Summer 2015

Solutions of Inequality Constrained Spline Optimization Problems with the Active Set Method

Joshua A. Holloway

SOLUTIONS OF INEQUALITY CONSTRAINED SPLINE OPTIMIZATION PROBLEMS WITH THE ACTIVE SET METHOD

by

JOSHUA HOLLOWAY

(Under the Direction of Scott Kersey)

ABSTRACT

We solve the problem of finding a near-interpolant curve, subject to constraints, which minimizes the bending energy of the curve. Using B-splines as our tools, we give a brief overview of spline properties and develop several different cases of inequality constrained optimization problems of this type. In particular, we develop the active set method and use it to solve these problems, emphasizing the fact that this algorithm will converge to a solution in finite iterations. Our solution will solve an open problem regarding near-interpolant spline curves. Furthermore, we supplement this with an iterative technique for better choosing data sites so as to further minimize the bending energy of the spline curve, offering an easy solution to the problem of free data sites.

Key Words: Spline Curve, B-Spline, Constrained Optimization, Quadratic Programming, Active Set Method

2009 *Mathematics Subject Classification*: 41A15, 65D07, 65K10, 90C20

SOLUTIONS OF INEQUALITY CONSTRAINED SPLINE OPTIMIZATION PROBLEMS WITH THE ACTIVE SET METHOD

by

JOSHUA HOLLOWAY

B.S., Mercer University, 2013

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment

of the Requirement for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

SOLUTIONS OF INEQUALITY CONSTRAINED SPLINE OPTIMIZATION PROBLEMS WITH THE ACTIVE SET METHOD

by

JOSHUA HOLLOWAY

Major Professor:	Scott Kersey
Committee:	Xiezhang Li
	Hua Wang

Electronic Version Approved:

July 2015

DEDICATION

This thesis is dedicated to my wonderful partner Drake Stancill, who keeps me grounded, happy, and who inspires me to work hard and dream big.

ACKNOWLEDGMENTS

I wish to acknowledge first and foremost my committee members who have pushed me to be a better student, teacher, and mathematician. In particular I would like to acknolwedge Dr. Scott Kersey, who has perhaps taught me the most, and has inspired me to think differently and helped me to solve more complicated problems. I would like to thank him for spending so many hours with me, and for being an excellent mentor. I would also like to acknolwedge the many friends I have made during my studies. With the aid of people such as Ryan Morley, I was able to accomplish something much greater than alone. Lastly I would like to acknowledge my good friend Kamila Gabka, who has struggled with me for many years on this journey, and who has inspired me to become a better person.

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A parametric curve $f : [a, b] \to \mathbb{R}^n$ is called near-interpolant if, given data $z_{i,j}$ and tolerances $\epsilon_{i,j}$ at data sites t_i if the curve satisfies $|f^{(j-1)} - z_{i,j}| \leq \epsilon_{i,j}$ for $i = 1, ..., n$ and for $j = 1, ..., m$. Furthermore, this is called the best near-interpolant curve if the curve minimizes the bending energy, $\int_a^b |f^{(m)}|^2$. This problem is discussed in [6], and smoothing problems, with obstacles, of this type are solved in [8] using a method of adding and removing knots, from another unpublished source. Solving spline problems with obstacles, or constraints, is also discussed in [2], [1], and [10]. However, Leetma shows that the method of adding and removing knots can create a cycle, and that it is still an open problem to find an algorithm to solve this problem in finite steps. In this thesis, we solve this open problem. We focus on the univariate case, and apply methods of quadratic programming to solve a similar problem. In particular, we develop the active set method, and apply this method on functions with various types of constraints. We then generalize to the parametric case, and highlight convergence of the method in finite iterations in both cases. Also, we develop an iterative technique for solving the problem of free data sites. This is a simple technique that can be used to solve very complicated, non-linear problems. We begin by giving a brief survey of splines and their properties.

Given a data set $\{(x_i, y_i), i = 1, ..., n\}$, several techniques exist to find a curve that matches each data point. This is the notion of interpolation. Polynomial interpolation is well studied in applicable in a variety of situations. However, there are several disadvantages to polynomial interpolation. Give n points, we interpolate with a polynomial of degree $n - 1$. Not only could this possibly yield to numerical error, it is also difficult to explain phenomena that are modeled with such high degree polynomials. Further, polynomial interpolation exhibits the Runge error phenomenon.

While there exist methods to reduce this error, it is a natural question to wonder how else we might fit a curve through a given set of points. one answer to this question is to use the notion of spline curves. The main points of the thesis are as follows:

- Properties of B-splines, and how we can use these properties and spline curves as a tool in answering optimization questions.
- Basic optimization theory, including the Karush-Kuhn-Tucker conditions, and developing in detail the Active-Set method for quadratic programming. In particular, we highlight its application to spline optimization questions.
- Solutions of inequality constrained spline optimization problems with various different types of constraints. Moreover, we develop a generalization of this problem to parametric curves.
- A brief discussion on an iterative technique to choose optimal data sites for a better solution to inequality constrained spline optimization questions.

CHAPTER 2

SPLINES

2.1 B-Spline Basics

A spline can be thought of as a piecewise polynomial. Here, splines are thought of as linear combinations of B-splines, or *basis* splines. B-splines are defined by using a non-decreasing sequence of real numbers, $T = \{t_0, t_1, ...\}$, called a *knot sequence*, which provides information about continuity of the curve and its derivatives. We say a knot has multiplicity m if it appears m times in the knot sequence. It is important to note that the knot sequence *entirely* determines the B-Spline.

Definition 2.1. *A **B-spline** is defined recursively as the following:*

$$B_i^0(t) := \begin{cases} 1, & t \in [t_i, t_{i+1}) \\ 0, & \text{otherwise.} \end{cases}$$

$$B_i^d(t) := \frac{t - t_i}{t_{i+d} - t_i} B_i^{d-1}(t) + \frac{t_{i+d+1} - t}{t_{i+d+1} - t_{i+1}} B_{i+1}^{d-1}(t)$$

provided $d \geq 1, \frac{0}{0} := 0$.

Definition 2.2. *A **spline space**, $\mathbb{S}$, is a pair (d, T) where $d \in \mathbb{N}$ is a particular degree and T is a fixed knot sequence.*

Definition 2.3. *A **spline function**, $s(t)$, is a linear combination of B-splines, $\sum_{i=0}^{n-d-1} c_i B_i^d(t)$, where n is the length of the knot sequence.*

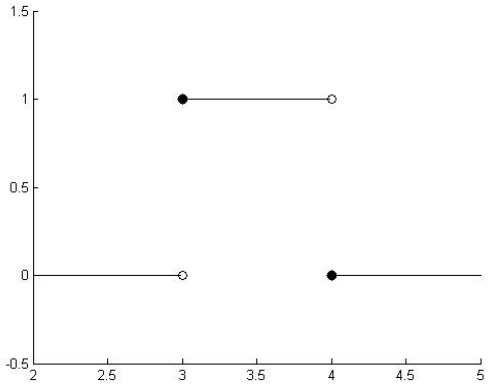


Figure 2.1: $B_0^2(t)$ on the interval $[2, 5]$ with knot sequence $T = [3, 3, 5, 5]$

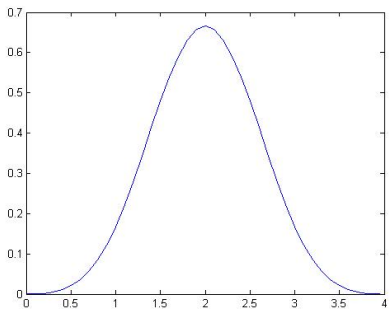
The knot sequence is very important in regards to a B-spline. A given knot sequence will determine the shape of a B-spline, where it is continuous at, and at where it is differentiable. Depending on the type of knot sequence, we get different types of spline curves:

- Uniform: $t_{i+1} - t_i$ is constant.
- Open: The multiplicity of the first knot and last knot is $d + 1$. This will interpolate end points and derivatives.
- Closed and Periodic: When $t_{n+i} = t_{n+i-1} + (t_i - t_{i+1})$.

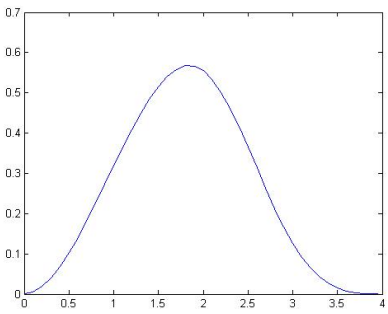
For this paper, we typically use open, interpolatory knots unless otherwise specified.

B-Splines typically have low degree and can incorporate smoothness. Furthermore, B-splines have numerous properties that are useful in solving interpolation problems.

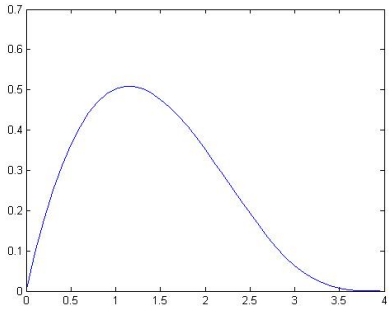
- A B-spline is a polynomial between knots, and can only possibly lose continuity at knots.



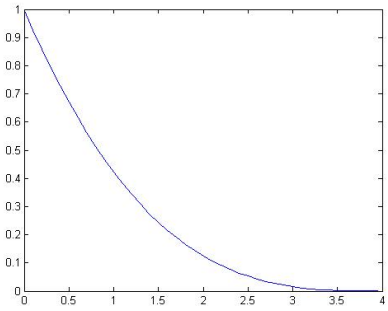
(a) $B_1^3(t)$ with $T = 0, 1, 2, 3, 4$



(b) $B_1^3(t)$ with $T = 0, 0, 2, 3, 4$



(c) $B_1^3(t)$ with $T = 0, 0, 0, 3, 4$



(d) $B_1^3(t)$ with $T = 0, 0, 0, 0, 4$

Figure 2.2: Various different B-Splines

- $B_i^d(t) = 0$ for t outside of $[t_i, t_{i+d+1}]$
- $B_i^d(t) > 0$ for $t \in [t_i, t_{i+d+1}]$

Since a B-spline can be thought of as a polynomial, it is useful to consider taking its derivative. This formula will prove useful in formulating inequality constrained optimization problems.

Theorem 2.4. *The derivative of a B-Spline is given by:*

$$B_i^d(t)' = \frac{d}{t_{i+d} - t_i} B_i^{d-1}(t) - \frac{d}{t_{i+d+1} - t_{i+1}} B_{i+1}^{d-1}(t)$$

Proof. We will proceed by mathematical induction on d. First, it follows from definition 2.1 that $B_i^0(t)' = 0$ everywhere except at t_i, t_{i+1}, where the derivative is undefined.

Furthermore, consider the case where $d = 1$. Again, by definition 2.1, the first degree B-spline is given by

$$B_i^1(t) = \frac{t - t_i}{t_{i+1} - t_i} B_i^0(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} B_{i+1}^0(t)$$

Using product rule, we arrive at

$$B_i^1(t)' = \frac{1}{t_{i+1} - t_i} B_i^0(t) - \frac{1}{t_{i+2} - t_{i+1}} B_{i+1}^0(t)$$

This agrees with the derivative formula. Now, assume that the formula holds up to degree d.

Observe, by definition, that

$$B_i^{d+1}(t) = \frac{t - t_i}{t_{i+d+1} - t_i} B_i^d(t) + \frac{t_{i+d+2} - t}{t_{i+d+2} - t_{i+1}} B_{i+1}^d(t)$$

We can calculate the derivative again using the product rule.

$$\begin{aligned} B_i^{d+1}(t)' = &\frac{t - t_i}{t_{i+d+1} - t_i} B_i^d(t)' + \frac{1}{t_{i+d+1} - t_i} B_i^d(t) \\ &+ \frac{t_{i+d+2} - t}{t_{i+d+2} - t_{i+1}} B_{i+1}^d(t)' - \frac{1}{t_{i+d+2} - t_{i+1}} B_{i+1}^d(t) \end{aligned} \tag{2.1}$$

By the induction hypothesis, we have that

$$B_i^d(t)' = \frac{d}{t_{i+d} - t_i} B_i^{d-1}(t) - \frac{d}{t_{i+d+1} - t_{i+1}} B_{i+1}^{d-1}(t)$$

$$B_{i+1}^d(t)' = \frac{d}{t_{i+d+1} - t_{i+1}} B_{i+1}^{d-1}(t) - \frac{d}{t_{i+d+2} - t_{i+2}} B_{i+2}^{d-1}(t)$$

Substitute these formulas into 2.1 gives:

$$\begin{aligned} B_i^{d+1}(t)' = &\frac{t - t_i}{t_{i+d+1} - t_i} [\frac{d}{t_{i+d} - t_i} B_i^{d-1}(t) - \frac{d}{t_{i+d+1} - t_{i+1}} B_{i+1}^{d-1}(t)] + \frac{1}{t_{i+d+1} - t_i} B_i^d(t) \\ &+ \frac{t_{i+d+2} - t}{t_{i+d+2} - t_{i+1}} [\frac{d}{t_{i+d+1} - t_{i+1}} B_{i+1}^{d-1}(t) - \frac{d}{t_{i+d+2} - t_{i+2}} B_{i+2}^{d-1}(t)] \\ &- \frac{1}{t_{i+d+2} - t_{i+1}} B_{i+1}^d(t) \end{aligned} \tag{2.2}$$

We can simplify the terms containing $B_{i+1}^{d-1}(t)$ in the following way:

$$
\begin{aligned}
&dB_{i+1}^{d-1}(t)\left(\frac{t_i - t}{(t_{i+d+1} - t_i)(t_{i+d+1} - t_{i+1})} + \frac{t_{i+d+2} - t}{(t_{i+d+2} - t_{i+1})(t_{i+d+1} - t_{i+1})}\right) \\
&= dB_{i+1}^{d-1}(t)\left(\frac{t_i t_{i+d+2} - t_i t_{i+1} - t t_{i+d+2} + t t_{i+1} + t_{i+d+2} t_{i+d+1} - t_{i+d+2} t_i - t t_{i+d+1} + t t_i}{(t_{i+d+1} - t_i)(t_{i+d+1} - t_{i+1})(t_{i+d+2} - t_{i+1})}\right) \\
&= dB_{i+1}^{d-1}(t)\left(\frac{t_{i+d+2} t_{i+d+1} - t t_{i+d+2} - \mathbf{t_{i+1} t_{i+d+1}} + t t_{i+1}}{(t_{i+d+1} - t_i)(t_{i+d+1} - t_{i+1})(t_{i+d+2} - t_{i+1})}\right) \\
&+ dB_{i+1}^{d-1}(t)\left(\frac{\mathbf{t_{i+1} t_{i+d+1}} - t t_{i+d+1} - t_i t_{i+1} + t t_i}{(t_{i+d+1} - t_i)(t_{i+d+1} - t_{i+1})(t_{i+d+2} - t_{i+1})}\right) \\
&= dB_{i+1}^{d-1}(t)\left(\frac{(t_{i+d+2} - t_{i+1})(t_{i+d+1} - t) + (t_{i+d+1} - t_i)(t_{i+1} - t)}{(t_{i+d+1} - t_i)(t_{i+d+1} - t_{i+1})(t_{i+d+2} - t_{i+1})}\right) \\
&= dB_{i+1}^{d-1}(t)\left(\frac{t_{i+d+1} - t}{(t_{i+d+1} - t_i)(t_{i+d+1} - t_{i+1})} - \frac{t - t_{i+1}}{(t_{i+d+1} - t_{i+1})(t_{i+d+2} - t_{i+1})}\right)
\end{aligned}
$$

Substituting this back into 2.2 implies that the $d + 1$ case holds. Hence, by mathematical induction, it follows that the derivative formula holds. □

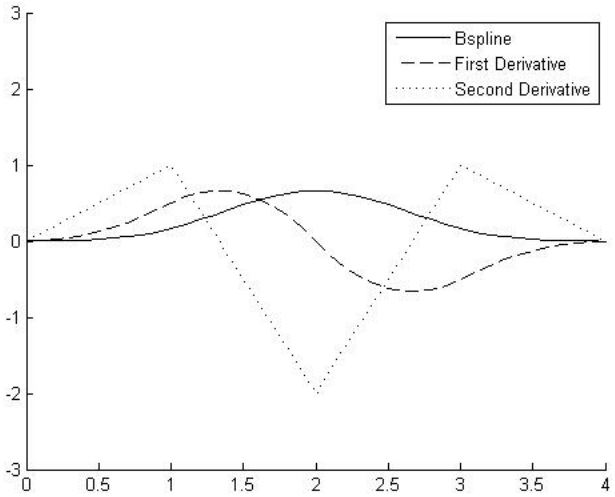


Figure 2.3: B_0^3 and derivatives with knots $(0, 1, 2, 3, 4)$

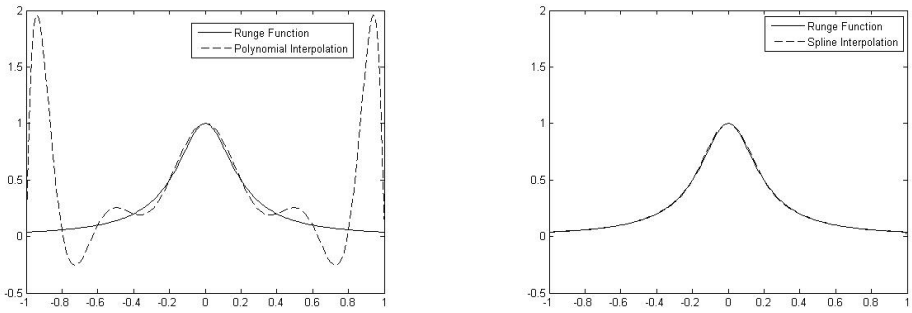
Using B-splines in interpolation is advantageous in that we can construct smooth curves of relatively low degree. Moreover, these spline curves typically yield less numerical error and can be easily and efficiently implemented.

2.2 Spline Interpolation

Typical polynomial interpolation can be achieved and improved upon in a variety of waves, but spline curves are much more useful in many cases. Using spline curves to interpolate can avoid error such as the Runge phenomenon

Consider the following Lagrange interpolation and Spline interpolation of the Runge function

$$f(x) = \frac{1}{1 + 25x^2}$$



(a) Polynomial Interpolation (b) Spline Interpolation

Figure 2.4: Comparison of Polynomial and Spline interpolation

As is evident in Figure 2.4, the spline curve interpolant and the Runge function are almost indistinguishable. Our goal is to use these interpolant spline curves as feasible starting points for solving optimization problems, so we need to be able to guarantee there existence and be able to calculate them easily.

Problem 2.5. *Given data points* (x_i, y_i) *for* $i = 1, ..., n$ *and a spline space* $\mathbb{S}$*, find a spline curve* $s(t)$ *such that* $s(x_i) = y_i$*.*

This problem is straightforward given certain conditions. First, consider the system in matrix form

$$Bc = y$$

$$\begin{bmatrix} B_1^d(x_1) & \cdots & B_n^d(x_1) \\ \vdots & \ddots & \vdots \\ B_1^d(x_n) & \cdots & B_n^d(x_n) \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

This system of equations will have a solution provide the B-spline collocation matrix, A, is nonsingular.

Theorem 2.6. *[3] The B-spline collocation matrix, A, is nonsingular if and only if the diagonal elements are positive for* $i = 1, ..., n$.

This condition is equivalent to the *Schoenberg-Whitney Conditions*:

$$t_i < x_i < t_{i+d+1}, i = 1, 2, ..., n$$

provided $x_i = t_i$ if $t_i = \cdots = t_{i+d}$.

Thus, we have conditions for the existence and uniqueness of the interpolating spline curve. Note that the Schoenberg-Whitney conditions imply that the rows of the B-spline collocation matrix are linearly independent. However, we are more interested in subjecting these spline curves to some kind of constraints. That is, what if we allowed the spline curve to achieve values in a type of neighborhood around certain nodes? Or perhaps we controlled the derivative of the spline curve, or the shape. In order to answer these questions, we need to utilize optimization tools, namely the active set method.

CHAPTER 3

OPTIMIZATION THEORY

3.1 Quadratic Programming

Our inequality-constrained spline optimization problem involves minimizing a function of quadratic form with linear equality constraint, i.e. quadratic programming. The problem of quadratic programming is as follows:

Problem 3.1. *Minimize the function*

$$\begin{aligned} &f(x) = \frac{1}{2}x^T Hx - x^T b; \\ &\textit{subject to: } Ax = B, Ex \leq F \end{aligned} \tag{3.1}$$

Here, we refer to $Ax = B$ as equality constraints, and $Ex \leq F$ as inequality constraints. While several techniques exist to solve this type of problem, the one we employ here is the *active set method*. The idea of the active set method is to solve a problem with linear equality and inequality constraints by iterations of solutions of equality-constrained problems with only linear equality constraints.

Definition 3.2. *A point, x^*, is called* ***feasible*** *provided it satisfies the constraints. That is, $Ax^* = B, Ex^* \leq F$.*

Definition 3.3. *A constraint E_i is called* ***active*** *if it is either an equality constraint, or if it is an inequality constraint that achieves equality, at some point x_k.*

Observe that in problem 3.1, if H is a positive semi-definite matrix, then we have a convex problem. That is, the function we are minimizing is convex, and the constraints form a convex set. This is important in that this implies that local minimizers are in fact global minimizers.

Theorem 3.4. *If H is positive semi-definite, then Problem 3.1 is a convex problem.*

Proof. Assume H is positive semi-definite, and let $\alpha \in (0,1)$. First, define $\Omega = \{x|Ax = B, Ex \leq F\}$. Let $x_1, x_2 \in \Omega$. Then we have $Ax_1 = B, Ax_2 = B$. Consider $(1-\alpha)x_1 + \alpha x_2$. Then, it follows that

$$\begin{aligned} A((1-\alpha)x_1 + \alpha x_2) &= A(1-\alpha)x_1 + A\alpha x_2 \\ &= (1-\alpha)B + \alpha B \\ &= B \end{aligned}$$

Similarly, we have $E((1-\alpha)x_1 + \alpha x_2) \leq F$. Hence, Ω is a convex set. However, it remains to show that f is a convex function, that is that $f((1-\alpha)x_1 + \alpha x_2) \leq (1-\alpha)f(x_1) + \alpha f(x_2)$.

$$\begin{aligned} f((1-\alpha)x_1 + \alpha x_2) &= \frac{1}{2}[(1-\alpha)x_1 + \alpha x_2]^T H[(1-\alpha)x_1 + \alpha x_2] - [(1-\alpha)x_1 + \alpha x_2]^T b \\ &= \frac{1}{2}[(1-\alpha)^2 x_1^T H x_1 + 2\alpha(1-\alpha)x_1^T H x_2 + \alpha^2 x_2^T H x_2] \\ &- (1-\alpha)x_1^T b - \alpha x_2^T b \end{aligned}$$

Hence, it follows that

$$\begin{aligned} &f((1-\alpha)x_1 + \alpha x_2) - (1-\alpha)f(x_1) - \alpha f(x_2) \\ &= \frac{1}{2}[(1-\alpha)^2 - (1-\alpha)]x_1^T H x_1 \\ &+ \alpha(1-\alpha)x_1^T H x_2 + \frac{1}{2}(\alpha^2 - \alpha)x_2^T H x_2 \\ &= \frac{1}{2}(\alpha^2 - \alpha)[x_1^T H x_1 - 2x_1^T H x_2 + x_2^T H x_2] \\ &= \frac{1}{2}(\alpha^2 - \alpha)[x_1 - x_2]^T H[x_1 - x_2] \leq 0 \end{aligned}$$

The last line follows since H is positive definite, and since $\alpha \in (0,1)$.

□

Quadratic programming is also based heavily on the well known Karush-Kuhn-Tucker (KKT) conditions. These conditions give us criterion by which to decide if

during our iterations in the active set method, we are at a solution.

Theorem 3.5. *Let x^* be a local minimizer of (3.1). Then there exist $\bar{\lambda}, \bar{\mu}$ such that*

- $\bar{\mu} \geq \bar{0}$
- $Hx^* - b + A^T\bar{\lambda} + E^T\bar{\mu} = \bar{0}$
- $\bar{\mu^T}(F - Ex^*) = 0$
- $Ax^* = B$
- $Ex^* \leq F$

The KKT conditions help provide existence and uniqueness conditions for the quadratic programming problem. This is discussed in detail in [11], and a theorem is presented here without proof. The basic idea is that we can ensure that a feasible point is a local minimizer, and since the problem is convex, this is a unique solution. This is a necessary and sufficient condition for optimality.

Theorem 3.6. *Let x^* be a feasible point for the problem 3.1. Then x^* is a local minimizer provided that $x^*, \bar{\lambda}$ satisfy the KKT conditions, and also that $d^T H d \geq 0$, for all d such that $A_i d = 0$ for equality constraints, $E_i d \geq 0$ for active inequality constraints, and $E_i d = 0$ for active inequality constraints with positive lagrange multipliers.*

So, first, let us consider the equality constrained case, with no inequality constraints. We develop here in detail the generalized variable elimination method, which is described in [11]

Theorem 3.7. *Let Y be a matrix whose columns form a basis of $R(A^T)$, and let Z be a matrix whose columns for a basis for $N(A)$. Assume that A is full row rank, and*

that $Z^T HZ$ is invertible. Suppose x solves problem 3.1 with lagrange multipliers $\bar{\lambda}$. Then

$$x = Y(AY)^{-1}B - Z(Z^THZ)^{-1}Z^T(HY(AY)^{-1}B - b)$$
$$\bar{\lambda} = (AY)^{-T}Y^T(b - Hx)$$

Proof. We begin by partitioning $\mathbb{R}^n$ as $R(A^T) \oplus N(A)$. Let Y be a basis for $R(A^T)$ and let Z be a basis for $N(A)$. Then we can say $x = Y\bar{x} + Z\hat{x}$, with $\bar{x} \in R(A^T)$ and $\hat{x} \in N(A)$. From the equality constraint, $Ax = B$, we have that

$$A(Y\bar{x} + Z\hat{x}) = B \Rightarrow AY\bar{x} = B \Rightarrow \bar{x} = (AY)^{-1}B$$

Note that AY is nonsingular by construction of Y. Hence,

$$x = Y(AY)^{-1}B + Z\hat{x} \tag{3.2}$$

Our goal is to develop a formula for the solution of the the quadratic program. substituting form of x into our original problem gives:

$$\frac{1}{2}x^THx - x^Tb = \frac{1}{2}[Y(AY)^{-1}B + Z\hat{x}]^TH[Y(AY)^{-1}B + Z\hat{x}] - [Y(AY)^{-1}B + Z\hat{x}]^Tb$$
$$= \frac{1}{2}[B^T(AY)^{-T}Y^T + \hat{x}^TZ^T]H[Y(AY)^{-1}B + Z\hat{x}] - [B^T(AY)^{-T}Y^T + \hat{x}^TZ^T]b$$
$$\frac{1}{2}B^T(AY)^{-T}Y^THY(AY)^{-1}B + \frac{1}{2}B^T(AY)^{-T}Y^THZ\hat{x} + \frac{1}{2}\hat{x}^TZ^THY(AY)^{-1}B$$
$$+\frac{1}{2}\hat{x}^TZ^THZ\hat{x} - B^T(AY)^{-T}Y^Tb - \hat{x}^TZ^Tb$$

This gives us an unconstrained problem:

Problem 3.8. *Minimize the following function, assuming that A has full row rank:*

$$\frac{1}{2}\hat{x}^TZ^THZ\hat{x} + \hat{x}^TZ^T(\frac{1}{2}HY(AY)^{-1}B - b) \tag{3.3}$$

Assuming that $Z^T HZ$ is positive definite, then we have that

$$\hat{x} = -(Z^T HZ)^{-1} Z^T (HY(AY)^{-1} B - b)$$

Substituting this back into 3.2, we arrive at a solution for this problem:

$$x = Y(AY)^{-1} B - Z(Z^T HZ)^{-1} Z^T (HY(AY)^{-1} B - b) \tag{3.4}$$

Furthermore, from the KKT conditions, we have that

$$\begin{aligned} Hx - b + A^T \bar{\lambda} &= 0 \\ A^T \bar{\lambda} &= b - Hx \\ Y^T A^T \bar{\lambda} &= Y^T (b - Hx) \\ (AY)^T A^T \bar{\lambda} &= Y^T (b - Hx) \\ \bar{\lambda} &= (AY)^{-T} Y^T (b - Hx) \end{aligned} \tag{3.5}$$

□

3.2 Active Set Method

Now, with a method for solving equality constrained problems, we can develop the *active set method* for solving problems with linear inequality constraints. The idea is as follows: given an initial, feasible point, we aim to calculate a step direction to iterate the point. On each iteration the point should remain feasible, and on each iteration, to calculate this step direction, we solve a subproblem with only equality constraints. If, after we iterate the point, the point is no longer feasible, then we need to add constraints to consider. If, however, the point is feasible, then we need to examine if the KKT conditions are satisfied. If not, then we can drop a constraint and continue searching for the solution.

Let x_k be the kth iterate during the active set method. In order to iterate and find $x_{k+1} = x_k + \alpha_k d_k$, we need to find the descent direction d_k by solving the following:

Problem 3.9. *Minimize the function*

$$\begin{aligned} &\frac{1}{2}x_{k+1}^T H x_{k+1} - x_{k+1}^T b; \\ &\textit{subject to: } Ax_{k+1} = B, Ex_{k+1} = F \end{aligned} \tag{3.6}$$

where we are considering the active inequality constraints.

It is straightforward to verify that equation 3.6 is equivalent to

$$\frac{1}{2}d^T H d - (b - Hx_k)^T d$$

where $Ad = \bar{0}; Ed = \bar{0}$ for active inequality constraints. This is an equality constrained optimization problem, and can be solved using the generalized variable elimination method, and equation 3.4. Now, if the step direction is zero, we have a chance of being at a solution. Using 3.5, we check to see if the lagrange multipliers are all positive. If so, we have arrived at an answer. If not, then we drop the constraint associated with the smallest multiplier.

Definition 3.10. *The working set S_k is comprised of the equality and active inequality constraints, at the k^{th} iteration of the active set method.*

If $d_k \neq 0$, then we need to continue searching for a solution and add a constraint into the working set. To do so, we find a $j \neq S$ such that $E_j(x_{k+1}) = F_j$, where E_j, F_j represent the jth inequality constraint. This j is chosen according by

$$\min\{\frac{F_i - E_i x_k}{E_i d_k}\}$$

where E_i is a row of the matrix E, consisting of the constraints chosen in the working set. Index j will be the index associated with this minimum value. Moreover, we

choose a step length α_k by choosing either 1 or this value, whichever is smallest. After adding this constraint to the working set, the process is repeated until a solution is reached.

This discussion is summarized in Algorithm 1 and code is provided in appendix A.1.

Algorithm 3.11 (Active Set Method)**.**

1. *Given a feasible starting point x_1, initialize the working set S_1.*

2. *Find the step direction by solving the equality constrained sub-problem 3.6.*

   - *If $d_k = 0$, check the Lagrange multipliers.*
     - *If all of the Lagrange multipliers are non-negative in the working set, stop.*
     - *Otherwise, drop the constraint associated with the negative Lagrange multiplier.*
   - *If $d_k \neq 0$, then find α_k as above.*
     - *If $\alpha_k = 1$, iterate, set $x_{k+1} = x_k + \alpha_k d_k$, and return to step 2.*
     - *Otherwise, find $j \notin S_k$ such that*

$$E_j^T(x_k + \alpha_k d_k) = F_j$$

Then, add this constraint to the working set. Return to step 2.

3.3 Convergence

It is worth mentioning that we focus on the case in which the matrix H is positive semi-definite, and thus, a convex problem. Other techniques exist for solving problems that are not of this form, but optimizing inequality constrained splines will satisfy

these conditions. In particular we want to ensure this method converges in finite iterations and that a solution exists. This is discussed in detail in [11].

Theorem 3.12. *If, $\forall k$, the working set S_k is linearly independent, then either the sequence generated from the Active Set method converges to a KKT point of the problem in finite iterations, or the problem is unbounded below.*

So, a critical notion is that in order for this method to work on our problems, we need to have linear independence in the working set. Since the working set is a subset of all equality and inequality constraints, it is easy to imagine that some of the constraints could be linearly independent. However, we will show that for our particular inequality constrained spline optimization problems that this is not the case.

CHAPTER 4

INEQUALITY CONSTRAINED SPLINE OPTIMIZATION FOR SPLINE FUNCTIONS

In applying active set method to solving spline optimization problems, we wish to find a spline curve subject to some constraints, which has minimal "bending energy". We will develop several types of constraints, but first consider simple constraints of bounding the function around data points.

Definition 4.1. *Given a function $f(x)$ on an interval $[a, b]$, the bending energy of the function, $E(f)$, is defined as*

$$E(f) := \int_a^b |f''(x)|^2 dx.$$

4.1 Function Value Constraints

Problem 4.2. *Given data (x_i, y_i) and a spline space $\mathbb{S}$, find a spline curve $s(t)$ such that $a_i \leq s(x_i) \leq b_i$ and minimizes $E(s)$.*

To formulate this problem to match the notation of problem 3.1, recall that we define spline curves in terms of linear combinations of B-splines. So, if we have a basis Φ and coefficients α, then we have that $s = \Phi\alpha$ and

$$\int |s''(x)|^2 = \int (\Phi''\alpha)^2 = \alpha^T (\int \Phi''^T \Phi'')\alpha.$$

Considering the integration in the perspective of an inner product, then the matrix given by

$$H = \int \Phi''^T \Phi'' \tag{4.1}$$

is known as the Gramian matrix. This matrix can be shown to be positive semi-definite. In order to take derivatives of the B-splines, we will use Theorem 2.4. In

order to complete the integral, we will use Gaussian quadrature, which can be shown to be exact for polynomials of certain degree. Since splines are polynomials in-between knots, we can integrate them over each knot interval using this technique.

Remark 4.3. *For a degree 3 polynomial, the Gaussian quadrature rule*

$$\int_{-1}^{1} f(x)dx = f(\frac{-\sqrt{3}}{3}) + f(\frac{\sqrt{3}}{3})$$

is exact. For many examples in this thesis, we choose our spline space to consist of degree 3 B-splines

Remark 4.4. *Each spline function is a polynomial in-between knots, so we integrate over each knot interval and then take a sum. In order to accomplish this with gauss quadrature, we can use the following coordinate transformation for the sub-interval* $[a, b]$*:*

$$t = \frac{b-a}{2}x + \frac{b+a}{2}$$

Code reflecting this process can be found in Appendix A.2.

Now, our goal is to first develop the quadratic program to be solved. First, consider the function value constraints that we have in this problem: $a_i \leq s(x_i) \leq b_i$. Considering the right inequality first, we have:

$$s(x_i) < b_i \Rightarrow \sum_{j=1}^{n} c_j B_j^d(x_i) < b_i$$

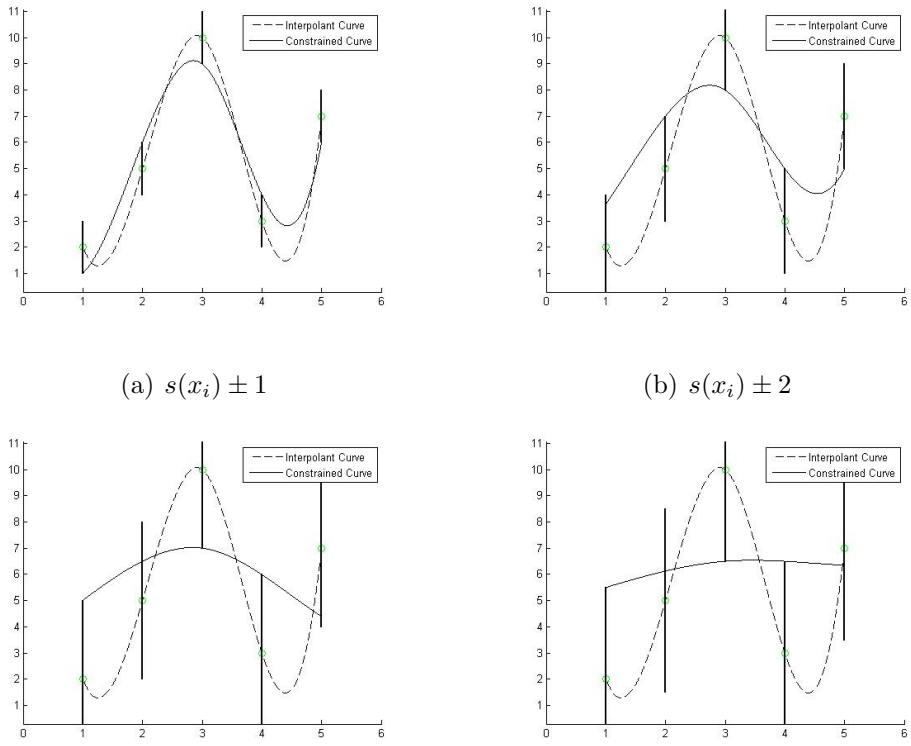
$$\begin{bmatrix} B_1^d(x_1) & \cdots & B_n^d(x_1) \\ \vdots & \ddots & \vdots \\ B_1^d(x_n) & \cdots & B_n^d(x_n) \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} \leq \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

Similarly, from the left hand inequality, we can conclude that

$$\begin{bmatrix} -B_1^d(x_1) & \cdots & -B_n^d(x_1) \\ \vdots & \ddots & \vdots \\ -B_1^d(x_n) & \cdots & -B_n^d(x_n) \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} \leq \begin{bmatrix} -a_1 \\ \vdots \\ -a_n \end{bmatrix}$$

Concatenating these matrices gives us the E and F matrices for the inequality constraints of the quadratic programming problem. In the basic problem, we do not consider having equality constraints, but we could easily impose some in a similar fashion. Furthermore, we choose to start with the coefficients of the interpolant spline curve for a feasible starting point for this problem. We typically start with these coefficients, but given the constraints, a different starting point may be required.

Example 4.5. Consider the data points $\{(1,2),(2,5),(3,10),(4,3),(5,7)\}$ We begin by finding the interpolating spline, and use this as a starting point in the active set method. In four different applications of the algorithm, we use increasingly relaxed constraints to exemplify that the bending energy is indeed decreasing.



(a) $s(x_i) \pm 1$ (b) $s(x_i) \pm 2$

(c) $s(x_i) \pm 3$ (d) $s(x_i) \pm 3.5$

Figure 4.1: Increasingly relaxed function value constraints

Example 4.6. This example is a counter-example to a problem in [8], but we solve here using the active set method. Consider data points

$$\{(1.5, 1.7), (2, 2.7), (3, 4.2), (4, 5.1), (6, 4.7), (7, 4.8)\}$$

We impose function value constraints of $\pm.7$ around each data point,except at the last two end points, where we impose equality constraints. In addition to this we wish to impose that *natural end conditions* be met. That is, that the second derivative of the spline curve vanishes at the end points of this interval.

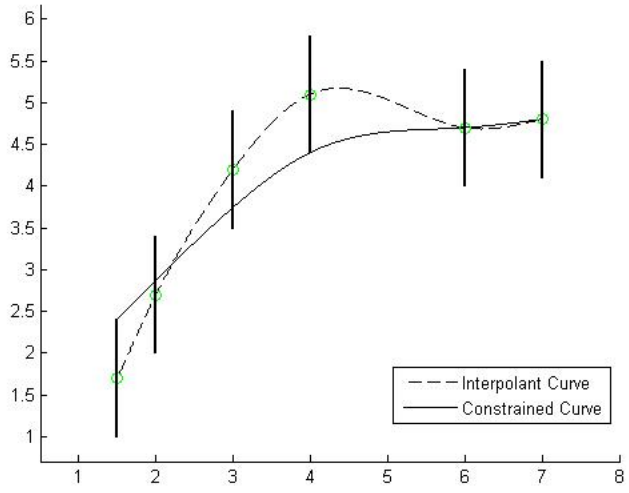


Figure 4.2: Example 3.5

4.2 Convergence

By theorem 3.12, we know that the active set method will converge to a solution in finite time provided that the working set is linearly independent. We achieve linear independence based on the following:

Lemma 4.7. *With function-valued constraints, no two constraints around a single data point can be active at the same time.*

It is clear that a function cannot, of course, achieve two different y values. However, we want to ensure that the active set method will not choose to add a constraint at a data point if there is already a constraint for that data point in the working set. This is achieved by the way that we choose to add constraints to the working set. Suppose that we have the constraint E_i in the working set. Then $E_i x_k = F_i$. The constraint on the "other side" of the point can be written as $-E_i$. This point, however, is chosen such that $i \notin S_k$ and that $E_i d_k < 0$. However, since E_i is in the working set already, we have $E_i d_k = 0 \Rightarrow -E_i d_k = 0$. Hence, this constraint will never be chosen to be added to the working set.

Lemma 4.8. *With function valued constraints, constraints around two different data points will be linearly independent.*

This follows directly from the Schoenberg-Whitney conditions. Combining these two lemmas gives the following theorem:

Theorem 4.9. *In applying the active set method to problem 4.2, the working set will always be linearly independent. Hence, the active set method will converge in finite time, or the problem is unbounded below.*

Proof. We proceed by induction on the iteration, k. The initial working set will consist of any equality constraints and any active inequality constraints at the initial point,

and these constraints must be linearly independent by the previous two lemmas. If at the k^{th} iteration the working set is linearly independent, then we need only consider the case in which we add a constraint to generate S_{k+1}, as removing a constraint will certainly not alter linear independence. The constraint which will be added will not be a constraint "opposite" a constraint already existing in the working set, per the above discussion. Hence, the constraint will be at some other data point, and will hence be active. Therefore, the working set will always be linearly independent. □

4.3 Derivative Constraints and Shape Preserving Constraints

Problem 4.10. *Given data* (x_i, y_i) *and a spline space* $\mathbb{S}$ *find a spline curve* $s(t)$ *such that* $a'_i \leq s'(x_i) \leq b'_i$ *and minimizes* $E(s)$.

We can set this problem up in the exact same way as the function value constraint problem, replacing the B-spline functions with their respective derivatives.

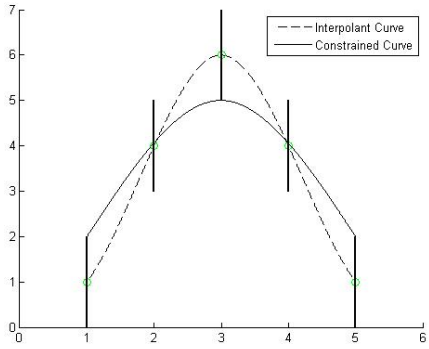
Example 4.11. Consider the data points $\{(1,1),(2,4),(3,6),(4,4),(5,1)\}$. We fix our spline space with degree $d = 3$ and knot sequence $T = [1,1,1,1,3,5,5,5,5]$. First, we impose function value constraints of ± 1 around each data point. Next, we further add derivative constraints of ± 1 at $x = [1.3, 1.8, 2.5, 4.5]$.

Shape preserving constraints are similar in that we can employ a technique of simply putting an upper and lower bound on some property of the data we are given, or curve we are starting with. In this case we place a constraint on the second differences of the curve.

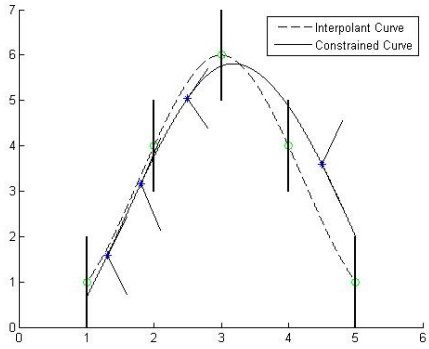
Example 4.12. Consider the data points

$$\{(1,2),(2,5),(3,7.3),(4,4.5),(5,1),(6,3),(7,-4),(8,10)\}$$

. We fix our spline space with degree $d = 3$ and knot sequence $T = [1,1,1,1,3,5,5,5,5]$.

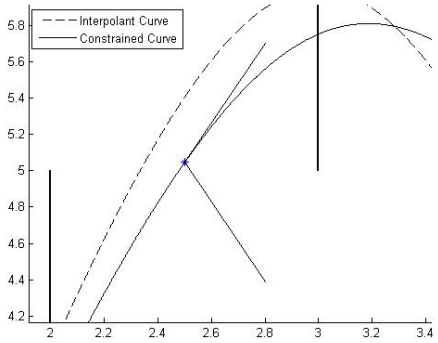


(a) No Derivative Constraints



(b) With Derivative Contraints

Figure 4.3: Spline Curve with Derivative Constraints

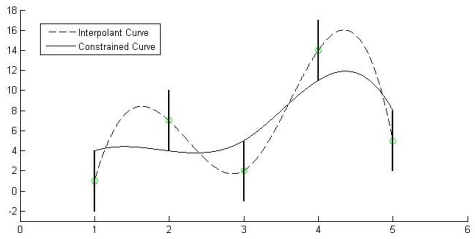


(a) Bounds on Derivative

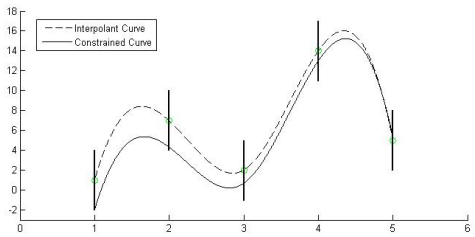
Remark 4.13. *In applying the active set method to these problems, the working set is also guaranteed to be linearly independent, for similar reasons as in the function value constrained problem.*

4.4 Choosing Data Sites

In all the examples in this chapter, the active set method converges to a unique solution which minimizes bending energy, given the initial data points. However, if we allow ourself to change these points slightly, we can come up with an even better answer, which might prove useful in some general function approximation sense. The



(b) No Shape Preserving Constraints



(c) With Shape Preserving Constraints

Figure 4.4: Spline Curve with Shape Preserving Constraints

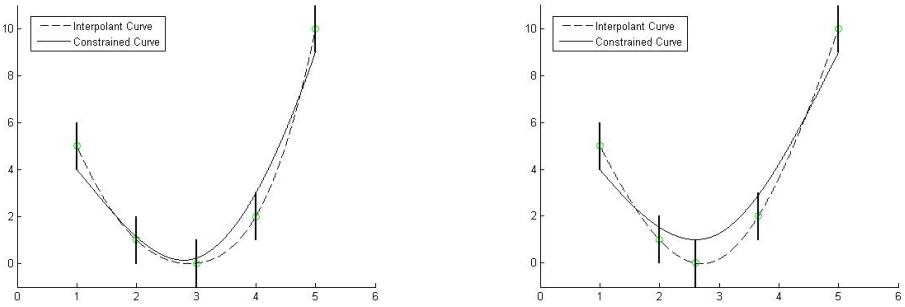
idea is as follows: after solving for the spline coefficients which solve an inequality constrained optimization problem, find which constraints are considered active, except for the end point constraints. For these, update $x_i = x_i + \Delta x_i$ where

$$\Delta x_i = \frac{-(y_i - s(x_i))s'(x_i)}{(y_i - s(x_i))s''(x_i) + (s'(x_i))^2} \tag{4.2}$$

This will cause the derivative to vanish at the active constraints, decreasing the bending energy.

Example 4.14. Consider the data points $\{(1,5),(2,1),(3,0),(4,2),(5,10)\}$ with a spline space with degree $d = 3$ and knot sequence $T = [1,1,1,1,3,5,5,5,5]$. We apply equation 4.2 repeatedly and record the bending energy, $E = \int_a^b |s''(t)|^2$

i	0	1	2	3	4	5
E	32.5174	26.9026	22.7724	21.0450	20.9198	20.8081



(a) Before Iterating (b) After Iterating

Figure 4.5: Iterating Data points

Formula 4.2 is derived as follows: first, consider the function $F(x_i) = (y_i - s(x_i))^2$. By Taylor's theorem, we have that

$$F(x_i + \Delta x_i) \approx F(x_i) + F'(x_i)\Delta x_i + F''(x_i)\frac{\Delta x_i^2}{2}$$

We want to minimize the derivative of this expansion. We can do so with a Newton iteration.

$$\frac{dF(x_i)}{d\Delta x_i} = F'(x_i) + F''(x_i)\Delta x_i = 0$$

$$\Rightarrow \Delta x_i = -\frac{F'(x_i)}{F''(x_i)}$$

Differentiating the original function, we have that

$$\begin{aligned} F'(x_i) &= 2(y_i - s(x_i))(-s'(x_i)) \\ &= -2y_i s'(x_i) + 2s(x_i)s'(x_i) \\ F''(x_i) &= -2(y_i - s(x_i))(s''(x_i)) + 2(s'(x_i))^2 \\ &= -2y_i s''(x_i) + 2s(x_i)s''(x_i) + 2(s'(x_i))^2 \end{aligned}$$

Substituting these back in for Δx_i gives us the desired formula.

CHAPTER 5

INEQUALITY CONSTRAINED SPLINE OPTIMIZATION FOR SPLINE CURVES

5.1 Set-up and Constraints

Problem 5.1. *Given a data set (x_i, y_i) and spline space $\mathbb{S}$, generated from a parametric curve $s(t) = \langle x(t), y(t) \rangle$, find a spline curve*

$$s_x(t) = \sum c_{x,j} B_j^d(t)$$
$$s_y(t) = \sum c_{y,j} B_j^d(t)$$

such that the curve fits inside regular, convex polygons around the data points, and minimizes

$$\int_a^b |s''(t)|^2 dt = \int_a^b (s_x''(t))^2 + (s_y''(t))^2 dt$$

Problem 5.1 is slightly different in that we need to find two sets of coefficients, but we still want to minimize the bending energy. We can define the bending energy as follows:

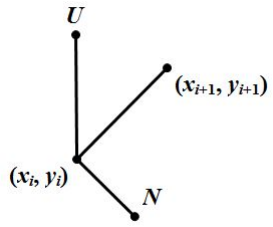
$$\begin{aligned} E &= \int |s''(t)|^2 dt = \int (s_x''(t))^2 + (s_y''(t))^2 dt \\ &= \alpha_x^T H \alpha_x + \alpha_y^T H \alpha_y \\ &= \begin{bmatrix} \alpha_x & \alpha_y \end{bmatrix}^T \begin{bmatrix} H & 0 \\ 0 & H \end{bmatrix} \begin{bmatrix} \alpha_x \\ \alpha_y \end{bmatrix} \end{aligned}$$

where H is the Gramian matrix as described in 4.1. Since H is positive semi-definite, the matrix here will be as well, by construction. In this problem we consider inequality constraints to be normal polygons around each data point. In order to do this, for a

polygon of n sides we choose n equally spaced points around a parametrically defined circle. Each side of the polygon will be a linear constraint. To write these constraints down rigorously, we loop around the polygon and take dot products of the surface normals. Suppose $v = \langle x_{i+1} - x_i, y_{i+1} - y_i \rangle$, $N = \langle y_{i+1} - y_i, -(x_{i+1} - x_i) \rangle$, and $u = \langle x - x_i, y - y_i \rangle$. We want that $u \cdot N \leq 0$. Hence,

$$(x - x_i)(y_{i+1} - y_i) - (y - y_i)(x_{i+1} - x_i) \leq 0$$

$$(y_{i+1} - y_i)x - (x_{i+1} - x_i)y \leq x_i(y_{i+1} - y_i) - y_i(x_{i+1} - x_i)$$

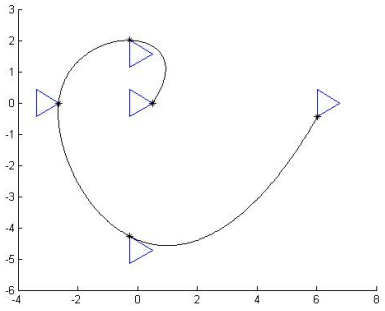


We can then iterate about each point, iterate through each side of the polygon surrounding the point, and generate n linear constraints. MATLAB code reflecting this process can be found in appendix A.2. In this regard, the problem is in exactly the form we need it to be to utilize the active set method.

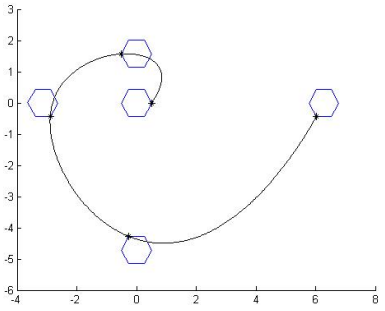
Example 5.2. Consider the parametric curve

$$x(t) = t\cos(t); y(t) = t\sin(t)$$

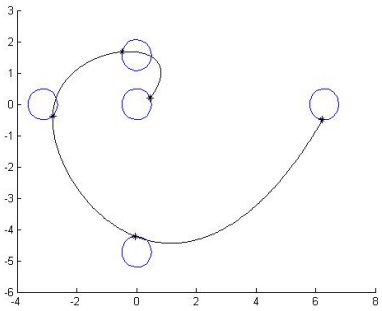
and obtain data points from the parameter values $t = [0, \pi/2, \pi, 3\pi/2, 2\pi]$. We choose degree $d = 3$ and knot sequence $T = [0, 0, 0, 0, \pi, 2\pi, 2\pi, 2\pi, 2\pi]$ for our spline space.



(a) Three sides



(b) Six sides



(c) Fifteen sides

Figure 5.1: Parametric Spline Curves

We demonstrate in Figure 5.2 that any number of sides can be used for the polygonal constraints, although large values for n will cost more CPU time. In this way, the constraints in this problem can approach a circle, a non-linear constraint.

Example 5.3. Consider the parametric curve

$$x(t) = 2\cos(t)(1-\cos(t)); y(t) = 2\sin(t)(1-\cos(t))$$

with parameter values $t = [0, \pi/2, \pi, 3\pi/2, 2\pi]$. We choose degree $d = 3$ and knot sequence $T = [0, 0, 0, 0, \pi, 2\pi, 2\pi, 2\pi, 2\pi]$. We also use regular convex pentagons to impose constraints.

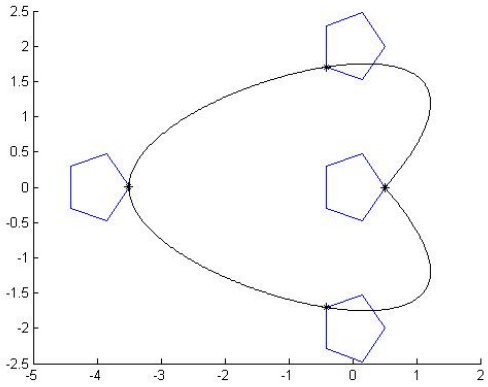


Figure 5.2: Cardioid

5.2 Convergence

Again, in order to claim that the active set method converges on this problem, we need only show that the working set will be linearly independent, and invoke theorem 3.12. In problem 5.1, the constraints can be viewed as convex polygons surrounding data points. There are only two possibilities for a point to be considered active - either only one side of the polygon is active, or the curve hits the polygon at a corner point, in which two constraints matching that data point will be active at the same time.

Lemma 5.4. *Polygonal constraints around different points will be linearly independent.*

This follows, again, from the Schoenberg-Whitney conditions.

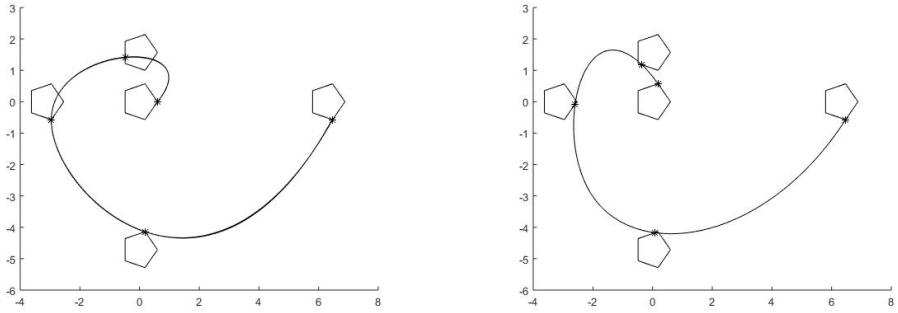
Lemma 5.5. *In applying the active set method to problem 5.1, a linearly dependent constraint will never be added to the working set.*

This follows from a very similar discussion to that of lemma 4.7. Combining these ideas ensures the converges of the active set method on this problem.

Theorem 5.6. *In applying the active set method to problem 5.1, the working set will always be linearly independent, and hence the active set method will converge in finite time to a solution, or the problem is unbounded below.*

5.3 Choosing Data Sites

We can also iterate to choose optimal data sites just as in the function case. Considering example 5.2 , we iterate to choose optimal data sites. Here, we use a larger pentagon to emphasize the iteration. As shown in the following table, the bending energy of the curve does in fact decrease.



(a) Before Iterating (b) After Iterating

Figure 5.3: Iterations to reduce bending energy

i	0	1	2	3	4	5	6	7	8
E	48.56	39.57	33.26	28.46	24.53	21.43	20.90	20.87	20.46

In order to derive the formula for this iteration, we follow the same process as described in deriving 4.2, except we need to utilize a dot product instead of a normal product. That is to say, $F(x_i) = \langle y_i - s(x_i), y_i - s(x_i)\rangle$, where $\langle \cdot, \cdot \rangle$ represents a dot product. Here, we have that

$$F'(x_i) = -2\langle y_i - s(x_i), s'(x_i)\rangle$$
$$F''(x_i) = -2\langle y_i - s(x_i), s''(x_i)\rangle + 2\langle s'(x_i), s'(x_i)\rangle$$

Substituting gives the desired formula. A MATLAB example of this can be found in appendix A.2

5.4 Concluding Remarks

Inequality constrained spline optimization problems can be solved very effectively with the active set method. The method is guaranteed to converge to a solution in finite time, provided the problem is posed in the correct way and that a solution exists. Moreover, the method is efficient and easy to implement. In the future, we would like to further improve on our code so as to be able to solve this problem more efficiently than general optimization packages. Also, we would like to develop a new iteration involving the parametric case; that is, to approximate non-linear constraints using the linear polygonal constraints.

REFERENCES

[1] M. Atteia, *Fonctions "Spline" Avec Constraintes Lineaires de type Inegalite* Congrès de l'AFRIO, Nancy, Mai. 43 – 54 (1967)

[2] M. Atteia, *Fonctions "spline" definies sur un ensemble convexe* Numer. Math. 12, 192 - 210 (1968)

[3] C. de Boor, *Total Positivity of the Spline Collocation Matrix* Indiana University Mathematics Journal (1976)

[4] C. de Boor, *A Practical Guide for Splines* Springer (New York) (1978)

[5] M. Ignatov and A. Pevnyj, *Natural Splines of Several Variables* Nauka, Leningrad. Otdel, Leningrad, (in Russian). (1991)

[6] S. Kersey, *Near-Interpolation* Numer. Math. 94 523 540. (2003)

[7] S. Kersey, *On the problems of smoothing and near-interpolation* Math. Comp. 72 1873 - 1885. (2003)

[8] E. Leetma and P. Oja, *A Method of Adding-Removing Knots for Solving Smoothing Problems with Obstacles* European Journal of Operational Research (2009)

[9] J. Prochazkova, *Derivative of B-Spline Function* Konference O Geometrii a Počítačoveé Grafice

[10] L. Schumaker and O. Mangasarian, *Splines via Optimal Control* Applications with special emphasis on spline functions, ed. J. Schoenberg Academic Press 119 - 156 (1969)

[11] W. Sun and Y. Yuan, *Optimization Theory and Methods: Nonlinear Programming* Springer (New York) (2006)

Appendix A

MATLAB CODE

Included here is code written in MATLAB to execute the active set method on a quadratic programming problem. Further, several routines are included for Splines.

A.1 Active Set Method

```
function [ x, lambda, mu, niter,S ] = ...
    joshactivesetoptim(H,b,A,B,E,F,x )
%A program which uses the active set method in order to solve a convex
%quadratic optimization problem. In other wordse, minimize:
%1/2x'Hx - x'b
%Subject to the equality constraints Ax = B
%Subject to the inequality constraints Ex <= F
%With initial guess x. The initial guess must be feasible.
%%Josh Holloway, Dr. Scott Kersey
%%Georgia Southern University
%%2015
lambda = [];
mu = [];
tol = 10^-10;
%Unconstrained problem
if nargin == 2
    if(rank(H) == rank([H,b]))
        if(det(H) < tol)
            x = pinv(H)*b;
        else
            x = H\b;
        end
```

```
    else
        fprint('There is not a solution to this problem');
        x = [0;0];
    end
end
%Equality Constrained Problem
if nargin == 4
    Y = orth(A');
    Z = null(A);
    x = Y*inv(A*Y)*B -Z*inv(Z'*H*Z)*Z'*(H*Y*inv(A*Y)*B-b); %good
    lambda = inv(A*Y)'*Y'*(b - H*x);
end
%Inequality Constrained Probelm
if nargin == 7
    S = find(E*x - F > -tol); S = S(:)';
    mu = -1;
    maxiter = 50;
    niter = 0;
    lambda = 0;
    [m,n] = size(A);
    [s,t] = size(E);
    while(any((mu)<0)) && (niter < maxiter)
       niter = niter + 1;
       AA = [A; E(S,:)];
       zz = zeros(length(AA(:,1)),1);
       bb = b-H*x;
       [d,lambda,mu] = joshactivesetoptim(H,bb,AA,zz);
       alpha = 1;
       if norm(d)<tol
           mu = zeros(s,1);
           [mm,nn] = size(AA);
```

```
            zz = [B; F(S,:)];
            [xx,ll,mu] = joshactivesetoptim(H,bb,AA,zz);
            lambda = ll(1:m);
            mu(S) = ll(m+1:end);
            if any(mu<0)
                i = find(mu == min(mu));
                S = setdiff(S,i);
            end
        else
            mu = -1;
            Sc = setdiff(1:s, S);
            dSc = Sc(find(E(Sc,:)*d >0));
            alpha = 1;
            if ~isempty(dSc)
                v = (F(dSc) - E(dSc,:)*x)./(E(dSc,:)*d);
                alpha = min(1,min(v));
            end
            if alpha ~=1
               j = dSc(find(v==alpha, 1 ));
               S = sort([S,j]);
            end
        end
        x = x + alpha*d;
    end
niter
end
end
```

A.2 Spline Code

```
function y = bspline( d,i, t, T, b)
%Recursive definition of b-spline
%d is the degree of the ith spline, evaluated at t, with knot vector ...
   T and
%b the endpoint of the interval.
%Base-Case. Includes modification to correct for the end-point ...
   interpolation
%problem with open splines.
%%Josh Holloway, Dr. Scott Kersey
%%Georgia Southern University
%%2015
y=0;
if d == 0
   %4 cases included to correct for issue where 0/0 = 0.
   if (t >= T(i) && t < T(i+1)) || (T(i) < T(i+1) && T(i+1) ==b && t ...
      >= T(i) && t <= T(i+1))
       y = 1;
   end
elseif t>=T(i) && t<=T(i+d+1)
  if T(i+d) > T(i)
    y = ((t - T(i))/(T(i+d) - T(i)))*bspline(d-1,i,t,T,b);
  end
  if T(i+d+1) > T(i+1)
   y = y + ((T(i+d+1) - t)/(T(i+d+1)-T(i+1)))*bspline(d-1,i+1,t,T,b);
  end
end
end
```

```
function [ y ] = bsplinederiv( d,i,t,T,b)
%Evaluates the derivative of a b-spline of degree d, at t, with knot
%sequence T.
y = 0;
if(T(i+d) > T(i))
   if(T(i+d+1) > T(i+1))
       y = (d/(T(i+d)-T(i)))*bspline(d-1, i, t, T, b) - (d/(T(i+d+1) ...
          - T(i+1)))*bspline(d-1,i+1,t,T,b);
   elseif(T(i+d+1) == T(i+1))
       y = (d/(T(i+d)-T(i)))*bspline(d-1, i, t, T, b);
   end
elseif(T(i+d+1)>T(i+1))
    y = -(d/(T(i+d+1) - T(i+1)))*bspline(d-1,i+1,t,T,b);
end
end
```

```
function [ y ] = bsplineseconderiv( d,i,t,T,b )
%Calculate the second derivative of a bspline.
%Could code in a recursive way, but currently this is sufficient.
y = 0;
if(T(i+d) > T(i))
    if(T(i+d+1) > T(i+1))
         y = ((d/(T(i+d)-T(i)))*bsplinederiv(d-1,i,t, T,b) - ...
            (d/(T(i+d+1)-T(i+1)))*bsplinederiv(d-1,i+1,t,T,b));
    else
        y = (d/(T(i+d)-T(i)))*bsplinederiv(d-1,i,t, T,b);
    end
else
    y = -(d/(T(i+d+1)-T(i+1)))*bsplinederiv(d-1,i+1,t,T,b);
end
```

```
end


function [ c ] = splinefinder( x, y, d, T, b)
%Solves the problem Ac = y, where A is a matrix of B-splines of ...
   degree d,
%evaluated at xi in x, with respect to the knot vector t. y are the
%corresponding values to the x vector for the data we wish to ...
   interpolate.
%b is the endpoint of the interval.

m = length(x);
n = length(T)-d-1;
A = [];
for i= 1:m
    for j = 1:n
        A(i,j) = bspline(d,j,x(i),T,b);
    end
end
c = A\y';
%Natural spline end conditions with second derivative equal to zero.
% for j = 1:n
%     A(m+1,j) = bsplinesecondderiv(d,j,x(1),T,b);
%     A(m+2,j) = bsplinesecondderiv(d,j,x(end),T,b);
% end
% y = [y 0 0];
% c = A\y';


function [ H ] = splinegaussquad( d, T, n )
%d - degree
```

```
%T  - knot sequence
H = zeros(n);
if d == 3
for i = 1:n
    for j = 1:n
        sum = 0;
        for k = 1:(length(T)-1)

           if T(k) < T(k+1)
               s = T(k);
               t = T(k+1);
               num = .5*((t-s) * (sqrt(3)/3) + (t+s));
               num2 = .5*((t-s) * (-sqrt(3)/3) + (t+s));
               Wi1 = bsplinesecondderiv(d,i,num,T,T(end));
               Wj1 = bsplinesecondderiv(d,j,num,T,T(end));
               Wi2 = bsplinesecondderiv(d,i,num2,T,T(end));
               Wj2 = bsplinesecondderiv(d,j,num2,T,T(end));
               sum = sum + Wi1*Wj1 + Wi2*Wj2;
           end
        end
        H(i,j) = sum;
    end
end
end
if d == 4
    for i = 1:n
        for j = 1:n
            sum = 0;
            for k = 1:(length(T)-1)
                if T(k) < T(k+1)
                   s = T(k);
```

```
                    t = T(k+1);
                    num = .5*((t-s) * (sqrt(3/5)) + (t+s));
                    num2 = .5*((t-s) * (-sqrt(3/5)) + (t+s));
                    num3 = .5*((t-s) * (0) + (t+s));
                    Wi1 = bsplinesecondderiv(d,i,num,T,T(length(T)));
                    Wj1 = bsplinesecondderiv(d,j,num,T,T(length(T)));
                    Wi2 = bsplinesecondderiv(d,i,num2,T,T(length(T)));
                    Wj2 = bsplinesecondderiv(d,j,num2,T,T(length(T)));
                    Wi3 = bsplinesecondderiv(d,i,num3,T,T(length(T)));
                    Wj3 = bsplinesecondderiv(d,j,num3,T,T(length(T)));
                    sum = sum + (5/9)*(Wi1*Wj1)^4 + (5/9)*(Wi2*Wj2)^4 ...
                        + (8/9)*(Wi3*Wj3)^4;
                end
            end
            H(i,j) = sum;
        end
    end
end
end


function [ polygons,E,F ] = polyCon2(t,x,y,n,d,T)
%t - parameter ,x - x(t), y - y(t)
%n - number of sides
%d - degree
%T - knots
polygons = cell(2*length(x), 1);
for i = 1:length(x)
    polyx = [];
    polyy = [];
    for j = 1:n
```

```
        %find the point on the circle we want
        tbar = (2*pi/n)*(j-1);
        polyx(j) = x(i) + .5*cos(tbar);
        polyy(j) = y(i) + .5*sin(tbar);
    end
    polygons{2*i-1} = polyx;
    polygons{2*i} = polyy;
end
num = length(T)-d-1;
E = zeros(n*length(t),2*num);
F = zeros(n*length(t), 1);
tcount = 0;
for i = 1:2:length(polygons)
   tcount = tcount + 1; %which polygon are we on?
   polyx = polygons{i};%xvalues for polygon
   polyy = polygons{i+1};%yvalues for polygon
   for k = 1:length(polyx)%for each point, we will have a constraint
       if(k == length(polyx))
           xi = polyx(k); xi1 = polyx(1);
           yi = polyy(k); yi1 = polyy(1);
       else
           xi = polyx(k); xi1 = polyx(k+1);
           yi = polyy(k); yi1 = polyy(k+1);
       end
         row= k + n*(tcount - 1);
         F(row) = xi*(yi1-yi) - yi*(xi1-xi);
       for col = 1:num%x-constraints
         E(row, col) = (yi1 - yi)*bspline(d,col, t(tcount), T, T(end));
       end
       for col = num+1:2*num%y-constraints
```

```
            E(row, col) = -(xi1 - xi)*bspline(d,col-num,t(tcount),T, ...
                T(end));
        end
    end
end
end


%Iterate in the paramateric case
%Sprial Example
%Assuming pentagonal constraints
clear; clc; clf; format compact; close all;
t = 0:pi/2:2*pi;
x = t.*cos(t);
y = t.*sin(t);
T = optknt(t,4);
[xcoeff,ycoeff,S] = solveParaIEQSpline(t,x,y,T);
for iter = 1:10
toremove = [];
count = 0;
for i = 1:length(S)
    if( S(i) <= 5 || S(i) >=21)
        count = count + 1;
        toremove(count) = S(i);
    end
end
S = setdiff(S, toremove);
for i = 1:length(S)
  if (S(i) >= 6 && S(i) <= 10)
      S(i) = 2;
  elseif (S(i) >= 11 && S(i) <=15)
```

```
        S(i) = 3;
    elseif( S(i) >= 16 && S(i) <=20)
        S(i) = 4;
    elseif( (S(i) >= 21 && S(i) <= 25))
        S(i) = 5;
    end
end
S = unique(S);
if (~isempty(S))
for i = 1:length(S)
    cix = x(S(i));
    ciy = y(S(i));
    ci = [cix, ciy];
    rx = 0;
    ry = 0;
    for j = 1:length(T)-3-1
        rx = rx + xcoeff(j)*bspline(3,j,x(S(i)), T, T(end));
        ry = ry + ycoeff(j)*bspline(3,j,y(S(i)), T, T(end));
    end
    r = [rx,ry];
    rprimex = 0;
    rprimey = 0;
    rdprimex = 0;
    rdprimey = 0;
    for j = 1:length(T)-3-2
        rprimex = rprimex + xcoeff(j)*bsplinederiv(3,j,x(S(i)), T, ...
            T(end));
        rprimey = rprimey + ycoeff(j)*bsplinederiv(3,j,y(S(i)), T, ...
            T(end));
        rdprimex = rdprimex + ...
            xcoeff(j)*bsplinesecondderiv(3,j,x(S(i)), T, T(end));
```

```
        rdprimey = rdprimey + ...
            ycoeff(j)*bsplineseconddderiv(3,j,y(S(i)), T, T(end));
    end
    rprime = [rprimex, rprimey];
    rdprime = [rdprimex, rdprimey];
    delta = (dot(ci-r, rprime))/(1+dot(rprime, rprime));
    t(S(i)) = t(S(i)) + delta;
end
x = t.*cos(t);
y = t.*sin(t);
figure(iter+1)
[xcoeff,ycoeff,S] = solveParaIEQSpline(t,x,y,T);
end
end
```