfy $F(p(s), z(s), \xi(s)) = 0$. This gives us a way to associate a value of u to each ξ_0 by finding the curve and the parameter s_0 for which $\xi_0 = \xi(s_0)$, i.e. $u(\xi_0) = u(\xi(s_0)) = z(s_0)$. The parametrization for the curve (p, z, ξ) is obtained by solving the characteristic ODEs whose initial values are given by the boundary conditions of the PDE:

$$
\begin{aligned}
p'(s) &= -\partial_\xi F(p, z, \xi) - \partial_z F(p, z, \xi) p(s) \\
z'(s) &= \partial_p F(p, z, \xi) \cdot p(s) \\
\xi'(s) &= \partial_p F(p, z, \xi)
\end{aligned}
$$

For the case of linear homogeneous PDEs of the form

$$F(Du, u, \xi) = b(\xi) \cdot Du(\xi) + c(\xi) u(\xi) = 0$$

the characteristic equations simplify to:

$$z'(s) = -c(\xi(s))z(s) \tag{2.18}$$
$$\xi'(s) = b(\xi(s))$$

In this paper, we are especially interested in the advection equation:

$$\Phi_t(\vec{x}, t) + \vec{v}(\vec{x}, t) \cdot \nabla\Phi(\vec{x}, t) = 0 \tag{2.19}$$
$$\Phi(\vec{x}, 0) = 0$$

We can rewrite (2.19) in the previous formulation as:

$$F(D\Phi, \Phi, \xi) = \begin{pmatrix} \vec{v}(\xi) & 1 \end{pmatrix}^T \cdot D\Phi(\xi) = 0 \tag{2.20}$$

where ξ denotes the vector $\begin{pmatrix} \vec{x} & t \end{pmatrix}^T$. In the context of (2.18), $c(\xi) = 0$ and $b(\xi) = \begin{pmatrix} \vec{v}(\xi) & 1 \end{pmatrix}^T$. Then, the characteristic equations are:

$$z'(s) = 0 \tag{2.21}$$
$$\xi'(s) = \begin{pmatrix} \vec{v} & 1 \end{pmatrix}^T \tag{2.22}$$

This gives us that $s = t + t_0$ for t_0 a constant and $\frac{d\vec{x}}{dt} = \vec{v}(\vec{x}, t)$. Also, $\frac{d\Phi}{ds} = \frac{d\Phi}{dt} = 0$, that is, Φ is constant with time along $\vec{x}(t)$. In summary:

$$\frac{d}{dt}\Phi(\vec{x}(t), t) = 0 \tag{2.23}$$
$$\frac{d}{dt}\vec{x}(t) = \vec{v}(\vec{x}(t), t)$$

This tells us that $\Phi(\vec{x}_1, t_1) = \Phi(\vec{x}_0, t_0)$ as long as $\vec{x}(t_0) = \vec{x}_0$ and $\vec{x}(t_1) = \vec{x}_1$.

CHAPTER 3
Gradient Augmented Level Set and Characteristics Mappings Methods

Oftentimes, in problems that involve moving fluids such as air or water, we are interested in the behavior of objects under the influence of the fluid flows. Examples include ink in water, smoke in air or air bubbles in water. These are few of many situations where surface advection can be used to produce numerical simulations. There are two main approaches to the advection problem: Eulerian and Lagrangian. The difference lies in the reference frame we use at a given time in the advection. The Eulerian approach uses the current time space as its reference frame, whereas the Lagrangian approach uses the initial space.

The classical analogy is as follows: using the Eulerian frame is like sitting on the river banks and observing the water at the same place as time advances. The Lagrangian frame would correspond to selecting a piece or "parcel" of water and follow it down the river along the flow: the reference frame is the initial space.

For simulating moving surfaces under some velocity flow, one of these two approaches can be more appropriate depending on the way the surface is defined. For surfaces defined implicitly as a level set, the Eulerian specification is more natural. This gives rise to Level Set Advection methods. For surfaces defined explicitly via parametrization, we use the Lagrangian approach.

The implicit methods represent the surface as the zero contour of a scalar level set function defined in the ambient space. Standard level set methods advect the

level set function using the linear advection equation [9]. Improvements on overall accuracy and subgrid structures representation were proposed in [8]. It is important to note that while implicit methods work well for closed orientable surfaces, their application to surfaces with boundaries or non-orientable surfaces is problematic.

On the other hand, explicit methods represent the surface parametrically. Mathematically speaking, we are evolving the parametrization function so that it represents the moving surface as time advances. Numerically, a collection of sample points on the surface are selected to represent its shape and position. We simply trace out the trajectories of these sample points in time to produce the simulation. These methods are fast and straightforward, and unlike level set methods, they perform equally well on all surfaces.

For problems where multiple surfaces are evolved under the same velocity field, we may wish to use both implicit and explicit methods at the same time. A version of characteristics mapping (CM) methods was introduced in [8] which, using implicit definition of surfaces, allows for the advection of multiple surfaces with essentially the same computational complexity as standard gradient augmented level set (GALS) methods. In this chapter we extend the CM methods to parametric surfaces. This extension, in conjunction with the CM method in [8] allows us to evolve multiple surfaces, both implicit and explicit, at the same time.

In our setup, the simulated surface is subject to a time dependent velocity field $\vec{v} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$. We let each $\vec{x}_0 \in S_0$ be the initial condition of the following initial value problem:

$$\frac{d}{dt}\vec{X}(\vec{x}_0, t_0, t) = \vec{v}(\vec{X}(\vec{x}_0, t_0, t), t) \tag{3.1}$$
$$\vec{X}(\vec{x}_0, t_0, t_0) = \vec{x}_0$$

These equations essentially say that $\vec{X}$ is the position of a particle at time t if it was at position $\vec{x}_0$ at time t_0. The notation doesn't assume that $t > t_0$ and is also used to describe positions of particles in the past. The map $\vec{X}(\cdot, t_1, t_2) : \mathbb{R}^d \to \mathbb{R}^d$ have the special structure that for adjacent time intervals $[t_1, t_2]$ and $[t_2, t_3]$, (t_1, t_2, t_3 completely arbitrary), a composition of maps gives a new map for $[t_1, t_3]$:

$$\big(\vec{X}(\cdot, t_2, t_3) \circ \vec{X}(\cdot, t_1, t_2)\big)(\vec{x}) = \vec{X}(\vec{X}(\vec{x}, t_1, t_2), t_2, t_3) = \vec{X}(\vec{x}, t_1, t_3) \tag{3.2}$$

This also implies that every map has an inverse:

$$\vec{X}(\cdot, t_1, t_2) \circ \vec{X}(\cdot, t_2, t_1) = \vec{X}(\cdot, t_2, t_1) \circ \vec{X}(\cdot, t_1, t_2) = id(\cdot) \tag{3.3}$$

We see that the map $\vec{X}$ encodes the motion generated by $\vec{v}$, hence, we use it to define the moving surface.

Definition 3.0.1. Given velocity field $\vec{v}(\vec{x}, t)$ and an initial surface S_0 at time t_0, the moved surface $S(t)$ at time t is defined as:

$$S(t) = \Big\{\vec{X}(\vec{x}_0, t_0, t) \big| \vec{x}_0 \in S_0\Big\} \tag{3.4}$$

3.1 Mathematical Setup for the GALS method

Level set methods use a level set function $\Phi_0 : \mathbb{R}^d \to \mathbb{R}$ to represent a $(d-1)$-manifold $S_0 \subset \mathbb{R}^d$ as its zero contour. The classical level set method consists of

finding $S(t)$ by advecting the level set function Φ through the following PDE:

$$\begin{aligned} \Phi_t + \vec{v} \cdot \nabla \Phi &= 0 \\ \Phi(\vec{x}, 0) &= \Phi_0(\vec{x}) \end{aligned} \tag{3.5}$$

From Chapter 2, we know that equation (3.5) is a linear advection equation. It has the property that the solutions to equation (3.1) are exactly the characteristic curves of the PDE. That is, Φ is constant along the curve $\vec{X}(\vec{x}_0, t_0, t)$ parametrized by t. This implies that given $S_0 = \{\vec{x} \in \mathbb{R}^d | \Phi_0(\vec{x}) = 0\}$, we have

$$S(t) = \{\vec{x} \in \mathbb{R}^d | \Phi(\vec{x}, t) = 0\} \tag{3.6}$$

This is the definition of the surface we use in the level set advection context.

The GALS method improves on the classical one by adding an equation to (3.5) specifying the evolution of the gradient $\nabla\Phi$:

$$\begin{aligned} \Phi_t + \vec{v} \cdot \nabla \Phi &= 0 \\ (\nabla\Phi)_t + \vec{v} \cdot \nabla(\nabla\Phi) &= -\nabla\Phi \cdot \nabla\vec{v} \\ \Phi(\vec{x}, 0) &= \Phi_0(\vec{x}) \end{aligned} \tag{3.7}$$

The second equation in (3.7) is obtained by simply applying the gradient operator on both sides of (3.5). However, given that $\frac{d\vec{x}}{dt} = \vec{v}(\vec{x}, t)$, (3.7) can be rewritten as

$$\begin{aligned} \frac{d}{dt}\Phi(\vec{x}(t), t) &= 0 \\ \frac{d}{dt}(\nabla\Phi(\vec{x}(t), t)) &= -\nabla\Phi \cdot \nabla\vec{v} \end{aligned} \tag{3.8}$$

This formulation is useful for the numerical implementation of the method. Although mathematically speaking, the gradient equation is redundant, it can be used to improve numerical accuracy. Indeed, the gradient of Φ allows us to construct a Hermite Cubic interpolant which can provide better approximations.

To build the numerical scheme, we will use the fact that Φ stays constant along characteristic curves. Essentially, the method will update the value of Φ at time $t_2 > t_1$ using Φ at t_1 through the following equations:

$$\Phi(\vec{x}, t_2) = \Phi(\vec{X}(\vec{x}, t_2, t_1), t_1) \tag{3.9}$$

$$\nabla\Phi(\vec{x}, t_2) = \nabla\Phi(\vec{X}(\vec{x}, t_2, t_1), t_1) - \int_{t_1}^{t_2} \nabla\Phi \cdot \nabla\vec{v} dt \tag{3.10}$$

The function evaluations will be performed by interpolation and the integral will be approximated using one step of an ODE solver.

3.1.1 Numerical Implementation of GALS

The formulation and notations we use in this section are sumarized from [8], more details on GALS methods can be found there.

We will approximate the level set function at discrete times steps $n\Delta t$ where Δt is the step size and n is integer. We denote by $\phi(\vec{x}, n\Delta t)$ the approximation to $\Phi(\vec{x}, t)$ at $t = n\Delta t$. This will be obtained by iteratively applying equation (3.9) with $t_1 = n\Delta t$ and $t_2 = (n+1)\Delta t$.

The approximation of the level set function is constructed using an approximation of Φ at grid points. We denote by $\hat{\phi}(n\Delta t)$ the array of estimated function and derivative values of Φ at the grid points. We define the following approximate level

set function using the Hermite Cubic interpolant (see definition 2.0.3):

$$\phi(\vec{x}, n\Delta t) = H[\vec{x}, \hat{\phi}(n\Delta t)] \tag{3.11}$$

The estimated level set function $\phi(\vec{x}, t)$ can be advected by updating the grid values $\hat{\phi}(t)$ at discrete time steps $n\Delta t$. Given $\hat{\phi}(n\Delta t)$ we can find $\hat{\phi}((n+1)\Delta t)$ using the characteristic equations stated in (3.9). Since Φ is constant along characteristic curves, for a grid point $\vec{x}_g$, we have $\Phi(\vec{x}_g, (n+1)\Delta t) = \Phi(\vec{x}_{foot}, n\Delta t)$, where $\vec{x}_{foot} = \vec{X}(\vec{x}_g, (n+1)\Delta t, n\Delta t)$. That is, $\vec{x}_{foot}$ is the position at time $n\Delta t$ of the curve passing through $\vec{x}_g$ at time $(n+1)\Delta t$; we simply trace the velocity for Δt backwards in time starting from $\vec{x}_g$. Hence, we discretize this by the following update rule:

$$\phi(\vec{x}_g, (n+1)\Delta t) = H[\mathring{\vec{x}}_g, \hat{\phi}(n\Delta t)] \tag{3.12}$$

where $\mathring{\vec{x}}_g$ is the numerical approximation of $\vec{x}_{foot}$.

The position $\mathring{\vec{x}}_g$ can be found by integrating the velocity field backwards in time for a Δt step starting from $\vec{x}_g$ using an ODE solver such as Runge-Kutta 3.

Once we found the foot point $\mathring{\vec{x}}_g$ we can proceed to updating the gradient values of $\hat{\phi}$ by solving the second equation in (3.8). This is done by integrating $-\nabla\phi \cdot \nabla\vec{v}$ for a time step Δt forward in time starting from $\nabla\mathring{\phi} = \nabla\phi(\mathring{\vec{x}}_g, n\Delta t)$.

The higher order mixed partials in $\hat{\phi}((n+1)\Delta t)$ can be obtained by cell based finite difference. We would then have enough data to define the Hermite cubic interpolant for $\phi(\cdot, (n+1)\Delta t)$.

Updating $\nabla\phi$ directly using the velocity field allows us to easily build the Hermite cubic interpolant. Furthermore, the extra information provided by the gradient of the velocity enables us to better represent subgrid structures [8].

3.2 Characteristics Mapping Method

Consider the maps $\vec{X}(\cdot, t_0, t)$ introduced in equation (3.1). Since for a fixed initial $\vec{x}_0$, these correspond exactly to the characteristic curves of the advection equation, we call them characteristics maps.

$$\vec{X}(\cdot, t_0, t_1) : \mathbb{R}^d \to \mathbb{R}^d \qquad \vec{X}(\cdot, t_0, t_1) : \vec{x}_0 \mapsto \vec{x}_1 \tag{3.13}$$

We can think of $\vec{X}$ as a mapping that identifies a particle at position $\vec{x}_0$ at time t_0 with the particle at $\vec{x}_1$ at t_1. It encodes the result of the transport generated by $\vec{v}$ between t_0 and t_1. Hence, using the fact that $\Phi(\vec{x}, t)$ is constant along characteristics, we get the following alternative expression for Φ:

$$\begin{aligned} \Phi(\vec{x}, t) &= \Phi(\vec{X}(\vec{x}, t, 0), 0) = \Phi_0(\vec{X}(\vec{x}, t, 0)) \\ \nabla\Phi(\vec{x}, t) &= \nabla\Phi_0\big|_{\vec{X}(\vec{x},t,0)} \cdot \nabla\vec{X}(\vec{x}, t, 0) \end{aligned} \tag{3.14}$$

where the second line is obtained simply by applying the gradient operator to line 1.

As we can see, equation (3.14) is the formula for $\Phi(\vec{x}, t)$ obtained by the method of characteristics. This means that for the level set method, we can simply compose the initial Φ_0 with the characteristics map to obtain the advected function.

The benefit of solving for this characteristics map $\vec{X}$ from time 0 to t is obvious: we can represent multiple initial surfaces with different level set functions and obtain

the corresponding advected level set functions by composition with $\vec{X}$. The computation of the map needs to be done only once and can be used for multiple surfaces. Furthermore, this approach allows for multiscaling using dynamic grid sizes as shown in [7]. Essentially, one can compute the characteristics maps on small subintervals of time during which the map can be properly represented on a coarse grid. The final map is obtained by composition of the individual small interval maps.

For computational purposes, we distinguish two characteristics maps: a forward and a backward map defined as below:

$$\begin{aligned} \vec{X}^B(\vec{x},t) &= \vec{X}(\vec{x},t,0) \ \text{ the backward map} \\ \vec{X}^F(\vec{x},t) &= \vec{X}(\vec{x},0,t) \ \text{ the forward map} \end{aligned} \tag{3.15}$$

The backward map traces back along the characteristic curve from $\vec{x}$ at time t to the initial starting position $\vec{X}(\vec{x},t,0)$ of the curve. The forward map finds the corresponding position on the curve at time t given a starting position $\vec{x}$ at time 0.

We have the following equations for the time derivatives of these maps and their Jacobian matrices:

$$\begin{aligned} &\vec{X}^B_t + \nabla\vec{X}^B \cdot \vec{v}(\vec{x},t) = 0 \\ &(\nabla\vec{X}^B)_t + \nabla\nabla(\vec{X}^B) = -\nabla\vec{X}^B \cdot \nabla\vec{v}(\vec{x},t) \end{aligned} \tag{3.16}$$

As with the GALS method, we rewrite the second equation as $\frac{d}{dt}\nabla\vec{X}^B = -\nabla\vec{X}^B \cdot \nabla\vec{v}$.

For the forward map, we have:

$$\vec{X}_t^F = \vec{v}(\vec{X}^F, t) \tag{3.17}$$
$$(\nabla \vec{X}^F)_t = \nabla \vec{v}(\vec{X}^F, t) \cdot \nabla \vec{X}^F$$

From now on, we denote by $\vec{\chi}^B(\vec{x}, t)$ and $\vec{\chi}^F(\vec{x}, t)$ the numerical approximations for the backward and forward maps.

The numerical implementation for the backward map is essentially the same as the one used for the advection of the level set function in GALS. We first find the foot point by pulling back the position of grid points from $t + \Delta t$ to t using $\vec{x}_{foot} = \vec{X}(\vec{x}_g, t+\Delta t, t)$. The map value is then updated by evaluating $\vec{X}^B(\vec{x}_{foot}, t) = \vec{X}^B(\vec{x}_g, t + \Delta t)$. The Jacobian matrix is updated by approximating the integral

$$\nabla \vec{X}^B(\vec{x}_g, t + \Delta t) = \nabla \vec{X}^B(\vec{x}_{foot}, t) - \int_t^{t+\Delta t} \left(\nabla \vec{X}^B \cdot \nabla \vec{v} \right) \Big|_{(\vec{X}(\vec{x}_{foot}, t, \tau), \tau)} d\tau \tag{3.18}$$

It is shown in [8] that using RK3 for finding the foot point and RK2 to evaluate the integral for the Jacobian yields a third order convergent method.

We also propose in this thesis an implementation of the forward map using Runge-Kutta time stepping and Hermite Cubic spacial interpolation. We can see from (3.17) that in order to approximate $\vec{X}^F(\vec{x}, t)$ at grid points, no evaluation of $\vec{X}^F$ at off-grid points are required; indeed, no function is composed with (to the right of) $\vec{X}^F$. Unlike the backward map, $\vec{X}^F$ and $\nabla \vec{X}^F$ can simply be pushed forward using an ODE solver. We will use RK4 to integrate $\vec{X}^F$ forward in time and RK3 to integrate the Jacobian matrix in the following expressions:

$$\vec{X}^F(\vec{x}_g, t+\Delta t) = \vec{X}^F(\vec{x}_g, t) + \int_t^{t+\Delta t} \vec{v}(\vec{X}^F(\vec{x}_g, \tau), \tau) d\tau \tag{3.19}$$

$$\nabla\vec{X}^F(\vec{x}_g, t+\Delta t) = \nabla\vec{X}^F(\vec{x}_g, t) + \int_t^{t+\Delta t} \nabla\vec{v}(\vec{X}^F(\vec{x}_g, \tau), \tau) \cdot \nabla\vec{X}^F(\vec{x}_g, \tau) d\tau$$

From this, we obtain numerical approximations $\vec{\chi}^F(\vec{x}_g, n\Delta t)$ and $\nabla\vec{\chi}^F(\vec{x}_g, n\Delta t)$ at grid points $\vec{x}_g$. Cell based finite difference on $\nabla\vec{\chi}^F$ can be used to evaluate higher order derivatives. This allows us to construct the Hermite Cubic interpolation $\vec{\chi}^F(\vec{x}, n\Delta t) = H[\vec{x}, \widehat{\vec{\chi}^F}(n\Delta t)]$. The next section will provide a proof that $\vec{X}^F$ can be approximated to $O(h^4)$ and $\nabla\vec{X}^F$ to $O(h^3)$ on the whole domain in the 3 dimensional case.

The two maps can be run independently at the same time as described in the following pseudo-code. We will try in the next section to couple the two maps in order to improve accuracy. Here is a pseudo-code for the CM methods:

Algorithm 1 Forward and Backward Characteristics Maps

Inputs: $\vec{v}, \vec{x}_g, \Delta t, T$ (here $\vec{x}_g$ refers to the collection of grid points of the mesh grid)

Outputs: $\vec{\chi}^F(\cdot, T), \vec{\chi}^B(\cdot, T)$

Initialize $\vec{\chi}^F(\vec{x}_g, 0) = \vec{\chi}^B(\vec{x}_g, 0) = \vec{x}_g,\ \nabla\vec{\chi}^F(\vec{x}_g, 0) = \nabla\vec{\chi}^B(\vec{x}_g, 0) = I_d$ and $t = 0$

while $t < T$ **do**

  $\vec{\chi}^F(\vec{x}_g, t + \Delta t) \leftarrow \vec{\chi}^F(\vec{x}_g, t) + \int_t^{t+\Delta t} \vec{v}(\vec{\chi}^F(\vec{x}_g, \tau), \tau) d\tau$ by RK4.

  $\nabla\vec{\chi}^F(\vec{x}_g, t + \Delta t) \leftarrow \nabla\vec{\chi}^F(\vec{x}_g, t) + \int_t^{t+\Delta t} \nabla\vec{v}(\vec{\chi}^F(\vec{x}_g, \tau), \tau) \cdot \nabla\vec{\chi}^F(\vec{x}_g, \tau) d\tau$ by RK3.

  $\mathring{\vec{x}}_g \leftarrow \vec{x}_g + \int_{t+\Delta t}^{t} \vec{v}(\vec{X}(\vec{x}_g, t + \Delta t, \tau), \tau) d\tau$ by RK3.

  $\vec{\chi}^B(\vec{x}_g, t + \Delta t) \leftarrow H[\mathring{\vec{x}}_g, \widehat{\vec{\chi}^B}(t)]$ by Hermite cubic interpolation.

  $\nabla\vec{\chi}^B(\vec{x}_g, t + \Delta t) \leftarrow \nabla H[\mathring{\vec{x}}_g, \widehat{\vec{\chi}^B}(t)] - \int_t^{t+\Delta t} \left(\nabla\vec{\chi}^B \cdot \nabla\vec{v}\right)\big|_{(\vec{X}(\mathring{\vec{x}}_g, t, \tau), \tau)} d\tau$ by RK2.

  $\vec{\chi}^F(\cdot, t + \Delta t) \leftarrow H[\cdot, \widehat{\vec{\chi}^F}(t + \Delta t)]$

  $\vec{\chi}^B(\cdot, t + \Delta t) \leftarrow H[\cdot, \widehat{\vec{\chi}^B}(t + \Delta t)]$

  $\Delta t \leftarrow \min(T - t, \Delta t),\ t \leftarrow t + \Delta t$

end while

3.3 Analysis of Convergence in 3 Dimensional Case

3.3.1 Forward Map

Given the Hermite cubic approximation

$$H[\vec{x}, \hat{f}] = \sum_{\vec{v}, \vec{\alpha} \in \{0,1\}^3} f_{\vec{\alpha}}^{\vec{v}} W_{\vec{\alpha}}^{\vec{v}} \left(\frac{\vec{x} - \vec{x}_{[0]}}{h} \right) h^{|\vec{\alpha}|}$$

For notation's sake, we will abbreviate $\frac{\vec{x} - \vec{x}_{[0]}}{h}$ by $[\vec{x}]_h$. This is the position of $\vec{x}$ after mapping the cell containing it to the unit cube.

Define the error at grid points as follows:

$$E_i = \max_{\substack{|\vec{\alpha}|=i \\ \vec{v}\in\{0,1\}^3}} |\partial^{\vec{\alpha}} F(\vec{x}_{[\vec{v}]}) - f_{\vec{\alpha}}^{\vec{v}}|$$

Let $e_0(\vec{x}) = |F(\vec{x}) - H[\vec{x}, \hat{f}]|$ the error in function value at an off-grid point $\vec{x}$ and let $e_1(\vec{x}) = \max_{|\vec{\alpha}|=1} |\partial^{\vec{\alpha}} F(\vec{x}) - \partial^{\vec{\alpha}} H[\vec{x}, \hat{f}]|$, the error in first order derivatives.

Lemma 3.3.1. *We have the following bounds for $e_0(\vec{x})$ and $e_1(\vec{x})$ for arbitrary $\vec{x}$: $e_0(\vec{x}) \le c_0 h^4 + \sum_{i=0}^{3} E_i h^i$. $e_1(\vec{x}) \le c_1 h^3 + \sum_{i=0}^{3} 3E_i h^i$, where c_0 and c_1 are the Hermite cubic error constants for F and ∇F.*

Proof.

$$e_0(\vec{x}) = |F(\vec{x}) - H[\vec{x}, \hat{f}]| = |F(\vec{x}) - H[\vec{x}, \hat{F} - (\hat{F} - \hat{f})]| = |F(\vec{x}) - H[\vec{x}, \hat{F}] - H[\vec{x}, \hat{F} - \hat{f}]|$$
$$\le |F(\vec{x}) - H[\vec{x}, \hat{F}]| + |H[\vec{x}, \hat{F} - \hat{f}]| \quad (3.20)$$

This holds since $H[\vec{x}, \hat{f}]$ is linear in $\hat{f}$. We have the following bound for $|H[\vec{x}, \hat{F} - \hat{f}]|$:

$$|H[\vec{x}, \hat{F} - \hat{f}]| = \left| \sum_{\vec{v},\vec{\alpha}\in\{0,1\}^3} (F_{\vec{\alpha}}^{\vec{v}} - f_{\vec{\alpha}}^{\vec{v}}) W_{\vec{\alpha}}^{\vec{v}}([\vec{x}]_h) h^{|\vec{\alpha}|} \right| \le \sum_{i=0}^{3} \sum_{\substack{\vec{v}\in\{0,1\}^3 \\ |\vec{\alpha}|=i}} \left|\partial^{\vec{\alpha}} F^{\vec{v}} - f_{\vec{\alpha}}^{\vec{v}}\right| \left|W_{\vec{\alpha}}^{\vec{v}}([\vec{x}]_h) h^{|\vec{\alpha}|}\right|$$

$$\le \sum_{i=0}^{3} \max_{\substack{\vec{v}\in\{0,1\}^3 \\ |\vec{\alpha}|=i}} \left|\partial^{\vec{\alpha}} F^{\vec{v}} - f_{\vec{\alpha}}^{\vec{v}}\right| \sum_{\substack{\vec{v}\in\{0,1\}^3 \\ |\vec{\alpha}|=i}} \left|W_{\vec{\alpha}}^{\vec{v}}([\vec{x}]_h) h^{|\vec{\alpha}|}\right| = \sum_{i=0}^{3} E_i h^i \sum_{\substack{\vec{v}\in\{0,1\}^3 \\ |\vec{\alpha}|=i}} \left|W_{\vec{\alpha}}^{\vec{v}}([\vec{x}]_h)\right| \quad (3.21)$$

The interpolation error $|F(\vec{x}) - H[\vec{x}, \hat{F}]|$ for Hermite Cubics is known to be order $O(h^4)$. Using Mathematica, we find that $\sum_{\substack{\vec{v}\in\{0,1\}^3 \\ |\vec{\alpha}|=i}} \left|W_{\vec{\alpha}}^{\vec{v}}(\vec{y})\right| \le 1 \; \forall \vec{y} \in [0,1]^3$. Hence,

from (3.20) and (3.21), we get that

$$e_0(\vec{x}) \leq |F(\vec{x}) - H[\vec{x}, \hat{F}]| + \sum_{i=0}^{3} E_i h^i = c_0 h^4 + \sum_{i=0}^{3} E_i h^i$$

For $|\vec{\beta}| = 1$, $\sum_{\substack{\vec{v}\in\{0,1\}^3 \\ |\vec{\alpha}|=i}} \left|\partial^{\vec{\beta}} W_{\vec{\alpha}}^{\vec{v}}(\vec{y})\right| \leq 3 \; \forall \vec{y} \in [0,1]^3$, and we have the following bound for e_1:

$$\begin{aligned} e_1(\vec{x}) &\leq |\partial^{\vec{\beta}} F(\vec{x}) - \partial^{\vec{\beta}} H[\vec{x}, \hat{F}]| + |\partial^{\vec{\beta}} H[\vec{x}, \hat{F} - \hat{f}]| \\ &\leq c_1 h^3 + \sum_{i=0}^{3} E_i h^{i-1} \sum_{\substack{\vec{v}\in\{0,1\}^3 \\ |\vec{\alpha}|=i}} \left|\partial^{\vec{\beta}} W_{\vec{\alpha}}^{\vec{v}}([\vec{x}]_h)\right| \leq c_1 h^3 + \sum_{i=0}^{3} 3 E_i h^{i-1} \end{aligned}$$

□

Theorem 3.3.2. *The numerical approximation $\vec{\chi}^F(\vec{x}, t)$ has global truncation error of $O(h^4)$ with respect to the sup norm.*

Proof. The grid values $\vec{\chi}^F(\vec{x}_g, t)$ and $\nabla\vec{\chi}^F(\vec{x}_g, t)$ are obtained by $RK4$ and $RK3$ respectively, they have global truncation error of order 4 and 3. In the context of Lemma 3.3.1, $E_0 = O(h^4)$ and $E_1 = O(h^3)$. The higher order derivatives are obtained by cell-based finite difference on the first order derivatives of $\vec{\chi}^F$. We lose an order of convergence for every derivative we take, so $E_2 = O(h^2)$ and $E_3 = O(h)$. Therefore, we can obtain the statement of the theorem by applying Lemma 3.3.1. □

3.3.2 Backward Map using the Gradient of the Forward Map

Based on the results we have for the convergence of the forward map, we are also able to prove convergence of a modified method for the backward map. In this modified method, we use the fact that the backward and forward characteristic maps

compose to identity to find the gradient of the backward map. The inverse function theorem implies that if $\vec{X}(\vec{X}(\vec{x}, t_0, t_1), t_1, t_0) = \vec{x}\ \forall \vec{x}$, then

$$\nabla \vec{X}(\cdot, t_0, t_1)\Big|_{\vec{x}} = \Big(\nabla \vec{X}(\cdot, t_1, t_0)\Big)^{-1}\Big|_{\vec{X}(\vec{x}, t_0, t_1)}$$

or simply

$$\nabla \vec{X}^B(\vec{x}, t) = (\nabla \vec{X}^F(\cdot, t))^{-1}\Big|_{\vec{X}^B(\vec{x}, t)}$$

Hence, if we define the approximate gradient of the backward map this way then the error on $\nabla \vec{\chi}^B(\vec{x}, t)$ is the same as the error on $\nabla \vec{\chi}^F\big|_{\vec{\chi}^B(\vec{x}, t)}$ provided that $\nabla \vec{\chi}^F$ is not ill-conditioned. In the implementation, this amounts to changing line 7 in Algorithm 1 to $\Big(\nabla \vec{\chi}^F\big|_{\vec{\chi}^B}\Big)^{-1}$. We will show below that this gives an order $O(h^3)$ method.

For notation, we let $\vec{X}_n^B(\vec{x}) = \vec{X}^B(\vec{x}, n\Delta t)$ and let $\vec{\chi}_n^B$ be the corresponding numerical approximation.

Let E_i^n be the error of the backward map on grid points defined as follows:

$$E_i^n = \max_{\substack{|\vec{a}|=i \\ \vec{v}\in\{0,1\}^3}} \left|\partial^{\vec{a}}(\vec{X}_n^B)^{\vec{v}} - (\vec{\chi}_n^B)_{\vec{a}}^{\vec{v}}\right|$$

Theorem 3.3.3. *The numerical approximation $\vec{\chi}^B(\vec{x}, t)$ has global truncation error of $O(h^3)$ with respect to the sup norm.*

Proof. We will show that $E_0^n = O(h^3)$, $E_1^n = O(h^3)$, $E_2^n = O(h^2)$ and $E_3^n = O(h)$.

Given the characteristic maps at step n, suppose we have errors E_i^n for the backward map, we will inductively derive a bound for E_i^{n+1}. Since

$$\vec{X}_{n+1}^B(\vec{x}_g) = \vec{X}_n^B(\vec{x}_{foot}) \approx \vec{\chi}_n^B(\mathring{\vec{x}}_g)$$

we have that

$$E_0^{n+1} = \left|\vec{X}_n^B(\vec{x}_{foot}) - \vec{\chi}_n^B(\mathring{\vec{x}}_g)\right| \leq \left|\vec{X}_n^B(\vec{x}_{foot}) - \vec{X}_n^B(\mathring{\vec{x}}_g)\right| + \left|\vec{X}_n^B(\mathring{\vec{x}}_g) - \vec{\chi}_n^B(\mathring{\vec{x}}_g)\right| \tag{3.22}$$

Assuming that $\vec{X} \in C^1$, the first term in (3.22) can be bounded by some $c_0 h^4$ since $|\vec{x}_{foot} - \mathring{\vec{x}}_g| \propto h^4$. Lemma 3.3.1 allows us to bound the second term by $c_1 h^4 + \sum_{i=0}^{3} E_i^n h^i$. Hence, we set the upper bound on the error as:

$$E_0^{n+1} \leq ch^4 + \sum_{i=0}^{3} E_i^n h^i \tag{3.23}$$

The approximation to $\nabla \vec{X}_{n+1}^B(\vec{x}_g)$ is obtained by evaluating the inverse of $\nabla \vec{\chi}_{n+1}^F$ at $\vec{\chi}_{n+1}^B(\vec{x}_g)$. Assuming that numerical error from inverting $\nabla \vec{\chi}^F$ is negligible, we use the error bound for $\nabla \vec{\chi}_{n+1}^F$ at $\vec{\chi}_{n+1}^B(\vec{x}_g)$ as E_1 bound for the backward map.

$$\begin{aligned} E_1^n &= \left|\nabla \vec{X}_n^F(\vec{X}_n^B(\vec{x}_g)) - \nabla \vec{\chi}_n^F(\vec{\chi}_n^B(\vec{x}_g))\right| \\ &\leq \left|\nabla \vec{X}_n^F(\vec{X}_n^B(\vec{x}_g)) - \nabla \vec{X}_n^F(\vec{\chi}_n^B(\vec{x}_g))\right| + \left|\nabla \vec{X}_n^F(\vec{\chi}_n^B(\vec{x}_g)) - \nabla \vec{\chi}_n^F(\vec{\chi}_n^B(\vec{x}_g))\right| \\ &\leq d_0 \left|\vec{X}_n^B(\vec{x}_g) - \vec{\chi}_n^B(\vec{x}_g)\right| + e_1^n\left(\vec{\chi}_n^B(\vec{x}_g)\right) \end{aligned}$$

We had $\left|\vec{X}_n^B(\vec{x}_g) - \vec{\chi}_n^B(\vec{x}_g)\right| \leq E_0^n$. Also, e_1^n is the error on the gradient of the forward map at off-grid points. This is order 3 accurate by Theorem 3.3.2.

We get that $E_1^n \leq d_0 E_0^n + d_1 h^3$, where d_0 is the Lipschitz constant for $\nabla \vec{X}^F$ and d_1 is the constant from the Hermite cubic interpolation in the forward map. E_2^n and E_3^n are obtained by cell-based finite difference on E_1^n so $E_2^n = E_1^n/h + O(h^2)$ and $E_3^n = E_1^n/h^2 + O(h^2)$.

From (3.23), we have the following rule for the evolution of E_0 with n:

$$E_0^{n+1} = ch^4 + \sum_{i=0}^{3} E_i^n h^i \leq ch^4 + E_0^n + 3hE_1^n = (1 + 3d_0h)E_0^n + (c + 3d_1)h^4 \quad (3.24)$$

We can convert (3.24) into the following:

$$E_0^{n+1} \leq (1 + 3d_0h)^n E_0^1 + (c + 3d_1)h^4 \frac{(1 + 3d_0h)^n - 1}{3d_0h}$$

Taking $n = t/h$, we get the GTE result for the backward map:

$$GTE(t) \leq (1 + 3d_0h)^{t/h} E_0^1 + (c + 3d_1)h^3 \frac{(1 + 3d_0h)^{t/h} - 1}{3d_0}$$
$$\leq e^{3d_0t} E_0^1 + \frac{e^{3d_0t} - 1}{3d_0} h^3(c + 3d_1)$$

where E_0^1 comes from one step of $RK3$ which has error $O(h^4)$.

□

CHAPTER 4
Application to Surface Evolution and Numerical Results

In this chapter, we will apply the backward and forward mapping methods to surface evolution and show some numerical results.

The backward map is used as an Eulerian method tracking the surfaces through the advected level set function. The forward map is a Lagrangian method that represents the surface by tracing the trajectories of sample points. Recall the two ways we used to define the time dependent moving surface $S(t)$ in Chapter 3:

Given the initial surface $S_0 \subset \mathbb{R}^d$, we let $S(t) = \vec{X}(S_0, 0, t) = \vec{X}^F(S_0, t)$ for the Lagrangian approach, and let $S(t) = \{\vec{x} \in \mathbb{R}^d | \vec{X}(\vec{x}, t, 0) = \vec{X}^B(\vec{x}, t) \in S_0\}$ for the Eulerian approach. If $\vec{X}(\cdot, 0, t)$ is a diffeomorphism then $\vec{X}^F(\cdot, t) = \left(\vec{X}^B\right)^{-1}(\cdot, t)$ and the two definitions are equivalent.

We will use the backward map for the Eulerian definition of $S(t)$. This is done by defining a function $\Phi_0 : \mathbb{R}^d \to \mathbb{R}$ such that S_0 is exactly the zero level set of Φ_0. The criteria for a good choice of level set function are explained in the preliminaries chapter.

In relation to the backward map, we have the following equivalences:

$$\vec{x} \in S(t) \iff \vec{X}^B(\vec{x}, t) \in S_0 \iff \Phi_0(\vec{X}^B(\vec{x}, t)) = 0 \iff \Phi(\vec{x}, t) = 0 \qquad (4.1)$$

Hence, for the level set advection approach, we let $S(t) = \{\vec{x} \in \mathbb{R}^d | \Phi(\vec{x}, t) = 0\}$.

For the Lagrangian approach, we will use a parametric definition of S_0. Let $\vec{P}_0 : \Omega \to S_0 \subset \mathbb{R}^d$ be a parametrization of the surface. Ω is the $(d-1)$-dimensional parametric space, usually $[0,1]^{d-1}$. We then parametrize $S(t)$ by the composition of the forward map with $\vec{P}_0$:

$$\begin{aligned} &\vec{P}(\cdot,t) : \Omega \to S(t) \subset \mathbb{R}^d \\ &\vec{P}(\cdot,t) = \vec{X}^F(\vec{P}_0(\cdot),t) \end{aligned} \tag{4.2}$$

The Jacobian of both the backward and forward maps provide useful information on the surface being evolved. The gradient of the level set function $\Phi(\vec{x},t)$ is normal to the surface at $\vec{x} \in S(t)$. Hence, we have that the normal of the surface $S(t)$ at any time is given by $\nabla\Phi(\vec{x},t) = \nabla\Phi_0 \cdot \nabla\vec{X}^B$. Other quantities such as curvature can also be obtained by analytic formulæ explained in [8].

For the forward map, $\nabla\vec{X}^F$ gives the pushforward of the tangent space $T_{\vec{x}}S_0$ to $T_{\vec{X}(\vec{x},t)}S(t)$. This allows us to easily relate the first fundamental form of S_0 with that of $S(t)$.

The two methods each have their drawbacks. For the level set method, one main disadvantage is the difficulty of finding appropriate level set functions Φ_0 for certain types of surfaces. When choosing level set functions, we prefer functions that are smooth and change signs when crossing the surface. This facilitates the process of solving for the zero contour. For non-orientable surfaces such as the Möbius strip, it is impossible to meet both criteria since there is no continuous choice of normal vector along the strip. Furthermore, even if we start with a good level set function

with $|\nabla\Phi(\vec{x},0)| = 1$ on its zero level set, there is no guarantee that after advection to time t, the function $\Phi(\vec{x},t)$ will maintain this property.

For the Lagrangian method with the forward map, the main problems are topology changes and particle distribution. Unlike in the level set method where splitting or merging of surfaces are naturally represented by the level set function, we need to actively detect changes in topology in the forward map method. Particle management is also an issue since initially well distributed sample points can be arbitrarily clustered or scattered after some time. We will look into this in more details in the next section.

In some cases, it might be useful to compute both maps at the same time. Suppose we want to simulate the movement of an elastic sphere and a Möbius strip embedded in the same ambient space and under the influence of the same velocity field. We can use an implicit definition for the sphere, and an explicit one for the Möbius trip. Then at every time step, the image of the sphere at time t can be reconstructed using the composition of the level set function with the backward map. The Möbius strip can be obtained by composing the forward map with the parametrization of the initial strip. Simulating these surfaces simultaneously is easy since both the forward and backward maps are stored at the same times and on the same grid, the two maps are updated at the same time.

We now present some numerical result we obtained using these methods. In the 3D simulations, we used the following velocity field $\vec{v} = (u, v, w)$ in the flat 3-torus encoded as $[0, 1]^3$ with periodic boundary conditions. Throughout the simulation, we

assumed that this velocity is provided analytically at all times, hence we also have access to the exact derivatives used to integrate the gradients of the maps.

$$
\begin{aligned}
u(x,y,z,t) &= 2\cos\left(\frac{\pi t}{2}\right)\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z) \\
v(x,y,z,t) &= -\cos\left(\frac{\pi t}{2}\right)\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z) \\
w(x,y,z,t) &= -\cos\left(\frac{\pi t}{2}\right)\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)
\end{aligned}
\tag{4.3}
$$

We carried out a few tests using this velocity field: figures 4–1 and 4–2 show the results of simulations for surface evolution. Figures 4–3 to 4–5 show the results of the convergence tests for the CM methods.

In the experiments, we simulated the evolution of a sphere and a Möbius strip. We chose a sphere of radius 0.1 centered at $(0.35, 0.35, 0.35)$ and a Möbius strip of radius 0.1 and width 0.05 centered at $(0.65, 0.65, 0.65)$. The sphere is defined as the zero level set of $\Phi_0(\vec{x}) = |\vec{x} - \vec{c}_s| - r$, where r is the radius and $\vec{c}_s$ is the center. The parametrization of the strip is given by:

$$
\begin{aligned}
\vec{X}(u,v) &= [R + u\cos(\frac{1}{2}v)]\cos(v) + c_x \\
\vec{Y}(u,v) &= u\sin(\frac{1}{2}v) + c_y \\
\vec{Z}(u,v) &= [R + u\cos(\frac{1}{2}v)]\sin(v) + c_z
\end{aligned}
\tag{4.4}
$$

for $u \in [-w/2, w/2]$ (w is width) and $v \in [0, 2\pi]$. $c_x = c_y = c_z = 0.65$ and $R = 0.1$.

The forward and the backward maps are updated simultaneously at every time step. We compose the backward map with Φ_0 for the sphere to obtain the advected level set function for some time $t \in [0, 2]$. We also compose the parametrization of

the initial Möbius strip with the forward map to obtain a parametrization of the evolved Möbius strip.

Figure 4–1 shows our results. For these tests, we used the remapping method proposed in [7]. The computations are carried out on a coarse grid of 32^3, the maps are regularly remapped on a fine grid of 128^3 after which the coarse grid computations are reinitialized.

We also compared the original backward maps with the modified method which uses the analytic inverse of the Jacobian of the forward map, see figure 4–2. The same sphere as above is used, however the maps are computed on a fixed 64^3 grid. The level set function is still defined on a 128^3 grid, and extra grid point values are computed using the Hermite Cubic interpolation.

To test the accuracy of the CM method in the whole domain, we ran convergence tests for the maps at various times between 0 and 2. The results are shown in figures 4–3 to 4–5. We carried out the tests as follows: First, we randomly select N points $\vec{x}_k \in [0,1]^3$. We also have m times $t_1, \ldots, t_m$ when we wish to test the maps. For each of these points and for each time t_i, we approximate the exact value of $\vec{X}^F(\vec{x}_k, t_i) = \vec{X}(\vec{x}_k, 0, t_i)$ and $\vec{X}^B(\vec{x}_k, t_i) = \vec{X}(\vec{x}_k, t_i, 0)$ by integrating the following ODEs from $t = 0$ to t_i.

$$\begin{aligned} \frac{d}{d\tau}\vec{y}(\tau) &= \vec{v}(\vec{y}(\tau), \tau) \quad \vec{y}(0) = \vec{x}_k \\ \frac{d}{d\tau}\vec{z}(\tau) &= -\vec{v}(\vec{z}(\tau), t_i - \tau) \quad \vec{z}(0) = \vec{x}_k \end{aligned} \tag{4.5}$$

We have that $\vec{y}(t_i) = \vec{X}^F(\vec{x}_k, t_i)$ and $\vec{z}(t_i) = \vec{X}^B(\vec{x}_k, t_i)$. These ODEs are solved using RK4 with a time step much smaller than the one used for the characteristics

maps. The error from integrating the ODEs should be negligible compared to the characteristics map errors. We used these as "exact solutions" to generate convergence plots for the forward and backward maps. Figures 4–3 to 4–5b show our results using 10000 uniformly randomly distributed test points at 20 equally spaced times in $[0, 2]$.

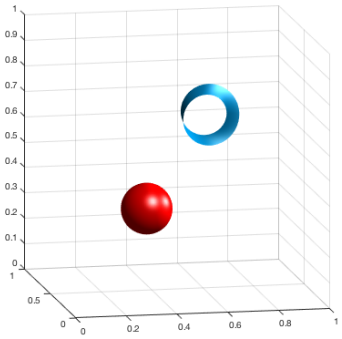
We can see from figure 4–5a that the backward map is asymptotically 3^{rd} order accurate and the forward map is 4^{th} order. It is interesting to notice that the error of the forward map is highest at $t = 1$, the middle of the simulation where distortions are the biggest. However, this error decreases afterwards. One hypothesis consistent with this observation is that the error is mainly due to the interpolation. Indeed, the grid point values of the forward map at various time steps do not rely on interpolation. They are simply solutions of the characteristic equations integrated in time using Runge-Kutta methods. Furthermore, the velocity we used are very well-behaved and a RK4 time integration for individual particle paths should be very precise. The error at grid points should be small but increasing almost monotonically with time. The off-grid points however depend on the interpolation. The error from interpolation can grow as the interpolated function becomes less well-behaved. Indeed, even though the velocity field is simple, the characteristics maps may accumulate large distortions after a long time. The Lipschitz constants associated with the function change with time and can affect the precision of the interpolant.

Figures 4–4 and 4–5b are error plots for the backward map using analytic inverse of the Jacobian of the forward map. Although we provided a proof that this method should have 3^{rd} order convergence, figure 4–5b suggests that there are problems

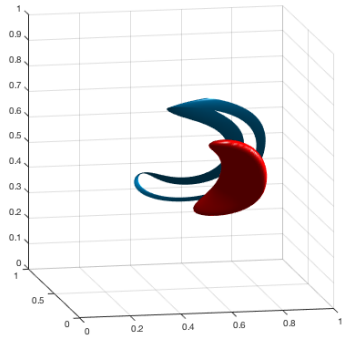
in the 200^3 grid test. The reason why this happened isn't clear. It could be due to an error in the code. Another possibility is that additional error was introduced when inverting the Jacobian of the forward map. The inverse was computed using an analytic formula which wasn't numerically optimal. If the Jacobian is ill-conditioned, the matrix inversion can produce unexpected errors.

Furthermore, the limited number of test points might not capture the worst case scenario in every test. In other words, the accuracy of the method for the coarse grid cases could have been misrepresented. The actual maximum error in those cases might be much higher and in line with that of the 200^3 grid test.

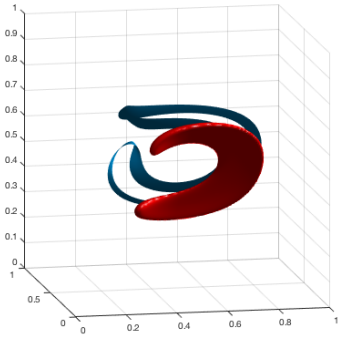
We would need to carry out further tests and corrections on this method to evaluate its performance. Indeed, even though there seems to be an increase in error as we shrink the grid size, the absolute error is still smaller than that of the original CM method. Furthermore, we can look at the convergence plot of the averaged error (average taken over all test points). This can be viewed as an L_1 error with the integral carried out by Monte-Carlo integration using 10000 uniformly random samples. We see in figure 4–5d that the modified method has better absolute accuracy and exhibits the asymptotic order 3 earlier than the original. These tests offer some evidence that the method is 3^{rd} order convergent at least in L^1 norm. However, this is merely an indication that the method could be made to work; further research is required to analyze this method.



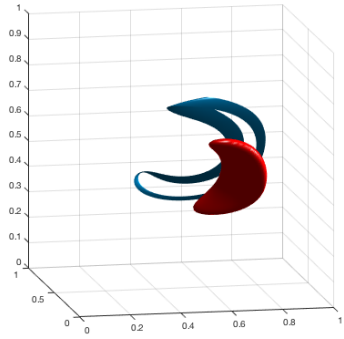
(a) $t = 0$



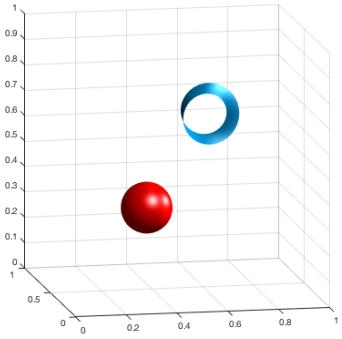
(b) $t = 0.5$



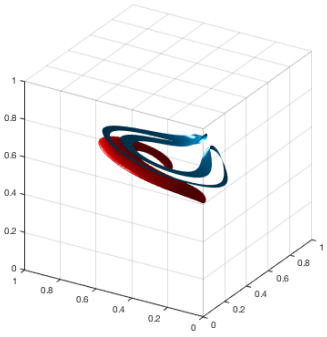
(c) $t = 1$



(d) $t = 1.5$

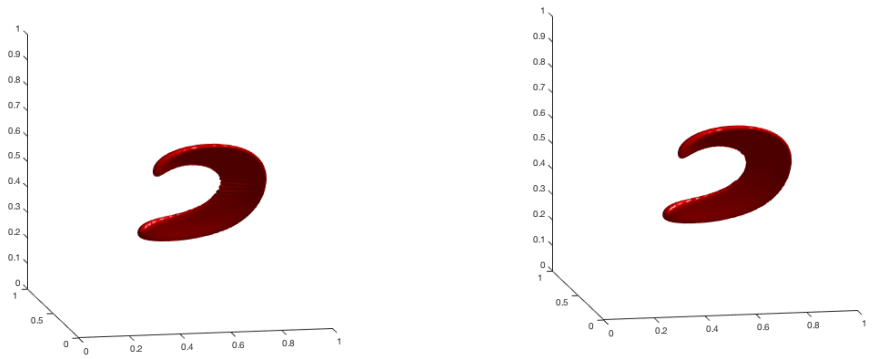


(e) $t = 2$



(f) Another view of $t = 1$

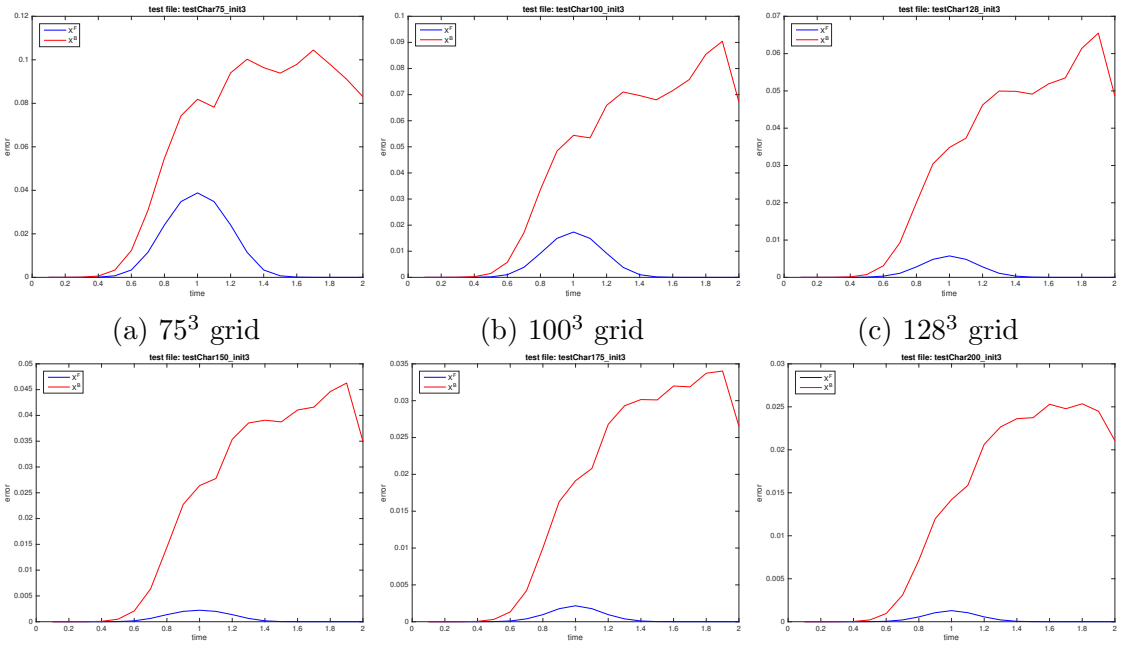
Figure 4–1: Surfaces evolved simultaneously using the forward and backward maps. Backward map for implicit definition and forward map for explicit definition.



(a) CM, $t = 1$

(b) CM with inverse forward Jacobian, $t = 1$

Figure 4–2: Surface Evolution using the Inverted Jacobian CM



(a) 75^3 grid

(b) 100^3 grid

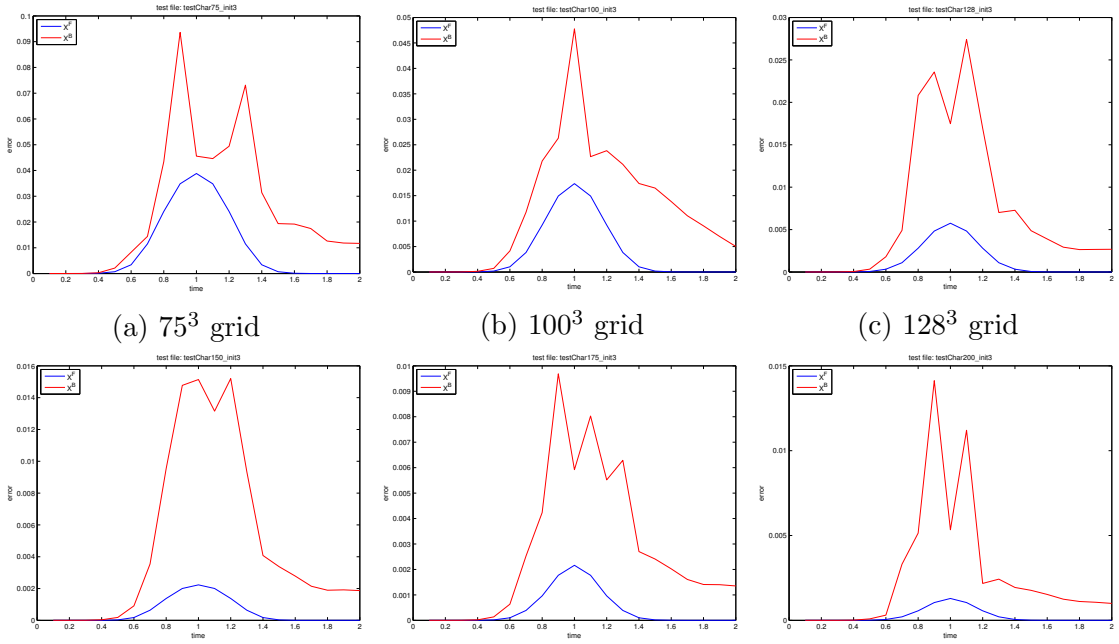
(c) 128^3 grid

(d) 150^3 grid

(e) 175^3 grid

(f) 200^3 grid

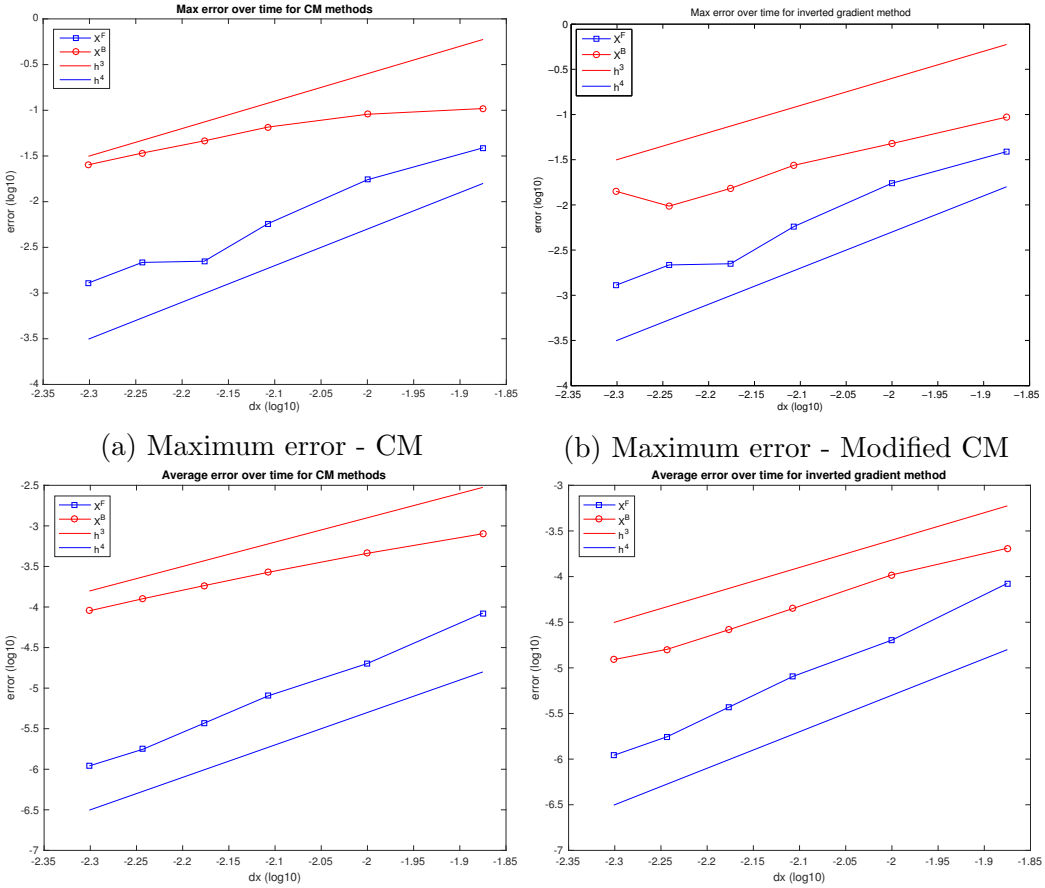
Figure 4–3: Original Gradient Augmented Characteristics Mapping Method: Evolution over time of the error in the forward (blue) and backward (red) maps for various grid sizes.



(a) 75^3 grid (b) 100^3 grid (c) 128^3 grid

(d) 150^3 grid (e) 175^3 grid (f) 200^3 grid

Figure 4–4: Modified Gradient Augmented Characteristics Mapping Method: Evolution over time of the error in the forward (blue) and backward (red) maps for various grid sizes.



(a) Maximum error - CM

(b) Maximum error - Modified CM

(c) Average error - CM

(d) Average error - Modified CM

Figure 4–5: Convergence plot for the forward and backward maps on Log_{10} scale using maximum and averaged error over test points

CHAPTER 5
Mesh Management

5.1 Introduction and Motivation

The forward characteristics mapping method, like most Lagrangian methods, has the advantage of being able to track more general surfaces. However since the surface is tracked using marker points, the distribution of these points on the surface needs to be controlled. The goal of this section is to study numerical methods for controlling the discrete representation of the moving surfaces optimally throughout its evolution.

We will work under the following setup:

Denote by $S(t) \subset \mathbb{R}^d$ the surface of interest at some time t. We assume that the initial surface is given by some parametrization $\vec{P}_0 : \Omega \to S(0) \subset \mathbb{R}^d$ where Ω is the 2 dimensional parametric space. Under the framework of the Lagrangian forward characteristics map, $S(t)$ for $t > 0$ is given by the push-forward of $S(0)$ by the forward characteristics map $\vec{X}^F$. Recall the forward map is defined as follows:

$$\partial_t \vec{X}^F = \vec{v}(\vec{X}^F, t)$$
$$\vec{X}^F(\vec{x}, 0) = \vec{x}$$

Then, we can write $S(t) = \vec{X}^F(S(0), t)$. This naturally gives us a parametrization of the surface $S(t)$:

$$\vec{P}(\cdot, t) : \Omega \to S(t) \subset \mathbb{R}^d \quad \text{where} \quad \vec{P}(\cdot, t) = \vec{X}^F(\vec{P}_0(\cdot), t) \tag{5.1}$$

We assume that at any time, the surface is tessellated using a mesh on Ω. That is, let the mesh $\{A_i\}$ be a partition of Ω, then $S(t) = \cup_i \vec{P}(A_i, t)$. Usually, A_i's are taken to be triangles or squares. We can say that $\{A_i\}$ is an equiareal mesh for $S(t)$ if the area of $\vec{P}(A_i, t)$ is the same for all i. This is called the equidistribution or equiareal principle.

A more thorough formulation of the problem can be found in the book *Adaptive Moving Mesh Methods* by Huang and Russell [6]. Here, we will summarize a few main concepts we need for designing our method in sections 5.2 and 5.3.

The equidistribution condition can be formulated as follows:

Let $M(\cdot, t)$ be the first fundamental form associated with the parametrization $\vec{P}(\cdot, t)$. Then the area of $\vec{P}(A_i, t)$ is given by:

$$\alpha_i(t) = \int_{A_i} \sqrt{det(M)} d\vec{x}$$

Finding an equidistributing mesh then boils down to finding a mesh $\{A_i\}$ such that α_i is the same for all i at a given time t. This would yield equal area cells on the surface $S(t)$. A stronger additional condition would be to require that the A_i's be equilateral with respect to the metric M. This would yield $P(A_i, t)$'s that are equilateral on $S(t)$.

In general, if the geometry of the surface is not the only concern, we will work with a chosen metric tensor M called a monitor function. An M-uniform mesh is a mesh $\{A_i\}$ which satisfies the equidistribution and equilateral properties with respect to the metric M. The goal would be to find a mesh that is nearly M-uniform.

There are three main families of methods for adapting the mesh: mesh subdivision or refinement, p-adaptivity for finite elements and moving mesh methods.

Mesh refinement mainly focuses on equidistribution. The general idea is to subdivide a cell A_i into smaller cells whenever α_i is above some threshold. For instance, a 2 dimensional region can be subdivided using a quadtree implementation [11]. These methods have the advantage of being fast and robust but tend to be sub-optimal in the equidistribution sense. Indeed, although the result will have all α_i below the fixed threshold, the variations occurring from the subdivisions can be quite large. This can lead to a larger than necessary number of cells which is computationally wasteful.

P-adaptivity methods for finite elements use the monitor function to choose basis functions of the appropriate order for each element. Basis polynomials of higher order are used in elements requiring higher precision. The mesh itself is not modified.

The methods we will examine in further detail here are the moving mesh methods. These methods consist generally of obtaining the position and shape of the mesh elements as the solution of some mesh PDE. Possible mesh PDEs include Monge-Ampère equations [3] or variational formulations of minimization problems of given adaptation functionals [12]. A number of adaptive moving mesh methods are presented and analyzed in [6].

5.2 Theoretical Construction of the Mesh Transformation

In this part we will assume that the surface has evolved to some fixed time τ. We then consider its associated monitor function M which can be taken as the first fundamental form of $S(\tau)$ at time τ. We pause the time evolution of the surface and construct a mesh transformation which adapts the initial mesh to M. The method will use a pseudo-time evolution which can also be interpreted as an iterative process. The variable t used in this section refers to an imaginary pseudo-time and not to the time of the evolution of the surface. Accordingly, the notations $\vec{X}^F$ and $\vec{X}^B$ for characteristics maps will refer to the maps on the parametric space Ω generated by the pseudo-time evolution/iteration.

For simplicity, we can assume that the domain Ω is simple (square, rectangle, flat torus). For general domains, we will need a simple computational space Ω_c that is mapped to Ω by some transformation. The method still applies, we simply take the pull-back of M to Ω_c where we compute a mesh. The mesh will then be mapped to Ω using the given transformation.

One possible approach to adapt the mesh is to construct a change of coordinate map from Ω to Ω. We start with a simple, potentially uniform mesh $\{R_i\}$ on Ω and we construct a map $T : \Omega \to \Omega$ such that the transformed mesh $\{A_i\}$, where $A_i = T(R_i)$, are equidistributed. That is, for a mesh consisting of N cells, we want

$$\int_{A_i} \rho_M d\vec{x} = \frac{1}{N} \int_{\Omega} \rho_M d\vec{x} \tag{5.2}$$

where $\rho_M = \sqrt{det(M)}$ is called the mesh density. More generally, the mesh density needs not to be defined pointwise, we can simply work with a given measure μ_M and construct A_i such that $\mu_M(A_i)$ are the same for all i.

The method we designed in this paper is inspired from the optimal transport problem. See for instance [2] and [4] for application of optimal transport to mesh adaptivity. We refered to *Optimal Transport, Old and New* by Villani [10] for the following definition on transport maps.

Definition 5.2.1. Let (X, Σ_X, μ_X) and (Y, Σ_Y, μ_Y) be two probability spaces. A measurable function $f : X \to Y$ is a transport map if

$$\int_Y \phi(y) d\mu_Y = \int_X \phi(f(x)) d\mu_X \quad \forall \phi \text{ integrable w.r.t. } \mu_Y \tag{5.3}$$

We can check that this is equivalent to

$$\mu_Y(R) = \mu_X(f^{-1}(R)) \quad \forall R \text{ measurable w.r.t. } \Sigma_Y \tag{5.4}$$

The main difficulty is to find a transport map that moves μ_X to μ_Y. An approach to finding such a map using a fluid dynamics intuition of the L^2 Monge-Kantorovich problem was given by Benamou and Brenier [1]. Their setup is as follows:

Given densities ρ_0 and ρ_M. For any pair $\rho(\vec{x}, t)$ and $\vec{v}(\vec{x}, t)$ satisfying theses initial conditions and conservation laws:

$$\begin{aligned} &\rho(\vec{x}, 0) = \rho_0(\vec{x}) \\ &\rho(\vec{x}, 1) = \rho_M(\vec{x}) \\ &\partial_t \rho + \nabla \cdot (\rho \vec{v}) = 0 \end{aligned} \tag{5.5}$$

we have that the transport map moving ρ_0 to ρ_M is given by the forward characteristics map generated by $\vec{v}$ at time $t = 1$. More specifically, let $\vec{X}^F(\vec{x}, t)$ be such that $\vec{X}^F(\vec{x}, 0) = \vec{x}$ and $\partial_t \vec{X}^F = \vec{v}(\vec{X}, t)$, then $\vec{X}^F(\vec{x}, 1)$ is a transport map moving ρ_0 to ρ_M.

The method we propose here is based on this framework. Let μ_0 and μ_M be the measures induced by the densities ρ_0 and ρ_M respectively.

Firstly, we invert the roles of μ_0 and μ_M and use the backward map instead of the forward map to move μ_0 to μ_M. Indeed, by switching μ_0 with μ_M, the forward maps gives us a transport map from μ_M to μ_0. Then, from equation (5.4), we have:

$$\mu_0(R_i) = \mu_M \left((\vec{X}^F)^{-1}(R_i)\right) = \mu_M(\vec{X}^B(R_i)) = \mu_M(A_i) \tag{5.6}$$

We will use the backward map $\vec{X}^B$ as the mesh transformation. This allows us to directly choose the value of $\mu_M(A_i)$ by defining the appropriate measure for $\mu_0(R_i)$ on the corresponding initial cell. If we want μ_M to be equidistributed on A_i, it suffices to pick $\mu_0(R_i)$ to be the same constant for every i. That is, equidistributing a density ρ_M can be done by finding a transport map moving a uniform density ρ_0 to ρ_M.

Secondly, we do not optimize the transport map. We will not solve for the optimal velocity field, instead we choose $\vec{v}$ to be the velocity field associated with the heat equation by Fick's law of diffusion. With this velocity, we will evolve the characteristics map until a large time until the density is close enough to the targeted equilibrium.

These two choices make the algorithm simple and easy to implement. The use of the backward map allows us to compute velocities only at grid points without need for any extra interpolation. The choice of velocity eliminates several costly steps at the expense of optimality but still maintains some good convergence properties.

Assuming for now that we have measure density functions ρ_0 and ρ_M corresponding to μ_0 and μ_M, we define the velocity field to be

$$\vec{v}(\vec{x}, t) := -\alpha \frac{\nabla(\rho - \rho_0)}{\rho} \tag{5.7}$$

In the setup of (5.5), our initial conditions and conservation law read:

$$\rho(\vec{x}, 0) = \rho_M(\vec{x}) \tag{5.8}$$

$$\partial_t \rho = \operatorname{div}(\alpha \nabla(\rho - \rho_0)) \tag{5.9}$$

which is a heat equation with ρ_M as initial condition and $\alpha(\vec{x}, t)$ as diffusion coefficient. We impose that ρ has the same normal derivative at the boundaries as ρ_0. This implies that ρ_0 will be the equilibrium solution.

The backward characteristics map is evolved using the following equations:

$$\vec{X}^B(\vec{x}, 0) = \vec{x} \tag{5.10}$$

$$\partial_t \vec{X}^B + \nabla \vec{X}^B \cdot \vec{v} = 0 \tag{5.11}$$

We compute the map until a large time t_f. Since the forward map $\vec{X}^F(\cdot, t_f)$ moves $\rho_M(\vec{x})$ to $\rho(\vec{x}, t_f)$, $\vec{X}^B(\vec{x}, t_f)$ moves $\rho(\vec{x}, t_f)$ to $\rho_M(\vec{x})$, and $\rho(\vec{x}, t_f)$ approximates the equilibrium ρ_0 for large enough t_f. We will analyze this in detail later.

Here we provide a summary of a semi-discretization of the method using Euler steps in time. $\vec{\chi}^B$ denotes the numerical approximation of $\vec{X}^B$, and u, the approximation of ρ where $\rho(x,t) = \rho(\vec{X}^B, 0)\det(\nabla \vec{X}^B)$ by semi-Lagrangian form of conservation of mass (change of variables). Using this definition, we will need that ρ is strictly positive at all times. This is not guaranteed for heat equation with constant diffusion coefficient. Indeed, if the source term $-\Delta\rho_0$ is non-positive, i.e. ρ_0 not concave, then the solution $\rho(\vec{x},t)$ can be less or equal to 0 even though the initial and equilibrium states are both strictly positive. Hence, we introduced a diffusion coefficient $\alpha(\vec{x},t)$. We want to pick α to be 1 if $\rho >> 0$ and we choose $\alpha \in (0,1)$ accordingly if $\rho(\vec{x},t)$ is close to 0 so that $\rho(\vec{x}, t+\Delta t) > 0$. Its purpose is essentially to slow down or pause the diffusion when u is close to 0. We'll see later that choosing $\alpha(\vec{x},t) = c\rho(\vec{x},t)$ is sufficient, however, this is not always necessary.

$$Algorithm: \tag{5.12}$$

1. Initialize $u(\vec{x},0) = \rho_M(\vec{x})$, $\vec{\chi}^B(\vec{x},0) = \vec{x}$.
2. At time t, update $u(\vec{x},t) = \rho_M(\vec{\chi}^B(\vec{x},t))\det\left(\nabla\vec{\chi}^B(\vec{x},t)\right)$
3. Compute footpoint
$$\mathring{\vec{x}} = \vec{x} + \alpha\frac{\nabla(u-\rho_0)}{u}(\vec{x},t)\Delta t$$
choosing $\alpha(\vec{x},t)$ such that $\rho_M(\vec{\chi}^B(\mathring{\vec{x}},t))\det\left(\nabla\vec{\chi}^B(\mathring{\vec{x}},t)\right) > 0$. Also, enforce the boundary condition $\mathring{\vec{x}} \in \partial\Omega$ for all $\vec{x} \in \partial\Omega$.
4. Evolve characteristics map $\vec{\chi}^B(\vec{x}, t+\Delta t) = \vec{\chi}^B(\mathring{\vec{x}}, t)$
5. Advance to time $t+\Delta t$. Repeat steps 2 through 4 until time t_f when u is a "good enough" approximation of ρ_0.

The issue of "good enough" stopping time will be addressed later on.

In the following lemma, we make a brute force Taylor expansion of $u(\vec{x}, t+\Delta t)$ which allows us to describe the evolution of u and its asymptotic behaviors.

Lemma 5.2.2. *The above Euler time stepping semi-discretization gives us that* $u(\vec{x}, t+\Delta t) = u(\vec{x}, t) + \operatorname{div}(\alpha\nabla(u-\rho_0))\Delta t + O(\Delta t^2)$

Proof.

$$\begin{aligned}
u(\vec{x}, t+\Delta t) &= \rho_M\left(\vec{\chi}^B(\vec{x}, t+\Delta t)\right)\det\left(\nabla_{\vec{x}}\vec{\chi}^B(\vec{x}, t+\Delta t)\right) \\
&= \rho_M\left(\vec{\chi}^B(\mathring{\vec{x}}, t)\right)\det\left(\nabla_{\vec{x}}\vec{\chi}^B\left(\vec{x} + \frac{\alpha\nabla(u-\rho_0)}{u}(\vec{x}, t)\Delta t, t\right)\right) \\
&= \rho_M\left(\vec{\chi}^B(\mathring{\vec{x}}, t)\right)\det\left(\nabla\vec{\chi}^B\Big|_{(\mathring{\vec{x}}, t)} \cdot \left(I + \nabla\frac{\alpha\nabla(u-\rho_0)}{u}\Delta t\right)\right) \\
&= \left[\rho_M\left(\vec{\chi}^B(\mathring{\vec{x}}, t)\right)\det\left(\nabla\vec{\chi}^B\left(\mathring{\vec{x}}, t\right)\right)\right]\det\left(I + \nabla\frac{\alpha\nabla(u-\rho_0)}{u}\Delta t\right) \\
&= u\left(\mathring{\vec{x}}, t\right)\det\left(I + \nabla\frac{\alpha\nabla(u-\rho_0)}{u}\Delta t\right) \qquad (5.13)
\end{aligned}$$

It suffices to define $\alpha(\vec{x})$ such that $\det\left(I + \nabla\frac{\alpha\nabla(u-\rho_0)}{u}\Delta t\right) > 0$.

We can estimate each factor in the previous expression:

$$u\left(\vec{x} + \frac{\alpha\nabla(u-\rho_0)}{u}\Delta t, t\right) = u(\vec{x}, t) + \frac{\alpha\nabla u\cdot\nabla(u-\rho_0)}{u}\Delta t + O(\Delta t^2) \qquad (5.14)$$

$$\begin{aligned}
&\det\left(I + \nabla\frac{\alpha\nabla(u-\rho_0)}{u}\Delta t\right) = 1 + tr\left(\nabla\frac{\alpha\nabla(u-\rho_0)}{u}\Delta t\right) + O(\Delta t^2) \qquad (5.15) \\
&= 1 + div\left(\frac{\alpha\nabla(u-\rho_0)}{u}\Delta t\right) + O(\Delta t^2) \\
&= 1 + \frac{\alpha u\nabla^2(u-\rho_0) - \alpha\nabla u\cdot\nabla(u-\rho_0) + u\nabla\alpha\cdot\nabla(u-\rho_0)}{u^2}\Delta t + O(\Delta t^2)
\end{aligned}$$

From this, we see that it suffices to pick α such that $1 + \operatorname{div}\left(\frac{\alpha\nabla(u-\rho_0)}{u}\Delta t\right) > 0$. One way to achieve this is to pick $\alpha = cu$ for some constant c depending on $\nabla^2(u-\rho_0)$.

Finally, multiplying (5.14) and (5.15), we get

$$\begin{aligned} u(\vec{x}, t+\Delta t) &= u(\vec{x}, t) + \alpha\nabla^2(u-\rho_0)\Delta t + \nabla\alpha\cdot\nabla(u-\rho_0)\Delta t + O(\Delta t^2) \\ &= u(\vec{x}, t) + \operatorname{div}(\alpha\nabla(u-\rho_0))\Delta t + O(\Delta t^2) \quad (5.16) \end{aligned}$$

□

Corollary 5.2.3. *Assuming that $\alpha(\vec{x}) \geq \alpha_{min} > 0$, the L^2 error $\|u-\rho_0\|_{L^2}$ decays exponentially in time.*

Proof. Consider the squared L^2 distance $E(t) = \int_\Omega |u(\vec{x}, t) - \rho_0(\vec{x})|^2 d\vec{x}$.

$$\begin{aligned} E(t+\Delta t) - E(t) &\leq \int_\Omega \left[u(\vec{x}, t+\Delta t) + u(\vec{x}, t) - 2\rho_0(\vec{x})\right]\left[u(\vec{x}, t+\Delta t) - u(\vec{x}, t)\right] d\vec{x} \\ &= \int_\Omega 2(u(\vec{x}, t) - \rho_0(\vec{x}))\operatorname{div}\left(\alpha\nabla(u(\vec{x}, t) - \rho_0(\vec{x}))\,\Delta t d\vec{x} + O(\Delta t^2)\right. \\ &= -2\Delta t\int_\Omega \alpha\left|\nabla(u(\vec{x}, t) - \rho_0(\vec{x}))\right|^2 d\vec{x} + O(\Delta t^2) \\ &\leq -2\alpha_{min}\|\nabla(u-\rho_0)\|_{L^2}^2\Delta t + O(\Delta t^2) \quad (5.17) \end{aligned}$$

we use the fact that the normal derivative of $(u-\rho_0)$ vanishes on the boundary by our choice of boundary condition to go from line 2 to line 3.

Since $\int_\Omega \rho_0 d\vec{x} = \int_\Omega \rho_M d\vec{x}$ and

$$\int_\Omega u(\vec{x}, t)d\vec{x} = \int_\Omega \rho_M(\vec{\chi}^B)\det(\nabla\vec{\chi}^B)d\vec{x} = \int_{\vec{\chi}^B(\Omega)} \rho_M d\vec{x} = \int_\Omega \rho_M d\vec{x}$$

we have that

$$\frac{1}{|\Omega|}\int_{\Omega}(u-\rho_0)d\vec{x}=0 \qquad \forall t\geq 0 \tag{5.18}$$

Hence, we can apply the Poincare-Wirtinger inequality: $\|u-\rho_0\|_{L^2}\leq C\|\nabla(u-\rho_0)\|_{L^2}$, which, combined with (5.17) gives us:

$$E(t+\Delta t)\leq(1-\alpha_{min}C\Delta t)E(t)+O(\Delta t^2) \tag{5.19}$$

□

This allows us to run the algorithm until a time t_f for which $\|u-\rho_0\|_{L^2}<\epsilon$. ϵ is some chosen threshold which should not be on the same order of magnitude as Δt^2. Otherwise, the convergence will be polluted by the $O(\Delta t^2)$ noise.

Theorem 5.2.4. *The algorithm* (5.12) *outputs a map T which moves ρ_0 to ρ_M. The L^2 error can be bounded by a selected ϵ as long as Δt is taken to be much smaller than $\sqrt{\epsilon}$.*

Proof. We set $T=\vec{\chi}^B(\cdot,t_f)$ for a t_f such that $\|u(\vec{x},t_f)-\rho_0(\vec{x})\|_{L^2}<\epsilon$. Replacing u with its definition, we have that

$$\begin{aligned}&\|\rho_M(\vec{\chi}^B(\vec{x},t_f))\det(\nabla\vec{\chi}^B(\vec{x},t_f))-\rho_0\|_{L^2}<\epsilon\\ \implies&\|\rho_M(T)\det(\nabla T)-\rho_0\|_{L^2}<\epsilon\end{aligned}$$

□

Then, for any measureable set R,

$$\begin{aligned}|\mu_0(R) - \mu_M(T(R))| &= \left| \int_R \rho_0(\vec{x})d\vec{x} - \int_{T(R)} \rho_M(\vec{x})d\vec{x} \right| \\ &\leq \int_R |\rho_0(\vec{x}) - \rho_M(T(\vec{x}))\det(\nabla T(\vec{x}))|\, d\vec{x} \\ &\leq \sqrt{|R|}\left(\int_R [\rho_0 - \rho_M(T)\det(\nabla T)]^2 d\vec{x} \right)^{1/2} < \epsilon\sqrt{|R|}\end{aligned}$$

Since $T(R_i) = A_i$, we can control $\mu_M(A_i)$ to be within $\epsilon\sqrt{|R_i|}$ of $\mu_0(R_i)$ which we can choose by defining an appropriate ρ_0.

The method has problems when the diffusion coefficient α is chosen to be very small at certain points. This happens when $u(\vec{x}, t)$ approaches 0. However, if the equilibrium density ρ_0 is concave ($\nabla^2\rho_0 \leq 0$), then by maximum principle, u remains strictly positive. For the purpose of generating an equiareal mesh, the equilibrium density is simply defined to be uniform in which case $u > 0$ for all t and we can choose $\alpha \equiv 1$.

5.3 Numerical Discretization

The spacial discretization of this method is less straightforward. Numerical tests seem to suggest that gridpoint based finite difference with analytically computed $\det(\nabla\vec{\chi}^B)$ yields unstable schemes. Instead, we avoid computing $\det(\nabla\vec{\chi}^B)$ by working directly with $\mu_M(\vec{\chi}^B(R_i, t))$. Since u in the algorithm is supposed to be a measure density of $\mu_M(\vec{\chi}^B(\cdot, t))$, this in principle should be the same as using the Jacobian of the backward map. However, it has the advantage of being tied directly

to the measures of the cells and avoids error from approximating the measure density ρ. The method can be thought of as an iterative process that terminates when the cell measures are within a threshold of the target value.

For simplicity, assume that Ω is a rectangular domain equipped with a uniform mesh of dimensions $\Delta x \times \Delta y$ (see figure 5–1 for grid organization). Let $R_{i,j}$ denote the cell on the i^{th} row and j^{th} column. Let the corners of $R_{i,j}$ be $x_{i,j}, x_{i+1,j}, x_{i,j+1}$ and $x_{i+1,j+1}$. Let $u_{i,j}$ be the average integral of $\rho_M(\vec{\chi}^B)\det(\nabla\vec{\chi}^B)$ on $R_{i,j}$. We also let $\rho(\vec{x},t) = \rho_M(\vec{\chi}^B(\vec{x},t))\det(\nabla\vec{\chi}^B(\vec{x},t))$ to facilitate the notation of integrals.

$$u_{i,j} = \frac{1}{|R_{i,j}|}\int_{R_{i,j}} \rho d\vec{x} = \frac{1}{|R_{i,j}|}\int_{\vec{\chi}^B(R_{i,j})} \rho_M d\vec{x} = \frac{1}{|R_{i,j}|}\mu_M(\vec{\chi}^B(R_{i,j})) \tag{5.20}$$

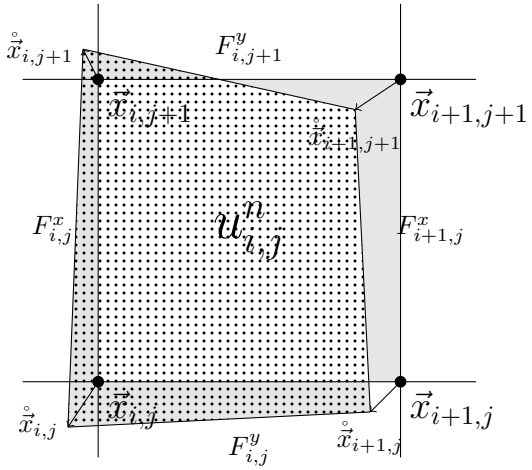


Figure 5–1: Positions of grid points, cell averages and discrete flux. The dotted area is the region where the average integral is $u^{n+1}_{i,j}$. The grey area corresponds to the numerical flux.

Since we use the backward characteristics mapping method, the velocities at time t only need to be defined on grid points. We would like to define these velocities such that the cell averages $u_{i,j}$ behave like a heat equation. This is equivalent to

enforcing the flux generated by $\vec{v}$ on the boundary between adjacent regions to be proportional to their difference. However it is difficult to solve for a velocity that satisfies the flux specifications. Instead, we look for a simpler definition of velocity field that satisfies the required flux up to some order of accuracy.

We can analyze the 1D case where Ω is an interval with grid points $\{x_i\}_{i=1}^N$ and grid size $\Delta x = h$. We use a periodic or Neumann boundary condition.

$$u_i = \frac{1}{h}\mu_M(\vec{\chi}^B([x_i, x_{i+1}])) = \frac{1}{h}\int_{\vec{\chi}^B(x_i)}^{\vec{\chi}^B(x_{i+1})} \rho_M dx = \frac{1}{h}\int_{x_i}^{x_{i+1}} \rho_M(\vec{\chi}^B)\frac{d\vec{\chi}^B}{dx}dx \tag{5.21}$$

Let q_i be the average integral of the residual:

$$q_i = \frac{1}{h}\int_{x_i}^{x_{i+1}} (\rho - \rho_0)d\vec{x} = \frac{1}{h}\left(u_i - \int_{x_i}^{x_{i+1}} \rho_0 d\vec{x}\right) \tag{5.22}$$

We define the velocity as follows:

$$\vec{v}_i = -\frac{2}{h}\frac{q_i - q_{i-1}}{u_i + u_{i-1}} \tag{5.23}$$

We also cap the velocity such that $\vec{v}\Delta t$ is less than $\frac{1}{3}$ of the cell length. This is the discrete equivalent of defining the diffusion coefficient $\alpha(\vec{x}, t)$ in the continuous version; doing so prevents the characteristic curves from crossing.

Lemma 5.3.1. *Using the velocity defined above, we have that*

$$u_i^{n+1} - u_i^n = \frac{\Delta t}{h^2}(q_{i+1}^n - 2q_i^n + q_{i-1}^n) + O(h\Delta t) \tag{5.24}$$

Proof. Let F_i denote the discrete flux from cell $R_{i-1} = [x_{i-1}, x_i]$ to R_i from time step n to $n+1$. We have that $u_i^{n+1} - u_i^n = \frac{1}{h}(F_{i+1} - F_i)$. We can approximate the flux:

$$F_i = \int_{x_i}^{x_i - \vec{v}_i \Delta t} \rho d\vec{x} = \vec{v}_i \rho(\vec{x}_i, t)\Delta t + O(\Delta t^2) \tag{5.25}$$

The approximate flux $-\vec{v}_i\rho(\vec{x}_i)$ is essentially $q_i - q_{i-1}$. In order to make this claim, we need to control the difference between $\rho(\vec{x}_i)$ and $\frac{1}{2}(u_i + u_{i-1})$.

$$\frac{u_i + u_{i-1}}{2} - \rho(\vec{x}_i) = \frac{1}{2h}\left(\int_{x_{i-1}}^{x_{i+1}} \rho(\vec{x})d\vec{x} - 2h\rho(\vec{x}_i)\right) = O(h^2) \tag{5.26}$$

Since the error of the midpoint rule for the integral approximation is $O(h^3)$, the above error is of order h^2.

Hence, we get an order $O(h^2\Delta t)$ approximation to the flux:

$$\begin{aligned} -\vec{v}_i\rho(\vec{x}_i)\Delta t &= \frac{2}{h}\frac{q_i - q_{i-1}}{u_i + u_{i-1}}\rho(\vec{x}_i) = \frac{q_i - q_{i-1}}{h}\Delta t(1 + O(h^2)) \\ &= \frac{\Delta t}{h}(q_i - q_{i-1}) + O(h^2\Delta t) \end{aligned} \tag{5.27}$$

Since for stability of the scheme, we will pick $\Delta t \propto h^2$, we have that $F_i = \frac{\Delta t}{h}(q_i - q_{i-1}) + O(h^2\Delta t)$. Therefore,

$$u_i^{n+1} - u_i^n = \frac{1}{h}(F_{i+1} - F_i) = \frac{\Delta t}{h^2}(q_{i+1}^n - 2q_i^n + q_{i-1}^n) + O(h\Delta t)$$

□

Lemma (5.3.1) shows that the choice of $\vec{v}_i$ in (5.23) results in the time evolution of u_i^n being consistent with the heat equation to order h.

We notice that by definition, $u_i^{n+1} - u_i^n = q_i^{n+1} - q_i^n$. Therefore, choosing $\Delta t = \frac{h^2}{2}$, we get that $q_i^{n+1} = \frac{1}{2}(q_{i+1}^n + q_{i-1}^n) + O(h^3)$. This scheme is stable, the $O(h^3)$ error

are negligible and won't affect stability. We can conclude that q_i^n will converge to a constant by using periodic or Neumann boundary conditions. Furthermore, since u^n and ρ_0, being probability distributions, both sum to 1, we have that q^n converges to 0. Hence, u^n converges to $\int_{x_i}^{x_{i+1}} \rho_0 d\vec{x}$.

Theorem 5.3.2. *This scheme is consistent with the heat equation in the 2-dimensional case.*

The proof of lemma (5.3.1) generalizes to 2 dimensions. We work on a grid with $\Delta x \times \Delta y$ cells. Δx and Δy are of order h and Δt is of order h^2. We choose velocity to be

$$\vec{v}_{i,j} = \frac{-(\nabla_h q)_{i,j}}{\frac{1}{4}(u_{i,j} + u_{i,j-1} + u_{i-1,j} + u_{i-1,j-1})} \tag{5.28}$$

where $q_{i,j}$ denotes $\frac{1}{|R_{i,j}|}\int_{R_{i,j}}(\rho - \rho_0)d\vec{x}$ and $(\nabla_h q)_{i,j}$ denotes the discrete finite difference gradient with grid size h.

We define x and y direction flux for each cell (see figure 5–1):

$$F_{i,j}^x := \int_{B_{i,j}^x} \rho(\vec{x}, t)\,\mathrm{sgn}(x - x_i)d\vec{x} \tag{5.29}$$

$$F_{i,j}^y := \int_{B_{i,j}^y} \rho(\vec{x}, t)\,\mathrm{sgn}(y - y_i)d\vec{x} \tag{5.30}$$

where $F_{i,j}^x$ approximates the x-direction flux from cell $R_{i-1,j}$ to $R_{i,j}$. $B_{i,j}^x$ denotes the quadrilateral region with corners $\vec{x}_{i,j}, \mathring{\vec{x}}_{i,j}, \vec{x}_{i,j+1}, \mathring{\vec{x}}_{i,j+1}$ where $\mathring{\vec{x}}_{i,j} = \vec{x}_{i,j} - \vec{v}_{i,j}\Delta t$. $B_{i,j}^y$ is the region bounded by $\vec{x}_{i,j}, \mathring{\vec{x}}_{i,j}, \vec{x}_{i+1,j}, \mathring{\vec{x}}_{i+1,j}$. The signum function determines for given $\vec{x}$ whether $\rho(\vec{x}, t)$ contributes to the flux going in or out of the cell. With these definitions, we have that

$$u_{i,j}^{n+1} = u_{i,j}^n + \frac{1}{|R_{i,j}|}(F_{i+1,j}^x - F_{i,j}^x + F_{i,j+1}^y - F_{i,j}^y) \tag{5.31}$$

One can check that with $F_{i,j}$ defined using the signum functions, this equality is exact.

For the x-direction flux, we will approximate the region $B^x_{i,j}$ with the trapezoid defined by $\vec{x}_{i,j}, \vec{x}_{i,j+1}, \vec{x}_{i,j} - v^x_{i,j}\Delta t, \vec{x}_{i,j+1} - v^x_{i,j+1}\Delta t$ where v^x denotes the x component of the velocity vector. This approximation is of order $O(\Delta y \Delta t^2)$. Hence,

$$\begin{aligned} F^x_{i,j} &= \int_{y_j}^{y_{j+1}} \int_{x_i}^{x_i - v^x(x_i,y)\Delta t} \rho(x,y,t)dxdy + O(\Delta y \Delta t^2) \\ &= \int_{y_j}^{y_{j+1}} -\rho(x_i,y,t)v^x(x_i,y)\Delta t + O(\Delta t^2)dy + O(\Delta y \Delta t^2) \\ &= -\frac{1}{2}[\rho(\vec{x}_{i,j},t)v^x_{i,j} + \rho(\vec{x}_{i,j+1},t)v^x_{i,j+1}]\Delta y \Delta t + O(\Delta y \Delta t^2) + O(\Delta y^3 \Delta t) \end{aligned} \tag{5.32}$$

where the last line is the trapezoidal approximation of the integral with $O(\Delta y^3)$ error. From the midpoint rule, we can approximate $\rho(\vec{x}_{i,j}, t)$ as

$$\rho(\vec{x}_{i,j},t) = \frac{1}{4}(u^n_{i-1,j-1} + u^n_{i,j-1} + u^n_{i-1,j} + u^n_{i,j}) + O(h^2) \tag{5.33}$$

Hence, we have that

$$\begin{aligned} F^x_{i,j} &= \frac{1}{2}[(\nabla_{h,x}q^n)_{i,j} + (\nabla_{h,x}q^n)_{i,j+1}]\Delta y \Delta t + O(h\Delta t^2) \\ F^y_{i,j} &= \frac{1}{2}[(\nabla_{h,y}q^n)_{i,j} + (\nabla_{h,y}q^n)_{i+1,j}]\Delta x \Delta t + O(h\Delta t^2) \end{aligned} \tag{5.34}$$

which gives us

$$
\begin{aligned}
u_{i,j}^{n+1} =& u_{i,j}^{n} + \frac{1}{|R_{i,j}|}(F_{i+1,j}^{x} - F_{i,j}^{x} + F_{i,j+1}^{y} - F_{i,j}^{y}) \qquad (5.35)\\
=& u_{i,j}^{n} + \Bigg[\frac{1}{2\Delta x}\big((\nabla_{h,x}q^{n})_{i+1,j} + (\nabla_{h,x}q^{n})_{i+1,j+1} - (\nabla_{h,x}q^{n})_{i,j} - (\nabla_{h,x}q^{n})_{i,j+1}\big)\\
&+\frac{1}{2\Delta y}\big((\nabla_{h,y}q^{n})_{i+1,j+1} + (\nabla_{h,y}q^{n})_{i,j+1} - (\nabla_{h,y}q^{n})_{i,j} - (\nabla_{h,y}q^{n})_{i+1,j}\big)\Bigg]\Delta t + O(h\Delta t)\\
=& u_{i,j}^{n} + (\nabla_{h}^{2}q^{n})_{i,j}\Delta t + O(h\Delta t)
\end{aligned}
$$

where $(\nabla_h^2 q^n)$ denotes the discrete Laplacian obtained by the above finite difference divergence of gradients.

In 2 dimensions, we need to be careful in choosing a definition for $\vec{v}$. We can use a straightforward 4 points stencil finite difference for $\nabla_h q$:

$$
v_{i,j}^{x} = -\frac{2}{\Delta x}\frac{q_{i,j} + q_{i,j-1} - q_{i-1,j} - q_{i-1,j-1}}{u_{i,j} + u_{i,j-1} + u_{i-1,j} + u_{i-1,j-1}} \qquad (5.36)
$$

However, this velocity results in a diagonal stencil for the Laplacian. The step $u_i^{n+1} - u_i^n$ has cell weights distributes as in figure 5–2. This discretized Laplacian can cause Red-Black (or checkerboard) instabilities. Indeed, in each time step, a cell only communicates with its diagonal neighbors; the system decouples into two regions like on a checkerboard. The red squares and the black squares are decoupled. Although solving the same equation, they may converge to a different constant away from the equilibrium state.

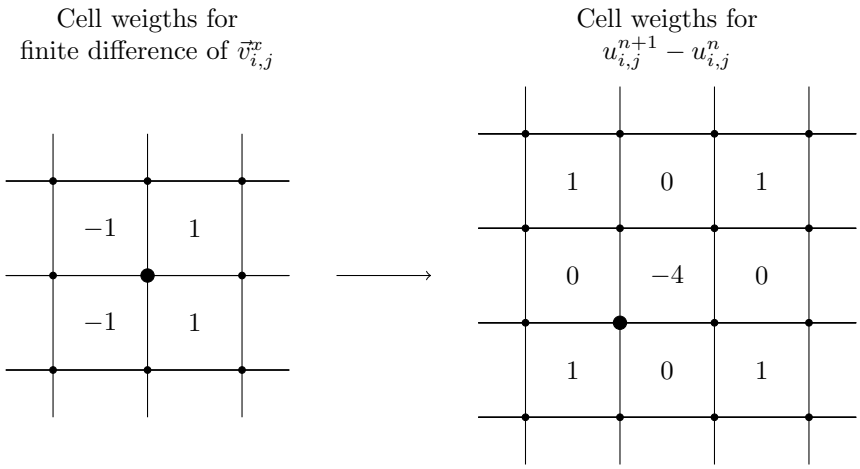


Figure 5–2: Weight distribution for velocity (5.36)

One solution is to use a wider stencil for the approximation of $\vec{v}$:

$$v^x_{i,j} = -\frac{q_{i+1,j} + q_{i+1,j-1} + 2q_{i,j} + 2q_{i,j-1} - 2q_{i-1,j} - 2q_{i-1,j-1} - q_{i-2,j} - q_{i-2,j-1}}{\Delta x(u_{i,j} + u_{i,j-1} + u_{i-1,j} + u_{i-1,j-1})}$$

$$v^y_{i,j} = -\frac{q_{i,j+1} + q_{i-1,j+1} + 2q_{i,j} + 2q_{i-1,j} - 2q_{i,j-1} - 2q_{i-1,j-1} - q_{i,j-2} - q_{i-1,j-2}}{\Delta y(u_{i,j} + u_{i,j-1} + u_{i-1,j} + u_{i-1,j-1})} \tag{5.37}$$

We can check that the stencil on the right in figure 5–3 is consistent with the Laplacian operator by Taylor expansion, it is also stable by von Neumann analysis.

Here is a summary of the method in pseudocode:

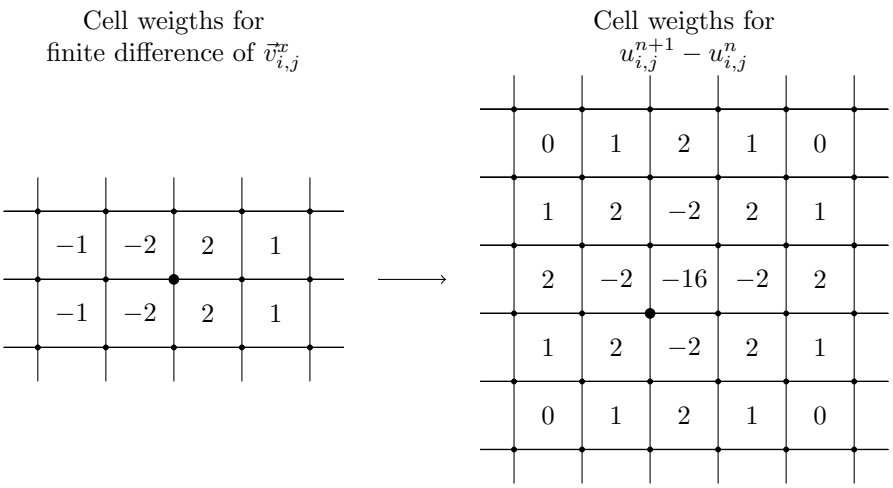


Figure 5–3: Weight distribution for velocity (5.37)

Algorithm 2 Mesh Transformation from Heat Flow

Inputs: $\Omega, \rho_0, \rho_M, \epsilon$

Outputs: $\vec{T}, u^n$

Let $\vec{x}_{ij}$ be grid points of a regular mesh on a simple domain. Let $R_{i,j}$ be the cell with $\vec{x}_{i,j}$ as its lower-left corner.
$r_{i,j} \leftarrow \int_{R_{i,j}} \rho_0(\vec{x}) d\vec{x}$
$u^0_{i,j} \leftarrow \int_{R_{i,j}} \rho_M(\vec{x}) d\vec{x}$
$\vec{\chi}^B(\vec{x}_{i,j}, 0) \leftarrow \vec{x}_{i,j}$
$res \leftarrow \max(|u^0_{i,j} - r_{i,j}|)/|R_{i,j}|$
$n \leftarrow 0$
while $res > \epsilon$ **do**
    $\vec{v}^n_{i,j} \leftarrow \frac{-1}{Average(u^n_{i,j})}(\nabla_h(u^n - r))_{i,j}$
    Cap the length of $\vec{v}^n_{i,j}\Delta t$ at $\frac{h}{3}$ at every grid point.
    Project $\vec{v}^n_{i,j}$ onto the tangent direction of the boundary for any $\vec{x}_{i,j} \in \partial\Omega$.
    $\mathring{\vec{x}}^n_{i,j} \leftarrow \vec{x}_{i,j} - \vec{v}^n_{i,j}\Delta t$
    $\vec{\chi}^B(\vec{x}_{i,j}, (n+1)\Delta t) \leftarrow \vec{\chi}^B(\mathring{\vec{x}}^n_{i,j}, n\Delta t)$ by linear interpolation
    $u^{n+1}_{i,j} \leftarrow \int_{\vec{\chi}^B(R_{i,j},(n+1)\Delta t)} \rho_M(\vec{x}) d\vec{x}$ by standard quadrature
    $res \leftarrow \max_{i,j}(|u^{n+1}_{i,j} - r_{i,j}|)/|R_{i,j}|$
    $n \leftarrow n + 1$
end while
$\vec{T} \leftarrow \vec{\chi}^B(\cdot, n\Delta t)$

5.4 Numerical Results

We present a few numerical experiments in this section. Figure 5–4 shows the results of the algorithm applied to some given monitor function. Figures 5–5 to 5–7 show the results of this method applied to evolving parametrized surfaces.

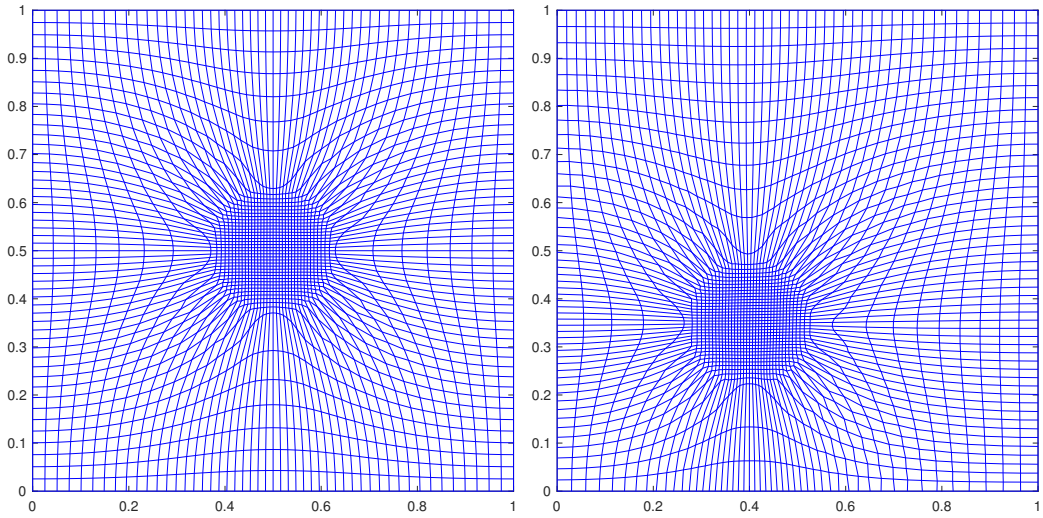
For figure 5–4, the maps are computed on a 50×50 cells grid, with $\Delta x = \Delta y = 1/50$. We chose $\epsilon = 5 \times 10^{-2}$; the iterations stop when the error $|\mu_M(\vec{\chi}^B(R_{i,j})) - \mu_0(R_{i,j})|$ is bounded by $\epsilon |R_{i,j}| = 2 \times 10^{-5}$. The subfigures illustrate $\vec{\chi}^B$ obtained from different monitor functions. To define these functions, we used the bump function given by:

$$\eta_r(s) = \begin{cases} \frac{C}{r^2} \exp\left(-\frac{1}{1-(s/r)^2}\right) & if\ |s| < 1 \\ 0 & if\ |s| \geq 1 \end{cases} \tag{5.38}$$

where the constant C is chosen so that η_r integrates to 1. The bump function has the property of being smooth with compact support in $[-r, r]$. We define the monitor functions by composing η_r with some distance function. For instance, $\rho_M(\vec{x}) = \eta_r(d(\vec{x}, K)) + 1$ for some set K. This way the mass of ρ_M is concentrated in an r-neighborhood of K.

We can see in figure 5–4d, one of the shortcomings of this method. Since only the cell areas are taken into account, we do not control the shape of the cells. As a consequence, we can find very elongated cells on the top right and bottom left of figure 5–4d. This needs to be corrected with some additional adjustments.

We can also apply this method to redistribute sample points on a moving parametric surface. For these experiments, we used the 3D swirl velocity field defined in (4.3) in Chapter 4. The results are shown in figures 5–5 to 5–7.



(a) $M(\vec{x}) = \eta_r(|\vec{x} - \vec{x}_0|^2) + 1$
$\vec{x}_0 = (0.5, 0.5)$

(b) $M(\vec{x}) = \eta_r(|\vec{x} - \vec{x}_0|^2) + 1$
$\vec{x}_0 = (0.4, 0.35)$

(c) $M(\vec{x}) = \eta_r(|\vec{x} - \vec{x}_0|^2 - 4) + 1$
$\vec{x}_0 = (0.5, 0.5)$

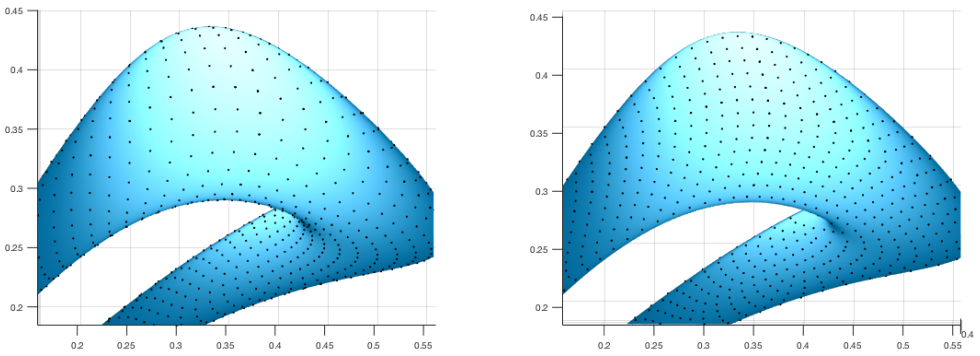
(d) $M(\vec{x}) = \eta_r((\vec{x} - \vec{x}_0) \cdot \vec{l}) + 1$
$\vec{x}_0 = (0.3, 0.5)$, $\vec{l} = (-1, 1)$

Figure 5–4: These mesh maps are computed using given monitor functions assuming a uniform distribution for the initial mesh. In all cases, r is taken to be 0.15 and the functions are normalized so that the integral is 1. All maps are computed using Neumann boundary conditions.

Figure 5–5 shows the sample point distribution on a torus evolved under the swirl flow. For this test, we used as initial surface a torus of major radius 0.1 and minor radius 0.025, parametrized by $\vec{P}_0 : [0,1]^2 \to \mathbb{T}$. The torus is centered at $(0.2, 0.2, 0.2)$. We distributed 2500 sample points across the unit square parametric space, 25 in the minor axis and 100 in the major axis. The torus and the sample points are then evolved forward in time using the characteristics mapping method in Chapter 3. We stopped the evolution at time $t = 1$, $S(1)$ is the pushforward of the torus, parametrized by $\vec{P}_1$. We computed a mesh transformation on the parametric space, and the new sample points are obtained by evaluating the mesh map at the original sample points and plotting them on $S(1)$ using P_1. From the point of view of transport maps, the mesh transformation moves uniformly distributed sample points to points that are distributed according to the area density of the surface with respect to the parametrization.

Directly applying the method in this chapter can redistribute these sample points evenly. However, in order to compensate for the fact that $S(1)$ and the unit square $[0,1]^2$ are not isometric, we multiply the velocity field by the "inverse" of the differential $\nabla \vec{P}_1$: Assume that at a point $\vec{s} \in S(1)$, we have a parametrization $\vec{Q} : U \to S(1)$ which is a local isometry at $\vec{s}$. We multiply the velocity field by the matrix given by $\left((\nabla \vec{Q})^{-1}(\nabla \vec{P}_1)\right)^{-1}$, which amounts to dividing each component of the velocity by the norm of $\partial_x \vec{P}_1$ and $\partial_y \vec{P}_1$ respectively. Intuitively, this is meant to mimic the diffusion process happening on the surface instead of the parametric space. It avoids the problem of having the diffusion carry out at drastically different speeds depending on the metric induced by the parametrization. The theoretical results should

still hold. Indeed, since the matrix multiplying the velocity is diagonal and positive definite, we are in the case of an anisotropic diffusion.



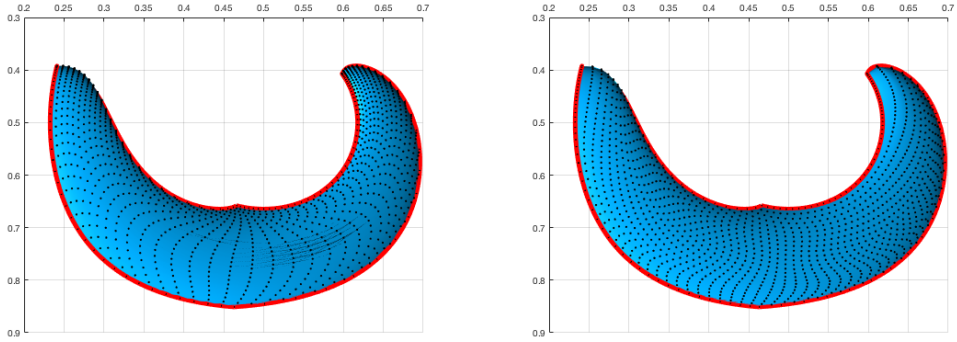
(a) Original Sample Points (b) Adapted Sample Points

Figure 5–5: Sample Points Distribution (Torus) - Adapted vs. Original

We also ran the same tests on a square given by $x = 0.35$, $y \in [0.25, 0.45]$ and $z \in [0.25, 0.45]$ as well as on a Möbius strip centered at $(0.35, 0.35, 0.35)$ of width and radius 0.1 whose central circle lies in the $x = 0.35$ plane. The mesh map is computed on a 64×64 grid for the square and a 32×128 grid for the Möbius strip. Figures 5–6 and 5–7 show the results; the square is drawn with 50×25 points, and the Möbius strip, with 8×100 points.

5.5 Acceleration by Grid Size Adjustment

The speed of this method is mostly limited by the size Δt of the time steps we take. Indeed, in order to reach a good approximation, we must find the backward map for a large t_f. If the times steps we take are small, many iterations are needed. The size of the time step is limited by the stability condition of the heat equation, that is, we need to take Δt on the same order as h^2 where h is the cell width. Clearly, taking

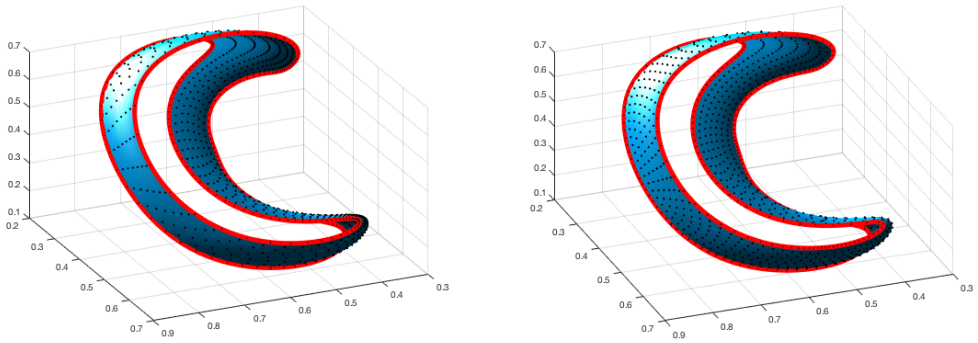


(a) Original Sample Points (b) Adapted Sample Points

Figure 5–6: Sample Points Distribution (Square) - Adapted vs. Original

finer grids allows for higher accuracy but can quickly becomes computationally costly. However, we can exploit the underlying heat equation to accelerate the method: the acceleration is achieved by starting the computations on a coarse grid and gradually refining towards the targeted resolution.

On a N points grid, the CFL stability condition allows us to take time steps of $O(1/N^2)$. Since the time it takes for a frequency ω signal to decay below a fixed threshold is $O(1/\omega)$, we would need $O(N^2/\omega)$ steps. However, the highest frequency this grid can represent is the Nyquist frequency of $N/2$, so it takes $O(N)$ steps to eliminate the highest observable frequency. If we gradually increase grid size from a coarse N_0 grid to an N_f fine grid, each grid of size N_i only deals with the highest frequency and takes $O(N_i)$ steps. Indeed, lower frequencies were eliminated by previous coarser grids. The total number of iterations should be on the order of $\sum N_i$. In our implementation, we increase grid size by doubling the number of grid points in each dimension, this should result in $O(N_f)$ number of iterations in

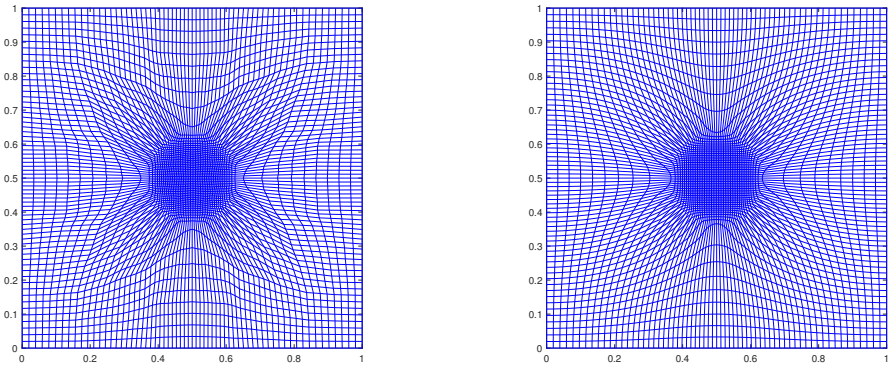


(a) Original Sample Points (b) Adapted Sample Points

Figure 5–7: Sample Points Distribution (Möbius) - Adapted vs. Original

total. Whereas if we made all iterations on the fine grid, the lowest frequencies would require $O(N_f^2)$ steps to decay.

We tested the method again for the bump function in figure 5–4a with $\epsilon = 5 \times 10^{-2}$. Using a gradually refined grid from 8 to 64, the computations can be carried out in about 0.6 seconds, whereas on a fixed 64×64 grid, the algorithm takes around 4 seconds to run.



(a) Map from Gradually Refined Grid (b) Map from Fixed Grid

Figure 5–8: Computation on fixed and gradually refined grid.

CHAPTER 6
Conclusion

In this thesis, we examined the numerical implementation and applications of the CM method. We designed a forward CM method which performs time integration using Runge-Kutta 4 and spacial interpolation using Hermite cubic splines. We analyzed the convergence of this method in 3 dimensions and performed some numerical tests. The results provide good evidence that the method is 4^{th} order convergent.

Since many situations require evolving implicitly and explicitly defined surfaces at the same time, the forward and backward characteristics maps should be computed simultaneously. In Chapter 3 we proposed a variation of the backward CM method which is coupled with the forward method to improve accuracy. We've also proved that the method should be 3^{rd} order accurate. However, the numerical tests did not make evident the 3^{rd} order convergence we were expecting. It is unclear now whether the test or the method was flawed. We have conjectured several possible reasons for the error. Further analysis and testing are required.

Finally, in Chapter 5, we applied the CM method to the moving adaptive mesh and particle management problem. The method is based on the continuity equation behind the heat equation and obtains the mesh map as the backward characteristics map of a chosen velocity field. We have shown in the 2 dimensional case that this map can be used to equidistribute a given mesh density function among grid cells. From

a mass transport point of view, it also allows us to move sample particles from one distribution to another. However, this method has limitations on the accuracy it can achieve. Also, for mesh adaptation purposes, it focuses only on the equidistribution principle and cannot control for mesh regularity or alignment. There remains a lot of work to be done in order to properly apply the CM method to the mesh management problem.

References

[1] Jean-David Benamou and Yann Brenier. The optimal time-continuous mass transport problem and its augmented lagrangian numerical resolution. *Rapport de recherche - Institut national de recherche en informatique et en automatique*, 1998.

[2] Chris J Budd and JF Williams. Parabolic monge–ampère methods for blow-up problems in several spatial dimensions. *Journal of Physics A: Mathematical and General*, 39(19):5425, 2006.

[3] Chris J Budd and JF Williams. Moving mesh generation using the parabolic monge-ampere equation. *SIAM Journal on Scientific Computing*, 31(5):3438–3465, 2009.

[4] Gian Luca Delzanno, Luis Chacón, John M Finn, Y Chung, and Giovanni Lapenta. An optimal robust equidistribution method for two-dimensional grid adaptation based on monge–kantorovich optimization. *Journal of Computational Physics*, 227(23):9841–9864, 2008.

[5] Lawrence C Evans. Partial differential equations. *Graduate Studies in Mathematics*, 19, 1998.

[6] Weizhang Huang and Robert D Russell. *Adaptive moving mesh methods*, volume 174. Springer Science & Business Media, 2010.

[7] Olivier Mercier and Jean-Christophe Nave. The characteristic mapping method for the linear advection of arbitrary sets. *arXiv preprint arXiv:1309.2731*, 2013.

[8] Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. A gradient-augmented level set method with an optimally local, coherent advection scheme. *Journal of Computational Physics*, 229(10):3802–3827, 2010.

[9] Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153. Springer Science & Business Media, 2006.

[10] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[11] Brian Von Herzen and Alan H Barr. Accurate triangulations of deformed, intersecting surfaces. *ACM SIGGRAPH Computer Graphics*, 21(4):103–110, 1987.

[12] Alan M Winslow. Adaptive-mesh zoning by the equipotential method. Technical report, Lawrence Livermore National Lab., CA (USA), 1981.