CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY

RESEARCH REPORT

Monotonicity in Interpolation and Approximation

Pavel Mrázek

mrazekp@cmp.felk.cvut.cz

№ K335-CMP/99/179

January 15, 1999

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/mrazek/Mrazek-RR-99-179.ps.gz

This research was supported by the Grant Agency of the Czech Republic, grants 102/97/0480 and 201/97/0437, by the grant Aktion Österreich-Tschechien 16p12, and by the Czech Ministry of Education, grant VS96049.

Center for Machine Perception, Department of Control Engineering
Faculty of Electrical Engineering, Czech Technical University
Karlovo náměstí 13, 121 35 Prague 2, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7465, www: http://cmp.felk.cvut.cz

Abstract

Motivated by processing of data from colour-coded range finder, an active computer vision system for 3D reconstruction of a scene, we study the problems relative to monotonicity in interpolations and approximations. First part of the paper concentrates on interpolations, discusses their desired properties, such as linearity, smoothness or comonotonicity, and presents several interpolation methods. We deal with approximation to ensure (piecewise) monotonicity in the second part of the text. Previous works on monotone approximation are reviewed, a linear and nonlinear smoothing theories introduced after an analysis of the experimental data properties. Then we present an approximation method consisting of edge detection (to prevent inter-region smoothing) combined with approximation through weighted averaging in a locally adaptive neighbourhood (to remove noise and ensure monotonicity), and suggest areas for further research.

Acknowledgements

I would like to express my gratitude to Mirko Navara who inspired many of the ideas found in the text, and gave me a lot of advice and valuable comments on the manuscript.

Work on the project would be also much more difficult without the help of other members of the Center for Machine Perception and the friendly and motivating atmosphere there.

Last but not least, I would like to acknowledge the support of the Grant Agency of the Czech Republic, grants 102/97/0480 and 201/97/0437, of the grant Aktion Österreich-Tschechien 16p12, and of the Czech Ministry of Education, grant VS96049.

Pavel Mrázek, December 1998

Contents

1

Introduction

Reconstruction of the full three-dimensional shape of real objects from visual information, i.e. from essentially two-dimensional data, belongs to main domains of computer vision applications. Only a brief introduction of all the methods which have been employed would result in a long text, we will mention here only a few (possibly the most) important groups.

The main representative of passive vision, *stereo*, starts from (at least) two images of a scene, taken from different positions (either by two or more cameras, or by one moving camera). Then from the known change of camera positions between the two images and by finding the corresponding points (belonging to the same object/feature) in both images it is possible to compute angles from cameras to the point in 3D space and thus reconstruct its 3D position.

Stereo is intrinsically a difficult task, partly because of the computationally expensive search for corresponding points in different images. Where possible, it is helpful to restrict the generality of the approach and try a simpler one. So *active vision* techniques came into play for moderately sized scenes; with them, the correspondence problem is avoided by introducing special light conditions artificially. Various structured light types can be used. Perhaps the most commonly used *laser-plane range finder* lights the observed scene by a single plane of laser light, a single strip of light is then seen in the camera. From the known mutual position of laser and camera and the position of bright pixels in the image two angles are obtained and the 3D shape of the scene is reconstructed.

The major advantage of this approach lies in its simplicity and accuracy (see [29]); on the other hand it reveals a drawback that a single plane is illuminated by the laser light at one moment, and only points residing in that plane can be reconstructed from a single image. Many images (which implies longer time) are needed to capture a complete scene, which makes the method unusable for any moving, non-rigid or non-stationary objects.

A different lighting scheme may provide a better solution. In [4] several possibilities are reviewed and a colour light coding method (using projection of several planes) developed as a compromise between accuracy of the reconstruction and the acquisition period.

We would like to accomplish the reconstruction from a single image and so employed colour coded illumination in a system called *rainbow range finder* [42, 38, 39]: the light projected into the scene forms a rainbow with the wavelength

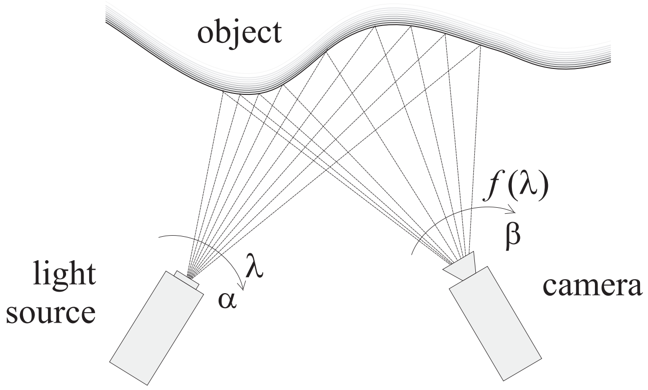

Figure 1.1: *Rainbow range finder - a system for 3D reconstruction by active vision: light of monotonically increasing wavelength is projected into the scene and full 3D shape is reconstructed from the colour image observed by the camera.*

increasing in one direction (see figure 1.1 for a sketch of the setting).

As the colours are created by decomposing white light by an interference filter, each plane consists of light of a single wavelength; then the colour in the image observed by the camera is not affected by reflections from the scene and should also exhibit the property of monotonically increasing wavelength in one direction, at least piecewise, with discontinuities on object boundaries. We should be able to assign a point in the image to a single plane passing through the light projector uniquely, based only on the observed colour of that point. The plane's angle from the projector, α, together with the point's location in the image, corresponding to angle β, and with the relative position of the projector and the camera determines the 3D, real-world position of that point.

The unique and precise identification of the point's position from its observed colour is complicated by an imprecision in colour measurement and by noise present in the experiment. The data which should be increasing in one direction do not always comply with this rule. We concentrate on the noise and violated monotonicity problem in our paper. The main task could be formulated as follows: filter (or approximate) the experimental, two-dimensional data to restore their desired (piecewise) monotonicity. Unfortunately, this task is far from trivial; the classical methods (such as polynomial or spline interpolation), and, more generally speaking, no method linearly dependent on the input data can guarantee monotonicity of the approximation. Moreover, as the input image may consist of only piecewise continuous data (corresponding to multiple objects in the scene) and we do not want to approximate across discontinuities, we will often have to approximate data forming irregular, non-rectangular regions extracted from the two-dimensional data.

The structure of this paper is as follows: chapter 2 summarizes some ideas and questions we encountered when dealing with interpolations in one and two dimensions, describes desirable properties of a good interpolation, and discusses several methods. Chapter 3 concentrates on an application of constrained approximation: previous work in monotone approximation is reviewed, we supply introduction to edge detection, and offer an insight into the quickly developing field of linear and nonlinear diffusion applied to image processing. Later on in the text we try to employ the ideas of the preceding parts to develop a nonlinear, locally adaptive filter to remove noise and restore the desired properties of the data, with the focus on monotonicity. The paper is concluded in chapter 4, where we also suggest the directions of further development.

2

On the Search for Optimal Interpolation Method

2.1 Introduction and interpolation properties

Interpolation is one of the very old tasks of numerical analysis. Various methods are described in the textbooks (see e.g. [17] or algorithmically oriented [34]), most of them based on fitting a polynomial curve to the input points. Although the classical methods pretend to solve the problem, we find them inappropriate in some physically oriented applications and have therefore tried to face the problem from a different viewpoint. The distant goal we kept in mind was to create an interpolation which would not create any local extrema not present in the input data. This can be also understood as a monotonicity-preservation constraint, monotonically increasing data should be interpolated by monotonically increasing function. In our case this motivation originated in the processing of monotonic data from rainbow range finder[1].

The task formulation in one dimension follows. Let us have a set of input data, measured at n discrete points. The vectors $\vec{x}$ and $\vec{y}$ store the $x-$ and $y-$coordinates of these input points (also called *knots*)

$$\begin{aligned} \vec{x} &= (x_1, x_2, \ldots, x_n) \\ \vec{y} &= (y_1, y_2, \ldots, y_n) \end{aligned} \tag{2.1}$$

We would like to interpolate the points, i.e. to construct a suitable function $F(t, \vec{x}, \vec{y})$ to run through the input data, where t is a parameter covering the interval of input values x_i. Figure 2.1 illustrates the situation. For exact interpolation, the function F has to comply with the equation

$$F(x_i, \vec{x}, \vec{y}) = y_i \tag{2.2}$$

There are many possibilities how to construct the interpolating function F, different solutions will be preferred for the same data under different presump-

[1] Actually the rainbow range finder data are monotonic only if no noise is present, which is not a realistic assumption. The ideas presented in this and the following sections have been employed recently to construct a filter which should enforce monotonicity of the data rather than interpolate them precisely (see chapter 3).

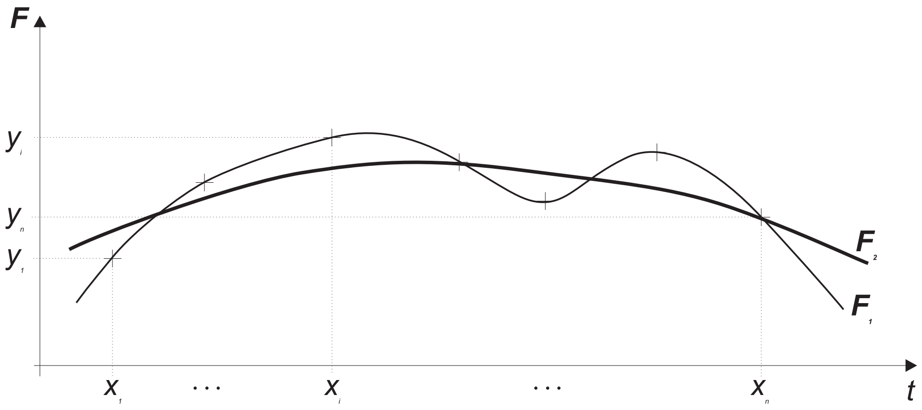

Figure 2.1: *Interpolation. A set of input points (x_i, y_i) and two of many possible functions: F_1 interpolates all the input points exactly, F_2 approximates them only.*

tions. Some of the properties to be considered when designing an interpolation method are mentioned in the following paragraphs.

2.1.1 Continuity

We will generally require the function F to be *continuous* (if the data are composed of multiple non-continuous parts, they can be divided and interpolated separately, producing a piecewise continuous result). We would also usually like it *smooth*, i.e. with continuous (at least) the first derivative.

2.1.2 Local behaviour

For practical reasons, we would like the function to be only *locally sensitive* to the variations of input data. If two input sets differ only at one point (which may be e.g. shifted due to noise in the experiment), we expect their two interpolating functions to be very similar except in a small region around that point. The global sensitiveness to noise is one of the main objections to polynomial interpolation, another are the oscillations of higher order polynomials (see figure 2.2 for illustration).

2.1.3 Monotonicity

Another property we may be eager to obtain is the preservation of *monotonicity*. Unlike the polynomial producing wild oscillations in figure 2.2, we may prefer to interpolate the first six input points, which increase monotonically, by a function F increasing monotonically in the same interval, i.e. the function F should for

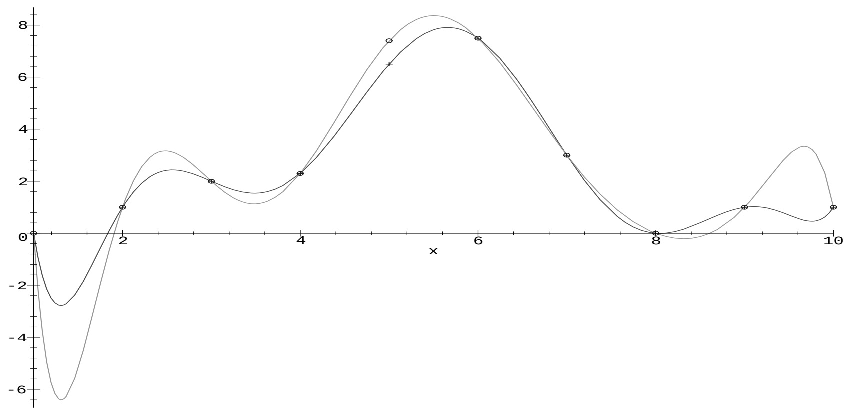

Figure 2.2: Ten input points interpolated by a polynomial. A small local change in input data (at the fifth point) produces a large global change of the output.

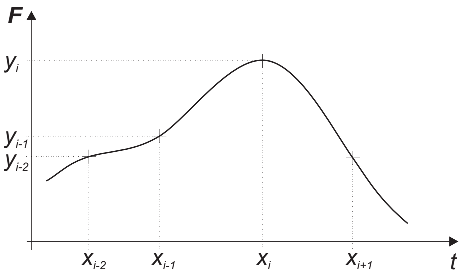

Figure 2.3: A comonotone interpolation as defined by equations (2.3) and (2.4): the result increases and descends simultaneously with the input values.

all $k_1, k_2 \in 1, \ldots, n, k_1 < k_2$ conform to the following conditions:

$$
\begin{aligned}
&(\forall i, j,\ k_1 \le i < j \le k_2 :\ y_i \le y_j) \\
&\qquad \Rightarrow\ \forall t, u \in (x_{k_1}, x_{k_2}),\ t < u :\ F(t, \vec{x}, \vec{y}) \le F(u, \vec{x}, \vec{y}) \\
&(\forall i, j,\ k_1 \le i < j \le k_2 :\ y_i \ge y_j) \\
&\qquad \Rightarrow\ \forall t, u \in (x_{k_1}, x_{k_2}),\ t < u :\ F(t, \vec{x}, \vec{y}) \ge F(u, \vec{x}, \vec{y})
\end{aligned}
\tag{2.3}
$$

The expressions (2.3), together with the interpolation property $F(x_i) = y_i$, lead directly to these consequences:

- an interval of constant input is interpolated by a function equal to the same constant on that interval
- if there is a local maximum at the i-th point of the input, i.e. $y_{i-1} < y_i$ and $y_i > y_{i+1}$, then the interpolating function will change from non-decreasing to non-increasing exactly[2] at x_i (similarly for a local minimum). Figure 2.3 may help your imagination.

[2]It may be acceptable to weaken this condition and have the local maximum somewhere in the interval (x_{i-1}, x_{i+1}); such a behaviour would require a more complex formulation of monotonicity.

We find it quite natural to replace the inequalities in equation (2.3) by their strict counterparts: $y_i < y_j \Rightarrow F(t) < F(u)$ (analogically for decrease), or even ask for the first derivative of the interpolating function F to exist and to be strictly positive in the intervals of increasing input data and negative where the input decreases (such an interpolation F is usually called *comonotone* in the literature), i.e. let F for all $k_1, k_2 \in 1, \ldots, n, k_1 < k_2$ fulfill

$$\begin{array}{lll} (\forall i,j; k_1 \le i < j \le k_2: \; y_i < y_j) & \Rightarrow & \forall t \in (x_{k_1}, x_{k_2}): \; F'(t) > 0 \\ (\forall i,j; k_1 \le i < j \le k_2: \; y_i > y_j) & \Rightarrow & \forall t \in (x_{k_1}, x_{k_2}): \; F'(t) < 0 \end{array} \tag{2.4}$$

As we have already mentioned, we are particularly interested in the desired (co-)monotone behaviour, being involved in the processing of the data from rainbow range finder where monotonicity is a crucial property. Although it may not seem obvious at first sight, monotone interpolation is not a trivial task, and articles on the topic (often trying to introduce monotonicity into polynomial or spline interpolation) have constituted an important group in the approximation theory literature; see section 3.2.1 for a brief overview.

2.1.4 Linearity

If the scale of the input values changes, the function should change its scale accordingly and otherwise remain unchanged. In the language of mathematics, F is *homogeneous* (in its third argument) if the following holds:

$$F(t, \vec{x}, c \cdot \vec{y}) = c \cdot F(t, \vec{x}, \vec{y}) \quad \text{for all } c \in R \tag{2.5}$$

Furthermore, F is called *additive* (in its third argument) if

$$F(t, \vec{x}, \vec{y_1} + \vec{y_2}) = F(t, \vec{x}, \vec{y_1}) + F(t, \vec{x}, \vec{y_2}) \quad \text{for all } \vec{y_1}, \vec{y_2} \in R^n \tag{2.6}$$

and it is called *linear* (in its third argument) if it is both homogeneous and additive.

If F is linear, then all functions $F(t, \vec{x}, \vec{y})$ of variable t, where $\vec{x}$ is a fixed vector from R^n and $\vec{y}$ runs through all elements of R^n, form a *linear space* with respect to the input vectors $\vec{y}$. Let's introduce the *standard base* B of the linear space of input vectors $\vec{y}$ as

$$B = ((1,0,0,\ldots),(0,1,0,\ldots),\ldots,(0,\ldots,0,1)) = (\vec{b}_i)_{i=1}^n \tag{2.7}$$

Then any function F can be expressed as a linear combination

$$F(t, \vec{x}, \vec{y}) = \sum_i y_i \cdot F(t, \vec{x}, \vec{b}_i) \tag{2.8}$$

where y_i is the i-th element of the vector $\vec{y}$ and $\vec{b}_i$ is a vector of the same dimension as $\vec{y}$ with all zeros except a unit at position i. [3]

[3] Note that the base interpolating functions $F(t, \vec{x}, \vec{b}_i)$ have to follow

$$F(t, \vec{x}, \vec{b}_i) = \begin{cases} 1 & t = x_i \\ 0 & t = x_j, \; j \neq i \end{cases}$$

if the function $F(t, \vec{x}, \vec{y})$ computed by the equation 2.8 is to interpolate the input precisely, $F(x_k, \vec{x}, \vec{y}) = y_k$, all k.

Here we have come to a very important topic of linear algebra and an extremely useful one for our task: provided that functions F form a linear space with respect to the input vectors $\vec{y}$, it is sufficient to find the interpolating functions $F(t, \vec{x}, \vec{b}_i)$ for the vectors $\vec{b}_i$ from the base B. Responses to all input vectors $\vec{y}$ can then be expressed using the formula (2.8) as a linear combination of the interpolations of vectors from the standard base.

This is a pleasant property: instead of solving the (possibly complex) task of interpolation for each input $\vec{y}$, we may prefer to precompute the solution for the vectors $\vec{b}_i$ and then only combine them into the final solution. However, storing a large number (equal to n, the dimension of the input space) of the interpolating functions is not often tractable. We dare to make a further simplification based on two premises:

1. The input points are spread with equal distances along the x-axis, i.e. $\Delta x = x_i - x_{i-1}$ is a constant.
2. The differences between the interpolating responses to individual vectors from the base $B_{\vec{y}}$ are not important, i.e. $F(t, \vec{x}, \vec{b}_i)$ can be obtained from $F(t, \vec{x}, \vec{b}_j)$ by a shift $(x_j - x_i)$ in the x-coordinate:
$$F(t, \vec{x}, \vec{b}_i) = F(t - (x_j - x_i), \vec{x}, \vec{b}_j) \tag{2.9}$$

By the first assumption we want to ensure that a base interpolating function which passes through points

$$(x_k, y_k), \quad y_k = \begin{cases} 1 & \text{for } k = i, \\ 0 & \text{for } k = 1, \ldots, n, \ k \neq i \end{cases} \tag{2.10}$$

may be shifted by an amount $(x_j - x_i) = d \cdot \Delta x$, $d \in Z$, in the direction of the x-axis, and the new function will pass through all the points

$$(x_k, y_k), \quad y_k = \begin{cases} 1 & \text{for } k = j, \\ 0 & \text{for } k = 1, \ldots, n, \ k \neq j. \end{cases} \tag{2.11}$$

The generality we lose by this requirement is compensated for by gained simplicity (see below). The uniform sampling grid of the input data needs not always represent a severe limitation, especially if the input data were measured at some uniform sampling frequency of form a uniform grid (as it is often the case in the experiments in some fields; image data in computer vision represent such an example).

With the second premise we may replace an interpolation $F(t, \vec{x}, \vec{b}_i)$ by a shifted function $F(t, \vec{x}, \vec{b}_j)$ (alternatively, we may accept a difference within a reasonable error bound). This is not always possible; cubic splines can serve as a counterexample of a linear space where the interpolations of individual base vectors may differ significantly. See section 2.2.3 for details and discussion.

The two conditions being granted, it suffices to find the interpolation of one base vector $\vec{b} = (0, \ldots, 0, 1, 0, \ldots, 0)$. Let us name this 'base' interpolating function $F_b(t)$, where $t = 0$ corresponds exactly to the point x_i for which there is the unit in the base vector. The function F is now computed easily as follows:

$$F(t, \vec{x}, \vec{y}) = \sum_i y_i \cdot F_b(t - x_i) \tag{2.12}$$

which means that the interpolation is constructed as a linear combination of a single function and its shifts in the x-axis direction.

2.1.5 Outlook

We have mentioned several desirable properties of an 'ideal' interpolation. However, not all of them can be fulfilled at the same time; so, for example, the theorem below shows that the strict monotonicity with positive (everywhere defined) derivative cannot be ensured by any linear method. When designing an interpolation method, we will have to look for a wise compromise between the conflicting requirements.

Theorem 2.1. *Let F_b be a base function such that the corresponding interpolation obtained using (2.12) is comonotone (according to (2.3)) and results always in a smooth function F. Than the derivative of the interpolating function F vanishes at all knots.*

Proof. Let us have $\Delta x = x_j - x_{j-1}$ constant, and let us take for the input data $\vec{y} \in R^n$ the i-th vector of the standard base, i.e.,

$$y_j = \begin{cases} 1 & \text{if } j = i, \\ 0 & \text{otherwise,} \end{cases}$$

where $2 \le i \le n-1$. The interpolating function is $F(t, \vec{x}, \vec{y}) = F_b(t - x_i)$. Due to comonotonicity condition (2.3), we have the following restrictions on F_b:

$$F_b(t) = 0 \text{ for } t \in (-\infty, -\Delta x] \cup [\Delta x, \infty),$$

$$F_b \text{ is increasing on } [-\Delta x, 0],$$

$$F_b \text{ is decreasing on } [0, \Delta x].$$

In particular, F_b has the global maximum at 0. Thus the derivative of F_b vanishes at all integer multiples of Δx. This property is preserved by linear combinations. □

In the following paragraphs we suggest several interpolation methods and study their properties. We start by simple linear algorithms, study cubic splines, continue by a non-linear method using basis functions, present a flexible interpolation by Bézier curves. Then we pursue the transition from one to two (and more) dimensions, and, at the end of the chapter, summarize and discuss the properties of the approaches presented.

2.2 Interpolation through a base function

The charm (or drawback) of the interpolation as a combination of shifted and multiplied base functions as given by equation (2.12) lies in the fact that the result inherits properties of the base function. If F_b is continuous, resp. smooth, the composition will be continuous, resp. smooth, too. Because of linearity, the behaviour of the interpolation depends completely on the choice of the base function F_b. We discuss several reasonable base functions and the impact they have on the resulting function F in this section.

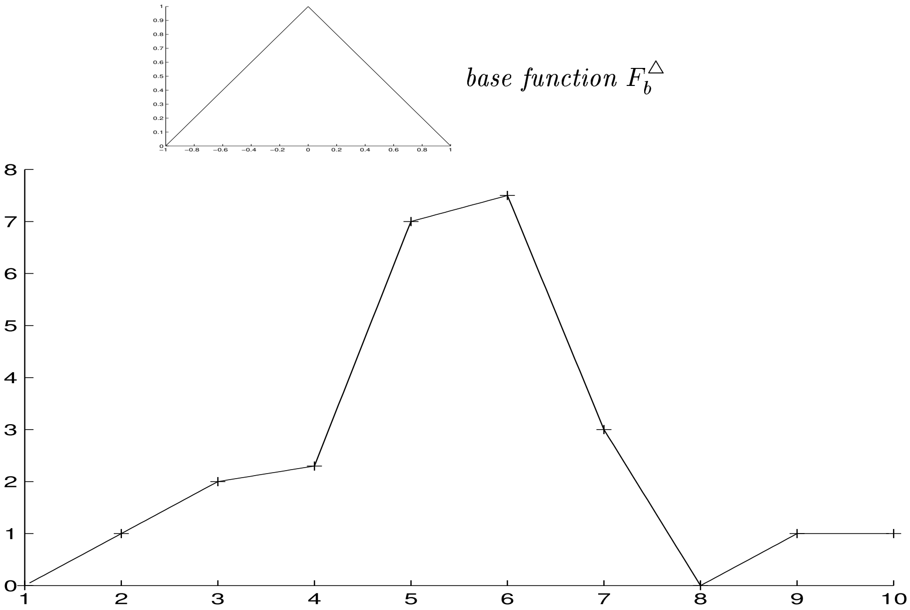

Figure 2.4: *Triangular base function and its corresponding piecewise linear interpolation.*

2.2.1 Piecewise linear interpolation

Perhaps the simplest to be thought of, the triangular function comes first to mind. Employing the base function

$$F_b^{\triangle}(t) = \begin{cases} t+1 & \text{if } -1 \leq t < 0 \\ 1-t & \text{if } 0 \leq t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

leads to the classical piecewise linear (or broken line) interpolation. Both the base and the result are shown in figure 2.4. As the base function runs through all input points, the composed function behaves in the same way. The result is continuous but not smooth, already the first derivative is undefined at the points $t = x_i$. Thank to its simplicity, this method is used in many cases where smoothness is not a crucial point.

2.2.2 Piecewise quadratic interpolation

To form an interpolation with the first derivative defined everywhere, we may employ a smooth 'hill' base function. The base as well as the resulting interpolation are shown in figure 2.5. The base function may be defined by the following quadratic formula (but other forms of a hill function may also lead to similar results):

$$F_b^{q}(t) = \begin{cases} 2(t+1)^2 & \text{if } -1 \leq t < -1/2 \\ 1-2t^2 & \text{if } -1/2 \leq t < 1/2 \\ 2(t-1)^2 & \text{if } 1/2 \leq t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

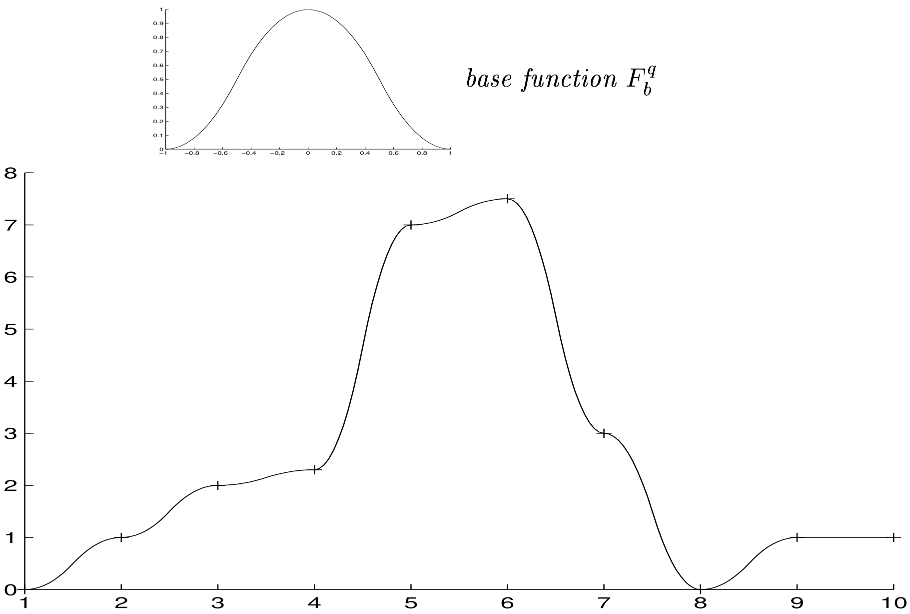

Figure 2.5: *Base function F_b^q and the interpolation.*

The base being smooth, the interpolating function has a continuous first derivative (i.e is smooth), too. However, this function inherits also an unpleasant property from its basis F_b^q: the first derivative of the interpolation is zero at each input point $t = x_i$, including the periods where the input data increase (or decrease) monotonically. This is easily visible in figure 2.9.

2.2.3 Cubic spline interpolation and B-splines

Spline interpolation offers a partial remedy for the main drawbacks of interpolating polynomials: fitting polynomials of limited order to subintervals of the input space, splines restrict oscillations and ensure a semi-local response to a local change of input data. Although splines can be based on polynomials of any order, cubic splines are the most popular. In principle, we fit one cubic polynomial $(a_{i1}x^3 + a_{i2}x^2 + a_{i3}x + a_{i4})$ to each pair of neighbouring input points (which need not be equidistant), while the first and second derivatives are required to be continuous everywhere including the joining points of two polynomials. This requirement of two continuous derivatives, together with the interpolation property $F(x_i) = y_i$ and the chosen boundary conditions at the ends of the input interval (where there is no joining polynomial and therefore two constraints are missing) lead to a tridiagonal set of linear equations for the parameters a_{ij}, which can be solved by Gaussian elimination without pivoting [6].

Spline interpolation belongs to widely accepted techniques and can be found in any textbook on numerical analysis, see e.g. [17] for a more detailed presentation. What we want to stress here (and what is well known to mathematicians familiar with linear algebra, compare [27], as well as to spline experts [6]) is that

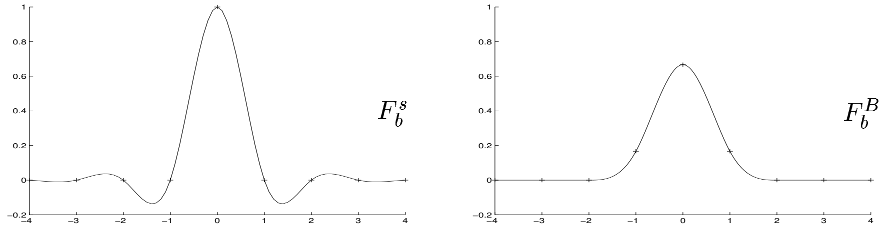

Figure 2.6: *Base functions for cubic spline interpolation: function F_b^s (left), B-spline F_b^B (right).*

splines form a linear space and can thus be fully expressed by base functions and equation (2.8). This way, having constructed the base functions once, there is no need for computation of any of the parameters a_{ij} repeatedly. However, it would be hardly tractable to store a large number of base functions. We dare to simplify further by following the reasoning of section 2.1.4, and employ one single base function plus the equation (2.12) which we reproduce for the spline basis here:

$$F(t) = \sum_i y_i \cdot F_b^s(t - x_i) \tag{2.15}$$

i.e. we interpolate by a linear combination of a base spline function F_b^s, shifted in the x-axis direction and multiplied by the input values y_i. The base F_b^s, obtained as a cubic spline interpolation to the infinite input $(\ldots\ 0, 0, 1, 0, 0,\ \ldots)$, is seen in figure 2.6 (a possible construction of F_b^s will be suggested in Theorem 2.2 on page 14).

However, a serious drawback of the spline interpolation through the base function F_b^s has to be mentioned: this basis has a support (i.e. the interval where its value is not identically zero) of infinite length, so all the input points contribute to the value of each output value (except at x_i), which is computationally hardly tractable. This unpleasant property may be limited by clipping the function F_b^s at some distance $k \cdot \Delta x$ from the center x_0 and discarding further values (here $F_b^s(x_0) = 1$ and $\Delta x = x_{i+1} - x_i$ is the sampling distance of the input). Not much precision needs be lost by this clipping in a discrete setting as the base function F_b^s converges to zero and its amplitude decreases fast; it is less than 0.01 for $|x - x_0| \geq 3\Delta x$ and less than 0.0007 for $|x - x_0| \geq 5\Delta x$ but, strictly speaking, the resulting function is no more twice continuously differentiable. (Continuous derivatives can be obtained if we let the base function 'fade out' smoothly, but this results in a more complex formula.)

We may employ an alternative approach as there exist functions of finite support which can serve as a basis for splines. In spline literature (de Boor [6]) it is shown that *B-splines* form such a basis for any piecewise polynomial functions (of which the cubic splines in this section represent a particular example).

B-splines can be defined by a recursive formula

$$\begin{aligned} B_{i,1}(t) &= \begin{cases} 1 & \tau_i \leq t < \tau_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ B_{i,k}(t) &= \frac{t - \tau_i}{\tau_{i+k-1} - \tau_i} B_{i,k-1} + \frac{\tau_{i+k} - t}{\tau_{i+k} - \tau_{i+1}} B_{i+1,k-1} \end{aligned} \tag{2.16}$$

where τ is the sequence of $n+k$ knots, i.e. the points where the piecewise polynomial (pp) changes its form, with n the number of input points, k the order (equal to the degree plus one) of the B-spline; the knot sequence is closely connected to the properties of the resulting pp function.

A detailed discussion of B-spline properties has filled numerous books (apart from those cited earlier, B-splines of various forms are employed frequently in computer-aided geometric design, see e.g. [33, 12]), we restrict further presentation here to B-splines for cubic spline interpolation.

In the cubic case ($k=4$) we have n input points x_i and need $n+4$ knots τ_i. The interior n knots can be set to coincide with the input x-coordinates, $\tau_{i+2} = x_i$, $i = 1, \ldots, n$. The remaining knots have to follow $\tau_i < x_i < \tau_{i+k}$ (from [6], Theorem XIII.1); their choice influences behaviour of the spline near the boundary.

More specifically, with uniform sampling distance of the input in the x-direction, we may set also the knot sequence equidistant, $\tau_{i+1} - \tau_i = \Delta x$, $i = 1, \ldots, n+k-1$, with $\tau_3 = x_1$ (for simplicity, we will also presume $\Delta x = 1$). Then all the necessary cubic B-splines become identical (up to a shift) and we can write

$$B_{1,4}(t) = F_b^B(t) = \begin{cases} 0 & t \le -2 \\ \frac{1}{6}\,t^3 + t^2 + 2\,t + \frac{4}{3} & -2 < t \le -1 \\ -\frac{1}{2}\,t^3 - t^2 + \frac{2}{3} & -1 < t \le 0 \\ \frac{1}{2}\,t^3 - t^2 + \frac{2}{3} & 0 < t \le 1 \\ -\frac{1}{6}\,t^3 + t^2 - 2\,t + \frac{4}{3} & 1 < t \le 2 \\ 0 & 2 < t \end{cases} \tag{2.17}$$

and construct the spline interpolation through

$$F(t) = \sum_{i=1}^{n} \alpha_i \cdot F_b^B(t - x_i) \tag{2.18}$$

The cubic B-spline F_b^B is shown in figure 2.6. For a given interval $x_j \le t \le x_{j+1}$ the sum (2.18) simplifies to

$$F(t) = \sum_{i=j-1}^{j+2} \alpha_i \cdot F_b^B(t - x_i) \tag{2.19}$$

as all except four cubic B-splines vanish between x_j and x_{j+1}.

The n coefficients α_i can be computed either from n linear equations $F(x_i) = y_i$ similarly to the direct computation of cubic spline at the beginning of the chapter, by recursive filtering [44, 45], or obtained by the following scheme.

Let us find the B-spline coefficients for an interpolation of the input vector $(\ldots, 0, 0, 1, 0, 0, \ldots)$ where unit is in the zero x-coordinate, and name them β_j, $j = -\infty \ldots \infty$. Then the original basis function $F_b^s(t)$ may be expressed in terms of B-splines as

$$F_b^s(t) = \sum_{j=-\infty}^{\infty} \beta_j \cdot F_b^B(t - x_j) \tag{2.20}$$

and the interpolation from equation (2.15) becomes

$$F(t) = \sum_i [y_i \cdot \sum_{j=-\infty}^{\infty} \beta_{j-i} \cdot F_b^B(t - x_j - x_i)] = \sum_i \alpha_i \cdot F_b^B(t - x_i) \tag{2.21}$$

Considering how the shifted base functions overlap, we can write

$$\alpha_i = \sum_{j=-\infty}^{\infty} y_j \cdot \beta_{j-i} \tag{2.22}$$

A similar observation as with clipping the spline basis F_b^s can be mentioned here: the infinite summation in equation (2.22) can be replaced by a finite sum over the input space without any loss, as only there the values y_j are not zero; if we discard more, we lose some precision.

In an idealized infinite case (and as nearly optimal solution for a finite input) the coefficients β_i needed in equations (2.20) to (2.22) may be constructed as suggested by the following theorem:

Theorem 2.2. *Let $m \in Z$, let the knot coordinates $x_i = i$, $i = -m, \ldots, m$, and write the function F_b^s using the cubic B-spline basis F_b^B as*

$$F_b^s(t) = \sum_{j=-\infty}^{\infty} \beta_j \cdot F_b^B(t-j)$$

Then F_b^s minimizes $\int_{-m}^{m}(f''(x))^2 dx$ for $m \to \infty$ over all cubic spline functions f on the same knot sequence which interpolate the same input (…, 0, 0, 1, 0, 0, …) if the coefficients are chosen as follows:

$$\beta_0 = \sqrt{3}, \quad \beta_i = \sqrt{3}\left(-2+\sqrt{3}\right)^i, \quad \beta_{-i} = \beta_i, \quad i = 1, 2, \ldots, \infty$$

Proof. Let's consider the knot sequence $(-m, -m+1, \ldots, -1, 0, 1, \ldots, m-1, m)$ and interpolation of a symmetric input formed by a unit with m zeros added on either side, $\vec{y} = (0, \ldots, 0, 1, 0, \ldots, 0)$. The interpolation f will be also symmetric, $f(-t) = f(t)$, formed by

$$f(t) = \sum_{i=-m-1}^{m+1} \beta_i \cdot F_b^B(t-i) \tag{2.23}$$

where also the coefficients will be symmetric: $\beta_{-i} = \beta_i$. Only three shifted B-splines F_b^B overlay at any knot, and the interpolating conditions $f(i) = y_i$ (for $i \geq 0$ only, as the symmetric negative part does not bring more information) translate into a system of $m+1$ equations for $m+2$ unknown variables β_i, $i = 0, \ldots, m+1$:

$$\begin{gathered} f(0) = \beta_{-1}F_b^B(-1) + \beta_0 F_b^B(0) + \beta_1 F_b^B(1) = \beta_0 F_b^B(0) + 2\beta_1 F_b^B(1) = 1 \\ f(j) = \beta_{j-1}F_b^B(-1) + \beta_j F_b^B(0) + \beta_{j+1}F_b^B(1) = 0, \quad \text{for} \quad j = 1 \ldots m \end{gathered} \tag{2.24}$$

As we have $F_b^B(0) = \frac{2}{3}$, $F_b^B(-1) = F_b^B(1) = \frac{1}{6}$, the system (2.24) becomes

$$\begin{gathered} \tfrac{2}{3}\beta_0 + \tfrac{1}{3}\beta_1 = 1 \\ \tfrac{1}{6}\beta_{j-1} + \tfrac{2}{3}\beta_j + \tfrac{1}{6}\beta_{j+1} = 0 \qquad \text{for} \quad j = 1 \ldots m \end{gathered} \tag{2.25}$$

This can be rewritten into a system of recurrence equations

$$
\begin{aligned}
\beta_1 &= 3 - 2\beta_0 \\
\beta_{j+1} &= -4\beta_j - \beta_{j-1} \quad j = 1 \dots\ m
\end{aligned}
\tag{2.26}
$$

which leads to the expression (for $j = 1, \dots, m+1$)

$$
\beta_j = \frac{1}{2}\left(\beta_0 - \sqrt{3}\right)\left(\frac{1}{-2+\sqrt{3}}\right)^j + \frac{1}{2}\left(\beta_0 + \sqrt{3}\right)\left(-2+\sqrt{3}\right)^j \tag{2.27}
$$

The first summand is divergent, the second is convergent for increasing j. Let $f_m(j)$ be the extremal value of $f(t)$ in the interval $[j, j+1]$. Divergent summand in (2.27) leads to divergent β_j and that leads by equation (2.23) to function $f(t)$, which has the characteristics $f_m(j)$ divergent for increasing j, and oscillating, $sgn(f_m(j)) \neq sgn(f_m(j+1))$. Clearly, such a divergent function f cannot minimize the integral $\int_{-m}^{m}(f''(t))^2 dt$ for m sufficiently large if there exists another, convergent function f, and that exists: the divergent summand in (2.27) vanishes if we set

$$
\beta_0 = \sqrt{3}
$$

The remaining coefficients become (directly from (2.27))

$$
\beta_j = \sqrt{3}\left(-2+\sqrt{3}\right)^j
$$

□

Constructing a base function which minimizes $\int_{-\infty}^{\infty}(f''(x))^2 dx$ is of special importance because it is exactly this minimizing property which distinguishes cubic splines (with boundary conditions properly chosen) from other twice differentiable piecewise cubic functions. The curve composed from the base function F_b^s by (2.15) will be itself a (curvature-minimizing) cubic spline[4].

Our 'base-formed' spline is in figure 2.7 shown together with the spline computed directly by a Matlab function. If you compare the two curves, you can see that they differ visibly only at the ends of the input interval. This calls for an explanation.

When we suggested to move from many base functions to a single one in section 2.1.4, we based the reasoning on two premises: equal spacing of the input points' x coordinates, and similarity of the base functions (except for a shift in the x direction). With splines, the latter is not fulfilled completely: the cubic spline interpolation of input vectors $\vec{y}$ with a single unit near either of its ends (for example $\vec{y}_1 = (1, 0, \dots, 0)$ or $\vec{y}_2 = (0, 1, 0, \dots, 0)$) could lead to base functions which differ significantly from the employed base function F_b^s which originates from the input $\vec{y}_i = (\dots, 0, 1, 0, \dots)$ (the difference depends on the choice of the boundary conditions for a finite input).

This way, our base spline interpolation behaves as if there were zeros added to the original input, $\vec{y}_{new} = (0, \dots 0, \vec{y}, 0, \dots 0)$. However, as we have to choose the boundary conditions of the spline arbitrarily anyway, we claim the spline interpolation through a base function to be an equally good solution as the original one, computed through a set of equations.

[4]Mind that this solution is only approximate for a finite input.

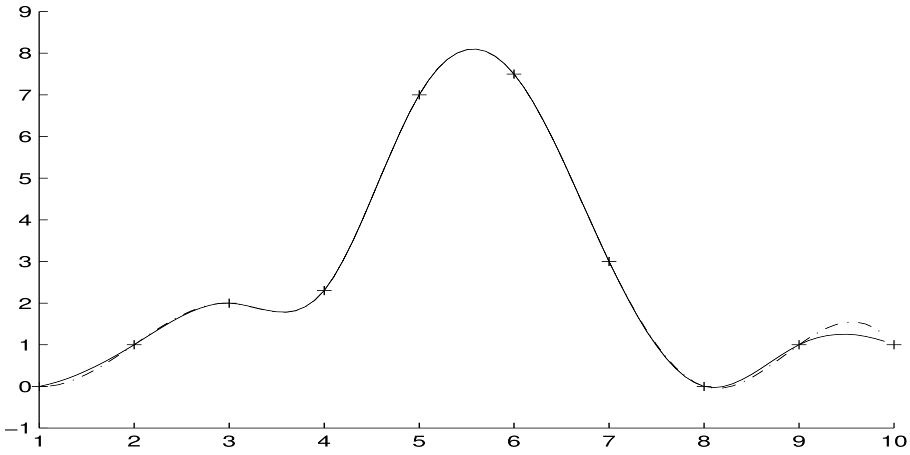

Figure 2.7: *The interpolation through the base function F_b^s (solid line) and the spline interpolation with the not-a-knot end conditions (dashed line, computed in Matlab).*

We are never completely satisfied, though: because the spline interpolation is forced to the continuity of the first two derivatives, its inertia (or lack of flexibility) may introduce unpleasant oscillations. Figure 2.7 illustrates such a case: although the first six points of input data increase monotonically, the interpolation contains local extrema (between points 3 and 4, and between 5 and 6). Andersson in [1] suggests an algorithm to introduce monotonicity into spline interpolation, but that is no more a linear method (see section 3.2.1 for more information).

2.2.4 Approximate interpolation

Until now, the base functions were constructed in such a way that they passed through all the input points, and consequently the interpolation passed through all the input points as well. In the real-world situations the input data are often affected by noise; then the interpolating function F is not required to run through the points exactly but only approximately[5] and we perform a sort of smoothing while processing the data. Omitting the "pass-through-points" constraint leaves us completely free while deciding on the interpolating function behaviour, and there are many possibilities how to design the base function. A thorough discussion of approximation and de-noising techniques is deferred until later chapters, we present only one possibility and the result in figure 2.8. The base function of section 2.2.2 was stretched to a double width (and the amplitude diminished accordingly, so that the contributions of weights given

[5]There exist another term, approximation, which we find somehow similar to the notion of interpolation. The task of approximation consists of finding another (usually simpler) representation for a given function, while interpolation creates a function representation for a set of data, it fits a curve to the points. However, the two terms are often employed interchangeably when constructing a function to run approximately through a set of tabulated points.

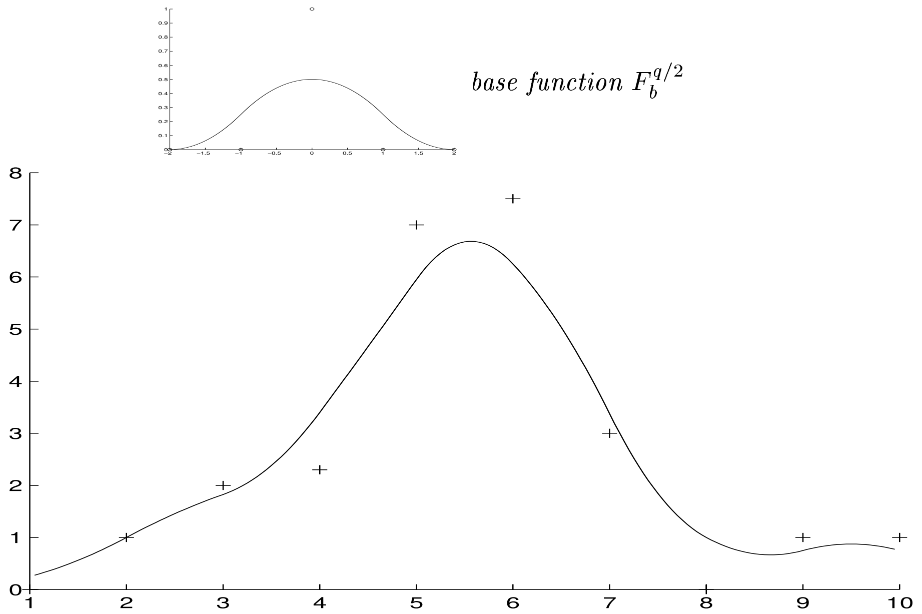

Figure 2.8: *Approximate base function and the interpolation.*

by the base function at input points sum to one) to produce the new base. To obtain a very similar result we may employ the filter $Gaussian_{\frac{1}{\sqrt{2}}}(x) = 2^{-2x^2}$ as the base function. The Gaussian is discussed together with several other filters for image resampling in [43], and its smoothing properties are studies by *scale space theory* (see chapter 3.2.2 or Lindeberg's book [24])

2.2.5 Conclusion of linear interpolation methods

Interpolation through the base function represents a very simple, yet powerful approach. If we precompute the base function values in the necessary resolution and store it in an array, we can then perform the interpolation by summing the multiplied and shifted base arrays, which is computationally efficient and very easy to implement. However, this scheme is limited to equally spaced input points.

Although the linearity of the interpolating function space is a nice mathematical property, it introduces strict limitations on the resulting function shape. Problems of smoothness and derivatives, violations of monotonicity, or imprecise interpolation, that is the inevitable choice. We would be sometimes willing to sacrifice the simplicity and regularity of linear methods and would prefer a different solution. Some of them are presented in the following sections.

2.3 Two base functions

The first non-linear approach to be presented comes as a natural extension of the interpolation through quadratic base described in section 2.2.2. As the

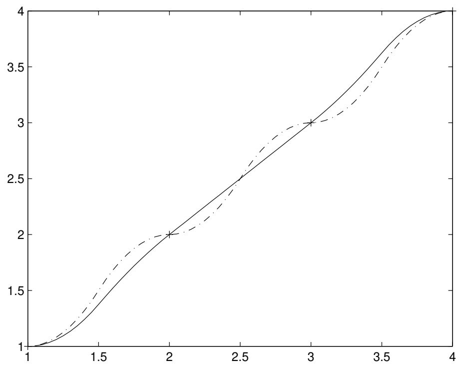

Figure 2.9: *Interpolation through base function F_b^q (dotted line) compared with interpolation using two base functions (solid line).*

major objection to that method we observed that the first derivative is zero at each input point: $F'(x_i) = 0$, i.e. the tangent at each knot x_i is parallel to the x-axis. In periods of linear growth, the interpolation resembles a "worn down staircase". Figure 2.9 shows the interpolation through the base function contrasted with the interpolation method which we are going to describe now.

We are quite satisfied with the performance of the base function F_b^q except for the intervals where the input data increase or decrease monotonically. The remedy is simple: we employ a special rule to improve the function shape in these intervals. The rule is represented by a second base function F_b^g, added to the original input in the areas of monotonicity, which straightens the result a little so that no points of zero derivative remain in the growing regions. However, we must be careful not to violate the nice property of the original interpolation which was the non-existence of oscillations, i.e. false local extrema where there were none of them in the input data.

It seems natural that an interpolation should simplify to a straight line in the intervals where the input data grow (or decrease) linearly. Let's consider several input points, $(x_{i-k_1}, y_{i-k_1}) \ldots (x_{i+k_2}, y_{i+k_2})$, where $k_1, k_2 > 1$. Suppose that y_j depend linearly on x_j in the interval $[x_{i-k_1}, x_{i+k_2}]$. Then there are constants $c_1 \in R\backslash\{0\}$, $c_2 \in R^+$ such that $x_j - x_{j-1} = c_1$, $y_j - y_{j-1} = c_2$ for all j, $j = i - k_1 + 1, \ldots, i + k_2$, and the interpolation composed of the first base function F_b^q and the 'growth' base function F_b^g should satisfy the equation

$$F(t) = \sum_{j=i-k_1}^{i+k_2} y_j \cdot F_b^q(t - x_j) + w_j \cdot F_b^g(t - x_j) = \frac{c_2}{c_1}(t - x_i) + y(i) \qquad (2.28)$$

for $t \in (x_{i-k_1+1}, x_{i+k_2-1})$. Solving the equation we obtain

$$F_b^g(t) = \begin{cases} -\frac{1}{2}(t+1)^2 & \text{if } -1 \le t < -1/2 \\ \frac{3}{2}t^2 + t & \text{if } -1/2 \le t < 0 \\ -\frac{3}{2}t^2 + t & \text{if } 0 \le t < 1/2 \\ \frac{1}{2}(t-1)^2 & \text{if } 1/2 \le t < 1 \\ 0 & \text{otherwise} \end{cases} \qquad (2.29)$$

The final interpolation is computed from two base functions by the formula

$$F(t, \vec{x}, \vec{y}) = \sum_i [y_i \cdot F_b^q(t - x_i)] + \sum_{j \in J} [w_j \cdot F_b^g(t - x_j)] \qquad (2.30)$$

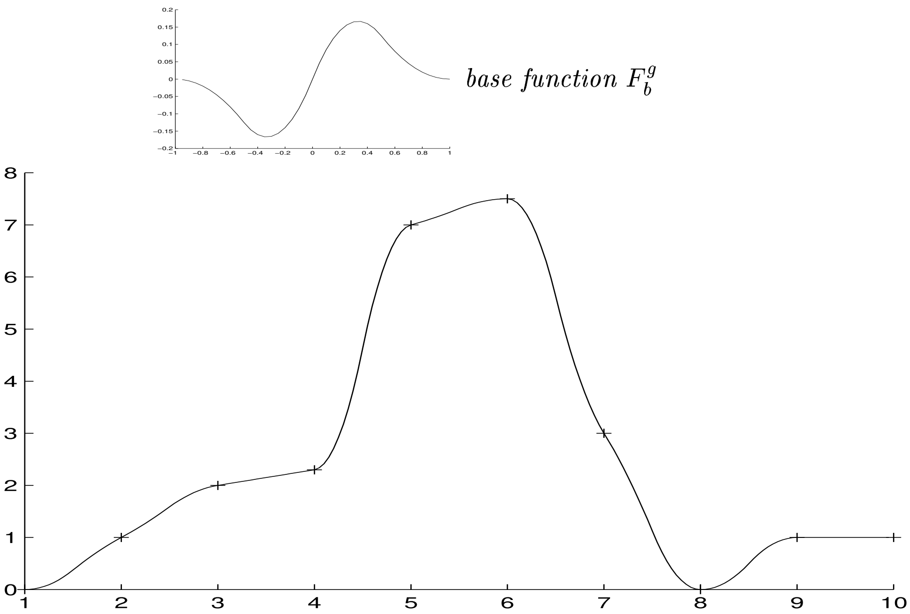

Figure 2.10: *The second base function (top, note that the maximum amplitude is only 1/6) employed for the interpolation by two base functions:* F_b^q *and* F_b^g.

where i is taken for all input points, j from the index set J such that

$$j \in J \iff (y_{j-1} < y_j < y_{j+1}) \text{ or } (y_{j-1} > y_j > y_{j+1})$$

The 'growth' base F_b^g is multiplied by the weight w_j (positive for a growing, negative for a decreasing interval); its magnitude is given by

$$|w_j| = min(\ |y_j - y_{j-1}|,\ |y_{j+1} - y_j|\ )$$

To state that in words, the growing sum is added if the input data increase or decrease through y_j to its neighbours monotonically, being scaled by the smaller of the y-distances from point j to its neighbours. As the growth base function's derivative at position zero is equal to one, the tangent of the interpolation at point j in growing/decreasing region will point towards the closer of the neighbouring points. Figure 2.10 shows the base function F_b^g and the interpolation of our sample data.

We may also remark that the derivative will be greater then zero in the whole interval where the input data increase monotonically, and negative where they decrease monotonically; this interpolation by two base functions is thus the first we have seen so far to conform with the strict comonotonicity requirement (2.4) from section 2.1.3. We have already mentioned that this two-base interpolation does not belong to linear methods. The interpolating function F is not additive (except for the following: we may define the cone of nondecreasing functions and preserve linear combinations with nonnegative coefficients on it) but it still fulfills the condition of homogeneity and a scaled input leads to a scaled, otherwise identical output.

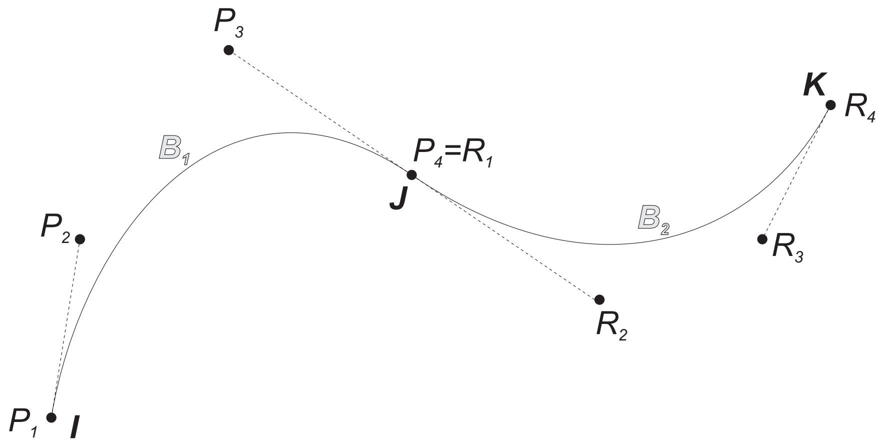

Figure 2.11: *Two cubic Bézier curves joined smoothly. Points P_1, P_2, P_3, P_4 control the first curve's shape, $R_1 \ldots R_4$ are the control points of the second Bézier cubic.*

2.4 Interpolation using cubic Bézier curves

2.4.1 Background on Bézier curves

Bézier curves are parametric curves widely used in computer graphics (presented in most computer graphics textbooks, see e.g. [15]) and thanks to their straightforward behaviour generally employed for drawing smooth curves in graphics editors. Their shape is determined by the position of the control points. The number of points implies the degree of the curve; cubic Bézier curves are the most frequent and the only ones to be mentioned from now on.

The shape of the Bézier cubic is determined by four control points P_1, P_2, P_3, and P_4, the x and y-coordinates of each of them will be denoted P_i^x and P_i^y, respectively. A point $B(u)$ of the curve, given by its two coordinates $B^x(u)$, $B^y(u)$, is defined by the following parametric functions (where $0 \leq u \leq 1$):

$$\begin{aligned} B^x(u) &= (1-u)^3 P_1^x + 3(1-u)^2 u P_2^x + 3(1-u)u^2 P_3^x + u^3 P_4^x \\ B^y(u) &= (1-u)^3 P_1^y + 3(1-u)^2 u P_2^y + 3(1-u)u^2 P_3^y + u^3 P_4^y \end{aligned} \tag{2.31}$$

The four polynomials playing the role of weights of the points P_i are called Bernstein polynomials, and exhibit the following properties: they are positive for $u \in [0, 1]$, their sum is equal to one for every u from the same interval. Therefore, each point on the Bézier cubic is computed by (2.31) as a convex combination of the control points and the whole curve is contained inside the convex hull of the points P_i. For $u = 0$ all coefficients except the first are equal to zero: the curve starts at point P_1. For $u = 1$ there remains only the fourth non-zero member and the curve ends at the last control point P_4. The Bézier cubic does not generally pass through its two intermediate control points. The tangent vector at point P_1 aims towards the second control point P_2; similarly, the tangent at P_4 is collinear with the line joining the last two control points

P_3 and P_4.

Longer, or rather more complex curves can be composed of several Bézier cubics concatenated one after another. Two such cubics are sketched in figure 2.11. Points P_1 to P_4 belong to the first segment (B_1), while points R_1, R_2, R_3 and R_4 control the second one (B_2). To join the two curves B_1 and B_2, the point R_1 must be set identical to P_4; then from equation 2.31 it follows that $B_1(1) = P_4 = R_1 = B_2(0)$.

If we want to join the two sections smoothly (and we do require that), the first derivative at the shared point must be identical, for which there are two necessary and sufficient conditions:

- the three points P_3, $P_4 = R_1$, and R_2 must be collinear, and
- the distances between the pairs must equal: $d(P_3, P_4) = d(R_1, R_2)$.

Such a smooth join is drawn in figure 2.11.

Did we require even the second derivative to be continuous, the points P_2, $P_4 = R_1$ and R_3 would have to be collinear. Cubic spline could result from such a setting (as in [35], see the following section) but the curve would be less flexible.

2.4.2 Bézier curves applied to interpolation

As mentioned above, Bézier curves do not pass directly through all their control points, but only approximate them. If we intend to interpolate a set of points by Bézier curve, we cannot use the input data directly as control Bézier points and have to employ some other technique to provide the control points. So Elber in [11] fits a curve of degree k to the $(k + 1)$ input points, but we find the high degree of the curve rather objectionable.

A more interesting idea is mentioned in [35], where Rasala constructs a cubic Bézier curve to replace a cubic spline passing through a sequence of points, because Bézier curve is then computationally cheaper to evaluate then splines. To accomplish that, he suggests adding two control Bézier points for each input point. The control points' positions are computed through a set of equations; practically (for a reasonable precision) 5 input pairs around a point influence his added control points.

We base our interpolation on the same idea, we fit a Bézier cubic curve to each pair of the input points, the interpolation is then formed by a series of such curves. The section is controlled by four points: P_1 and P_4 coincide with two neighbouring input points, and we have to add two control points P_2 and P_3 in between. The manner how we do that determines entirely the resulting interpolating curve. In this the Bézier-based interpolation is by far the most flexible of the methods we have encountered so far (especially so if we do not require even the points P_1 and P_4 to coincide with the input data).

But being almost completely free in choosing the interpolating function behaviour, we come again to the crucial question, what is the optimal interpolation like? After contemplations and experiments we have designed the Bézier interpolating curve with the following properties: it is smooth (the first derivative is continuous, not necessarily the second), it is homogeneous (but not necessarily additive), it is strictly co-monotone with the input data. In all this, the interpolation resembles the one constructed through two base functions in section 2.3,

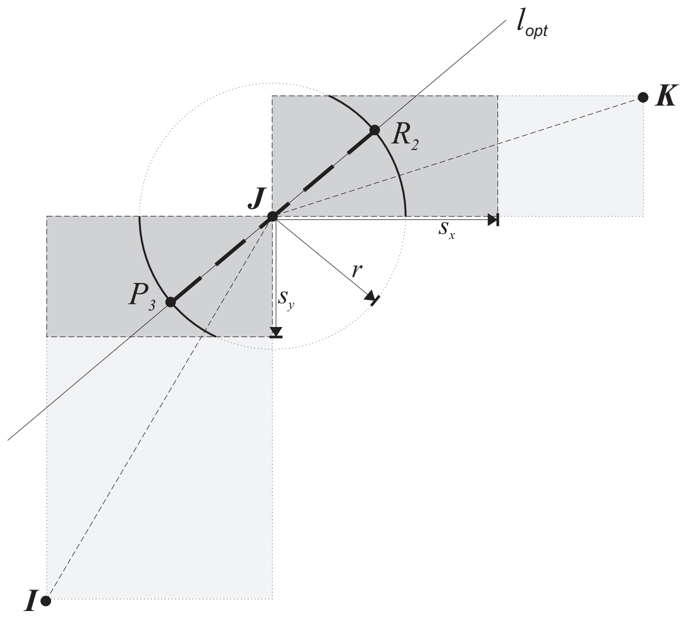

Figure 2.12: *Placement of the control points P_3 and R_2. They are required to lie on the circle with diameter r centered at point J, within the rectangles, on a line passing through J.*

but is not restricted to equally spaced input points as the base-based methods, and is perhaps visually nicer with controllable curvature at the knots.

On the other hand, there is a complication: the function is parametric with the parameter u which does not correspond to the axis direction as in the previous sections. If we want to compute the y-coordinate for a given x, we have first to find u to solve $x = B^x(u)$, which can be accomplished either numerically by Newton's algorithm in a few iteration steps, or analytically, but evaluation of the formula $u = (B^x)^{-1}(x)$ may be computationally more expensive than the numerical solution. Then the value u is to be substituted into $y = B^y(u)$. (All these complications can be avoided if we place the intermediate control point P_2 to one third and P_3 to two thirds of the x-interval between P_1 and P_4. Then the x-coordinate depends on u linearly and the solution of $u = (B^x)^{-1}(x)$ becomes trivial.)

Now to the essential part of the method, placement of the control Bézier points. Let us take three succeeding input points I, J and K. The two intervals between them have to be filled by two Bézier cubic curves joined together at point J. We keep the notation of figure 2.11, points P_1 to P_4 control the curve B_1 between I and J, points R_1 to R_4 the second, B_2, between J and K. The easy part comes first: control point P_1 is set to coincide with the input point I, points P_4 and R_1 with input point J, and R_4 with K. Next we find the positions for the points P_3 and R_2, which are the closest to the point J, central from the three input points. (The remaining control points P_2 and R_3 will be placed using other points from the input by the same procedure as follows.)

We want the intermediate control points of one segment to be contained inside a rectangle with its sides parallel to the direction of the x- and y-axes, and which is defined by the end control points as two of its corners. If this is granted, the curve segment $B(u)$ will run from the first to the fourth control point monotonically in both coordinates x and y. In our particular situation, the point P_3 is bounded by the rectangle between the input points I and J, and point R_2, similarly, must be placed inside a rectangle between J and K (see the light grey rectangles in figure 2.12). These two constraints can be unified: because both control points must lie on a line passing through J and in the same distance from it, let us take s_x as the smaller of the above mentioned rectangles' sizes in the direction of the axis x, and s_y smaller of the size in the direction of the axis y. Now let us draw two rectangles with the point J as one of their corners and contained in the rectangles drawn before. The control points we are looking for must be found somewhere inside these smaller rectangles (the darker two in figure 2.12). Let

$$d = \sqrt{{s_x}^2 + {s_y}^2} \tag{2.32}$$

be the length of these new rectangles' diagonal.

Now we have to decide the distance of the control points P_3 and R_2 from the point J. That is equivalent to choosing a radius r of a circle centered at J and forcing the control points to lie on this circle. We compute r by a simple formula:

$$r = \alpha\ d \tag{2.33}$$

where d is the diagonal size from the preceding paragraph and $\alpha \in (0, 1)$ a parameter influencing the curvature of the Bézier cubic at the point J. As the parameter α increases, the curvature at point J decreases (but may increase somewhere else on the curve section). The parameter α to minimize the integral of curvature along the path depends on the input points' positions. Generally α in the range $(0.25, 0.4)$ gives the best results.

One last clue remains to be supplied to find the position of the control points P_3 and R_2. The points are known to be collinear together with the point J (for a smooth join of two Bézier curves); we must decide on the direction of the line passing through them. We do this by choosing a preferred direction: if possible (i.e. if such a line has a common point with the intersection of the circle and the rectangles from the previous paragraphs), take the direction parallel with the line joining two neighbouring input points I and K. If there is no common point, take the closest direction while obeying the circle-and-rectangle constraint.

However arbitrarily we designed this preferred direction of the tangent at the input point, we find that it yields visually pleasing outputs. A similar idea appeared in Foley – van Damm [15] (p. 505): the smooth interpolating line whose tangent at each input point is parallel to the line between the adjacent points was called 'Catmull – Rom spline' there.

As the result of the procedure described above, if the input points I, J, and K increase (resp. decrease) monotonically, the interpolating curve will increase (resp. decrease) monotonically as well (and the derivative will be positive resp. negative). If, on the other hand, there is a change of monotonicity, i.e. J is above (or below) both I and K, the bounding rectangles will force the control points to a line passing through J and parallel to the x-axis, and the interpolation will have a local extreme at point J. This conforms with the expectations introduced on monotonicity in section 2.1.3.

When run on our sample data, the procedure supplied the Bézier control points seen in figure 2.13. Figure 2.14 then presents the interpolating curve.

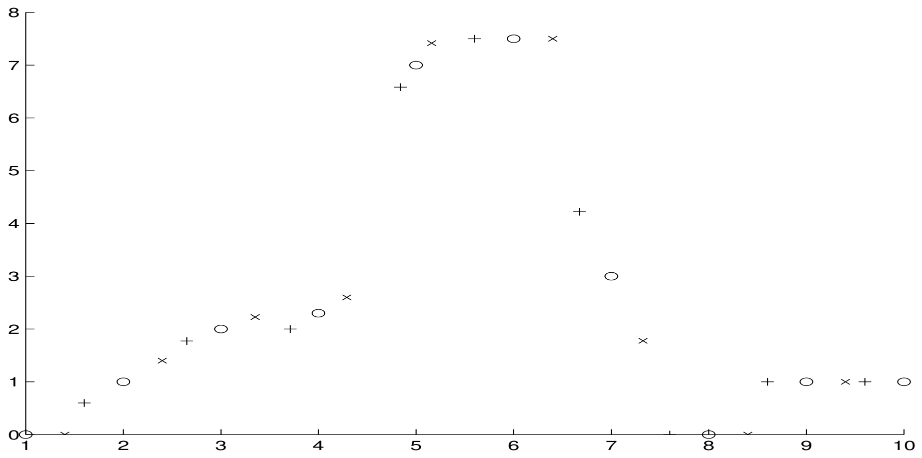

Figure 2.13: *Control points for the Bézier interpolation. Circle represents the input point; between each pair of input points the symbol* × *is used for the control point P_2 and the symbol + belongs to control point P_3. (The parameter α was set to 0.4)*

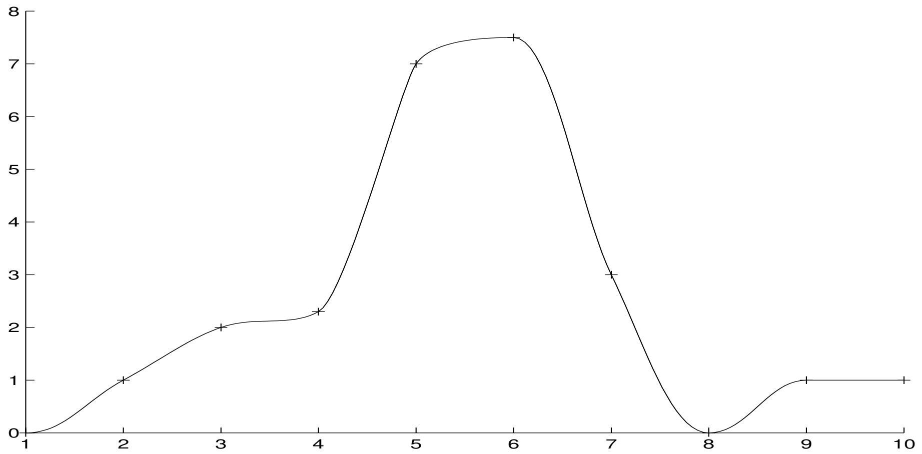

Figure 2.14: *Interpolation by Bézier cubics resulting from the control points' placement in the previous figure.*

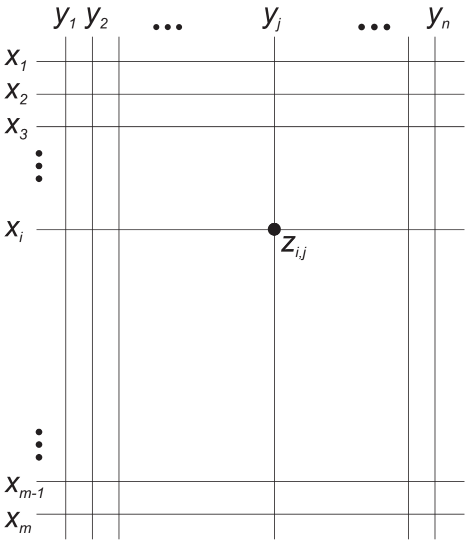

Figure 2.15:
Two-dimensional grid of input data values z_{ij} sampled at positions (x_i, y_j).

2.5 More dimensions

We have presented the interpolation methods in the simplest form when the input data were formed by a (one-dimensional) sequence of points. How to proceed if we want to extend the procedures to a more complex situation of more dimensions? We are going to answer this question, although limiting the difficulty and presenting the solution in detail only for the input data forming a two-dimensional, rectangular grid (as seen in figure 2.15). However, the transition to a higher dimension does not present an unsurmountable problem, the idea remains the same as for the move from one to two dimensions.

Let us introduce the situation in 2D:

$$\begin{aligned} \vec{x} &= (x_1, x_2, \ldots, x_m), \text{ and} \\ \vec{y} &= (y_1, y_2, \ldots, y_n) \end{aligned} \tag{2.34}$$

are the x- and y-coordinates of the lines where the input data are sampled; the pair (x_i, y_j) gives the 2D position of the input value z_{ij}. Taking all combinations of the parameters $i = 1, \ldots, m$ and $j = 1, \ldots, n$ the input data z_{ij} will form a two-dimensional array

$$Z = \begin{bmatrix} z_{11} & \cdots & z_{1n} \\ \vdots & \ddots & \vdots \\ z_{m1} & \cdots & z_{mn} \end{bmatrix} \tag{2.35}$$

The matrix Z may be viewed as a 2D image, with the values z_{ij} representing the gray level (brightness) of the pixel in i-th column and j-th row of a digitized picture. This is a common representation in the fields of computer vision and image processing. The sampling points are then evenly spaced, with both $(x_{i+1} - x_i) = \Delta x$ and $(y_{j+1} - y_j) = \Delta y$ constant for all i and j; usually also $\Delta x = \Delta y$ and the sampling points form a regular, rectangular grid.

Alternatively, we may understand z_{ij} as a third coordinate; the triple (x_i, y_j, z_{ij}) would then represent the position of an input point in 3D. Although the results will be visualized this way, we call the interpolation task a two-dimensional one because of the two independent variables present in the interpolating function.

We intend to construct a function $F(t, u, \vec{x}, \vec{y}, Z)$ to interpolate the input points. The function parameters t and u are chosen to run parallel with the direction of the axes x and y, respectively.

2.5.1 Base functions in 2D

Similarly to the one-dimensional case, interpolating functions in two dimensions may form a linear space with respect to the input data. Then all we have to do to solve the interpolation problem is to find a suitable base function F_b^{2D} and compose the result by the following equation:

$$F(t,u,\vec{x},\vec{y},Z) = \sum_{i=1}^{m}\sum_{j=1}^{n} z_{ij} \cdot F_b^{2D}(t-x_i, u-y_j) \tag{2.36}$$

The interpolating function is found as a composition of the base functions, where each base function is added with its center shifted to the input point (x_i, y_j) and multiplied by the input value at that point.

If we have a base function F_b^{1D} for interpolation in one dimension, the corresponding interpolating base function extended into two dimensions can be obtained by the formula

$$F_b^{2D}(t,u) = F_b^{1D}(t) \cdot F_b^{1D}(u) \tag{2.37}$$

In one dimension, having decided on the desired resolution of the result, we stored the base function in an array of real numbers, representing the shape of the base function sampled at discrete points. We name this array A_b and compute its i-th member by

$$A_b(i) = F_b^{1D}\left(\frac{i-\frac{n}{2}}{n_1}\right) \tag{2.38}$$

where n is the total number of samples in the array, and n_1 states the number of samples per unit interval. So the fraction $\frac{n}{n_1}$ defines the size of the support of the base function, i.e. the size of the interval $[\inf(A), \sup(A)]$ where $A = \{x : f(x) \neq 0\}$. (Note that in 1D all the methods studied here except the spline and approximate base functions had the support $[-1, 1]$).

The sampled base function in more than one dimension can be easily constructed as a matrix product of the sampled base functions in 1D. The formula for two dimensions follows:

$$A_b^{2D} = {A_b}^T \cdot A_b \tag{2.39}$$

If the original base array was of size n, the new base will be stored in a $n * n$ matrix of real numbers. Efficiency stands as the main advantage of this representation: we precompute the values during the initialization, then we have only to add a matrix multiplied by a scalar for each input point during the interpolation.

Having introduced the two-dimensional situation, we proceed with a brief presentation of individual 2D base functions and their properties when applied to interpolation. The methods are tested on a matrix of input points of size 5x6, where a triangular set of points has the zero z-value, the rest forms a plane increasing in directions of both axes x and y; see figure 2.16.

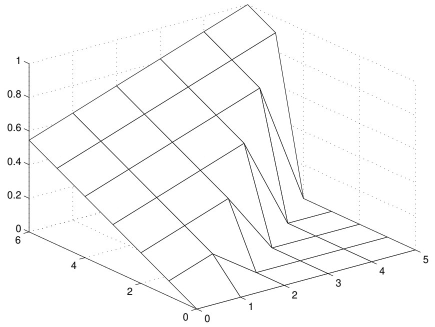

Figure 2.16: *Testing input for the 2D interpolation methods (an input point is situated at each line intersection).*

2.5.2 Linear methods in two dimensions

As there is no magic in extending the 1D approaches into two dimensions by equations 2.36 and 2.37, and as the methods inherit most of their properties mentioned already for one dimension, we will not discuss the 2D interpolations thoroughly but only present the results. Figure 2.17 shows the bilinear interpolation formed as the two-dimensional equivalent of the method from section 2.2.1; the extension of piecewise quadratic interpolation of section 2.2.2 is seen in figure 2.18; the 2D spline base function and the corresponding interpolation is found in figure 2.19.

2.5.3 Growth base functions in 2D

In section 2.3 we introduced a specialized second base function to improve the behaviour of the interpolation through the "hill" base function F_b^q in the regions of monotonically increasing or decreasing data. The 'hill'-based interpolation in 2D needs improvement as well: if you observe the right picture in figure 2.18 a ski-slope may come into your mind. To obtain a result resembling the plane in the input data, we may tend to employ a base function to straighten the interpolation at the input points in the regions of monotone growth. But unlike the 1D case, one base function will not do in two dimensions. The data may increase in the direction of the axis x, in the direction of the axis y, and also diagonally, in a combination of the x and y directions. However, we found that the hill basis plus two supplementary base functions are sufficient, the growth can be tested in the directions of axes x and y separately and the combination of the two base functions serves well, where necessary, for a combined growth.

The growth base function in the direction of the axis x is computed from the 1D growth and the 1D standard base function from section 2.3:

$$F_b^{g_x 2D}(t,u) = F_b^g(t) \cdot F_b^q(u) \tag{2.40}$$

The second growth base function, $F_b^{g_y 2D}$ is constructed in the same way, only with the parameters t and u swapped on the right side, which results in a function rotated by 90 degrees.

The formation of the final interpolating function is driven by the formula

$$\begin{aligned} F(t,u,\vec{x},\vec{y},Z) &= \sum_{i,j} [z_{ij} \cdot F_b^{q2D}(t-x_i, u-y_j) + w_{ij}^x \cdot F_b^{g_x 2D}(t-x_i, u-y_j) \\ &\quad + w_{ij}^y \cdot F_b^{g_y 2D}(t-x_i, u-y_j)] \end{aligned} \tag{2.41}$$

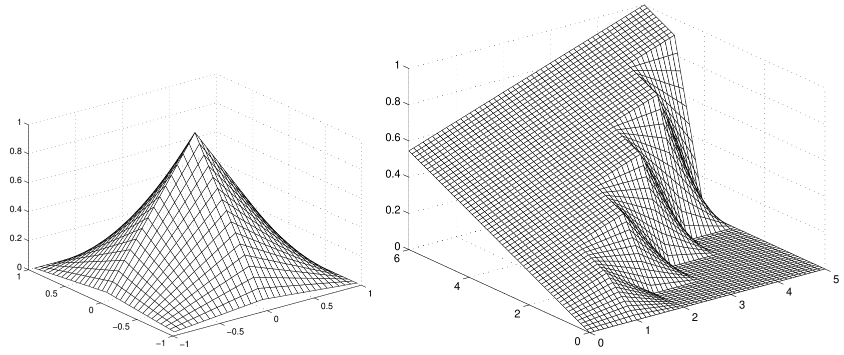

Figure 2.17: *Bilinear interpolation. Left: the base function $F_b^{\triangle 2D}$ obtained as the extension of the one-dimensional base function $F_b^{\triangle}$. Right: bilinear interpolation constructed through the base function on the left.*

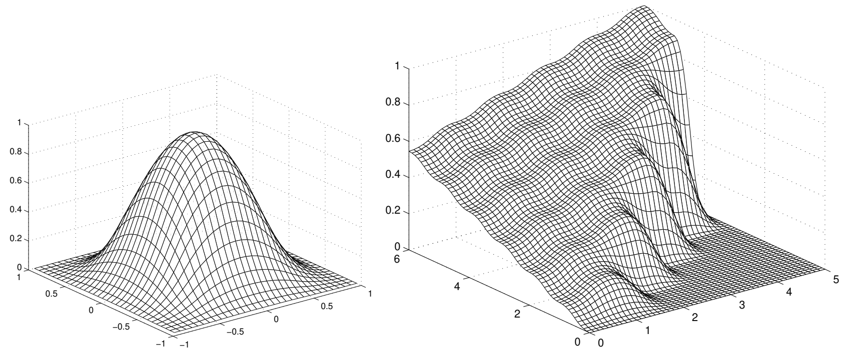

Figure 2.18: *Left: the "hill" base function F_b^{q2D} obtained as the extension of the 1D base function F_b^{q}. Right: interpolation constructed through the base function on the left.*

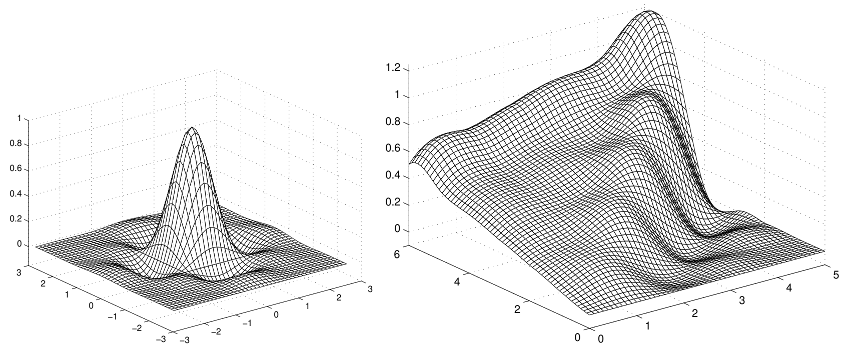

Figure 2.19: *Two dimensional spline interpolation. Left: the spline base function F_b^{s2D} formed as the extension of the one-dimensional base function F_b^{s}. Right: 2D spline interpolation constructed through the base function on the left.*

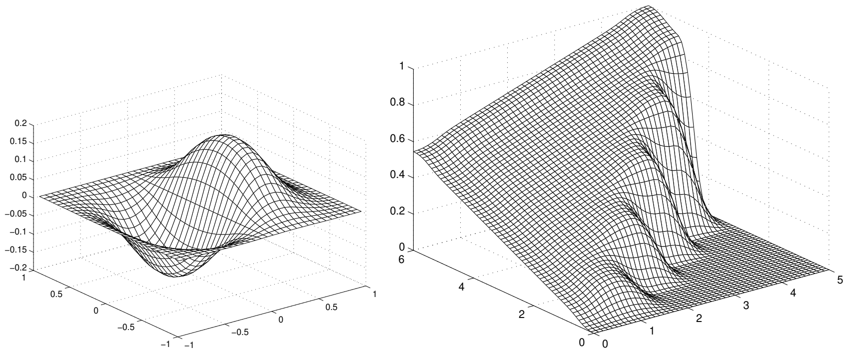

Figure 2.20: *Two dimensional interpolation by three base functions. Left: $F_b^{g_x 2D}$, one of two base functions added in the growing regions (the other, $F_b^{g_y 2D}$, is rotated by 90 degrees anti-clockwise). Right: 2D interpolation constructed through the base functions F_b^{q2D}, $F_b^{g_x 2D}$ and $F_b^{g_y 2D}$.*

where i, j are taken for all input points, and the weights w_{ij}^x and w_{ij}^y are defined by the following rules:

$$w_{ij}^x = \begin{cases} min(z_{i,j} - z_{i,j-1}, z_{i,j+1} - z_{i,j}) & \text{if } z_{i,j-1} < z_{ij} < z_{i,j+1} \\ max(z_{i,j} - z_{i,j-1}, z_{i,j+1} - z_{i,j}) & \text{if } z_{i,j-1} > z_{ij} > z_{i,j+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.42}$$

$$w_{ij}^y = \begin{cases} min(z_{i,j} - z_{i-1,j}, z_{i+1,j} - z_{i,j}) & \text{if } z_{i-1,j} < z_{i,j} < z_{i+1,j} \\ max(z_{i,j} - z_{i-1,j}, z_{i+1,j} - z_{i,j}) & \text{if } z_{i-1,j} > z_{i,j} > z_{i+1,j} \\ 0 & \text{otherwise} \end{cases} \tag{2.43}$$

One of the growth bases is shown together with the final interpolation in figure 2.20.

2.5.4 Interpolation by bicubic Bézier surfaces

Compared to the interpolation through base functions, the extension of the interpolating Bézier curves from one to more dimensions is a little more intricate and the computational complexity increases faster. However, we still may choose Bézier surfaces as the best solution in two dimensions, especially when the input data points are not equally spaced and the summation of base functions is therefore not applicable. While we created a cubic Bézier curve segment by adding two extra control points between two input points in 1D, to interpolate the input data in two dimensions we have to form a surface patch for each rectangle of four input points. This bicubic Bézier surface patch is controlled by 16 points, i.e. the four input points plus twelve additional control points (see figure 2.21). Let us arrange the control points in a 4x4 matrix

$$M = \begin{bmatrix} P_{11} & \dots & P_{14} \\ \vdots & \ddots & \vdots \\ P_{41} & \dots & P_{44} \end{bmatrix} \tag{2.44}$$

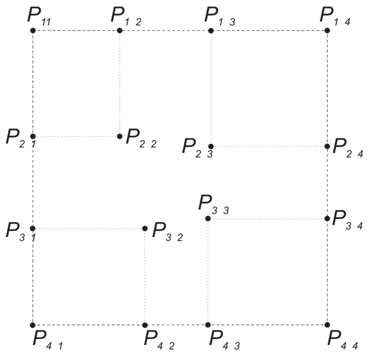

Figure 2.21: *Control points of the bicubic Bézier surface.*

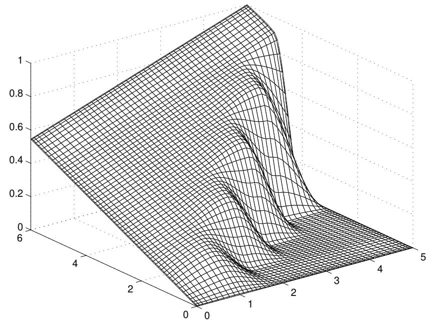

Figure 2.22: *The testing data interpolated by bicubic Bézier surfaces.*

where P_{11}, P_{14}, P_{41} and P_{44} coincide with the input points. Then the Bézier surface can be written in the form

$$B^{2D}(u,v) = [w_1(u), w_2(u), w_3(u), w_4(u)] \cdot M \cdot [w_1(v), w_2(v), w_3(v), w_4(v)]^T \tag{2.45}$$

The weights w_i are the same Bernstein polynomials as in equation 2.31 which controlled the one dimensional Bézier curve. They have the following form:

$$\begin{aligned} w_1(u) &= (1-u)^3 \\ w_2(u) &= 3(1-u)^2 u \\ w_3(u) &= 3(1-u)u^2 \\ w_4(u) &= u^3 \end{aligned} \tag{2.46}$$

The two-dimensional surface retains the nice property of staying inside the convex hull of its control points.

Now return to the initial step of the procedure, placement of the control points. Although at first sight it may look much more complicated than in the one dimensional case, it is not so. We employ a simplifying trick: the position of the control points on the boundary of the rectangle will be determined only from the input points in the direction of that boundary using the one dimensional rules presented in section 2.4. We will then require each control point inside the rectangle to lie in a plane with the input point and the two control points nearest to it. An example may cast some light on this process: P_{11} is an input point, P_{12} and P_{21} are two control points on the patch boundary placed using the positions of P_{11} and some other input points, and P_{22} is the non-boundary control point forced to form a rectangle with the points P_{11}, P_{12} and P_{21}.

Employing these independent rules for the directions of axes x and y, then supplying the control points inside the bicubic Bézier surface, that is a straightforward procedure which results in a natural extension of 1D Bézier curves into more dimensions. The whole surface will be a smooth one, the individual patches are joined with continuous zero and first derivatives.

Figure 2.22 shows this method applied to interpolation of our sample data. Note that the growing plane is perfectly flat, but the lines in the mesh are not equally dense in the whole region. This is caused by the way the surface is computed: we employ a constant step in the variables u and v, which leads to varying increments in the variables x and y computed through the equation 2.45. To overcome this unpleasant aspect of the method, we would have to employ a more sophisticated procedure to compute the suitable values of u and v for a chosen position (x, y).

2.6 Summary and Conclusion

We mentioned in the introduction several properties which an 'ideal' interpolating function should exhibit, then we discussed whether the particular methods do or do not behave in the desired way. Here we try to summarize the notes spread throughout the text and compare the interpolating functions properties.

interpolation method	*no. of derivatives*	*no. of input points per output point*[a]	*comonotone*	*homogeneous*	*linear*	*extension to more dimensions*
piecewise linear/bilinear	0	2	yes[b]	yes	yes	easy
"hill" basis	1	2	yes[c]	yes	yes	easy
cubic spline	2	10-14[d]	no	yes	yes	easy
approximate basis	1	> 2	no	yes	yes	easy
more base functions	1	3	yes	yes	no	possible
Bézier cubic/bicubic	1	3[e]	yes	yes	no	possible

[a]number given in 1D, the value in dimension n computed as the n-th power of this number
[b]first derivative does not exist at the input points
[c]first derivative equal to zero at the input points
[d]theoretically infinity, the number 10-14 stated in [35] for a reasonable precision
[e]three input points play role during the placement of the control points; a point at the output is then found using the position of these (four) control points

Table 2.1: *Summary of the properties of the interpolating methods.*

An overview is seen in table 2.1. The first column states the number of existing continuous derivatives of the interpolation. Most frequently one derivative exists which means that the resulting function is smooth.

The second column in the table gives the number of input points which take part in computation of the value of one output point in one dimension. This number describes the local (smaller number) or global character of the procedure. The simplest methods place the output point based only at the values of its nearest neighbours (2 in one dimension), while cubic spline, to ensure the continuity of the second derivative, uses 10 to 14 input points (for a chosen precision). The last two methods (more base functions and Bézier) need the third input point to decide on the type of region the output point belongs to (growing, decreasing, other), and on the rule which should be employed for its placement.

Another comment concerns the last column – extendibility to more dimensions. The extension is very easy with the linear methods: the new base functions are obtained as the Cartesian product of the 1D case. The transition is only a little more difficult for more base functions, as different base functions must be added to the result for the growth in different directions. The Bézier-based approach is more complicated: in more dimensions we need more control points for the surface, although they can be found using simple adjustment of the 1D procedure.

The computational complexity of the methods is also related to the number of dimensions. Basically speaking, the cost of all the methods depends linearly on the number of input points. The basis-based methods (all except Bézier) can be implemented very efficiently if the base function is sampled into a matrix for a given resolution; then the cost of the method is influenced by the width of such matrix (compare the second column in table 2.1). The Bézier interpolation depends also linearly on the number of input points, but is more computationally expensive because of the need to place the control points first, and because of the cubic parametric equations defining the function's shape from the control points.

To conclude we may say there is no perfect interpolation for all cases, we can choose from a variety of methods in accordance with the properties we consider more important for the result. Linear methods are easier to implement, computationally more efficient, but suitable only for a regularly spaced set of input data; Bézier curves or surfaces used for interpolation are very flexible and easily controllable through the placement of their control points; and the method based on multiple base functions is somewhere in between, showing some good and some bad properties of either of the above mentioned approaches.

3

Approximation by a Monotone Function

3.1 Motivation, and complications

The problem of approximation by a (piecewise) monotone function is in our case motivated by noisy data from color-coded, or rainbow range finder (RRF), a system for 3-D reconstruction by active computer vision techniques (see the Introduction and figure 1.1 on page 2). In this section we discuss the data properties in more detail and formulate the task.

3.1.1 1D monotonicity and 2D continuity

The digital image grabbed by the camera in the RRF system is formed by a two-dimensional array, where each element (pixel) contains the information on colour projected onto the image plane from a certain angle from the scene. This colour information is usually stored in triples (R, G, B) representing the red, green and blue component. Let us name the digital image A, with components $(a^R_{i,j}, a^G_{i,j}, a^B_{i,j}),\ i = 1, \ldots, m,\ j = 1, \ldots, n$ forming a $m \times n$ matrix. As the light projected into the scene is composed of monochromatic stripes, also the observed light should be monochromatic (except when inter-reflections in the scene occur) and we can convert the image into another two-dimensional array M of m rows and n columns, where each element $m_{i,j}$ will represent the wavelength of the observed colour. We do not deal with the problem of determining the wavelength from the RGB signal here; we consider this procedure as already completed, so that the input of our algorithms are the estimated wavelengths in the pixels.

Let the columns of M be parallel to the monochromatic light planes projected from the light source. Then values in a row of matrix M should be monotonically increasing, $m_{i,j} < m_{i,j+1}$, if there is one continuous surface lit continuously by the rainbow light source. Figure 1.1 on page 2 illustrates such a situation.

If there are non-convex objects in the scene (with bays or holes lit non-continuously), or if there is more than a single object visible, the values $m_{i,j}$, for a fixed i and increasing j, are expected to be only *piecewise monotonically increasing*, with discontinuities at region boundaries. An example is seen in figure 3.1. Note that shadows in the scene may lead to some dark, no information containing regions in the image, in which case the corresponding values $m_{i,j}$

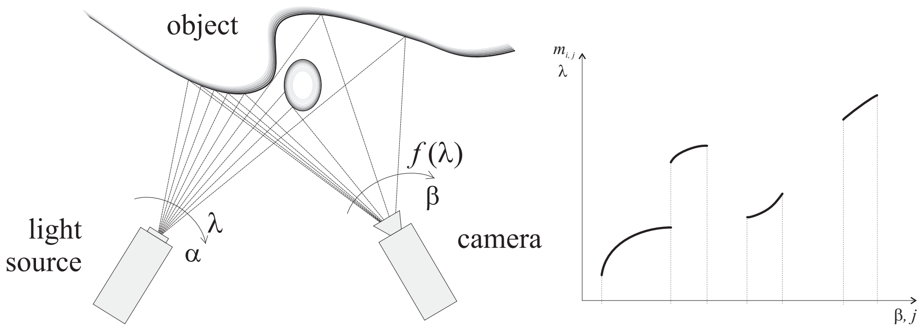

Figure 3.1: *More complicated setting of the scene for the rainbow range finder (left): if the visible surfaces are lit non-continuously, the wavelength of light observed by the camera will be piecewise monotonically increasing (right).*

would be undefined.

If two neighbouring elements $m_{i,j}$ and $m_{i+1,j}$ or $m_{i,j+1}$ of the matrix M belong to the same surface, they will probably be lit similarly (except if there are some shadows cast by obstacles) and also their values will be close to each other. Although no monotonicity can be presumed between rows of the matrix M, this *piecewise continuity* of neighbouring elements both in rows and columns of M should be taken into consideration during further data processing.

We should remark that similar situations are encountered in most other range finders as well, e.g. in those based on structured light or laser plane. They differ basically only in the causes and amplitudes of noise.

3.1.2 Noise

We mentioned above that the values $m_{i,j}$ should be (piecewise) monotonically increasing for increasing j. In real experiments the noise, imprecise colour identification, low illumination, discretization, quantization, and properties of the optics (see [38]) cause the data to lose this property of monotonicity.

3.1.3 Task formulation

Our task is motivated by the previous paragraph. As the input we obtain the matrix M whose elements $m_{i,j}$ are expected to be, but not necessarily are, piecewise monotonically increasing in the direction of increasing index j. We want to restore the desired data properties, to remove noise and to ensure monotonicity[1] of the data, i.e. to construct either a piecewise monotone matrix P of m rows and n columns, with elements $p_{i,j}$ respecting the condition

$$p_{i,j} < p_{i,j+1} \qquad \begin{array}{l} \text{for all } i = 1, \ldots, m, \; j = 1, \ldots, n-1 \\ \text{if there in no discontinuity between } p_{i,j} \text{ and } p_{i,j+1} \end{array} \tag{3.1}$$

[1]The notion of *monotonicity* here is different from that in section 2.1.3. The comonotone interpolation there respected the increasing/descending properties of input data, while the monotone approximation here intends to ensure (piecewise) increasing output even for some non-monotone inputs.

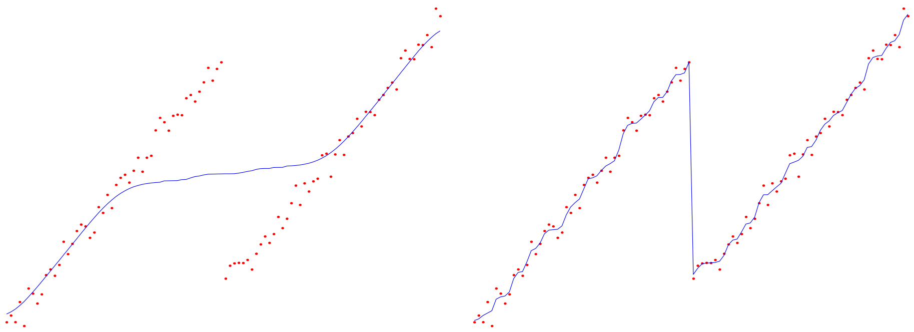

Figure 3.2: *Comparison of a continuous (left) and discontinuous (right) monotone approximation to the same input data. (Legend: dots ... input data; line ... approximation.)*

or to find a piecewise monotone function $P(x, y)$ with

$$\begin{array}{l} P(x,y) < P(x+\epsilon,y) \quad \text{for all } x,y \text{ from the input space, } \epsilon > 0 \\ \text{if there in no discontinuity between } P(x,y) \text{ and } \; P(x+\epsilon,y) \end{array} \tag{3.2}$$

Both matrix P and function $P(x, y)$ should approximate the input values $m_{i,j}$, i.e. be as 'close' to the original values as possible while conforming to the monotonicity criterion. We return to this point in section 3.1.7.

In this work we study the problem of the construction of a *piecewise monotone approximation* to the original data. We concentrate on the discrete version and suggest a method to find a piecewise monotone matrix P for given input data M. The continuous alternative can be either deducted directly from the algorithms below (some of which originate in the continuous domain) or obtained as an interpolation to the array P, an interpolation which may be computed by a piecewise continuous extension of any of the monotonicity preserving interpolation methods presented in chapter 2.

Before we continue to build up our piecewise monotone approximation algorithm, let us discuss some aspects and problems inherent in the input data in more detail.

3.1.4 Crucial edge detection

In both discrete and continuous task formulations, (3.1) and (3.2), a modest phrase appeared: "if there is no discontinuity ...". This condition introduces strong nonlinearity into the problem and causes much trouble with problem analysis.

Compare two illustrations in figure 3.2. The same input data lead to very different approximation results, in dependence on whether we classify the input as continuous or discontinuous. This makes it an ill conditioned task: the (correct) output depends non-continuously on the change of the parameters for discontinuity detection, or, stated the other way round, *the (correct) output depends non-continuously on a small change in the input data.*

The detection of discontinuities (called edges in image processing) becomes the decisive step of the whole approximation procedure. If we make a mistake in the edge detection, the (smoothed) piecewise monotone result may differ significantly from the input. We should mention that this risk is higher with omitted discontinuities than if we detect more edges than necessary; in the latter case, the output may still be noisy and unpleasant to view, but it will be close to the initial values.

3.1.5 Level of belief

We mentioned in section 3.1.1 that the input of the approximation procedure, array M, was formed by a transformation of a colour image A into a matrix where each element stored the wavelength of light. Some information is lost by this transform; for example, matrix M contains no information about brightness of observed light. As the information in brighter pixels is more reliable (with higher signal-to-noise ratio), it seems reasonable to extract one more array from A, array of weights $W = (w_{i,j}),\ i = 1, \ldots, m,\ j = 1, \ldots, n$, and use this information later during approximation, with elements with more weight (of higher level of belief) influencing the output more significantly.

Also other sources or types of information may contribute to the weights $w_{i,j}$, such as the precision and reliability of colour identification in a particualar wavelength interval (due to the hardware properties), consistency of a local neighbourhood of the pixel (i, j), etc.

3.1.6 2D continuity revisited

We have formulated our task as a piecewise monotone approximation which should not destroy the discontinuities of the data. We have also shown in section 3.1.4 how the edges in the input data influence the output. Couldn't we detect the discontinuities first, separate the data into (one dimensional) continuous regions, and approximate them by a monotone approximation, a method easier to analyze and solved previously (see section 3.2.1 for an overview)?

Unfortunately, it seems that such a simplification of the task by separation into subproblems is not trivial. In fact, the method we suggest later in this text detects the edges prior to approximation, but the division of the input space into distinct regions is complicated by the desired two-dimensional continuity of both the input and the result. First, the (nearly monotone parts of) rows of the matrix M cannot be processed separately as that would omit the influences from the neighbouring rows and could lead to a creation of a new inter-row discontinuity. Second, the edge-separated two dimensional regions may (and will) have a very complex boundary, hence a row belonging to a single continuous region may be itself only piecewise continuous, and extracting such region would not be of any help.

3.1.7 On optimality of solution

In the approximation theory, the main results often consist in computing an error bound, taken e.g. as the maximum difference between the original data and the approximating function, or in choosing the approximation from a chosen class of functions to optimize some criteria, e.g. the square distance.

We have not based the solution on a class of functions or a criterion to optimize. Interesting results are reviewed in section 3.2.1 for one-dimensional, continuous problems. In our case, the analysis is complicated; two dimensions, discontinuities, non-rectangular, irregular regions, all this makes the piecewise monotone approximation a non-trivial task. An optimality theorem would represent a beautiful theoretical result (if attainable at all), whose importance is diminished by its inevitable dependence on the discontinuity detection. Moreover, the monotone function closest to the original data would lead to periods of zero (or very small) derivative in each interval where the monotonicity in the input was corrupt; such an 'optimal' result is usually not desirable (for the physical properties of the rainbow range finder: the periods of zero derivative of the wavelengths may occur only in areas where an observed object's surface is parallel with the rays coming from the light source) and means that we have removed only a part of the noise present in the data.

So, instead of trying to construct an optimal solution in the least squares sense, we concentrate on practically applicable local operations to accomplish the task of a piecewise mononone approximation.

3.2 Background

In this section we review previous work on monotone approximation, and introduce briefly the topics needed for, or connected to, our piecewise monotone approximation, which is going to be presented in section 3.3.

3.2.1 Previous work on monotone approximation

Monotone interpolation and approximation, as a special case of constrained approximations, have played a considerable role among the problems studied in the field of approximation theory. We list some of the main ideas, although the survey cannot be considered exhaustive.

The first group of papers deals with (co-)monotone polynomial interpolation. The early papers include [25, 37] and [16]. In [32, 31, 28, 7] the authors study comonotone approximation to a function f (or to a sequence of function samples at discrete points x_i), i.e. an approximation which increases and decreases simultaneously with f; the development consists basically in that newer papers improve estimates of the degree of polynomial needed for a comonotone interpolation, and the estimates of the error between the original function and the approximation for a chosen degree of the approximation polynomial.

Other papers on approximation with the monotonicity constraints include [3] where the original function is approximated by reciprocals of polynomials, and [9] studying approximations constructed as a composition of linearly indepedent functions $g_i(x)$ by the equation $P(c,x) = \sum_{i=1}^{n} c_i g_i(x)$, intending to find the coeficients c to minimize the difference between the approximation P and the original function f subject to the constraint on the k-th derivative of the approximating polynomial $P^{(k)}(x) \geq 0$. Choosing $k = 0$ leads to positive, $k = 1$ to monotone approximation.

Then there is a group of papers on constrained spline interpolation and approximation: starting with piecewise monotone interpolation to discrete data points in [30], we only mention comonotone approximation of [23] and copositive of [18], and come to two papers on cubic splines which will be described in more detail.

Fisher and Opfer in [13] consider the fact that a spline which interpolates positive function values is itself not necessarily positive, and suggest an algorithm to produce a non-negative spline interpolant provided that the function to be interpolated is positive. They accomplish that by adding few extra knots in the intervals where the unconstrained cubic spline was negative; by choosing the additional knots carefully they try to minimize the strain energy, and obtain formulae for 'optimal' splines which conform with the constraint and leave the original natural spline (minimizing the strain energy globally) unchanged except for the intervals (between a pair of neighbouring input points x_i and x_{i+1}) where a negative value was detected.

The article [1] by Andersson and Elfving is the closest to our problem of rainbow range finder data smoothing. The authors study the reconstruction of a function of one variable from given (possibly noisy) sampled data values, under constraints on monotonicity and shape of the result. After a historical overview of the field of monotone splines, the constrained interpolation and approximation problems ($\mathbf{P}_i$ and $\mathbf{P}_a$, respectively) are formulated as follows:

$$\begin{aligned}\mathbf{P}_i\text{: } & \text{Minimize} \int_{x_1}^{x_n} f''^2(x)\,dx \\ & \text{when } f(x_i) = y_i,\ i = 1 \ldots n, \\ & \text{and } f \in W^2(x_1, x_n) : \varphi(x) \leq f'(x) \leq \psi(x).\end{aligned} \tag{3.3}$$

$$\begin{aligned}\mathbf{P}_a\text{: } & \text{Minimize } p \int_{x_1}^{x_n} f''^2(x)\,dx + (\vec{z} - \vec{y})^T Q (\vec{z} - \vec{y}) \\ & \text{when } A\vec{z} = \vec{d},\ f(x_i) = z_i,\ i = 1 \ldots n, \\ & \text{and } f \in W^2(x_1, x_n) : \varphi(x) \leq f'(x) \leq \psi(x).\end{aligned} \tag{3.4}$$

where f is the monotone function to be found, f' and f'' its first and second derivatives, pairs (x_i, y_i) represent the input points, $W^2(a, b)$ is the Sobolev space of all functions f such that f' is absolutely continuous on (a, b) and $f'' \in L^2(a, b)$, and the functions φ and ψ impose constraints on the function derivative (for monotonicity only, set $\varphi = 0$ and $\psi = \infty$). The matrix Q represents correlation between stochastic errors in the data vector $\vec{y}$, $p > 0$ is a smoothing parameter, and the equation $A\vec{z} = \vec{d}$ with A a $m \times n$ matrix, imposes m linear constraints on the vector $\vec{z} = (f(x_1), f(x_2), \ldots, f(x_n))$.

The problems are then solved in [1] separately for fixed and free end derivatives of the interpolating spline, by projection operators and Newton-type iterative algorithms (which can hardly be called straightforward).

A brief historical remark: we find it interesting how the approaches differ depending on the year when the papers were compiled. While older texts study polynomials (of any order) and the main result often consists of an existence theorem within a chosen error bound, newer articles tend to employ a class of functions of limited order (and possibly their linear composition), formulate the task as minimization problem with respect to a chosen criterion (typically least squares), solve it, and present algorithms and numerical experiments.

The last group to be listed, [5, 19, 41] represent a sample of papers dealing with approximations under monotonicity constraints from a theoretical point of view.

3.2.2 Linear scale space

The task of monotone approximation is closely linked to that of replacing the input by a smoother version with fewer decreasing regions, which means also with fewer local minima and maxima. An illustrative formalism expressing the idea of gradually removing local extrema was presented by Lindeberg in [24]. Starting from the original data $f(x)$, $x \in R^N$, he adds a *scale parameter* t and constructs *scale-space representation* L in the following way:

$$\begin{aligned} L(x,0) &= f(x) \\ L(\cdot,t) &= g(\cdot,t) * f \end{aligned} \tag{3.5}$$

For $t = 0$ the representation is identical with the original data, a higher level $L(x,t)$ is obtained as a convolution of f and a smoothing kernel g.

The smoothing kernel g is the Gaussian kernel, given in the continuous case by

$$g(x,t) = \frac{1}{(2\pi t)^{N/2}}\ e^{-x^T x/2t} \tag{3.6}$$

As shown by Lindeberg, the Gaussian kernel is the only one to fulfill a set of requirements which includes the crucial property of never introducing new local extrema into the data during the smoothing process. Other useful features include the semigroup property, a wider Gaussian may be obtained by a convolution of two narrower kernels

$$g(\cdot,t_1+t_2) = g(\cdot,t_1) * g(\cdot,t_2) \tag{3.7}$$

Together with the associativity of convolution, equation (3.7) leads to the fact that a scale t_2 of the representation L may be constructed from a smaller scale, t_1, and not necessarily directly from the original data. This is called *cascade smoothing* and reduces computational cost:

$$\forall t_1, t_2 \in R_0^+,\ t_2 > t_1 : \quad L(\cdot,t_2) = g(\cdot,t_2-t_1) * L(\cdot,t_1) \tag{3.8}$$

In the case of discrete input data $f(x_i)$, $i = 1,\ldots,n$, the convolution in equation (3.5) can be understood as smoothing through weighted averaging:

$$F(x) = \frac{\sum_{i=1}^n\ f(x_i)\cdot g(x_i - x)}{\sum_{i=1}^n\ g(x_i - x)} \tag{3.9}$$

The scale parameter t corresponds to the width of the smoothing kernel (or weighting function) g, higher t means more input points contribute significantly to the computation of an output point, and the output data become smoother. This important observation is illustrated in figure 3.3.

The notion of scale plays an important role in computer vision; many aspects and applications are discussed in the referenced book [24] and in the collection [36]. However, the linear scale-space concept has its limitations: smoothing uniformly by a rotationally symmetric Gaussian kernel may not only remove undesirable local extrema but also harm important features of the image and dislocate edges. To overcome these drawbacks, nonlinear extensions may be employed (see sections 3.2.3 and 3.3.2).

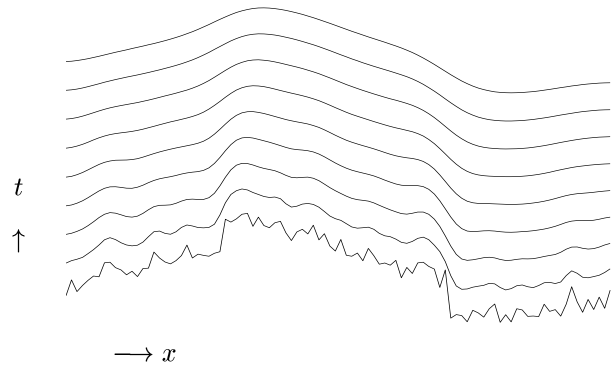

Figure 3.3: *Linear scale-space representation: signal becomes smoother as the scale parameter t increases.*

3.2.3 Nonlinear diffusion

One important drawback of homogeneous linear scale-space from the preceding paragraph is that it may blur important features of the image, such as edges, while reducing the noise. Non-linear scale-space theories based on partial differential equations offer one possible solution to this problem and have attracted considerable attention recently; an excellent introduction and overview can be found in [47].

Diffusion as a physical process equilibrates concentration differences without creating of destroying mass. This observation can be translated into the following:

- Fick's law
$$j = -D \cdot \nabla u \tag{3.10}$$
says that a concentration gradient ∇u causes a flux j which aims to compensate this gradient; the relation between ∇u and j is described by the diffusion tensor D.

- The continuity equation expresses the idea that mass can be only transported, not created or destroyed,
$$\partial_t u = -\mathrm{div}\; j \tag{3.11}$$
where t stands for time.

The two equations can be combined to yield the *diffusion equation*

$$\partial_t u = \mathrm{div}\; (D \cdot \nabla u) \tag{3.12}$$

The simply looking equation (3.12) and other forms derived from it describe a large class of image processing operations. If we replace the diffusion tensor D by a scalar value, the process is equivalent to uniform smoothing and the linear scale-space from the preceding section. Other types and applications include smoothing while preserving edges (if the diffusion is driven by a suitable function of the gradient image), coherence-enhancing and many others (see for example [47, 36, 8, 22, 26]).

If many operations can be described in the mathematically sound framework of partial differential equations, computational efficiency represents an argument against using it; that is why many papers concentrate on numerical implementation of the nonlinear diffusion scheme [46, 2, 21].

3.2.4 Edge detection

At the beginning of the data processing chain, we need to obtain information on discontinuities in the input image which should be respected later during the approximation phase. In computer vision, the discontinuities are called edges and the operators extracting them *edge detectors*. The book [40] can provide the interested reader with an overview, we mention here only Canny edge detection which we have employed in our application.

The Canny operator processes the input in three stages. First the image is smoothed by Gaussian convolution to make the procedure less sensitive to noise. Then a gradient image is computed from the smoothed input by a simple 2-D first derivative operator. Last, a tracking mechanism extracts the edges: it starts where the gradient is greater than first threshold T_1, and continues in both directions along the path of local gradient maximum until the gradient value falls below a second threshold, T_2. Non-maximum suppression is performed during the tracking process: pixels not on the maximum-following path are discarded so that an edge results in a thin line in the output. We refer to [20] or [14] for a more detailed description.

The threshold T_1 is greater than T_2 - this hysteresis ensures that tracking is started at strong edges only, and helps not to break noisy edges into multiple edge fragments. Together with σ (width of the Gaussian smoothing kernel), the three parameters can be tuned to make the Canny operator sufficiently insensitive to noise while not omitting any important edges.

3.3 Filtering to ensure piecewise monotonicity

3.3.1 Overview

Motivated by the situation presented in section 3.1, we need to approximate experimental, noisy data by a (piecewise) monotonically increasing function. This section introduces a solution consisting of edge detection (to prevent interregion smoothing) combined with a non-linear, (semi)locally adaptive filter to remove false local extrema and ensure monotonicity of the data in continuous areas. An overview of the method is sketched in figure 3.3.1.

First, we apply an edge detector (see section 3.2.4 for details) to locate discontinuities, or edges, in the data. The result of the edge detection is stored in an edge map, which is employed later during the approximation step.

As we want to keep the separate parts in the input data (possibly belonging to different objects in the scene to be reconstructed) independent, no subsequent operations should destroy the discontinuities found by the edge detector. Section 3.3.2 shows how to build a discrete edge-preserving scale-space representation, i.e. how to smooth the input data while not harming the a priori

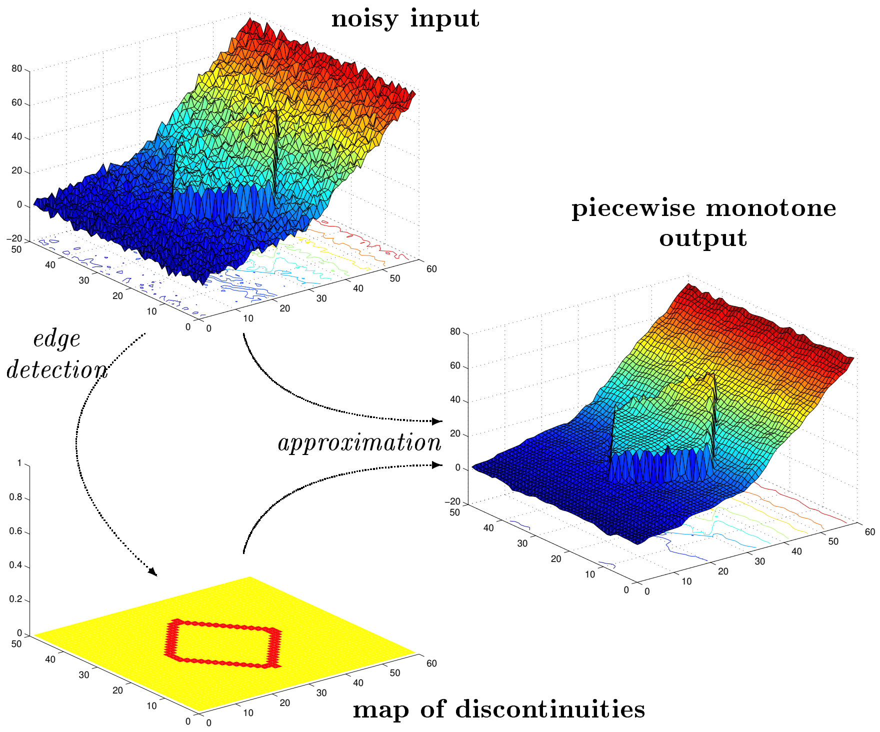

Figure 3.4: *Overview of the method: approximation is preceded by edge detection and employs its results to preserve discontinuities.*

defined discontinuities. The subsequent sections, 3.3.3 and 3.3.4, suggest to use this edge-preserving smoothing to ensure the desired piecewise monotonicity of the input data.

3.3.2 Discontinuity-preserving smoothing

It has been mentioned that the linear scale-space representation (section 3.2.2) may harm important features of the data while smoothing them, and that non-linear diffusion may offer a solution to this problem (section 3.2.3). We suggest a different approach here: we construct an edge-preserving scale-space representation in a similar way as the linear one (i.e. by convolution with a Gaussian kernel or by weighted averaging), but we prohibit smoothing across edges by omitting points at the boundary from the equation. We present the algorithm for the discrete case (discrete both in space and scale).

Let's start with the (one-dimensional) input values m_i, $i = 1, \ldots, n$, and an array of weights θ_i associated with those values, setting

$$\begin{array}{ll} \theta_i = 1 & \text{for points inside a continuous region} \\ \theta_i = 0 & \text{in the neighbourhood of an edge.} \end{array} \tag{3.13}$$

Then the edge-preserving scale-space representation is for $t = 0$ equal to the original values, $E(k, 0) = m_k$, while smoother levels are obtained by recurrent

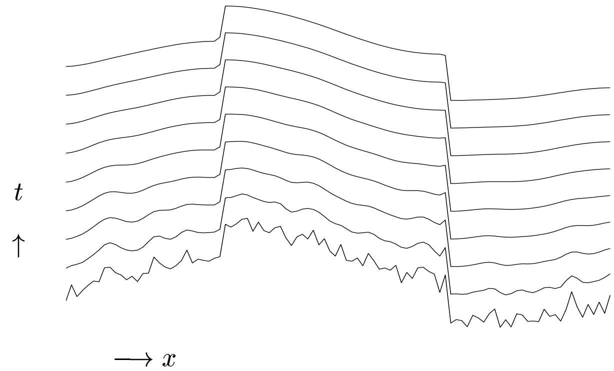

Figure 3.5: *Edge-preserving scale-space representation: signal becomes smoother as the scale parameter t increases, but smoothing is not allowed across discontinuities.*

evaluation of the formula

$$E(i, t+\Delta_t) = \frac{\sum\limits_{k=-1}^{1} E(i+k,t)\cdot\theta_{i+k}\cdot g(k)}{\sum\limits_{k=-1}^{1} \theta_{i+k}\cdot g(k)} \tag{3.14}$$

where Δ_t is a unit scale step which corresponds to smoothing by a discrete analogue of the Gaussian kernel (3.6), given by the values

$$(g(-1),\ g(0),\ g(1)) = \left(\frac{1}{4},\ \frac{1}{2},\ \frac{1}{4}\right) \tag{3.15}$$

When convolved with themselves repeatedly, these values lead to a wider Gaussian smoothing kernel. The cascade smoothing by iterative application of equation (3.14) does not represent an increase in complexity compared to a direct averaging in a wider neighbourhood, and the small smoothing window allows an easy control of which points are involved into the computations near an edge so that smoothing across object boundaries may be avoided.

Transformation of the one-dimensional equation (3.14) to a higher dimension is straightforward; for two dimensions the weighted average will be computed in a 3×3 smoothing window. Let the input data form a $m\times n$ matrix M of elements $m_{i,j}$; we run an edge detector on the data, and construct a matrix of weights Θ with the elements

$$\begin{array}{ll} \theta_{i,j}=1 & \text{inside a continuous region} \\ \theta_{i,j}=0 & \text{in the neighbourhood of an edge.} \end{array} \tag{3.16}$$

Then again the first level of the edge-preserving scale-space representation coincides with the input data, and higher levels of scale are computed by the recurrent system

$$E(i,j,0) = m_{i,j}$$

$$E(i,j,t+\Delta_t) = \frac{\sum_{k=-1}^{1} \sum_{l=-1}^{1} E(i+k, j+l, t) \cdot \theta_{i+k,j+l} \cdot g(i) \cdot g(j)}{\sum_{i=-1}^{1} \sum_{j=-1}^{1} \theta_{i+k,j+l} \cdot g(i) \cdot g(j)} \tag{3.17}$$

We mentioned in section 3.1.5 that it may be desirable to assign a weight $w_{i,j}$ to each input value $m_{i,j}$. These weights, representing the initial level of belief in the measured data, can be easily incorporated into the equations if we replace (3.16) by

$$\begin{array}{ll} \theta_{i,j} = w_{i,j} & \text{inside a continuous region} \\ \theta_{i,j} = 0 & \text{in the neighbourhood of an edge.} \end{array} \tag{3.18}$$

As we construct the edge-preserving scale-space representation E, lines or surfaces become smoother at larger scale t within continuous regions, while the discontinuities between regions remain unharmed and well localized. This behaviour is illustrated in figure 3.5. It was created for the same input as figure 3.3; you can confront the two results.

3.3.3 Monotonicity through global smoothing

We may exploit the above mentioned variation-diminishing properties of the edge-preserving scale-space representation E to construct a piecewise monotone approximation to the input data. The idea remains the same in one and two dimensions, we stick to 1D for the simplicity of explanations.

We start from the premise that ideal data h increase monotonically in homogeneous regions, i.e.

$$\begin{array}{l} h(x) \leq h(y) \\ x, y \in R,\ x < y \ \ \text{and there is no edge between } x \text{ and } y \end{array} \tag{3.19}$$

while experimental, noisy data f violate this monotonicity condition. Starting from $f(x) = E(x,0)$ we construct the scale-space representation E, and find t_0 the smallest scale parameter for which the data $E(x,t_0)$ do increase monotonically except for discontinuities. We set the filtered data F equal to the this level of scale-space representation, $F(x) = L(x,t_0)$.

We may remark that we are sure to have constant, i.e. non-decreasing data in each continuous region at scale $t = \infty$. However, that certainly would not be a desirable result of the approximation. With reasonable input data we should reach monotonicity at a smaller scale.

3.3.4 Locally adaptive scale parameter

With the approach described in the previous paragraph it may easily happen that monotonicity violation at one end of the input space requires more smoothing, and thus the scale is incremented (and the data smoothed, i.e. details of

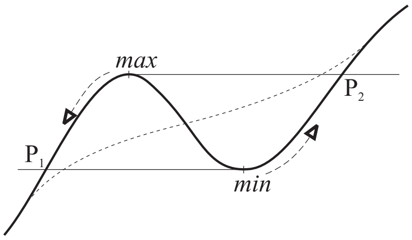

Figure 3.6: *Detection of local extrema: scale has to be increased (at least) in the interval* $[P_1,\ P_2]$. *(Original function in bold, desired approximation in dotted line)*

shape destroyed) at the whole input space. In such a case, a local increment of the smoothness may be preferred to a global one. Changing the method from the previous paragraph slightly we obtain the following: the scale parameter will depend on position in space, and its value will be incremented only locally where needed to ensure monotonicity of the data. This local scope of smoothing will be limited only by the requirement of continuity imposed on the scale parameter in each edge-separated region. An algorithmic specification follows:

1. Initialization: scale parameter $t(x) = 0$; construct the edge-preserving scale-space representation $E(x,t)$ starting from the input data: $E(x,0) = f(x)$.
2. Check the function $E(x,t(x))$ for local maxima and minima, i.e. look for intervals where monotonicity condition is violated (except across discontinuities); see figure 3.6.
3. Increase the scale parameter $t(x)$ in unsatisfactory intervals (if any) and return to step 2 of the procedure.
4. Filtered output $F(x) = E(x,t(x))$, (which is a picewise continuous surface extracted from the scale-space E).

To illustrate the behaviour of the (piecewise) monotone approximation described in this chapter, we present several figures. Figure 3.7 shows phases of a noisy input being transformed into a picewese monotone function using the procedure presented above. A comparison of the results of a global smoothing and the locally adaptive scale parameter is visualised in figure 3.8.

An example of the locally adaptive filter performance when applied on noisy, piecewise continuous two-dimensional data can be seen in the introductory figure 3.3.1. Last but not least, figure 3.9 presents results of the filtering of experimental data from rainbow range finder, although only for a single continuous plane in the scene.

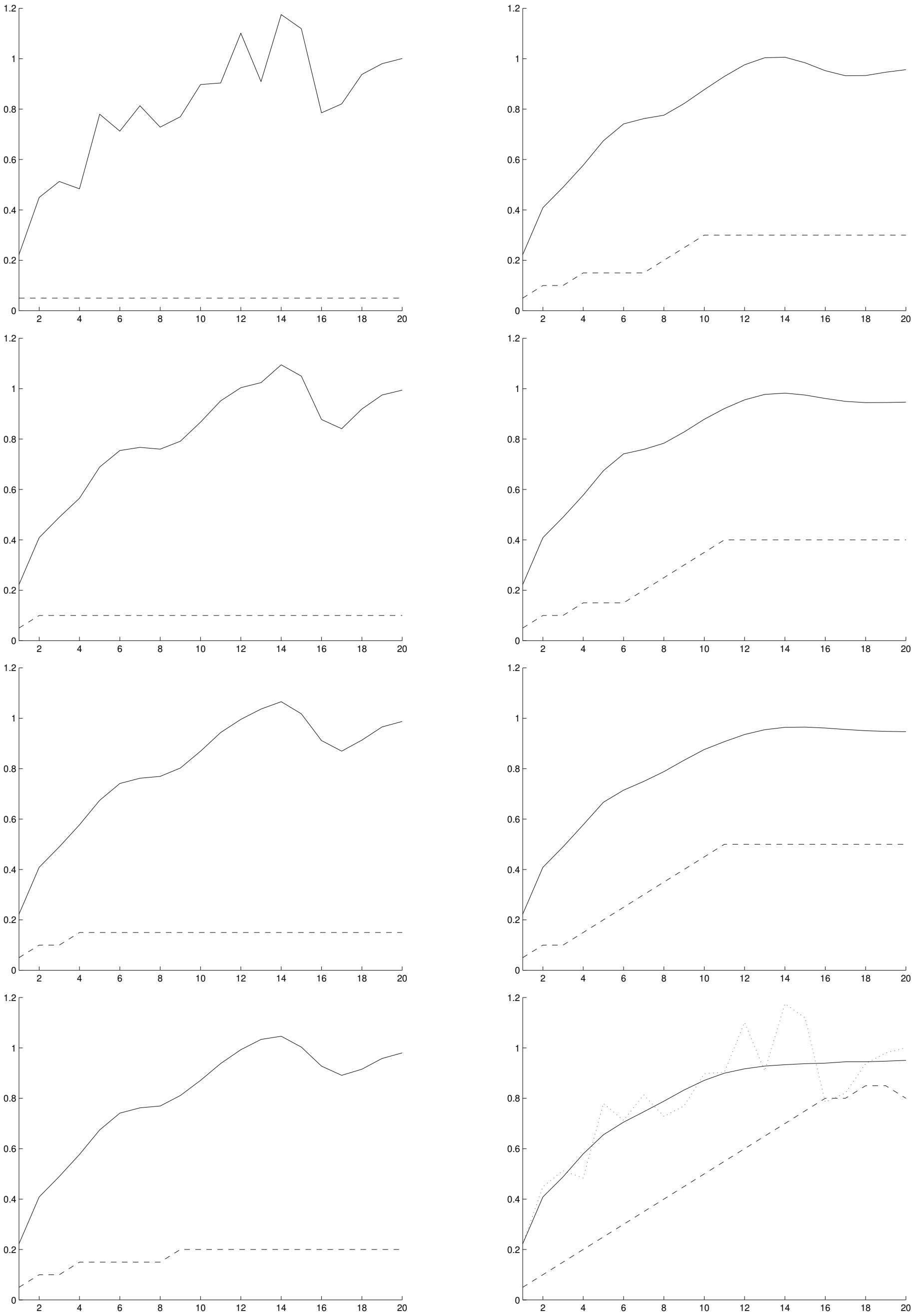

Figure 3.7: *Simulation of how the monotone approximation with a locally adaptive smoothing parameter develops. Left, top to bottom: input data, iterations 1, 2 and 3. Right, top to bottom: iterations 5, 7, 9, 16 (final).*
Legend: approximating function (solid line), level of smoothing (dashed line), input data (dots, only in the last picture).

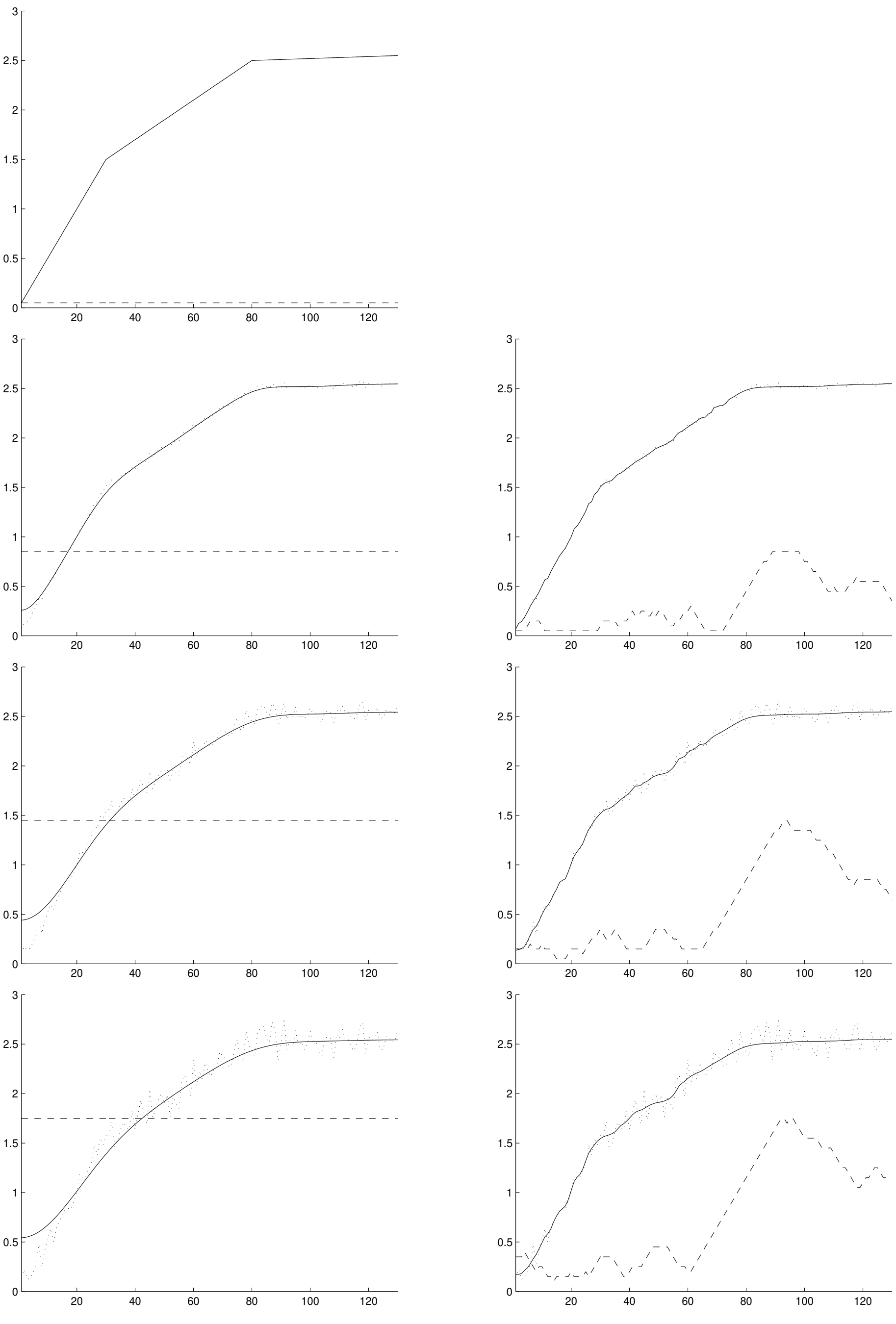

Figure 3.8: *Global and locally adaptive smoothing. Top: input data; from top to bottom: level of additive noise increases. Left: smoothing to ensure monotonicity with a global scale (section 3.3.3). Right: locally adaptive scale parameter (section 3.3.3).*

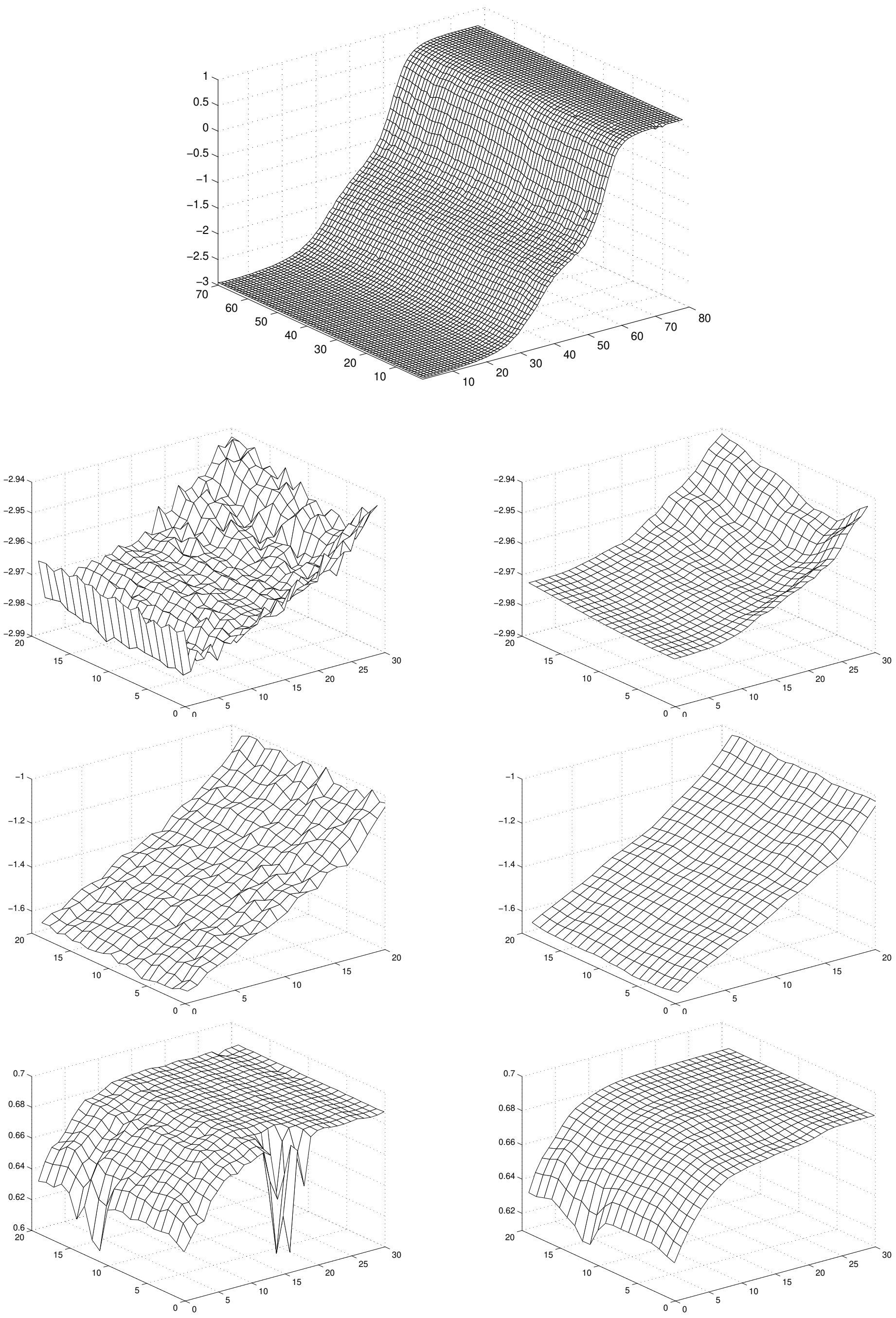

Figure 3.9: *Data from rainbow range-finder. The whole input is shown at the top, enlarged parts below: original on the left, filtered data on the right.*

4

Conclusion, or prelude

Motivated by the expected properties of the data from rainbow range finder (RRF), we have concentrated on monotonicity preserving and monotonicity ensuring mathematical methods in this paper. Interpolation was the main topic of Chapter 2; having introduced desirable interpolation properties, we studied linear methods including cubic splines with the stress on their bases, a non-linear method using multiple base functions, and interpolation by cubic Bézier curves.

Chapter 3 then discussed the RRF (and other range finders) data properties in more detail, reviewed previous work on monotone approximation briefly, and suggested a hybrid method combining discontinuity detection, scale-space theory and locally adaptive smoothing to remove noise and ensure the desired piecewise monotonicity of the data.

Although section 3.3 describes a functional solution, we consider it a sketch rather than a complete and final answer to the problem of piecewise monotone approximation, noise reduction and processing of uncertainty in experimental data. It should correspond to the purpose of this thesis proposal, hopefully a useful piece of text in itself, but also a prelude to the doctoral thesis. Instead of concluding the work by an affirmative statement, we supply a collection of open questions by mentioning several of many possible (and several of many impossible) directions for further research.

- Improvement of the piecewise monotone approximation from section 3.3
  - detailed analysis of the method
  - autonomous deduction of optimal parameters for the edge-detection
  - parametrization: discrete scale, choice of its levels, continuity conditions on the scale parameter; parameter for required smoothness of the approximation
  - estimation of the necessary level of smoothing from some locally computed statistics instead of the iterative procedure
  - detection of outliers
- Different approaches to piecewise monotone approximation
  - adaptation of nonlinear diffusion (section 3.2.3) and its application to monotone approximation; comparison with the current solution

- Other methods to handle imprecision and uncertainty in the data
  - combination of multiple views for increased precision of the measurements

Bibliography

[1] Lars-Erik Andersson and Tommy Elfving. Interpolation and approximation by monotone cubic splines. *Journal of Approximation Theory*, 66:302–333, 1991.

[2] E. Baensch and K. Mikula. A coarsening finite element strategy in image selective smoothing. *Computing and Visualisation in Science*, pages 53–61, 1997.

[3] R. K. Beatson. Best monotone approximation by reciprocals of polynomials. *Journal of Approximation Theory*, 36:99–103, 1982.

[4] Dalit Caspi, Nahum Kiryati, and Joseph Shamir. Range imaging with adaptive color structured light. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):470–480, May 1998.

[5] Richard B. Darst and Salem Sahab. Approximation of continuous and quasi-continuous functions by monotone functions. *Journal of Approximation Theory*, 38:9–27, 1983.

[6] Carl de Boor. *A Practical Guide to Splines*. Springer-Verlag, 1978.

[7] Carl de Boor and Blair Schwartz. Piecewise monotone interpolation. *Journal of Approximation Theory*, 21:411–416, 1977.

[8] Rachid Deriche and Olivier Faugeras. Les edp en traitement des images et vision par ordinateur. Technical Report 2697, INRIA, Novembre 1995.

[9] Charles B. Dunham. Chebyshev approximation with non-negative derivative. *Journal of Approximation Theory*, 31:158–160, 1981.

[10] Andrew S. Glassner (ed.). *Graphics gems*. Academic Press, 1990.

[11] Gershon Elber. Interpolation using Bézier curves. In *Graphics Gems III*. Academic Press, 1992.

[12] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, 2-nd edition, 1990.

[13] Bernd Fisher and Gerhard Opfer. A local algorithm for constructing non-negative cubic splines. *Journal of Approximation Theory*, 64:1–16, 1991.

[14] Bob Fisher, Simon Perkins, et al. Hypermedia image processing reference. WWW page: `http://q127-3.coventry.ac.uk/~hipr/hipr_top.html`, Department of Artificial Intelligence, University of Edinburgh, 1994.

[15] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 1990.

[16] Wayne T. Ford and John A. Roulier. On interpolation by polynomials with monotone derivatives. *Journal of Approximation Theory*, 10:123–130, 1974.

[17] Curtis F. Gerald and Patrick O. Wheatley. *Applied Numerical Analysis*, chapter Interpolation and Curve Fitting, pages 210–305. Addison Wesley, 1994.

[18] Yingkang Hu, Dany Leviatan, and Xiang Ming Yu. Copositive polynomial and spline approxiamtion. *Journal of Approximation Theory*, 80:204–218, 1995.

[19] Robert Huotari. Best l_1-approximation of quasi continuous functions on [0, 1] by nondecreasing functions. *Journal of Approximation Theory*, 44:221–229, 1985.

[20] Canny J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), 1986.

[21] Jozef Kačur and Karol Mikula. Solution of nonlinear diffusion appearing in image smoothing and edge detection. *Applied Mathematics*, 17:47–59, 1995.

[22] Jozef Kačur and Karol Mikula. Slowed anisotropic diffusion. In *Scale-Space '97, Proceedings of First Int. Conf., Utrecht*, Lecture Notes in Comp. Sci. Springer, 1997.

[23] D. Leviatan and H. N. Mhaskar. Comonotone approximation by splines of piecewise monotone functions. *Journal of Approximation Theory*, 35:364–369, 1982.

[24] Tony Lindeberg. *Scale-Space Theory in Computer Vision.* Kluwer Academic Publishers, 1994.

[25] R. A. Lorentz. Uniqueness of best approximation by monotone polynomials. *Journal of Approximation Theory*, 4:401–418, 1971.

[26] Karol Mikula, Alessandro Sarti, and Claudio Lamberti. Geometrical diffusion in 3D-echocardiography. In *Proceeding of ALGORITMY'97, Conference of Scientific Computing*, pages 167–181, sept. 1997.

[27] Luboš Motl and Miloš Zahradník. *Pěstujeme lineární algebru.* Vydavatelství Karolinum, Univerzita Karlova, 1995.

[28] D. J. Newman. Efficient co-monotone approximation. *Journal of Approximation Theory*, 25:189–192, 1979.

[29] Tomáš Pajdla. Laser plane range finder - the implementation at the CVL. Technical Report Nr. K335-95-98, Czech Technical University, Prague, October 1995.

[30] Eli Passow. Piecewise montone spline interpolation. *Journal of Approximation Theory*, 12:240–241, 1974.

[31] Eli Passow. An improved estimate of the degree of monotone interpolation. *Journal of Approximation Theory*, 17:115–118, 1976.

[32] Eli Passow, Louis Raymon, and John A. Roulier. Comonotone polynomial approximation. *Journal of Approximation Theory*, 11:221–224, 1974.

[33] Less Piegl and Wayne Tiller. *The NURBS Book*. Springer Verlag, 2-nd edition, 1997.

[34] William H. Press, Saul A. Teulovsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipies in C. The Art of Scientific Computing*. Cambridge University Press, 2-nd edition, 1992.

[35] Richard Rasala. Explicit cubic spline interpolation formulas. In *Graphics Gems* [10].

[36] Bart M. Romeny, editor. *Geometry-Driven Diffusion in Computer Vision*. Kluwer Academic Publishers, 1994.

[37] John A. Roulier. Polynomials of best approximations which are monotone. *Journal of Approximation Theory*, 9:212–217, 1973.

[38] Vladimír Smutný. Analysis of rainbow range finder errors. In V. Hlaváč and T. Pajdla, editors, *1st Czech Pattern Recognition Workshop*, pages 59–66, Temešvár u Písku, November4–6 1993. Czech Pattern Recognition Society.

[39] Vladimír Smutný and Tomáš Pajdla. Rainbow Range Finder and its implementation at the CVL. Technical Report Nr. K335-96-130, Czech Technical University, Prague, May 1996.

[40] Milan Šonka, Václav Hlaváč, and Roger Boyle. *Image Processing, Analysis and Machine Vision*. Chapman and Hall Computing, 1993.

[41] J. J. Swetits and S. E. Weinstein. Construction of the best monotone approximation on $l_p[0,1]$. *Journal of Approximation Theory*, 61:118–130, 1990.

[42] Johji Tajima and Masato Iwakawa. 3-D data acquisition by rainbow range finder. In *Proc. 10th Int. Conf. on Pattern Recognition*, volume 1, pages 309–313, June 1990.

[43] Ken Turkowski. Filters for common resampling tasks. In *Graphics Gems* [10].

[44] Michael Unser, Akram Aldroubi, and Murray Eden. B-spline signal processing: Part I—theory. *IEEE Transactions on Signal Processing*, 41(2):821–833, Feb 1993.

[45] Michael Unser, Akram Aldroubi, and Murray Eden. B-spline signal processing: Part II—efficient design and applications. *IEEE Transactions on Signal Processing*, 41(2):834–848, Feb 1993.

[46] J. Weickert, B.M. ter Haar Romeny, and M.A. Viergever. Efficient and reliable schemes for nonlinear diffusion filtering. *IEEE Trans. Image Processing*, 7:398–410, 1998.

[47] Joachim Weickert. A review of nonlinear diffusion filtering. In B. ter Haar Romeny, L. Florack, J. Koenderink, and M. Viergever, editors, *Scale-Space Theory in Computer Vision*, volume 1252 of *Lecture Notes in Comp. Science*, pages 3–28. Springer, 1997. Invited paper.