Using Calculus of Variations to Optimize Paths of Descent through Ski Race Courses

by

Jason W. Christopher

Submitted to the Department of Physics
in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Author .....
Department of Physics
May 16, 2005

Certified by .....
Mehran Kardar
Professor
Thesis Supervisor

Accepted by .....
Professor David E. Pritchard
Senior Thesis Coordinator, Department of Physics

Using Calculus of Variations to Optimize Paths of Descent through Ski Race Courses

by

Jason W. Christopher

Submitted to the Department of Physics
on May 16, 2005, in partial fulfillment of the
requirements for the degree of
BACHELOR OF SCIENCE

Abstract

The goal of ski racing is to pass through a series of gates as quickly as possible. There are many paths from gate to gate, but there is only one path that is fastest. By knowing what the fastest path is, a racer could shave tenths of seconds off his or her time. That is a tremendous amount of time considering that races are won by hundredths of a second. This thesis attempts to calculate the fastest path through a ski race course using several simplifications such as neglecting friction. The method of attacking this problem is to modify the Brachistochrone problem. It is found that it is best if the skier places the apex of the turn at the gate, and that turning more after the gate is better than turning more above the gate. In the case of a rhythmical course, it is found that turning more below the gate is still true, but not as evident. Instead the optimal path appears more symmetric about the gate.

Thesis Supervisor: Mehran Kardar
Title: Professor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter begins with a brief description of the sport of ski racing. The idea is to give the reader a sense of how the calculations and equations pertain to real life situations. The chapter continues by describing the motivations and history of attempts to calculate the fastest path through a course.

Chapter two describes the Brachistochrone problem, and explains how the solution to that problem can be extended to find the fastest path through a ski race course through the use of cost functions. Several cost functions are proposed in chapter two, and the differential equation that results from each cost function is given. An analysis of the differential equations is conducted in chapter three, where the solutions of the differential equations are compared with the solution to the Brachistochrone problem. The differential equations are then used to calculate optimal paths through a set of example courses. These calculations allow for the comparison of cost functions, and enable a discussion of characteristics for optimal paths to ensue.

Chapter four concludes the thesis with a discussion of how the research can be continued. Methods for obtaining simpler differential equations are discussed, as well as how to include more complicated factors such as friction, aerodynamic drag and terrain in the computation of the optimal path.

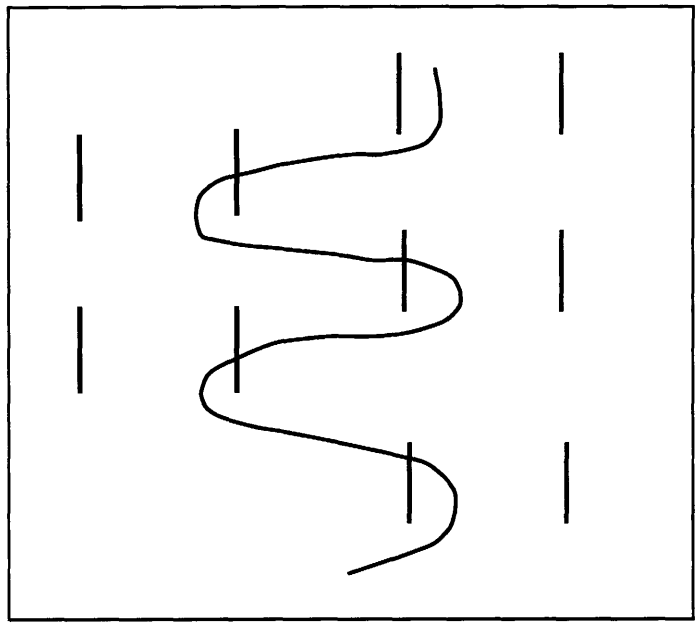

Figure 1-1: Diagram of a set of gates and a path that goes through all of them. The path is the black line, and the markers are the short blue and red lines. Each gate is a pair of markers located horizontally from one another The first gate is made of the two red markers in the upper right corner. The second gate is made of the two red markers in the upper left corner, and so forth.

1.1 Ski Racing

The most relevant aspects of ski racing are those concerning a race course. A race course consists of a set of gates, a start and a finish line. Each gate is made of two markers. The markers are plastic poles that have been screwed into the snow. To pass through a gate, a racer must cross an imaginary line drawn between the two markers. In order for a racer to complete a course, he or she must go through the start, every gate and cross the finish line. Figure 1-1 shows an example path through a series of gates. In this paper, a gate will be considered to be a single point through which the path must pass, which is a reasonable approximation because for the most part the fastest path is as close to one of the markers of a gate as possible.

The second most relevant aspect of ski racing is the way a ski turns. A ski is made with a wide tip and tail and narrow center. When a skier tips the ski on its edge the ski will bend into an arc. If the whole edge of the ski follows the same path along

the snow, then the ski will move with less friction. This is called carving and causes the ski to turn. If the whole edge of the ski is not following the same path along the snow, then the ski is said to be sliding. Sliding creates much more friction than carving, because some of the skier's kinetic energy is transfered into kinetic energy in the snow. The sliding of the ski is what causes snow to be kicked up in the form of a spray. Optimally a ski racer is always making the ski carve.

1.2 Motivation

The fastest skiers not only ski well, but also have an instinctive feel for what is the fastest path through a race course. Ski races are won by hundredths of a second, and the slightest difference in path between racers in could determine the winner of the race.

In 1988 an experiment was conducted at Squaw Valley Ski Resort to measure the effect of path on ski racer times [6]. The experiment was conducted on a 30 to 40 degree slope on which three racers of various sizes skied between two points. In the study there were two paths. One path was a straight line between the points, and the second path was an arc of a cycloid. It was found in fifteen out of eighteen trials that the cycloid path was faster than the straight line path. Further, in the three cases where the cycloid path was slower than the straight line path, the experimenters noticed considerable sliding in the racer's skis, which would slow the skier down and make the time slower regardless of path. The cycloid path was as much as .54 seconds faster than the straight line path.

Because the path taken through a course is so important to a racer's success, coaches have been attempting to teach racers how to find the fastest path through a course since ski racing began. Presently, racers are taught a set of tactics, which are rules of thumb used by a racer to determine the optimal path through the course. A racer is usually taught tactics about how they should adjust their path through a course depending on the steepness of the course and how to adjust their path over terrain features such as bumps. However, the tactics taught are only based on

speculations made from years of watching ski racing. Currently, there is no definitive answer to the question, "What is the fastest path through a given course?"

Being able to calculate the fastest path through a ski course would be a tremendous asset to coaches and racers, because it would enable them to weed out erroneous tactics and to discover new, better tactics. Further, by making measurements of the terrain and position of gates in a course, it would be possible to calculate the fastest path through a specific course on race day. Clearly, knowing the fastest path through the course would give a racer considerable advantage over the competition. It is even conceivable that calculations of the fastest path through courses could aid ski design and manufacturing in the production of faster, better skis.

1.3 History

Finding the fastest path through a ski race course is very similar to the Brachistochrone problem. In the Brachistochrone problem there are two points, A and B, that exist in a uniform gravitational field such that A is at a higher potential than B. An example of this can be seen in figure 1-2. The Brachistochrone problem is the problem of finding the fastest path from A to B without doing any work.

There are two factors to consider when finding the fastest path from A to B: the length of the path and the speed that can be attained on the path. A shorter path will take less time to traverse at a given speed, and the more speed a path offers, the faster a given distance can be traversed. Speed is created on a path by moving in the direction of the gravitational field, and a shorter distance is achieved by moving in the direction of the end point. If the gravitational field does not point in the direction to the end point, then the direction the path should move in must optimally balance the effects of increased speed with increased distance.

The Brachistochrone problem was solved in 1696 by Johann Bernoulli [3]. In the process of solving the Brachistochrone problem he made major break throughs in the development of the calculus of variations. He eventually found the solution to be an arc of a cycloid, which will be derived in chapter two. In 1975 Professor Neil Ashby

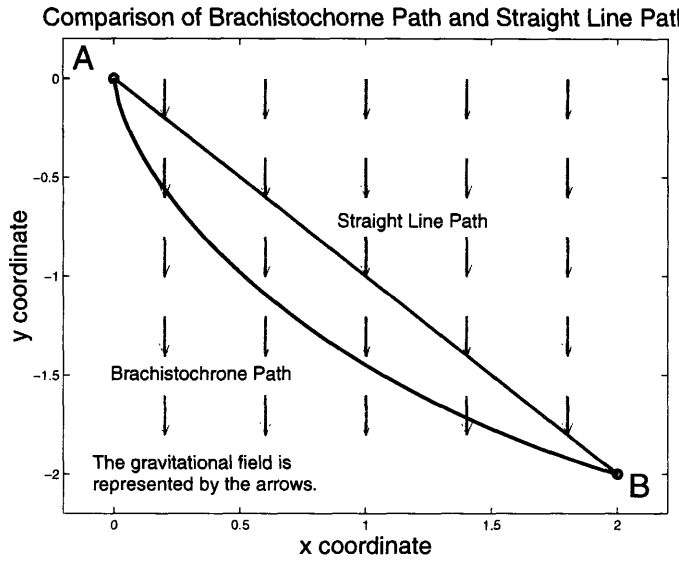

Figure 1-2: Diagram of an example of the Brachistochrone problem. A is the starting point and B is the end point. The gravitational field is depicted by the purple arrows, and both the straight line and fastest paths are shown.

at the University of Colorado at Boulder considered adding coulomb friction to the Brachistochrone problem [1]. He successfully solved the problem, which opened the door for further study into variations on the Brachistochrone problem. In recent years there have been papers extending the Brachistochrone problem to handle multi-body systems [2], non-conservative fields [7] and non-holonomic constraints [9]. In 1997 Vratanar and Saje solved the Brachistochrone problem while including a frictional force that is proportional to velocity. A frictional force that is proportional to velocity is a good first approximation to aerodynamic drag [7].

Extending the Brachistochrone problem to find the fastest path through a ski race course requires the addition of friction, aerodynamic drag, the constraint that the path be on the surface of a mountain, and being able to find the fastest path through multiple points. Friction is a relatively small factor in the case of ski racing. Skis glide so well that the coefficient of friction is as little as .02 [6]. Aerodynamics is only important in the fastest disciplines of ski racing, Super-G and Downhill, which limits its applicability. Constraining the solution to the surface of a mountain can be achieved through the use of undetermined Lagrange multipliers and a holonomic

constraint, which are subjects treated in most books on classical mechanics such as Goldstein's [4]. Because of the importance and lack of study on the subject, this thesis focuses on extending the Brachistochrone problem to finding the fastest path through multiple points, which will be referred to as the Multi-Point problem.

Chapter 2

Method of Solving the Multi-Point Problem

This chapter describes how the Brachistochrone problem is solved, which gives the necessary background for understanding how cost functions are used to extend the Brachistochrone problem to solve the Multi-Point problem. Several different cost functions are proposed for each of two different ways of parameterizing the problem, parameterization by x coordinate and by time. For each cost function that is proposed the differential equation that results from using that cost function is shown.

2.1 Solving the Brachistochrone Problem

The technique used to solve the Brachistochrone problem in this paper follows closely to that used by Weisstein [8], but is a bit more general by including effects due to non-zero initial velocity.

Let the initial point of the problem be A and the final point be B. The time it takes to traverse a given path can be calculated by the integral

$$\Delta t = \int_{t_A}^{t_B} dt = \int_0^{\ell} \frac{ds}{v}, \tag{2.1}$$

where s is the path length, v is the velocity and ℓ is the length of the entire path.

The only two forms of energy in this problem are kinetic and gravitational. Conservation of energy implies

$$\frac{1}{2}mv_A^2 + mgy_A = \frac{1}{2}mv^2 + mgy. \tag{2.2}$$

v_A and y_A are the velocity and y-coordinate at the point A, v and y are the instantaneous velocity and y-coordinate and m is the mass of the skier traversing the path. Using equation 2.2 to solve for v and substituting the result into equation 2.1 with the differential path length written as $\sqrt{1+y'^2}dx$ gives

$$\Delta t = \int_{x_A}^{x_B} \sqrt{\frac{1+y'^2}{2g\left(y_A - y\right) + v_A^2}}dx. \tag{2.3}$$

Letting $M = 2gy_A + v_A^2$, and using the calculus of variations to find the extremum of the integral yields the differential equation

$$\left(M - 2gy\right)y'' - g\left(1+y'^2\right) = 0. \tag{2.4}$$

Since the integrand of equation 2.3 is not directly dependent on time, equation 2.4 can be integrated once using the Beltrami identity [8] to find

$$\left[\left(1+y'^2\right)\left(M-2gy\right)\right]^{-1} = k_1. \tag{2.5}$$

Solving for y' gives two equations,

$$y' = \pm\sqrt{\frac{1-k_1^2\left(M-2gy\right)}{k_1^2\left(M-2gy\right)}}. \tag{2.6}$$

If the final point, B, is to the right of the initial point, A, then y' should be negative. If the final point is to the left of the initial point, then y' should be positive. Equation 2.6 can be most easily be integrated by using separation of variables and making the

substitution $2k_1^2(M - 2gy) = 1 - \cos\theta$. The result is

$$x = \mp\frac{\sin\theta - \theta}{4gk_1^2} + k_2 \tag{2.7}$$

$$y = \frac{M}{2g} - \frac{1 - \cos\theta}{4gk_1^2}. \tag{2.8}$$

k_1 and k_2 are solved for by requiring that the solution goes through the points A and B. Equations 2.7 and 2.8 describe a cycloid as mentioned in chapter one.

2.2 Extending the Brachistochrone Problem

The solution to the Brachistochrone problem will find the fastest path between any two points in a uniform gravitational field with a given initial velocity. Knowing how to find the fastest path between any two points, suggests that the Multi-Point problem could be solved by iteratively using the solution on two points of the problem at a time.

For example, consider the problem of finding the fastest path through three points, A, B and C such as in Figure 2-1. The solution to the Brachistochrone problem can be used to find the fastest path from A to B, P_1, and can be used to find the fastest path from B to C, P_2. Appending path P_2 to path P_1 creates the fastest path through all three points. However, the slope of the path is discontinuous at the point B. The discontinuity of the slope corresponds to an instantaneous change of direction, which would be experienced by the skier as an infinite amount of force. Though the resulting path is the fastest, it is unphysical.

In order to append two paths together to get a realistic path, the two paths must have the same slope at the point they are being joined. If the solution to the Brachistochrone problem enabled the specification of the slope at the end points of the path, then it could be used iteratively to solve the Multi-Point problem without discontinuities in the slope. Specification of the slope at the end points requires more undetermined coefficients in the solution to the differential equation that describes the path. The only way to gain more undetermined coefficients is to make the differential

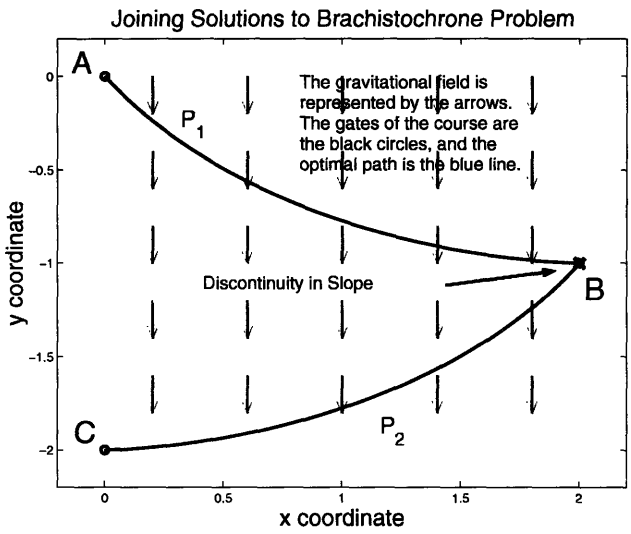

Figure 2-1: Diagram of how the solution to the Brachistochrone problem could be used to solve the Multi-Point problem through three points A, B and C. The solution to the Brachistochrone problem is used to calculate the fastest path from A to B, P_1, and the fastest path from B to C, P_2. The discontinuity in the slope at point B makes this solution unphysical, because the skier would experience the discontinuity as an infinite amount of force.

equation of higher order.

The differential equation resulting from the use of calculus of variations can be made to be of higher order by adding an appropriate cost function. The cost function must be a function of y'', so that the resulting differential equation is at least fourth order. Several different cost functions are considered in the following sections with two different parameterizations of the problem. In the fist section, y is parameterized with respect to x, and in the second section, both x and y are parameterized with respect to time, t.

2.3 Using the x Coordinate as the Parameterization Variable

The simplest choice for a cost function that is dependent on y'' is y''. However, using just y'' would cause the solutions to tend to bend in an arch so that $y'' < 0$. Similarly

using $-y''$ would cause the solutions to bend into a "U" shape so that $y'' > 0$. Squaring y'' will eliminate any preference in the direction that path curves, so the first cost function considered is $c^2y''^2/2$, which will be referred to as cost function A. The coefficient c is used to vary the strength of the cost function. In the limit as c goes to zero, the solution to the Brachistochrone problem should be regained.

Adding the cost function to equation 2.1 gives

$$S = \int \left[\sqrt{\frac{1+y'^2}{2g\left(y_A - y\right) + v_A^2}} + \frac{c^2}{2} y''^2 \right] dx. \tag{2.9}$$

Using the calculus of variations to to find the path that extremizes the integral results in the differential equation

$$c^2 y^{(4)} \left[\left(M - 2gy\right) \left(1 + y'^2\right) \right]^{3/2} = \left(M - 2gy\right) y'' - g\left(1 + y'^2\right). \tag{2.10}$$

Comparing equation 2.10 with equation 2.4 it is easy to see that when $c = 0$ the differential equation of the Brachistochrone problem is regained.

Centripetal acceleration is another natural choice for a cost function. It is a function of y'', and centripetal acceleration contributes to the normal force exerted by the skier on the snow causing an increase in friction. Further it is more difficult for a skier to ski on a path with lots of centripetal acceleration. Balance is tougher to maintain and the skier has to use a lot of strength to press with their legs against the increased force. Hence the amount of centripetal acceleration a skier can handle will depend on the racer's ability. In this case c can be interpreted as a measure of the ability of the racer. A good racer will be able to handle more centripetal force, so c will be lower making the centripetal acceleration a smaller factor in determining the path and causing the path to be closer to the solution to the Brachistochrone problem. Since the Brachistochrone solution is the fastest, the result is that a better skier will be able to take a faster path.

The first challenge in using centripetal acceleration as a cost function is writing it as a function of y. Let $\vec{\gamma}$ denote a path parameterized by some variable, and let

$\vec{\gamma}'$ and $\vec{\gamma}''$ denote differentiation of the path with respect to that variable. Then the curvature is given by the equation [5]

$$\mathcal{K} = \frac{|\vec{\gamma}'' \times \vec{\gamma}'|}{|\vec{\gamma}'|^3}. \tag{2.11}$$

For this problem $\vec{\gamma} = x\hat{x} + y(x)\hat{y}$, which upon substitution into equation 2.11 gives

$$\mathcal{K} = \left[\frac{y''^2}{(1+y'^2)^3}\right]^{1/2}. \tag{2.12}$$

Using equation 2.2 to solve for v the centripetal acceleration can be written solely as a function of y

$$A_c = \frac{v^2}{R} = v^2\mathcal{K} = \left[2g\left(y_A - y\right) + v_A^2\right]\left[\frac{y''^2}{(1+y'^2)^3}\right]^{1/2}. \tag{2.13}$$

By making the cost function equal to the square of the centripetal acceleration the square root is removed, which will make it easier to apply the calculus of variations to derive the differential equation. The square of the centripetal acceleration can be integrated with respect to time or distance. The longer in time the centripetal acceleration lasts, the greater the effect it will have on the skier's balance and strength. Hence integration with respect to time gives a measure of centripetal acceleration's effect on the skier. The longer in distance that the centripetal acceleration lasts, the greater the effect it will have on friction, so the integral of centripetal acceleration with respect to distance is a good measure of the effect of friction. Both are effects that should be minimized.

When minimizing the effect of friction the integral is

$$S = \int \left[\sqrt{\frac{1+y'^2}{2g\left(y_A - y\right) + v_A^2}} + \frac{c^2}{2}\left(2g\left(y_A - y\right) + v_A^2\right)^2 \frac{y''^2}{(1+y'^2)^{5/2}}\right] dx. \tag{2.14}$$

For clarity

$$\frac{c^2}{2}\left(2g\left(y_A - y\right) + v_A^2\right)^2 \frac{y''^2}{(1+y'^2)^{5/2}} \tag{2.15}$$

will be called cost function B. When calculus of variations is applied on equation 2.14, a messy fourth order differential equation emerges that offers little insight into he problem. Let $U = 1 + y'^2$, $D = M - 2gy$ and $F(a,b) = ay''D + bgU$, then the differential equation is

$$\begin{aligned} y^{(4)} &= y''(UD)^{-2}\left[y''D - 2y'^2F(3,1)\right]F(5/2,4) + y'y^{(3)}(UD)^{-1} \\ &\quad \times F(15/2,8) + gy''^2D^{-1} + c^{-2}D^{1/2}UF(1,-1) \end{aligned} \tag{2.16}$$

Similarly, when minimizing the effect of centripetal acceleration on the skier's balance and strength the integral is

$$S = \int \left[\sqrt{\frac{1+y'^2}{2g(y_A - y) + v_A^2}} + \frac{c^2}{2}\left(2g(y_A - y) + v_A^2\right)^{3/2}\frac{y''^2}{(1+y'^2)^{5/2}}\right]dx, \tag{2.17}$$

where

$$\frac{c^2}{2}\left(2g(y_A - y) + v_A^2\right)^{3/2}\frac{y''^2}{(1+y'^2)^{5/2}} \tag{2.18}$$

is called cost function C. Using the calculus of variations to extremize this integral gives a messy fourth order differential equation similar to equation 2.16. With the same conventions as those used in equation 2.16, the differential equation is

$$\begin{aligned} y^{(4)} &= -c^{-2}UD^{-3}F(-1,1) + \frac{3}{2}gy''^2D^{-1} + (UD)^{-2}\left(-y'F(7,1)\left[y'y''F(3,5/2)\right.\right. \\ &\quad \left. -y^{(3)}UD\right] + UD\left[\left(y''^2 + y'y^{(3)}\right)F(3.5/2) + y'y^{(3)}F(1/2.2)\right. \\ &\quad \left.\left. +y'^2y''^2\right]\right) \end{aligned} \tag{2.19}$$

2.4 Using time as the Parameterization Variable

Parameterization by time offers several advantages over parameterization by x. Sometimes the fastest path is one where the path initially moves to the left and then hooks back to the right. In that case there will be a region where the path is double valued for a given x-coordinate. Parameterization by time avoids the double value problem. Eventually the problem needs to be extended to three dimensions, in which case it is

preferable to use a parameterization that does not break the symmetry between the two horizontal coordinates. Time is such a parameterization.

Another nice feature about parameterization with time is that

$$\frac{\dot{x}^2+\dot{y}^2}{2g\left(y_A-y\right)-v_A^2}=1. \tag{2.20}$$

Because of equation 2.20 it is possible to interchange $\dot{x}^2+\dot{y}^2$ with $M-2gy$ at any point in a calculation, or to multiply any equation by equation 2.20. For example, in the time parameterized version of the Brachistochrone problem, the integral that is extremized is

$$\Delta t=\int\left[\frac{\dot{x}^2+\dot{y}^2}{2g\left(y_A-y\right)+v_A^2}\right]^N dt, \tag{2.21}$$

where N can be any real number. No matter what N is, equation 2.21 will yield upon using the calculus of variations the differential equation of the Brachistochrone problem, equation 2.4.

The same ideas as were applied in section 2.3 to come up with cost functions are used to come up with good cost functions when the path is parameterized by time. Analogous to the y'^2 cost function when x was the parameterization variable, is the cost function $\ddot{x}^2+\ddot{y}^2$ when time is the parameterization variable. This cost function will be referred to as cost function D. When cost function D is added to equation 2.21, the integral becomes

$$S=\int\left(\left[\frac{\dot{x}^2+\dot{y}^2}{2g\left(y_A-y\right)+v_A^2}\right]^N+\frac{c^2}{2}\left(\ddot{x}^2+\ddot{y}^2\right)\right)dt. \tag{2.22}$$

Using calculus of variations for both x and y yields two fourth order differential equations

$$x^{(4)} = \frac{2}{c^2}\left(M-2gy\right)^{-1}\left(\frac{2g\dot{x}\dot{y}}{M-2gy}+\ddot{x}\right) \tag{2.23}$$

$$y^{(4)} = \frac{2}{c^2}\left(M-2gy\right)^{-1}\left(g\frac{\dot{y}^2-\dot{x}^2}{M-2gy}+\ddot{y}\right), \tag{2.24}$$

where N has been taken to be 1.

The other two cost functions considered in section 2.3 were integrals of the centripetal acceleration. The curvature of a path that is parameterized by time is

$$\mathcal{K} = \sqrt{\frac{(\dot{y}\ddot{x} - \dot{x}\ddot{y})^2}{(\dot{x}^2 + \dot{y}^2)^3}} = \frac{|\dot{y}\ddot{x} - \dot{x}\ddot{y}|}{v^3}. \tag{2.25}$$

Using equation 2.25, the centripetal acceleration is

$$A_c = \frac{v^2}{R} = v^2\mathcal{K} = \frac{|\dot{y}\ddot{x} - \dot{x}\ddot{y}|}{v}. \tag{2.26}$$

With time as the parametric variable $ds = vdt$, as a result $A_c ds$ and $A_c^2 ds$ will have square roots in them. The troubles of dealing with the square roots is avoided by considering the case where the cost function is an integral of the centripetal acceleration over time. Adding the integral of the centripetal acceleration over time to equation 2.21 gives

$$S = \int \left(\left[\frac{\dot{x}^2 + \dot{y}^2}{M - 2gy} \right]^N + \frac{c^2}{2} \frac{(\dot{y}\ddot{x} - \dot{x}\ddot{y})^2}{M - 2gy} \right) dt. \tag{2.27}$$

Using calculus of variations to extremize this integral results in two fourth order differential equations that can not decouple x from y. The reason why x and y can not be decoupled stems from the $\dot{y}\ddot{x} - \dot{x}\ddot{y}$ component of the equation. This component contains the highest derivatives of the integrand, and thus will be responsible for the creation of the fourth order terms of the differential equation. The term with the fourth order derivatives will be of the form $\dot{y}x^{(4)} - \dot{x}y^{(4)}$ whether x is varied or y. Hence the term will be in both equations and it will be possible only to solve for $\dot{y}x^{(4)} - \dot{x}y^{(4)}$, but not for $x^{(4)}$ independent of $y^{(4)}$ or $y^{(4)}$ independent of $x^{(4)}$. For the

sake of completeness the differential equations are

$$
\begin{aligned}
x^{(4)} &= -\left(\dot{x}\dot{y}\right)^{-1}\left(M-2gy\right)^{-2}\left[4g\dot{y}\left(2\left[g\dot{x}\dot{y}+\left(M-2gy\right)\ddot{x}\right]\left(\dot{y}\ddot{x}-\dot{x}\ddot{y}\right)\right.\right. \\
&\quad \left.\left.+\left(M-2gy\right)\dot{x}\left(\dot{y}x^{(3)}-\dot{x}y^{(3)}\right)\right)+\left(M-2gy\right)^{2}\left[\left(5\dot{y}\ddot{x}-\dot{x}\ddot{y}\right)x^{(3)}-4\dot{x}\ddot{x}y^{(3)}\right]\right] \\
&\quad -\frac{2}{c^{2}\dot{x}\dot{y}}\left[g\frac{\dot{y}^{2}-\dot{x}^{2}}{M-2gy}+\ddot{y}\right]+\frac{\dot{x}}{\dot{y}}y^{(4)} \qquad (2.28) \\
y^{(4)} &= \left(\dot{x}\dot{y}\right)^{-1}\left(M-2gy\right)^{-2}\left[4g\dot{y}\left(2\left[g\dot{y}^{2}+\left(M-2gy\right)\ddot{y}\right]\left(\dot{y}\ddot{x}-\dot{x}\ddot{y}\right)\right.\right. \\
&\quad \left.\left.+\left(M-2gy\right)\dot{y}\left(\dot{y}x^{(3)}-\dot{x}y^{(3)}\right)\right)+\left(M-2gy\right)^{2}\left[\left(\dot{y}\ddot{x}-5\dot{x}\ddot{y}\right)+4\dot{y}\ddot{y}x^{(3)}\right]\right] \\
&\quad -\frac{2}{c^{2}\dot{x}\dot{y}}\left[\frac{2g\dot{x}\dot{y}}{M-2gy}+\ddot{x}\right]+\frac{\dot{y}}{\dot{x}}x^{(4)}. \qquad (2.29)
\end{aligned}
$$

Since these equations can not be decoupled, because of the $\dot{y}x^{(4)}-\dot{x}y^{(4)}$ component, the same problem arises in trying to use the centripetal acceleration integrated over distance as the cost function. Because neither $x^{(4)}$ nor $y^{(4)}$ can be solved for, there is no way for Matlab to solve the system of differential equations. Hence, these differential equations are not analyzed in chapter three.

Chapter 3

Analysis of Differential Equations

Chapter three begins with a section describing how the calculations in this thesis were carried out. It discusses the programming issues involved with the calculations and describes how the programs used work. The chapter continues by discussing the results of the calculations. One section analyzes how well the programs are working and how well the differential equations determine the fastest path by comparing the results with the analytical solution to the Brachistochrone problem. Another section looks at the paths the differential equation creates in various situations in an attempt to confirm or refute some ski racing tactics.

3.1 Programming

All of the computation for this thesis was done on Matlab version 6.1, release 12.1. Matlab has sophisticated software packages for solving differential equations numerically, calculating numerical integrals, finding local minima of arbitrary functions and solving boundary value problems. All of these tools proved useful in creating a set of functions that would calculate the fastest path through a set of points, and analyze the results.

Three types of programs were created. One set of programs finds the optimal path between two points given initial and final slopes and a specific differential equation. These programs setup the problem for Matlab's boundary value problem solver to

handle, so they are called BVP setup programs. A second set of programs uses a given BVP setup program to find the optimal path through multiple points. To find the optimal path, this set of programs numerically integrates a given path to figure out how long it takes to traverse the path, and then uses that information to adjust the slope at each of the points that the path is constrained to pass through. The slopes are varied until the minimal time is found. These programs used Matlab's local minimum finding software, and are called slope finding programs.

The third set of programs are those used to analyze the results obtained from the previous two programs. There are two programs in this set. brach-compare compares the optimal paths calculated by two different BVP setup programs by plotting both paths, and the difference between the paths. The program also calculates useful statistics on the difference between the paths such as the maximum and minimum differences, the mean and standard deviation. The other program in this set,move-middle-pt, uses a given BVP setup program and slope finding program to compute the optimal path through three points. The middle point is moved horizontally and the optimal path is calculated for several different situations.

Appendices A, B and C describes the BVP setup programs, slope finding programs and analysis programs respectively in greater detail. The appendices also have example code.

3.2 Analyzing Path Validity

The best way to determine if the programs are working and how well the differential equations determine the fastest path is by comparing the results from the programs with the analytical solution to the Brachistochrone problem. The better the program is working and the better the differential equation, the closer its solution will be to the analytical solution to the Brachistochrone problem. brach-compare was used to compare the analytical solution to the solutions calculated from the differential equations found in chapter two.

Differential equation 2.10, which resulted from adding cost function A to the

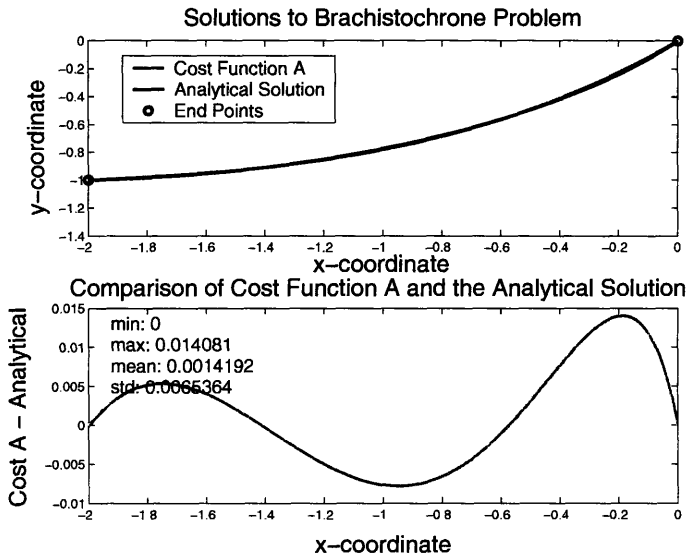

Figure 3-1: Comparison of the solution calculated by equation 2.10, in red, and the exact analytical solution calculated by equations 2.7 and 2.8, in blue. The lower graph shows the difference between the analytic solution and the solution calculated by equation 2.10 in magenta. For this calculation $c = 1$.

Brachistochrone problem, is compared to the analytical solution of equations 2.7 and 2.8 in figures 3-1, 3-2 and 3-3, where the strength parameter of the cost function is 1, 10 and 100 respectively. These figures show how well cost function A can be used to match the analytical solution. The figures also show the effects of varying the cost function strength parameter, c. Table 3.1 summarizes this information, and shows that as c is decreased the maximum difference between the analytical solution and the solution calculated using cost function A remains constant. However, as c decreases the difference on average gets slightly larger. The change is not much, but it is the opposite of what is expected. This oddity could be explained by the fact that as c is decreased the solution gets increasingly difficult to calculate and Matlab has to relax some of its tolerances to compute the answer. Or it could be that differential equation found using cost function A does a poor job of yielding the exact solution as c is decreased toward zero.

The differential equation that resulted from adding cost function B to the Brachistochrone problem, equation 2.16, is compared to the analytical solution in Figure 3-4

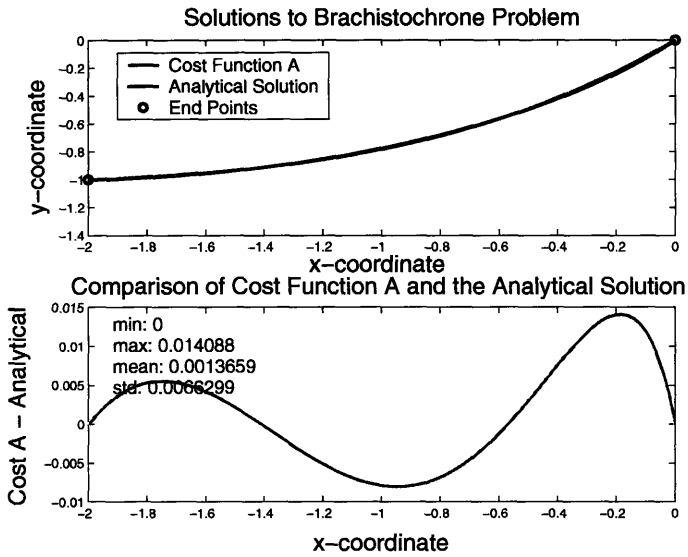

Figure 3-2: Comparison of the solution calculated by equation 2.10, in red, and the exact analytical solution calculated by equations 2.7 and 2.8, in blue. The lower graph shows the difference between the analytic solution and the solution calculated by equation 2.10 in magenta. For this calculation $c = 10$.

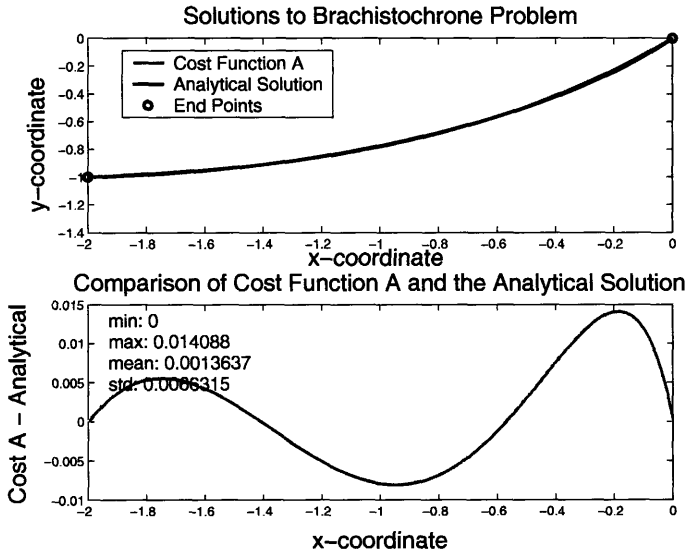

Figure 3-3: Comparison of the solution calculated by equation 2.10, in red, and the exact analytical solution calculated by equations 2.7 and 2.8, in blue. The lower graph shows the difference between the analytic solution and the solution calculated by equation 2.10 in magenta. For this calculation $c = 100$.

c	Max Difference	Mean Difference
1	0.014081	0.0014192
10	0.014088	0.0013659
100	0.014088	0.0013637

Table 3.1: Table showing how the maximum difference and mean difference between the solutions calculated by equation 2.10 and the analytical solution change as c is varied.

where $c = 1$. Because of the complexity of the differential equation, Matlab could not be used to compute a path for any other values of c. The maximum difference between the analytic solution and the solution calculated with cost function B is less than it was for cost function A, but on average the path computed with cost function B differed from the analytic solution more than the paths computed with cost function A.

Figures 3-5, 3-6 and 3-7 show how the differential equation that resulted from adding the cost function C to the Brachistochrone problem, equation 2.19, compares to the analytical solution, where the strength parameter of the cost function is 1, 10 and 100 respectively. Table 3.2 shows how the maximum difference and mean difference between the analytic solution and the solution computed with cost function C varies as c is changed. As c is decreased the maximum difference and mean difference decrease as is expected. Since the differential equation that results from using cost function C is much more complicated than the differential equation that results from using cost function A, it is expected that Matlab would have a tougher time calculating the paths for cost function C's differential equation than for cost function A's differential equation. However, the relationship between c and the maximum difference and average difference is as expected for cost function C, but opposite of what is found for cost function A. The paths for cost function C had smaller maximum and average differences than the paths for cost function A, which suggests that cost function C is a better cost function than A.

Parameterization with respect to time and using cost function D gives the differential equations 2.23 and 2.24. The solutions that these differential equations offer are

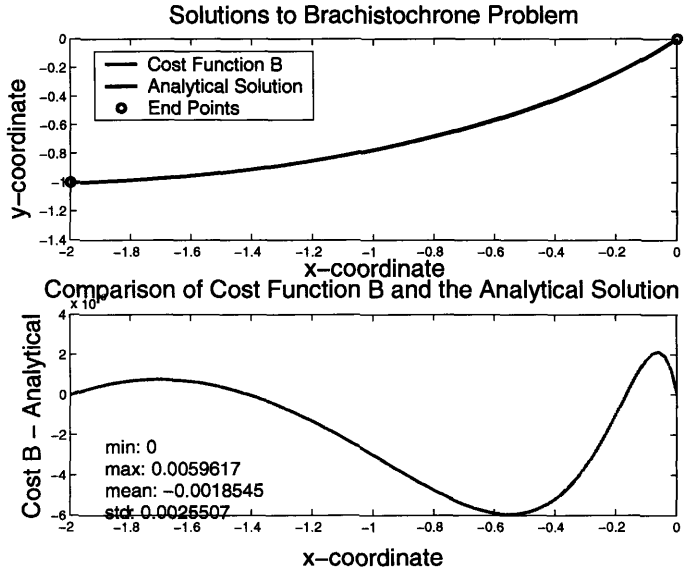

Figure 3-4: Comparison of the solution calculated by equation 2.16, in red, and the exact analytical solution calculated by equations 2.7 and 2.8, in blue. The lower graph shows the difference between the analytic solution and the solution calculated by equation 2.16 in magenta. For this calculation $c = 1$.

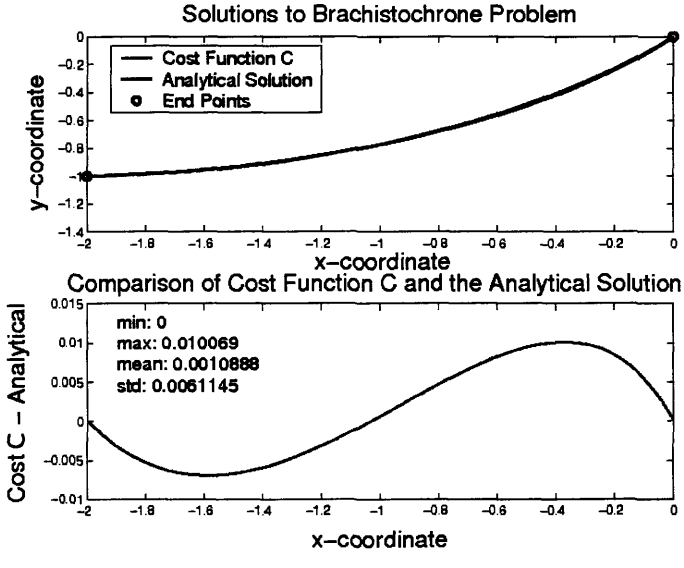

Figure 3-5: Comparison of the solution calculated by equation 2.19, in red, and the exact analytical solution calculated by equations 2.7 and 2.8, in blue. The lower graph shows the difference between the analytic solution and the solution calculated by equation 2.19 in magenta. For this calculation $c = 1$.

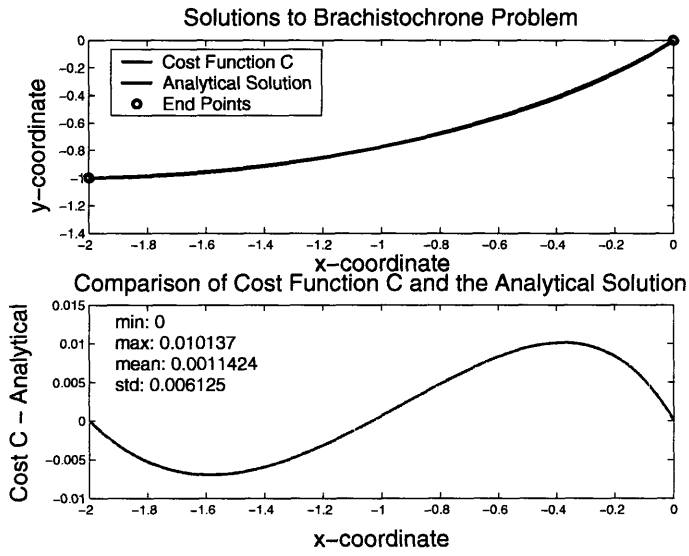

Figure 3-6: Comparison of the solution calculated by equation 2.19, in red, and the exact analytical solution calculated by equations 2.7 and 2.8, in blue. The lower graph shows the difference between the analytic solution and the solution calculated by equation 2.19 in magenta. For this calculation $c = 10$.

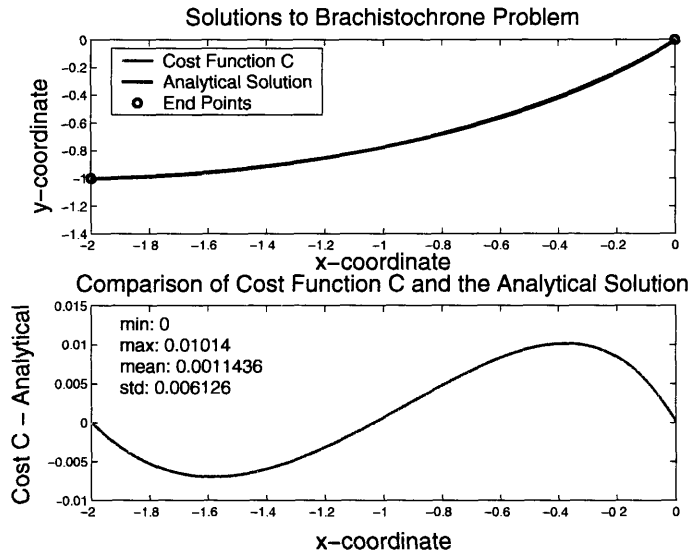

Figure 3-7: Comparison of the solution calculated by equation 2.19, in red, and the exact analytical solution calculated by equations 2.7 and 2.8, in blue. The lower graph shows the difference between the analytic solution and the solution calculated by equation 2.19 in magenta. For this calculation $c = 100$.

c	Max Difference	Mean Difference
1	0.010069	0.0010888
10	0.010137	0.0011424
100	0.01014	.0011436

Table 3.2: Table showing how the maximum difference and mean difference between the solutions calculated by equation 2.19 and the analytical solution change as c is varied.

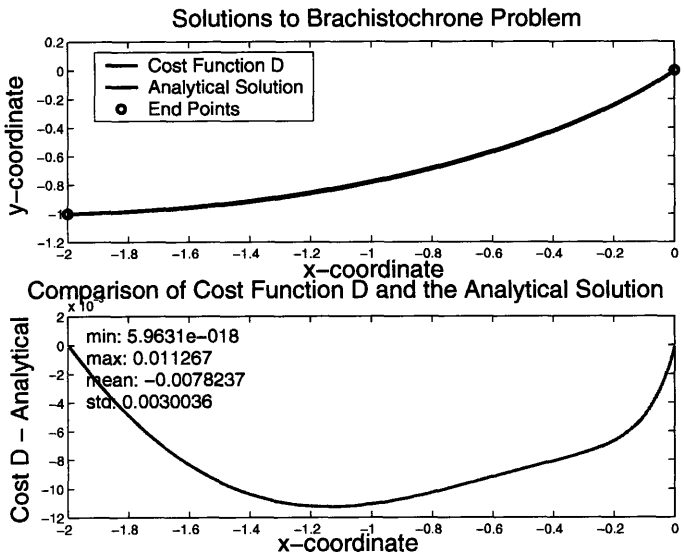

Figure 3-8: Comparison of the solution calculated by equations 2.23 and 2.24, in red, and the exact analytical solution calculated by equations 2.7 and 2.8, in blue. The lower graph shows the difference between the analytic solution and the solution calculated by equations 2.23 and 2.24 in magenta. For this calculation $c = 1$.

compared to the analytical solution in figures 3-8, 3-9 and 3-10, where the strength parameter of the cost function is 1, 10 and 100 respectively. Table 3.3 shows how the maximum and average differences vary with c. As c is decreased both the maximum and average differences decrease like they did for cost function C. The maximum differences from cost function D are less than those from cost function A, but the average differences from cost function D are greater than those from cost function A. However both the maximum and average differences from cost function C are less than those from cost function D. So the time parameterization works almost as well as the x coordinate parameterization.

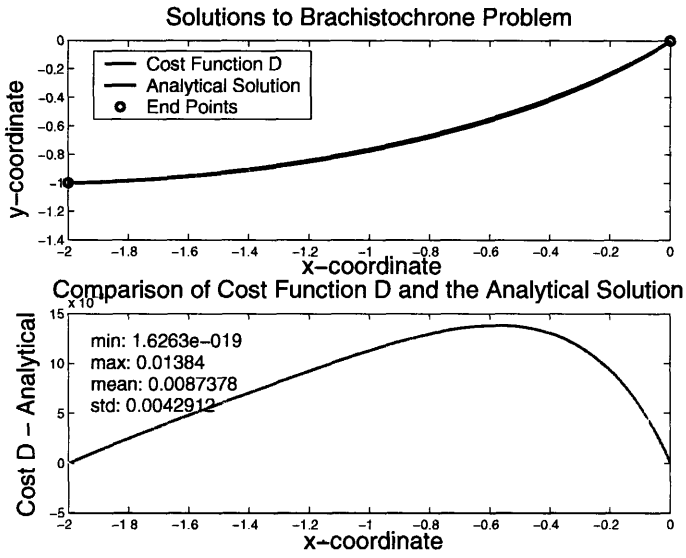

Figure 3-9: Comparison of the solution calculated by equations 2.23 and 2.24, in red, and the exact analytical solution calculated by equations 2.7 and 2.8, in blue. The lower graph shows the difference between the analytic solution and the solution calculated by equations 2.23 and 2.24 in magenta. For this calculation $c = 10$.

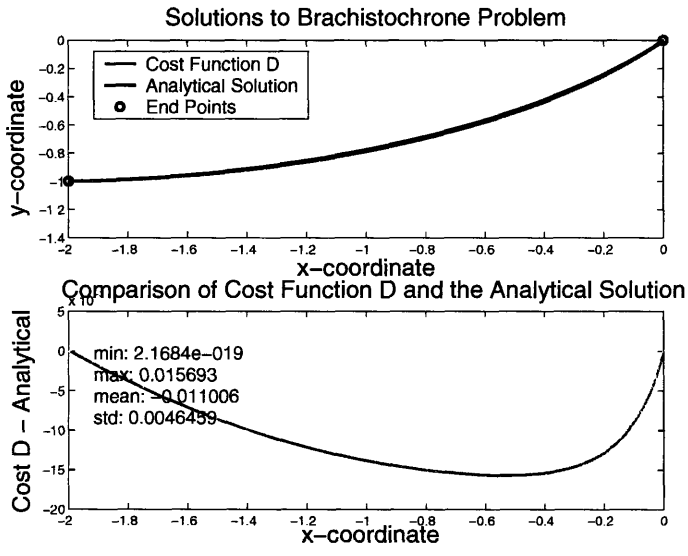

Figure 3-10: Comparison of the solutions calculated by equations 2.23 and 2.24, in red, and the exact analytical solution calculated by equations 2.7 and 2.8, in blue. The lower graph shows the difference between the analytic solution and the solution calculated by equations 2.23 and 2.24 in magenta. For this calculation $c = 100$.

c	Max Difference	Mean Difference
1	0.011267	-0.0078237
10	0.01384	0.0087378
100	0.015693	-0.011006

Table 3.3: Table showing how the maximum difference and mean difference between the solutions calculated by equations 2.23 and 2.24 and the analytical solution change as c is varied.

3.3 Examination of Solutions to Specific Examples

To examine the solutions of these differential equations, the BVP setup and slope finding programs are entered into the move-middle-pt program. move-middle-pt was run with an initial velocity of 1, the acceleration of gravity set to 1 and c set to 100. The start point was the origin and the finish coordinate was $(-2, -2)$. The middle point had a y-coordinate of -1, and had x-coordinates -1.9, -1.7, -1.5, -1.3, -1, -.7, -.5, -.3, -.1 depending on the path. The results for using cost function A are in figure 3-11, and the results for using cost function D are in figure 3-12. Cost functions B and C could not be used in this analysis because the differential equations are so complicated that Matlab had considerable trouble computing the paths.

Note that the middle point's x-coordinate is always between the x-coordinates of the start and finish points. If this choice was not made there would be a point where the slope would have to go to infinity. The slope goes to infinity where the path changes from moving left to moving right or vice versa. When that happens, the differential equation for the cost function A goes to infinity, and the path becomes uncomputable.

It is interesting to compare figure 3-11 with figure 3-12, because the paths are very similar but have distinct differences. The paths for cost function A tend to be much straighter than the paths for cost function D. This fact is very evident in the paths where the middle point has an x-coordinate of -1.9 and -.1. The observation that the paths for cost function A are straighter can be explained based on the fact that cost function D is proportional to the square of acceleration. As can be seen in figure 3-11, when the paths between points are straighter, there is a more punctuated

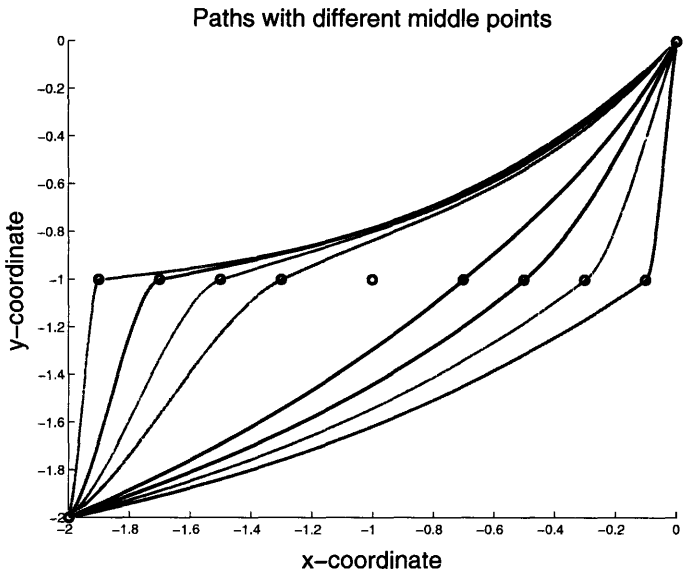

Figure 3-11: Computed using differential equation 2.10 in move-middle-pt. A set of optimized paths through three points where the middle point has been moved horizontally by .9, .7, .5, .3, 0, -.3, -.5, -.7, and -.9 from -1.

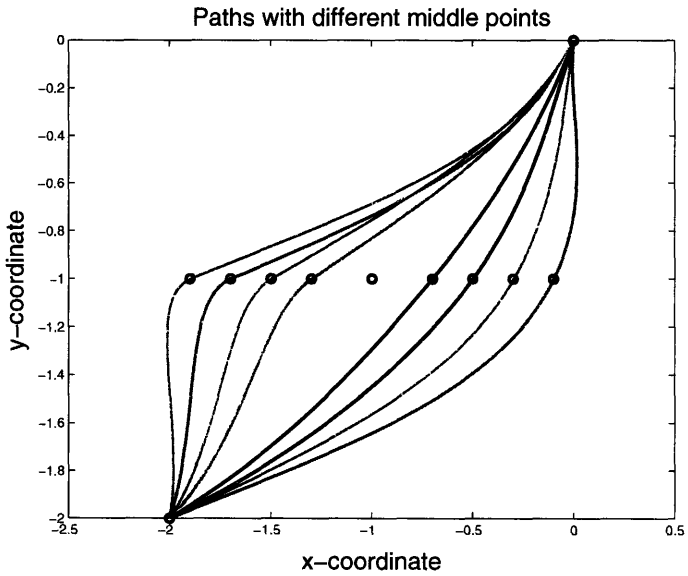

Figure 3-12: Computed using differential equations 2.23 and 2.24 in move-middle-pt. A set of optimized paths through three points where the middle point has been moved horizontally by the amounts .9, .7, .5, .3, 0, -.3, -.5, -.7, and -.9 from -1.

turn in the path at the middle point. The more punctuated the turn, the greater the force on skier traversing the path. Since cost function D attempts to minimize the acceleration, and thus force, it will not take a path that requires punctuated turns like those of cost function A and will tend to take a more slowly curving path.

The paths for cost function A appear very symmetric in comparison with those for cost function D. For cost function A's paths, the path through the middle point with x-coordinate -1.9 could be rotated 180 degrees and shift to look like a very similar path to the path that goes through the middle point with x-coordinate -.1. A similar relationship can be seen in the paths through the middle point with x-coordinate -1.7 and -.3, and to lesser extent the symmetry can be seen in paths that go through the middle point with x-coordinate -1.5 and -.5. However, this symmetry is not apparent at all in the paths for cost function D.

Another oddity is that the paths for cost function D cross each other whereas the paths for cost function A do not. For the paths of function D, it is most clear that the path through the middle point with x-coordinate -1.5 crosses two other paths, but the path through the middle point with x-coordinate -1.3 also crosses a path. It is questionable whether the paths cross because those really are the fastest paths, or if they cross due to inaccuracy in the numerical evaluation, or if they cross because of the addition of a cost function.

A particularly interesting path to examine is the path for cost function D that goes through the middle point with x-coordinate -.1. Moving from the starting point to the middle point, the path actually moves further away from the middle point at first by moving to the right. Then it gradually swings back left to go through the middle point. What is achieved by going out to the right a little bit is that the path goes straight down for a period of time, which gives the path more acceleration from gravity. In ski racing it is common to talk about putting your skis down the fall line, which means taking a path that goes in the direction of gravity longer. Here is a validation if that phenomena in the calculations.

When the path is parameterized with respect to time, it becomes possible to compute a path even if there is a place where the slope becomes infinite. This is

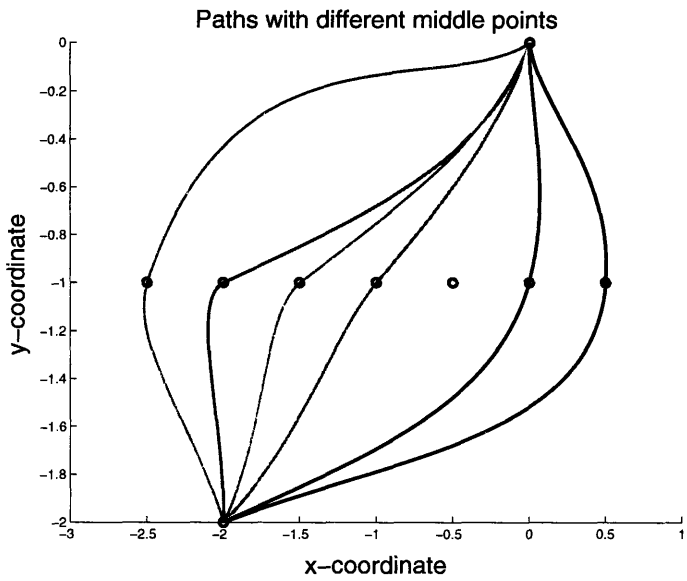

Figure 3-13: Computed using differential equations 2.23 and 2.24 in move-middle-pt. A set of optimized paths through three points where the middle point has been moved horizontally by 1.5, 1, .5, 0, -.5, -1 and -1.5 from -1.

because the slope is equal to $\dot{y}/\dot{x}$, so the singularity can be removed by multiplying by $\dot{x}$. Taking advantage of this fact, more examples have been calculated using cost function D. Figure 3-13 shows several paths where the middle point's x-coordinate was varied more, and allowed to go beyond the x-coordinates of the start and finish points. Figure 3-14 shows the optimized path through 6 points where the points alternate between having an x-coordinate of 0 and 1 and between each consecutive point there is the same difference in y-coordinate. A course that is set like this is called rhythmical in ski racing, and it is very common for there to be a rhythmical section in a race course. Two things to look for in these paths is where the paths are curved the most and where the apex of the turn is. The apex is where the path goes the furthest horizontally for the whole path, which is also where the path is going straight down. The three middle paths of figure 3-13 do not have apexes because the x-coordinates of the middle points lies between the x-coordinates of the start and finish points.

The paths through the middle points with x-coordinates -2.5 and -2 apex after the middle point and turn most sharply after the middle point. The paths through

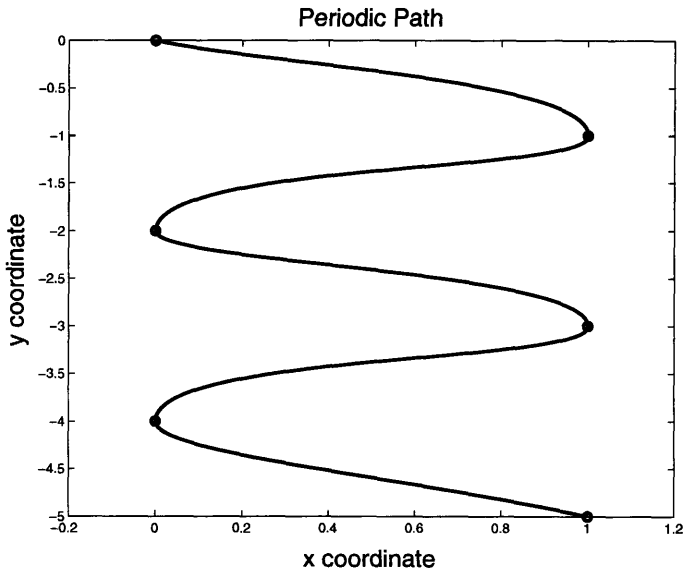

Figure 3-14: Computed using differential equations 2.23 and 2.24. An example of the optimal path through a periodic course.

the middle points with x-coordinates 0 and .5 apex before the middle point and turn most sharply after the middle point. The way this could be accounted for is, that the velocity above the middle point is less than the velocity below the middle point, which means that it will be more beneficial to move in the direction of gravity above the middle point than below the middle point. Hence it is better to turn more below the middle point, where it is not as beneficial to move in the direction of the gravitational field.

There are two distinct features of the path in figure 3-14: the apex for each turn is at the gate, and path above the gate is almost symmetric to the path below the gate. The path above the gate gets more symmetric with the path below the gate at the lower gates of the course. It has been known in ski racing for a long time that placing the apex of the turn at the gate is the fastest, so it is good that these paths reproduce that result. If a line is drawn through a gate horizontally, and then the page was folded on that line, the part of the path immediately below the gate would almost line up with the path immediately above the gate. There is no current ski racing tactic about this fact, but perhaps there should be. It is easy to visualize a

symmetric path, so it would be a relatively easy tactic to use.

Chapter 4

Future Directions of Research

4.1 Other Coordinate Systems

In Ashby's [1] and Vratanar's [7] papers, they both utilize a coordinate system where the velocity, v, and the angle of the slope of the path, ϕ, are the main coordinates of the system. In those coordinates centripetal acceleration takes on a particularly simple form

$$A_c = v^2 d\phi/ds = v d\phi/dt. \tag{4.1}$$

Using this coordinate system in the integrands of chapter two, where centripetal acceleration was the cost function, could result in simpler differential equations. This may even be solvable analytically as in the papers by Ashby and Vratanar.

Another reason that the v and ϕ coordinate system might be beneficial is that this system makes it easy to introduce friction and aerodynamic drag. Eventually the effects of these non-conservative forces would need to be included to increase the accuracy of the solution. One possible down side is that it is more difficult to extend the v and ϕ coordinate system to three dimensions than a Cartesian system. A three dimensional system will be necessary to study the effects of terrain on the optimal path, which is a very important subject in ski racing.

To make the results calculated useful to the ski racing community, they must be extended to include many more factors such as the constraint of the path to the

surface of a mountain, friction, and aerodynamic drag. Ashby's [1] and Vratanar's [7] papers offer great insight into how these factors can be added.

4.2 Better Analysis

To make an impact on the ski racing community, these calculations need to come up with a concise list of characteristics of the optimal path that ski racers can identify and visualize easily. If the results from analyzing the example optimal paths are too complicated, they will not be useful to a racer because they will not be able to apply them. It is important that the analysis of optimal paths gets better, so that useful characteristics can be identified. That is the only way that these calculations will find use within the ski racing community.

To start, it might be interesting to study the effects of adding more points. One way to test this would be to start with two points and calculate the optimal path that goes between them. Then add a new point somewhere between the original two, and find the new optimal path through all three points. Adding a new point and calculating the new optimal path through all the points could be repeated any number of times, and would show how the characteristics of the optimal path change with the addition of more and more points..

Keeping track of the curvature of the path would probably be the best thing to study. Perhaps the curvature could be plotted with the path by making the color of the path dependent upon the amount of curvature in the path. This is important because it tells the skier where to turn the skis the most, which is called the heart of the turn. It would also be good to mark where the curvature goes to zero, which is where the skier is switching from turning in one direction to the other. The part of the turn where the skier is switching the direction they are turning is called the transition, and is crucial to ski racing because it determines where the skier shifts their balance from one leg to the other. Marking the apex of the turns would be helpful, because knowing where the apex is helps the skier visualize the path through a course better. Though it appears from the analysis of figure 3-14 that the apex

will be at the gate, there may be cases such as non-rhythmical courses and terrain changes where the apex will not be at the gate.

One thing is for sure, and that is that as the number of effects included in the model increases, there will be more and more factors contributing to the determination of the optimal path. That will make it ever more difficult to understand what effects matter the most, and it will be more important to take care in analyzing the results to find the most important characteristics of the optimal path.

Appendix A

BVP Setup Programs

This appendix gives more detail on the BVP setup programs and provides sample code of one such program. The sample code can be seen below.

The power of the BVP setup functions rest on Matlab's boundary value problem solver, bvp4c. bvp4c takes three function handles in as arguments. One is a handle to a function that will calculate the derivatives of the path, yprime. Another is a function that is zero when the boundary conditions are met, bcfunc. The third function is a specially created guess, solinit. Solinit is created by the bvpinit function, which turns a guess at the solution to the differential equation, yguess, and a set of x-coordinates, xmesh, into a structured data object from which bvp4c starts calculating its solution. The initial guess in the BVP setup programs are straight line paths from point to point.

function SOL = solve−brach−slope(xi,yi,xf,yf,v,g,c,slopei,slopef)

%% *Calculates and returns the optimal path*

%% *between two points in a uniform*

%% *gravitational field using numerical*

%% *methods. Solves a fourth order differential*

%% *equation, so that the slope can be specified*

%% *at the initial and final points.*

%% *The cost function used to invoke the slope*

%% *constraint is:*

```
%%       (c^2/2)* \int (y")^2 *dx                                    10
%%
%% solve-brach-slope(xi,yi,xf,yf,v,g,c,slopei,slopef)
%% xi is the x-coordinate of the initial point.
%% yi is the y-coordinate of the initial point.
%% xf is the x-coordinate of the final point.
%% yf is the y-coordinate of the final point.
%% v is the initial velocity of the particle.
%% g is the strength of the gravitational field.
%% c is the strength of the cost function.
%% slopei is the desired slope of the solution at                     20
%%      the initial point.
%% slopef is the desired slope of the solution at
%%      the final point.
%% The field points in the negative y direction.
% M provides short hand notation.
M = 2*g*yi+v^2;
% Create a mesh of x-coordinates at which the guessed
%     solution can be evaluated.
N = 5;                  % Number of points in the guess
xmesh = linspace(xi,xf,N);                                           30
% Initialize the guess function, differential equation,
%     boundary condition functions.
yguess(0,xi,xf,yi,yf);
yprime(0,0,M,g,c);
bcfunc(0,0,yi,yf,slopei,slopef);
% Create initial solution.
solinit = bvpinit(xmesh,@yguess);
% Use the boundary value problem solver.
SOL = bvp4c(@yprime, @bcfunc, solinit);
```

```
% Below are sub-functions used to aid the above code.
% Guess at the solution: linear
function yguess = yguess(x, xi, xf, yi, yf)
persistent xa xb ya yb;
initializing = nargin > 1;
if initializing
    % Initializing the function:
    if (xi < xf)
        % Final point is to the right of the initial point
        xa = xi;
        ya = yi;
        xb = xf;
        yb = yf;
    else
        % Final point is to the left of the initial point
        xa = xf;
        ya = yf;
        xb = xi;
        yb = yi;
    end
else
    yguess(1) = (yb - ya)*(x - xa)/(xb - xa) + ya;
    yguess(2) = (yb - ya)/(xb - xa);
    yguess(3) = 0;
    yguess(4) = 0;
end
% Boundary Condition function:
function bcfunc = bcfunc(yinit, yfinal, yi_in, yf_in, slopei_in, slopef_in)
persistent yi yf slopei slopef;
initializing = nargin > 2;
```

```
if initializing                                                    70
    % Initializing the function:
    yi = yi_in;
    yf = yf_in;
    slopei = slopei_in;
    slopef = slopef_in;
else
    bcfunc = zeros(4,1);
    bcfunc(1) = yinit(1) - yi;      % Solution must start at yi.
    bcfunc(2) = yfinal(1) - yf;     % Solution must end at yf.
    bcfunc(3) = yinit(2) - slopei;  % Solution must start with slope = slopei.  80
    bcfunc(4) = yfinal(2) - slopef; % Solution must end with slope = slopef.
end
% Function for calculating the derivative
function yprime = yprime(x, y, M_in, g_in, c_in)
persistent M g c
initializing = nargin > 2;
if initializing
    % Initializing the function:
    M = M_in;
    g = g_in;                                                      90
    c = c_in;
else
    x = y(1);
    xprime = y(2);
    xdprime = y(3);
    xtprime = y(4);
    yprime = zeros(4,1);
    yprime(1) = xprime;
    yprime(2) = xdprime;
```

```
    yprime(3) = xtprime;                                                    100
    yprime(4) = ((M - 2*g*x)*xdprime - g*(1 + xprime^2)/...
        (c^2*((M - 2*g*x)*(1 + xprime))^(3/2));
end
```

Appendix B

Slope Finding Programs

This appendix gives more detail on the slope finding programs and provides sample code of one such program. The sample code can be seen below.

To find the optimum slopes, Matlab's fminsearch function is employed. fminsearch takes as arguments a function handle, an initial value and options. Starting at the initial value, fminsearch systematically finds a local minimum of the function whose handle it was given. In this case the function handle is the function deltaT. When deltaT is given a set of slopes, it will calculate a path through all the points with the given slopes and then compute the time it takes to traverse the path. Since deltaT outputs the time it takes to traverse the path, fminsearch will find the set of slopes that minimizes the time. deltT uses a numerical integrator to compute the time for traversing the path.

Both fminsearch and the numerical integrator, quad, have optional tolerance settings. Playing with these tolerances effects how quickly and how well the program finds the optimal slopes. Some differential equations are easier to compute than others. For example, the differential equation for cost function A is relatively simple, so higher tolerances can be demanded of quad and fminsearch without causing the solution to have a singularity.

In matlab's code a singularity can occur even when the differential equation is far from a place where the derivative goes to infinity. When Matlab is solving a boundary value problem, it creates a mesh of points over which it calculates the solution. The

higher the tolerances on fminsearch and quad, the finer the mesh to be computed, and the finer the mesh the more computationally intense the calculation. To curb computational complexity, Matlab calculates the amount the solution is changing over the entire mesh. If the sum is too large, then Matlab aborts the computation. A finer mesh will cause there to be more points over which the sum is taken, and can cause the sum to be too large.

The value of c also plays a crucial role in the occurrence of singularities. c^2 always shows up in the denominator somewhere in the differential equation. Hence the rate of change tends to be inversely proportional to c^2, which can easily cause problems when c gets below 1. It takes some trial and error to find a good balance between c and the tolerances of fminsearch and quad, but usually a tolerance of .1 will enable calculations with c at or just slightly below 1.

```
function SOL = solv_thesis_ns(brach_cost,points,v,g,c)
%% Calculates and returns the optimal path
%% through multiple points in a uniform
%% gravitational field using numerical
%% methods. To make the solution have a continuous
%% slope at each of the boundary points, a fourth
%% order differential equation must be solved so
%% the slope at the boundary points can be specified.
%%
%% solv-thesis-ns(brach_cost,points,v,g,c)                    10
%% brach_cost is a funciton handle to a function that
%%     solves the Brachistochrone problem with a cost
%%     function such that the slopes of the end points
%%     can be specified.
%% points is a matrix where each column is the position of
%%     one of the points the path must pass through. (The
%%     first row is all of the x-coordinates, and the
%%     second row is all of the y-coordinates.)
```

```
%% v is the initial velocity of the particle.
%% g is the strength of the gravitational field.
%% c is the strength of the cost function.
%% The field points in the negative y direction.
% Get the number of points the path must go through.
n = size(points,2);
% Make a guess at the slopes of the answer at each of
%      the boundary points.
m = zeros(n-1,1);
% Figure out straight line slope between end points.
for i = 1:1:n - 1
    if (points(1,i) < points(1,i+1))
        % Point i is to the right of point i + 1:
        m(i) = (points(2,i+1) - points(2,i))/(points(1,i+1) - points(1.i));
    else
        % Point is to the left of point i + 1:
        m(i) = (points(2,i) - points(2,i+1))/(points(1,i) - points(1,i+1));
    end
end
slope_init = zeros(n,1);
slope_init(1) = m(1);
for j = 2:1:n - 2
    % Want to get a slope half way between that of the straight line
    %      path above the boundary point and that of the path below.
    slope_init(j) = tan((atan(m(j-1)) + atan(m(j)))/2);
end
slope_init(n) = m(n-1);
% Initialize the time calculation function
deltaT(0,brach_cost,points,v,g,c)
% Set the tolerances and find the optimum slopes.
```

```
options = optimset('TolX',1e-1,'TolFun',1e-1);
slopes = fminsearch(@deltaT,slope_init,options);
% Calculate the solution.
for q = 1:1:n-1
    xi = points(1,q);
    yi = points(2,q);
    xf = points(1,q+1);
    yf = points(2,q+1);
    SOL(q) = feval(brach_cost,xi,yi,xf,yf,v,g,c,slopes(q),slopes(q+1));
end
% Below are sub-functions used to aid the above code.
% Creates a function that given the slopes at each of the boundary
%     points calculates the time for the path that solves the
%     differential equation with the given initial and final slopes.
function deltaT = deltaT(slope,brach_cost_in,points_in,v_in,g_in,c_in)
persistent brach_cost points v g c n
initializing = nargin > 1;
if initializing
    % Initializing the function:
    brach_cost = brach_cost_in;
    points = points_in;
    g = g_in;
    c = c_in;
    n = size(points,2);
    v = zeros(n-1,1);
    for w = 1:1:n-1
        % Calculate the velocity at each of the boundary points.
        v(w) = sqrt(2*g*(points(2,1) - points(2,w)) + v_in^2);
    end
else
```

```
    deltaT_sub = zeros(1,n-1);
    for p = 1:1:n-1                                                    80
        xi = points(1,p);
        yi = points(2,p);
        xf = points(1,p+1);
        yf = points(2,p+1);
        % Calculate the optimal path from point p to point p+1 with
        %     the slope(p) at point p and slope(p+1) at point p+1.
        SOL = feval(brach_cost,xi,yi,xf,yf,v(p),g,c,slope(p),slope(p+1));
        deltaT_sub(p) = time(SOL,xi,yi,xf,v(p),g);
    end
    % Add up the time to traverse each piece of the path to get the      90
    %     total time for traversing the whole path.
    deltaT = sum(deltaT_sub);
end
% Calculates the time it takes to traverse a given path.
function time = time(y_SOL,xi,yi,xf,v,g)
time_integrand(0,y_SOL,yi,v,g);     % Initialize the integrand function
if (xi < xf)
    % integrat form xi to xf
    xa = xi;
    xb = xf;                                                           100
else
    % integrate from xf to xi
    xa = xf;
    xb = xi;
end
time = quad(@time_integrand, xa, xb, 1.e-1);
% Creates the integrand for calculating the time of a path.
function time_integrand = time_integrand(x,y_SOL_in,yi_in,v_in,g_in)
```

```
persistent yi v g y_SOL
initializing = nargin > 1;                                          110
if initializing
    % Initializing the function:
    yi = yi_in;
    v = v_in;
    g = g_in;
    y_SOL = y_SOL_in;
else
    y = deval(y_SOL,x);
    time_integrand = sqrt((1 + y(2,:).^2)./(2*g*(yi - y(1,:)) + v^2));
end                                                                 120
```

Appendix C

Analysis Programs

This appendix gives more detail on the analysis programs whose code is provided below.

The first analysis program, brach-compare, allows the user to input two functions that will calculate optimal paths between two points. The two paths are then compared by taking the difference between them, and statistics such as the minimum, maximum, mean and standard deviation of the difference are calculated. It could be useful to take the absolute value of the differences and run the same statistical analysis on that set of data. Much of the program is for creating a nice graph with a legend, title, coordinate labels and so forth. All the code after line 59 does is dedicated to esthetics. The first ten lines are there to handle the case where one of the functions calculates the path as a function of time. In that case the domain of the function is passed into brach-compare in a structured object where one component is the function handle and the other is maximum of the time domain of the function. The minimum of the time domain is always 0.

brach-compare was used with the analytical solution to the Brachistochrone problem to evaluate the validity of the four differential equations tested in chapter three.

function brach−compare(func1,func2,xi,yi,xf,yf,v,g)

%% Given function handles to two functions

%% that solve the Brachistochrone problem,

%% will evaluate both functions with the given

```
%% inputs, graph each of their solutions and
%% give various statistics comparing the
%% solutions.
%%
%% brach-compare(func1,func2,xi,yi,xf,yf,v,g)
%% func1 is a function handle to the first function.
%%     func1 can return a structured solution.
%% func2 is a function handle to the second function.
%% xi is the x-coordinate of the initial point.
%% yi is the y-coordinate of the initial point.
%% xf is the x-coordinate of the final point.
%% yf is the y-coordinate of the final point.
%% v is the initial velocity of the particle.
%% g is the strength of the gravitational field.
%% The field points in the negative y direction.
% N is the number of points at which the two solutions
% will be compared at.
N = 100;
% Calculate the solutions for each function.
out1 = feval(func1,xi,yi,xf,yf,v,g);
path2 = feval(func2,xi,yi,xf,yf,v,g);
% Use the solutions to calculate the paths.
if isfield(out1,'function_handle')
    % func1 returns a structured solution.
    path1 = getfield(out1,'function_handle');
    tf = getfield(out1,'finish_time');
    % Evaluate the solutions at various points:
    t = linspace(0,tf,N);
    z = feval(path1,t);
    x = z(1,:);
```

```
    y1 = z(2,:);
    y2 = feval(path2,x);
else
    % func1 does not return a structured solution.
    path1 = out1;
    % Evaluate the solutions at various points:
    x = linspace(xi,xf,N);
    y1 = feval(path1, x);
    y2 = feval(path2, x);
end
% Take the difference between the paths.
diff = y1(1,:) - y2(1,:);
% Calculate statistics of the difference between solutions.
min = min(abs(diff));
max = max(abs(diff));
mean = mean(diff);
std = std(diff);
% Plot each solution and the difference between the solutions.
subplot(2,1,1)
% Solution found by func1 is ploted in red.
plot(x,y1(1,:),'r');
hold;
% Solution found by func2 is ploted in blue.
plot(x,y2(1,:),'b');
% Plot the end points of the problem.
plot([xi xf], [yi yf], 'ko');
% Label the graph.
title('Solutions to Brachistochrone Problem')
xlabel('x-coordinate')
ylabel('y-coordinate')
```

```
leg = legend(func2str(path1), func2str(path2), 'End Points of Problem', 0);
% Remove the Latex interpretation in the legend.
leg_chld = get(leg, 'Children');
set(leg_chld(end),'Interpreter','none');
hold;
subplot(2,1,2)                                                    70
% Plot the difference in magenta on a seperate graph.
plot(x,diff,'m');
% Label the graph.
title(['Comparison of ' func2str(path1) ' and ' func2str(path2)])
xlabel('x-coordinate')
ylabel([func2str(path1) ' - ' func2str(path2)])
% Remove the Latex interpretation in the y-label and title.
ax = gca;
title = get(ax, 'Title');
ylabel = get(ax, 'YLabel');                                       80
set(title, 'Interpreter', 'none');
set(ylabel, 'Interpreter', 'none');
% Print the statistics of the difference between paths.
v = axis;        % Get the scaling of the current plot
x_text = v(1);   % Text will be on left side of the graph
y_text = v(4);   % Text will be at the top of the graph
% Define the text string.
str(1) = {['min: ' num2str(min)]};
str(2) = {['max: ' num2str(max)]};
str(3) = {['mean: ' num2str(mean)]};                              90
str(4) = {['std: ' num2str(std)]};
text(x_text, y_text, str,...
    'VerticalAlignment', 'top',...
    'HorizontalAlignment', 'left')
```

The other analysis program examines solutions that pass through three points. The middle point of the three points, is moved around by this program to explore how a change in the middle point changes the optimal path. This program makes a call to a slope finding function, solve-thesis-ns, which does all the work of calculating the paths. All this program has to do is automate the use of solve-thesis-ns so that many example paths can be calculated and compared quickly and efficiently. Similar to brach-compare, a lot of the code is focused on creating the graph and legend.

function move−middle−pt(brach_cost,points,v,g,c,steps)

%% Calculates the optimal path through three
%% points in a uniform gravitational field
%% using numerical methods. This requires
%% solving a fourth order differential equation
%% so the slope at the middle point is the same
%% for the path below the middle point and for
%% the path above the middle point. A set of
%% solutions are calculated for different
%% locations of the middle point. The paths
%% are plotted.
%%
%% move-middle-pt(brach_cost,points,v,g,c,steps)
%% brach_cost is a funciton handle to a function that
%% solves the Brachistochrone problem with a cost
%% function such that the slopes of the end points
%% can be specified.
%% points is a matrix where each column is the position of
%% one of the points the path must pass through. (The
%% first row is all of the x-coordinates, and the
%% second row is all of the y-coordinates.)
%% v is the initial velocity of the particle.
%% g is the strength of the gravitational field.

```
%% c is the strength of the cost function.
%%
%% steps is a vector of displacements for the x-coordinate
%%      of the middle point.
%% The field points in the negative y direction.
% Figure out how many paths need to be calculated.
n = size(steps,2);                                          30
% Set the number of points used to graph the solutions.
M = 100;
xi = points(1,1);
yi = points(2,1);
xf = points(1,3);
yf = points(2,3);
% Initialize the plot.
subplot(1,1,1);
hold;
% y holds the calculated paths.                             40
y = zeros(M,4,n,2);
% Used to generate the legend and color the plots.
comm = 'legend(';
color = 'bgrcmyk';
x_mid_str = [];
% Calculate and graph the optimal path for different
%      locations of the middle point.
for i = 1:1:n
    % Calculate the midpoint.
    points_m = points + [0 steps(i) 0; 0 0 0];              50
    xm = points(1,2) + steps(i);
    % Calculate the solution.
    SOL = solv-thesis-ns(brach_cost,points_m,v,g,c);
```

```
    if (xi < xf)
        % Final point is to the right of the initial point.
        x1 = linspace(xi, xm, M);
        x2 = linspace(xm, xf, M);
    else
        % Final point is to the left of the initial point.
        x1 = linspace(xf, xm, M);                                    60
        x2 = linspace(xm, xi, M);
    end
    % Calculate the path.
    y(:,:,1,1) = deval(SOL(1),x1)';
    y(:,:,1,2) = deval(SOL(2),x2)';
    % Graph the path.
    plot(x1,y(:,1,1,1),color(mod(i,7)+1));
    plot(x2,y(:,1,1,2),color(mod(i,7)+1));
    % Graph the midpoint.
    plot(points_m(1,2), points_m(2,2), 'ko');                        70
    % Create text for the legend.
    x_mid_str = strvcat(x_mid_str, ['x_mid=' num2str(points_m(1,2))]);
    comm = [comm 'x_mid_str(' num2str(i) ',:), '];
end
% Finalize the graph.
plot([xi xf], [yi yf], 'ko');
% Label the graph.
title('Paths with different middle points');
xlabel('x-coordinate');
ylabel('y-coordinate');                                              80
% Create the legend.
end_pts = 'End Points of the Problem';
comm = [comm 'end_pts, 0)'];
```

```
fig = eval(comm);
% Remove the Latex interpretation in the legend.
leg_chld = get(leg, 'Children');
set(leg_chld(end),'Interpreter','none');
```

Bibliography

[1] N. Ashby, W.E. Brittin, W.F. Love, and W. Wyss. Brachistochrone with coulomb friction. *American Journal of Physics*, 43(10):902–906, oct 1975.

[2] V. Covic and M. Veskovic. Extension of the bernoulli's case of brachistochronic motion to the multibody system having the form of a kinematic chain with external constraints. *European Journal of Mechanics A-Solids*, 21(2):347–354, Mar-Apr 2002.

[3] Jr. Frederick W. Byron and Robert W. Fuller. *Mathematics of Classical and Quantum Physics*, chapter 2, pages 44–51. Dover, 1970.

[4] Herbert Goldstein, Charles Poole, and John Safko. *Classical Mechanics*, chapter 2.4, pages 45 50. Pearson Education, third edition, 2002.

[5] Andrew Pressley. *Elementary Differential Geometry*. Springer, 2001.

[6] George Twardokens. Longer line = shorter time. *The Professional Skier*, 1996. Fall Issue.

[7] B. Vratanar and M. Saje. On the analytical solution of the brachistochrone problem ina non-conservative field. *International Journal of Non-Linear Mechanics*, 33(3):489 -505, 1998.

[8] Eric W. Weisstein. Brachistochrone problem. World Wide Web, October 2004. From *M*ath World–A Wolfram Web Resource. http://mathworld.wolfram.com/BrachistochroneProblem.html.

[9] D. Zekovic. The brachistochrone motion of a mechanical system with nonholonomic, nonlinear and nonstationary constraints. *PMM Journal of Applied Mathematics and Mechanics*, 54(6):765–768, 1990.