12-1990

Shape Preserving Piecewise Cubic Interpolation

Thomas Bradford Sprague

Western Michigan University

Follow this and additional works at: https://scholarworks.wmich.edu/dissertations

Part of the Physical Sciences and Mathematics Commons

Recommended Citation

https://scholarworks.wmich.edu/dissertations/2102

This Dissertation-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Dissertations by an authorized administrator of ScholarWorks at WMU. For more information, please contact wmu-scholarworks@wmich.edu.
SHAPE PRESERVING PIECEWISE CUBIC INTERPOLATION

by

Thomas Bradford Sprague

A Dissertation
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Doctor of Philosophy
Department of Mathematics and Statistics

Western Michigan University
Kalamazoo, Michigan
December 1990
SHAPE PRESERVING PIECEWISE CUBIC INTERPOLATION

Thomas Bradford Sprague, Ph.D.

Western Michigan University, 1990

Given a set of points in the plane, \{\( (x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n) \) \}, where \( y_i = f(x_i) \), and \( [x_0, x_n] = [a, b] \), one frequently wants to construct a function \( \hat{f} \), satisfying \( y_i = \hat{f}(x_i) \). Typically, the underlying function \( f \) is inconvenient to work with, or is completely unknown. Of course this interpolation problem usually includes some additional constraints; for instance, it is usually expected that \( ||f - \hat{f}|| \) will converge to zero as the norm of the partition of \( [a, b] \) tends to zero.

Beyond merely interpolating data, one may also require that the interpolant \( \hat{f} \) preserves other properties of the data, such as minimizing the number of changes in sign in the first derivative, the second derivative or both. An interpolation algorithm which meets, or attempts to meet, such restrictions is called shape preserving. Some progress has been made in the last decade for shape preserving piecewise polynomial interpolants for data sets that are either monotone or convex.

In this dissertation, we propose two new algorithms for comonotone interpolants and one new algorithm for constructing coconvex piecewise cubic interpolants to arbitrary (functional) data sets. Our comonotone interpolants
achieve an optimal or nearly optimal order of convergence to the underlying function $f$. While such a high rate of convergence is not obtained by the coconvex algorithm, it still improves on some earlier schemes. All of these algorithms have reasonable time complexities, being bounded by a linear function of the number of data points.

In addition to constructing and analyzing these algorithms, we demonstrate them on a few data sets. The results of these numerical experiments suggest some interesting problems for further study, and give rise to new conjectures in the theory of piecewise cubic interpolation.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Shape-preserving piecewise cubic interpolation

Sprague, Thomas Bradford, Ph.D.
Western Michigan University, 1990
ACKNOWLEDGEMENTS

I wish to thank my advisor, Dr. Dennis Pence, for sharing with me his considerable knowledge of spline functions in particular, and his enthusiasm for mathematics generally. I thank Dr. Jay Treiman for his help on many occasions with the intricacies of the T\TeX typesetting system, and for the many hours he spent constructing the macros used to prepare this dissertation. I also thank my wife, Carol, for her patience and faithful support.

Thomas Bradford Sprague
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Piecewise Polynomial Interpolation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Shape Preserving Interpolation</td>
<td>6</td>
</tr>
<tr>
<td>II. COMONOTONE INTERPOLATION ALGORITHM ONE</td>
<td>11</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Previous Results</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Some Negative Results</td>
<td>14</td>
</tr>
<tr>
<td>2.4 Definitions and Preliminary Results</td>
<td>19</td>
</tr>
<tr>
<td>2.5 Algorithm and Analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.6 Examples</td>
<td>33</td>
</tr>
<tr>
<td>III. COMONOTONE INTERPOLATION ALGORITHM TWO</td>
<td>37</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>37</td>
</tr>
<tr>
<td>3.2 Definitions and Preliminary Results</td>
<td>37</td>
</tr>
<tr>
<td>3.3 Algorithm and Analysis</td>
<td>45</td>
</tr>
<tr>
<td>3.4 Implementation Considerations</td>
<td>49</td>
</tr>
</tbody>
</table>
## Table of Contents – Continued

### CHAPTER

<table>
<thead>
<tr>
<th>IV. COCONVEX INTERPOLATION ALGORITHM</th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Introduction</td>
<td>55</td>
</tr>
<tr>
<td>4.2 Definitions and Preliminary Results</td>
<td>57</td>
</tr>
<tr>
<td>4.3 Algorithm and Analysis</td>
<td>62</td>
</tr>
<tr>
<td>4.4 Examples</td>
<td>66</td>
</tr>
</tbody>
</table>

| V. SUMMARY AND CONCLUSIONS           | 70 |

### APPENDICES

<table>
<thead>
<tr>
<th>A. MatLab Code for Comonotone Interpolation</th>
<th>74</th>
</tr>
</thead>
<tbody>
<tr>
<td>B. MatLab Code for Coconvex Interpolation</td>
<td>107</td>
</tr>
</tbody>
</table>

### BIBLIOGRAPHY                        | 121|
LIST OF FIGURES

1. Polynomial Interpolation of the Runge Function. .................. 5
2. Cubic Spline Fit to the Runge Function. .......................... 5
3. Polynomial fit to the NASA data. ........................................ 7
4. Cubic Spline Fit to the NASA data. ................................. 7
5. The Fritz-Carlson monotonicity region, $\mathcal{M}$. .............. 12
6. Comonotone and cubic spline interpolants. ......................... 35
7. Comonotone and cubic spline interpolants. ......................... 35
8. Comonotone and cubic spline interpolants. ......................... 36
9. A data set $\mathcal{D}$ and corresponding graph. ................... 42
10. Coconvex interpolant and the cubic spline. ....................... 68
11. Coconvex interpolant and the cubic spline. ....................... 68
12. Coconvex interpolant and the cubic spline. ....................... 69
13. Algorithms 2.1, 3.1, and the Cubic Spline. ....................... 71
CHAPTER I

INTRODUCTION

1.1 Piecewise Polynomial Interpolation

A common problem in scientific and engineering applications is to find an interpolating function whose graph passes through a given set of points in the plane. Ordinarily there are additional constraints on the interpolant as well. Usually, for instance, the given points, \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \), are presumed to lie on the graph of some (probably unknown) function \( f \), and our goal is that \( \hat{f} \) approximate \( f \) globally on some interval \([a, b] \), where \( x_1 = a \), and \( x_n = b \).

A particularly simple way to solve this problem is with polynomial interpolation. Namely, given \( n \) points \( (x_i, f_i) \), where \( f_i = f(x_i) \) for some function \( f \), there exists a unique polynomial of degree \( n - 1 \) passing exactly through these points. In fact, a simple formula for the error committed in approximating \( f \) by its polynomial interpolant \( p \) is given in the following theorem (Stoer and Bulirsch, 1980).

**Theorem 1.1.** Let \( f \in C^n \), then for every argument \( \bar{x} \) there exists a number \( \xi \) in the smallest interval \( I[x_1, \ldots, x_n, \bar{x}] \) which contains \( \bar{x} \) and all support
abscesses $x_i$ satisfying

$$f(\bar{x}) - p(\bar{x}) = \frac{f^{(n)}(\xi)}{n!} \prod_{i=1}^{n} (\bar{x} - x_i).$$

(1.1)

In the case of equally spaced nodes, this reduces to the particularly simple form, (Cheney and Kincaid, 1985)

$$|f(x) - p(x)| \leq \frac{1}{4n} \left( \frac{b-a}{n-1} \right)^n ||f^{(n)}||_\infty.$$

(1.2)

Why then is any additional study in interpolation needed? There are at least three good reasons for seeking other interpolation schemes. First of all, if one has several points to interpolate, then the degree of the interpolating polynomial will be high. The error formulas above assume that $f$ has at least $n$ continuous derivatives, and if the error bounds are to be reasonable, these derivatives should not be too large. Frequently this is not satisfied for large $n$. In fact, it is easy to give examples where polynomial interpolants do not converge to an underlying function even if derivatives of all orders exist. Secondly, even when the chosen polynomials approximate the underlying function well, they are typically difficult to evaluate accurately when their degree is high. That is, as the number of interpolation points increase, so does the complexity of the interpolant. Finally, polynomial interpolants tend to oscillate. This is especially undesirable when theoretical considerations indicate that the underlying function has some additional properties such as monotonicity or convexity.
Several schemes have been suggested for combating the difficulties encountered with polynomial interpolation. One of the most successful is the use of piecewise polynomials. In this approach one selects a (probably different) polynomial of bounded degree on each subinterval. The points at which the polynomials change are called knots and the data to be interpolated are called nodes. Knots and nodes are often, but need not be, the same. In piecewise polynomial interpolation, as the number of knots and nodes increase, the interpolating function remains quite simple locally. This avoids the problems associated with high degree polynomials, and retains the inherent conceptual simplicity of polynomial interpolation.

A popular choice for piecewise polynomial interpolation has been cubic spline interpolation. First studied in the 1940s by I.J. Schoenberg (Greville, 1969), they are piecewise cubic functions interpolating the given data, and chosen so as to have two continuous derivatives. Up to the setting of two additional parameters, these functions are unique, and when properly chosen give remarkably good results.

The choice of the two remaining parameters has been a matter of some debate. Two schemes have been especially popular in applications. The first of these is the so-called "natural" cubic spline. In this scheme, the spline is chosen so that its second derivative is zero at the end points of the interval \([a, b]\). The theory concerning natural cubic splines is elegant, but in practice they are less accurate than some other approaches. Natural cubic splines converge to an
underlying $C^3$ function at only a $O(\delta^2)$ rate near the end points, where $\delta$ is the norm of the partition of $[a,b]$ (deBoor, 1978). The second—and usually better—choice is the “not-a-knot” condition of deBoor(1978). Here one forces the spline to have three continuous derivatives at knots two and $n-1$. That is, the spline restricted to the first two (or last two) intervals is a single cubic polynomial. The not-a-knot method, and some others, give $O(\delta^4)$ convergence to an underlying $C^3$ function throughout $[a,b]$.

We illustrate polynomial interpolation and not-a-knot cubic spline interpolation of the Runge function, which is defined below. Let

$$f(x) = \frac{1}{1 + x^2}, \quad x \in [-5,5].$$

As illustrated in Figures 1 and 2, the cubic spline usually is a dramatic improvement over polynomial interpolation. It can be shown that in the case of the Runge function on $[-5,5]$ with equally-spaced nodes of interpolation, the error in the cubic spline interpolant decreases as $\delta^4$, while the error in polynomial interpolation becomes unbounded as the number of interpolation points increase.

The cubic spline interpolant is not without its short-comings, however. The next example (Figure 3) shows polynomial and cubic spline interpolation of a data set where the deficiencies of the cubic spline are manifest. The data (which we shall refer to as the NASA data), was collected at the Goddard Space Flight Center in 1983 (Pence, 1987). The data reflect the position of a
Figure 1. Polynomial Interpolation of the Runge Function

Figure 2. Cubic Spline Fit to the Runge Function

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
solar at time $t$; the position is the geodetic latitude $\Theta$ measured in degrees.
The polynomial interpolant to these data oscillates wildly, even though four of the five data points are nearly colinear. While the cubic spline (Figure 4) reduces the oscillations of the polynomial interpolant, it is still far from monotone. In fact, while one might desire a monotone and concave interpolant in this example, the cubic spline is neither. Thus, if shape preserving constraints are needed, one certainly must look beyond the cubic spline.

1.2 Shape Preserving Interpolation

The previous example shows that cubic spline interpolation is not guaranteed to preserve monotonicity or concavity of an underlying function. We thus are led to ask whether there are interpolation schemes that can preserve such properties. This deficiency in the cubic spline was noted early. There were some early attempts to eliminate, or at least reduce oscillations. One of the first such attempts was to model the notion of an elastic band that passes through the points of interpolation. When the band is tightened, it moves away from the spline towards a piecewise linear fit. In 1978 deBoor identifies Schweikert as the first to attempt this scheme. Schweikert obtained, on each subinterval, an interpolant that was a linear combination of $1, x, e^{pz}$, and $e^{-pz}$, with $p$ a "tension parameter." These interpolants, with a sufficiently large tension, do remove extraneous points of inflection, but are more expensive to construct and evaluate than ordinary piecewise cubics. Furthermore,
Figure 3. Polynomial fit to the NASA data.

Figure 1. Cubic Spline Fit to the NASA data.
the choice of \( p \) is not determined solely by the data, so that some amount of interactive "adjustment" may be required.

In a later approach, Akima (deBoor, 1978) constructed a cubic interpolant on each subinterval, with slopes at the knots (identical to nodes in this case),

\[
s_i = \frac{w_{i+1}f[x_{i-1}, x_i] + w_{i-1}f[t_i, t_{i+1}]}{w_{i+1} + w_{i-1}}
\]

where,

\[
w_j = |f[x_j, x_{j+1}] - f[x_{j-1}, x_j]|.
\]

The method is not completely effective, nor is it as accurate as the cubic spline, with only \( O(\delta^2) \) convergence to an underlying \( C^3 \) function.

While some other attempts were made at finding an efficient and automatic method for constructing comonotone or coconvex interpolants, the next major breakthrough seems to be in the publication of the paper by F.N. Fritsch and R.E. Carlson in 1980. In this paper, the problem was restricted to monotone interpolation to monotone data. Before we describe their results, observe that a piecewise cubic interpolant is determined on each subinterval by four parameters. Two of these parameters are given by the interpolation constraint. The other two may be specified in terms of the first or second
derivatives of the function at nodes of interpolation. We say that the piecewise cubic interpolant $s$, specified by the values of the function and its first derivative at the endpoints of each subinterval, is the piecewise cubic Hermite interpolant. Fritz and Carlson characterized monotone cubics in terms of the slopes of the interpolant at adjacent knots.

The slopes scaled by the first divided difference in the data, $(\alpha, \beta)$, must lie in a certain region in the first quadrant. This suggested a new approach to the problem: an initial cubic spline interpolant could be fit to the data, and then modified to force monotonicity by projecting $(\alpha, \beta)$ onto this region. By choosing certain subsets of this region, Fritsch and Carlson were able to construct interpolants that were not only monotone, but were "visually pleasing."

The success of the Fritsch-Carlson method inspired a series of papers, all of which abandoned the old one-pass techniques for the two-pass approach. Eisenstat, Jackson, and Lewis (1985) subsequently analyzed some of these algorithms, which they termed "fit and modify," and showed that the Fritsch-Carlson method was only third order accurate to an underlying $C^3$ monotone function. They also produced a fourth order algorithm of their own. Likewise, Yan (Beatson and Wolkowicz, 1989) improved on the accuracy of the algorithms of Fritsch and Carlson by adding knots to his interpolant. Finally, Beatson and Wolkowicz (1989) gave algorithms with the optimal order approximation property.
Before we can give a formal definition of optimal order convergence, we need some additional notation. We denote by the symbol $\omega(f, \delta)$ the *modulus of continuity* of $f$, defined by

$$\omega(f, \delta) \equiv \sup \{|f(x_1) - f(x_2)| : x_1, x_2 \in [a, b], |x_1 - x_2| \leq \delta\}$$

A piecewise cubic interpolation algorithm has the optimal order approximation property if the error in approximating an underlying function $f \in C^j$ with the piecewise cubic interpolant $s$ satisfies

$$\|f - s\| \leq K\delta^j \omega(f^{(j)}, \delta), \quad 1 \leq j \leq 3, \; 0 \leq l \leq 1,$$

where $j$ and $l$ are integers.

Beatson and Wolkowicz (1989) achieved this accuracy by using a slightly larger monotonicity region than that used in the Fritsch-Carlson method, and adding at most one knot to the interior of each interval to recover monotonicity. They also illustrated the construction of "visually pleasing" interpolants of optimal order, by projecting the endpoint slopes slightly into the interior of the monotonicity region.

It should be emphasized that all of this activity was limited to monotone data. The techniques used for monotone data do not apply to the more general situation. It is to the general problem that we now turn our attention.
CHAPTER II

COMONOTONE INTERPOLATION ALGORITHM ONE

2.1 Introduction

In the following two chapters we shall consider two new algorithms for constructing comonotone piecewise cubic interpolants. Our goal is to construct interpolants which are comonotone with the data in the sense that the number of changes in sign of the first derivative of the interpolant should not exceed the number of changes in sign in the sequence of first divided differences of the data.

2.2 Previous Results

The algorithm developed below depends heavily on several previous results of Fritsch and Carlson (1980), Beatson (1986), and Beatson and Wolkowicz (1989).

**Lemma 2.1.** Let $p$ be a cubic polynomial on $[a, b]$, $a < b$, and let $p[a, b] = \frac{p(b) - p(a)}{b - a}$. Then necessary and sufficient conditions for $p$ to be monotone on $[a, b]$, are:

(a) If $p[a, b] = 0$, then $p' \equiv 0$ on $[a, b]$. 

11
(b) If $p[a,b] \neq 0$, and $(\alpha, \beta) = \frac{1}{p[a,b]}(p'(a), p'(b))$, then $(\alpha, \beta) \in \mathcal{M}$, (see Figure 5) where

$$\mathcal{M} = \{(\alpha, \beta) | 0 \leq \alpha \leq 3 \text{ and } 0 \leq \beta \leq 3\} \cup \{(\alpha, \beta) | \phi(\alpha, \beta) \leq 0\},$$

and,

$$\phi(\alpha, \beta) \equiv (\alpha - 1)^2 + (\alpha - 1)(\beta - 1) + (\beta - 1)^2 - 3(\alpha + \beta - 2).$$

![Figure 5. The Fritz-Carlson monotonicity region, $\mathcal{M}$.](image-url)
In the algorithms below, we shall need to produce optimal order estimates for the derivatives of a function $f$ on $[a, b]$. The following lemma, part of a more general result due to Beatson (1986), shows that if $f$ has at least two continuous derivatives, this may be accomplished with a cubic spline function.

**Lemma 2.2.** Let $a = t_0 < t_1 < \cdots < t_n = b$, and let $f \in C^j[a, b]$, $j = 2$, or 3. Let $s \in C^2$ be the cubic spline interpolant determined by the not-a-knot end condition of deBoor, or by putting $s^{(i)}(t_m) = \text{the corresponding values of local fourth order interpolants}$ for $l$ fixed at 1 or 2, and $m = 0$, and $n$. Then, if $f \in C^j[a, b]$ for $j = 2$ or 3, where $n \geq 3$ ($n \geq 5$ for the not-a-knot condition), there exists a constant $C$ so that

\[
\| (f - s)^{(k)} \|_{[t_i, t_{i+1}]} \leq C h_i^{2-k} \delta^{j-2} \omega(f^{(j)}, \delta), \quad 0 \leq i \leq n - 1, \quad k = 0, 1, 2.
\]

**Lemma 2.3.** Let $p$ be a polynomial of order 4 interpolating $f \in C^j[c, d]$ at $c$ and $d$, and satisfying

\[
|p'(x) - f'(x)| \leq \epsilon, \quad \text{for } x = c, d.
\]

Then,

\[
\|f^{(l)} - p^{(l)}\|_{[c, d]} \leq \frac{7}{2^{j-l+1}} (d-c)^{j-l} \omega(f^{(j)}, \delta) + \epsilon \left( \frac{d-c}{2} \right)^{1-l}, \quad l = 0, 1.
\]
In particular, Lemma 2.3 shows that if one has an optimal order algorithm to produce estimates of $f'$ at nodes of interpolation, then the piecewise cubic Hermite interpolant constructed from these slopes must satisfy (2.1).

**Lemma 2.4** (Beatson and Wolkowicz, 1989). Let $k \geq 3$. Let a mesh $t : t_0 < t_1 < \cdots < t_n$ be given with mesh size $\delta = \max\{t_{i+1} - t_i \mid 0 \leq i \leq n - 1\}$. Let a knot set $r : r_0 \leq r_1 \leq \cdots \leq r_m$, a superset of $t$ with $r_{i+k-2} > r_i$ for $0 \leq i \leq m - k + 2$, be given. Suppose that each interval $[t_i, t_{i+1}]$ contains at least $k$ knots $r_i$. Let $S_{r,k}$ be the set of piecewise polynomials of order $k$ on the knot set $r$, such that $s \in S_{r,k}$ has $k - l - 1$ continuous derivatives at $x_i$ when $r$ contains exactly $l$ knots $r_j$ equal to $x_i$, for some $l$ satisfying $0 \leq l \leq k$. Let $1 \leq j \leq k - 1$. Then for each monotone $f \in C^1[t_0, t_n]$, there exists a piecewise polynomial function $s \in S_{r,k} \subset C^1[t_0, t_n]$, such that

(a) $s(t_i) = f(t_i)$, $0 \leq i \leq n$;

(b) $s$ is monotone on $[t_0, t_n]$;

(c) $||f^{(l)} - s^{(l)}||_{[t_0, t_n]} \leq C_3 \delta^{k-l} \omega(f^{(j)}, \delta)$, $l = 0, 1$.

Furthermore, if $g$ is a $(C^2)$ cubic spline function satisfying the hypotheses of Lemma 2.2, then there exists an algorithm to construct such interpolants $s$ so that $|s'(x_i)| \leq |g'(x_i)|$ for each $i$.

**2.3 Some Negative Results**

Before attempting to generalize previous results to non-monotone data,
we first note a few negative results.

We begin by considering the implications of Lemma 2.1. This lemma requires that the derivative estimates used by the interpolant be "close" to the first divided difference in the data. This seems to be a trivial requirement, but is actually rather severe. It is true that the first divided difference converges to the derivative of the underlying function, but this result concerns the difference in these quantities, while monotonicity demands a small ratio. Mere convergence—in fact optimal order convergence—cannot guarantee even a bounded ratio. We formalize these remarks in the following proposition.

**Proposition 2.1.** For every \( j \geq 1 \), \( \exists f \in C^j \), an almost uniform sequence of partitions \( x^{(n)} \) of \([-1,1]\), and \( a_n, b_n \in x^{(n)} \) satisfying

\[
\left| \frac{f'(b_n)}{f[a_n,b_n]} \right| \rightarrow \infty.
\]

**Proof.** Define

\[
y^{(n)} := \left\{ \pm y_i : y_i = \frac{2i + 1}{n}, i = 0, 1, 2, \ldots, \left\lfloor \frac{n - 1}{2} \right\rfloor \right\},
\]

and let \( x^{(n)} \) be obtained from \( y^{(n)} \) by replacing \(-\frac{1}{n}\) with \(\frac{1}{n^2} - \frac{1}{n}\). Then \( x^{(n)} \) is almost uniform in the sense that

\[
\frac{||y^{(n)} - x^{(n)}||}{||y^{(n)}||} \rightarrow 0.
\]
Define
\[ f(x) := e^{x^2}, \]
and
\[ [a_n, b_n] := \left[ \left( \frac{1}{n^2} - \frac{1}{n} \right), \frac{1}{n} \right]. \]
Then
\[ f \in C^\infty \subseteq C^j, \]
and
\[
\lim_{n \to \infty} \left| \frac{f'(b_n)}{f[a_n, b_n]} \right| = \lim_{n \to \infty} \left| \frac{2 e^{1/n^2} \left( \frac{2}{n} - \frac{1}{n^2} \right)}{e^{1/n^2} - e^{(1/n-1/n^2)^2}} \right| \\
= \lim_{n \to \infty} \left| \frac{2 \left( \frac{2}{n^2} - \frac{1}{n^3} \right)}{1 - e^{(1/n^3-2/n^3)}} \right| \\
= \lim_{n \to \infty} \left| \frac{2 \left( \frac{-4}{n^3} + \frac{3}{n^4} \right)}{1 - e^{(1/n^3-2/n^3)}} \right| \\
= \lim_{n \to \infty} \frac{8 - \frac{6}{n}}{\frac{6 - \frac{4}{n^2}}{e^{(1/n^4-2/n^3)}}} \\
= \infty. \]

Before moving on, we note that in the proof above, \( f(x) = x^2 \) with the same partition will also work. Thus a corresponding statement is true whether or not we restrict to polynomials, or exclude them. Proposition 2.1 shows that even exact values of the derivative may fail to produce monotone interpolants, and leads to the following conclusion.

**Corollary 2.1.** For every \( k, j \geq 1 \), \( \exists f \in C^j \) and a sequence of \( k^{th} \) order estimates to \( f' \) which do not induce optimal order comonotone piecewise cubic interpolants.
While the previous proposition shows that derivative estimates of the proper magnitude may not be easy to construct, even more is true. It will not be sufficient to "correct" slope estimates in an arbitrary manner, if this will affect decisions concerning the monotonicity of the interpolant. For while the magnitude of the derivative of the underlying function $f$ is especially difficult to estimate near the critical points of $f$, it is precisely there that a good estimate is most needed. For the sign of $f$ is needed near its critical points to determine where $f$ is monotone. While the previous result leads us to believe that estimates leading to correct judgments concerning monotonicity may be difficult to obtain, the next result shows that errors in enforcing monotonicity can be disastrous for the accuracy of interpolation.

**Proposition 2.2.** Let $A$ be an algorithm to construct interpolants $s$ to underlying functions $f \in C^j$, $j \geq 1$. If for every non-monotone $f \in C^j$, and each sequence of partitions $x^{(n)}$, $A$ improperly enforces monotonicity in $s$ on at least one interval induced by each partition, then $A$ is at best second order accurate.

**Proof.** Let

$$f(x) = 1 - x^2 \in C^\infty \subseteq C^j, \quad x \in [-1, 1].$$

Define

$$y^{(n)} := \left\{ \pm y_i : y_i = \frac{2i + 1}{n}, i = 0, 1, 2, \ldots, \left\lfloor \frac{n-1}{2} \right\rfloor \right\},$$
and let $x^{(n)}$ be obtained from $y^{(n)}$ by replacing $-\frac{1}{n}$ with $-\frac{1}{n} - \frac{1}{n^2}$. Note that

$$
\delta = -\frac{1}{n} - \frac{1}{n^2} - \left(\frac{-3}{n}\right) = \frac{2}{n} - \frac{1}{n^2}.
$$

Now let

$$[a_n, b_n] = \left[-\left(\frac{1}{n} + \frac{1}{n^2}\right), \frac{1}{n}\right],$$

where $a_n, b_n \in x^{(n)}$.

Note that $[a_n, b_n]$ is the only interval in $y^{(n)}$ where monotonicity can be enforced incorrectly. Thus the algorithm must choose a non-decreasing interpolant $s$ on $[a_n, b_n]$. Now define

$$s^*(x) := \inf\{f(1/n), f(x)\}, \ x \in [a_n, b_n].$$

Clearly, of all monotone interpolants to $f$ on $[a_n, b_n]$, $s^*$ has minimum error $||f - s||$. Thus for any monotone interpolant $s$

$$||f - s|| \geq ||f - s^*||$$

$$= f(0) - f\left(\frac{1}{n}\right) = 1 - 1 + \frac{1}{n^2} = \left(\frac{1}{2 - 1/n}\right)^2 \delta^2 = O(\delta^2).$$
Now the example above is not in itself especially interesting. It is, after all, completely trivial to find a comonotone interpolant to \( f \). Indeed, any scheme—say a cubic spline—that reproduces quadratics will suffice. But it does serve to point out that caution must be exercised in constructing comonotone interpolants. If the worst cases are not avoided, the attempt to achieve comonotonicity by enforcing monotonicity on selected intervals will severely limit accuracy. It is this type of error that our algorithms must avoid.

2.4 Definitions and Preliminary Results

In order to apply earlier results to non-monotone data, they must first be generalized. This will be made easier with a few preliminary definitions. We denote the first divided differences in the data with the symbol \( d(i, i+1) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \).

**Definition 2.1.** Let \( D = \{ P_i \mid P_i = (x_i, y_i); \ i = 0, 1, \ldots, n; \ a = x_0 < x_1 < \cdots < x_n = b; \ f(x_i) = y_i \} \) be a set of \( n + 1 \) data points. We say that \( D \) is direction determining for \( f \) on \([a, b]\) if the number of changes in sign of \( f' \) over \([a, b]\) and the number of changes in sign in the sequence of first divided differences \( d(i, i + 1) \) are equal.

We shall also need to distinguish between the critical points of the underlying function \( f \), and the critical points of a given data set. We call a point \((x, y)\) a **critical point of** \( f \) if \( y = f(x) \), and either \( f'(x) = 0 \) or \( f'(x) \) does...
not exist. We define a critical data point below.

**Definition 2.2.** Let \( D = \{ P_i \mid P_i = (x_i, y_i); \ i = 0, 1, \ldots, n; \ f(x_i) = y_i \} \).

The critical data points of \( D \) are those satisfying either

(a) \( P_i \), if \( y_i < \min \{y_{i-1}, y_{i+1}\} \) or \( y_i > \max \{y_{i-1}, y_{i+1}\} \) where \( 0 < i < n \),

or

(b) \( P_i \) and \( P_{i+j} \) if \( \exists \ j \geq 1 \) and \( y_i = y_{i+1} = \cdots = y_{i+j} \) and either \( y_i < \min \{y_{i-1}, y_{i+j+1}\} \) or \( y_i > \max \{y_{i-1}, y_{i+j+1}\} \) where \( 0 < i < i + j < n \).

Let \( M = (P_i)_{i=l}^{r} \subseteq D, \ l \leq r \). Then \( M \) is a monotone sequence if \( (y_i)_{i=l}^{r} \) is monotone. Horizontal (or constant), increasing, and decreasing sequences are defined analogously. The length of \( M \) is \( \mathcal{L}(M) = r - l + 1 \). If \( \mathcal{L}(m) \geq 3 \), then the interior of \( M \) is \( M^\circ = (P_i)_{i=l+1}^{r-1} \), while if \( \mathcal{L}(m) < 3 \), then the interior is empty. The interval induced by \( M \) is \( I(M) = [x_l, x_r] \). \( M \) is trivial if \( r = l \) and non-trivial otherwise. Finally, a nontrivial horizontal sequence, \( M = (P_i)_{i=l}^{r} \), is called critical-horizontal if \( P_l \) and \( P_r \) are critical data points satisfying part (b) of Definition 2.2. We refine the idea of a monotone sequence into monotone sequences induced by the first divided differences of the data, and those induced by considering derivative approximations as well. Intuitively, the monotone sequences induced by the first divided differences are a partitioning of the subintervals into maximal sequences of adjacent subintervals on which the underlying function \( f \) is known from the data alone to be monotone (when the data set is direction determining). On the other hand, if we know the sign of the derivative of \( f \) at each critical data point, we...
might be able to extend some of the difference-induced monotone sequences by adding a critical data point on its left or right. Given a set of approximations, \( G = \{ g_i \mid i = 0, 1, \ldots, n \} \), to \( f'(x_i), i = 0, 1, \ldots, n \), we may proceed on the assumption that these approximations are accurate enough to determine the signs of the \( f'(x_i) \). If, proceeding under this assumption, we extend the difference-induced monotone sequences wherever possible (arbitrarily assigning critical data points where \( g_i = 0 \)), we obtain a new partitioning of the subintervals: the \( G \)-induced monotone sequences. We formalize these definitions below.

**Definition 2.3.** Let \( D = \{ P_i \mid P_i = (x_i, y_i); i = 0, 1, \ldots, n; a = x_0 < x_1 < \cdots < x_n = b; f(x_i) = y_i \} \) be given. The difference-induced monotone sequences of \( D \) are those obtained by partitioning the sequence \( (P_i)_{i=0}^n \) into monotone sequences, \( M_j \), satisfying the following three requirements:

1. If \( C \) is a critical-horizontal sequence, then \( C = M_j \) for some \( j \).
2. For every \( i < k \), where \( (P_i, P_{i+1}, \ldots, P_k) \) is in the interior of some monotone sequence, then \( (P_i, P_{i+1}, \ldots, P_k) \subseteq M_j \) for some \( j \).
3. If \( P_i \) is a critical data point, but not in any critical-horizontal sequence, then \( (P_i) = M_j \) for some \( j \).

Given such a partitioning \( Q \), we denote the sequence containing the data point \( P \) by \([P]_Q\), or simply \([P]\), when the partition is clear from the context.

**Proposition 2.3.** For every data set \( D \), there is a partitioning of \( (P_i)_{i=0}^n \) satisfying Definition 2.3.
PROOF. Let \((P_i)_{i=0}^n\) be given. We construct a partitioning. Begin by adding two false data points \(P_{-1}\) and \(P_{n+1}\) so that \(f[x_{-1}, x_0] = f[x_0, x_1]\) and \(f[x_{n-1}, x_n] = f[x_n, x_{n+1}]\). Then for each critical data point \(P_j\), and for \(j = -1, n + 1\), put \([P_j] = (P_i)_{i=j}\), a trivial sequence. Let \((P_j)\) and \((P_j+k)\) be two consecutive trivial sequences derived in this manner (since \((P_{-1})\) and \((P_{n+1})\) are two candidates, at least two such sequences must exist). Now for any data point, \(P_{j+l}\), between these sequences \(P_{j+l}\) is contained in the (necessarily monotone) sequence \(M = (P_i)_{i=j}\). If \(M\) is critical-horizontal, discard the trivial sequences \([P_j]\) and \([P_{j+k}]\), and select \(M\) as a sequence in the partition. Otherwise, select \((P_j), (P_{j+k})\), and \(M^o\) as sequences for the partition. Finally, discard the trivial sequences \([P_{-1}]\) and \([P_{n+1}]\).

To see that this partitioning produces monotone sequences we merely observe that each \([P_i]\) is either a critical-horizontal sequence, a single critical data point, or contains no critical data points at all. That conditions (1), (2), and (3) of the definition are met is obvious from the construction. 

PROPOSITION 2.4. The partitioning of the data set \(D\) as above into difference-induced monotone sequences is unique.

PROOF. Let \(\mathcal{P}\) and \(\mathcal{Q}\) be two partitionings of \(D\) into difference-induced monotone sequences satisfying Definition 2.3. Now either \(\mathcal{P}\) and \(\mathcal{Q}\) are identical, or (perhaps by switching the roles of \(\mathcal{P}\) and \(\mathcal{Q}\)) there is a data point \(S\) satisfying

\[
(2.4) \quad \exists T \in [S]_\mathcal{P} - [S]_\mathcal{Q}.
\]
Suppose that $S$ is a critical data point. Then by requirement (1) of Definition 2.3, if $S$ is an endpoint of a critical-horizontal sequence, $[S]_{\mathcal{P}} = [S]_{\mathcal{Q}}$, contrary to our assumption. Hence, $S$ is not an endpoint of a critical-horizontal sequence. Then by (3), $[S]_{\mathcal{P}} = [S]_{\mathcal{Q}}$, again a contradiction. We conclude that $S$ is not critical.

Suppose that $S$ is an endpoint, say $S = P_1$. Then there are integers $i$ and $j$ so that $i > j$ and $[S]_{\mathcal{P}} = (P_1, P_2, \ldots, P_i)$, and $[S]_{\mathcal{Q}} = (P_1, P_2, \ldots, P_j)$. But then $(P_2, \ldots, P_i)$ is in the interior of some monotone sequence, but not a subset of $[P_1]_{\mathcal{Q}}$, contradicting (2). An analogous argument can be given in case $S = P_n$, so we conclude that $S$ is not an endpoint.

We see from the above arguments that $S$ is neither critical nor an endpoint. Applying (2) to $Q$, we conclude that $S$ and $T$ cannot be in a common interior sequence. Thus there is a critical data point $V$ between $S$ and $T$. But then any sequence containing both $S$ and $T$ cannot be monotone, so that $[S]_{\mathcal{P}}$ is not monotone, a contradiction. Since any data point $S$ is certainly an endpoint, a critical data point, or in the interior of some monotone sequence, we conclude that the difference-induced monotone sequences are unique.

From propositions 2.3 and 2.4 and their proofs we obtain the following useful corollary.

**Corollary 2.2.** Let $M$ be a difference-induced monotone sequence. Then exactly one of the following must hold:

(a) $M$ consists of exactly one critical data point.
(b) $M = N^o$ for some monotone sequence $N$ of maximal length.

(c) $M$ contains an endpoint and possibly also the interior of some monotone sequence of maximal length.

(d) $M$ is a critical-horizontal sequence.

In the case that (a) holds, we call $M$ a critical singleton, $M$ is called an interior sequence, or an end sequence in the next two cases, respectively. $M$ is called a critical sequence if it satisfies either (a) or (d).

**Definition 2.4.** Let $D = \{P_i \mid P_i = (x_i, y_i); i = 0, 1, \ldots, n; a = x_0 < x_1 < \cdots < x_n = b; f(x_i) = y_i\}$ and slope approximations $G = \{g_i \mid i = 0, 1, \ldots, n\}$ be given. We say that $G$ is consistent with $D$, or that $G$ is $D$-consistent when the following conditions are met:

(a) If $M = (P_i)_{i=l}^r$ is an interior or end sequence, then

$$g_i \cdot (y_r - y_l) \geq 0 \quad \text{for} \quad l \leq i \leq r \quad \text{where} \quad l < r,$$

$$g_l \cdot (y_{l+1} - y_l) \geq 0 \quad \text{where} \quad l = r < n, \quad \text{or},$$

$$g_l \cdot (y_{l-1} - y_l) \geq 0 \quad \text{where} \quad l = r = n.$$

(b) If $M = (P_i)_{i=l}^{l+1}$ is a critical-horizontal sequence, then

$$g_l \cdot d(l-1, l) \geq 0, \quad \text{and} \quad g_{l+1} \cdot d(l+1, l+2) \geq 0,$$

(c) If $M = (P_i)_{i=l}^r$ is a critical-horizontal sequence of length at least three, then

$$g_i = 0, \quad l \leq i \leq r.$$
Hence, for each \( g_i \) we prescribe the sign of \( g_i \), unless \( P_i \) is a critical singleton.

Let \( D \) be a data set for \( f \) on \([a, b]\), and let \( G = \{g_i \mid i = 0, 1, \ldots, n\} \) be \( D \)-consistent slope approximations. Starting from the difference induced monotone sequences, we can construct a new set of sequences in the following way. We first split the critical-horizontal sequences of length two into two critical singletons. Then we concatenate each critical singleton with one of the sequences on its left or right, if this can be done so that the signs of the first divided differences and the signs of the slope estimates \( g_i \) all agree in sign in each of the resulting monotone sequences. In the paragraph below, we shall define such a set of sequences as \( G \)-induced monotone sequences. Observe that if for every critical point \( P_i, g_i \neq 0 \), then these \( G \)-induced monotone sequences are unique. If the collection of sequences defined below is not unique, any of them will suit our purposes in the sequel.

**Definition 2.5.** Let \( D = \{P_i \mid P_i = (x_i, y_i); \ i = 0, 1, \ldots, n; f(x_i) = y_i; \ a = x_0 < x_1 < \cdots < x_n = b \} \) and slope approximations, \( G = \{g_i \mid i = 0, 1, \ldots, n\} \), consistent with \( D \) be given. The \( G \)-induced monotone sequences \( M_j \) of \( D \) are those sequences satisfying

(a) Each difference-induced monotone sequence, except a critical-horizontal sequence of length two, is a subset of some \( G \)-induced monotone sequence.

(b) If \((P_i, P_{i+1})\) is a critical-horizontal sequence of length two, then \( P_i \) and \( P_{i+1} \) are in different \( G \)-induced monotone sequences.
(c) If $P_i$ and $P_{i+1}$ are in a common $G$-induced monotone sequence, then the product of any two of $g_i, g_{i+1},$ and $d(i, i + 1)$ is non-negative.

(d) If $M_j = (P_i)$ is a trivial $G$-induced monotone sequence, then concatenating $M_j$ with the sequence on its left or right (i.e. $M_{j-1}$ or $M_{j+1}$) results in a sequence that violates either (b) or (c) above.

In our analysis of the first algorithm, we shall obtain a weaker result than optimal order approximation. We will however produce approximations that are "asymptotic to" optimal order interpolants, in a sense made precise below.

**DEFINITION 2.6.** A $(J+1)^{st}$ order piecewise polynomial interpolation algorithm $A$ is said to have the almost optimal order approximation property, and both $A$ and the interpolants $h$ it produces are called almost optimal order if there is a constant $K$ such that for every sequence of finite partitions $x^{(n)}$ of $[a, b]$, where each is a refinement of the previous, and $||x^{(n)}|| \to 0$, and for every $f \in C^j[a, b]$, for some $1 \leq j \leq J$, the set of subintervals

$$F(n) = \{ [x_i^{(n)}, x_{i+1}^{(n)}] : ||f^{(l)} - h^{(l)}|| > K \delta^{j-l} \omega(f^{(j)}, \delta), l = 0, 1\}$$

satisfy the following.

(a) The sequence $|F(1)|, |F(2)|, \ldots$ is bounded,

(b) $G(1) \supseteq G(2) \supseteq G(3) \supseteq \ldots$, where

$$G(m) = \{ x \mid \exists [a, b] \in F(m) \exists x \in [a, b] \} , \text{ and}$$
(c) $\mu_m = \sum_{[c,d] \in F(m)}(d - c) \to 0$ as $m \to \infty$.

2.5 Algorithm and Analysis

We now present the first algorithm for computing comonotone piecewise cubic interpolants.

ALGORITHM 2.1. Let $D$ and $G$ be given, where $D = \{P_i \mid P_i = (x_i, y_i); \ i = 0,1,\ldots,n\}$, where $f(x_i) = y_i$, $a = x_0 < x_1 < \cdots < x_n = b$, and where $G = \{g_i \mid i = 0,1,\ldots,n\}$ are slope approximations. Furthermore, assume that there exists an algorithm to construct monotone piecewise cubic interpolants to monotone data. Compute a comonotone interpolant to $D$ as follows.

(a) If $G$ is not $D$-consistent, then replace $g_i$ by $-g_i$ wherever necessary to achieve consistency.

(b) Partition $D$ into $G$-induced monotone sequences.

(c) To every $G$-induced monotone sequence $M$ that is neither a horizontal-critical sequence nor a trivial sequence, construct a monotone interpolant.

(d) For each horizontal-critical $G$-induced monotone sequence $M$, interpolate $M$ with the appropriate constant function.

(e) On any remaining subintervals $[x_i, x_{i+1}]$ where $g_i, g_{i+1}$, and $f[x_i, x_{i+1}]$ agree in sign, construct a monotone interpolant to $P_i, P_{i+1}$.

(f) On any remaining subintervals $[x_i, x_{i+1}]$ where $g_i, g_{i+1}$, and $f[x_i, x_{i+1}]$...
do not all agree in sign, interpolate \( P_i, P_{i+1} \) with a cubic hermite interpolant.

Before analyzing the algorithm, we make a few comments. For each of the steps (a) through (e) above, it is possible that one or more derivative values \( g_i \) may be modified. In this case, it is intended that all following steps will proceed with the modified values. Secondly, it might be observed that in step (d) every horizontal-critical sequence is necessarily of length at least three. Otherwise, it would have been split at step (b). In step (e) and in the discussion below, we say that the sign of two quantities agree if their product is non-negative. Three or more quantities agree in sign if they do so pairwise.

Finally, observe that the interpolants \( s \) produced by Algorithm 2.1 may have two different types of local extrema. We may of course have the case of a single point \((x^*, y^*)\), where \( s(x^*) > s(x) \) (or \( s(x^*) < s(x) \)) for all \( x \) is some open interval containing \( x^* \). But in the case of a critical—or more, we will have a whole segment of extreme values, \( \{(x, k) \mid x_l \leq x \leq x_r\} \). We shall use the term local extreme sets to refer to both of these possibilities. In analogy to the cases with single points, we use the terms relative minimum sets and relative maximum sets in the obvious way. Note that we have not included the cases of end points or end sequences in this term. We now proceed to the analysis of the algorithm.

**Theorem 2.1.** Let \( s \) be the interpolant produced by Algorithm 2.1 to a data set \( D \). Then the number of local extreme sets of \( s \) on \((a, b)\) is the same as the
number of critical sequences in $D$.

**Proof.** Consider a nontrivial monotone sequence $M = (P_i)_{i=1}^r$ containing no critical data point and not part of a critical-horizontal sequence. By step (c) of Algorithm 2.1, $s$ is monotone on the interval $I = [x_i, x_r]$. Since $M$ is not part of a critical-horizontal sequence, the graph of $s$ on $I$ has no intersection with the extreme sets of $s$ on $(a, b)$. Thus we need only enumerate the number of extreme sets in regions near a critical sequence.

Consider the case of a critical singleton $P_i$. To simplify the discussion, we shall assume that $P_i$ is a relative minimum of $D$. We can easily modify the proof here for the case of a relative maximum. Now neither $P_{i-1}$ nor $P_{i+1}$ can be relative minima, since $P_i$ is a critical singleton. Since $s$ is continuous on $I = [x_{i-1}, x_{i+1}]$, it must have at least one relative minimum on $I$. The fact that $y_i$ is strictly less than both $y_{i-1}$ and $y_{i+1}$ forces $s$ to have a minimum on the interior of $I$. Suppose $s$ has at least two distinct relative minimum sets on the interior of $I$. Observe that if $s$ is not monotone on a subinterval $J = [x_j, x_{j+1}]$, then no knot will be added on $J$, and $s$ will be a single polynomial of order four there. Since a polynomial of order four can have at most two changes in the sign of its first derivative, two disjoint relative minimum sets cannot exist on a single subinterval $J$. Thus on both of the open intervals $I_l = (x_{i-1}, x_i)$ and $I_r = (x_i, x_{i+1})$, $s$ has a relative minimum set (is therefore not monotone) and has no interior knots. We show that this conclusion leads to a contradiction.

Suppose that $g_{i-1} > 0$. Since $s$ has a minimum on the open interval $I_l$,
and $s'$ can have no more than two zeros on $[x_{i-1}, x_i]$, $g_i \geq 0$. If $g_{i+1} \geq 0$, then $g_i, g_{i+1}$, and $d(i, i + 1)$ agree in sign, and $s$ is monotone on $I_r$, contradicting the assumption of a minimum on $I_r$. Hence, $g_{i+1} \geq 0$. Since $s'$ can have no more than two zeros on $I_r$, $s$ has a maximum but no minimum on $I_r$—a contradiction.

Since $g_{i-1}$ cannot be positive, we have that $g_{i-1} \leq 0$. Again by counting zeros in $s'$, we see that $g_i \geq 0$. This, once again prevents $s$ from having a minimum in $I_r$. Hence it is impossible for $s$ to have two distinct minimum sets in $[x_{i-1}, x_{i+1}]$. Finally, we note that if $s$ has a local maximum in $I_i$, it must be that $g_{i-1} > 0$. Since Algorithm 2.1 adjusts $G$ to be $D$-consistent, this would imply that $P_{i-1}$ is a local maximum of $D$. This maximum set would be detected in the analysis of critical sequences corresponding to local maxima of $D$; the same is true of any local maxima on $I_r$. We conclude that each critical singleton induces exactly one relative extreme point in $s$.

Now consider $M = (P_i)_{i=L}^R$, a critical horizontal sequence of length at least three. Again assume that this sequence corresponds to a local minimum value. We apply the same type of analysis as above, but now to the non-adjacent intervals $[x_{i-1}, x_i]$, and $[x_r, x_{r+1}]$. Noting that $s$ is constant on $[x_i, x_r]$, so that $g_i = g_r = 0$, we conclude that $s$ has exactly one local minimum set on $[x_{i-1}, x_{i+1}]$, and no local maxima unless $P_{i-1}$, or $P_{r+1}$ is a local maximum of $D$.

For the third and final case, consider a critical horizontal sequence of
length two, \( M = (P_i, P_{i+1}) \). Since the sequence is horizontal, \( d(i, i + 1) = 0 \).

Assume again that \( M \) corresponds to a local minimum. By the definition of \( G \)-induced monotone sequence, \( g_i \leq 0 \) and \( g_{i+1} \geq 0 \). This forces the fourth order polynomial \( s \) to have a single local minimum in \((x_{i-1}, x_{i+2})\) and no local maximum, unless at least one of \( P_{i-1} \) or \( P_{i+2} \) is relative maximum.

The three cases above, and the corresponding arguments for local maxima, show that \( s \) has exactly one local extreme set for each critical sequence of \( D \).

**Corollary 2.3.** If \( D \) is direction determining for \( f \) on \([a, b]\), then \( f \) and \( s \) have the same number of local extreme sets on \([a, b]\).

**Theorem 2.2.** Consider the collection of functions \( f \in C^j[a, b] \), where \( j \geq 1 \), and with finitely many changes in the sign of \( f' \) on \([a, b]\). For each such function \( f \), define \( \Delta_f = \min |\alpha - \beta| \), where \( \alpha \) and \( \beta \) are chosen so that \((\alpha, \gamma)\) and \((\beta, \delta)\) are in distinct relative extreme sets, or so that \( \gamma = \delta \) and \( \{(\zeta, \gamma) : \alpha \leq z \leq \beta, \alpha < \beta\} \) is a single relative extreme set of \( f \) on \([a, b]\). Then Algorithm 2.1 restricted to functions, and to data sets such that \( \delta < \Delta_f/6 \), has the almost optimal order approximation property, if the algorithms used to compute the derivative estimates \( g_i \) and monotone interpolants are of optimal order.

**Proof.** We first note that if the \( g_i \) provided to Algorithm 2.1 are of optimal order, then they remain so after step (a). For if the sign of any \( g_i \) is changed there, it must have been in error by at least \(|g_i|\). Thus multiplying it by \(-1\)
when it can be inferred from $D$ that its sign is incorrect, can only serve to reduce error. Let $x^{(n)}, n = 1, 2, \ldots$ be a sequence of partitions of $[a, b]$, each a refinement of the previous, such that $\delta_n = \|x^{(n)}\|$ converges to zero, and $\delta_1 < \Delta f/6$. Then the data sets $D_n$ induced by $x^{(n)}$ are not only direction determining, but the critical data points of $D_n$ are separated by at least four non-critical data points.

Since the algorithm used to construct monotone sequences is optimal order, the initial interpolants to interior sequences converge at the necessary rate. $D$ is direction determining, so the location of the relative extreme sets of the underlying function is determined to within two subintervals. Consider, for instance, the $i^{\text{th}}$ critical data point $x_k \in D_n$. We know that $f$ has a local extreme point (or endpoint of a local extreme segment) in the interval $[x_{k-1}, x_{k+1}]$. Successive refinements of $D_n$ indicate more precisely the location of the local extreme sets of $f$. Thus the intervals over which the desired rate of convergence has not been guaranteed will always remain subsets of these original "critical intervals." Algorithm 2.1 enforces monotonicity on one of $[x_{k-1}, x_k]$ or $[x_k, x_{k+1}]$, possibly modifying the interpolant on adjacent intervals in the process. Since we have not guaranteed that $f$ is monotone on the interval chosen, the convergence on $[x_{k-2}, x_{k+1}]$ or $[x_{k-1}, x_{k+2}]$ may be less than optimal. But with the exception of the intervals $[a_i^n, b_i^n] = [x_{k-2}^{(n)}, x_{k+2}^{(n)}]$, we have shown that (1.7) holds. It remains only to show that the sets on which we have not guaranteed optimal order convergence are reducing in measure to
zero. In the following computation, recall that the number of relative extreme sets \( m \) of \( f \) on \([a, b]\) is fixed. Let

\[
\mathcal{F}(n) = \bigcup_{i=1}^{m} [a_i^{(n)}, b_i^{(n)}],
\]

where there are \( m \) critical points of \( f \) on \([a, b]\). Then,

\[
\mu_n = \sum_{i=1}^{m} \left( b_i^{(n)} - a_i^{(n)} \right) \\
\leq 4m \delta_n \\
\rightarrow 0, \text{ as } n \rightarrow \infty.
\]

2.6 Examples

In this section we present three examples in Figures 6-8. We wish to draw special attention to certain aspects of the output of our algorithm. The first is that it avoids the oscillations in the cubic spline. The graph of the cubic spline with the not-a-knot end condition is drawn on the same axis with a broken line for comparison. Secondly, the comonotone interpolant may be much more stable in regions near—but outside—the interval of interpolation. While we make no claims of its usefulness for extrapolation purposes, it may be that this stability presents less risk for extrapolation in some applications.
than the cubic spline. Finally, the last two examples point out the different way in which critical-horizontal sequences of length two and greater are treated by the algorithm. Note that comonotonicity of the interpolant allows a "peak" somewhere in a critical-horizontal sequence of length at least three. We deliberately do not allow them in the interpolant. This is because, if the underlying function really has such a peak, then refinements of the partition will detect this. In this case, the former critical-horizontal sequence will be broken into two or more monotone sequences as more nodes are added. On the other hand, if the graph of the underlying function follows a horizontal line segment of positive length, and we allowed a "peak" in our interpolant, then no matter how fine the partition might be, our interpolant would always have the wrong shape!
Figure 6. Comonotone and cubic spline interpolants.

Figure 7. Comonotone and cubic spline interpolants.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 8. Cononotone and cubic spline interpolants.
CHAPTER III

COMONOTONE INTERPOLATION ALGORITHM TWO

3.1 Introduction

In this chapter we present a second algorithm for comonotone interpolation. This algorithm, though it involves significantly more computation than Algorithm 2.1, has the optimal order approximation property. It may be regarded as a generalization of the first method, so its analysis will provide additional insights into Algorithm 2.1 as well.

Our plan is to first show that there exist optimal order comonotone interpolants to direction determining data sets. We will identify a set of comonotone interpolants that must contain one of optimal order. We then direct our attention toward the construction of an efficient search strategy. Finally, we will demonstrate the algorithm on a few data sets.

3.2 Definitions and Preliminary Results

In this section, we shall assume that the algorithm used to construct monotone optimal order interpolants is that of Beatson and Wolkowicz. It is clear that any algorithm with the desired properties (analogues of the following lemmas) would do. We begin with two more lemmas which we shall employ in
our analysis of the next algorithm. The following two lemmas are taken from results due to Beatson and Wolkowicz (1989).

**Lemma 3.1.** Given monotone data \((t_i, f(t_i))_{i=0}^{n}\), let \(\delta\) denote the mesh size of \(t\). Choose a knot sequence \(t, t \supset t\), satisfying the hypotheses of Lemma 2.4. Let \(S = S_{r,k}, k \geq 3\), be the space of piecewise polynomial functions of order \(k\) on the knot set \(t\).

Let \(A\) be an algorithm with the following structure:

(a) Fit an initial interpolant \(s_1 \in S\) with, for some subset of the \(j\)'s in \(j = 1, 2, \ldots k - 1\), the optimal order approximation property.

(b) Modify \(s_1\) to obtain a monotone interpolant \(s_2 \in C \cap S\) with

\[
\|s'_1 - s'_2\| \leq A \inf_{s \in C \cap S} \|s'_1 - s\|,
\]

where \(u \supset r\) satisfies the hypotheses of Lemma 2.3, \(A\) is a constant not depending on \(t\), \(f\) or \(s_1\), and \(C\) is the space of monotone functions interpolating the given data values \(f(t_i)\) at the nodes \(t_i\).

Then \(A\) has the optimal order approximation property.

**Lemma 3.2.** The piecewise cubic interpolants \(s_2\) produced by the Beatson–(3.1), and are of optimal order.

We next show that there are comonotone piecewise cubic interpolants of optimal order. The proof of this fact leads us toward our next algorithm.
THEOREM 3.1. Let \( D = \{(x_i, y_i)\} \) be a direction determining data set for \( f \in C^j[a, b] \) for some \( j = 1, 2, \) or \( 3, \) and where \( f' \) has finitely many changes in sign on \([a, b]\). Then there exists a comonotone interpolant \( s \) to \( D \) satisfying (1.7).

PROOF. Let \( G \) be a set of optimal order approximations to \( \{f'(x_i)\} \), where \( g_{i-1}, g_i \) and \( d(i-1,i) \) agree in sign if and only if \( f \) is monotone on \([x_{i-1}, x_i]\).

Apply Algorithm 2.1., obtaining the interpolant \( s \). By Theorem 2.1, \( s \) is comonotone to \( f \) and \( D \). By Theorem 2.2, \( s \) satisfies (1.7), except possibly on subintervals \([a_i, b_i]\) near a critical data point.

Now in the proof of Theorem 2.2, we did not obtain optimal order convergence on the intervals \([a^{(n)}_i, b^{(n)}_i]\) containing a critical data point, simply because we did not know whether to enforce monotonicity on \([x_{k-1}, x_k]\), or \([x_k, x_{k+1}]\), for a critical singleton \( P_k \). But by assumption here, the signs of the \( g_i \) are chosen to avoid this problem. Hence, Algorithm 2.1 will enforce monotonicity in the interpolant \( s \) if and only if the underlying function \( f \) is monotone. Hence, \( s \) satisfies (1.7) for all of \([a, b]\). \( \blacksquare \)

Before moving on, we note that the condition that \( g_{i-1}, g_i \) and \( d(i-1,i) \) agree in sign if and only if \( f \) is monotone on \([x_{i-1}, x_i]\) may be slightly relaxed. Namely, in the case that \( P_k \) is a critical singleton, with \( f \) monotone on exactly one of \([x_{k-1}, x_k]\), or \([x_k, x_{k+1}]\). In this case, we must exclude the possibility that \( g_k = 0 \), and both \( g_{i-1} \) and \( g_{i+1} \) agree with \( d(i-1,i) \) and \( d(i,i+1) \), respectively. For then we would have agreement of the \( g_i \) and the divided
difference on an interval where \( f \) is not monotone. But this is easily handled, simply by considering the two cases of enforcing monotonicity of \( s \) on the left or right of \( x_k \). These cases correspond exactly to considering \( g_k \) to be either positive or negative, disregarding the fact that \( g_k = 0 \). Finally, we note that this procedure will not cause confusion in the event that \( f \) is monotone on both sides of a critical singleton \( P_k \) (i.e. \( (x_k, y_k) \) is a critical point of \( f \) as well as a critical singleton). For in this case, \( s \) will still have only one extreme point by enforcing monotonicity on only one side of \( x_k \). Furthermore, Lemma 2.3 shows that, since no knot will be added on an interval where monotonicity is not enforced, \( s \) will have the desired accuracy.

The above theorem contributes considerably more to our understanding of the problem than the mere existence of the desired interpolants. What it shows is that if we can construct \( D \)-consistent derivative estimates \( G \) of optimal order, we need only decide on the sign of \( g_i \) at each critical singleton. If we do this in the appropriate way, then Algorithm 2.1 will provide the desired accuracy. Thus our focus shifts from the continuous problem of constructing interpolants over an interval, to the discrete problem of finding a vector \( v = [v_1, v_2, \ldots, v_n] \) where \( v_i = \text{sgn}(g_i) \) for some set of \( g_i \)'s leading to an optimal order interpolant.

Let \( s \) be an optimal order interpolant to \( D \)—say a cubic spline function. Let the initial values \( g_i \) be the values of \( s'(x_i) \), corrected to make them \( D \)-consistent. In general, if there are \( n \) critical singletons in \( D \), then there will
be $2^n$ possible $D$-consistent sets $G$. Of all possible sets $G$, at least one leads to an optimal order comonotone interpolant.

We attempt to choose the set $G$ that leads to the interpolant $\phi$ minimizing $\|s - \phi\|$. Since $s$ is near $f$, the proof that the output of Algorithm 2.1 is optimal order is little more than an application of the triangle inequality.

If we are to use the strategy outlined above, we must find some efficient way to search through the set of candidate interpolants. We do this with a certain multistage graph.

Before giving a formal definition, we provide an intuitive explanation. Given a data set $D$ and derivative estimates $G$, $\mathcal{G}(D, G)$ is a multistage graph with one stage per data point in $D$. A stage corresponding to a critical singleton has two vertices, while other stages have only one. Two consecutive stages have all possible edges between them, and no other edges are in $\mathcal{G}(D, G)$. We illustrate the construction of the graph (without weights) in Figure 9 below.

The idea behind using this graph is that each critical data point presents us with two possible decisions: take the positive or negative estimate for $f'(x_i)$. Once these decisions are made, an essentially unique interpolant to $D$ is determined. Our goal is to minimize the change in initial optimal order estimates to $f'(x_i)$. We do this by constructing a weighting function $w(\varepsilon)$ for the edges of $\mathcal{G}$, so that the weight of an edge will measure the distance between the corresponding interpolant and an initial fit.
DEFINITION 3.1. Let \( D = \{P_i \mid i = 1, 2, \ldots, n\} \) be a data set, where \( P_i = (x_i, y_i) \), and \( y_i = f(x_i) \). Let \( G = \{g_i \mid i = 1, 2, \ldots, n\} \) be a set of \( D \)-consistent estimates for \( \{f'(x_i)\} \). \( G(D, G) \) denotes the weighted multistage graph with vertex set \( V \), edge set \( E \), and weight function \( w \) constructed in the following way.

The vertex set \( V \) is partitioned into \( n \) subsets \( S_i \) called stages, corresponding to the \( n \) data points \( P_i \in D \). For each data point \( P_i \) in \( D \) that is not a critical singleton, the stage \( S_i \) consists of exactly one vertex \( v_i \). For each critical singleton \( P_i \), stage \( S_i \) consists of two vertices, \( v_i^+ \), and \( v_i^- \). The edge set \( E \) contains only those pairs \( \{u, v\} \) where \( u \in S_i \) and \( v \in S_{i+1} \) for some \( i \), \( 1 \leq i \leq n - 1 \). Hence the edge set of \( G(D, G) \) is composed of all possible edges.
between stages with consecutive indices, and no others.

Construct the weight function $w$ in the following way. Compute monotone interpolants to nontrivial difference-induced monotone sequences of $D$, and adjust the values of the $g_i$ accordingly. For each vertex $v$ in $G$ we associate the corresponding derivative estimate $h_i$, where $h_i = \pm g_i$ as described above. Then there is a natural mapping from the edges $\{u_i, u_{i+1}\}$ of $G$ to pairs of derivative estimates $(h_i, h_{i+1})$.

Consider the case where $g_{i-1}$ and $g_i$ are both not zero. The data points $P_i$ and derivative estimates $g_i$ determine an initial cubic hermite interpolant $\psi$. If $h_i, h_{i+1}$, and $d(i, i+1)$ agree in sign, compute a new interpolant $\phi$ monotone on $[x_i, x_{i+1}]$ by modifying the initial fit $\psi$. Note that if $g_i$ and $g_{i+1}$ are modified to achieve monotonicity, $\phi$ may differ from the initial fit $\psi$ on $[x_{i-1}, x_{i+2}]$, but nowhere else. If $h_i, h_{i+1}$, and $d(i, i+1)$ do not agree in sign, we take $\phi$ to be the initial fit $\psi$. Then for each edge $e = \{u_i, u_{i+1}\}$ put $w(e) = ||\phi' - \psi'||_{[x_{i-1}, x_{i+2}]}$. Note that $w(e) = 0$ whenever the initial interpolant $\psi$ is unchanged.

Now consider the case where a derivative estimate, $g_i$, is zero. If $P_i$ is not a critical singleton, we proceed exactly as above. However, if $P_i$ is a critical singleton, and $g_i$ is zero, then our construction above associates $h_i = 0$ with both $v_i^+$ and $v_i^-$. For the purposes of determining when $h_i$ agrees in sign with other quantities, we consider $h_i$ to be positive when it is associated with $v_i^+$, and negative when it is associated with $v_i^-$. Under this convention we
construct \( \phi \) and \( w(e) \) exactly as in the previous case. This decision determines where monotonicity will be enforced in the final interpolant. This completes the definition of \( \mathcal{G}(D,G) \).

Now consider a path through \( \mathcal{G}(D,G) \) that passes through exactly one vertex in each stage of \( \mathcal{G} \). This path defines a choice for the sign for each critical point, and hence an interpolant. Let the cost of a path, \( \text{cost}(\mathcal{P}) \), be defined as the maximum of the weights of its edges. There is a useful relationship between the cost of a path and the distance between the associated interpolant and an initial fit.

**Lemma 3.3.** Let \( f \in C^1 \) have isolated relative extreme sets on \([a, b]\), and \( \Delta_f = \min |\alpha - \beta| \), where \( \alpha \) and \( \beta \) are chosen so that \((\alpha, \gamma)\), and \((\beta, \eta)\) are in distinct relative extreme sets, or so that \( \gamma = \eta \) and \( \{(\zeta, \gamma) : \alpha \leq \zeta \leq \beta, \alpha < \beta \} \) is a single relative extreme set. Let data points \( D \), and \( D \)-consistent derivative estimates \( G \) that induce monotone interpolants on the non-trivial difference-induced monotone sequences of \( D \) be given. Furthermore, assume that all derivative estimates are zero in critical horizontal sequences of length at least three. Denote by \( \mathcal{P} \) a minimum cost path passing through each stage of \( \mathcal{G}(D,G) \), and by \( \phi \) the corresponding interpolant. Let \( \psi \) be the cubic hermite interpolant to \( D \) and \( G \), and \( \delta \) be the mesh size of \( D \). If \( \delta \leq \Delta_f / 6 \), then

\[
\text{cost}(\mathcal{P}) = \| \phi' - \psi' \|_{[a, b]}.
\]
PROOF. Since $\delta \leq \Delta_f/6$, $D$ is direction determining. In fact, there are at least four non-critical data points between any two critical data points of $D$.

Suppose that $P_i$ and $P_{i+k}$ are consecutive critical data points. Since $\delta \leq \Delta_f/6$, $k \geq 5$.

In the construction of a comonotone interpolant $\phi$, $g_i$ and $g_{i+1}$ may be modified to achieve monotonicity over $[x_i, x_{i+1}]$. This may cause $\phi$ to differ from $\psi$ on $[x_i, x_{i+1}]$, and $[x_{i+1}, x_{i+2}]$, but not on $I = [x_{i+r}, x_{i+r+1}]$, where $r > 2$. Similarly, modifications of $g_{i+k-1}, g_{i+k}$ cannot change the initial fit $\psi$ on any interval $I = [x_{i+k-1-r}, x_{i+k-1}]$, where $r > 1$. Since $G$ induces monotone interpolants on intervals not containing a critical data point, $\phi \equiv \psi$ in at least one interval between consecutive critical singletons $P_i$ and $P_{i+k}$. Namely, $\phi \equiv \psi$ on $[x_{i+2}, x_{i+3}]$. As a result, the interpolant $\phi$ is determined in a completely local manner from $\mathcal{P}$. Then on each subinterval, $I = [x_{i-1}, x_{i+2}]$ where the initial fit is modified to achieve monotonicity on $[x_i, x_{i+1}]$, $||\phi' - \psi'||_I = w(e)$ for the edge $e$ in $\mathcal{P}$ corresponding to $[x_i, x_{i+1}]$. Thus

$$\text{cost}(\mathcal{P}) = \max_{e \in \mathcal{P}} w(e) = \max_I ||\phi' - \psi'||_I = ||\phi' - \psi'||_{[a,b]}.$$ 

3.3 Algorithm and Analysis

In this section we present our second algorithm for the construction of comonotone interpolants. This algorithm is of optimal order.
Algorithm 3.1. Let $D$ and $G$ be given, where $D = \{ P_i \mid P_i = (x_i, y_i); i = 0, 1, \ldots, n \}$, $a = x_0 < x_1 < \cdots < x_n = b$, and $G = \{ g_i \mid i = 0, 1, \ldots, n \}$ are slope approximations. Furthermore, assume there exists an algorithm to construct monotone piecewise cubic interpolants to monotone data (such as the Beatson–Wolkowicz algorithm). Compute a comonotone interpolant to $D$ as follows.

(a) If $G$ is not $D$-consistent, then replace $g_i$ by $-g_i$ wherever necessary to achieve consistency.

(b) Construct the labeled graph $G(D, G)$, and a minimum cost path $P$ passing through each of the stages of $G$ exactly once. Replace $G$ with the new derivative estimates $\hat{g}_i = \pm g_i$ induced by $P$.

(c) Apply Algorithm 2.1 to the data $D$, and modified slope estimates $G$.

Theorem 3.2. Let $s$ be the interpolant produced by Algorithm 3.1 to a data set $D$. Then the number of changes in sign in $s'$ on $[a, b]$ is the same as the number of critical sequences in $D$.

Proof. Algorithm 2.1 produces only comonotone interpolants.

Theorem 3.3. Consider the collection of functions $f \in C^j[a, b]$, $j = 1, 2$, or 3. Then Algorithm 3.1 restricted to such functions and data sets $D$ with $\delta \leq \Delta_f / 6$ has the optimal order approximation property, if the algorithms used to compute the derivative estimates $g_i$ and the monotone interpolants are of optimal order.
PROOF. Let $s$ be the interpolant produced by Algorithm 3.1, and let $\psi$ be the cubic hermite interpolant to $D$ and $G$, where $G$ is $D$-consistent. By Lemma 3.3 $\text{cost}(\mathcal{P}) = ||\psi' - s'||$. By Theorem 3.1 and its proof, there exists an optimal order interpolant $s^*$ corresponding to some path $\mathcal{Q}$ through $\mathcal{G}$. By the selection of $\mathcal{P}$, $\text{cost}(\mathcal{P}) \leq \text{cost}(\mathcal{Q})$. Hence,

$$||s' - \psi'|| = \text{cost}(\mathcal{P}) \leq \text{cost}(\mathcal{Q}) = ||(s^*)' - \psi'||.$$ 

But, both $s^*$ and $\psi$ satisfy (1.7), so we have

$$||s' - f'|| \leq ||s' - \psi'|| + ||\psi' - f'||$$

$$\leq ||(s^*)' - \psi'|| + ||\psi' - f'||$$

$$\leq C_1 \delta j^{-1} \omega(f^j, \delta) + C_2 \delta j^{-1} \omega(f^j, \delta)$$

$$\leq C_3 \delta j^{-1} \omega(f^j, \delta).$$

Since $s$ interpolates $f$, we obtain (1.7):

$$||f^{(1)} - s^{(1)}|| \leq K \delta j^{-1} \omega(f^{(j)}, \delta), \quad 1 \leq j \leq 3, \ 0 \leq l \leq 1.$$

**THEOREM 3.4.** Let the number of data points in the data set for either Algorithm 2.1 or Algorithm 3.1 be $n = |D|$. Then the algorithm can be executed in time linear in $n$. That is, the time complexity $T(n)$ satisfies

$$T(n) = O(n).$$
PROOF. We note that Algorithm 2.1 can be considered a special case of 3.1, where the weighting function \( w(e) \) is specified in a different manner. Namely, we set \( w(e) = 0 \) for the edge corresponding to the initial \((D\text{-consistent})\) values of the \( g_i \), and \( w(e) = \infty \) otherwise. The result for Algorithm 3.1 will therefore induce the corresponding statement for 2.1.

For each data point \( P \in D \), there is exactly one stage in \( \mathcal{G} \). Each stage of \( \mathcal{G} \) has at most two vertices, and there are at most four edges between stages. Since each additional stage requires the computation of at most four additional norms to label the graph, \( \mathcal{G} \) can be constructed in linear time.

The path \( \mathcal{P} \) can be found in linear time as follows. The selection of a minimum cost path to the first stage (which must consist of a single vertex) is trivial. Given minimum cost paths to all vertices in stage \( k \), one can extend them to minimum cost paths to all vertices in stage \( k + 1 \) by inspecting at most four edges. Hence the work required to extend a path by one step is bounded by a constant. If, at each vertex, we record the previous vertex on the path, once the last vertex \( v_n \) is reached, we have a record of the entire path. This path induces an interpolant.

Now on each subinterval \( I = [x_i, x_{i+1}] \), \( \mathcal{P} \) defines the derivative estimates \( h_i = \pm g_i \). This information, together with \( y_i \) and \( y_{i+1} \), completely determines the interpolant \( s \) on \( I \). Thus \( s \) is computed locally, and again can be found in linear time.

Although the determination of a set of optimal order derivative esti-
mates is not officially a part of Algorithm 3.1, it can be computed in linear time. Difference schemes based on a local fourth order polynomial are of optimal order. Thus the algorithms presented above provide a means for computing comonotone interpolants from the "raw" data \((x_i, f_i)\) in linear time.

3.4 Implementation Considerations

In this section, we consider some practical issues involved in implementing Algorithms 2.1 and 3.1. We first note that, though we have used the term "algorithms," it would perhaps be better to call them "meta-algorithms." For the process of constructing monotone piecewise cubic interpolants has not been discussed in any detail. Indeed, there is no reason that we should even be restricted to piecewise polynomials! We require only the ability to construct monotone interpolants to monotone functions of sufficient accuracy. If some other class of functions were used to interpolate the monotone sequences, we could then tie the resulting monotone sections together with cubic pieces. The only necessary restrictions are accuracy and the ability to control the number and location of relative extrema. Piecewise rational functions, for instance, might be a reasonable alternative.

The second issue to consider is the weight function for the edges of the graph \(G\). We have worked under the assumption that \(w(e) = ||\phi' - \psi'||_I\) for an appropriate subinterval \(I\). Of course it is not necessary that the norm itself be computed. The proof of Theorem 3.3 can be easily modified in case the cost
of the path $\mathcal{P}$ is merely equivalent to the norm in the sense that there exists a constant $K > 0$ satisfying

$$\frac{1}{K} \text{cost}(\mathcal{P}) \leq \|\phi' - \psi'\|_{[a,b]} \leq K \text{cost}(\mathcal{P}).$$

There are obviously a variety of functions that will do. If, for instance, a piecewise cubic interpolation scheme that does not add knots is used, the equivalence of norms in a finite dimensional linear space assures us that we need only check the change in derivative values at the endpoints of a subinterval. It is conceivable that an appropriate selection of the weight function could account for a substantial reduction in computational cost. Other benefits may also be realized by modifications of this sort.

Consider, for example, the use of the Fritsch-Carlson method (which does not add additional knots). In this case, all possible comonotone interpolants are contained in the finite dimensional linear space $V$ of piecewise cubic functions, with knots and nodes at $x_i, (x_i, y_i) \in D$. Then $\|s\| := \max\{|s(x_i)| : i = 1, 2, \ldots, n\}$ is a norm on $V$. A weight function $w$ consistent with this norm may be constructed as follows.

Let the edge $e$ correspond to derivative estimates $h_i$ and $h_{i+1}$, where $h_i = \pm g_i$ and $h_{i+1} = \pm g_{i+1}$. If $h_i, h_{i+1}$, and $d(i, i + 1)$ agree in sign, adjust the derivative estimates according to the Fritsch-Carlson algorithm, obtaining new estimates $\hat{h}_i$ and $\hat{h}_{i+1}$. Then put $w(e) = \max_{j=i,i+1} |h_j - \hat{h}_j|$. Otherwise, put $w(e) = \max_{j=i,i+1} \{|h_j - g_j|\}$. Note that in the latter case $w(e) = 0$.
or \( w(e) = 2|g_j| \) for \( j = i \) or \( j = i + 1 \). Using the norm above, the following relationship is then obtained for the path \( \mathcal{P} \), and the initial fit and comonotone interpolants \( \phi \) and \( \psi \).

\[
\frac{1}{2} \text{cost(}\mathcal{P}\text{)} \leq ||\phi' - \psi'||_{[a,b]} \leq \text{cost}(\mathcal{P}).
\]

This inequality is easily obtained with the following observations. If an initial derivative estimate \( g_i \) is replaced by \( h_i = -g_i \) to avoid a projection onto the set of monotone interpolants, then a projection on an adjacent interval may only reduce the absolute value of \( h_i \). Such a projection, however, may not change the sign of \( h_i \). Hence an actual change of at least \( |g_i| \) and at most \( 2|g_i| \) must occur in the construction of the final interpolant at \( x_i \). If \( \text{cost}(\mathcal{P}) \) is based on this change, we are done. However, if \( \text{cost}(\mathcal{P}) \) is incurred on some interval involving a projection onto the set of monotone interpolants, \( \text{cost}(\mathcal{P}) \) will exactly equal the weight of the corresponding edge \( e \).

Now in addition to a reduction in computational cost of the weight function \( w \), it may be noted that the above argument does not require that \( \delta \leq \Delta_f/6 \). In fact, results analogous to Lemma 3.3 and Theorem 3.3 may be obtained, where \( \delta \leq \Delta_f/6 \) is replaced by the requirement that \( D \) be direction determining. This version of Algorithm 3.1 was not presented as the main result because the Fritsch-Carlson algorithm is only third-order accurate. The result is a third-order algorithm for constructing co-monotone interpolants, with weaker hypotheses than Algorithm 3.1.
The third and final issue we wish to discuss is the representation of the interpolant $s$. Algorithms 2.1 and 3.1 are essentially algorithms to compute derivative estimates $g_i$ that lead to comonotone interpolants. In many cases, the cubic spline function (with a reasonable choice of end conditions) is either comonotone, or nearly comonotone with the data. If we begin with a cubic spline fit $\psi$, and modify it to obtain a comonotone interpolant $\phi$, it may be the case that $\phi$ differs from $\psi$ on very few subintervals. If $\phi$ is represented in B-spline form, we need only one knot at each node where $\phi$ has two continuous derivatives. In such cases, the information needed to store $\phi$ can be locally reduced by a factor of two. If the spline was not modified at most knots, this could result in a substantial savings.

It is therefore suggested that one keep track of data points $(x_i,y_i)$, where $g_{i-1}, g_i,$ and $g_{i+1}$ have not been changed, and where no new break points were added on either $[x_{i-1},x_i]$ or $[x_i,x_{i+1}]$. At these points a single knot will suffice. This information may be used to construct a knot sequence $t$, a superset of $x$, satisfying

$$t : t_0 = \cdots = t_3 = x_1 < t_4 \leq t_5 \cdots \leq t_{m-3} = \cdots = t_m = x_n,$$

and $t_i < t_{i+2}$, $3 \leq i \leq m - 3$. The output of Algorithm 2.1 or 3.1 may then be used to construct the B-spline coefficients. In dealing with piecewise cubic hermite interpolation, we ordinarily do not have the continuity conditions
implied by the knot sequence \( t \). Furthermore, it cannot be accurately determined numerically due to round-off error. Hence, formulas for constructing the B-spline coefficients from hermite data are not usually given in the literature. In fact, we are not aware of any source for these formulas. They are therefore provided below.

One can easily construct the coefficients for the shifted power form of the polynomial pieces. Conte and deBoor (1980) provide an elementary description, and FORTRAN code on page 116. What is essentially a translation of this code to MatLab may be found in Appendix A. Following deBoor (1978) the B-spline coefficients may be computed with the operators \( \lambda_i \) below. We also use the fact that

\[
\lambda_i(B_j) = \delta_{i,j},
\]

where \( B_i \) is the \( i^{\text{th}} \) B-spline for the knot sequence \( t \), \( \lambda_i \) is defined by

\[
\lambda_i(f) := \sum_{r=0}^{k-1} (-1)^{k-1-r} \psi_i^{(k-1-r)}(\tau_i) D^r f(\tau_i),
\]

\[
\psi_i(t) := (t_{i+1} - t) \ldots (t_{i+k-1} - t)/(k-1)!, \quad \text{and} \quad \tau_i \in (t_i, t_{i+k}) = \text{supp} B_i.
\]

If we take \( k = 4 \) (i.e. cubic B-splines) and choose

\[
\tau_i = t_{i+2}^+ \quad \text{for } 1 \leq i \leq m - 3,
\]

\[
\tau_0 = \frac{t_1 + t_2 + t_3}{3}, \quad \text{and} \quad \tau_{m-4} = \frac{t_{m-3} + t_{m-2} + t_{m-1}}{3},
\]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
we may compute the coefficients \( \alpha_i = \lambda_i(s) \), obtaining

\[
\alpha_1 = f_1
\]

\[
\alpha_{m-4} = f_n
\]

\[
\alpha_i = f_i + \frac{s_i}{3} (\Delta t_{i+2} - \Delta t_{i+1}) + \left( \frac{\Delta t_{i+2} \Delta t_{i+1}}{3} \right) \left( \frac{3f[x_i, x_{i+1}] - s_{i+1}}{\Delta x_i} \right),
\]

for \( 2 \leq i \leq m - 5 \), where \( x_i = t_{i+2} \), and \( \Delta t_i = t_{i+1} - t_i \).
CHAPTER IV

COCONVEX INTERPOLATION ALGORITHM

4.1 Introduction

In this chapter, we turn to the problem of co-convex interpolation. As with the comonotone problem, our goal is to preserve the shape of the data and underlying function. Furthermore, we desire a solution that is efficient, both in terms of the time and space required to construct and evaluate an interpolant. Our techniques should apply to arbitrary functional data sets, should preserve shape and should give good accuracy under reasonable assumptions. We desire interpolants that are local to the data. Finally, we prefer schemes that are conceptually simple.

Several researchers have contributed to the solution of this problem, but previous work on the problem can be conveniently described as of two basic types. The first of these approaches is to construct piecewise cubic interpolants, and attempt to enforce convexity/concavity requirements by carefully choosing the slopes of the interpolant at the knots. Typical of these techniques is the work of Mettke (1983), Costantini and Morandi (1983), and of Costanini (1986). Here one constructs an algorithm to recognize data sets for which a $C^1$ piecewise cubic convex interpolant with knots and nodes iden-
tified can be constructed. For such data sets—and there are convex data sets for which there is no such interpolant—suitable derivatives are chosen. Perhaps the greatest difficulty with these techniques is that they do not apply to general convex data sets, let alone data requiring points of inflection in the interpolant. Furthermore, even though general fourth order piecewise polynomial interpolation can achieve fourth order accuracy, these techniques are plagued by poor accuracy. The error in the interpolants constructed by both Mettke (1983), and Costantini and Morandi (1983) is a disappointing $O(h)$, although those of the latter are $O(h^2)$ for all subintervals except the first and last. Finally, the interpolants produced depend globally on the data. Thus they are unsuitable for applications where the data may be frequently changed, or where one wishes to compute the interpolant as the data arrives.

The possibility of points of inflection in cubics motivates the second technique. Here one constructs a piecewise quadratic interpolant. Since its derivative is piecewise linear, it is easy to enforce convexity in the interpolant by adjusting its first derivative at the knots. Given an arbitrary selection of two data points and derivative values, a quadratic interpolant cannot always be found, thus it may be necessary to add knots. This technique is employed quite successfully by Schumaker (1983). His method applies to arbitrary data sets, the interpolants constructed are comonotone with the data, and may be made coconvex with some additional interactive steps. Unfortunately, Schumaker apparently did not consider accuracy as a goal. In fact he does not even provide
an analysis of the error. However, due to the large amount of flexibility built into his algorithm, it seems unlikely that the results are much better than the piecewise cubic schemes.

We suggest that a similar, but new approach to the problem be taken. We propose to use piecewise cubic interpolation, and to add additional knots where necessary to gain the existence of the desired interpolant. We present a brief theoretical discussion to demonstrate that this approach is reasonable. This discussion leads us to a preliminary algorithm which gives remarkably good results.

4.2 Definitions and Preliminary Results

In this section we present some results preliminary to our next algorithm. We first consider the problem of the existence of convex and concave interpolants. The following theorem is reported by Mettke (1983).

**Theorem 4.1.** There exists a convex set of five data points, for which no convex piecewise polynomial interpolant of order four, with knots and nodes identical, can be found.

The previous theorem indicates that we cannot hope to construct co-convex interpolants to arbitrary data sets, unless we are willing to increase the maximum degree of the interpolant, or to allow extra knots. Fortunately, at most one knot must be added to each subinterval. The following well known
technical lemma will be convenient in the proof of this fact.

**Lemma 4.1.** Let \( f \) be the hermite interpolant of order four to \((x_1, y_1)\), and \((x_2, y_2)\), with \( m_1 = f'(x_1) \), and \( m_2 = f'(x_2) \). Then necessary and sufficient conditions for \( f \) to have degree less than or equal to two, is that

\[
y_2 = \frac{(m_1 + m_2)}{2} (x_2 - x_1) + y_1.
\]

In this case, \( f \) is given by

\[
f(x) = y_1 + m_1 (x - x_1) + \frac{y_2 - y_1 - m_1 (x_2 - x_1)}{(x_2 - x_1)^2} (x - x_1)^2.
\]

**Proof.** We need only compute the leading coefficient of \( f \), \( f[x_1, x_1, x_2, x_2] \), and determine when it is zero. By a direct computation, we arrive at

\[
f[x_1, x_1, x_2, x_2] = \frac{m_1 + m_2 - 2 \frac{y_1 - y_2}{x_1 - x_2}}{(x_2 - x_1)^2}.
\]

Solving

\[
m_1 + m_2 - 2 \frac{y_1 - y_2}{x_1 - x_2} = 0
\]

produces the first result. Since \( f \) is uniquely determined by the four parameters given, it is sufficient to check that the expression above interpolates the data, with the proper endpoint derivatives. This is an elementary calculation.
We say that a data set is *convex* or *concave* if for each $i$ the second divided differences satisfy $f[x_i, x_{i+1}, x_{i+2}] \geq 0$ or $f[x_i, x_{i+1}, x_{i+2}] \leq 0$, respectively. The following fact has been known for some time (see for instance, Schumaker (1983)). We present, however, a very elementary and perhaps new approach to its proof.

**Theorem 4.2.** For every set $D = \{(x_i, y_i)\}$ of convex or concave (functional) data, there exists a correspondingly convex or concave $C^1$ piecewise polynomial interpolant of order four with at most one interior knot in each subinterval $[x_i, x_{i+1}]$.

**Proof.** Consider first a single interval $[x_1, x_2]$, with data points $(x_1, y_1),$ and $(x_2, y_2),$ and prescribed derivative values $m_1 = f'(x_1),$ and $m_2 = f'(x_2)$. Assume that $m_1 > m_2$ or $m_1 > f[x_1, x_2] > m_2$, so that a concave interpolant is desired (the other case is handled in a similar manner). We construct an interpolant as follows:

Let $ABC$ be the triangle with vertices $A(x_1, y_1)$, $B(x_2, y_2)$, and $C(x_c, y_c)$, where $C$ is the intersection of the lines with equations $y = m_1(x - x_1) + y_1$, and $y = m_2(x - x_2) + y_2$. Clearly, any concave interpolant must lie within this triangle. Our goal is to locate a "false" data point $K$ and derivative value, so that on each piece the third degree coefficient of the new piecewise cubic hermite interpolant will be zero. We do this by building a certain symmetry into the two subintervals we construct.

Let $D(x_d, y_d)$, and $E(x_e, y_e)$ be the midpoints of line segments $AC$...
and $BC$, respectively. By construction, triangles $ABC$ and $DEC$ are similar, and the sides of $DEC$ are one half the length of the corresponding sides of $ABC$. Finally, let $K(x_k,y_k)$ be the point on segment $DE$ chosen so that $KC$ is vertical.

We claim that the cubic hermite interpolant $h$ to $A, K, \text{ and } B$, with slopes $m_1, f[x_1, x_2]$, and $m_2$, is concave on $[x_1, x_2]$. To see this, observe that $x_d = \frac{x_1 + x_k}{2}$, and $x_e = \frac{x_k + x_2}{2}$. Since the slope of $DE$ must be the same as that of $AB$, we have

\[ y_k = \frac{m_1 + f[x_1, x_2]}{2}(x_k - x_1) + y_1, \]

and

\[ y_2 = f[x_1, x_2] + m_2(x_2 - x_k) + y_k. \]

Applying Lemma 1, we see that $h$ is piecewise quadratic on $[x_1, x_2]$. It is $C^1$ and concave by the selection of slope values. Thus we can achieve a concave interpolant on a single interval to concave data, by adding at most one interior knot. Now suppose that we have more than one interval to interpolate, with data abscissae $x_1, x_2, \ldots, x_n$, where $n \geq 3$. At each point, $x_i$, $2 \leq i \leq n - 1$, select a derivative value $m_i$ satisfying $f[x_{i-1}, x_i] \geq m_i \geq f[x_i, x_{i+1}]$. Similarly, choose $m_1 > f[x_1, x_2]$, and $m_n < f[x_{n-1}, x_n]$. Then to construct a piecewise concave $C^1$ interpolant $s$ on each interval where $s'(x_i) = m_i$, using the local construction described above. Since $s \in C^1$ and the sequence $(m_i)$ is monotonically decreasing, $s$ is globally concave on $[x_1, x_n]$. \[\square\]
Before continuing, we remark that the interpolant $s$ need not be limited to second degree. In fact, one might use the second degree interpolant only when a particular method for constructing third degree pieces fails. Furthermore, the argument above shows that there is at least one convex or concave interpolant in the set of piecewise polynomials of order 4 with at most one additional interior knot on each subinterval. Arguing by the continuity of the hermite interpolation operator, we see that a concave or convex interpolant of strictly third degree can certainly be obtained, if no three data points are colinear.

Finally, in preparation for our algorithm, we make three more definitions.

**Definition 4.1.** Let $D$ be a functional data set. A monotone sequence $M = (x_i, y_i)_{i=1}^l$, with length at least three, is called colinear if the $(x_i, y_i)$ lie on a common line. A colinear sequence of length three, $(x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1})$ is called an inflected colinear sequence if $f[x_{i-2}, x_{i-1}, x_i]$ and $f[x_i, x_{i+1}, x_{i+2}]$ exist and $f[x_{i-2}, x_{i-1}, x_i] \cdot f[x_i, x_{i+1}, x_{i+2}] < 0$.

**Definition 4.2.** Let $f \in C^2[a, b]$ and $a < x_1 < x_2 < b$. We say that $[x_1, x_2]$ is an interval or region of inflection, if in some open interval containing $[x_1, x_2]$, $\xi < x_1$ and $\eta > x_2$ implies $f''(\xi)f''(\eta) < 0$, while for $x_1 \leq \zeta \leq x_2, f''(\zeta) = 0$.

**Definition 4.3.** Let $D = \{(x_i, y_i) \mid i = 0, 1, \ldots, n, f(x_i) = y_i\}$ be given, where $f \in C^1$, and $f''$ is piecewise continuous on $[x_0, x_n]$. Then $D$ is convexity
determining to \( f \), if the number of changes in sign in \( f'' \) on \([x_0, x_n]\) is equal to the number of changes in sign in the sequence of second divided differences of \( D \).

4.3 Algorithm and Analysis

We are now prepared to present and analyze our algorithm.

**Algorithm 4.1.** Let \( D = \{(x_i, y_i) \mid i = 0, 1, \ldots, n\} \) be a functional data set, where \( n \geq 3 \). We construct a function \( s \in C^1 \) interpolating \( D \) so that \( D \) is convexity determining for \( s \).

(a) Let \( s'(x_0) = q'(x_0) \), where \( q \) is the local quadratic interpolant to \( D \) at \( x_0, x_1, \) and \( x_2 \). Construct \( s'(x_n) \) similarly, and put \( i = 2 \).

(b) While \( i \leq n - 1 \),

(i) If \( P_i \) is the left endpoint of an infected colinear sequence, take 
\[ s'(x_j) \] to be a weighted mean of \( f[x_{j-1}, x_j] \) and \( f[x_j, x_{j+1}] \), for \( j = i, \) and \( i + 2 \). Put \( s'(x_{i+1}) = q'(x_{i+1}) \), where \( q \) is the quadratic interpolant to \( P_{i-1}, P_i, \) and \( P_{i+1} \). Then increment \( i \) to \( i + 3 \).

(ii) If \( P_i \) is the left endpoint of a non-infected colinear sequence, 
\( (P_j)_{j=i}^{i+k} \), put \( s'(x_j) = f[x_i, x_{i+1}] \), for \( j = i, i + 1, \ldots, i + k \). Then increment \( i \) to \( i + k + 1 \).

(iii) In all other cases, put \( s'(x_i) \) equal to a weighted mean of 
\( f[x_{i-1}, x_i] \) and \( f[x_i, x_{i+1}] \). Increment \( i \) to \( i + 1 \).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(c) On each subinterval \([x_i, x_{i+1}]\), if \(f[x_i, x_{i+1}]\) is between \(s'(x_i)\) and \\
\(s'(x_{i+1})\), interpolate with a concave or convex piecewise polynomial \\
interpolant of order four, with a minimum number of knots.

(d) On all other subintervals, construct the piecewise cubic hermite inter-
polant.

THEOREM 4.3. Let \(D = \{(x_i, y_i) \mid i = 0, 1, \ldots, n\}\), where \(y_i = f(x_i), f \in C^3\), 
and let \(s\) be an interpolant computed according to Algorithm 4.3, where convex 
and concave piecewise polynomial interpolants are either the cubic hermite 
interpolant, or the piecewise quadratics described in the proof of Theorem 
4.2. Then there is a constant \(K\), independent of \(D\), so that

\[
\|s - f\| \leq K\delta^2,
\]

where \(\delta = \max|x_{i+1} - x_i|\).

PROOF. Consider a subinterval \(I = [x_i, x_{i+1}]\). Note first of all, that since the 
derivative estimates \(s'(x_i)\) are weighted means of the divided differences on 
the left and right, or are computed by a local quadratic, we must have

\[
\max_i |s'(x_i) - f'(x_i)| \leq C_1 \delta,
\]

for some constant \(C_1\), independent of \(D\). Thus, in the case that \(s\) is the cubic 
hermite interpolant on \(I\), we need only apply Lemma 2.3.
Suppose then that $s$ is constructed as in the proof of Theorem 4.2; we adopt the notation of that proof. Let $m_i = s'(x_i)$. Let $A$ correspond to the point with coordinates $(x_i, y_i)$, $B$ to $(x_{i+1}, y_{i+1})$, and $C$ to $(x_c, y_c)$, where the lines with equations $y = m_i(x - x_i) + y_i$ and $y = m_{i+1}(x - x_{i+1}) + y_{i+1}$ intersect at $(x_c, y_c)$. Let $M(x_m, y_m)$ be the intersection of the vertical line passing through $C$ with the line segment $AB$, and note that $x_m = x_c$.

The line segment $AB$ joining $(x_i, y_i)$ and $(x_{i+1}, y_{i+1})$ is the piecewise linear interpolant $l$ to $f$ restricted to $I$. Thus, by error the formulas for polynomial interpolation (see equation 1.2), $l$ satisfies

\[(4.2) \quad ||l - f|| \leq C_1 \delta^2.\]

Furthermore, $m_i$ and $d(i, i + 1)$ are first order approximations to $f'(x_i)$. That is, there is a constant $C_2$, independent of $D$ satisfying

\[|f'(x_i) - m_i| < C_2 \delta,\]

and

\[|f'(x_i) - d(i, i + 1)| < C_2 \delta.\]

Now it is easy to see that of all vertical line segments with endpoints on two sides of the triangle $ABC$, the one of maximum height passes through the vertex $C$, and point $M$. Since $s$ is contained in the interior of $ABC$, we
need only show that the height of this segment collapses as $\delta^2$. By direct computation, the vertical segment has height $|y_c - y_m|$ satisfying

$$0 \leq |y_c - y_m| = |(x_c - x_i)[m_i - d(i, i + 1)]|$$

$$< (x_{i+1} - x_i)[m_i - d(i, i + 1)]$$

$$= (x_{i+1} - x_i)[m_i - f'(x_i) + f'(x_i) - d(i, i + 1)]$$

$$\leq (x_{i+1} - x_i)[C_2\delta + C_2\delta]$$

$$\leq \delta^2 C_3,$$

where $C_3 = 2C_2$. Taking $C_4$ to be the maximum of the $C_i$, we obtain

$$||l - s|| \leq C_4 \delta^2.$$ 

Combining (4.2) and (4.3) we obtain (4.1).

**Theorem 4.3.** Let $D = \{(x_i, y_i) \mid i = 0, 1, \ldots, n\}$, where $y_i = f(x_i)$, $f \in C^3$, and $f$ is coconvex to $D$. Let $s$ be an interpolant computed according to Algorithm 4.3, where convex and concave piecewise polynomial interpolants are either the cubic hermite interpolant, or the piecewise quadratics described in the proof of Theorem 4.2. Then $s$ is coconvex with $f$ on $[x_0, x_n]$.

**Proof.** Since each derivative estimate $s'(x_i)$ is computed according to a local quadratic or a weighted mean of neighboring first divided differences, the sequence $s'(x_i)$ must be monotone in regions where $f$ is convex or concave. 

---

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Furthermore, on each subinterval \( s \) has at most one point of inflection, and then can only have one if no convex or concave interpolant exists to the data and slope estimates. Hence \( s'' \) changes sign exactly once for each change in sign in the sequence of second divided differences. Thus \( s \) is coconvex with \( D \), and hence with \( f \).

In addition to showing that Algorithm 4.1 is practical as it stands, Theorems 4.2 and 4.3 lead us to hope that fairly accurate interpolants can be found by appropriate choices for the derivative estimates, and perhaps by a more sophisticated method of knot insertion. Indeed, Theorem 4.2 puts a lower bound on the accuracy of the interpolant, but does not indicate that more accuracy is not achieved. Numerical experiments show that with slope averages derived by averaging the arctangents of angles of intersection of local linear interpolants, or by a \( O(\delta^2) \) two-sided estimate to \( f' \), \( s \) seems to approximate \( f \) with third order. Though extensive computations have not been carried out, due in part to limited computing facilities, we have not yet found an example where the algorithm does worse than third order, for \( f \in C^3 \).

4.4 Examples

In this section we present the results of performing Algorithm 4.1 on a few examples. Figure 10 shows the comonotone interpolant to a data set with rapid changes in the first derivative. Figure 11 demonstrates the comonotone
interpolant on the NASA data set, referred to earlier in Chapter I. Finally, in Figure 12, we demonstrate how the algorithm is able to handle linear regions in an underlying function—something the cubic spline is unable to do.
Figure 10. Coconvex interpolant and the cubic spline.

Figure 11. Coconvex interpolant and the cubic spline.
Figure 12. Coconvex interpolant and the cubic spline.
CHAPTER V

SUMMARY AND CONCLUSIONS

In this dissertation, we have presented three new shape-preserving algorithms: two for comonotone and one for coconvex piecewise cubic interpolation. A natural question is how these algorithms relate to one another. The relationship between the comonotone and coconvex algorithms is difficult to describe, since the two algorithms have different objectives. We can, however, fruitfully consider the relationship between the two comonotone algorithms.

In comparing Algorithms 2.1 and 3.1, we note first that they are indeed different, not only from the standpoint of their description, but also from their output. We exhibit an example in Figure 13 to illustrate that the cubic spline, Algorithm 2.1 and Algorithm 3.1 may all produce different interpolants. In the example, we have used the not-a-knot end conditions for the $C^2$ cubic spline, and have set the initial slope estimates $g_i$ for Algorithms 2.1 and 3.1 by evaluating the derivative of this spline at the $x_i$. Unless stated otherwise, we shall accept this method as the convention for the purposes of this chapter.

In studying this example, we immediately note that the data look contrived, and not at all typical of what is encountered in practice. This is certainly correct, and we shall return to this very interesting observation later. But first, we discuss the intuition that lies behind the construction of such an
Figure 13. Algorithms 2.1, 3.1, and the Cubic Spline.

example. If Algorithms 2.1 and 3.1 are to produce different interpolants, then we must find a data set so that on at least one subinterval, it is "cheaper" to change the sign of the slope estimate $g_i$ than to carry out the projection of the initial hermite interpolant onto the set of monotone interpolants. We desire an interval $I = [x_{i-1}, x_i]$ where $g_{i-1}, g_i$, and $d(i - 1, i)$ all agree in sign. Further, we want $|g_{i-1}| >> 0$, but $g_i \approx 0$, and $g_i \neq 0$. We then incur a cost on the order of $|g_{i-1}|$ to project onto the set of monotone functions, but a cost dependent on $g_i \approx 0$ and $g_{i+1}$ to avoid the projection. If the relative sizes of $g_i$ and $g_{i+1}$ are small enough compared to $g_{i-1}$, we have the desired example.

Now that we have demonstrated the difference in the algorithms, we continue in our comparison of them. There seem to be two primary points of
interest in comparing these algorithms: speed and accuracy. It is clear that Algorithm 2.1 has the advantage from the standpoint of speed; it does not require the construction of the graph or a minimum cost path. But it should be remembered that whatever computational cost is required, both Algorithms 2.1 and 3.1 are of linear time complexity (if linear time algorithms are used to compute the slope estimates).

When considering accuracy, the scales seem to tip in the other direction. We have proved that interpolants computed according to Algorithm 3.1 have fourth order convergence to an underlying $C^3$ function. We have not proved quite as strong a result for Algorithm 2.1. But the comparison is not as clear-cut it may seem at first sight. A few numerical experiments have been run, and it seems that Algorithm 3.1 has a slight edge over 2.1 when the norm of the partition is large, but this is far from the whole story. These experiments brought to light two interesting facts. It was observed, first of all, that in every example, Algorithm 2.1 produced interpolants that apparently converged to an underlying $C^3$ function as $\delta^4$! In fact, in each of these cases, for very reasonable choices of $\delta$ (the norm of the partition), Algorithms 2.1 and 3.1 gave precisely the same interpolants! We conjecture that under suitable hypotheses, this will always be the case. It seems that this occurs because of the nature of the cubic spline interpolant itself—from which we obtained the slope estimates $g_i$. We formalize our intuition in the following conjectures.

**Conjecture 5.1.** Let $f \in C^1[a, b]$, $\phi$ be an optimal order $C^2$ cubic spline
interpolant to \( f \), and \( s_1 \) and \( s_2 \) be interpolants to \( f \) computed according to Algorithm 2.1 and Algorithm 3.1, respectively, where the initial slope estimates \( g_i \) satisfy \( g_i = \phi'(x_i) \). Let each of the interpolants be computed from a partition \( \mathbf{x} \) of \([a, b]\), with norm \( \delta \). If \( \delta \) is sufficiently small, then \( s_1 \equiv s_2 \).

**Conjecture 5.2.** Let \( f \in C^1[a, b] \), and \( \phi \) be an optimal order \( C^2 \) cubic spline interpolant to \( f \), based on a partition \( \mathbf{x} \) of \([a, b]\) with norm \( \delta \). If \( \delta \) is sufficiently small, then \( \phi \) is a comonotone interpolant to \( f \).
APPENDIX A.

MatLab Code for Comonotone Interpolation
function pp=comonotone(x,f,d);

% pp=comonotone(x,f,d)
% This function returns a comonotone piecewise polynomial
% function to the data x,f,d, where (x,f) are the
% coordinates of the points in the plane, and d are
% derivative estimates at those points. If d is not
% specified, it will be estimated by a cubic spline.
% See also, mkpp, unmkpp, spline
%
% This function implements Algorithm 2.1 of the
% dissertation, by actually building the graph, as in
% algorithm 3.1, but assigning weights of infinity to
% edges corresponding to slope values not considered by
% this algorithm. That is, there exists only one path of
% finite cost passing through each stage of the graph
% exactly once.

n=size(x);
m=size(f);
if (n(1)~=1) | (m(1)~=1) | (n(2)~=m(2)),
    x
    f
    error ( 'x and f must be row vectors of the same length' );
end
if (nargin==2)
    d=ppval(ppder(spline(x,f)),x);
else
    m=size(d);
    if (m(1)~=1) | (m(2)~=n(2)),
        x
        f
        d
        error ( 'x, f and d must be row vectors of the same length' );
    end
end
seq_x=makseqs(x,f);
n=size(seq_x);

if(n(1)==1), % Only one sequence==>monotone data!
    n=0;
    [t,ft,dt,n]=moncubic(x,f,d,n);
else
    seq_info=classify(seq_x,f);
end
[seq_info, seq_brks, seq_f, seq_d] = moninterp(seq_info, seq_x, x, f, d);
stage = makgraph(seq_info, seq_brks, seq_d);
[dleft, dright, weight] = labelgraph(seq_info, seq_x, x, ...
    seq_f, seq_d, stage);
[cost, pathdata] = mincostpath(weight);
[t, ft, dt] = assemble(seq_info, seq_brks, seq_f, seq_d, ...
    pathdata, dleft, dright, stage);
end

pp = hermite(t, ft, dt);
end
function pp=comon2(x,f,d);

% This function returns a comonotone piecewise polynomial
% function to the data x, f, d, where (x, f) are the
% coordinates of the points in the plane, and d are
% derivative estimates at those points. If d is not
% specified, it will be estimated by a cubic spline.
% See also, mkpp, unmkpp, spline
%

% This code implements Algorithm 3.1 of the dissertation

n=size(x);
m=size(f);
if (n(1)~=1) | (m(1)~=1) | (n(2)==m(2)),
    x
    f
    error ('x and f must be row vectors of the same length');
end

if (nargin==2)
    d=ppval(ppder(spline(x,f)),x);
else
    m=size(d);
    if(m(1)==1) | (m(2)==n(2)),
        x
        f
        d
        error ('x, f and d must be row vectors of the same... length');
    end
end

seq_x=makseqs(x,f);
n=size(seq_x);

if(n(1)==1), % Only one sequence==>monotone data!
    n=0;
    [t,ft,dt,n]=moncubic(x,f,d,n);
else

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
seq_info=classify(seq_x,f);
[seq_info, seq_brks, seq_f, seq_d]=moninterp(seq_info,seq_x,x,f,d);
stage=makgraph2(seq_info,seq_brks,seq_d);
[dleft,dright,weight]=labelgraph(seq_info,seq_x,x,seq_f,seq_d,stage);
[cost,pathdata]=mincostpath(weight);
[t,ft,dt]=assemble(seq_info,seq_brks,seq_f,seq_d,...
pathdata,dleft,dright,stage);
end

pp=hermite(t,ft,dt);
end
function dspl=ppder(spl)

% Derivative of a piece-wise cubic polynomial
% dspl=ppder(spl)

% Given the pp function spl, this function returns the pp function dspl such that the breaks of dspl are those of spl and the pieces of dspl are the derivatives of the corresponding pieces of spl.

[breaks,coef,1,k]=unmkpp(spl);
coef=coef(:,1:k-1)*diag([k-1:-1:1]);
dspl=mkpp(breaks,coef);

end %END{ppder}
function seq_x=makseqs(x,f);
%
% seq_x=makseqs(x,f)
% This functions returns the set of monotone sequences.
%
% Last modified on Thursday, July 5, 1990.
%
lastdata=length(x);
seq_x(1,1)=1;
xindex=2;
xml=1;
xpl=3;
sindex=1;
lastcrit=-2;

while (xindex<lastdata),
    direc(1)=sign(f(xindex)-f(xml));
    direc(2)=sign(f(xpl)-f(xindex));

    if(direc(1)==0),
        [xend,critical]=horizontal(xml,x,f);
        if (critical),
            seq_x(sindex,2)=xend;
            sindex=sindex+1;
            xindex=xend+1;
            seq_x(sindex,1)=xindex;
        else
            xindex=xend;
        end
    elseif(direc(2)==0),
        [xend,critical]=horizontal(xindex,x,f);
        if (critical),
            if (lastcrit<xml),  % xml already closed if critical
                seq_x(sindex,2)=xml;
                sindex=sindex+1;
            end
            seq_x(sindex,1:2)=[xindex,xend];
            sindex=sindex+1;
            xindex=xend+1;
            seq_x(sindex,1)=xindex;
        else
            xindex=xend;
        end
    else
        % Handle non-monotonic case
    end
end
else
    if (direc(2)*direc(1)<0),
        if (lastcrit<xml), % xml already closed if critical
            seq_x(sindex,2)=xml;
            sindex=sindex+1;
        end
        seq_x(sindex,1:2)=[xindex,xindex];
        lastcrit=xindex;
        sindex=sindex+1;
        seq_x(sindex,1)=xpl;
    end
end
end %of while loop

seq_x(sindex,2)=lastdata;
end % of makseqs.
function [xend,critical]=horizontal(start,x,f)

% [xend,critical]=horizontal(start,x,f)
% This function is called by makseqs to determine the
% right endpoint of a horizontal sequence, given its left
% endpoint. It also classifies the horizontal sequence as
% critical or not.

% written Thursday, June 28, 1990.
% original version.
% lastdata=length(x);
xml=start;
xend=start+1;
direc=sign(f(xend)-f(xml));
if(direc~=0), error('this is not a horizontal sequence!');
end

while (direc==0) & (xend<lastdata),
    :ml=xend;
    :end=xend+1;
    :direc=sign(f(xend)-f(xml));
end
if (direc==0),
    critical=0;
else
    xend=xml;
    if (start==1),
        critical=0;
    else
        direcL=sign(f(start)-f(start-1));
        critical=(direcL*direc<0);
    end
end
end
function seq=classify(seq_x,f);
%
% seq=classify(seq_x,f);
%
% This function classifies the sequences in seq_x as
% interior(1), end(2), critical point(3), or critical
% horizontal(4).
%
%
% written Thursday, June 28, 1990.
% Last modified, Tuesday, July 3.
%
interior=1;
endsequence=2;
criticalpt=3;
criticalhor=4;
lastdata=length(f);
[lastseq,junk]=size(seq_x);
clear junk

for sindex=1:lastseq,
    if (seq_x(sindex,1)==1) | (seq_x(sindex,2)==lastdata),
        seq(sindex,1)=endsequence;
    else
        L=seq_x(sindex,1);
        R=seq_x(sindex,2);
        direcL=sign(f(L)-f(L-1));
        direcR=sign(f(R+1)-f(R));
        if (direcL*direcR<0),
            if(R-L>0),
                seq(sindex,1)=criticalhor;
            else
                seq(sindex,1)=criticalpt;
            end
        else
            seq(sindex,1)=interior;
        end
    end
end
function [seq_info,seq_brks,seq_f,seq_d]=moninterp... 
    (seq_info,seq_x,x,f,d) 

%  [seq_info,seq_brks,seq_f,seq_d]=moninterp... 
%  (seq_info,seq_x,x,f,d) 
%  %  This function computes break points, derivatives, and 
%  %  functional values for monotone interpolants to the data 
%  %  specified by mon. sequences in seq_x. It also computes 
%  %  the length of each sequence & stores it in column 2 of 
%  %  seq_info. (Seq_info is assumed to have the 
%  %  classification of each sequence in column 1.) 
%  
%  written Thursday, June 28, 1990 
%  last modified Thursday, July 5, 1990 
%  
[lastseq, junk]=size(seq_x); 
clear junk 
lastx=length(x); 

interior=1; 
endsequence=2; 
criticalpt=3; 
criticalhor=4; 

for sindex=1:lastseq, 
    L=seq_x(sindex,1); 
    R=seq_x(sindex,2); 
    CLASS=seq_info(sindex,1); 
    lngth=R-L+1; 
    seq_info(sindex,2)=lngth; 

    if (lngth==1), 
        if (CLASS==criticalpt), 
            if (R<lastx), 
                direc=sign(f(R+1)-f(R)); 
            else 
                direc=sign(f(L)-f(L-1)); 
            end 
            seq_d(sindex,1)=abs(d(L))*direc(1); 
            seq_f(sindex,1)=f(L); 
            seq_brks(sindex,1)=x(L); 
        else 
            seq_d(sindex,1)=d(L); 
            seq_f(sindex,1)=f(L); 
        end 
    end 
end 

% Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
seq_brks(sindex,1)=x(L);
end

elseif (length==2)
    if (CLASS==criticalhor),
        direc=sign([f(L)-f(L-1),(f(R+1)-f(R))]);
        seq_d(sindex,1:2)=abs(d(L:L)).*direc(1:2);
        seq_f(sindex,1:2)=f(L:L);
        seq_brks(sindex,1:2)=x(L:L);
    else
        nknocksadd=0;
        [btemp, ftemp, dtemp, nknocksadd]=moncubic...
            (x(L:L),f(L:L),d(L:L),nknocksadd);
        seq_brks(sindex,1:length(btemp))=btemp;
        seq_f(sindex,1:length(ftemp))=ftemp;
        seq_d(sindex,1:length(dtemp))=dtemp;
        seq_info(sindex,2)=seq_info(sindex,2)+nknocksadd;
    end
else % length >=3
    if (CLASS==criticalhor),
        seq_brks(sindex,1:(R-L+1))=x(L:L);
        seq_f(sindex,1:(R-L+1))=f(L:L);
        seq_d(sindex,1:(R-L+1))=zeros(1,length);
    else
        [btemp, ftemp, dtemp, nknocksadd]=moncubic...
            (x(L:L),f(L:L),d(L:L),nknocksadd);
        seq_info(sindex,2)=seq_info(sindex,2)+nknocksadd;
        seq_brks(sindex,1:length(btemp))=btemp;
        seq_f(sindex,1:length(ftemp))=ftemp;
        seq_d(sindex,1:length(dtemp))=dtemp;
    end
end
end

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
function [D1,D2]=project(d1,d2,s1);

% This ftn is called by moncubic to adjust slopes so that
% they are in the ext'd monotony region. This adjustment
% is a relaxed, almost orthogonal projection into the
% Beatson-Wolcowicz extended monotonicity region.
%
% last modified July 9, 1990
%
if s1==0
    a=0; b=0;
else
    a = d1/s1; b = d2/s1; bm1 = b-1; am1 = a-1;
    if (a <=1) & (b >=4)
        l = 3/bm1;
        if (l<2/3),
            gl = l/2;
        else,
            gl = 2*l-1;
        end
        b = l+gl*bm1;
    elseif (b <=1) & (a >=4)
        l = 3/am1;
        if (l<2/3),
            gl = l/2;
        else,
            gl = 2*l-1;
        end
        a = l+gl*am1;
    elseif (a+b>3) & (a^2+b^2+a*b-6*(a+b)+9>0)
        l = 3*(a+b-2)/((a-1)^2+(b-1)^2+(a-1)*(b-1));
        if (l<2/3),
            gl = l/2;
        else,
            gl = 2*l-1;
        end
        a = 1+gl*am1;
        b = l+gl*bm1;
    end
end

% if all of above fail, then (a,b) is
% already in ext'd mon. region.
end

D1 = s1*a;
D2 = s1*b;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
function [xnew,ynew,dnew]=addknot(x,f,d,i,u,incr);

% This function adds a knot to the hermite data x,f,d.
% The new x-coord, u, must fall between x(i) and x(i+1).
% The new y-coordinate will be that of the cubic hermite
% interpolant at u, plus the value incr. The derivative
% of the cubic hermite interp to the new data, will be
% the same as that of the cubic hermite interp to the
% old data.
%
% last modified Monday, July 2, 1990
%

n=length(x);

if (u<=x(i))&(x(i+1)<=u),
    u=u
    i=i
    x=x
    error('u not between x(i) and x(i+1)');
end

% compute coefficients
%
dx=x(i+1)-x(i);
divdf1=(f(i+1)-f(i))/dx;
divdf3=d(i)+d(i+1)-2*divdf1;
a(1)=divdf3/(dx*dx);
a(2)=(divdf1-d(i)-divdf3)/dx;
a(3:4)=[d(i),f(i)];

% evaluate cubic hermite int to old data and its
derivative at the new knot
%
xtemp=u-x(i);
v=((a(1)*xtemp +a(2))*xtemp +a(3))*xtemp+a(4)+incr;
w=(3*a(1)*xtemp +2*a(2))*xtemp +a(3);

% construct modified x,y, and derivative sequences
%
xnew=[x(1:i),u,x(i+1:n)];
ynew=[f(1:i),v,f(i+1:n)];
dnew=[d(1:i),w,d(i+1:n)];
end
function stage=makgraph(seq_info, seq_brks, seq_d);

stage=makgraph(seq_info, seq_brks, seq_d);

% This ftn constructs the graph (without weights) that
% is used to locate a comonotone interpolant. The graph
% consists of 1 stage for each mono sequence of length
% one, and two stages for each mono seq of length two
% or more. Each stage consists of 2 verts, with all edges
% between two stages if they have consecutive indices,
% and no edges otherwise. The graph is stored as a
% 2-dim array. The row index=stage index, column 1
% contains the sequence index corresponding to this
% stage, cols 2 and 3 contain info concerning derivative
% estimates at the break point corres'g to this stage.
% Col 2 corres's to a non-negative estimate, and col
% 3 to a non-positive one. If an estimate of a
% given sign is not possible, then the corres'g col will
% be assigned a NaN (not-a-number) value.

% WRITTEN MONDAY, JULY 2, 1990
% Last modified Tuesday, July 3, 1990
%

interior=1; endsequence=2; criticalpt=3; criticalhor=4;

stageindex=1;

[numseqs,junk]=size(seq_brks);
clear junk

for seq_index=1:numseqs,
    length=seq_info(seq_index,2);
    CLASS=seq_info(seq_index,1);
    L=seq_brks(seq_index,1);
    R=seq_brks(seq_index,length);

    if (length==1), % add 1 stage

        dtemp=seq_d(seq_index,1);

        if (CLASS==criticalpt),
            % both pos & neg derivatives possible
            dtemp=abs(dtemp);
            stage(stageindex,1:3)=[seq_index,dtemp,-1*dtemp];
        else
            if (dtemp>0),
                stage(stageindex,1:3)=[seq_index,dtemp,NaN];
            elseif (dtemp==0),
....
stage(stageindex,1:3)=[seq_index,0,0];
else
    stage(stageindex,1:3)=[seq_index,NaN,dtemp];
end
end

else
    % length >=2 add 1 stage for both lft- & rt-most data pts
    dtemp=seq_d(seq_index,1);
    if (dtemp>0),
        stage(stageindex,1:3)=[seq_index,dtemp,NaN];
    elseif (dtemp==0),
        stage(stageindex,1:3)=[seq_index,0,0];
    else
        stage(stageindex,1:3)=[seq_index,NaN,dtemp];
    end
    stageindex=stageindex+1;
    dtemp=seq_d(seq_index,length);
    if (dtemp>0),
        stage(stageindex,1:3)=[seq_index,dtemp,NaN];
    elseif (dtemp==0),
        stage(stageindex,1:3)=[seq_index,0,0];
    else
        stage(stageindex,1:3)=[seq_index,NaN,dtemp];
    end
    end
    seq_index=seq_index+1;
    stageindex=stageindex+1;
end % for-loop
end
% [dleft,dright,cost]=labelgraph1...
% (seq_info,seq_x,x,seq_f,seq_d,stage)
% This ftn returns the weights for each edge of the graph
% representing the various co-monotone interpolants. It
% also ret's the modified deriv values corresponding to
% vert's at the left and rt of the edge. These are
% returned in 2 dimen arrays. For each of these, the
% row index corres's to the stage indx on the left of the
% edge-set, and the col indx indicates which of the edges
% in that stage is labeled. Cols 1-4 corres to ++, +-, 
% -+,-- cases, respectively.
% This is the version used for Algorithm 2.1, called 
% by COMONOTONE.
% written Monday, July 2, 1990
% last modified, Thursday, July 5, 1990
%
[numstages,m]=size(stage);
clear m
seq_indR=stage(1,1);
for stindex=1:numstages-1,
    stipl=stindex+1;
    seq_indL=seq_indR;
    seq_indR=stage(stipl,1);
    DLP=stage(stindex,2); % derivative left plus
    DLM=stage(stindex,3); % derivative left minus
    DRP=stage(stipl,2);  % etc.
    DRM=stage(stipl,3);

    if (seq_indL==seq_indR),
        % Case 1 ++
        if isnan(DLP)|isnan(DRP),
            weight(stindex,1)=inf;
        else
            weight(stindex,1)=0;
        end
        % Case 2 +- 
        if isnan(DLP)|isnan(DRM),

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
weight(stindex,2)=\inf;
else
  weight(stindex,2)=0;
end

% Case 3 —
if isnan(DLM) | isnan(DRP),
  weight(stindex,3)=\inf;
else
  weight(stindex,3)=0;
end

% Case 4 —
if isnan(DLM) | isnan(DRM),
  weight(stindex,4)=\inf;
else
  weight(stindex,4)=0;
end
else % data not from a single sequence

  lengthL=seq_info(seq_indL,2);
  XL=x(seq_x(seq_indL,2));
  XR=x(seq_x(seq_indR,1));
  FL=seq_f(seq_indL,lengthL);
  FR=seq_f(seq_indR,1);
  divdf=(FR-FL)/(XR-XL);
  
  if (isnan(DLP)), % if pos. slope impossible
    DL=DLM; % correct is negative
  elseif (isnan(DLM)), % if neg. slope impossible ...
    DL=DLP;
  else % if both OK, work from original % estimate.
    DL=seq_d(seq_indL,lengthL);
  end

  if (isnan(DRP)),
    DR=DRM;
  elseif (isnan(DRM)),
    DR=DRP;
  else
    DR=seq_d(seq_indR,1);
  end

% Case 1 ++
if isnan(DLP)|isnan(DRP),
    weight(stindex,1)=inf;
    dleft(stindex,1)=DLP;
    dright(stindex,1)=DRP;
else
    [DL2,DR2]=project(DLP,DRP,divdf);
    weight(stindex,1)=max(abs([[DL-DL2,DR-DR2]]));
    dleft(stindex,1)=DL2;
    dright(stindex,1)=DR2;
end

% Case 2 --
if isnan(DLP)|isnan(DRM),
    weight(stindex,2)=inf;
    dleft(stindex,2)=DLP;
    dright(stindex,2)=DRM;
elseif (DLP==0)|(DRM==0),
    [DL2,DR2]=project(DLP,DRM,divdf);
    weight(stindex,2)=max(abs([[DL-DL2,DR-DR2]]));
    dleft(stindex,2)=DL2;
    dright(stindex,2)=DR2;
else
    weight(stindex,2)=max(abs([[DL-DLP,DR-DRM]]));
    dleft(stindex,2)=DLP;
    dright(stindex,2)=DRM;
end

% Case 3 --
if isnan(DLM)|isnan(DRP),
    weight(stindex,3)=inf;
    dleft(stindex,3)=DLM;
    dright(stindex,3)=DRP;
elseif (DLM==0)|(DRP==0),
    [DL2,DR2]=project(DLM,DRP,divdf);
    weight(stindex,3)=max(abs([[DL-DL2,DR-DR2]]));
    dleft(stindex,3)=DL2;
    dright(stindex,3)=DR2;
else
    weight(stindex,3)=max(abs([[DL-DLM,DR-DRP]]));
    dleft(stindex,3)=DLM;
    dright(stindex,3)=DRP;
end

% Case 4 --
if isnan(DLM)|isnan(DRM),
    weight(stindex,4)=inf;
    dleft(stindex,4)=DLM;
    dright(stindex,4)=DRM;
else
    [DL2, DR2] = project(DLM, DRM, divdf);
    weight(stindex, 4) = max(abs([DL-DL2, DR-DR2]));
    dleft(stindex, 4) = DL2;
    dright(stindex, 4) = DR2;
end
end
end
function [dleft,dright,weight]=labelgraph...
    (seq_info,seq_x,x,seq_f,seq_d,stage)

%  [dleft,dright,cost]=labelgraph...
%      (seq_info,seq_x,x,seq_f,seq_d,stage)
%
%  This ftn returns the weights for each edge of the graph
%  representing the various co-monotone interpolants. It
%  also ret's the modified deriv values corresponding to
%  vert's at the left and rt of the edge. These are
%  returned in 2 dimen arrays. For each of these, the
%  row index corres's to the stage indx on the left of the
%  edge-set, and the col indx indicates which of the edges
%  in that stage is labeled. Cols 1-4 corres to ++, +-,
%  -+, -- cases, respectively.
%
%  This version used for Alghm 3.1, called by COMON2.
%
%  written Monday, July 2, 1990
%  last modified, Thursday, July 5, 1990
%
[numstages,m]=size(stage);
clear m
seq_indR=stage(1,1);
for stindex=1:numstages-1,
    stipl=stindex+1;
    seq_indL=seq_indR;
    seq_indR=stage(stipl,1);
    DLP=stage(stindex,2);  % derivative left plus
    DLM=stage(stindex,3);  % derivative left minus
    DRF=stage(stipl,2);    % etc.
    DRM=stage(stipl,3);
    if (seq_indL==seq_indR),
        % Case 1 ++
        if isnan(DLP)|isnan(DRP),
            weight(stindex,1)=inf;
        else
            weight(stindex,1)=0;
        end
    end
% Case 2 +-  
if isnan(DLP) | isnan(DRM),  
    weight(stindex, 2) = inf;  
else  
    weight(stindex, 2) = 0;  
end  

% Case 3 ++  
if isnan(DLM) | isnan(DRP),  
    weight(stindex, 3) = inf;  
else  
    weight(stindex, 3) = 0;  
end  

% Case 4 --  
if isnan(DLM) | isnan(DRM),  
    weight(stindex, 4) = inf;  
else  
    weight(stindex, 4) = 0;  
end  

else % data not from a single sequence  

lngthL = seq_info(seq_indL, 2);  
XL = x(seq_x(seq_indL, 2));  
XR = x(seq_x(seq_indR, 1));  
FL = seq_f(seq_indL, lngthL);  
FR = seq_f(seq_indR, 1);  
divdf = (FR - FL) / (XR - XL);  

% % compute orig deriv estimate, with corrected sign  
%  
if (isnan(DLP)),  % if pos. slope impossible  
    DL = DLM;  % correct is negative  
elseif (isnan(DLM)), % if neg. slope impossible ...  
    DL = DLP;  
else  % if both possible, work from % original estimate.  
    DL = seq_d(seq_indL, lngthL);  
end  

if (isnan(DRP)),  
    DR = DRM;  
elseif (isnan(DRM)),  
    DR = DRP;  
else  
    DR = seq_d(seq_indR, 1);  
end
% Case 1 ++
if isnan(DLP) | isnan(DRP),
    weight(stindex,1)=inf;
    dleft(stindex,1)=DLP;
    dright(stindex,1)=DRP;
else
    [DL2,DR2]=project(DLP,DRP,divdf);
    weight(stindex,1)=weightftn...
    ([XL,XR],[FL,FR],[DL,DR],[DL2,DR2]);
    dleft(stindex,1)=DL2;
    dright(stindex,1)=DR2;
end

% Case 2 +- 
if isnan(DLP) | isnan(DRM),
    weight(stindex,2)=inf;
    dleft(stindex,2)=DLP;
    dright(stindex,2)=DRM;
elseif (DLP==0) | (DRM==0),
    [DL2,DR2]=project(DLP,DRM,divdf)
    weight(stindex,2)=weightftn...
    ([XL,XR],[FL,FR],[DL,DR],[DL2,DR2]);
    dleft(stindex,2)=DL2;
    dright(stindex,2)=DR2;
else
    weight(stindex,2)=weightftn...
    ([XL,XR],[FL,FR],[DL,DR],[DL,DRM]);
    dleft(stindex,2)=DLP;
    dright(stindex,2)=DRM;
end

% Case 3 -+
if isnan(DLM) | isnan(DRP),
    weight(stindex,3)=inf;
    dleft(stindex,3)=DLM;
    dright(stindex,3)=DRP;
elseif (DLM==0) | (DRP==0),
    [DL2,DR2]=project(DLM,DRP,divdf)
    weight(stindex,3)=weightftn...
    ([XL,XR],[FL,FR],[DL,DR],[DL2,DR2]);
    dleft(stindex,3)=DL2;
    dright(stindex,3)=DR2;
else
    weight(stindex,3)=weightftn...
    ([XL,XR],[FL,FR],[DL,DR],[DLM,DRP]);
    dleft(stindex,3)=DLM;
    dright(stindex,3)=DRP; end
% Case 4 --
if isnan(DLM) | isnan(DRM),
    weight(stindex,4) = inf;
    dleft(stindex,4) = DLM;
    dright(stindex,4) = DRM;
else
    [DL2, DR2] = project(DLM, DRM, divdf);
    weight(stindex,4) = weightftn...
        ([XL, XR], [FL, FR], [DL, DR], [DL2, DR2]);
    dleft(stindex,4) = DL2;
    dright(stindex,4) = DR2;
end
end
end
function weight=weightftn(x,f,dh,ds);
% weight=weightftn(x,f,dh,ds);
% This ftn returns the weight for the edge of the graph
% corresponding to the data points stored in x and f.
% dh and ds are the end point deriv's of the piecewise
% cubic interpolants. dh is the hermite interp for the
% initial fit; ds is the hermite interp for the
% comonotone fit. Norm over [x(i-1),x(i+2)] is estimated
% by norm over [x(i),x(i+1)].
%
% <Step 1> Compute new knot if any is added.
% [xnew,fnew,dnew,newknot]=moncubic(x,f,ds);
if(newknot>0),
% <Step 2a> If new knot added, update f
% dx=diff(x);
df1=diff(f)/dx;
df3=dh(2)+dh(1)-2*df1;
a(4)=f(1);
a(3)=dh(1);
a(2)=(df1-d(1)-df3)/dx;
a(1)=df3/(dx*dx);
t=x(2)-xnew(2);
f=[f(1),a(1),f(2)];

for i=2:4,
    fh(2)=fh(2)*t+a(i);
end
%
% <Step 3a> Compute norm on two subintervals & take max
% d=[dh(1),dnew(2),dh(2)]-dnew % yes, d(2)=0 !!
f=f-fnew;
weight=max(derivnorm(xnew(1:2),f(1:2),d(1:2)),
        derivnorm(xnew(2:3),f(2:3),d(2:3))
else
% <Step 2a> If no new knot, compute norm on single
% interval
    weight=derivnorm(x,[0,0],(dh-ds));
end
end;
function norm=derivnorm(x,f,d);
    
    % This ftn returns the maximum value of the derivative
    % of the cubic hermite interpolant to the data x,f,d.
    
    %<STEP ONE> Compute coefficients of hermite interpolant
    dx=diff(x);
dfl=diff(f)/dx;
df3=d(2)+d(1)-2*df1;
    a(3)=d(1);
a(2)=(df1-d(1)-df3)/dx;
a(1)=df3/(dx*dx);
    
    %<STEP TWO> Find interior crit pnt of the derivative, if any
    if (a(1)==0), cp=-a(2)/(3*a(1)); end
    
    %<STEP THREE> Evaluate deriv at each critical point, and take max
    if (a(1)==0) & (x(1)<cp) & (cp<x(2)),
        norm=(3*a(1)*cp+2*a(2)*cp)+a(3);
    else
        norm=0;
    end
    
    norm=max(abs([d(1),norm,d(2)]));
end;
function [cost,pathdata]=mincostpath(weight);
% 
% [cost,pathdata]=mincostpath(weight);
% 
% This function returns a minimum cost path and its
% associated cost from the first to the last stage in
% the graph representing the co-monotone interpolants.
% Each entry in the vector 'pathdata' is a '1' or a '2',
% corresponding to either the positive or negative
% choice of slope at that stage, respectively.
% 
% Written Monday, July 3, 1990
% Last modified, Wednesday, July 4, 1990
% 
% first label each vert with the min cost to reach it, and
% the previous vert on a minimum cost path to that vertex.
% 
[nedgesets,junk]=size(weight);
numstages=nedgesets+1;
clear junk;
clear nedgesets;

previous(1,1:2)=[NaN,NaN]; % prev. vertices on a min cost
                         % path.
                         % col 1 is for positive vert.
                         % col 2 for neg. (NaN for first
                         % vertex on path).

costsofar(1,1:2)=[0,0]; % cost to get to corresponding
                       % vertex.
                       % col 1 is pos.

for stindex=2:numstages,
    edgeindx=stindex-1;
    %
    % Label positive vertex
    %
    costfrml=max(costsofar(edgeindx,1),weight(edgeindx,1));
    costfrm2=max(costsofar(edgeindx,2),weight(edgeindx,3));
    if (costfrml<=costfrm2),
previous(stindex,1)=1;
costsofar(stindex,1)=costfrm1;
else
  previous(stindex,1)=2;
costsofar(stindex,1)=costfrm2;
end;

% % Label negative vertex
% costfrm1=max(costsofar(edgeindx,1),weight(edgeindx,2));
costfrm2=max(costsofar(edgeindx,2),weight(edgeindx,4));
if (costfrm1<=costfrm2),
  previous(stindex,2)=1;
costsofar(stindex,2)=costfrm1;
else
  previous(stindex,2)=2;
costsofar(stindex,2)=costfrm2;
end;
end

% % Now back-track from last stage to get min-cost path
%
pathdata=zeros(1,numstages);
if (costsofar(numstages,1)<=costsofar(numstages,2)),
  pathdata(numstages)=1;
cost=costsofar(numstages,1);
else
  pathdata(numstages)=2;
cost=costsofar(numstages,2);
end
stipl=numstages;
for stindex=(numstages-1):-1:1,
  pathdata(stindex)=previous(stipl,pathdata(stipl));
  stipl=stindex;
end;
end
function [t,ft,dt]=assemble(seq_info,seq_brks,...
    seq_f,seq_d,pathdata,dleft,dright,stage);

% [t,ft,dt]=assemble(seq_info,seq_brks,...
%    seq_f,seq_d,pathdata,dleft,dright,stage)

% This ftn assembles the hermite data for the comonotone
% interpolant from the path through the graph.
%
% Written Wednesday, July 4, 1990.
% Last Modified, Thursday, July 5.
%
nstages=size(stage);
nstages=nstages(1); % throw away # columns
Lstage=1; % stage at left of current edge-set
%
% <STEP 1> Use path to modify derivative sequences
%
for Rstage=2:nstages,
    edge=2*(pathdata(Lstage)-1)+pathdata(Rstage);
    Lseq=stage(Lstage,1);
    Llength=seq_info(Lseq,2);
    Rseq=stage(Rstage,1);

    if(Lseq<Rseq), % no changes inside a monotone sequence
        if(Llength==1) & (Rseq>2), % may be projected on
            either
                % left or right.
                d1=dright(Lstml,prevedge);
                d2=dleft(Lstage,edge);
                seq_d(Lseq,Llength)=sign(d1)*min(abs([d1,d2]));
                %sign(d1)=sign(d2)
            else
                seq_d(Lseq,Llength)=dleft(Lstage,edge); % left
                stage index
                    % = edge-set
                index
            end
        end
    end

    seq_d(Rseq,1)=dright(Lstage,edge);
end
Lstml=Lstage;
Lstage=Rstage;
prevedge=edge;
end
%
% <STEP 2> Construct interpolants between sequences

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
%       ("joints"), and assemble pieces.
%
%
%       <STEP 2-a> Start by adding first sequence
%
lngh=seq_info(1,2);
t(1:lngh)=seq_brks(1,1:lngh);
ft(1:lngh)=seq_f(1,1:lngh);
dt(1:lngh)=seq_d(1,1:lngh);
nsegs=size(seq_brks);
nsegs=nsegs(1);
L=lngh+1; % L points to next available location in t.
previ=1; % previ points to sequence processed one step back
prevl=lngh; % prevl keeps track of length of previous sequence
%
%       <STEP 2-b> For 2nd thru next-to-last seq,
%       add the joint on the left, and the
%       interior of the current sequence.
%
for sindex=2:nsegs;
%
      Add joint on left
%
      xtemp=[seq_brks(previ,prevl),seq_brks(sindex,1)];
ftemp=[seq_f(previ,prevl),seq_f(sindex,1)];
dtemp=[seq_d(previ,prevl),seq_d(sindex,1)];
      [xtemp,ftemp,dtemp,mlngth]=...
      joint(xtemp,ftemp,dtemp,mlngth);
R=L+mlngth-2;
t(L:R)=xtemp(2:mlngth);
ft(L:R)=ftemp(2:mlngth);
dt(L:R)=dtemp(2:mlngth);
%
      Add interior of next sequence
%
      L=R+1;
lngh=seq_info(sindex,2);

if(lngh>1), % lngh=1 ==> only end point, added above
      R=L+lngh-2;
      t(L:R)=seq_brks(sindex,2:lngh);
      ft(L:R)=seq_f(sindex,2:lngh);
      dt(L:R)=seq_d(sindex,2:lngh);
      L=R+1;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
end

previ=sindex;
prevl=lnghth;
end
end
function [X,Y,d,lngth]= joint(X,Y,d,lngth);
%
% [X,Y,d,lngth]= joint(X,Y,d,lngth)
%
% This ftn determines whether the joint between adjacent
% monotone seq's is monotone or not. If so, it will add
% a knot if necessary, the deriv values are assumed
% to already be corrected. If the joint is not mono, the
% data will be unchanged. 'lngth' is the length of the
% arrays X,Y, and d.
%
% Written Wednesday, July 4, 1990.
%
% lnghth=length(X);
if (lngth~=2) | (lngth~=length(Y)) | (lngth~=length(d)),
    error('the length of the X,Y, and d arrays must be 2!');
end
%
% If signs of derivative data differ,
if(sign(d(1))\times sign(d(2))>=0), \% the joint is not mono, so
% exit w/o more computation
    dx= diff(X);
    S=diff(Y)./dx;
    if (S==0),
        d=[0,0];
    else,
        a=d(1)./S;
        b=d(2)./S;
        if ((a+b)>3)&(a^2+b^2+a*b-6*(a+b)+9 >0) % Then add a knot.
            lnghth=lngth+1;
            delta= dx(1)*(2*a+b-3)/(3*(a+b-2));
            epsilon=S*(((2*a+b-3)^2/(3*(a+b-2)))-a);
            yincr= 4*epsilon*delta/3;
        end
    end
elseif(b<1) \% lower right corner.
    % if neither of these, test is
    % detecting round-off error only
        delta= dx-delta;
        u= X(2)-2*delta;
        [X,Y,d]=addknot(X,Y,d,1,u,-1*yincr);
end
end
end

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
function pp=hermite(breaks,f,s);
%   pp=hermite(breaks,f,s);
%   This ftn returns a piece-wise polynomial function of
%   order 4 with break points (knots) as given in breaks,
%   interpolating the functional values in f and the first
%   derivative values in s (slopes).
%   That is, pp will be the cubic hermite interpolant to the
%   given data.
%   
% n=length(f)-1;   % number of subintervals

if(length(s)-1~=n),
    fprintf('The number of functional values and the number');
    error ('of slope values must be equal!');
end

for i=1:n
    dx=breaks(i+1)-breaks(i);
    coef(i,3:4)=[s(i),f(i)];
    divdf1=(f(i+1)-f(i))/dx;
    divdf3=s(i)+s(i+1)-2*divdf1;
    coef(i,2)=(divdf1-s(i)-divdf3)/dx;
    coef(i,1)=divdf3/(dx*dx);
end
pp=mkpp(breaks,coef);
APPENDIX B.

MatLab Code for Coconvex Interpolation
COCON SCRIPT FILE

SECOND ORDER SLOPE AVERAGE

This script returns a pw polynomial interpolating the data \((x_i, f_i)\), with a min # of changes in sign in the second derivative. The interp has 1 contin deriv, and is composed of quadratic and cubic pieces. The interpolants conve to an underlying function with 2nd order (third on "nice" subintervals).

This script uses a two-sided (2nd-order) slope average as in cubic bessel interp. A related scrip COCON2 uses another average.

written Saturday, June 16, 1990

INITIALIZATION

n=length(x);
if (length(f)==n), fprintf('error from coconvex');
   error('the number of x- and y- coordinates not equal');
end

if n<=1,
   error('cannot interpolate \(<= 1\) point!');
return
end

if (n<=3),
   coef=polyfit(x,f,3);
   breaks=[x(1),x(n)];
   pp=mkpp(breaks,coef);
   return
end
lastx=n; % number of data points
n=n-1; % n is now the number of sub-intervals
breaks=[];
coef=[];
linflag=0;

% PROCESS FIRST INTERVAL
%
for i=2:3,
    iminus1=i-1;
    dx(i)=x(i)-x(iminus1);
    divdf(i)=(f(i)-f(iminus1))/dx(i);
end

dxsum=dx(2) + dx(3);
xindex=1;
knotindex=1;
delta2=divdf(3)-divdf(2);
deltal=delta2;
linear=(delta2==0);

if (linear),
    stillinear=1;
    k=3;

    while (k<lastx) & (stillinear),
        km1=k;
        k=k+1;
        nextdd=(f(k)-f(km1))/(x(k)-x(km1));
        stillinear=(divdf(2)==nextdd);
    end

    if (~stillinear), k=kml; end % k points to end of % linear region
    breaks=x(1);
    coef=[0 0 divdf(2) f(1)];
knotindex=2;
xindex=k;
s(2)=divdf(2); % initialize s for main loop
else % not linear
    s(1)=(divdf(2)*(2*dx(2)+dx(3))-divdf(3)*dx(2))/dxsum;
    ip2=3;
    LOOKAHEAD;

    if(linflag==1),
        s(2)=divdf(3);
    else
        s(2)=slopeavel([dx(2:3)],[divdf(2:3)]);
    end

    ip1=2;
    FLEX;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
% MAIN LOOP BEGINS HERE
%

while (xindex<n),
    dx(1)=x(xindex)-x(xindex-1);
    divdf(1)=divdf(2);
    ip1=xindex+1; ip2=xindex+2;
    dx(2)=x(ip1)-x(xindex);
    divdf(2)=(f(ip1)-f(xindex))/dx(2);
    dx(3)=x(ip2)-x(ip1);
    divdf(3)=(f(ip2)-f(ip1))/dx(3);
    dxsum=dx(2)+dx(3);
    s(1)=s(2);
    delta1=divdf(2)-divdf(1);
    delta2=divdf(3)-divdf(2);
    linear=(delta2==0);
    if (linear),
        FLAT1;
    else
        LOOKAHEAD;
        if (linflag==1),
            s(2)=divdf(3);
        else
            s(2)=slopeatl([dx(2:3)],[divdf(2:3)]);
        end
    end
    FLEX;
end

end % While loop

if (xindex > n),
    breaks(knotindex)=x(lastx);
    pp=mkpp(breaks,coef);
    return
end
%

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
% PROCESS INTERVAL N
%
\( dx(1) = x(x_{index}) - x(x_{index-1}); \)
\( \text{divdf}(1) = \text{divdf}(2); \)
\( dx(2) = x(\text{lastx}) - x(x_{index}); \)
\( \text{divdf}(2) = (f(\text{lastx}) - f(x_{index}))/dx(2); \)
\( s(1) = s(2); \)
% slope at right end point approximated by local quadratic
\( s(2) = \text{divdf}(2) + (\text{divdf}(2) - \text{divdf}(1)) * dx(2)/(dx(2)+dx(1)); \)
\( \text{ipl} = x_{index} + 1; \)
\( \text{deltal} = \text{delta2}; \)
FLEX;
\( \text{breaks(knotindex)} = x(\text{lastx}); \)
\( \text{pp} = \text{mkpp(breaks, coef)}; \)
end;
COCON2 SCRIPT FILE
GEOMETRIC SLOPE AVERAGING

This script returns a piecewise polynomial interpolating the data \((x_i, f_i)\), with a min number of changes in sign in the 2nd derivative. The interpolant has 1 cont deriv, and is composed of quadratic and cubic pieces. The interpolants converge to an underlying ftn with second order (third on "nice" subintervals).

This script uses as the average slope, the slope of the line bisecting the angle formed by the lines passing through the data on consecutive subintervals. See the script slopeav.

written Saturday, June 16, 1990

INITIALIZATION

```
n = length(x);
if (length(f)~=n), fprintf('error from coconvex');
    error('the number of x- and y- coordinates not equal');
end
if n<=1,
    error('cannot interpolate <= 1 point!');
    return
end
if (n<=3),
    coef=polyfit(x,f,3);
    breaks=[x(1),x(n)];
    pp=mkpp(breaks,coef);
    return
end
lastx=n; % number of data points
n=n-1; % n is now the number of sub-intervals
breaks=[];
coef=[];
linflag=0;
```

% PROCESS FIRST INTERVAL
for i=2:3,
  iminus1=i-1;
  dx(i)=x(i)-x(iminus1);
  divdf(i)=(f(i)-f(iminus1))/dx(i);
end

dxsum=dx(2) + dx(3);
xindex=1;
knotindex=1;
delta2=divdf(3)-divdf(2);
delta1=delta2;
linear=(delta2==0);

if (linear),
  stillinear=1;
  k=3;

  while (k<lastx) & (stillinear),
    kml=k;
    k=k+1;
    nextdd=(f(k)-f(kml))/(x(k)-x(kml));
    stillinear=(divdf(2)==nextdd);
  end

  if (~stillinear),  k=kml;  end  %  k  points  to  end  of
  %  the  linear  region

  breaks=x(1);
  coef=[0 0  divdf(2)  f(1)];
  knotindex=2;
  xindex=k;
  s(2)=divdf(2);  %  initialize  s  for  main  loop

else  %  not  linear
  s(1)=(divdf(2)*(2*dx(2)+dx(3))-divdf(3)*dx(2))/dxsum;
  ip2=3;
  LOOKAHEAD;

  if(linflag==1),
    s(2)=divdf(3);
  else
    s(2)=slopeave2([dx(2:3)], [divdf(2:3)]);
  end

  ip1=2;
  FLEX;
end
while (xindex < n),

   dx(1) = x(xindex) - x(xindex-1);
   divdf(1) = divdf(2);
   ip1 = xindex + 1; ip2 = xindex + 2;
   dx(2) = x(ip1) - x(xindex);
   divdf(2) = (f(ip1) - f(xindex))/dx(2);
   dx(3) = x(ip2) - x(ip1);
   divdf(3) = (f(ip2) - f(ip1))/dx(3);
   dxsum = dx(2) + dx(3);
   s(1) = s(2);
   delta1 = divdf(2) - divdf(1);
   delta2 = divdf(3) - divdf(2);

   linear = (delta2 == 0);

   if (linear),
      FLAT;
   else
      LOOKAHEAD;

      if (linflag == 1),
         s(2) = divdf(3);
      else
         s(2) = slopeave2([dx(2:3)], [divdf(2:3)]);
      end

      FLEX;
   end
end % While loop

if (xindex > n),
   breaks(knotindex) = x(lastx);
   pp = mkpp(breaks, coef);
   return
end

% % PROCESS INTERVAL N
\% \ dx(1)=x(xindex)-x(xindex-1);
\divdf(1)=\divdf(2);
\dx(2)=x(lastx)-x(xindex);
\divdf(2)=(f(lastx)-f(xindex))/\dx(2);
\s(1)=s(2);
% slope at right end point approximated by local quadratic
\s(2)=\divdf(2) + (\divdf(2)-\divdf(1))*\dx(2)/(\dx(2)+\dx(1));
\ipl=xindex+1;
deltal=delta2;
FLEX;
breaks(knotindex)=x(lastx);
pp=mkpp(breaks, coef);
end;
% FLAT1 SCRIPT FILE
% This script computes a linear interpolant over a region
% where the data is linear, except in the single special
% case where the data requires a point of inflection.
% There a piecewise cubic is constructed.
% This script calls "slopeavel" script file, for a second
% order two-sided slope average. Change these calls to
% slopeave2 to get geometric slope averaging--FLAT2
% (see slopeave2).
% Current version as of Saturday, June 16, 1990
% stillinear=1;
% k=xindex+2;

while (k < lastx) & (stillinear),
    kml = k;
    k = k + 1;
    newdd = (f(k) - f(kml)) / (x(k) - x(kml));
    stillinear = (divdf(2) == newdd);
end

if (~stillinear), k = k - 1; end % k points to end of linear region

kpl = k + 1;

if (kpl <= lastx),
    divdfend = (f(kpl) - f(k)) / (x(kpl) - x(k));
    deltaend = divdfend - divdf(2);
else
    deltaend = 0;
end

if (k - xindex == 2) & (sign(deltaend) * sign(delta2) < 0),
    s(1) = s(2);
    s(2) = divdf(1) +
        (divdf(2) - divdf(1)) * (2 * dx(2) + dx(1)) / dxsum;
    FLEX;
    LOOKAHEAD;
    if (linflag == 1),
        s(2) = divdfend;
    else
        s(2) = slopeavel([dx(2), newdx], [divdf(2), newdd]);
end
end
FLEX;
else
  if (coef(knotindex-1,1:2)==[0 0]),
    % check to see if coming off linear region
    s(1)=divdf(1);
    s(2)=divdf(3);
    range=[xindex xindex+1];
    breaks(knotindex)=x(xindex);
    coef(knotindex, 1:4)=hermcoef(x(range),f(range),s);
    knotindex=knotindex + 1;
    xindex=xindex + 1;
  end
  breaks(knotindex)=x(xindex);
  coef(knotindex,:)=[0 0 divdf(2) f(xindex)];
  xindex=k;
  knotindex=knotindex+1;
  s(2)=divdf(2);
end
end % FLAT1
This script builds a pwise cubic or quad interp w/out points of infl over the int \([x(i), x(i+1)]\). In the case of the p.w. quad, a knot is added. Knotindex is assumed to point to the next avail location in breaks at calling. At exit, knotindex points to the next available location in breaks.

Current version as of Saturday, June 16, 1990

\[
A = \text{hermcoef}(x(xindex:ipl), f(xindex:ipl), s); \\
\text{allowinfl} = (\text{delta1*delta2} \leq 0); \\
\text{noinflection} = ((3*A(1)*dx(2) + A(2))*A(2) \geq 0); \\
\text{if} (\text{noinflection}) | (\text{allowinfl}), \\
\quad \text{coef(knotindex,:) = A;} \\
\quad \text{breaks(knotindex)} = x(xindex); \\
\quad \text{knotindex} = \text{knotindex} + 1; \\
\text{else} % insert knot and interpolate with quadratic pieces. \\
\quad \text{newx} = ((f(ipl) - f(xindex)) + s(1)*x(xindex) - s(2)*x(ipl)) / (s(1) - s(2)); \\
\quad \text{newy} = (s(1) + \text{divdf}(2)) * (\text{newx} - x(xindex)) / 2 + f(xindex); \\
\quad \text{slopes} = [s(1), \text{divdf}(2)]; \\
\quad \text{coef(knotindex,:)} = \text{hermcoef}([x(xindex), newx], \\
\quad \quad [f(xindex), newy], slopes); \\
\quad \text{breaks(knotindex:knotindex+1)} = [x(xindex), newx]; \\
\quad \text{knotindex} = \text{knotindex} + 1; \\
\quad \text{coef(knotindex,:) = hermcoef([newx, x(ipl)], [newy, ... \\
\quad \quad f(ipl)], [\text{divdf}(2), s(2)]);} \\
\quad \text{knotindex} = \text{knotindex} + 1; \\
\text{end xindex=ipl; \\
\text{end % flex}
LOOKAHEAD SCRIPT FILE

Current version as of Saturday, June 16, 1990

This script looks ahead to determine if the next interval will be interpolated by a linear interpolant. If so, "linflag" is set to 1, otherwise it is set to 0.

ip3=xindex+3; ip4=xindex+4;

if (lastx < ip3)
  linflag=0;
else
  divdf(4)=(f(ip3)-f(ip2))/(x(ip3)-x(ip2));
  delta=divdf(4)-divdf(3);

if(delta~=0),
  linflag=0;
else
  if (lastx < ip4),
    linflag=1;
  else
    divdf(5)=(f(ip4)-f(ip3))/(x(ip4)-x(ip3));
    delta=divdf(5)-divdf(4);

    if (delta*delta2>=0),
      linflag=1;
    else
      linflag=0;
  end
end
end
end % of LOOKAHEAD
function ave=slopeavel(dx,divdf);
% This function returns a weighted mean of the divided
% differences given by divdf(1:2) with corresponding x
% increments x(1:2). This is the usual second-order
% approxtn to \( f' \), as used in cubic bessel interpolation.
% written Saturday, June 16, 1990
%    ave=(divdf(1)*dx(2)+divdf(2)*dx(1))/(dx(1)+dx(2));
end;

function ave=slopeave2(dx,divdf);
% This ftn averages slopes so that a line segment with
% the average slope will bisect the angle made by line
% segments with slopes divdf(1) and divdf(2). The 'dx'
% argument is not used, but is included for convenience of
% coding (it matches the form of 'slopeavel').
% written Saturday, June 16, 1990
%    ave=tan((atan(divdf(1))+atan(divdf(2)))/2);
end;
BIBLIOGRAPHY


Pence, Dennis, Unpublished manuscript 1987.


