ABSTRACT

KING, SARAH AMALIE. Multi-Moment Methods for PDEs and GPUs for Large-Scale Scientific Computations. (Under the direction of Kazufumi Ito.)

The scope of this thesis is broad but focuses on developing effective numerical methods and efficient implementations. We investigate numerical solution methods for hyperbolic partial differential equations, numerical optimization methods, and implementation of fast numerical algorithms on graphics processor units (GPUs).

For partial differential equations we develop numerical methods for the transport and advection equations. Our method is based on the method of characteristics and multi-moment approximation of functions. At uniform grid points update formulas for solutions and their derivatives are derived for variable wave speed.

For numerical optimization we develop a nonsmooth optimization method for solving the elastic contact problem. The Signorini contact problem is a variational problem that minimizes the elastic deformation energy subject to the contact inequality. The Coulomb friction problem is a minimization of the deformable energy at the boundary. We develop a numerical optimization method of the form of Primal-Dual active set methods for Lagrange multiplier methods and semi-smooth newton method for these variational problems. To solve these problems numerically we approximate the variational problems with a multi-moment scheme based on Adini's elements which involves the use of the function values as well as the gradient values at nodes. These solution methods are then combined to solve the full contact problem.

Lastly we develop GPU implementations that combine algorithmic efficiency and computing power. We look at constrained and nonsmooth optimization algorithms as well as an inverse Hamiltonian based Riccati solver. Our efforts enhance numerical optimization and control problems for large-scale scientific systems.

Multi-Moment Methods for PDEs and GPUs for Large-Scale Scientific Computations

by
Sarah Amalie King

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Applied Mathematics

Raleigh, North Carolina

2012

APPROVED BY:

DEDICATION

To my parents, Leslie and Steven King.

BIOGRAPHY

Sarah Amalie King was born in Washington, District of Columbia, in 1985. She earned her Bachelors of Science in Applied Mathematics in 2007 from Georgia Institute of Technology. She began her graduate work at North Carolina State University in 2007 and earned her Masters of Science in Applied Mathematics in 2009.

She has accepted a research associateship from the National Research Council at the Naval Research Lab in Monterey, California.

ACKNOWLEDGEMENTS

First and foremost I would like to express my gratitude to my to my advisor, Professor Kazufumi Ito, for his continued support and guidance. His dedication and insight has encouraged me to continually improve as a mathematician. I am not sure what I was expecting the first day I walked into his office, but I definitely got more than I bargained for.

I would also like to thank my committee members especially Professors Zhilin Li and Ralph Smith for their professional support.

I am grateful to my family Leslie, Steven, Ashley, Dana, Alex and Jenna for their support, encouragement, and patience.

Finally I would like to thank to my friends and fellow graduate students. The late nights and the looniness are memories I will always cherish.

TABLE OF CONTENTS

List of Tables . . . . **vii**

List of Figures . . . . **viii**

Chapter 1 Introduction **1**

Chapter 2 CIP Methods for Hyperbolic Equations in One Dimension . **5**

2.1 CIP Method 5

2.2 One Dimensional CIP Updates 6

2.2.1 Transport Equation 6

2.2.2 Advection Equation 8

2.3 CIP Profiling 10

2.4 One Dimensional CIP Numerical Results 15

Chapter 3 CIP Methods for Hyperbolic Equations in Two Dimensions **20**

3.1 Two Dimensional Updates 20

3.1.1 Two Dimensional Transport 20

3.1.2 Two Dimensional Advection 23

3.1.3 Operator Splitting 26

3.2 CIP Profiling 28

3.3 Two Dimensional CIP Numerical Results 30

Chapter 4 Nonsmooth Optimization for the Elastic Contact Problem . **34**

4.1 Model Description 34

4.2 Variational Formulation of Signorini and Coulomb Friction Problems . . 36

4.2.1 Signorini Contact Problem 37

4.2.2 Coulomb Friction Problem 37

4.3 Nonsmooth Optimization Algorithms 39

4.3.1 Primal-Dual Active Set Method for Coulomb Friction Problem . . 41

4.3.2 Primal-Dual Active Set Method for Signorini Contact Problem . . 43

4.3.3 Full Contact Problem 44

Chapter 5 Finite Element Approximation using Adini Elements **45**

5.1 Numerical Method 45

5.1.1 Adini's Element Discretization 46

5.1.2 Accuracy Test for Eigenvalue Computations 49

5.1.3 Adini Elements on Irregular Domains 53

5.2 Test Example 55

Chapter 6 CUDA Acceleration **59**

6.1 Optimization 60
6.1.1 Lagrange Multiplier Method 61
6.2 Controls 64
6.2.1 Heat Equation Example 66
6.3 Future Work 69
6.3.1 Numerical Weather Prediction 71

References **73**

Appendix **75**
Appendix A CUDA Review 76
A.0.2 Programming Model 76
A.0.3 Linear system example 81
A.0.4 Eigen code example 81

LIST OF TABLES

Table 2.1 Error in numerical solutions for transport equation in ℓ_2. 16
Table 2.2 Error in numerical solutions for transport equation in ℓ_1. 17
Table 2.3 Error in numerical solutions for advection equation in ℓ_2. 19

Table 5.1 Relative error in approximated eigenvalues for Neumann conditions 50
Table 5.2 Relative error in approximated eigenvalues for Dirichlet conditions 51
Table 5.3 Estimated Convergence rates for nonzero eigenvalues for the elastic system with Neumann conditions 52
Table 5.4 Relative error in approximated eigenvalues for irregular domain with Dirichlet conditions 55

Table 6.1 Performance in seconds with dense matrix with a subspace size of 25 and calculating 10 eigenvalues. We use either a quadcore (four Intel i7s) or a single GPU (Tesla C2050) 66
Table 6.2 Performance in seconds with dense matrices with a subspace size of L. We use either a quadcore (four Intel i7s) or a single GPU (GeForce GT 640M LE) 69

LIST OF FIGURES

Figure 2.1 Diagram of backward characteristics in time for system (2.4). . . . 7

Figure 2.2 Numerical solution $u(x,t)$ using linear profile for $t = 0, 1.2, 2.4$, and 6. 16

Figure 2.3 (a) Numerical solution $u(x,t)$ and (b) the solution derivative $v(x,t)$ using multi-moment piecewise linear profile for $t = 0, 1.2, 2.4$, and 6. 16

Figure 2.4 (a) Numerical solution $u(x,t)$ and (b) the solution derivative $v(x,t)$ using multi-moment cubic profile for $t = 0, 1.2, 2.4$, and 6. 17

Figure 2.5 Error in the numerical solutions at $t = 1$ in ℓ_2. 17

Figure 2.6 Error in the numerical solutions in ℓ_2 verse $CFL = .5, 1$, and 2. . 18

Figure 2.7 Numerical solution $u(x,t)$ using (a) the multi-moment piecewise linear profile (2.18) and (b) the multi-moment cubic profile (2.22) for $t = 0, 0.8, 1.2$, and 2. 19

Figure 3.1 Diagram of backward characteristics for system (3.4) in two dimensions. 22

Figure 3.2 (a) Initial conditions (b) $t = .3$ (c) $t = .6$ (d) t=.9, and (e) $t = 1.2$. (f) is a 1-D cut at $x = -.02$ and $y = -2$ to 2 when $t = 1.2$ for example 1. 32

Figure 3.3 (a) Initial conditions (b) $t = 8\pi$ (c) $t = 16\pi$ (d) 1-D cut at $x = -.84$ and $y = -2$ to 2 when $t = 8\pi$ for example 2. 33

Figure 5.1 Shape functions $\phi_k(x,y)$ for $k = 1,..,12$. 47

Figure 5.2 Block tri-Diagonal matrix of block tri-diagonal matrices 48

Figure 5.3 (a) Error plot for Neumann boundary conditions, (b) error plot for Dirichlet boundary conditions and (c) error plot for elastic system with stress free boundary conditions. 52

Figure 5.4 Square cells with (a) one interior node, (b) two interior nodes, and (c) three interior nodes. 53

Figure 5.5 Deformed elastic body in contact with a rigid foundation. 56

Figure 5.6 (a) Vertical displacement, (b) enhanced vertical displacement, (c) enhanced horizontal displacement, and (d) μ. 57

Figure 6.1 Matlab and CUDA computational times (s) with random matrices 63

Figure A.1 CUDA programming model 77

Chapter 1

Introduction

This thesis explores the scientific methodology of numerical partial differential equations, inverse problems, and control problems. Specifically we investigate numerical methods for hyperbolic partial differential equations, elastic contact problems, and constrained and nonsmooth optimization. Additionally we develop fast numerical optimization methods and Riccati equation solver for large-scale problems and their implementations on graphics card processors (GPUs).

The scope of the thesis is very broad but focuses on developing effective numerical methods and efficient implementations. For numerical PDEs we develop multi-moment approximations based on the Hermite interpolation, i.e., the solution and its derivatives are simultaneously calculated. For hyperbolic equations this approach uses exact time integration method for the solution and its derivatives based on the characteristic method and the cubic Hermite interpolation for the solution profile. Consequently, we have high order and a less dissipative numerical methods. For elliptic systems we employ Adini elements and finite element approximation based on weak formulations.

We discuss a nonsmooth optimization method based on the Lagrange multiplier theory and the semismooth Newton method. Specifically, we study elastic contact problems. Numerical tests of the proposed algorithm are performed using the Adini element based finite element approximation. The numerical methods for PDEs and optimization problems may seem unrelated at first but consider the estimation of the coefficient c for the hyperbolic equation

$$u_t + c(x)u_x = 0$$

$$u(x,0) = f(x) \text{ and } u(x,T) = z(x)$$

which can be formulated as the least squares problem

$$\min \tfrac{1}{2}|u(x,T) - z|^2 + \tfrac{\alpha}{2}||c||^2 \text{ over } c$$

$$\text{subject to } u_t + cu_x = 0,\ u(x,0) = f \text{ and } c(x) \geq \beta > 0.$$

To solve this optimization problem the initial value problem for the hyperbolic equation must be solved. As is often the case in mathematics there are overlapping aspects that must be integrated as illustrated above.

The constrained and nonsmooth optimization algorithms we developed are implemented on GPUs. GPUs, originally developed for graphics rendering, have evolved into a highly parallel, multi-threaded, many-core processors with high memory bandwidth for distributed computing. In terms of scientific computations this means that GPUs are hardware designed for large-scale problems with intense computations. Leveraging GPU parallelism and powerful algorithms our efforts in numerical optimization and control problems offer a significant advancement and enhance applications in for large- scale systems. The combined efficiency and effectiveness of the algorithms developed and GPU implementations are demonstrated through numerical tests. Included in the GPU implementations is the inverse Hamiltonian based Riccati solver which uses the implicit restarted Arnoldi method for partial eigenvalue problems. Our intent is to build tools that allow for large-scale problems to be solved rapidly and efficiently which will enlarge possible applications.

The following is the outline of this thesis. Specific aspects of the topics and our approach are described:

In Chapter 2 of this thesis we investigate numerical methods for hyperbolic partial differential equations. Our approach is to use a multi-moment method and extend previously developed cubic interpolation profiling (CIP) methods. This approach is developed for the transport equation

$$u_t + c(x)u_x = 0$$

and the advection equation

$$u_t + (c(x)u)_x = 0.$$

The method developed involves multi-moment approximation of the solution and its gradient as well as exact time integration formulas based on the method of characteristics. Several solution profiles, i.e. Hermite interpolations, are developed to obtain accurate approximations for sharp gradient solutions. A detailed description of the method and numerical tests are performed to demonstrate the applicability of the proposed methods.

In Chapter 3 we extend our methods developed in Chapter 2 to the two dimensional case. The first method we present is a direct extension to two dimensions of our CIP scheme. The second method is based on time splitting integration and is suitable for even higher dimensions. The methods are developed for the transport and advection equations and numerical tests of the methods are presented.

In Chapter 4 we develop a nonsmooth optimization method for solving the elastic contact problem. The Signorini contact problem is a variational problem that minimizes the elastic deformation energy subject to the contact inequality, i.e., the normal displacement at a given point of the boundary bounded above by an obstacle function. The Coulomb friction problem is a minimization of the deformable energy with a L^1 friction term at the boundary. We develop an effective numerical optimization method using the Lagrange multiplier theory and the semi-smooth Newton method for the both variational problems. The method is of the form of Primal-Dual active set methods for the solution and Lagrange multipliers. Then both methods are combined to develop an iterative method for solving the full elastic contact problem.

In Chapter 5 we present a multi-moment finite element scheme based on Adini's elements to approximate the solution to the linear elastic equations numerically. The accuracy and feasibility of the method developed are demonstrated for eigenvalue problems for the linear elastic system. Then we apply the method for solving the two variational problems and full contact problem.

Lastly, in Chapter 6 we address software development for large-scale problems in optimization and control theory. We discuss strategy and capability for developing general purpose GPU software for scientific computing as well as the capability of such software. To accompany this discussion we have included a review of CUDA C based on our experiences in Appendix A. Currently we have developed software for the quadratic constrained optimization problem as well as software for finding a partial set of eigenvalues and eigenvectors of a given matrix. We have included the algorithm descriptions for both problems and present basic benchmark tests. We also include a discussion of future directions of our work and a potential application that combines the techniques and approaches developed in this thesis for PDEs, optimization, and large-scale problems.

Chapter 2

CIP Methods for Hyperbolic Equations in One Dimension

In this chapter we develop a multi-moment method based on the method of characteristics for hyperbolic equations with variable wave speeds. The method is a variation of the cubic interpolation method (CIP) [17] in which Hermite polynomial based spatial profiles over each grid cell are used to update the solution and its derivatives simultaneously. We will apply our method to the transport and advection equations and test our method numerically.

2.1 CIP Method

Our method originates from the cubic interpolated profile (CIP) method for the one-dimensional transport equation. The CIP method proposed in [3] has previously been proven to be numerically stable and have less numerical dispersion than other methods, e.g. upwinding, in the presence of singularities [17, 19, 16, 14]. Since its introduction CIP methodology has been applied to many problems like acoustic wave propagation [13], or incompressible flow [15] and several variations of the CIP method have been developed, eg. CIP-MOC [18] or CIP-IMM [5]. In order to include all of these variations the method is now sometimes referred to as the constrained interpolation method.

The general framework of these methods is a spatial profile for each grid cell is developed using function values and derivative values then for each time step these profiles are used to update each grid point. The conventional CIP method uses a cubic polynomial for the spatial profile for each grid cell and the constant transport part of the equation is updated by the characteristic method in time. Additional terms appearing in equations are integrated in time by the operator splitting method.

Our approach differs from the conventional CIP scheme primarily in two ways. First we use exact integration in time by the method of characteristics and secondly we derive exact gradient updates in time. That is, we extend the method of characteristics for the derivative of solutions (the exact derivative update) along with the solution. Then we obtain a higher-order time-space discretization method using the interpolation methods (for example, cubic Hermite interpolation) for the solution profile at each square cell and then apply the solution and derivative update formula by evaluating for the local profile of the solution.

Our approach is advantageous as it allows for arbitrary time steps (i.e., no CFL limitation) while maintaining its stability and accuracy. Additionally for the advection equation existing CIP schemes use the non-conservative form of the advection equation and split it into advection and non-advection phases to obtain updates. On the other hand we developed exact time integration and do not need to split equations.

In order to describe our method we first develop the update formulas for the solution and solution derivative for the one dimensional transport equation as well as the advection equation with variable wave speed. We then provide three different profiling techniques for these formulas. Numerical tests are preformed to demonstrate the effectiveness of our method. We will extend this method for the multi-dimensional case for the transport and advection equations in Chapter 3.

2.2 One Dimensional CIP Updates

In this section we will formulate the solution and solution derivative updates for the transport and advection equations in one dimension based on the method of characteristics.

2.2.1 Transport Equation

First we consider the one dimensional transport equation given by

$$u_t + c(x)u_x = 0,\ t > 0,\ t \in \mathbb{R},\ u(0,x) = u_0(x),\ x \in \mathbb{R} \tag{2.1}$$

where $c(x)$ is the variable wave speed. We begin applying the method of characteristics by setting $U(t) = u(x(t), t)$ along the characteristics where $x(t)$ satisfies the characteristic equation

$$\frac{d}{dt}x(t) = c(x(t))$$

and it follows from (2.1) that

$$\begin{aligned} \frac{dU(t)}{dt} &= \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x}\frac{dx(t)}{dt} \\ &= \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x}c(x(t)) = 0. \end{aligned} \tag{2.2}$$

Discretizing with $x_k = k\Delta x$ we obtain the solution update formula

$$U(t + \Delta t) = u(x_k, t + \Delta t) = u(y_k, t) \tag{2.3}$$

where y_k is determined by the backward characteristic (depicted by Figure 2.1)

$$\begin{cases} \dfrac{dx}{dt} &= c(x(t)) \\ x(0) &= x_k \\ x(-\Delta t) &= y_k \end{cases} \tag{2.4}$$

for this system x_k is the current value and y_k is the previous value. If $c(x) = c$ is constant then $y_k = x_k - c\Delta t$ and (2.3) reduces to $u(x_k, t + \Delta t) = u(x_k - c\Delta t, t)$.

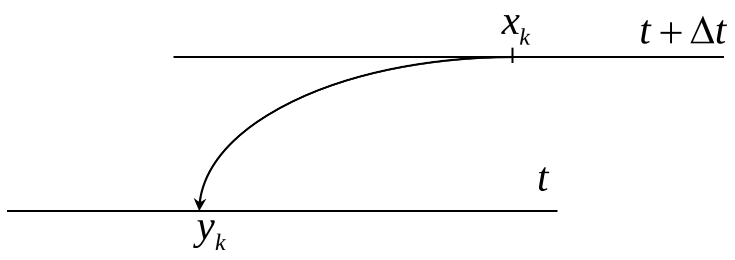


Figure 2.1: Diagram of backward characteristics in time for system (2.4).

We now develop the formula for updating the derivative u'_k. Taking the spatial derivative of (2.1) gives

$$u'_t + c(x)u'_x + c'(x)u' = 0. \tag{2.5}$$

We let $V(t) = u'(x(t), t)$ along the characteristics and apply (2.5) to get

$$\begin{aligned}\frac{dV}{dt} &= \frac{\partial}{\partial t}u' + \frac{\partial}{\partial x}u'\frac{dx(t)}{dt} \\ \\ &= -c'(x(t))V(t)\end{aligned}$$

thus, we obtain

$$V(t + \Delta t) = u'(x_k, t + \Delta t) = u'(y_k, t)exp\left(-\int_{-\Delta t}^{0} c'(x(t))dt\right). \tag{2.6}$$

To evaluate this integral we let $dx = c(x(t))dt$ then

$$\begin{aligned}\int_{-\Delta t}^{0} c'(x(t))dt &= \int_{y_k}^{x_k} c'(x)\frac{dx}{c(x)} \\ &= \ln\left(\frac{c(x_k)}{c(y_k)}\right)\end{aligned} \tag{2.7}$$

and applying (2.7) to (2.6) gives the update formula

$$u'(x_k, t + \Delta t) = u'(y_k, t)\frac{c(y_k)}{c(x_k)}. \tag{2.8}$$

We now have explicit update formulas for the solution and the solution derivative of the transport equation. Later we will discuss profiling techniques to evaluate $u(y_k, t)$ and $u'(y_k, t)$ in our CIP scheme.

2.2.2 Advection Equation

We now consider the advection equation

$$p_t + (c(x)p)_x = 0 \tag{2.9}$$

or equivalently

$$p_t + c(x)p_x + c'(x)p = 0 \tag{2.10}$$

where $p(x,t)$ is for example a probability density function. Setting $P(t) = p(x(t),t)$ along the characteristics, $P(t)$ satisfies

$$\frac{d}{dt}P(t) = -c'(x(t))P(t).$$

Noting that this is the same as (2.5) for u' we arrive at the update of $p(x(t),t)$

$$p(x_k, t+\Delta t) = p(y_k, t)\frac{c(y_k)}{c(x_k)}. \tag{2.11}$$

We now derive the formula for update of $p'(x(t),t)$. Taking the spatial derivative of (2.9) gives

$$p'_t + c(x)p'_x + 2c'(x)p' + c''p = 0. \tag{2.12}$$

Let $Q(t) = p'(x(t),t)$ along the characteristics and then

$$\begin{aligned} \frac{dQ(t)}{dt} &= \frac{\partial}{\partial t}p' + \frac{\partial}{\partial x}p'\frac{dx}{dt} \\ &= -2c'(x(t))Q(t) - c''(x(t))P(t) \end{aligned}$$

by (2.12) where $P(t) = p(x(t),t)$. So

$$\begin{aligned} p'(x(t), t+\Delta t) &= Q(t+\Delta t) \\ &= \exp\left(-2\int_{-\Delta t}^{0} c'(x(t))dx\right) p'(y_k, t) \\ &- \int_{-\Delta t}^{0} \exp\left(\int_{-s}^{0} -2c'(x(t))dt\right) c''(x(s))p(x(s), s)ds. \end{aligned} \tag{2.13}$$

In order to evaluate (2.13) consider the time derivative of (2.9) given by

$$\dot{p}_t + (c(x)\dot{p})_x = 0$$

then

$$\dot{p} + c(x)\dot{p}' + c'(x)\dot{p} = 0 \tag{2.14}$$

which satisfies (2.10) and we have

$$\dot{p}(x_k, t + \Delta t) = \dot{p}(y_k, t)\frac{c(y_k)}{c(x_k)}. \tag{2.15}$$

Applying (2.14) to (2.15) gives

$$\begin{aligned}
(-c(x)p' - c'(x)p)(x_k, t + \Delta t) &= (-c(y)p' - c'(y)p)(y_k, t)\frac{c(y_k)}{c(x_k)} \\
\Rightarrow \quad -c(x_k)p'(x_k, t + \Delta t) - c'(x_k)p(x_k, t + \Delta t) &= (-c(x)p' - c'(x)p)(y_k, t)\frac{c(y_k)}{c(x_k)}
\end{aligned}$$

$$\Rightarrow p'(x_k, t + \Delta t) = -\frac{c'(x_k)}{c(x_k)}p(x_k, t + \Delta t) + \left(\frac{c(y_k)}{c(x_k)}p'(y_k, t) + \frac{c'(y_k)}{c(x_k)}p(y_k, t)\right)\frac{c(y_k)}{c(x_k)}.$$

Now we apply (2.11) and the update of the derivative becomes

$$p'(x_k, t + \Delta t) = -\frac{c'(x_k)}{c(x_k)}\frac{c(y_k)}{c(x_k)}p(y_k, t) + \left(\frac{c(y_k)}{c(x_k)}\right)^2 p'(y_k, t) + \frac{c'(y_k)}{c(x_k)}\frac{c(y_k)}{c(x_k)}p(y_k, t). \tag{2.16}$$

We now have explicit update formulas for the solution and the solution derivative of the advection equation in one dimension. We will discuss profiling techniques to evaluate $p(y_k, t)$ and $p'(y_k, t)$ in section 2.3.

2.3 CIP Profiling

We now develop the interpolation methods that give an accurate profile of the solution incorporating the values of the solution and its derivatives in a local cell. We have already developed explicit update formulas for the solution and the solution derivative in previous sections and we will use these interpolation techniques to evaluate these formulas.

For the one dimensional case we develop a piecewise liner interpolation as well as the standard Taylor series based methods for each interval using the solution and derivatives at the endpoints of each interval. In section 2.4 we will demonstrate numerically the efficiency and accuracy of these profiling techniques using numerical examples. Our procedure for each of the profiling methods is as follows

1. Solve the characteristic ODEs. For the transport equation this would be $\frac{d}{dt}x(t) = c(x(t))$ subject to $x(\Delta t) = x_k$ to find y_k's or wave speed.

2. For each time level t_n construct a profile for each subinterval (x_{k-1}, x_k).

3. Update solution and solution derivatives based on the update formulas using the profiles to evaluate.

Note that we need only perform the first step (which is also the most time consuming) once. We will now describe linear profiling, piecewise linear profiling and cubic profiling; we provide the the function definitions that we use for the solution updates. We use the appropriate derivatives of these functions to evaluate the derivative update solutions.

Linear Profile

Consider the data set given by $\{u_k\}$ where u_k is the function value. On the interval $[-h, 0]$ the updates for u_k is approximated by the linear function $\theta(x) = a_0 + a_1$ where

$$\begin{aligned} u_0 &= \theta(-h) = a_0 - a_1 h \\ u_1 &= \theta(0) = a_0. \end{aligned}$$

Solving this system gives

$$\begin{aligned} a_0 &= u_1 \\ a_1 &= \frac{u_1 - u_0}{h}. \end{aligned}$$

Then for $y_j \in [x_{k-1}, x_k]$

$$\theta_k(y_j) = u_k + \frac{u_k - u_{k-1}}{\Delta x}(y_k - x_k).$$

Multi-Moment Piecewise Linear Profile

In our multi-moment scheme a linear profile is created for each subinterval based on the function values and it derivative at the two ends. The updates for u_k and u'_k are calculated individually according to this profile. Given the data set $\{u_k, u'_k, x_k, \Delta t\}$ consider the node

x on the interval $[-h/2, h/2]$ then define

$$\phi_k(x) = \begin{cases} \phi_\ell = \left(x - \dfrac{h}{2}\right) u'_{k-1} + u_{k-1} & x \leq x^* \\ \phi_r = \left(x + \dfrac{h}{2}\right) u'_k + u_k & x > x^* \end{cases} \tag{2.17}$$

where

$$x^* = \frac{(u'_k + u'_{k-1})h - 2(u_k - u_{k-1})}{2(u_k - u_{k-1})}$$

assuming ϕ_ℓ and ϕ_r intersect. This is a hat function for each subinterval as our profile to approximate $u(y_k, t)$. There are several potential issues that arise with this definition of $\phi(x)$:

1. Division by small numbers when ϕ_ℓ and ϕ_r are nearly parallel.
2. There are multiple slopes for x when $x = x^*$.
3. No intersection between ϕ_ℓ and ϕ_r.

To tackle these issues the approach taken is to adjust the definition of (2.17) in calculations. Reformulating $\phi_k(x)$ gives the expression

$$\begin{aligned} \phi_k \quad &= \frac{\dfrac{\phi_r}{1+\exp\left(\dfrac{-(\phi_r-\phi_\ell)}{c}\right)} + \dfrac{\phi_\ell}{1+\exp\left(\dfrac{(\phi_r-\phi_\ell)}{c}\right)}}{1+\exp\left(\dfrac{-(u'_{k-1}-u'_k)}{c}\right)} \\ &+ \frac{\dfrac{\phi_r}{1+\exp\left(\dfrac{(\phi_r-\phi_\ell)}{c}\right)} + \dfrac{\phi_\ell}{1+\exp\left(\dfrac{-(\phi_r-\phi_\ell)}{c}\right)}}{1+\exp\left(\dfrac{(u'_{k-1}-u'_k)}{c}\right)} \end{aligned} \tag{2.18}$$

where c is a small constant (in our computations we used $c = 10^{-6}$). Using (2.18) eliminates division by small numbers when ϕ_ℓ and ϕ_r are nearly parallel to give the piecewise linear profile for u_k. To address the possibility of multiple slopes at x^* we modify ϕ_ℓ and ϕ_r a little. Define

$$\hat{\phi}_\ell = \left((x \pm \delta) - \frac{h}{2}\right) u'_{k-1} + u_{k-1} \tag{2.19}$$

and

$$\hat{\phi}_r = \left((x \pm \delta) + \frac{h}{2}\right) u'_k + u_k \tag{2.20}$$

where δ small and sign(δ) depends on the direction of motion. This formulation also allows for differentiation; the expression used to evaluate $\phi'_k(x)$ is

$$\begin{aligned} \phi'_k \;=\; & \frac{\dfrac{u'_k}{1+\exp\left(\dfrac{-(\hat{\phi}_r - \hat{\phi}_\ell)}{c}\right)} + \dfrac{u'_{k+1}}{1+\exp\left(\dfrac{(\hat{\phi}_r - \hat{\phi}_\ell)}{c}\right)}}{1+\exp\left(\dfrac{-(u'_{k-1} - u'_k)}{c}\right)} \\ & + \frac{\dfrac{u'_k}{1+\exp\left(\dfrac{(\hat{\phi}_r - \hat{\phi}_\ell)}{c}\right)} + \dfrac{u'_{k-1}}{1+\exp\left(\dfrac{-(\hat{\phi}_r - \hat{\phi}_\ell)}{c}\right)}}{1+\exp\left(\dfrac{(u'_{k-1} - u'_k)}{c}\right)}. \end{aligned} \tag{2.21}$$

Multi-Moment Cubic Profile

Consider the data set given by $\{u_k, u'_k\}$ where u_k is the function value and u'_k is the derivative value. On the interval $[-h, 0]$ the updates for u_k and u'_k are approximated by the cubic polynomial $\psi(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ where

$$\begin{aligned} u_0 &= \psi(-h) = a_0 - a_1 h + a_2 h^2 - a_3 h^3 \\ u'_0 &= \psi'(-h) = a_1 - 2a_2 h + 3a_3 h^2 \\ u_1 &= \psi(0) = a_0 \\ u'_1 &= \psi'(0) = a_1 \end{aligned}$$

Solving this system gives

$$\begin{aligned}
a_0 &= u_1 \\
a_1 &= u_1' \\
a_2 &= \frac{(2u_1' + u_0')h - 3(u_1 - u_0)}{h^2} \\
a_3 &= \frac{(u_1' + u_0')h - 2(u_1 - u_0)}{h^3}.
\end{aligned}$$

Then for $y_j \in [x_{k-1}, x_k]$

$$\begin{aligned}
\psi_k(y_j) &= u_k + u_k'(y_k - x_k) + \frac{(2u_k' + u_{k-1}')(\Delta x) - 3(u_k - u_{k-1})}{(\Delta x)^2}(y_j - x_k)^2 \\
&\quad + \frac{(u_k' + u_{k-1}')(\Delta x) - 2(u_k - u_{k-1})}{(\Delta x)^3}(y_j - x_k)^3.
\end{aligned} \tag{2.22}$$

and

$$\begin{aligned}
\psi_k'(y_j) &= u_k' + \frac{2(2u_k' - u_{k-1}')(\Delta x) - 6(u_k - u_{k-1})}{(\Delta x)^2}(y_j - x_k) \\
&\quad + \frac{3(u_k' - u_{k-1}')(\Delta x) - 6(u_k - u_{k-1})}{(\Delta x)^3}(y_j - x_k)^2.
\end{aligned} \tag{2.23}$$

Once a type of profile is selected the update to the solution to the transport equation, u_k, in (2.3) is given by

$$u_k^{n+1} = F_k(y_k). \tag{2.24}$$

and the update for u_k' is given by

$$(u_k')^{n+1} = F_k'(y_k)\frac{c(y_k)}{c(x_k)} \tag{2.25}$$

where F is the interpolation function. Similarly for the advection equation (2.9) the update is given by

$$p_k^{n+1} = F_k(y_k)\frac{c(y_k)}{c(x_k)} \tag{2.26}$$

and

$$(p_k')^{n+1} = -\frac{c'(x_k)}{c(x_k)}\frac{c(y_k)}{c(x_k)}F_k(y_k) + \left(\frac{c(y_k)}{c(x_k)}\right)^2 F_k'(y_k) + \frac{c'(y_k)}{c(x_k)}\frac{c(y_k)}{c(x_k)}F_k(y_k). \tag{2.27}$$

We have presented three profiles for the interpolation required to evaluate our solution update formulas, next we will discuss the numerical accuracy of our proposed profiles using numerical tests.

2.4 One Dimensional CIP Numerical Results

Example 1 (Transport Equation):

We consider the case of the transport equation with $c(x) = x - x^3$ and $u(x, 0) = exp(-10x^2)$. Then (2.1) becomes

$$u_t + (x - x^3)u_x = 0 \tag{2.28}$$

where $x \in [-2, 2]$ and $t > 0$. The analytic solution to (2.29) is given by

$$u(x,t) = exp\left(-10\left(x\sqrt{\frac{e^{-2t}}{-x^2 + e^{-2t}x^2 + 1}}\right)^2\right).$$

We apply our CIP scheme with the linear and non-linear profiles. In Figure 2.2, Figure 2.3, and Figure 2.4 our numerical solutions were calculated with $\Delta x = \Delta t = .02$. The three profiles effectively capture the discontinuities that form in our solution at $x = -1, 1$. It should be noted that oscillations or overshooting may occur when using the cubic profile in the presence of sharp discontinuities.

In Figure 2.5 the cubic profile achieves a fourth order convergence. In Table 2.1 and Table 2.2 we can observe that the linear profile requires sixteen times the number of points as the piecewise linear profile to achieve the same level of accuracy. Even if we double the number of unknowns for the linear profile to equal the same number of unknowns for the multi-moment piecewise profile we can not achieve the same performance. For the multi-moment methods (piecewise linear and cubic) we have in Figure 2.3(b) and Figure 2.4(b) included the gradient solutions, we do not use gradient updates for the linear case. Lastly in Figure 2.6 we show the error for various CFL numbers for the three profiles.

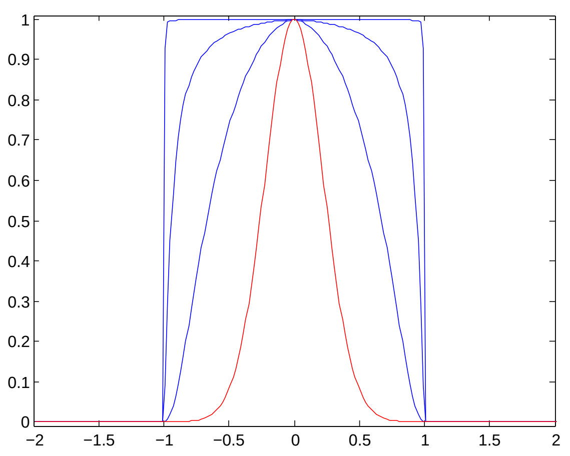


Figure 2.2: Numerical solution $u(x, t)$ using linear profile for $t = 0, 1.2, 2.4$, and 6.

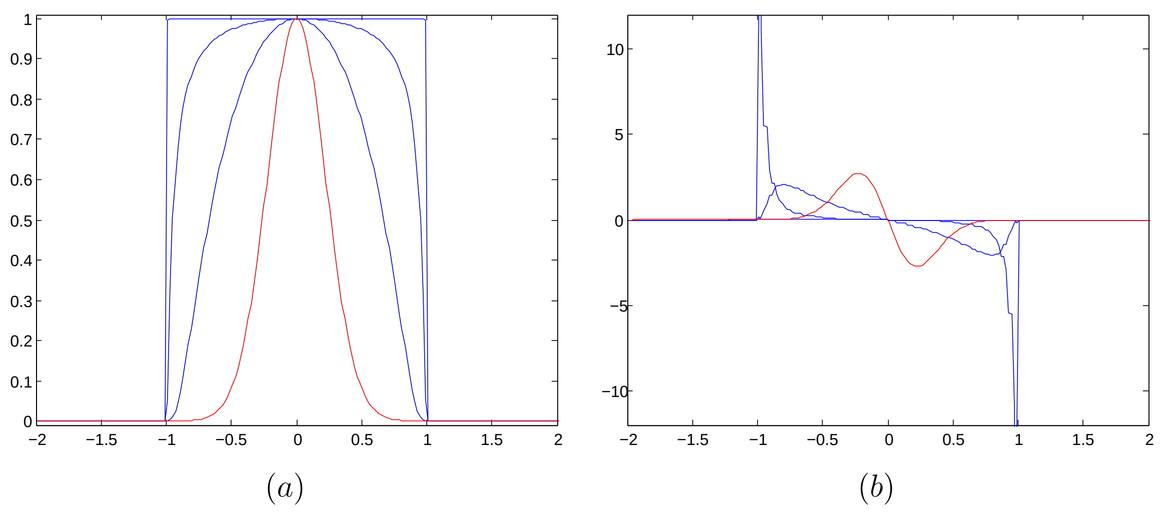


Figure 2.3: (a) Numerical solution $u(x, t)$ and (b) the solution derivative $v(x, t)$ using multi-moment piecewise linear profile for $t = 0, 1.2, 2.4$, and 6.

Table 2.1: Error in numerical solutions for transport equation in ℓ_2.

N	200	400	800	1600	3200
Linear	1.024e-2	5.206e-3	2.626e-3	1.319e-3	6.608e-4
Piecewise Linear	4.488e-4	1.108e-4	2.917e-5	7.492e-6	2.196e-6
Cubic	1.569e-4	2.834e-5	5.062e-6	8.993e-7	1.594e-7

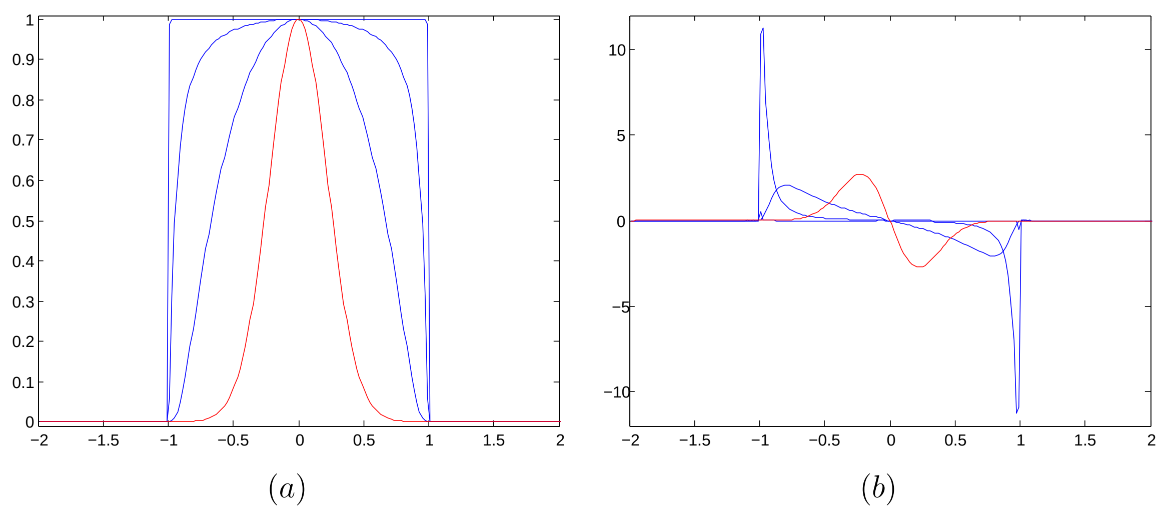


Figure 2.4: (a) Numerical solution $u(x,t)$ and (b) the solution derivative $v(x,t)$ using multi-moment cubic profile for $t = 0, 1.2, 2.4$, and 6.

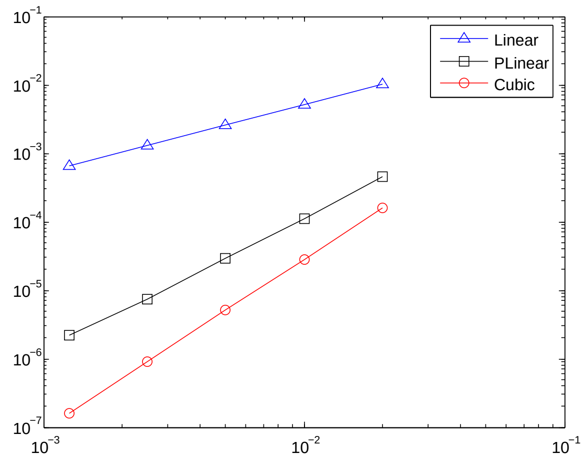


Figure 2.5: Error in the numerical solutions at $t = 1$ in ℓ_2.

Table 2.2: Error in numerical solutions for transport equation in ℓ_1.

N	200	400	800	1600	3200
Linear	1.271e-2	6.422e-3	3.230e-3	1.620e-3	8.115e-4
Piecewise Linear	4.878e-4	1.160e-4	3.044e-5	7.85e-6	2.320e-6
Cubic	5.875e-5	7.585e-6	9.616e-7	1.207e-7	1.579e-8

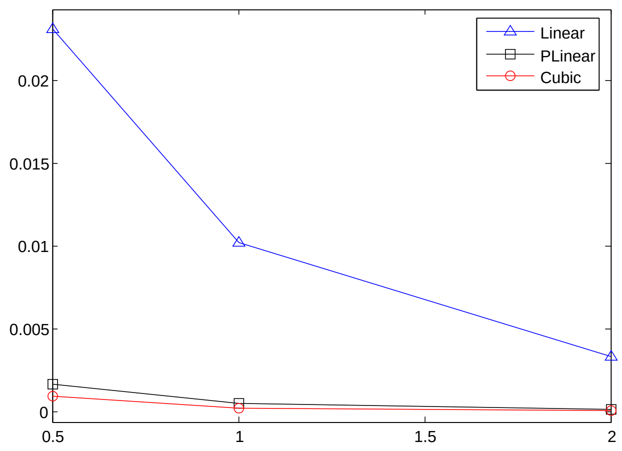


Figure 2.6: Error in the numerical solutions in ℓ_2 verse $CFL = .5, 1,$ and 2.

Example 2 (Advection Equation):

We consider the case of the advection equation with $c(x) = x - x^3$ and $u(x, 0) = exp(-10x^2)$. And so (2.1) becomes

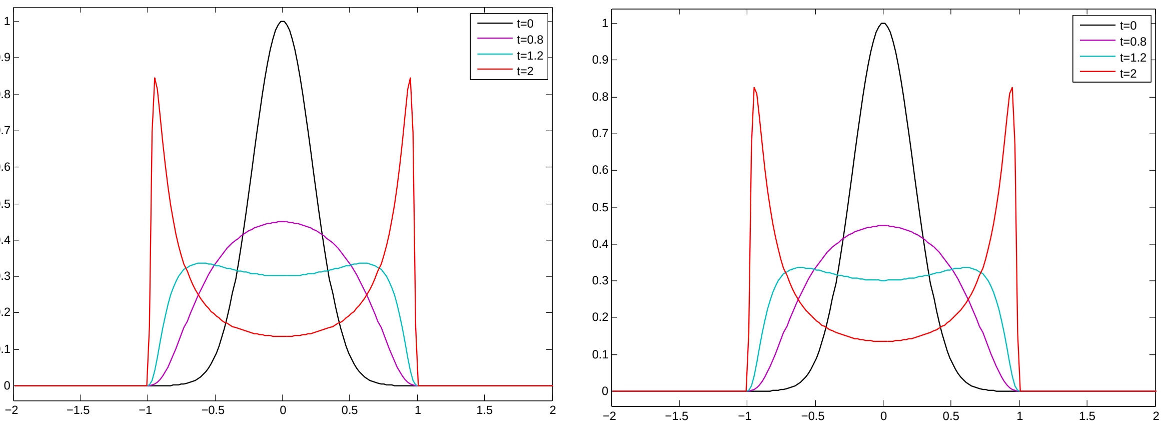
$$u_t + ((x - x^3)u)_x = 0 \tag{2.29}$$

where $x \in [-2, 2]$ and $t > 0$. The analytic solution to (2.29) is given by

$$u(x,t) = \frac{e^{\dfrac{(-2t + 2tx^2)e^{2t} - 2x^2(-5+t)}{-x^2 - e^{2t} + e^{2t}x^2}}}{\sqrt{\dfrac{x^2 + e^{2t} - e^{2t}x^2}{x^2}((-x + x^3)e^{2t} - x^3)}}.$$

We apply our CIP scheme with the multi-moment piecewise linear and cubic profiles. We calculated numerical solutions with $\Delta x = \Delta t = .02$ at various times. In Figure 2.7 we see that the discontinuities that form are effectively captured. We observe in Table 2.3 that the error associated with each of the profiles are comparable however the convergence rate for the piecewise linear achieves second order spatial convergence while the cubic is sub-second order. This is a nonsmooth problem so we expect a decrease in performance for the cubic profile.

In the CIP scheme we have presented three types of profiles to accompany the solution



(a) (b)

Figure 2.7: Numerical solution $u(x,t)$ using (a) the multi-moment piecewise linear profile (2.18) and (b) the multi-moment cubic profile (2.22) for $t = 0, 0.8, 1.2$, and 2.

Table 2.3: Error in numerical solutions for advection equation in ℓ_2.

N	100	200	400	800
Piecewise Linear	3.9877e-4	7.9853e-5	1.9647e-5	5.6754e-6
Cubic	1.1027e-4	3.2622e-5	1.1291e-5	4.4563e-6

and solution gradient update formulas we developed. Using a multi-moment method we are able to better capture the formation of discontinuities. For the transport equation the multi-moment cubic profile achieves the highest accuracy. For the advection equation the multi-moment piecewise cubic achieves a comparable error to the cubic profile. One of the advantages of the multi-moment piecewise cubic profile is that it will not oscillate when discontinuities form unlike the multi-moment cubic profile for coarse grids.

Chapter 3

CIP Methods for Hyperbolic Equations in Two Dimensions

3.1 Two Dimensional Updates

To extend our method to two dimensions we will consider two strategies. First we consider a direct extension of our method, i.e., develop exact time integration in two dimensions for the transport equation and the advection equation. We follow the same steps as the one dimensional case starting with applying the method of characteristics on the full system to develop solution and gradient update formulas. We then apply a two dimensional profile to evaluate these update formulas. The second strategy is to use an operator splitting method for integrating the transport equation in time thus this strategy uses the CIP profiling developed in Chapter 2 for the one dimensional case.

3.1.1 Two Dimensional Transport

Consider the two dimensional transport equation

$$\frac{\partial u}{\partial t} + \vec{c}(x)u_x = 0 \tag{3.1}$$

where

$$\vec{c} = \begin{pmatrix} a(x_1(t), x_2(t)) \\ b(x_1(t), x_2(t)) \end{pmatrix}$$

for $t > 0$, $t \in \mathbb{R}$ and $x \in \mathbb{R}^2$. To directly extend our method we first apply the method of characteristics with $U(t) = u(x_1(t), x_2(t), t)$. Taking the derivative of $U(t)$ gives

$$\begin{aligned} \frac{dU(t)}{dt} &= \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x_1}\frac{dx_1}{dt} + \frac{\partial u}{\partial x_2}\frac{dx_2}{dt} \text{ (by the chain rule)} \\ &= \frac{\partial u}{\partial t} + a(x_1, x_2)\frac{\partial u}{\partial x_1} + b(x_1, x_2)\frac{\partial u}{\partial x_2} \qquad (3.2) \\ &= 0 \end{aligned}$$

by (3.1). Thus we obtain the update formula

$$U(t + \Delta t) = u(x_1, x_2, t + \Delta t) = u(y_1, y_2, t) \qquad (3.3)$$

where (y_1, y_2) are determined by the backward characteristic depicted by Figure 3.1 for the system (x_1, x_2):

$$\frac{d}{dt}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a(x_1, x_2) \\ b(x_1, x_2) \end{pmatrix} \qquad (3.4)$$

where

$$x(0) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = x(-\Delta t).$$

Now we determine the update formula for the gradient (u_{x_1}, u_{x_2}). Define $V_1(t) = u_{x_1}(x_1(t), x_2(t), t)$ and $V_2(t) = u_{x_2}(x_1(t), x_2(t), t)$ along the characteristics. Taking the derivative of (3.1) with respect to x_1 gives

$$\frac{\partial u_{x_1}}{\partial t} + a(x_1, x_2)(u_{x_1})_{x_1} + b(x_1, x_2)(u_{x_1})_{x_2} + a_{x_1}(x_1, x_2)u_{x_1} + b_{x_1}(x_1, x_2)u_{x_2} = 0. \qquad (3.5)$$

Similarly taking the derivative of (3.1) with respect to x_2 gives

$$\frac{\partial u_{x_2}}{\partial t} + a(x_1, x_2)(u_{x_2})_{x_1} + b(x_1, x_2)(u_{x_2})_{x_2} + a_{x_2}(x_1, x_2)u_{x_1} + b_{x_2}(x_1, x_2)u_{x_2} = 0. \qquad (3.6)$$

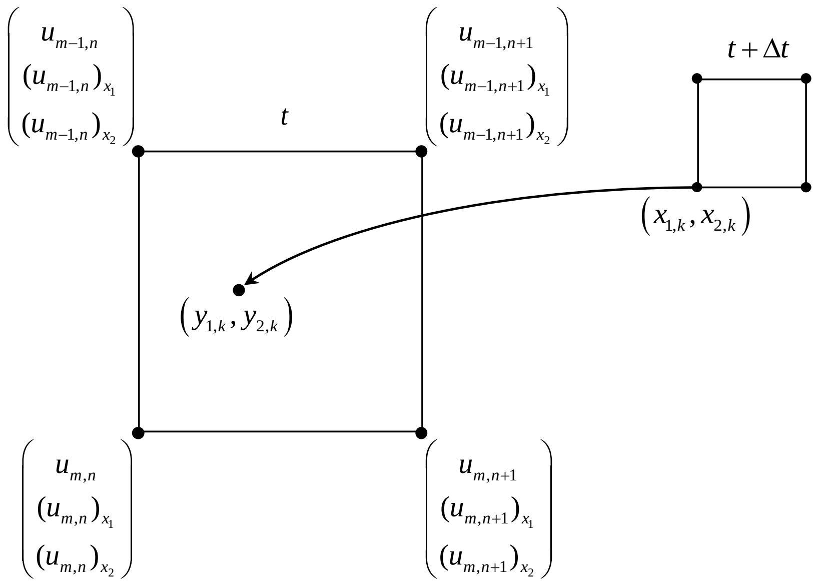

$\begin{pmatrix} u_{m-1,n} \\ (u_{m-1,n})_{x_1} \\ (u_{m-1,n})_{x_2} \end{pmatrix}$
t
$\begin{pmatrix} u_{m-1,n+1} \\ (u_{m-1,n+1})_{x_1} \\ (u_{m-1,n+1})_{x_2} \end{pmatrix}$
$t+\Delta t$
$(x_{1,k}, x_{2,k})$
$(y_{1,k}, y_{2,k})$
$\begin{pmatrix} u_{m,n} \\ (u_{m,n})_{x_1} \\ (u_{m,n})_{x_2} \end{pmatrix}$
$\begin{pmatrix} u_{m,n+1} \\ (u_{m,n+1})_{x_1} \\ (u_{m,n+1})_{x_2} \end{pmatrix}$

Figure 3.1: Diagram of backward characteristics for system (3.4) in two dimensions.

Thus,

$$\begin{aligned} \frac{d}{dt}V_1 &= \frac{\partial}{\partial t}u_{x_1} + \frac{\partial u_{x_1}}{\partial x_1}\frac{d}{dt}x_1(t) + \frac{\partial u_{x_1}}{\partial x_2}\frac{d}{dt}x_2(t) \text{ by the chain rule} \\ &= -(a_{x_1}(x_1,x_2)u_{x_1} + b_{x_1}(x_1,x_2)u_{x_2}) \text{ at } (x_1(t),x_2(t),t) \text{ by (3.5)} \end{aligned} \tag{3.7}$$

and

$$\begin{aligned} \frac{d}{dt}V_2 &= \frac{\partial}{\partial t}u_{x_2} + \frac{\partial u_{x_2}}{\partial t}\frac{d}{dt}x_1(t) + \frac{\partial u_{x_2}}{\partial x_2}\frac{d}{dt}x_2(t) \text{ by the chain rule} \\ &= -(a_{x_2}(x_1,x_2)u_{x_1} + b_{x_2}(x_1,x_2)u_{x_2}) \text{ at } (x_1(t),x_2(t),t) \text{ by (3.6).} \end{aligned} \tag{3.8}$$

We write (3.7) and (3.8) as a system of equations for $(V_1(t), V_2(t))$:

$$\frac{d}{dt}\begin{pmatrix} V_1(t) \\ V_2(t) \end{pmatrix} + J(t)\begin{pmatrix} V_1(t) \\ V_2(t) \end{pmatrix} = 0 \tag{3.9}$$

where

$$J(t) = \begin{pmatrix} a_{x_1}(x_1(t), x_2(t)) & b_{x_1}(x_1(t), x_2(t)) \\ a_{x_2}(x_1(t), x_2(t)) & b_{x_2}(x_1(t), x_2(t)) \end{pmatrix}.$$

Note that $(V_1(t), V_2(t)) = P(t)(V_1(0), V_2(0))$ where the transition matrix $P(t) \in R^{2\times 2}$ satisfies

$$\frac{d}{dt}P(t) + J(t)P(t) = 0.$$

To solve this system we define $X = (x_1, \cdots, x_6)$ by

$(X_1(t), X_2(t)) = (x_1(t), x_2(t)),\ X_3(t) = P_{11}(t),\ X_4(t) = P_{12}(t),\ X_5(t) = P_{21},$ and $X_6(t) = P_{22}(t).$

Then the equation for $X_t(t)$ is

$$\frac{d}{dt}X = \begin{pmatrix} a(X_1, X_2) \\ b(X_1, X_2) \\ -a_{x_1}(X_1, X_2)X_3 - b_{x_1}(X_1, X_2)X_5 \\ -a_{x_1}(X_1, X_2)X_4 - b_{x_1}(X_1, X_2)X_6 \\ -a_{x_2}(X_1, X_2)X_3 - b_{x_2}(X_1, X_2)X_5 \\ -a_{x_2}(X_1, X_2)X_4 - b_{x_2}(X_1, X_2)X_6 \end{pmatrix} \tag{3.10}$$

with the initial conditions $(X_1(0), X_2(0)) = (x_1, x_2)$ and $X_3(0) = 1;\ X_4(0) = 0;\ X_5(0) = 0;\ X_6(0) = 1$. Let $Y = X(-\Delta t) = (y_1, ...y_6)$ satisfy (3.10), then the updates for u is given by (3.3) and the update for u_{x_1} and u_{x_2} are given by

$$\begin{pmatrix} u_{x_1}(x_1, x_2, t+\Delta t) \\ u_{x_2}(x_1, x_2, t+\Delta t) \end{pmatrix} = \begin{pmatrix} y_3 & y_4 \\ y_5 & y_6 \end{pmatrix}^{-1} \begin{pmatrix} u_{x_1}(y_1, y_2, t) \\ u_{x_2}(y_1, y_2, t) \end{pmatrix}. \tag{3.11}$$

3.1.2 Two Dimensional Advection

We now consider the system for the advection equation

$$\frac{\partial p}{\partial t} + \frac{\partial}{\partial x}\left(\vec{c}(x)p(x_1(t), x_2(t), t)\right) = 0$$

or alternatively

$$\frac{\partial p}{\partial t} + \vec{c}(x)p_x + div(\vec{c}(x))p = 0 \tag{3.12}$$

for $t > 0$, $t \in \mathbb{R}$ and $x \in \mathbb{R}^2$. We will apply the method of characteristics with $P(t) = p(x_1(t), x_2(t), t)$. Taking the derivative of $P(t)$ gives

$$\begin{aligned}
\frac{dP(t)}{dt} &= \frac{\partial p}{\partial t} + \frac{\partial p}{\partial x_1}\frac{dx_1}{dt} + \frac{\partial p}{dx_2}\frac{dx_2}{dt} \\
&= \frac{\partial P}{\partial t} + a(x_1,x_2)\frac{\partial p}{\partial x_1} + b(x_1,x_2)\frac{\partial p}{\partial x_2} \\
&= -div(\vec{c})P(t).
\end{aligned}$$

Define $x_3(t)$ by

$$\frac{d}{dt}x_3(t) = -div(\vec{c}(x_1(t), x_2(t))x_3(t), \quad x_3(0) = 1 \tag{3.13}$$

and let y_3 be the solution to (3.13) then we obtain the solution update formula

$$P(t) = p(x_1, x_2, t + \Delta t) = y_3 p(y_1, y_2, t). \tag{3.14}$$

Next we determine the update for the gradient (p_{x_1}, p_{x_2}). Define $W_1(t) = \frac{\partial}{\partial x_1}p(x_1(t), x_2(t), t)$ and $W_2(t) = \frac{\partial}{\partial x_2}p(x_1(t), x_2(t), t)$ along the characteristics. From (3.12) p_{x_1} satisfies

$$\begin{aligned}
&\frac{\partial p_{x_1}}{\partial t} + a(x_1,x_2)(p_{x_1})_{x_1} + b(x_1,x_2)(p_{x_1})_{x_2} + a_{x_1}(x_1,x_2)p_{x_1} + b_{x_1}(x_1,x_2)p_{x_2} \\
&\qquad + div(\vec{c}(x))_{x_1}p + div(\vec{c}(x))p_{x_1} = 0
\end{aligned} \tag{3.15}$$

and p_{x_2} satisfies

$$\begin{aligned}
&\frac{\partial p_{x_2}}{\partial t} + a(x_1,x_2)(p_{x_2})_{x_1} + b(x_1,x_2)(p_{x_2})_{x_2} + a_{x_2}(x_1,x_2)p_{x_1} + b_{x_2}(x_1,x_2)p_{x_2} \\
&\qquad + div(\vec{c}(x))_{x_2}p + div(\vec{c}(x))p_{x_2} = 0.
\end{aligned} \tag{3.16}$$

Using the chain rule and applying (3.15) we have

$$\begin{aligned}
\frac{dW_1}{dt} &= \frac{\partial}{\partial t}p_{x_1} + \frac{\partial p_{x_1}}{\partial x_1}\frac{dx}{dt} + \frac{\partial p_{x_1}}{\partial x_2}\frac{dx_2}{dt} \\
&= \frac{\partial}{\partial t}p_{x_1} + a(x_1,x_2)(p_{x_1})_{x_1} + b(x_1,x_2)(p_{x_1})_{x_2} \\
&= -(a_{x_1}(x_1,x_2)p_{x_1} + b_{x_1}(x_1,x_2)p_{x_2} + (div(\vec{c}(x)))_{x_1}p + div(\vec{c}(x))p_{x_1})
\end{aligned}$$

and similarly by (3.16) we have

$$\begin{aligned}
\frac{dW_2}{dt} &= \frac{\partial}{\partial t}p_{x_2} + \frac{\partial p_{x_2}}{\partial x_1}\frac{dx_1}{dt} + \frac{\partial p_{x_2}}{\partial x_2}\frac{dx_2}{dt} \\
&= \frac{\partial}{\partial t}p_{x_2} + a(x_1,x_2)(p_{x_2})_{x_1} + b(x_1,x_2)(p_{x_2})_{x_2} \\
&= -(a_{x_2}(x_1,x_2)p_{x_1} + b_{x_2}(x_1,x_2)p_{x_2} + (div(\vec{c}(x)))_{x_2}p + div(\vec{c}(x))p_{x_2}).
\end{aligned}$$

Thus

$$\frac{d}{dt}\begin{pmatrix} W_1 \\ W_2 \end{pmatrix} + J(t)\begin{pmatrix} W_1 \\ W_2 \end{pmatrix} + \begin{pmatrix} (div(\vec{c}(x)))_{x_1} \\ (div(\vec{c}(x)))_{x_2} \end{pmatrix} P + div(\vec{c}(x))\begin{pmatrix} W_1 \\ W_2 \end{pmatrix} = 0 \tag{3.17}$$

where

$$J(t) = \begin{pmatrix} a_{x_1}(x_1(t),x_2(t)) & b_{x_1}(x_1(t),x_2(t)) \\ a_{x_2}(x_1(t),x_2(t)) & b_{x_2}(x_1(t),x_2(t)) \end{pmatrix}.$$

Define the fundamental solution T for this system by

$$\frac{dT(t)}{dt} + \begin{pmatrix} div(\vec{c}(x)) & 0 & 0 \\ (div(\vec{c}(x)))_{x_1} & a_{x_1}(x) + div(\vec{c}(x)) & b_{x_1}(x) \\ (div(\vec{c}(x)))_{x_2} & a_{x_2}(x) & b_{x_2}(x) + div(\vec{c}(x)) \end{pmatrix} T(t) = 0$$

where $x = (x_1(t), x_2(t))$. Then we have

$$\begin{pmatrix} P(t) \\ W_1(t) \\ W_2(t) \end{pmatrix} = T(t)\begin{pmatrix} 1 \\ W_1(0) \\ W_2(0) \end{pmatrix}.$$

To complete update we define $X = (X_1, \cdots, X_9)$ by

$$(X_1(t), X_2(t)) = (x_1(t), x_2(t)),\ X_3(t) = P(t)$$

$$X_4 = T_{22}(t),\ X_5(t) = T_{23}(t),\ X_6(t) = T_{32},\ X_7(t) = T_{33}(t),\ X_8(t) = T_{21}(t),\ X_9(t) = T_{31}(t)$$

and the equation for $\frac{d}{dt}X(t)$ is:

$$X_t = \begin{pmatrix} a(X_1,X_2) \\ b(X_1,X_2) \\ -div(\vec{c}(X_1,X_2)))X_3 \\ -(a_{x_1}(X_1,X_2)+div(\vec{c}(X_1,X_2)))X_4 - b_{x_1}(X_1,X_2)X_6 \\ -(a_{x_1}(X_1,X_2)+div(\vec{c}(X_1,X_2)))X_5 - b_{x_1}(X_1,X_2)X_7 \\ -a_{x_2}(X_1,X_2)X_4 - (b_{x_2}(X_1,X_2)+div(\vec{c}(X_1,X_2)))X_6 \\ -a_{x_2}(X_1,X_2)X_5 - (b_{x_2}(X_1,X_2)+div(\vec{c}(X_1,X_2)))X_7 \\ -div(\vec{c}(X_1,X_2))_{x_1}X_3 - (a_{x_1}(X_1,X_2)+div(\vec{c}(X_1,X_2)))X_8 - b_{x_1}(X_1,X_2)X_9 \\ -div(\vec{c}(X_1,X_2))_{x_2}X_3 - a_{x_2}(X_1,X_2)X_8 - (b_{x_2}(X_1,X_2)+div(\vec{c}(X_1,X_2)))X_9 \end{pmatrix} \tag{3.18}$$

with $(X_1(0),X_2(0)) = (x_1,x_2)$ and $X_4(0) = X_7(0) = 1;\ X_5(0) = X_6(0) = X_8(0) = X_9(0) = 0$. Let $Y = X(-\Delta t) = (y_1, ..., y_9)$ satisfy (3.18). Then the updates for p_{x_1} and p_{x_2} are given by the system

$$\begin{pmatrix} p_{x_1}(x_1,x_2,t+\Delta t) \\ p_{x_2}(x_1,x_2,t+\Delta t) \end{pmatrix} = \begin{pmatrix} y_4 & y_5 \\ y_6 & y_7 \end{pmatrix}^{-1} \begin{pmatrix} p_{x_1}(y_1,y_2,t) - \frac{y_8}{y_3}p(y_1,y_2,t) \\ p_{x_2}(y_1,y_2,t) - \frac{y_9}{y_3}p(y_1,y_2,t) \end{pmatrix}. \tag{3.19}$$

3.1.3 Operator Splitting

The approach developed in section 3.1.1 for the transport equation significantly increases the complexity of our one dimensional method and may therefore not be advisable or desirable for higher dimensions (more than two). For this reason we also extend our method via time splitting integration for the transport equation which maintains the simplicity of our method in higher dimensions to reduce the operations while preserving the accuracy. This procedure can be viewed as a divide an conquer tactic, i.e., for each direction of motion we create independent update formulas. For the two dimensional case we will first consider the x_1 direction while assuming x_2 is constant. After developing the formulas for the x_1 direction we will hold x_1 constant to develop update formulas for the x_2 direction. To begin first we assume $x_2(t) = s$ where s is a constant then (3.2) becomes

$$\frac{dU(t)}{dt} = \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x_1}\frac{dx_1}{dt} = 0$$

which is equivalent to (2.2) and so the solution update in x_1 direction is

$$u(x_1,s,t+\Delta t) = u(y_1,s,t) \tag{3.20}$$

where y_1 is the solution to the characteristic equation $\frac{d}{dt}x_1(t) = a(x_1, s)$. To update the derivative value for u_{x_1} in the x_1 direction we observe

$$\frac{\partial u_{x_1}}{\partial t} + a(x_1, s)\frac{\partial u_{x_1}}{\partial x_1} + a_{x_1}(x_1, s)u_{x_1} = 0$$

which is equivalent to (2.5) and so

$$u_{x_1}(x_1, s, t + \Delta t) = u_{x_1}(y_1, s, t)\frac{a(y_1, s)}{a(x_1, s)}. \tag{3.21}$$

To update u_{x_2} in the x_1 direction we take the derivative of (3.1) with respect to x_2 to get

$$\frac{\partial u_{x_2}}{\partial t} + a(x_1, s)\frac{\partial u_{x_2}}{\partial x_1} + a_{x_2}(x_1, s)u_{x_1} = 0. \tag{3.22}$$

Now let $V(t) = u_{x_2}(x_1, s, t)$ then

$$\begin{aligned} \frac{dV}{dt} &= \frac{\partial u_{x_2}}{\partial t} + \frac{\partial u_{x_2}}{\partial x_1}\frac{dx_1}{dt} \\ &= \frac{\partial u_{x_2}}{\partial t} + a(x_1, s)\frac{\partial u_{x_2}}{\partial x_1} \\ &= -a_{x_2}(x_1, s)u_{x_1} \end{aligned} \tag{3.23}$$

by (3.22) which can be evaluated by and ODE solver. Alternatively we can apply a quadrature rule to

$$V(t + \Delta t) - V(t) = -\int_t^{t+\Delta t} a_{x_2}(x_1, s)u_{x_1} dt. \tag{3.24}$$

For example if we apply the trapezoidal rule then (3.24) becomes

$$u_{x_2}(x_1, s, t + \Delta t) = u_{x_2}(x_1, s, t) - \frac{\Delta t}{2}(a_{x_2}(x_1, s)u_{x_1}(x_1, s, t) + a_{x_2}(\hat{x}_1, s)u_{x_1}(\hat{x}_1, s, t + \Delta t))$$

where $\hat{x}_1 = x_1(t + \Delta t)$. We now have the formulas for updating the solution and solution derivatives in the x_1 direction. We repeat the same procedure to get the update formulas in the x_2 direction when $x_1(t) = \tilde{s}$ is constant then (3.2) becomes

$$\frac{dU(t)}{dt} = \frac{\partial u}{\partial t} + \frac{\partial u}{\partial x_2}\frac{dx_2}{dt} = 0$$

which is equivalent to (2.2) and so the solution update in x_2 direction is

$$u(\tilde{s}, x_2, t + \Delta t) = u(\tilde{s}, y_2, t) \tag{3.25}$$

where y_2 is the solution to the characteristic equation $\frac{d}{dt}x_2(t) = a(\tilde{s}, x_2)$. To update the derivative value u_{x_2} in the x_2 direction we observe

$$\frac{\partial u_{x_2}}{\partial t} + b(\tilde{s}, x_2)\frac{\partial u_{x_2}}{\partial x_2} + b_{x_2}(\tilde{s}, x_2)u_{x_2} = 0$$

which is equivalent to (2.5) and so

$$u_{x_2}(\tilde{s}, x_2, t + \Delta t) = u_{x_2}(\tilde{s}, x_2, t)\frac{b(\tilde{s}, y_2)}{b(\tilde{s}, x_2)}. \tag{3.26}$$

To update u_{x_1} in the x_2 direction we take the derivative of (3.1) with respect to x_1 to get

$$\frac{\partial u_{x_1}}{\partial t} + b(\tilde{s}, x_2)\frac{\partial u_{x_1}}{\partial x_2} + b_{x_1}(\tilde{s}, x_2)u_{x_2} = 0. \tag{3.27}$$

Now let $V(t) = u_{x_1}(\tilde{s}, x_2, t)$ then

$$\begin{aligned} \frac{dV}{dt} &= \frac{\partial u_{x_1}}{\partial t} + \frac{\partial u_{x_1}}{\partial x_2}\frac{dx_2}{dt} \\ &= \frac{\partial u_{x_1}}{\partial t} + b(\tilde{s}, x_2)\frac{\partial u_{x_1}}{\partial x_2} \\ &= -b_{x_1}(\tilde{s}, x_2)u_{x_2} \end{aligned} \tag{3.28}$$

by (3.27) which can be evaluated by and ODE solver. We now have the update formulas for the function and gradient values using two different techniques. For direct extension of our method in sections 3.1.1 and 3.1.2 we need to develop two dimensional profiling techniques. For the operator splitting technique we can utilize the one dimensional profiling techniques developed for the one dimensional case in Chapter 2.

3.2 CIP Profiling

To evaulate $u(y_{1,k}, y_{2,k})$ used in the update formulas developed in 3.1.1 and 3.1.2 we develop a two dimensional profiling technique. For each square cell we have the data

$(u_k, (u_k)_{x_1}, (u_k)_{x_2})$ at each corner as depicted in Figure 3.1. We use the Adini polynomials defined by

$$\phi(x,y) = \sum_{x^k y^\ell \in P} a_{k\ell} x_1^k x_2^\ell \tag{3.29}$$

where

$$P = \{1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3, x_1^3x_2, x_1x_2^3\}.$$

We are able to solve for each $a_{k\ell}$ uniquely since we have twelve nodal values and twelve coefficients. For example when $\Delta x = 1$ we have the map M from the coefficient space to the solution space and the map M^{-1} from the solution space to the coefficient space given by

$$M = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&0&0\\ 1&1&0&1&0&0&1&0&0&0&0&0\\ 0&1&0&2&0&0&3&0&0&0&0&0\\ 0&0&1&0&1&0&0&1&0&0&1&0\\ 1&0&1&0&0&1&0&0&0&1&0&0\\ 0&1&0&0&1&0&0&0&1&0&0&1\\ 0&0&1&0&0&2&0&0&0&3&0&0\\ 1&1&1&1&1&1&1&1&1&1&1&1\\ 0&1&0&2&1&0&3&2&1&0&3&1\\ 0&0&1&0&1&2&0&1&2&3&1&3 \end{bmatrix},$$

and

$$M^{-1} \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&0&0\\ -3&-2&0&3&-1&0&0&0&0&0&0&0\\ -1&-1&-1&1&0&1&1&1&0&-1&0&0\\ -3&0&-2&0&0&0&3&0&-1&0&0&0\\ 2&1&0&-2&1&0&0&0&0&0&0&0\\ 3&2&0&-3&1&0&-3&-2&0&3&-1&0\\ 3&0&2&-3&0&-2&-3&0&1&3&0&-1\\ 2&0&1&0&0&0&-2&0&1&0&0&0\\ -2&-1&0&2&-1&0&2&1&0&-2&1&0\\ -2&0&-1&2&0&1&2&0&-1&-2&0&1 \end{bmatrix}.$$

Using (3.29) we can compute interpolated the value of $u(y_{1,k}, y_{2,k}, t), u_{x_1}(y_{1,k}, y_{2,k}, t)$ and $u_{x_2}(y_{1,k}, y_{2,k}, t)$. Our updates are now are given by

$$u_k^{n+1}(x_{1,k}, x_{2,k}, t) = \phi_k(y_{1,k}, y_{2,k}) \tag{3.30}$$

and

$$(u_{x_1})_k^{n+1} = \alpha(y_6\phi_k'(y_{1,k}, y_{2,k}) - y_4\phi_k'(y_{1,k}, y_{2,k})) \tag{3.31}$$

and

$$(u_{x_2})_k^{n+1} = \alpha(-y_5\phi_k'(y_{1,k}, y_{2,k}) + y_3\phi_k'(y_{1,k}, y_{2,k})) \tag{3.32}$$

where $\alpha = (y_3y_6 - y_4y_5)^{-1}$.

3.3 Two Dimensional CIP Numerical Results

In this section we consider some numerical examples to test our methods.

Example 1:

We consider the transport equation given by

$$u_t + x_2u_{x_1} + x_1u_{x_2} = 0 \tag{3.33}$$

To compute this solution we use a Strang splitting (time operator splitting) scheme where we first update our solution where for each time step we first update the x direction for half a time step, update in the y direction for a full time step, and lastly update in the x-direction for another half time step, ie, for each step

$$S = S_{x^{(1/2)}}S_yS_{x^{(1/2)}}.$$

Using the update formulas developed in section 3.1.3 we consider the case when $u(x, y, 0) = exp(-10(x^2 + y^2)$. Then (3.33) has the analytic solution given by

$$u(x,y,t) = exp\left(-10\left(\left(-\frac{1}{2}\frac{(y-x)e^t - (x+y)e^{-t}}{e^te^{-t}}\right)^2 + \left(\frac{1}{2}\frac{(y-x)e^t + (y+x)e^{-t}}{e^te^{-t}}\right)^2\right)\right).$$

For the numerical solutions we use $\Delta x = \Delta t = .04$ (for this method the CFL condition must be met). In Figure 3.2(a) we see the initial distribution and in Figure 3.2(b), Figure 3.2(c), and Figure 3.2(d) we can see the distribution contract and blade. At $t = 1.2$ we see the solution in Figure 3.2 as well as 1-D slice in Figure 3.2(f) which has ℓ_2 error 0.0035 in at $t = 1.2$.

Example 2:

We consider the rigid body rotation given by

$$\frac{\partial u}{\partial t} + x_2 \frac{\partial u}{\partial x_1} - x_1 \frac{\partial u}{\partial x_2} = 0 \tag{3.34}$$

The initial condition we used is in Figure 3.3(a) is a smoothed slotted disk which is discontinuous at $1.4^2 - (x^2 + y^2) = 0$ on a 100×100 mesh. For this example problem we use our method developed in section 3.1.1 which has a smoothness requirement so we are using a smoothed domain instead of the standard slotted disk. The numerical solution when $\Delta x = .01$ and $\Delta t = 2\pi/200 \approx .0157$ (we have no CFL condition for this method) after four and eight revolutions is in Figure 3.3(b) and Figure 3.3(c) respectively, these figures are meshes oriented to display the solution features. In these figures we observe slight numerical dispersion in our solution. In Figure 3.3(d) we take a one dimensional slice of our solution we can see oscillatory behavior near the discontinuities which as we are using cubic polynomials is to be expected. The ℓ^2 error this slice is .0048 so we are able to capture the behavior of this discontinuous domain with a high degree of accuracy using this method.

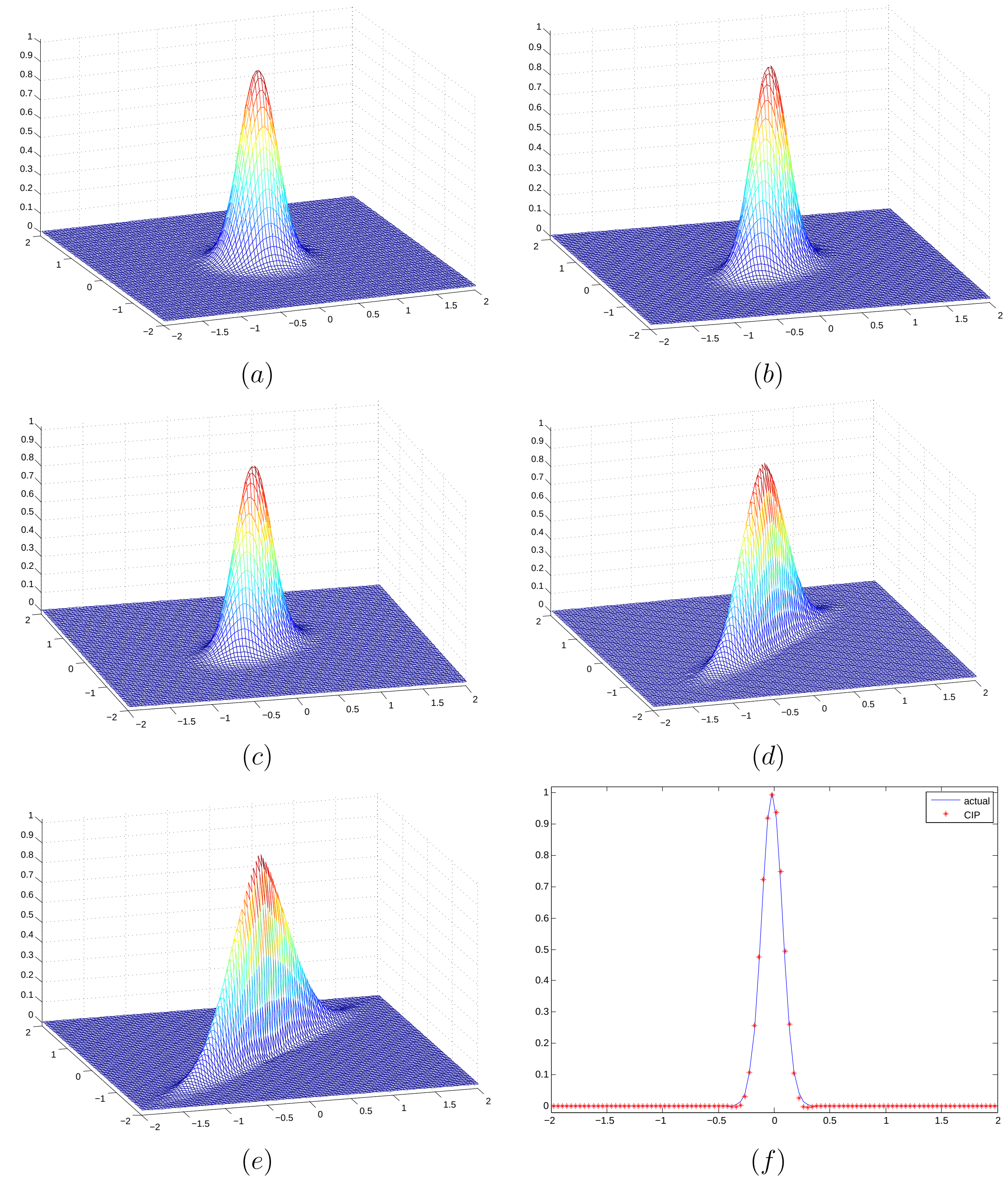


Figure 3.2: (a) Initial conditions (b) $t = .3$ (c) $t = .6$ (d) t=.9, and (e) $t = 1.2$. (f) is a 1-D cut at $x = -.02$ and $y = -2$ to 2 when $t = 1.2$ for example 1.

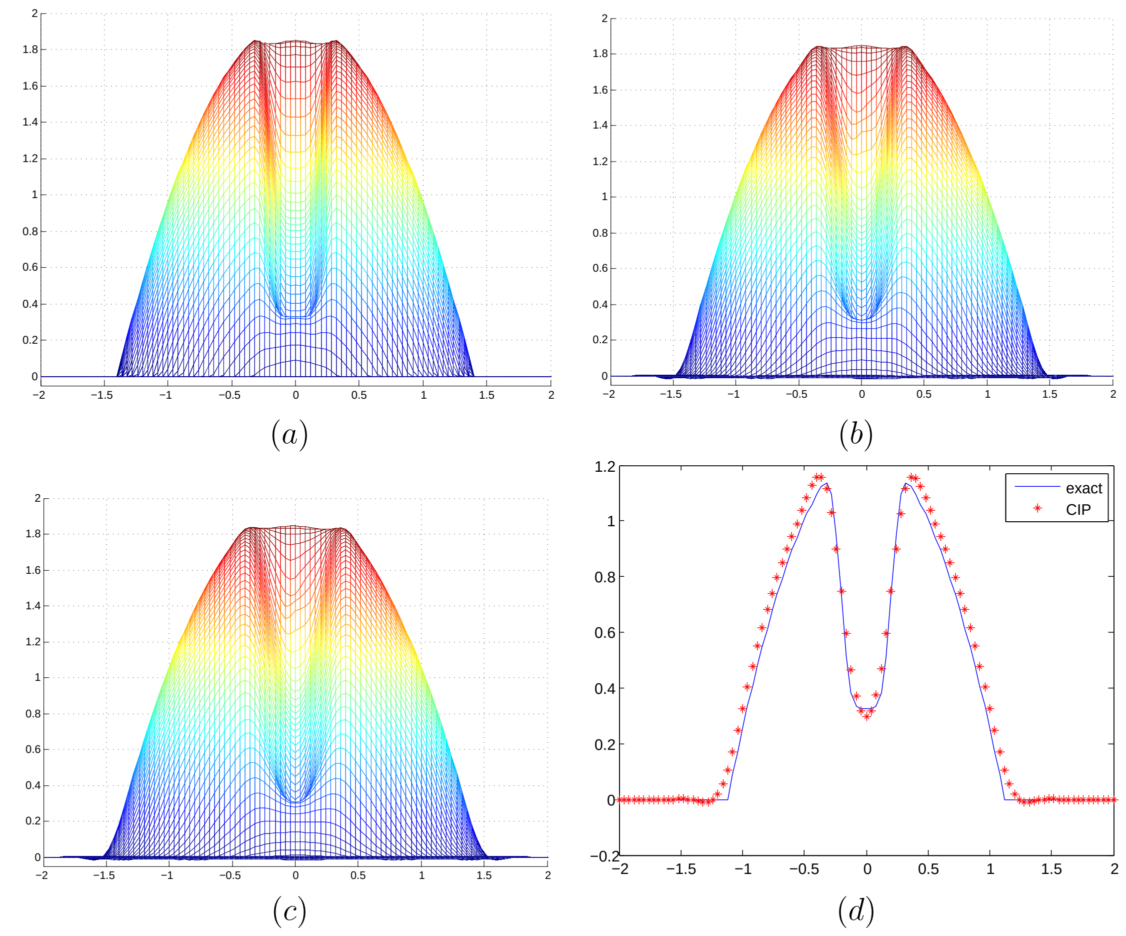


Figure 3.3: (a) Initial conditions (b) $t = 8\pi$ (c) $t = 16\pi$ (d) 1-D cut at $x = -.84$ and $y = -2$ to 2 when $t = 8\pi$ for example 2.

Chapter 4

Nonsmooth Optimization for the Elastic Contact Problem

In this chapter we discuss nonsmooth optimization methods with application to the elastic contact problem with friction in two dimensions. The Signorini contact problem with Coulomb friction minimizes the elastic deformation energy in the normal and tangential directions of an applied force. We begin by introducing the problem and discussing optimization techniques. We then apply the optimization techniques developed to the linear elastic contact problem.

4.1 Model Description

Contact problems with friction arise in many important industrial and engineering applications for example material failure and load bearing. We will consider the mechanical contact between an elastic body and a rigid body which is modeled by the Signorini contact problem with Coulomb friction [11, 4]. The Signorini contact models the constrained motion in the normal direction of the elastic body due to its contact with the rigid body while the Coulomb friction models the tangential frictional force in the contact region. We begin with the full contact problem;

$$\begin{aligned} -div\sigma &= f \text{ in } \Omega, \\ \sigma n|_{\Gamma_0} &= g, \\ u|_{\Gamma_1} &= 0, \end{aligned} \tag{4.1}$$

$$u_n - d \leq 0, \quad \sigma_N \geq 0, \quad (u_n - d)\sigma_N = 0 \tag{4.2}$$

$$\begin{gathered} |\sigma_\tau u| \leq \mathcal{F}|\sigma_N|, \quad \text{if } u_\tau = 0 \\ \sigma_\tau = -\mathcal{F}\sigma_N \frac{u_\tau}{|u_\tau|}, \quad u_\tau \neq 0 \end{gathered} \tag{4.3}$$

where Ω is the domain on which the deformation vector u is defined, Γ the boundary of Ω, d is the gap between the elastic body and rigid body, u_n is the normal displacement, u_τ is the tangential displacement, σn is the normal stress, σ_N is the normal component of the normal stress, σ_τ is the tangential component of the normal stress, and $\mathcal{F}$ is the coefficient of friction. We assume linear elasticity strain tensor;

$$\epsilon = \frac{1}{2}\left(\nabla u + (\nabla u)^T\right) \tag{4.4}$$

or equivalently

$$\epsilon = \begin{pmatrix} \dfrac{\partial u_1}{\partial x_1} & \frac{1}{2}\left(\dfrac{\partial u_2}{\partial x_1} + \dfrac{\partial u_1}{\partial x_2}\right) \\ \frac{1}{2}\left(\dfrac{\partial u_2}{\partial x_1} + \dfrac{\partial u_1}{\partial x_2}\right) & \dfrac{\partial u_2}{\partial x_2} \end{pmatrix}$$

for the two dimension case. We use Hooke's law for the strain stress-strain relationship,

$$\sigma = 2\mu\epsilon + \lambda tr\epsilon I \tag{4.5}$$

where σ is the strain tensor, and μ and λ are the Lamé constants. The Lamé constants may be computed using

$$\lambda = \frac{(E\nu)}{(1+\nu)(1-2\nu)} \text{ and } \mu = \frac{E}{2(1+\nu)}$$

where $E > 0$ is Young's modulus and $\nu \in (0, 0.5)$ is the Poisson ration. The full contact problem has no direct variational formulation and there is no guaranteed existence and uniqueness for large friction coefficients.

To develop our numerical optimization methods we first review the variational formulation of the Signorini problem and Coulomb friction problem. The Lagrange multiplier theory [4] is used and the necessary optimality and sufficient optimality conditions are written as complementarity systems. We develop a semi-smooth Newton method for both complementarity systems for the Signorini problem and the Coulomb friction problem. They are of the form of Primal-Dual Active Set methods. The semi-smooth method is a very effective method for nonsmooth optimization problems and converges super linearly (for finite dimensional problems the active set methods convergence in finite steps). Combining these two schemes we develop an algorithm to solve the full elastic contact problem with minimized elastic deformation. In the next chapter we will describe a discretization technique to solve the problem numerically as well as providing a test example.

4.2 Variational Formulation of Signorini and Coulomb Friction Problems

In this section we begin by reviewing the formulation of the variational Signorini and Coulomb friction problems. For both of these problems we apply numerical optimization methods based on semismooth Newton methods. To begin we integrate (4.2) against test functions $\phi, \phi_2 \in H^1(\Omega)$ to get the weak form

$$-\int_\Omega div\sigma \cdot \begin{pmatrix}\phi_1\\ \phi_2\end{pmatrix} = \int_\Omega \vec{f} \cdot \begin{pmatrix}\phi_1\\ \phi_2\end{pmatrix} \tag{4.6}$$

where

$$\begin{aligned}\int_\Omega div\sigma \cdot \begin{pmatrix}\phi_1\\ \phi_2\end{pmatrix} &= \int_\Omega div(\sigma_{11},\sigma_{12})\phi_1 dx + \int_\Omega div(\sigma_{12},\sigma_{22})\phi_2 dx\\ &= -\int_\Omega (\sigma_{11},\sigma_{22})\cdot\triangledown\phi_1 - \int_\Omega(\sigma_{12},\sigma_{22})\cdot\triangledown\phi_2\\ &\quad + \int_\Gamma \vec{n}\cdot(\sigma_{11},\sigma_{22})\phi_1 ds + \int_\Gamma \vec{n}\cdot(\sigma_{12},\sigma_{22})\phi_2 ds\end{aligned}$$

by the divergence theorem. Thus the weak form of the linear elastic equation is given by

$$\int_\Omega \sigma : \epsilon(\phi)dx = \int_\Gamma \vec{\phi}\cdot\sigma n\, ds + \int_\Omega \vec{f}\cdot\vec{\phi}\, dx \tag{4.7}$$

where

$$\epsilon(\phi) = \frac{1}{2}\left(\frac{\partial\phi_i}{\partial x_j} + \frac{\partial\phi_j}{\partial x_i}\right)$$

we use this form in the following sections to develop the variational form of the Signorini contact problem as well as the Coulomb friction problem.

4.2.1 Signorini Contact Problem

The variational form of the Signorini Contact problem is

$$\min_u J(u) = \frac{1}{2}\int_\Omega \sigma : \epsilon - \int_{\Gamma_0} u_n g ds - \int_\Omega (f, u) dx$$
$$\text{subject to } u_n \le d \text{ on } \Gamma_c \tag{4.8}$$

where Γ_c is the contact region. Note that this formulation is equivalent to the full contact problem with $\mathcal{F} = 0$ at Γ_c and σ_n is the Lagrange multiplier for the inequality constraint $u_n - d \le 0$. That is, define the Lagrange functional

$$L(u, \mu) = J(u) + \int_{\Gamma_c} (\mu, u_n - d)\, ds$$

and the optimality condition is given by

$$L_u(u, \mu)(\phi) = \int \sigma : \epsilon(\phi)\, dx - \int_{\Gamma_0} (g, \phi_n) - \int_{\Gamma_c} (\mu, \phi_n)\, ds = 0$$

for all $\phi \in H^1(\Omega)^2$ and

$$\mu \ge 0 \text{ and } \int_{\Gamma_c} (\mu, u_n - d)\, ds = 0$$

assuming L^2 lagrange multiplier μ exists.

4.2.2 Coulomb Friction Problem

For the Coulomb friction problem using (4.7) we have the variational formulation

$$\min_u \frac{1}{2}\int_\Omega \sigma : \epsilon - \int_{\Gamma_0} (u_n, g)\, ds - \int_\Omega (f, u) ds + \int_{\Gamma_c} \mathcal{F}|u_\tau|\, dx \tag{4.9}$$

where $\sigma_N = 0$ at Γ_c. The necessary optimality condition is given by

$$\int_\Omega \sigma : \epsilon(\phi) - \int_{\Gamma_0} (\phi_n, g)\, ds - \int_\Omega (f, u)\, dx - \int_{\Gamma_c} \mathcal{F}(\lambda, \phi_\tau)\, ds = 0,$$

where

$$\lambda \cdot u_\tau = |u_\tau|, \quad \text{and} \quad |\lambda| \leq 1.$$

Thus, the complementary condition is

$$|\sigma_\tau| \leq \mathcal{F}|\sigma_\tau| \text{ if } u_\tau = 0, \quad \text{and} \quad \sigma_\tau = \mathcal{F}\sigma_N \text{ if } u_\tau \neq 0.$$

Let V be a Hilbert space and a be a symmetric, coercive form on $V \times V$, and $f \in V^*$ is a bounded linear functional on V. Then, one can write (4.8) and (4.9) as variational inequalities [10, 4] of the form

$$\min_{u \in V} \frac{1}{2} a(u, u) - f(u) + j(u) \tag{4.10}$$

where $V = H^1(\Omega) \times H^1(\Omega)$ and

$$a(u, \phi) = \int_\Omega \sigma : \epsilon(\phi)\, dx.$$

The nonsmooth convex functional j on V is defined by

$$j(u) = I_{u \leq d} \text{ for Signorini contact (4.8) and}$$

$$j(u) = \int_{\Gamma_c} \mathcal{F}|u_\tau|\, ds \text{ for Coulomb friction (4.9).}$$

We will use the variational inequality form (4.10) of the Signorini contact and the Coulomb friction problems to develop our solution techniques. We will be employing the primal-dual active set strategy. This is an efficient method for quadratic problems like (4.10) that is based on the complementary condition.

4.3 Nonsmooth Optimization Algorithms

Before we can develop solution techniques for the Coulomb friction problem (4.9) we need to address the L^1 term $\int_{\Gamma_c} \mathcal{F}|u_\tau|$. To do this we let ψ_ϵ be the regularization of $|\cdot|$;

$$\psi_\epsilon(s) = \begin{cases} |s| & \text{if } |s| \geq \epsilon \\ \frac{1}{2\epsilon}|s|^2 + \frac{\epsilon}{2} & \text{if } |s| \leq \epsilon. \end{cases}$$

$\psi_\epsilon \in C^1(R^d)$ where $0 < \epsilon << 1$ which we differentiate as

$$\psi'_\epsilon(s) = \begin{cases} \frac{s}{|s|} & \text{if } |s| > \epsilon \\ \frac{s}{\epsilon} & \text{if } |s| \leq \epsilon. \end{cases}$$

Using this regularization (4.10) is now the regularized problem

$$\min_{u \in V} \quad J_\epsilon(u) = \frac{1}{2}a(u,u) - f(u) + j_\epsilon(u) \tag{4.11}$$

where

$$j_\epsilon(u) = \int_{\Gamma_c} \mathcal{F}\psi_\epsilon(u_\tau)\, ds.$$

The necessary and sufficient optimality condition for the regularized problem is written as

$$a(u,\phi) - f(\phi) - (\lambda_\epsilon, \phi), \quad \lambda_\epsilon = \psi'_\epsilon(u_\tau)$$

for all $\phi \in V$. Note that

$$|\lambda_\epsilon| \leq 1, \quad |\lambda_\epsilon u_\tau| = |u_\tau| \max(0, |u_\tau| - \epsilon).$$

It can be proven that the solutions $(u^\epsilon, \lambda^\epsilon)$ converge weakly to to the solution (u, λ) to (4.10) in $V \times L^2(\Gamma_c)$ if the trace operator γ of V onto $L^2(\Gamma_c)$ is compact.

To solve (4.11) consider the following fixed point method

$$Au^k + \gamma^* \left(\frac{\mathcal{F}}{\max(\epsilon, |\gamma u^{k-1}|)}\, \gamma u^k \right) - f = 0 \tag{4.12}$$

where $A \in \mathcal{L}(V, V^*)$ is defined by

$$a(u, \phi) = \langle Au, \phi \rangle_{V^* \times V}$$

and $\gamma^* : L(\Gamma_c) \to V^*$ is the dual operator of γ.

Theorem 1 The fixed point method (4.12) is globally convergent.

Proof: Equation (4.12) is equivalently written as

$$a(u^{k+1}, \phi) - (f, \phi) + (\psi'_\epsilon(u^k_\tau)\, u^{k+1}_\tau, \phi)_{\Gamma_c} = 0 \tag{4.13}$$

for all $\phi \in V$. Note that if we let $u^{k+1} = u^+$ and $u^k = u$ and $\lambda = \psi'_\epsilon(u^k_\tau)$, then

$$\lambda \cdot (u^+ - u)_\tau = \psi'_\epsilon(|u_\tau|)\frac{1}{2}(|u^+_\tau|^2 - |u_\tau|^2) + |(u^+ - u)_\tau|^2).$$

Since $\Psi_\epsilon(\sqrt{|s|})$ is concave,

$$\lambda \cdot (u^+ - u)_\tau \leq \psi_\epsilon(|u^+_\tau|) - \psi_\epsilon(|u_\tau|).$$

Letting $\phi = u^+ - u$ in (4.13), we obtain

$$J_\epsilon(u^{k+1}) - J_\epsilon(u^k) \leq 0$$

where equality only holds if $u^{k+1} = u^k$. Thus, there exist a $u^* \in V$ and $\lambda^* \in L^\infty(\Gamma_c)$ such that $u^* \to u^*$ weakly in V and $\lambda^k \to \lambda^*$ weakly star in $L^\infty(\Gamma_c)$. Since γ is compact $u^k_\tau \to u^*_\tau$ strongly. Without loss of generality $u^k_\tau \to u^*$ strongly in $L^2(\Gamma_c)$ and $\lambda^k \to \lambda^*$ strongly in $L^\infty(\Gamma_c)$. Thus, $u^* \in V$ is the unique solution to (4.15) and $a(u^k, u^k) \to a(u^*, u^*)$ and (4.12) is globally convergent for all $\epsilon > 0$. $\square$

The fixed point method works sufficiently well for the friction problem since A is an elliptic operator.

4.3.1 Primal-Dual Active Set Method for Coulomb Friction Problem

In general, for (4.10) we can use the second order method based on the semi-smooth Newton method. A Newton derivative of ψ'_ϵ is given by

$$\psi''_\epsilon(s) = \begin{cases} 0 & |s| \geq \epsilon \\ \frac{1}{\epsilon} & |s| \leq \epsilon. \end{cases}$$

Thus the Jacobian ψ'_ϵ is singular at the solution. In order to overcome this difficulty we use the Lagrange multiplier approach. The method is based on the complementarity system;

$$Au + \gamma^*(\mathcal{F}\,\lambda) = f$$

$$\max(1, |\lambda + c\,v|)\lambda = \lambda + c\,v, \quad v = \gamma u.$$

where $\gamma \in \mathcal{L}(V, Y)$ represents $\gamma u = u_\tau \in L^2(\Gamma_c)$ (the surface trace for 3D case) for the friction problem. We apply the semismooth Newton method:

$$|\lambda + c\,v|\,\delta\lambda + \lambda\left(\frac{\lambda + c\,v}{|\lambda + c\,v|}\right)^t (\delta\lambda + c\delta v) + |\lambda + c\,v|\lambda - (\lambda^+ + c\,v^+) = 0,$$

if $d = |\lambda + c\,v| > 1$, pointwise, i.e on the inactive set where δv and $\delta\lambda$ are increments; $\delta v = v^+ - v$ and $\delta\lambda = \lambda^+ - \lambda$, or equivalently

$$\lambda^+|\lambda + c\,v| + \lambda\left(\frac{\lambda + c\,v}{|\lambda + c\,v|}\right)^t (\lambda^+ + c\,v^+) - \lambda|\lambda + c\,v| = \lambda^+ + c\,v^+. \tag{4.14}$$

Next, we apply the damping and regularization to (4.14);

$$\lambda^+|\lambda + c\,v| + \beta\,\frac{\lambda}{|\lambda| \wedge 1}\left(\frac{\lambda + c\,v}{|\lambda + c\,v|}\right)^t (\lambda^+ + c\,v^+) - \beta\,|\lambda + c\,v|\frac{\lambda}{|\lambda| \wedge 1} = \lambda^+ + c\,v^+. \tag{4.15}$$

Here, the purpose of the regularization $\frac{\lambda}{|\lambda|\wedge 1}$ is to automatically restrict the dual variable λ to the unit ball. The damping factor β is automatically selected to achieve stability.

To this end, we let

$$d = |\lambda + c\,v|, \quad \eta = d - 1, \quad a = \frac{\lambda}{|\lambda| \wedge 1}, \quad b = \frac{\lambda + c\,v}{|\lambda + c\,v|}, \quad F = ab^t$$

and we have

$$(\eta\, I + \beta\, F)\lambda^+ = (I - \beta\, F)(c\,v^+) + \beta d\, a.$$

Since $F^2 = (a \cdot b)\, F$, by Sherman-Morrison formula

$$\lambda^+ = \frac{1}{\eta}(I - \frac{\beta\, d}{\eta + \beta\, a \cdot b}\, F)c\,v^+ + \frac{\beta\, d}{\eta + \beta a \cdot b}\, a.$$

In order to achieve stability, we let $\frac{\beta\, d}{\eta + \beta\, a \cdot b} = 1$, i.e., $\beta = \frac{d-1}{d - a \cdot b} \leq 1$. Consequently, we obtain a Newton step

$$\lambda^+ = \frac{1}{d-1}(I - F)(c\,v^+) + \frac{\lambda}{|\lambda| \wedge 1} \tag{4.16}$$

and can use the primal dual active set strategy.

Primal-Dual Active Method (Coulomb Friction Problem)

1. Initialize: $\lambda^0 = 0$ and solve $Au^0 = f$ for u^0. Set $k = 0$.

2. Set inactive set $\mathcal{I}_k$ and active set $\mathcal{A}_k$ by

$$\mathcal{I}_k = \{|\lambda^k + c\,u_\tau^k| > 1\} \quad \text{and} \quad \mathcal{A}_k = \{|\lambda^k + c\,u_\tau^k| \leq 1\}.$$

3. Solve for $(u^{k+1}, \lambda^{k+1}) \in V \times L^2(\Gamma_c)$:

$$\begin{cases} Au^{k+1} + \gamma^*(\mathcal{F}\,\lambda^{k+1}) = f \\ \\ \lambda^{k+1} = \dfrac{1}{d^k - 1}(I - F^k)(c\,u_\tau^{k+1}) + \dfrac{\lambda^k}{|\lambda^k| \wedge 1}, \text{ in } \mathcal{A}_k \text{ and } u_\tau^{k+1} = 0 \text{ in } \mathcal{I}_k. \end{cases}$$

4. Convergent or set $k = k + 1$ and Return to Step 2.

The algorithm is unconditionally stable and rapidly convergent.

4.3.2 Primal-Dual Active Set Method for Signorini Contact Problem

Let γ be the trace operator of $H^1(\Omega)$ onto $L^2(\Gamma_c)$. The constrained minimization

$$\min_u \frac{1}{2}a(u,u) - (f,u) \quad \text{subject to } \gamma u \le d$$

has a unique solution $u \in V$ and there exists the Lagrange multiplier $\mu \in L^2(\Gamma_c)$ such that the necessary optimality holds;

$$\begin{cases} Au + \gamma^* \mu = f \\ \\ \mu = \max(0, \mu + c\,(\gamma u - d)) \end{cases}$$

for all $c > 0$, based on this complementarity system, we now apply the primal-dual active set method [4].

Primal-Dual Active Set Method (Signorini's Problem)

1. Initialize: $\mu^0 = 0$ and solve $Au^0 = f$ for u^0. Set $k = 0$.

2. Set inactive set $\mathcal{I}_k$ and active set $\mathcal{A}_k$ by

$$\mathcal{I}_k = \{\mu^k + c(\gamma u^k - d) \le 0\} \quad \text{and} \quad \mathcal{A}_k = \{\mu^k + c(\gamma u^k - d) > 0\}.$$

3. Solve for $(u_{k+1}, \mu_{k+1}) \in V \times L^2(\Gamma_c)$:

$$\begin{cases} Au^{k+1} + \gamma^* \mu^{k+1} = f \\ \\ \gamma u^{k+1} = d \text{ in } \mathcal{A}_k \text{ and } \mu^{k+1} = 0 \text{ in } \mathcal{I}_k \end{cases}$$

4. Convergent or set $k = k + 1$ and return to Step 2.

It involves solving the linear system for $(u, \mu_{\mathcal{A}})$ of the form

$$\begin{pmatrix} A & \gamma_{\mathcal{A}}^* \\ \gamma_{\mathcal{A}} & 0 \end{pmatrix} \begin{pmatrix} u \\ \mu_{\mathcal{A}} \end{pmatrix} = \begin{pmatrix} f \\ d_{\mathcal{A}} \end{pmatrix}$$

where $\cdot_{\mathcal{A}}$ is the restriction on the active set $\mathcal{A}$. For the convergence of the primal-dual method we refer to [8, 4].

4.3.3 Full Contact Problem

To solve the full contact problem combine the algorithm for the Signorini problem and Coulomb friction problem to develop Algorithm 1.

Algorithm 1 Iterative algorithm for the full contact problem

1. Initialize: we solve the Signorini problem with $\mathcal{F} = 0$ to obtain (u^0, μ^0) and set $k = 0$.
2. Update A for the friction problem by
$$A_k u = Au + \gamma^* \left(\frac{\mathcal{F}_k}{\epsilon \wedge |\gamma u_k|} (\gamma u) \right), \quad \mathcal{F}_k = \mathcal{F}\, \mu_k.$$
3. Solve the Signorini problem with $A = A_k$ to obtain (u_k, μ_k).
4. Convergent or set $k = k + 1$ an return to Step 2.

For this algorithm we are using the primal dual active set method for Signorini contact to initialize the problem in step 1 we also use this method in step 3 when we are updating the solution u_k and the Lagrange multiplier μ_k. In step 2 we are using the fixed point method developed for the Coulomb friction problem to update $\mathcal{A}_k$ but the primal dual active set method for the Coulomb friction problem may also be used.

Chapter 5

Finite Element Approximation using Adini Elements

5.1 Numerical Method

In Chapter 4 we developed variational formulations for the Signorini contact problem (4.10) and the Coulomb friction problem (4.11). In this chapter we develop a multi-moment finite element scheme based on Adini's elements which uses the function value and gradient values at nodes to approximate the variational formulations. Specifically the Adini's elements at square cell have 12 elements which determined by the solution value and the solution derivatives in the x and y directions at the four corners of the cell. Adini elements have previously been used for Poisson's equation and is known to achieve superconvergence to the true solution. Additionally for the Poisson equation with Neumann conditions using an Adini based finite element schmeme is more accurate and more economical in computations on a regular domains [2]. The Adini elements belongs to $H^1(\Omega)$ and are conformal. The conformal finite element method is a well established numerical method for solving the elastic problem.

We will describe implementation details and demonstrate the high order (fourth) accuracy of this discretization method for an eigenvalue problem on a regular domain, i.e., a domain which may be divided into small rectangles. We will also describe extension of this technique to an irregular domain, i.e. non-rectangular. and test this procedure for an eigenvalue problem for the Laplace equation on an elliptic domain. After we develop the Adini finite element scheme we then apply this discretization scheme to the full linear contact problem and use the iterative algorithm developed in section 4.3.3 to solve an example problem numerically. Specifically we will look at the test problem of solving the elastic contact problem on the square domain $\Omega = [0,1]^2$, as in Figure 5.5 as well as provide the discretized iterative scheme for solving the full elastic contact problem.

5.1.1 Adini's Element Discretization

We form the stiffness matrix S_0 using a multimoment Reiz-Galerkin approximation, i.e., use quadrature to evaluate (4.7) for the Adini's elements. Applying (4.4) and (4.5) to $\displaystyle\int_\Omega \sigma : \epsilon \, dx$ gives

$$\int_\Omega \left((2\mu+\lambda)\left(\left|\frac{\partial u_1}{\partial x}\right|^2 + \left|\frac{\partial u_2}{\partial y}\right|^2 \right) + \mu \left(\left|\frac{\partial u_1}{\partial y}\right|^2 + \left|\frac{\partial u_2}{\partial x}\right|^2 \right) + \mu \left(\frac{\partial u_1}{\partial y}\frac{\partial u_2}{\partial x} \right) + \left(\lambda \frac{\partial u_1}{\partial x}\frac{\partial u_2}{\partial y} \right) \right).$$

To approximate u on Ω we assume

$$u = u(x,y) = \sum_{x^k y^\ell \in P} a_{k\ell} x^k y^\ell \tag{5.1}$$

where

$$P = \{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^3y, xy^3\}.$$

At each square cell we have (u, u_x, u_y) at the four corners and $a_{k\ell}$ is completely determined by the values of (u, u_x, u_y) at the four corners. That is, we have 12 elements $\phi_k,\ 1 \le k \le 12$ and the Adini polynomial

$$\begin{aligned} u(x,y) = (u^{(1)})_k\phi_1(x,y) + (u_x^{(1)})_k\phi_2(x,y) + (u_y^{(1)})_k\phi_3(x,y) + (u^{(2)})_k\phi_4(x,y) \\ +(u_x^{(2)})_k\phi_5(x,y) + (u_y^{(2)})_k\phi_6(x,y) + (u^{(3)})_k\phi_7(x,y) + (u_x^{(3)})_k\phi_8(x,y) \\ +(u_y^{(3)})_k\phi_9(x,y) + (u^{(4)})_k\phi_{10}(x,y) + (u_x^{(4)})_k\phi_{11}(x,y) + (u_y^{(4)})_k\phi_{12}(x,y) \end{aligned}$$

where $u^{(1)}$ is located at the lower left corner of the square cell, $u^{(2)}$ is in the upper left corner, $u^{(3)}$ is in the lower right corner, and $u^{(4)}$ is in the upper right corner. In Figure 5.1 we have the 12 shape elements ϕ_k; in the first column we have the shape functions corresponding to the $u^{(k)}$'s, in the second column the shape functions for the $u_x^{(k)}$'s, and in the third column the shape functions for the $u_y^{(k)}$'s. We observe that the shape functions are C^1.

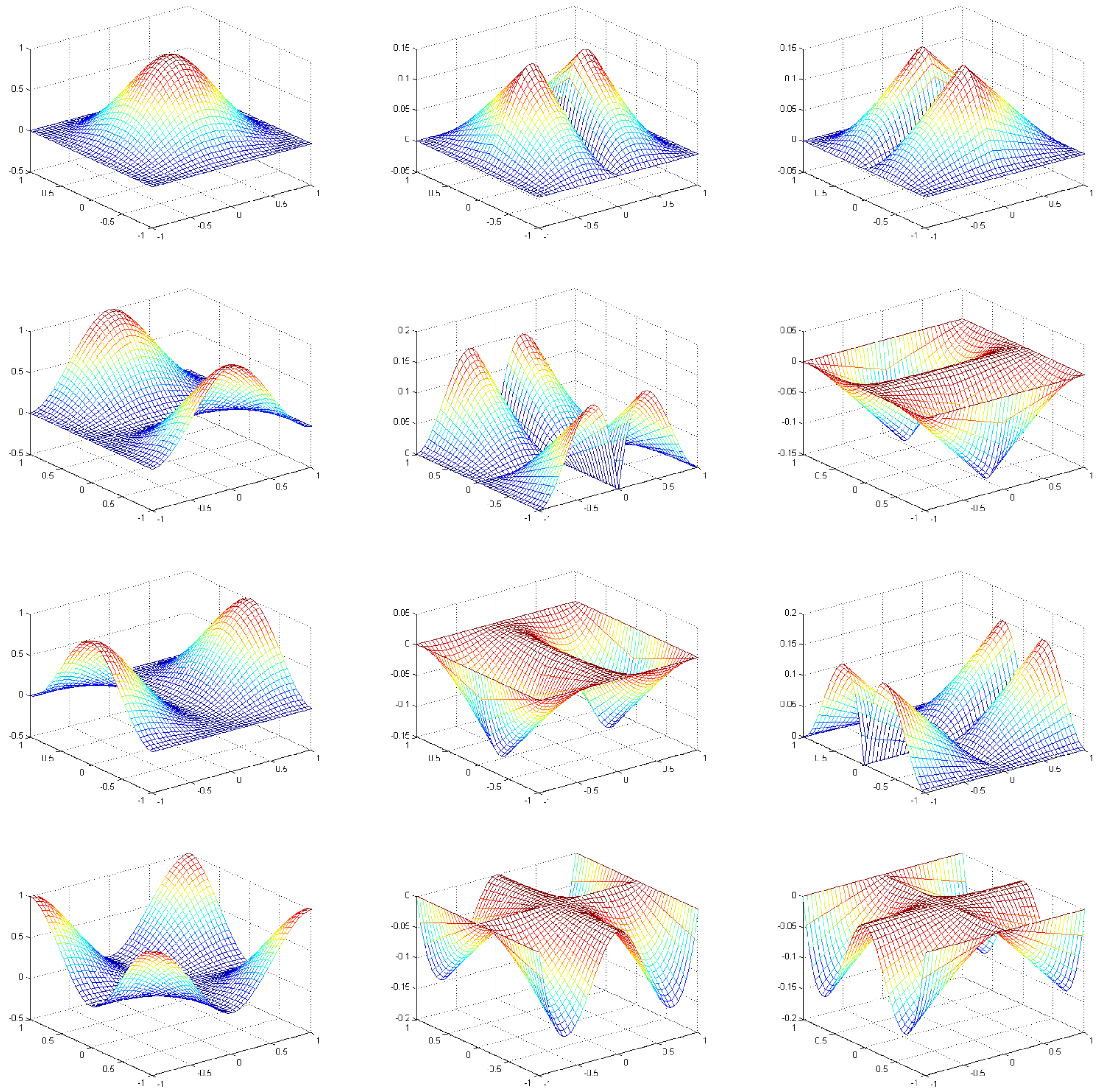


Figure 5.1: Shape functions $\phi_k(x, y)$ for $k = 1, .., 12$.

We compute 12×12 stiffness matrices for

$$\int_C (2\mu+\lambda)\frac{\partial \phi_k}{\partial x}\frac{\partial \phi_\ell}{\partial x}\, dxdy, \quad \int_C (2\mu+\lambda)\frac{\partial \phi_k}{\partial y}\frac{\partial \phi_\ell}{\partial y}\, dxdy,$$

$$\int_C \mu\frac{\partial \phi_k}{\partial x}\frac{\partial \phi_\ell}{\partial x}\, dxdy, \quad \int_C \mu\frac{\partial \phi_k}{\partial y}\frac{\partial \phi_\ell}{\partial y}\, dxdy,$$

$$\int_C \mu\frac{\partial \phi_k}{\partial x}\frac{\partial \phi_\ell}{\partial y}\, dxdy \quad \int_C \lambda\frac{\partial \phi_k}{\partial y}\frac{\partial \phi_\ell}{\partial x}\, dx,$$

at each cell C. If $\mu,\ \lambda$ are constant we only need to compute three stiffness matrices. We assemble the discretized elastic matrix S_0 by cell-by-cell procedure, i.e, unknowns (u, u_x, u_y) at the node $\{i, j\}$ has the contributions from unknowns (u, u_x, u_y) at the nine neighboring nodes $\{i+[-1,0,1], j+[-1,0,1]\}$. Thus, the discretized matrix S_0 is of the form

$$S_0 = \begin{pmatrix} H_{11} & H_{12} \\ H_{12}^T & H_{22} \end{pmatrix}$$

and $H_{11},\ H_{12},\ H_{22}$ are the block tri-diagonal matrices and each block is tri-diagonal matrix of block size of 3 as shown in Figure 5.2 using the standard ordering of unknowns.

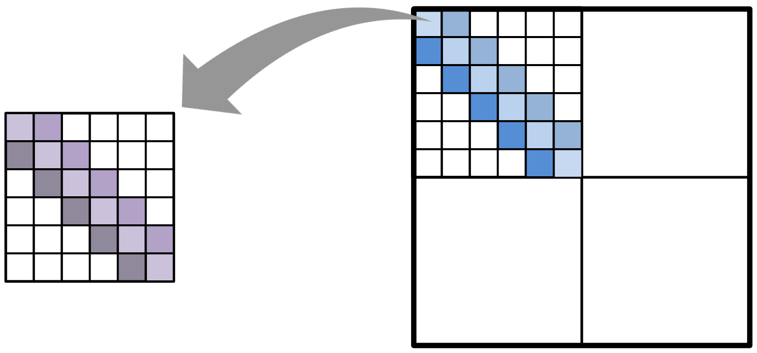


Figure 5.2: Block tri-Diagonal matrix of block tri-diagonal matrices

Next we approximate the trace operator γ by

$$Gu = u(x_k, 0)$$

where x_k is nodes $i\Delta x,\ (i+\frac{1}{3})\Delta x,\ (i+\frac{2}{3})\Delta x$ for $0 \le i \le N-1$ with $\Delta x = \frac{1}{N}$. We are taking intermediate points along the boundary to prevent oscillations caused by the cubic polynomials used in our interpolations and to ensure that the boundary conditions are held. Also, we use the approximation

$$\int_0^1 |u_\tau(x,0)|\,dx \sim \sum_{k=0}^{3N} |(Gu)_k|$$

for our friction term. Hence we have the discretized Signorini problem

$$S_0 u + G^T\mu = 0, \quad \mu = \max(0, \mu + c\,(Gu - d)), \tag{5.2}$$

and the discretized friction problem

$$S_0 + G^T\left(\frac{\mathcal{F}}{\epsilon \wedge |Gu|}\,Gu\right) = f. \tag{5.3}$$

Let $S = S_0$ and the corresponding algorithm for Signorini's problem is written in Algorithm 2.

5.1.2 Accuracy Test for Eigenvalue Computations

To validate our method we consider the eigenvalue problem for the Laplacian

$$-\Delta u = \lambda u \text{ in } \Omega = [0,1]^2 \tag{5.4}$$

with the Neumann boundary condition: $\frac{\partial}{\partial\nu}u = 0$ at Γ, which has the eigen pairs

$$\lambda_{k,\ell} = (k\pi)^2 + (\ell\pi)^2, \quad u_{k,\ell} = \cos(k\pi x)\cos(\ell\pi y), \ \ k,\ell \ge 0.$$

The weak form of (5.4) is

$$\int_\Omega (\nabla u \cdot \nabla\phi) = \lambda \int_\Omega u\phi dx. \tag{5.5}$$

Algorithm 2 Primal-Dual Active Set Method (Discretized Signorini's Problem)

```
1. Initialize: $\mu^0 = 0$ and solve $Su^0 = f$ for $u^0$. Set $k = 0$.
2. Set inactive set $\mathcal{I}_k$ and active set $\mathcal{A}_k$ by
       $\mathcal{I}_k = \{\mu^k + c(Gu^k - d) \le 0\}$ and $\mathcal{A}_k = \{\mu^k + c(Gu^k - d) > 0\}$.
3. Solve for $(u_{k+1}, \mu_{k+1}) \in V \times L^2(\Gamma_c)$:
       $\begin{cases} Su^{k+1} + G^T \mu^{k+1} = f \\ Gu^{k+1} = d \text{ in } \mathcal{A}_k \text{ and } \mu^{k+1} = 0 \text{ in } \mathcal{I}_k. \end{cases}$
4. Convergent or set $k = k + 1$ and Return to Step 2.
```

Let Q be the mass matrix defined by $Q_{k,\ell} = \int_\Omega \phi_k(x)\phi_\ell(x)\,dx$ and the stiffness matrix defined by $H_{k,\ell} = \int_\Omega \nabla\phi_k(x)\cdot\nabla\phi_\ell(x)$. Then, the approximation of (5.5) by Adini's element is given by

$$Qu = \lambda Hu.$$

So, the eigenvalues of the matrix $Q^{-1}H$ gives approximated eigenvalues of those for (5.4). In Table 5.1 we show the relative error of calculated eigenvalues where n^2 is the number of sub-cells of the square domain Ω. As shown Figure 5.3(a) it achieves better than fifth order convergence.

Table 5.1: Relative error in approximated eigenvalues for Neumann conditions

k, ℓ	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$
$k = 1, \ell = 0$	3.8083e-6	8.7959e-8	1.6405e-9	3.7556e-11	1.2961e-11
$k = 1, \ell = 1$	8.9036e-5	2.3275e-6	4.5774e-8	7.8850e-10	3.0473e-11
$k = 2, \ell = 0$	2.0115e-4	5.0876e-6	1.0148e-7	1.7570e-9	3.8138e-11
$k = 2, \ell = 1$	7.7840e-4	2.0755e-5	4.3074e-7	7.5863e-9	1.3291e-10
$k = 2, \ell = 2$	5.4814e-3	1.3377e-4	2.8081e-6	4.9887e-8	8.1512e-10
$k = 3, \ell = 0$	1.6684e-3	4.9626e-5	1.0946e-6	1.9680e-8	3.1876e-10

For the Dirchlet boundary condition: $u = 0$ at Γ we set

$$u = 0, \quad u_\tau = \text{the tangential derivative} = 0$$

at the boundary nodes for our method, i.e. they are eliminated from the unknowns. In this case the eigen-pairs are given by

$$\lambda_{k,\ell} = (k\pi)^2 + (\ell\pi)^2, \quad u_{k,\ell} = \sin(k\pi x)\sin(\ell\pi y), \quad k, \ell \geq 1.$$

In Table 5.2 we show the relative error of the calculated eigenvalues and in Figure 5.3(b) it is shown we achieve better than fourth order convergence.

Table 5.2: Relative error in approximated eigenvalues for Dirichlet conditions

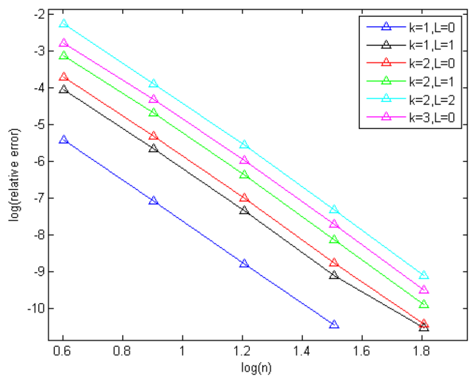
k, ℓ	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$
$k = 1, \ell = 1$	1.7520e-04	3.2782e-06	5.3965e-08	8.6487e-10	1.0645e-11
$k = 2, \ell = 1$	1.1614e-03	2.7884e-05	5.0444e-07	8.2056e-09	1.2010e-10
$k = 2, \ell = 2$	7.2218e-03	1.7520e-04	3.2782e-06	5.3967e-08	8.6395e-10
$k = 3, \ell = 1$	3.5719e-03	1.1969e-04	2.5367e-06	4.3395e-08	6.9228e-10
$k = 3, \ell = 2$	1.8746e-02	5.3439e-04	1.1077e-05	1.8910e-07	3.0259e-09

We also tested the method for calculating eigenvalues for the full elastic system with the stress free boundary condition on the unit square. For this particular problem we do not have a closed form of the solution so we only investigate the convergence rates. For Table 5.3 the relative error was computed using

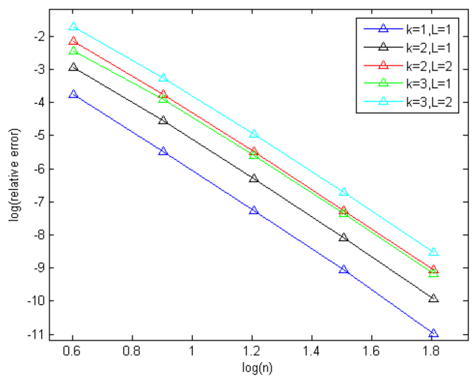
$$\text{Relative Error} = \frac{|\lambda_k^n - \lambda_k^{64}|}{|\lambda_k^n|}$$

where $n = 4, 8, 16$, and 32 which was then used to compute the convergence rates. In Figure 5.3(c) and Table 5.3 observe the convergence rate better than the fifth order.

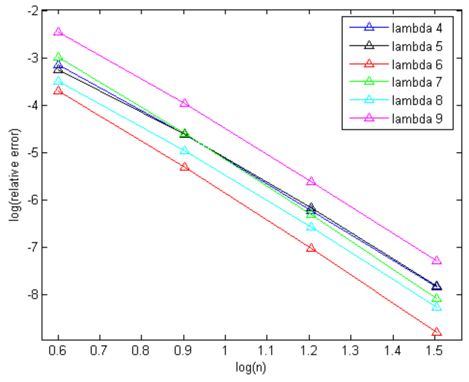
In summary, our tests for the accuracy of approximating the eigenvalues for the Laplace and elastic operator validate the high order accuracy of the proposed numerical approximations based on Adini's elements for a regular domain.



(a)



(b)



(c)

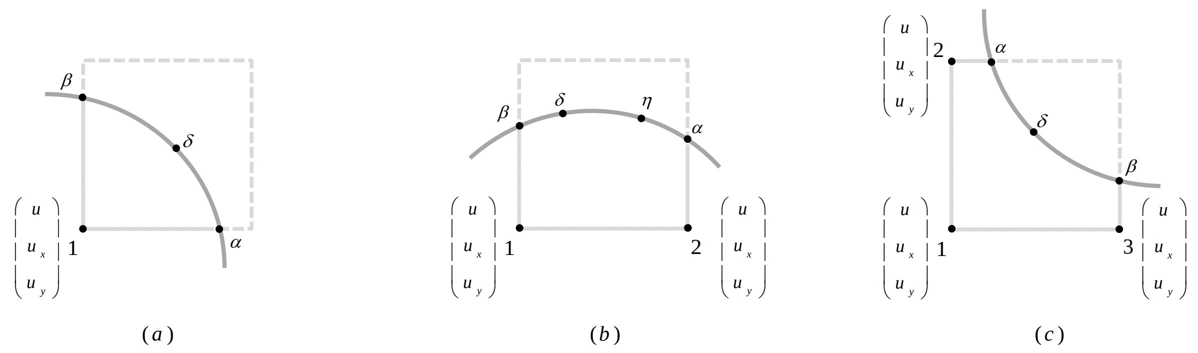
Figure 5.3: (a) Error plot for Neumann boundary conditions, (b) error plot for Dirichlet boundary conditions and (c) error plot for elastic system with stress free boundary conditions.

Table 5.3: Estimated Convergence rates for nonzero eigenvalues for the elastic system with Neumann conditions

λ_k	α_1	α_2	α_3
λ_4	4.9181	5.3406	5.3245
λ_5	4.5546	5.1730	5.4569
λ_6	5.3312	5.7216	5.9098
λ_7	5.3829	5.7365	5.8982
λ_8	4.8280	5.4208	5.6167
λ_9	5.0492	5.5053	5.5596

5.1.3 Adini Elements on Irregular Domains

For non-rectangular domains we consider domains which cannot be subdivided into rectangles with interior nodes at each corner. We will devise a strategy for rectangles with one or more exterior nodes. This strategy involves the use of boundary data to amend the conditions. For each cell there is a possibility of twelve different boundary cuts where one node is in the interior, two nodes are in the interior, or three nodes are in the interior for example see Figure 5.4.



β
δ
α
$\begin{pmatrix} u \\ u_x \\ u_y \end{pmatrix}$ 1
(a)
β δ η α
$\begin{pmatrix} u \\ u_x \\ u_y \end{pmatrix}$ 1
2 $\begin{pmatrix} u \\ u_x \\ u_y \end{pmatrix}$
(b)
$\begin{pmatrix} u \\ u_x \\ u_y \end{pmatrix}$ 2
α
δ
β
$\begin{pmatrix} u \\ u_x \\ u_y \end{pmatrix}$ 1
3 $\begin{pmatrix} u \\ u_x \\ u_y \end{pmatrix}$
(c)

Figure 5.4: Square cells with (a) one interior node, (b) two interior nodes, and (c) three interior nodes.

For the cases with a single interior node we will use the second order interpolation polynomial given by

$$u(x, y) = a_1 + a_2x + a_3y + a_4x^2 + a_5xy + a_6y^2$$

where we use intercepts α and β as well as an intermediate point δ on the boundary in addition to the data at the node to compute the coefficients a_j of $u(x, y)$. For the cases with two interior nodes we use the third order polynomial

$$u(x, y) = a_1 + a_2x + a_3y + a_4x^2 + a_5xy + a_6y^2 + a_7x^3 + a_8x^2y + a_9xy^2 + a_{10}y^3$$

with the intercepts α and β and two interior points δ and η along with the data at the two nodes to calculate the coefficients a_j. For the cases where we have three interior points we use the Adini polynomial

$$u(x, y) = a_1+a_2x+a_3y+a_4x^2+a_5xy+a_6y^2+a_7x^3+a_8x^2y+a_9xy^2+a_{10}y^3+a_{11}x^3y+a_{12}xy^3$$

where similarly to the case of one interior point we use the intercepts α and β as well as an intermediate point δ on the boundary to calculate the coefficients. With these new interpolation polynomials for the irregular square cell systems can be discretized in the same manor outlined in section 5.1.1.

Eigenvalue Problem on Irregular Domain

To investigate the accuracy and convergence of our scheme we consider the eigenvalue problem for the Laplacian on a circular domain Ω is centered at $(.5, .5)$ with radius $a = .411$

$$\begin{gathered} -\Delta u = \lambda u \text{ for } r^2 = x^2 + y^2 < a^2 \\ u = 0 \text{ for } r^2 = x^2 + y^2 = a^2 \end{gathered} \tag{5.6}$$

where with the Neumann boundary condition: $\frac{\partial}{\partial \nu} u = 0$ at the boundary Γ. The eigenvalues for (5.6) are given by

$$\lambda_m^i = \left[\frac{X_m^i}{a}\right]^2, \ i = 1, 2, ...$$

where X_m^i is the ith root of the Bessel function of index m. We have the relative error of the calculated eigenvalues in Table 5.4. The rate of convergence is significantly inferior to the regular domain case at less than second order convergence. This method is consistent and works reasonably but the element construction depends on the chosen orientation. This implies that the method does not work as desired and warrants further investigation.

Table 5.4: Relative error in approximated eigenvalues for irregular domain with Dirichlet conditions

λ_m^i	Exact	$n=4$	$n=8$	$n=16$	$n=32$	$n=64$
λ_0^1	34.2360	1.167e-1	3.690e-2	3.378e-3	4.835e-3	7.966e-4
λ_1^1	86.9162	1.251e-1	7.023e-3	9.053e-3	1.251e-2	4.417e-3
λ_1^1	86.9162	1.118e-1	6.005e-2	1.458e-2	2.618e-3	2.766e-3
λ_2^1	156.136	1.488e-1	1.399e-2	7.594e-3	1.434e-2	3.994e-3
λ_2^1	156.136	1.012e-1	4.527e-2	1.032e-2	3.675e-3	2.131e-3
λ_0^2	180.388	9.233e-2	3.443e-2	3.336e-3	4.394e-3	7.180e-4
λ_3^1	240.979	1.845e-1	1.431e-2	5.664e-3	7.926e-3	2.144e-3
λ_3^1	240.979	6.987e-2	5.016e-2	7.297e-3	3.284e-3	2.107e-4
λ_1^2	291.370	7.311e-2	8.801e-3	9.370e-3	1.199e-2	4.427e-3
λ_1^2	291.370	3.204e-2	4.849e-2	1.266e-3	2.946e-3	2.796e-3

5.2 Test Example

Recall the linear elastic contact problem from Chapter 4:

$$\begin{aligned} -div\sigma &= f \text{ in } \Omega, \\ \sigma n|_{\Gamma_0} &= g, \\ u|_{\Gamma_1} &= 0, \end{aligned} \tag{5.7}$$

$$u_n - d \leq 0, \quad \sigma_N \geq 0, \quad (u_n - d)\sigma_N = 0 \tag{5.8}$$

$$\begin{gathered} |\sigma_\tau u| \leq \mathcal{F}|\sigma_N|, \quad \text{if } u_\tau = 0 \\ \sigma_\tau = -\mathcal{F}\sigma_N \frac{u_\tau}{|u_\tau|}, \quad u_\tau \neq 0. \end{gathered} \tag{5.9}$$

We consider the case with the regular domain $\Omega = [0,1]^2$, the applied force $g(x) = 1$ on Γ_0, the obstacle function $d = .03$ at Γ_c, friction coefficient $\mathcal{F} = 0.9$ as in Figure 5.5. We use the following boundary conditions:

$$u = u_x = 0, \text{ on } \Gamma_0 \quad \text{and} \quad v = v_y = 0 \text{ on } \Gamma_1.$$

Note that in order to enforce Adini elements we added the Neumann condition which is not required for Galerkin or finite element methods. Our combined algorithm for the full contact problem as in section 4.3.3 is written in Algorithm 3.

We use 10×10 subdivisions (i.e.$\Delta x = 0.1$). For our tests our proposed algorithm

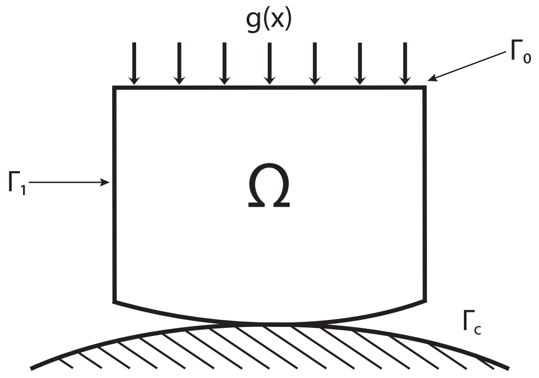


Figure 5.5: Deformed elastic body in contact with a rigid foundation.

Algorithm 3 Iterative algorithm for the discretized full contact problem

1. Initialize: we solve the Signorini problem with $\mathcal{F} = 0$ using Primal-Dual Active method to obtain (u_0, μ_0) and set $k = 0$.

2. Let
$$S_k = S_0 + G^T \left(\frac{\mathcal{F}_k}{\epsilon \wedge |Gu_k|} \, G \right), \quad \mathcal{F}_k = \mathcal{F} \, \mu_k.$$

3. Solve the Signorini problem with $S = S_k$ using Primal-Dual Active method to obtain (u_k, μ_k).

4. Convergent or set $k = k + 1$ an return to Step 2.

converged less than 10 iterates to obtain three digits accuracy. Figure 5.6(a) is the displacement v in the vertical direction where the effect of the obstacle can be seen along the bottom boundary Γ_c. In Figure 5.6(b) we have increased the resolution of our results using the multi-moment interpolation (5.1). In 5.6(c) is the displacement u in the horizontal direction with improved resolution. We recall that for the stress free condition at Γ_1 and the friction on the contact. Lastly 5.6(d) is the computed Lagrange multiplier μ where it can be seen that $\mu_k > 0$ along the contact region Γ_c and $\mu_k = 0$ otherwise.

We developed a numerical method for nonsmooth optimization problems, especially for solving the elastic contact problem. In Chapter 4 we detailed our approach based on the variational formulation of the Signorini Contact and Coulomb friction problems.

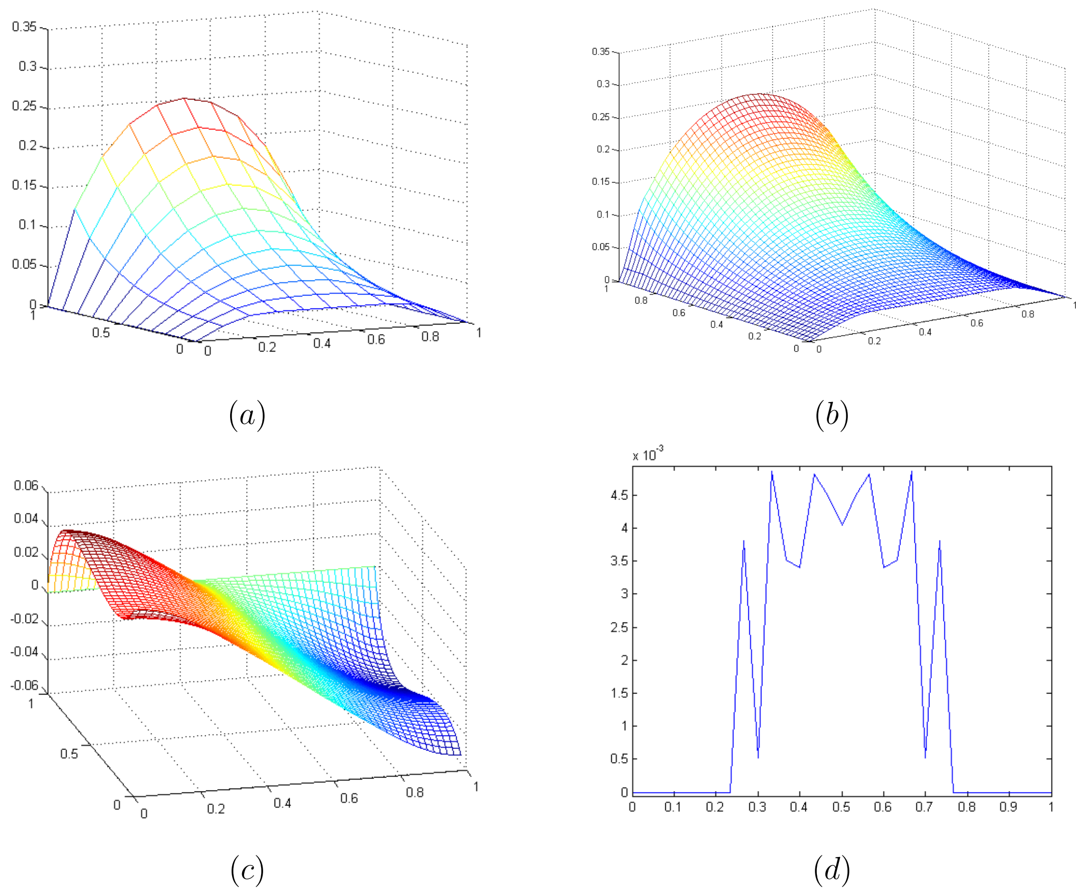


Figure 5.6: (a) Vertical displacement, (b) enhanced vertical displacement, (c) enhanced horizontal displacement, and (d) μ.

For both problems the optimality condition is given by the complementarity systems for the primal and dual variables. An effective optimization method using the semi-smooth Newton method based on the complementarity systems is then developed. The semi-smooth method is a very effective method for nonsmooth optimization problems and converges super linearly (for finite dimensional problems the active set methods convergence in a finite number of steps). Combining these two algorithms we developed an algorithm to solve the full elastic contact problem.

In Chapter 5 we use a finite element scheme based on the Adini's elements for our numerical approximations and demonstrated numerically the high order accuracy (fourth order) of the proposed numerical method for eigenvalue computations for regular domains. We also extended our numerical method for irregular (non-rectangular) domains and tested the method on a circular domain. The combined algorithm for solving the full elastic contact problem was tested numerically using our numerical approximations.

Chapter 6

CUDA Acceleration

Graphics card programming has become increasingly essential to high performance computing. Originally graphics processing unit (GPU) development was spurred on by the demand for 3D environments for PC gaming and as such evolved into hardware designed for intense, highly parallel computations required for graphics rendering. However since NVIDIA launched compute unified device architecture (CUDA) with CUDA C, the accompanying computer language, GPU processing has developed to address general-purpose tasks as well as graphics rendering tasks.

GPUs differ from traditional central processing units (CPUs) in that more transistors are devoted to data processing as opposed to memory and flow control. For program execution this means while a quad-core processors can only run four execution threads concurrently the smallest executable unit for NVIDIA GPUs is thirty two execution threads so a GPU with four multi-processors can have at least one hundred twenty eight threads [9]. Additionally GPUs achieve better performance per dollar and per watt that equivalent CPU based technology [12]. Currently heterogeneous platforms with both CPUs and GPUs are used in three of the top ten fastest super computers in the world which includes the formerly second fastest computer in the world the Tianhe-1A in China [1].

We will address several needs in large-scale optimization and large-scale control theory using CUDA based GPUs for distributed computing. With this software package large-scale problems that may have been considered impractical or costly may be tackled. Our goal in the development of this software was to design general purpose software for a variety of applications. A general strategy for creating optimized software is:

1. using the optimized libraries for linear algebra operations, particularly BLAS 3 operations where CUDA has the highest performance and

2. memory management to reduce memory transfers.

This strategy allows for rapid development of crucial software and can be viewed as a starting point for creating optimized software. By using available libraries we are taking advantage of available software and as these libraries further develop we will benefit from their advances. The second part of our strategy is memory management. This is a very important aspect when using GPUs as memory transfers are expensive especially memory transfers from the GPU to the CPU and vice versa. We have included a general strategy for memory management as part of our CUDA review in Appendix A. Using this strategy we have developed software for the quadratic minimization problem as well as partial eigenvalue and eigenvector computations.

6.1 Optimization

To begin to develop our general purpose software we first consider constrained minimization problems which arise in a variety of applications including optimal control and design, signals image analysis, parameter estimation, mechanical contact problems, image reconstruction, and mathematical finance. Specifically we consider problems with the variational form

$$\min J(x) \text{ over } x \in \mathcal{C} \tag{6.1}$$

where $\mathcal{C}$ is a closed convex set. A typical example of a constrained optimization problem is the quadratic programming which is to find the solution of

$$\min J(x) = \frac{1}{2}x^T Ax + b^T x \text{ subject to } Ex = c \text{ and } Gu \leq g \tag{6.2}$$

where $A \in \mathbb{R}^{n\times n}$ is a non-negative definite system matrix, $E \in \mathbb{R}^{k\times n}$ describes the equality conditions, $G \in \mathbb{R}^{\ell\times n}$ describes the inequality conditions, $x \in \mathbb{R}^n$ is the solution, $b \in \mathbb{R}^n$ is given, $c \in \mathbb{R}^k$ is given, and $g \in \mathbb{R}^\ell$ is given. Quadratic programing is used for general constrained minimization (6.1) in terms of sequential quadratic programing (SQP), i.e.,

$J(x)$ is sequntially linearized by

$$J(x) \approx J'(x_k) + J'(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T H_k (x - x_k)$$

where H_k is the Hessian. Quadratic programing for large-scale (n large) CUDA implementation should prove to be very effective as it is inherently parallelizeable. To solve the quadratic programming problem we employ the Lagrange multiplier method.

6.1.1 Lagrange Multiplier Method

In this section we will describe the Lagrange multiplier method and the algorithm based on the primal-dual active set method to solve (6.2). We begin by reformulating (6.2) as the unconstrained problem with penalty method with $\beta >> 1$

$$J_\beta(x) = \frac{1}{2}x^T A x - b^T x + \frac{\beta}{2}|Ex - c|^2 + \frac{\beta}{2}|\max(0, Gx - g)|^2$$

then we have the necessary optimality

$$F'_\beta = Ax - b + \beta E^T(Ex - c) + \beta\chi G^T \max(0, Gx - g) = 0 \tag{6.3}$$

where

$$\chi = \begin{cases} 1 & Gx - b > 0 \\ 0 & Gx - b \leq 0. \end{cases}$$

Setting the multipliers $\lambda_\beta = \beta(Ex_\beta - c)$ and $\mu_\beta = \beta\chi(Gx_\beta - g)$, (6.3) is equivalently written as

$$F'_\beta = Ax - b + E^T\lambda_\beta + G^T\mu_\beta = 0.$$

Note that one can prove that the solutions x_β converge to the unconstrained problem solution x^* of (6.2). Additionally $\lambda_\beta \to \lambda$ and $\mu_\beta \to \mu$ as $\beta \to \infty$ [4] and we have the system

$$\begin{cases} Ax + E^T\lambda + G^T\mu = 0 \\ Ex = c \\ \mu = \max(0, \mu + \beta(Gx - g)). \end{cases}$$

If $\begin{pmatrix} E \\ G_a \end{pmatrix}$ is surjective where G_a be the row vectors of G corresponding to the active indices, i.e. the indicices of x where $(Gx^* - g)_i = 0$, then (μ, λ) converges and under the Kuhn-Tucker (KKT) condition $\lambda \in \mathbb{R}^k$ and $\mu \in \mathbb{R}^\ell$ exist. The primal-dual active set method (see Algorithm 4) is based on the complementary condition

$$\begin{cases} Gx_i < g_i & \Rightarrow \quad \mu_i = 0 \text{ (inactice)} \\ Gx_i = g_i & \Rightarrow \quad \mu_i \geq 0 \text{ (active)} \\ (Gx - g) \cdot g & = \quad 0. \end{cases} \tag{6.4}$$

Updates to the current dual variables (λ, μ) are determined by the active and inactive sets

$$\mathcal{A} = \{k \in (\mu + (c(Gx - g))_k > 0\}, \;\; \mathcal{I} = \{j \in (\mu + c(Gx - g))_j \leq 0\}$$

where $c \in \mathbb{R}$ resulting in a Newton-like method in Algorithm 4. Note that if we have bilateral constraints $\phi \leq x \leq \psi$ we can embed the bilateral constraint into the inequality constraint

$$\begin{pmatrix} G \\ I \\ -I \end{pmatrix} x \leq \begin{pmatrix} g \\ \psi \\ \phi \end{pmatrix}$$

and continue as before.

Algorithm 4 Primal dual active set method

1. *Initialize:* x^0 and μ_0. Set $k = 0$.
2. *Set:* $\mathcal{A}_k = \{\mu_k + c(Gx_k - g) > 0\}, \;\; \mathcal{I} + k = \{\mu_k + c(Gx_k - g) \leq 0\}$.
3. *Solve* for $(x_{k+1}, \lambda_{k+1}, \mu_{k+1})$:

$$\begin{array}{l} Ax_{k+1} + E^T\lambda_{k+1} + G^T\mu_{k+1} \\ Ex_{k+1} = c, \\ Gx_{k+1} = g \;\; \in \mathcal{A}_k \text{ and } \mu_{k+1} = 0 \; in\mathcal{I}_k. \end{array} \tag{6.5}$$

4. *Stop;* or set $k = k + 1$ and return to (3).

To implement Algorithm 4 we first note that (6.5) is equivalent to solving the system

$$\begin{pmatrix} A & E^T & G_a^T \\ E & 0 & 0 \\ G_a & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \\ \mu_a \end{pmatrix} = \begin{pmatrix} b \\ c \\ g_a \end{pmatrix}. \tag{6.6}$$

Solving (6.6) for large-scale systems is expensive which is why we have elected to use distributed computing. In terms of design we have chosen to leverage the available optimized libraries CUBLAS, Thrust, and CULA. We use the NVidia CUBLAS library of functions for BLAS linear algebra operations, the Thrust Library for function primitives, and the EM Photonics CULA library for LAPACK linear algebra functions. We have tested our software for solving the quadratic programming problem via the primal-dual active set method using a quad core of Intel Core i7 CPUs for our baseline CPU matlab computations and a single Nvidia Telsa C2050 for our GPU computations with matlab as an interface. To benchmark our GPU implementation we used identical algorithms and randomly generated matrices where A is $n \times n$ where $n = 400, 800, 1600, 3200$ and 6400, E is $m \times n$ where $m = 40, 80, 80, 160$, and 320, G is $\ell \times n$ where $\ell = 20, 40, 40, 80$ and 3200 respectively.

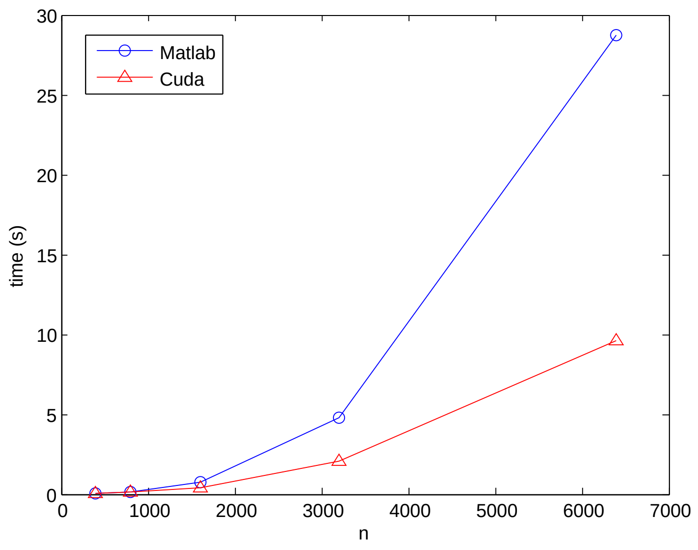


Figure 6.1: Matlab and CUDA computational times (s) with random matrices

As seen in Figure 6.1 we have been able to achieve significantly less computational time using our GPU optimization code. For system with $n = 6400$, which is a typical size for large scale quadratic programming, the CUDA implementation was approximately three times faster than the conventional architecture.

6.2 Controls

Large-scale dynamical systems problems arise via discretization of partial differential equations, for example fluid or vibration control problems. We will be considering the linear quadratic regulator problem

$$\min \int_0^\infty \left((x(t), Qx(t))_X + |u(t)|^2\right) dt \text{ over } u \in L^2(0, \infty, U)$$

subject to the linear control dynamics

$$\frac{d}{dt}x(t) = Ax(t) + Bu(t),\ x(0) = x_0 \in X$$

where $x(t) \in X$ is the state function and $u(t) \in U$ is the control function. Then the optimal control is given in the feedback form

$$u(t) = -B^*\Pi x(t)$$

where Π is bounded, self-adjoint operator on X and is the solution to the algebraic Riccati equation (ARE)

$$A^*\Pi x + \Pi Ax - \Pi BB^*\Pi x + Qx = 0 \tag{6.7}$$

for all $x \in dom(A)$. Solving (6.7) directly is expensive and somewhat impractical for large-scale problems. So to construct the optimal feedback gain $K = B^*\Pi$ via ARE we use a algorithm based on the inverse of the Hamiltonian operator on a reduced order subspace developed in [7]. If we let $p(t) = \Pi x(t)$ then

$$\frac{d}{dt}(x(t), p(t))^T = \mathcal{H}(x(t), p(t))^T$$

where the Hamiltonian operator $\mathcal{H}$ is given by

$$\mathcal{H} = \begin{pmatrix} A & -BB^* \\ -Q & -A^* \end{pmatrix}.$$

Let λ be an eigenvalue of $A - BB^*\Pi$, then λ is an eigenvalue of $\mathcal{H}$. Additionally if $x \in dom(A)$ is an eigenvector of $(A - BB^*\Pi)$ associated with λ then

$$\begin{aligned} (A - BB^T\Pi)x &= \lambda x \\ -Qx - A^T\Pi x &= \Pi(A - BB^*\Pi)x = \lambda\Pi x \end{aligned}$$

and $(x, \Pi x)$ is an eigenfunction of $\mathcal{H}$ and $\Pi = YV^{-1}$ where (V, Y) are the eigenvectors of $\lambda(\mathcal{H})$ with $Re\lambda < 0$. This process is described in Algorithm 5. The implementation

Algorithm 5 Reduced Hamiltonian ARE

1. Find the eigenpairs $(\lambda_i, (x_i, p_i))$ of $\mathcal{H}$ with $Re(\lambda_i) < 0$ where
$$\mathcal{H} = \begin{pmatrix} A & -BB^* \\ -Q & -A^* \end{pmatrix}$$
2. Form the matrices V and Y consisting of the eigenvectors of x_i and p_i.
3. Define $K^L\phi = B^*Y(V^TV)^{-1}(V^*, \phi)_X$ where $\phi \in X$.

of this algorithm relies on the implicit restarted Arnoldi method (IRAM) in Step 1 to compute the sub-invariant subspace of a given general matrix M for large magnitude eigenvalues. The IRAM is an iterative method which creates a reduced subspace by the Arnoldi process. Arnoldi iteration process is based on the idea of Krylov subspaces $\mathcal{K}_n$ defined by

$$\mathcal{K}_n = \{f, Mf, M^2f, \cdots, M^{n-1}f\}$$

where $f \in \mathbb{R}^n$ is a random seed vector. Using a Gram-Schmidt process on $\mathcal{K}_n$ an orthonormal basis $\{v_1, v_2, ..., v_n\}$ that spans the Krylov subspace can be formed and $V = [v_1 \ v_2 \ \cdots \ v_n]$ is an orthogonal matrix such that $MV_m = V_mH_m + f_me_m^T$ where H

is upper Hessenberg. The Arnoldi process creates a reduced order subspace and involves a number of BLAS 2, e.g. Householder decompositions, operations where we can expect CUDA use to be advantageous. Algorithm 6 describes the procedure used in IRAM. For this algorithm V and H are initialized to zero and e is a unit vector. In step 2 the shifts $\mu_1, \mu_2, \ldots, \mu_p$ are the unwanted eigenvalues. At the completion of this algorithm

$$MV_m y_{\lambda_i} = V_m \lambda_i y_{\lambda_i}$$

where λ_i are the eigenvalues of H and y_{λ_i} are the corresponding eigenvectors of H. In Table 6.1 we have a comparison of computational times for computing the ten eigenvalues of largest magnitude with a subspace size of twenty five for Matlab CPU computation versus GPU with a C++ interface computational times. For systems with over a million elements we see the benefit of the CUDA architecture. For the largest matrices tested our CUDA software computational time was approximately 4.5 times faster. While we are discussing IRAM in the context of the LQR problem the computation of partial sets of eigenvalues and eigenvectors is important to a large variety of problems.

Table 6.1: Performance in seconds with dense matrix with a subspace size of 25 and calculating 10 eigenvalues. We use either a quadcore (four Intel i7s) or a single GPU (Tesla C2050)

Matrix Order	0.4k	0.8k	1.6k	3.2k	6.4k
Intel	0.02596	0.04439	0.19215	0.64021	2.29447
Tesla	0.15194	0.16896	0.16221	0.20552	0.50485
Speedup	0.17088	0.26275	1.18455	3.11507	4.54488

6.2.1 Heat Equation Example

As an example LQR problem we consider the boundary control of the heat equation

$$\min \quad \int_0^\infty \left(\int_\Omega |y(x,t)|^2 \, dx + \sum_{k=1}^{4} \int_\Omega |u_k(t)|^2 \right) dt \tag{6.8}$$

Algorithm 6 Implicit Restarted Arnoldi Method

```
1. Initialize: $(M, V, H, f)$ with $MV_m = V_m H_m + f_m e_m^T$, and m-step Arnoldi Factorization;

2. Compute $\sigma(H_m)$ and select $p$ shifts $\mu_1, \mu_2, ..., \mu_p$.

3. $q^T = e_m^T$

4. For $j = 1, 2, ..., p$
     Factor$[Q_j, R_j] = qr(H_m - \mu_j I)$;
     $H_m = Q_j^H H_m Q_j$; $V_m = V_m Q$;
     $q = q^H Q_j$;
   End

5. Beginning with the $k$-step Arnoldi factorization
       $MV_k = V_k H_k + f_k e_k^T$,
   apply $p$ additional steps of the arnoldi process to obtain a new $m$-step Arnoldi factorization
       $MV_m = V_m H_m + f_m e_m^T$

6. Convergent or return to Step 2.
```

over the domain $\Omega = (0,1)^n$ subject to the heat equation

$$y_t = \Delta y \tag{6.9}$$

with the Neumann boundary control

$$\frac{\partial y}{\partial \nu} = \sum_{k=1}^{m} b_k(x) u_k(t), \quad x \in \Gamma. \tag{6.10}$$

We approximate (6.9)-(6.10) by the spectral method, i.e. we seek a solution of the form

$$y(x_1, x_2, t) = \sum_{0 \le k, \ell \le N} y_{k,\ell}(t) cos(k\pi x_1) cos(\ell \pi x_2).$$

We use to the weak formulation, i.e. integrating (6.9) against the test function $\phi(x_1, x_2) = cos(k\pi x)cos(\ell pix)$, to obtain the weak form:

$$\int_\Omega \frac{\partial y}{\partial t}\phi(x)\,dx = \int_\Omega \Delta y\phi\,dx = \int_\Gamma \frac{\partial y}{\partial \nu}\phi - \int_\Gamma \nabla y \nabla \phi\,dx$$

by Green's formula. For the two dimensional case

$$\int_\Gamma \frac{\partial y}{\partial \nu}\phi\,dx = \int_{\Gamma_1} u_1(t)b_1\phi + \int_{\Gamma_2} u_2(t)b_2\phi + \int_{\Gamma_3} u_3(t)b_3\phi + \int_{\Gamma_4} u_4(t)b_4\phi$$

where Γ_k is the four side of domain Ω. Thus, we obtain

$$\frac{dy_{k,\ell}}{dt} = -(k^2\pi^2 + \ell^2\pi^2)\pi^2\, y_{k,\ell}(t) + \sum_{j=1}^{4} b_{k,\ell}^{(j)} u_j(t) \tag{6.11}$$

where

$$b_{k,\ell}^{(1)} = \int_{\Gamma_1} b_1(x)\cos k\pi x\,dx$$

and the other coefficients $b^{(j)}$ are determined similarly. For the two dimensional system (6.11) is typically not large enough to consider using GPUs so we will focus on the three dimensional case. For this case we use the boundary control

$$b(x,y) = \chi_{(.3,.77)\times(.3,.77)}(x,y)$$

on two opposite sides $z = 0$ and $z = 1$ of $\Omega = (0,1)^3$. In order to construct the invariant subspace of the Hamiltonian $\mathcal{H}$ we apply the implicit Anordi method for $\mathcal{H}^{-1}$. We calculate $(x,y) = \mathcal{H}^{-1}(f,g)$ efficiently in the following manner

1. $s = A^{-1}f$ and $C = A^{-1}B$,
2. $t = I + C^TQC$,
3. $u = t^{-1}(-C^T(g + Qp))$
4. $x = Cu + p$,
5. $y = -A^{-T}(g + Qx)$

In Table 6.2 we summarize our results for determining the invariant subspace of $\mathcal{H}$ produced by the marginally stable eigenvalues of $\mathcal{H}$, i.e. the eigenvalues with the largest magnitude of $\mathcal{H}^{-1}$. Our partial test illustrates the speed of the algorithm and the additional capacity for further speedup provided by CUDA.

Table 6.2: Performance in seconds with dense matrices with a subspace size of L. We use either a quadcore (four Intel i7s) or a single GPU (GeForce GT 640M LE)

L	N	itr	CPU	GPU	Speedup
50	11^3	3	1.316	1.148	1.146
50	16^3	3	13.804	5.479	2.519
50	21^3	3	74.870	15.899	4.709
100	11^3	1	1.297	1.249	1.038
100	16^3	1	12.924	5.229	2.472
100	21^3	1	69.210	14.706	4.706
200	11^3	1	3.327	4.226	0.787
200	16^3	1	26.838	12.095	2.219
200	21^3	1	138.537	30.806	4.497

Remark 1. *As a consequence of the low dimensionality of the hyperbolic equations in Chapters 2 & 3 and the optimization problems in Chapters 4 & 5 these problems are unsuitable for CUDA use at this time. The software we have developed for solving the quadratic programing problem may be applied to the elastic contact problem in Chapter 5 in the future for cases with thousands or millions of nodes.*

6.3 Future Work

In this thesis we have discussed solutions to partial differential equations, nonsmooth optimization, and large-scale control problems. For hyperbolic PDEs we created a high order method using the method of characteristics and exact integration. We also applied the same methodology to the finite element method for elliptic equations for which we needed to discretize our nonsmooth optimization problems. Lastly we discussed applications of GPU computations to large-scale optimization and controls problems. Continuing our work in the future we would like to address the following:

- Our CIP method developed in Chapter 3 was able to accurately capture the true solution for cases like the slotted disk, in the future we need to address the oscillations at the discontinuities. It may be beneficial to adopt a scheme similar to the essentially non-oscillatory (ENO) or weighted ENO (WENO) schemes that actively choose the best nodal values to for profiling.

- We have focused solely on dense matrices for our GPU software but many applications including the heat equation would benefit from using sparse structures. Included in this would be the development of a Riccati solver based on Algorithm 7 developed in [7]. This algorithm uses a reduced subspace on a prescribed basis. Let W be the reduced order orthonormal basis of $\hat{X} \times \hat{X}$ (our restriction of $\mathcal{H}^{-1}$). Then
$$\hat{\mathcal{H}}^{-1} = \begin{pmatrix} W^* & 0 \\ 0 & W^* \end{pmatrix} \mathcal{H}^{-1} \begin{pmatrix} W & 0 \\ 0 & W \end{pmatrix}$$
and we can use $\hat{\mathcal{H}}$ to solve the ARE by Algorithm 7.

Algorithm 7 Reduced Hamiltonian ARE

```
1. Find the eigenpairs $(\lambda_i, (\hat{x}_i, \hat{p}_i))$ of $\hat{\mathcal{H}}^{-1}$ with $Re(\lambda_i) < 0$.
2. Form the matrices $\hat{V}$ and $\hat{Y}$ consisting of the eigenvectors of $x_i$ and $p_i$.
3. Define $\hat{K}^L\phi = B^*W\hat{Y}(\hat{V}^*\hat{V})^{-1}\hat{V}W^*\phi$ where $\phi \in X$.
```

- Another area that would benefit from GPU software is nonlinear filtering. Nonlinear filtering is a subject of interest because of its applications in science and engineering such as navigational and guidance systems or radar tracking and consists of estimating the state of a nonlinear stochastic system from observation data. Consider the process
$$x(k) = f(x(k-1)) + w(k) \tag{6.12}$$
for $x(k) \in \mathbb{R}_n$ and the observation process
$$y(k) = h(x(k)) + r(k) \tag{6.13}$$

where $w(k)$ and $v(k)$ are white noise. The extended Kalman filter is the most widely used filter for nonlinear filtering problems but its performance suffers in the presence of significant nonlinearities. For this reason we plan use algorithms based on Gaussian filters developed in [6]. As a first step in development we have built a fourth order Runge-Kutta ODE solver.

In addition to developing solutions for these problems we will begin a mathematical problem that combines all of these elements; numerical weather prediction (NWP).

6.3.1 Numerical Weather Prediction

In numerical weather prediction (NWP) mathematical models use current weather conditions to predict future weather conditions. In order to create these forecasts large amounts of observations are assimilated by computational algorithms to estimate system parameters needed by forecasting models. We will be addressing how to find optimal sensor locations for improved data assimilation and state estimation. The data assimilation problem can be viewed as a two level minimization. First, given parameter p we minimize the following cost over $x_n,\ 0 \leq N$. Let $J(p)$ be the minimum value given p. Then, we minimize $J(p)$ over admissible p where

$$\begin{aligned} J &= J_0^b + J^q + J^r, && (6.14)\\ J_0^b &= \frac{1}{2}[x_0^b - x_0]^T[P_0^b]^{-1}[x_0^b - x_0], && (6.15)\\ J^q &= \frac{1}{2}\sum_{n=1}^{N}[x_n - \mathcal{M}(x_{n-1})]^T Q_n^{-1}[x_n - \mathcal{M}(x_{n-1})], && (6.16)\\ J^r &= \frac{1}{2}\sum_{n=1}^{N}[y_n - \mathcal{H}(x_n)]^T R_n^{-1}[y_n - \mathcal{H}(x_n)], && (6.17)\end{aligned}$$

for $t_0 \leq t_n \leq t_N$ and J^r is the cost function for observation error, J^q is the cost function for model error, and J_0^b is the cost function for the background forecast error at $t = t_0$. Superscripts r, q, and b refer to the observations, model, and background. $\mathcal{M}$ is the forecast model (typically PDE based for example shallow water equations), P is the background error covariance matrix, Q is the model error covariance matrix and R is the observation error covariance matrix. The operator $\mathcal{H}(p)$ is the map from the state space to the observation space and depends on the sensor locations p where y is the observations. When $\mathcal{M}$ and $\mathcal{H}$ are linear we have $\mathcal{M}(x) = Mx$ and $\mathcal{H}(x) = Hx$. Then optimal solution for this system state x is given by

$$x^a = x^b + P^b H^T [HP^b H^T + R]^{-1} [y - Hx^b] \tag{6.18}$$

where

$$P^b = M[M_0 P_0^b M_0^T + Q]M^T$$

and the prior state background estimate x^b is

$$x_n^b = \mathcal{M}(x_{n-1}^b) \text{ subject to } x_0^b \tag{6.19}$$

for $1 \leq n \leq N$.This is a challenging problem primarily because of its size (millions of state variables). Solving this problem also will have implications for optimal sensor network design and data thinning.

REFERENCES

[1] TOP500 List, June 2012. `http://www.top500.org/`.

[2] Z. Li H. Huang and N. Yan. New Error Estimates of Adini's Elements for Poisson's Equation. *Applied Numerical Mathematics*, 50:49–74, 2004.

[3] A. Nishiguchi H. Takewaki and T. Yabe. Cubic Interpolated Pseudo-particle Method (CIP) for Solving Hyperbolic-Type Equations. *Journal of Computational Physics*, 61:261–268, 1985.

[4] K. Ito and K. Kunisch. *Lagrange Multiplier Approach to Variational Problems and Applications.* SIAM Advances in Design and Control, 2008.

[5] K. Ito and T. Takeuchi. CIP Immersed Interface Methods for Hyperbolic Equations with Discontinuous Coefficients. 2011.

[6] K. Ito and J. Toivanen. Gaussian Filters for Nonlinear Filtering Problems. *IEEE Transactions on Automatic Control*, 45:910–927, 2000.

[7] K. Ito and J. Toivanen. A Fast Algorithm for Determining Optimal Feedback Gain. Technical Report CRSC-TR08-18, Center for Research in Scientific Computing, North Carolina State University, November 2008.

[8] K. Ito M. Hintermüller and K. Kunisch. The Primal-Dual Active Set Strategy as a Semi- Smooth Newton Method. *SIAM Journal on Optimization*, 13:665–688, 2002.

[9] NVidia. *CUDA C Best Practices Guide v4.2.* NVidia, Santa Clara, CA, 2012.

[10] J.L. Lions R. Glowinski and R. Tremolieres. *Numerical Analysis of Variational Inequalities.*

[11] G. Stadler S. Hüeber and B. Wohlmuth. A Primal-Dual Active Set Algorithm for Three-Dimensional Contact problems with Coulomb friction. *SIAM Journal Scientific Computation*, 30:572–596, 2008.

[12] J. Sanders and E. Kandrot. *Cuda by Example, An Introduction to General-Purpose GPU Programming.* Addison-Wesley, Boston, 2011.

[13] K. Shiraishi and T. Matsuoko. Wave Propagation Simulation using CIP Method of Characteristic Equations. *Communications in Computational Physics*, 3:121–135, 2008.

[14] T. Kunugi T. Utsumi and T. Aoki. Stability and Accuracy of the Cubic Interpolated Propagation Scheme. *Computer Physics Communications*, 101:9–20, 1997.

[15] F. Xiao T. Yabe and T. Utsumi. The Constrained Interpolation Profile Method for Multiphase Analysis. *Journal of Computational Physics*, 169:556–593, 2001.

[16] P.Y. Wang T. Aoki Y. Kadota T. Yabe, T. Ishikawa and F. Ikeda. A Universal Solver for Hyperbolic Equations by Cubic-Polynomial Interpolation: II. Two- and Three-Dimensional Solvers. *Computer Physics Communications*, 66:233–242, 1991.

[17] H. Takewaki and T. Yabe. The Cubic-Interpolated Pseudo Particle (CIP) Method: Application to Nonlinear and Multi-dimensional Hyperbolic Equations. *Journal of Computational Physics*, 70:355–372, 1987.

[18] T. Yabe Y. Ogata and K. Odagaki. An Accurate Numerical Scheme for Maxwell Equation with CIP-Method of Characteristics. *Communications in Computational Physics*, 1:311–335, 1996.

[19] T. Yabe and T. Aoki. A Universal Solver for Hyperbolic Equations by Cubic-Polynomial Interpolation: I. One- dimensional solver. *Computer Physics Communications*, 66:219–232, 1991.

APPENDIX

Appendix A

CUDA Review

Here we provide a brief review on compute unified device architecture (CUDA) C based on our experiences. We assume reader familiarity with C and/or C++ languages for the purposes of this discussion. In our experiences using CUDA for systems sufficiently large, say with at least a thousand variables, using existing GPU optimized libraries (CUBLAS, CULA, Thrust...) will result in decreased computational time. This means with minimal effort code previously implemented in C or C++ can take advantage of graphics card processing (GPU) computational power. In many cases it is just a matter of switching from LAPACK functions to CULA (the equivalent CUDA linear algebra package) functions. With this in mind we will briefly highlight basic aspects of CUDA C programming and give code examples for incorporating CUDA code into existing C++ code.

For the code provided in this review we use double precision computations which can only be run on GPUs with 1.3 compute capability or higher.

A.0.2 Programming Model

Implementing CUDA code involves running code on the *host* system which consists of one or more CPUs and the *device* which consists of one or more GPUs. This distinction is important because each system has its own distinct memory. The basic CUDA programming model is diagrammed in Figure A.1. Typically data starts on the *host* (CPU) and is transfered to the *device* (GPU). From here the problem is divided into computational blocks and threads. The computations are then carried out by the threads and then passed back to the *device* and potentially back to the *host*.

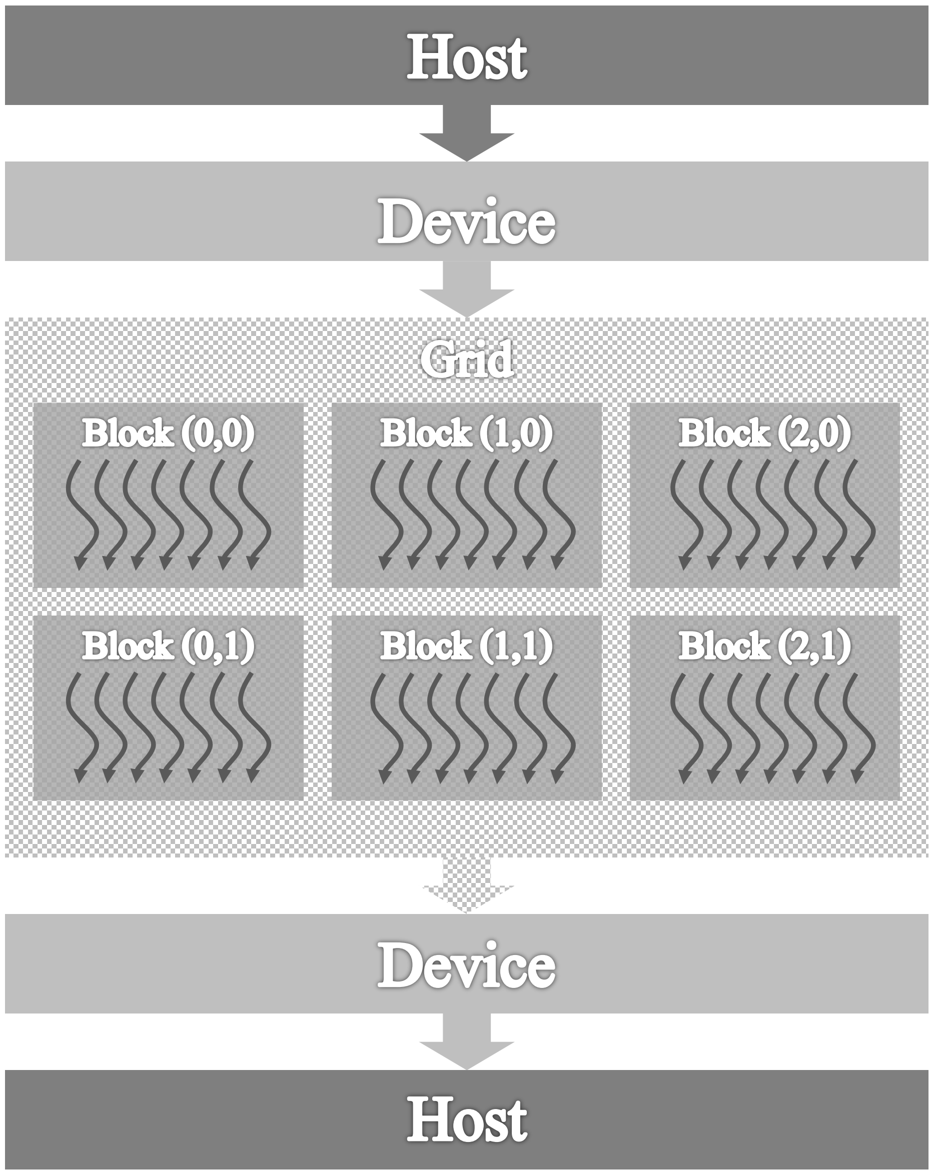


Figure A.1: CUDA programming model

To demonstrate the CUDA programming model we have an implementation of vector addition. We begin by initializing our data on the *host* and then begin by transferring our data onto the *device*. Data can be initialized on the *device* but we are assuming that it came from some other C++ routine. Once the data is on the *device* we will divide up our problem to perform our computations. Once our computations are complete we return the answer to the *host* as in Figure A.1. Below we have `vecAdd.cu` in its entirety and we will discuss the CUDA components of the code.

```
//vecAdd.cu
#define N 42

//CUDA kernel
__global__ void add(int *a, int *b, int size)
{
         int idx=threadIdx.x+blockIdx.x*blockDim.x;
        if(idx<size)
                b[idx]+=a[idx];
}

int main(void)
{


          //initialize host vectors
        int *a,*b,*ans;

        a=(int *)malloc(N*sizeof(int));
        b=(int *)malloc(N*sizeof(int));
        ans=(int *)malloc(N*sizeof(int));

        for (int i=0; i<N; i++)
        {
                a[i]=rand()%100;
                b[i]=rand()%100;
        }
```

```
          //initialize device vectors
        int *a_dev, *b_dev;
        cudaMalloc((void**) &a_dev, N*sizeof(int));
        cudaMalloc((void**) &b_dev, N*sizeof(int));

          //transfer data from host memory to device memory
        cudaMemcpy(a_dev, a, N*sizeof(int), cudaMemcpyHostToDevice);
        cudaMemcpy(b_dev, b, N*sizeof(int), cudaMemcpyHostToDevice);

          //CUDA kernel call
        add<<<6,7>>>(a_dev,b_dev,N);

          //transfer the data from device memory to host memory
        cudaMemcpy(ans, b_dev, N*sizeof(int), cudaMemcpyDeviceToHost);
}
```

The first lines of CUDA code are

```
        cudaMalloc((void**) &a_dev, N*sizeof(int));
        cudaMalloc((void**) &b_dev, N*sizeof(int));
```

where we allocate memory for the array `a_dev` on the device. We then transfer the arrays `a` and `b` which we initialized on the host to the device using

```
        cudaMemcpy(a_dev, a, N*sizeof(int), cudaMemcpyHostToDevice);
        cudaMemcpy(b_dev, b, N*sizeof(int), cudaMemcpyHostToDevice);
```

and we are now ready to run our CUDA kernel. The CUDA kernel is the list of tasks to be performed by each computational thread. Cuda kernels are functions that run on the *device* and use the label `__global__` or `__device__`. To run the kernel we have to specify the number of blocks and the number of threads CUDA should use. This subdivision becomes important for more complicated problems as threads in different blocks cannot communicate. In this example for our kernel call

```
        add<<<6,7>>>(a_dev,b_dev,N);
```

we use `<<<6,7>>>` to split our problem in to six blocks each running seven threads. The kernel for `vecAdd.cu` is

```
__global__ void add(int *a, int *b, int size)
{
         int idx=threadIdx.x+blockIdx.x*blockDim.x;
        if(idx<size)
               b[idx]+=a[idx];
}
```

and tells each thread to do the computation `b=a+b`. The indicies of each computation thread are indicated by

```
         int idx=threadIdx.x+blockIdx.x*blockDim.x;
```

which correspond to the indices of our vectors. We include `if(idx<size)` in case we wander outside of our allocated memory. After completing the vector addition we use

```
        cudaMemcpy(ans, b_dev, N*sizeof(int), cudaMemcpyDeviceToHost);
```

to copy our solution back to the *host* memory. To compile this code in double precision use the compiler flag `arch=sm_13` when calling the `NVCC` compiler. A few other things to keep in mind:

1. The number of threads should be a multiple of 64. Ideally one would use $128 - 256$ and the maximum threads per block is 512.

2. The number of threads per block should be a multiple of 32.

The first step in building optimized CUDA code is memory management. To begin with transferring data from the *host* to *device* and vice versa is expensive and should be minimized. When transferring global memory, coalescing should be used whenever possible to efficiently transfer data. Memory coalescing is when memory read by consecutive threads is combined by the hardware into several large memory reads, i.e., small memory transfers are batched into large ones. Other memory optimizations need to be taken with care due to memory supply constraints. Pinned memory may be used for higher bandwidth between the host and device but is limited. Once you are on the GPU for each multiprocessor there is register, shared, constant, and textured memory. Ideally we would want to use the memory on each multiprocessor which has higher bandwidth and lower latency but it is scarce. At the very minimum shared memory should be used to avoid repeated data transfers between the device and multiprocessor.

While memory management is important our focus has been to use the available tools to create optimized software. To that end we provide code examples that demonstrate CUDA library use.

A.0.3 Linear system example

For this example we use CUDA to solve the linear system $Ax = f$. The steps we need to take are

1. $A = QR$
2. $y = Q^T f$
3. $x = R^{-1}y$

After initializing your *device* variables `dA` with matrix $A \in \mathbb{R}^{m\times m}$ and `df` with vector $f \in \mathbb{R}^m$ allocate memory of size m for `dTAU`. Then use the commands

```
culaDeviceDgeqrf(m,m,dA,m,dTAU);
culaDeviceDormqr('R','T',m,1,m,dA,m,dTAU,df,m);
cublasDtrsv('u','n','n',m,dA,m,df,1);
```

and on completion `df` will contain the solution x to the linear system $Ax = f$. The first step is take the QR factorization of A using CULA (we are using version R14). The first CULA function `culaDeviceDgeqrf` computes the QR factorization of A and overwrittes `dA` with R above the diagonal. Below the diagonal are the elementary householder reflectors with scalar factors stored in `TAU`. The second CULA function `culaDeviceDormqr` performs the operation $Q^T f$ and stores the solution in `df`. The CUBLAS command `cublasDtrsv` computes $R^{-1}f$. The CUBLAS function `cublasDtrsv` can only be used to solve triangular systems.

A.0.4 Eigen code example

It may be beneficial to use C++ for some linear algebra computations especially if you are developing hybrid code. As part of developing software to create reduced systems it became beneficial to incorporate the Eigen C++ linear algebra library. After using CUDA to create a reduced system we used Eigen for our computations on that reduced system. The NVCC compiler does not handle template libraries such as Eigen well so to get around this you may need to divide up your code into seperate C/C++ and CUDA files and use a C/C++ compiler for the C/C++ parts. For example if you would like to use Eigen to calculate the eigenvalues of a matrix you would first create a separate C++ file eigA.cpp below:

```
//  eigA.cpp
#include <complex>
#include <Eigen/Core>
#include <Eigen/Dense>
#include <Eigen/Eigenvalues>

using namespace Eigen;
using namespace std;

extern "C" void eigA(double *, int);

void eigA(double *A, int n)
{


        Map<Matrix<double,Dynamic,Dynamic>> Aeig(A,n,n);
        EigenSolver<MatrixXd> ee(Aeig);
        VectorXcd evals=ee.eigenvalues();

         //display eigenvalues
        std::cout << "Here are the eigenvalues of A:\n"
                                                 <<evals<< std::endl;
}
```

To access this function from within a *.cu file add

```
        extern "C" void eigA(double *, int);
```

to the preamble then inside your function simply call

```
eigA(A,n);
```

to run the eigenvalue solver be sure that `A` is a *host* variable. In this example code

```
Map<Matrix<double,Dynamic,Dynamic>> Aeig(A,n,n);
```

maps the `host` memory stored in `A` into an Eigen expression `Aeig`. Then the eigen values are computed using the commands

```
EigenSolver<MatrixXd> ee(Aeig);
VectorXcd evals=ee.eigenvalues();
```

and displayed. This same procedure of dividing the code into separate files for compilation will work to access any functions in a C/C++ template library.