Regression Strategies for Low-Dimensional Problems with Application to Color Management

Eric K. Garcia

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2010

Program Authorized to Offer Degree: Department of Electrical Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Eric K. Garcia

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

_____

Maya R. Gupta

Reading Committee:

_____

Maya R. Gupta

_____

Linda Shapiro

_____

Vladimir Minin

Date: _____

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Proquest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 1-800-521-0600, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Regression Strategies for Low-Dimensional Problems
with Application to Color Management

Eric K. Garcia

Chair of the Supervisory Committee:
Professor Maya R. Gupta
Department of Electrical Engineering

Nonparametric regression is the task of estimating a relationship between predictor variables and response variables from a set of training examples while making no a priori assumptions about its functional form. It is useful in applications where a model is either unknown, transient, or too difficult to characterize, and it has proven useful in a wide variety of applications including earth sciences, meteorology, computer vision, and digital color management. This dissertation introduces concepts and algorithms for use in nonparametric regression, and while much of the inspiration and validation of the proposed techniques stem from estimating color transformations – involving three to four predictor variables – they are applicable to more general regression problems as well. We present two new concepts in nonparametric regression that – due to computational considerations – are applicable only in low-dimensional problems (1–6 predictor variables). First, we introduce *enclosing neighborhoods*: a definition of locality for local linear regression that provides estimates with bounded variance; we propose the *enclosing kNN* neighborhood as the smallest (and thus lowest bias) such neighborhood along with an algorithm for its construction. Second, we present a technique, *lattice regression*, for estimating look-up tables (suitable for applications where fast test throughput is required) where the estimation minimizes the training error of the *overall* estimated function. The proposed methods are tested in the color management of printers as well as a variety of other low-dimensional applications.

TABLE OF CONTENTS

Page

Chapter 1

INTRODUCTION

Nonparametric regression is the task of estimating a relationship between predictor variables and response variables from a set of training examples while making no a priori assumptions about the functional form of this relationship. This is in contrast to *parametric* regression in which one imposes a model of the underlying function (i.e. linear, polynomial, gaussian) and attempts to fit the parameters of this model. Nonparametric regression is useful in applications where such a model is either unknown, transient, or too difficult to characterize, and has proven useful in a wide variety of applications including earth sciences, meteorology, computer graphics, computer vision, robotic control, image processing, and digital color management. This dissertation introduces concepts and algorithms for use in nonparametric regression, and while much of the inspiration and validation of the proposed techniques stem from work on the empirical characterization of color transformations for digital color management – typically involving three to four predictor variables – they are applicable to more general regression problems as well. However, the computational tractability of the proposed techniques limit their application to low-dimensional domains consisting of roughly six or fewer predictor variables. Further, the proposed techniques align into two categories, the first focuses on local (i.e. nearest-neighbor) regression and is suited to a general class of problems while the second is primarily suitable for applications where the desired function representation must be evaluated efficiently at test time (i.e. implemented as a look-up-table).

To analyze regression algorithms, it is instructive to decompose the source of error into *bias* and *variance*. When the relationship between predictors and responses is estimated from a randomly drawn training set (a set of discrete pairs of predictors and responses), *bias quantifies the accuracy of the average estimate* over random draws of the training set

while *variance quantifies the variability of the estimate* over random draws of the training set.

Consider a space of real-valued predictor variables $\mathcal{X} \subseteq \mathbb{R}^d$ and that of real-valued response variables $\mathcal{Y} \subseteq \mathbb{R}^p$. Suppose that there is an unknown relationship between variables $f : \mathcal{X} \to \mathcal{Y}$ that is corrupted by white Gaussian noise such that $Y = f(X) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ for the random variable $X \in \mathcal{X}$. We'd like to estimate f using a random training set $\mathcal{D} = \{X_i, Y_i\}_{i=1}^n$ of n examples where $X_i \in \mathcal{X}$ are drawn independently and identically distributed (iid) and $Y_i = f(X_i) + \epsilon_i$. The *generalization error* – the error on a test point $x \in \mathcal{X}$ drawn from the same distribution as the training set – of the estimated function $\hat{f}$ is

$$\begin{aligned}
\text{err}(\hat{f}(x)) &= E_{\mathcal{D},\epsilon}\big[f(x) - \hat{f}(x)\big]^2 \\
&= E_{\mathcal{D},\epsilon}\Big[\big(\hat{f}(x) - E_{\mathcal{D},\epsilon}\big[\hat{f}(x)\big]\big)^2\Big] + \Big(f(x) - E_{\mathcal{D},\epsilon}\big[\hat{f}(x)\big]\Big)^2 \\
&= \text{var}\big(\hat{f}(x)\big) + \text{bias}^2\big(\hat{f}(x)\big).
\end{aligned}$$

We see that the generalization error is a sum of the variance of the estimated function (the expected squared-difference between the estimated response and the *expected* estimated response) and the squared-bias (the squared-difference between the true response and the expected estimated response).

Regression algorithms typically have a *complexity* parameter that trades bias for variance. A larger complexity allows the model to fit the training data more accurately – leading to a low bias at the cost of increased variance – while a smaller complexity adapts a rigid model to fit the data as best as possible – leading to low variance at the cost of high bias. To illustrate this phenomenon, consider the local linear regression algorithm (see section 2.1 for a full description). For a test sample $x \in \mathcal{X}$, k-local linear regression fits – via least-squares – a linear function to the k nearest training samples in $\mathcal{D}$ and estimates the response as this linear function evaluated at x. This process can be repeated ad infinitum for any point in the domain. In this model, the complexity is controlled by k. At one extreme, when k is large, local linear regression becomes traditional linear regression and incurs a large model

bias if the underlying function is not truly linear. At the other extreme, when k is small, the local behavior of the estimate depends on only a few points from the training set, and thus small changes to this training set will cause relatively large changes in the estimate.

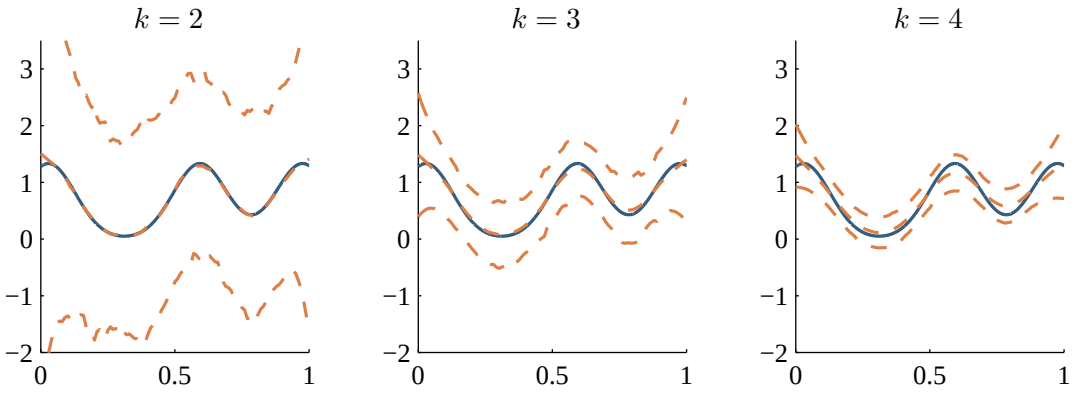


Figure 1.1: Local linear regression (LLR) applied to n=20 samples drawn with noise $\sigma^2 = .1$. Shown in dotted lines are the empirical mean and standard deviation of 10,000 LLR estimates computed on 10,000 randomly drawn training sets and noise for various neighborhood sizes: $k = 2$ (left), k=3 (middle), and k=4 (right).

Figure 1.1 illustrates this effect by plotting the sample mean and sample standard deviation of the local linear estimate for a range of neighborhood sizes. In this example, $\mathcal{X} = [0, 1]$, $\mathcal{Y} \subseteq \mathbb{R}$, $Y = f(x) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, .1)$. Each plot shows the true function f in solid blue as well as the sample mean and sample standard deviation in dotted orange (calculated from 10,000 uniform draws of 20 training samples and additive Gaussian noise) of the local linear regression estimate with the neighborhood size k. At $k = 2$, the mean estimate tracks the true function very closely (low bias) but the standard deviation of this estimate is quite large (high variance) while at $k = 4$, the mean estimate strays from the true function but the standard deviation is much smaller.

Figure 1.2 shows the generalization error of the estimates (averaged across 101 uniformly-spaced points in [0,1]). In this example, bias and variance conspire to produce a minimum generalization error when a neighborhood size of $k = 5$ is used. One can imagine that this value is highly application dependent and it may be useful to arrive at an optimal complexity in an automated way.

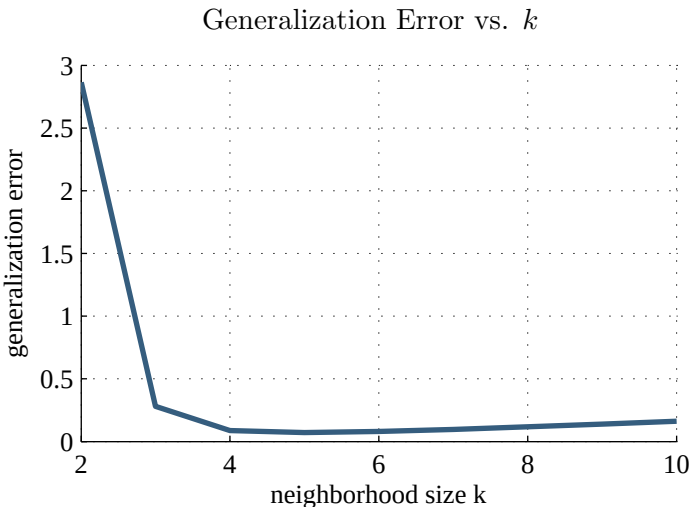


Figure 1.2: Generalization error of local linear regression from n=20 samples drawn with noise $\sigma^2 = .1$ for varying neighborhood sizes k. Computed from 101 uniformly spaced samples in [0,1].

1.1 Outline of Thesis

This dissertation presents two approaches for achieving low generalization error in nonparametric regression. A review of necessary concepts for the presentation of these techniques is provided in Chapter 2 and the chapter need *not* be read linearly, but rather used as a reference. Chapter 3 discusses an approach to limiting the variance of local linear regression by using neighborhoods that form a convex hull around the test sample. Theoretical results are discussed as well as the properties of the minimal such neighborhood – termed *enclosing k-NN* – which is meant to balance bias and variance in a straightforward manner. We then shift gears in Chapter 4 when we discuss *lattice regression* – a nonparametric regression technique that is aimed at minimizing the bias in estimating a function that is to be stored as look-up-table. Chapter 5 concludes the dissertation with a summary of findings and possible extensions of this work.

Chapter 2

RELATED WORK AND APPLICATIONS

We begin this chapter with a review of local linear regression and prior work on determining *locality* automatically (i.e. adaptively choosing a neighborhood). Then we go on to introduce applications that will serve as testing grounds for the algorithms presented in chapters 3 and 4. The reader is advised to read through section 2.1 and to continue to chapter 3, referring to the rest of this chapter as the individual applications are introduced in the *experiments* sections of chapters 3 and 4.

2.1 Local Linear Regression

Global linear regression – aka *linear regression* – is a classical statistical technique that is widely used in estimation [36, 37, 39, 49, 68]. The benefits of a linear model are its simplicity and ease of use, while its major drawback is high model bias: if the underlying function is not well approximated by an affine function, then linear regression produces poor results. *Local* linear regression exploits the fact that, over a small enough subset of the domain, any sufficiently nice function *can* be well-approximated by an affine function as is done in a first-order Taylor series expansion [61].

Suppose that for an unknown function $f : \mathbb{R}^d \to \mathbb{R}$, we are given a set of input samples $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and corresponding output samples $\mathcal{Y} = \{y_1, y_2, \ldots, y_n\}$ where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. The goal is to estimate the output $\hat{f}(g)$ for an arbitrary test point $g \in \mathbb{R}^d$. To form this estimate, local linear regression relies only on k "local" training samples indexed by the neighborhood $\mathcal{J}_g \subseteq \{1, \ldots, n\}$ where $|\mathcal{J}_g| = k$; this can be either the k nearest points in Euclidean distance or some adaptive notion of locality as is discussed in Section 2.1.1 and Section 3. Given this neighborhood, denote the relevant training data as $\mathcal{Y}_{\mathcal{J}_g} = \{y_j \mid j \in \mathcal{J}_g\}$ and $\mathcal{X}_{\mathcal{J}_g} = \{x_j \mid j \in \mathcal{J}_g\}$.

Local linear regression fits the least-squares hyperplane to the local neighborhood $\mathcal{J}_g$ of the test point, forming the estimate $\hat{f}(g) = \hat{\beta}^T \begin{bmatrix} g \\ 1 \end{bmatrix}$ where

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^{d+1}}{\arg\min} \sum_{j \in \mathcal{J}_g} \left(y_j - \beta^T \begin{bmatrix} x_j \\ 1 \end{bmatrix} \right)^2. \tag{2.1}$$

The size of the neighborhood $\mathcal{J}_g$ plays a significant role in the estimation result. A neighborhood that includes too many training points can result in a fit that is too smooth while a neighborhood with too few points can result in a fit with incorrectly steep extrapolation. One approach to reducing the estimation variance incurred by small neighborhoods is to regularize the regression, as is done in Tikhonov regression [39, 43]. Tikhonov regression also forms a hyperplane fit $\hat{f}(g) = \hat{\beta}_t^T \begin{bmatrix} g \\ 1 \end{bmatrix}$ but the coefficients $\hat{\beta}_t$ instead minimize a penalized least-squares criteria that regularize the local linear fit toward the global linear trend of the data. Let β_g be the least-squares hyperplane coefficients for the entire dataset:

$$\beta_g = \underset{\beta \in \mathbb{R}^{d+1}}{\arg\min} \sum_{i=1}^{n} \left(y_i - \beta^T \begin{bmatrix} x_i \\ 1 \end{bmatrix} \right)^2, \tag{2.2}$$

then

$$\hat{\beta}_t = \underset{\beta \in \mathbb{R}^{d+1}}{\arg\min} \sum_{j \in \mathcal{J}_g} \left(y_j - \beta^T \begin{bmatrix} x_j \\ 1 \end{bmatrix} \right)^2 + \lambda \, (\beta - \beta_g)^T \mathbf{I_0} (\beta - \beta_g), \tag{2.3}$$

where $\mathbf{I_0} = \mathbf{diag}\left([\mathbf{1}^T, 0] \right)$ and $\mathbf{1}$ is the $d \times 1$ vector of ones. The parameter λ controls the trade-off between minimizing the error and penalizing the deviation of the coefficients from the global trend; larger λ results in lower estimation variance, but higher estimation bias. Setting $\lambda = 0$ corresponds to traditional local linear regression. The closed-form solution to (2.3) is given by

$$\hat{\beta}_t = (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})^{-1} (\mathbf{X}y + \lambda \beta_g). \tag{2.4}$$

An alternative form of regularization is *ridge regression* [39, 41] which penalizes the *slope of the local hyperplanes* rather than their deviation from a global linear trend. This amounts to setting $\beta_g = \mathbf{0}$ in (2.3) and (2.4), discouraging fits with large values for $\|\beta\|_2^2$

and thus discouraging fits with steep slopes. Explicitly,

$$\hat{\beta}_r = \underset{\beta \in \mathbb{R}^{d+1}}{\arg\min} \sum_{j \in \mathcal{J}_g} \left(y_j - \beta^T \begin{bmatrix} x_j \\ 1 \end{bmatrix} \right)^2 + \lambda \, \beta^T \mathbf{I_0} \beta, \tag{2.5}$$

and the closed-form solution to (2.5) is given by

$$\hat{\beta}_r = (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{X} y. \tag{2.6}$$

2.1.1 Neighborhood Definitions for Local Learning

A fundamental question regarding local linear regression is: *Which neighbors should one choose for a given test sample?* In fact, this is an important question for all local learning methods, such as k-NN classification, local support vector machines [74], or local similarity-based learning [10].

The most common approach for local learning is to use some k nearest-neighbors, where the neighborhood size k is cross-validated and *nearest* is defined by either Euclidean distance or by a distance that is learned globally or locally [16, 18, 25, 26, 27, 38, 46, 53, 66, 71].

Using a fixed neighborhood size k, the nearest-neighbors algorithm can produce consistent local classifiers as the number of training samples $n \to \infty$ [17]. However, for finite data sets, we argue that it is not intuitive nor theoretically-motivated that the best estimation will arise from using a fixed neighborhood size *everywhere* in the domain. We hypothesize that the ideal neighborhood size may change as the density of samples changes throughout the domain, and that an ideal neighborhood would adapt to the spatial distribution of the training samples surrounding estimation point of interest.

This concept has been investigated by many researchers, though mostly in the context of k-NN classification. One approach is to adapt a neighborhood so that it surrounds the test point. This is the approach of Nock et al. [55] in defining the "symmetric nearest neighbors" which to be the i) nearest neighbor to the test sample and ii) all training samples for which the test sample is the nearest neighbor. A similar proposal was put forth by Gowda and Krishna in 1979, that they called *mutual nearest neighbor* [31]. And yet another related method is the k-SN neighbors in which a fixed number of neighbors k are selected in pairs:

first the nearest neighbor x_i not yet in the neighborhood is selected, then the next nearest neighbor x_j is selected that satisfies $d(x_i, x_j) > d(g, x_j)$ where g is the test point and d is a distance metric [75]. In order to choose neighbors that surround the test point, Chaudhuri proposed that a test point's neighbors should be near, but that also the mean of the neighbors should be close to the test point [12]. He proposed a greedy algorithm to meet this objective which requires specification of the number of neighbors k. Chaudhuri's neighbors have been shown by others to work consistently better than k-NN [62, 63]. Our Corollary 2 provides theoretical motivation for Chaudhuri's neighborhood definition.

Other approaches have been based on proximity-graphs from computational geometry. In 1981, Sibson proposed the *natural neighbors* for linear interpolation [67, 56]. The natural neighbors of g are defined to be those training points $\{x_i\}$ whose Voronoi cells [4] are adjacent to the cell containing g. An example is shown in the left diagram of Fig. 3.1. We discuss the natural neighbors further in Section 3.2.1.

The following sections in this chapter introduce applications that will be tested in the experiments of sections 3.4 and 4.3. When reading this document linearly, it is advised to skip now to chapter 3.

2.2 Color Management and Device Characterization

Digital color management allows for a consistent representation of color information among diverse digital imaging devices such as cameras, displays, and printers; it is a necessary part of many professional imaging workflows and popular among semi-professionals as well. An important component of any color management system is the characterization of the mapping between the native color space of a device (RGB for many digital displays and consumer printers), and a device-independent space such as CIE L*a*b* — abbreviated herein as Lab — in which distance approximates perceptual notions of color dissimilarity [64].

Device characterization can involve estimation of either the *forward mapping* from a device's native color space to a device-independent color space (i.e. RGB → Lab) or the *inverse mapping* from a device-independent color space to the device's native color space to (i.e. Lab → RGB). For display devices such as printers, one is generally interested in the *inverse device characterization*. With this, one can predict, for a given perceptual (Lab)

color one wishes to render, which device (RGB) input will produce this color. There are two common approaches to obtain an inverse device characterization: The first is to learn the parameters of a physical model of the device, such as the Neugebauer model of printers [5] and then to invert this model [51, 59]. The second is to fit a nonparametric function to empirical data (i.e. printed and measured color patches). Although the former approach commonly requires less data due to built-in (and hopefully appropriate) model bias, the latter is more common as it is applicable to a diverse set of devices and is less sensitive to ongoing improvements in the underlying technology.

Printer color management serves as the primary test bed for the strategies presented in this paper. Due to the complicated nature of the "display" process in printing (discussed above), printers exhibit among the most nonlinear and unpredictable color mappings among digital imaging devices, making them suitable for the nonparametric techniques that are the focus of this thesis. For printers, the empirical approach to inverse device characterization begins with printing a page of color patches for a set of input RGB values that span the gamut (range of displayable colors) of the device. In this thesis, we use the Gretag MacBeth TC9.18 RGB chart shown in Fig. 2.2. These patches are then measured with a spectrophotometer under standard illumination conditions, producing a single 3-dimensional Lab value for each patch. From these training pairs of (Lab, RGB) colors, one estimates the inverse mapping $f:$ Lab $\rightarrow$ RGB that specifies what RGB inputs to send to the printer in order to reproduce a desired Lab color. Estimating f is challenging for a number of reasons: 1) f is often highly nonlinear; 2) although it can be expected to be smooth over regions of the colorspace, it is affected by changes in the underlying printing mechanisms (for example, undercolor removal) that can introduce discontinuities [5]; and 3) device instabilities and measurement error introduce noise into the training data.

Fig. 2.1 shows a typical color-managed system. The *Learned Device Characterization* is most often implemented by a three-dimensional look-up table (3D LUT) with nodes $\{g_i\} \subset$ Lab that are regularly spaced in each dimension. This is followed by an array of one-dimensional look-up tables (1D LUTs) used for *calibration* (more on this below). Once estimated, the entire set of LUTs can be stored in an ICC profile – a standardized color management format, developed by the International Color Consortium (ICC). Input Lab

colors that are not a node of the 3D LUT are interpolated and – although interpolation technique is not specified in the standard – the most commonly used method is *trilinear interpolation* [44], a three-dimensional version of the common bilinear interpolation. This interpolation technique is computationally fast, and optimal in that it weights the neighboring nodes of the lattice as evenly as possible while still solving the linear interpolation equations.[1]

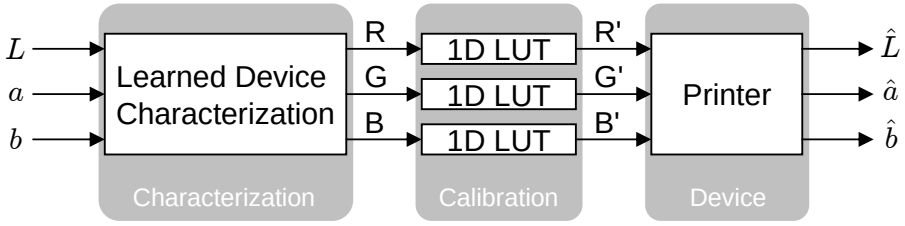


Figure 2.1: A color-managed printer system. For evaluation, errors are measured between (L, a, b) and $(\hat{L}, \hat{a}, \hat{b})$ for a given device characterization.

The following subsections introduce the typical process by which a device is color managed. This is the process that is used in the color management experiments in section 3.4.3 and section 4.3.3. As shown in Fig. 2.1, the inverse device characterization is split into two stages: *characterization* and *calibration*. The purpose of this dichotomy is twofold: First to reduce the amount of work in maintaining a color managed system; second to reduce the amount of data (number of printed patches) needed to build an accurate inverse device characterization. The details of calibration and characterization are presented in the following sections.

2.2.1 Gray-Balanced Calibration

In contrast to characterization, device calibration uses a smaller set of data and is performed more often in maintaining the color accuracy of a device over time. An added benefit conveyed by the most typical form of calibration, *gray-balanced calibration*, is that it transforms the data used in the device characterization in a way that linearizes the exponential nature

[1] It does so by choosing the maximum entropy solution to the linear interpolation equations [36, Theorem 2, p. 776]

(the *gamma* curve) of the color transform along the neutral axis. This amounts to shifting the training data to be more evenly distributed in the RGB color space when performing the second step of characterization. This redistribution of data effectively allows one to estimate a good characterization with fewer total samples. In this section we describe the details of gray-balanced device calibration as applied in section 3.4.3 and section 4.3.3.

As shown in Fig. 2.1, calibration is performed by three one-dimensional look-up tables (1D LUTs), often referred to as tone reproduction curves [65, Chapter 5.2.1]. The job of these 1D LUTs is to linearize each RGB channel independently, enforcing that input neutral RGB color values (R=G=B) will print neutral gray patches (as measured in Lab). That is, if one inputs the RGB color R= x, G= x, B= x for $x \in \{0, \ldots, 255\}$, the 1D LUTs will output the R'G'B' values that, when printed, correspond approximately to uniformly-spaced neutral gray steps in Lab space. The calibration step is commonly done with a set of specifically chosen RGB inputs that aim to characterize the neutral axis of the printer (these "gray ramps" can be seen on the upper-right of the TC9.18 chart in Fig. 2.2). Since fewer patches are needed to characterize the neutral axis than the entire color space (in the TC9.18 chart, there are seven ramps of length 18, for a total of 126 patches), calibration can be done with fewer measurements – and therefore with less expense – than full characterization.

The following outline illustrates the process of gray-balanced calibration as done in the experiments of section 3.4.3 and section 4.3.3. Again, the goal of the calibration is to find a set of 256 RGB values that print to a set of 256 evenly spaced neutral Lab values ($L \in \{L_{\min}, 1\Delta_L, 2\Delta_L, \ldots, L_{\max}\}\,, a = 0\,, b = 0$) where $L_{\min}$ and $L_{\max}$ are the minimum and maximum luminance outputs of the device, respectively and $\Delta_L = (L_{\max} - L_{\min})/255$. The RGB values corresponding to this equally spaced luminance ramp are used to construct the 1D LUTs which will map RGB $\rightarrow$ RGB.

The TC9.18 chart has eighteen clusters of neutral RGB values. Each cluster consists of seven RGB values that are intended to print Lab values near (preferably surrounding) the neutral axis ($a = 0, b = 0$). Of course, the actual Lab values printed are beyond the control of the user. The 1D LUTs (RGB $\rightarrow$ RGB) will be estimated from *anchor points* whose Lab values are estimated from these clusters (one anchor per cluster) as well as an estimated white point and black point of the device.

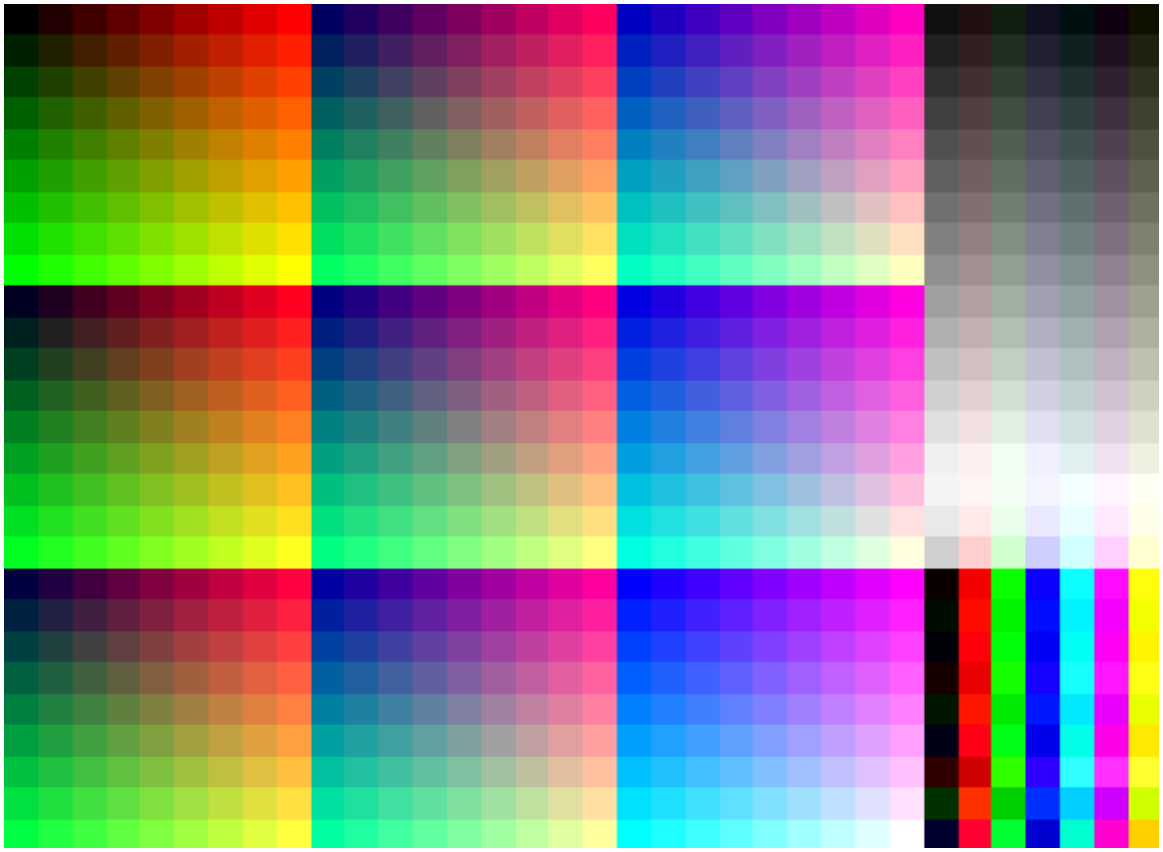


Figure 2.2: The Gretag MacBeth TC9.18 RGB image consists of 918 color patches that span the RGB colorspace. Of the patches, 729 are laid out in a grid covering the entire RGB cube (left), 126 form a set of seven neutral ramps (upper-right), and 63 form a set of saturated primaries (lower-right).

1) Print the Test Chart: Print the TC9.18 test chart and denote the jth RGB ($j = 1, 2, \ldots, 7$) value in the ith ($i = 1, 2, \ldots, 18$) neutral cluster by $\{R_{ij}, G_{ij}, B_{ij}\}$. Denote the corresponding printed and measured Lab values by $\{L_{ij}, a_{ij}, b_{ij}\}$.

2) Estimate the Neutral Anchors: For each of the $i = 1, 2, \ldots, 18$ clusters, set $\bar{L}_i = (1/7)\sum_{j=1}^{7} L_{ij}$ and set $\bar{a}_i = \bar{b}_i = 0$. Use a regression algorithm to estimate – from the *entire* set of measured TC9.18 patches *including the non-neutral clusters* – the $\{\bar{R}_i, \bar{G}_i, \bar{B}_i\}$ that correspond to each $\{\bar{L}_i, \bar{a}_i, \bar{b}_i\}$. This forms the set of neutral anchors from which the 1D LUTs will be estimated. Note that the choice of regression in this step will be varied in the experiments.

3) Estimate the White Point and Black Point: Set the black point luminance $\bar{L}_0$ to be the minimum L value in the *entire* measured TC9.18 target, set $\bar{a}_0 = \bar{b}_0 = 0$, and set $\bar{R}_0 = \bar{G}_0 = \bar{B}_0 = 0$. Set the white point luminance $\bar{L}_{19}$ to be the maximum L value in the *entire* measured TC9.18 target, set $\bar{a}_{19} = \bar{b}_{19} = 0$, and set $\bar{R}_{19} = \bar{G}_{19} = \bar{B}_{19} = 0$.

4) Linearly interpolate the 1D LUTs: Construct a set of 256 evenly spaced Lab values

$$\left\{x_0 = (\bar{L}_0, 0, 0),\ x_1 = (\Delta_L, 0, 0),\ x_2 = (2\Delta_L, 0, 0),\ \ldots,\ x_{255} = (\bar{L}_{19}, 0, 0)\right\}$$

where $\Delta_L = (\bar{L}_{19} - \bar{L}_0)/255$. Denote the corresponding outputs for each color channel $C \in R, G, B$ as $\{y_i^C\}_{i=0:255}$. For each color channel $C \in R, G, B$, use linear interpolation from the anchor pairs $\{\bar{L}_i, \bar{C}_i\}_{i=0:19}$ to estimate $\{y_i^C\}_{i=0:255}$.
The pairs $\{i, y_i^C\}_{i=0:255}$ form the three 1D calibration LUTs. As intended, these LUTs map the RGB values $\{(i, i, i)\}_{i=0:255}$ to the RGB values that produce a uniform luminance ramp at the output of the device.

5) Clip the LUTs: If any of the values $\{y_i^C\}_{i=0:255}$ are outside the range $[0, 255]$, clip them to this range.

Once constructed, the device calibration must be inverted in order to provide training data for the characterization stage. This is because the characterization and calibration will be applied serially to data sent to the printer (see Fig. 2.1). Inversion can be done by simply applying the calibration LUTs with inverted inputs (i.e. use the values $\{y_i^C, i\}_{i=0:255}$ instead of $\{i, y_i^C\}_{i=0:255}$). We will use this technique in the following section in order to prepare the TC9.18 data for estimating the 3D characterization LUT.

2.2.2 Characterization

In contrast to calibration, the *characterization* stage of inverse device characterization is performed less often and with more training data (i.e. more printed color patches). Whereas the calibration teases out non-linearities in the neutral behavior of the device, characterization is intended to estimate everything else that is necessary to ensure accurate color

reproduction. It is in this stage where the non-parametric techniques developed in this thesis will find a testing ground.

Characterization begins with a printed and measured test chart; in our case we use the Gretag MacBeth TC9.18 RGB chart shown in Fig. 2.2 which consists of 918 RGB color patches. Denote the RGB values of the patches as $\left\{(\tilde{y}_i^R, \tilde{y}_i^G, \tilde{y}_i^B)\right\}_{i=1:918} \subset \text{RGB}$ and the measured Lab values of these patches as $\{x_i\}_{i=1:918} \subset \text{Lab}$. Before estimating the characterization of the device, one must account for the effect of calibration on the RGB values. This is done, as described in section 2.2.1, by swapping the input/output values of the 1D LUTs and applying these inverted LUTs to the RGB values $\left\{(\tilde{y}_i^R, \tilde{y}_i^G, \tilde{y}_i^B)\right\}_{i=1:918}$, producing $\left\{(y_i^R, y_i^G, y_i^B)\right\}_{i=1:918}$. It is from the data $\left\{x_i, (y_i^R, y_i^G, y_i^B)\right\}$ that we build the characterization.

As mentioned in section 2.2, the characterization is typically implemented and stored as a three-dimensional look-up table (3D LUT). That is, a lattice of regularly-spaced nodes in the CIE L*a*b* domain $\text{Lab} = [0, 100] \times [-100, 100] \times [-100, 100]$. Such a lattice typically has $\tilde{m} \in \{9, 17, 33\}$ lattice nodes per dimension[2] for a total of $m = \tilde{m}^3$ nodes each with a corresponding RGB output. Thus, the task of estimation boils down to determining the m RGB outputs of the lattice that best represent the training data.

2.3 Geospatial Interpolation

Geospatial interpolation refers to the task of interpolating a quantity (such as elevation, average temperature, quantity of mineral deposits, annual rainfall, etc.) from a discrete set of measurements taken at known locations. On a small enough scale (i.e. not accounting for the curvature of the earth), this can be posed as a two-dimensional non-parametric regression task. In sections 4.3.2 and 3.4.2 we test the proposed techniques on the Spatial Interpolation Comparison 97 (SIC97) dataset [20] from the Journal of Geographic Information and Decision Analysis. This dataset is composed of 467 rainfall measurements made at

[2]Corresponding to $2^3, 2^4, 2^5$ lattice *cells* per each dimension, respectively. This arrangement eases the computational burden of trilinear interpolation as the interpolation weights for incoming Lab values can be calculated via binary operations such as bit-shifting when the data is encoded as fixed-point.

distinct locations across Switzerland. Of these, 100 randomly chosen sites were designated as training to predict the rainfall at the remaining 367 sites.

2.4 Omni-directional Image Super-resolution

This section describes the application of super-resolving (forming high-resolution images from multiple low-resolution images) omni-directional images. In section 4.3.4 we apply the proposed lattice regression as part of the registration procedure for super-resolution. Here, notation, application details, and related work on this topic are discussed to provide context for the experiments in section 4.3.4.

Modern advances in imaging technology and camera systems are driving many applications in computer vision and related areas. Omni-directional cameras provide a 360-degree view with catadioptric systems which capture the 3D scene on a convex mirror (parabolic or hyperbolic) that can be mapped onto a regular spherical grid [73]. An omni-directional scene may also be captured on the sphere using a pair of fisheye lenses [47]. These vision systems find applications in robotics, video surveillance, medical imaging and automatic face recognition. However, low-cost omnidirectional cameras may not provide sufficient image resolution for computer vision applications due to the wide field of view [54]. One solution is to super-resolve a higher-resolution image from multiple low-resolution images, but obtaining accurate registration information for the low-resolution images is costly.

Until recently, most of the efforts on image super-resolution with planar images focused on the scenario where perfect registration is known [23, 57]. Early efforts in super-resolution for omni-directional images also focused on reconstruction with full registration [45, 54]. In the last decade, there have been several important contributions that have advanced the research in super-resolution to the more realistic scenario where image-registration is not known a priori. A nonlinear least-squares approach was presented in [40] for simultaneous registration and reconstruction of planar images. In [69], a subspace-projection method was developed for bandlimited images to first learn image-registration and then perform reconstruction.

However, extension of existing super-resolution methods for planar images to omni-directional images is not straightforward. Omni-directional images are captured on a sphere

and require a signal-processing framework that respects that geometric structure. A novel method based on processing images in the spherical Fourier transform (SFT) domain was proposed in [2]. The authors extend the previous work in [40] to formulate a nonlinear least-squares optimization problem which is solved via the Levenberg-Marquardt algorithm. The work in [2] was further extended in [1, 3] by including an l_1-regularization term, thereby improving average picture-to-signal-ratio (PSNR) for the reconstructed image by around 1 dB. The spherical harmonics method [3] is based on previous work [2] which was awarded the best paper award at ICPR 2008 and is used as a benchmark for the experiments in section 4.3.4.

2.4.1 Problem Formulation

Let $\mathbf{S}^2$ denote the unit sphere in $\mathbb{R}^3$, i.e. $\mathbf{S}^2 = \{z \in \mathbb{R}^3 : ||z||_2 = 1\}$. In spherical coordinates, each point on the unit sphere can be represented by a unique pair $(\theta, \phi) \in \Omega_{\theta\phi}$ where $\Omega_{\theta\phi} = [0, \pi] \times [-\pi, \pi)$ and where θ is the co-latitude angle and ϕ is the longitude angle.

We parameterize the 3D rotation group[3] with Euler angles $(\alpha, \beta, \gamma) \in \Omega_g$ where $\Omega_g = [-\pi, \pi) \times [0, \pi] \times [-\pi, \pi)$ which represent angle of rotation (in radians) about the Z,Y and Z axis, serially. Let $g = (\alpha, \beta, \gamma)$; the corresponding rotation matrix can be written as

$$\mathbf{R}(g) = \mathbf{R}_{\mathrm{Z}}(\alpha)\ \mathbf{R}_{\mathrm{Y}}(\beta)\ \mathbf{R}_{\mathrm{Z}}(\gamma). \tag{2.7}$$

Given N low-resolution spherical images of size $M \times M$, we would like to reconstruct a single high-resolution image of size $L \times L$. To achieve this, we consider that there exists a square-integrable real-valued function $Z : \mathbf{S}^2 \to \mathbb{R}$ on the unit sphere. A spherical image is defined to be the quantized and discretized version of the underlying function Z sampled on a spherical grid. Consider the following regular spherical grid with M points in each coordinate direction:

$$\mathbf{G}_0 = \left\{ (\theta_j, \phi_k) = \Big((\pi/M)(j + 1/2),\ (2\pi/M)(k - M/2) \Big) : j, k = 0 : M - 1 \right\}. \tag{2.8}$$

[3] A rotation in $\mathbb{R}^3$ about the origin can be described by a matrix of dimensions 3×3 such that $\mathbf{R}^T\mathbf{R} = \mathbf{I}$, $\mathbf{R}\mathbf{R}^T = \mathbf{I}$ and $\det(\mathbf{R}) = 1$.

Note that $\mathbf{G}_0 \in \Omega_{\theta\phi}^{M\times M}$; it can be thought of as a matrix with (j,k)th entry given by the pair (θ_j, ϕ_k) in (2.8).

We consider that each of the N low-resolution images has undergone an unknown rotation and that the true registration of each image corresponds to a rotated version of $\mathbf{G}_0$. Let $\mathbf{G}^{g_i}$ denote the spherical grid obtained by rotating the reference grid by arbitrary rotation g_i,

$$\mathbf{G}^{g_i} = \left\{(\theta', \phi') \in \mathbf{S}^2 : \tau(\theta', \phi') = \mathbf{R}(g_i)\, \tau(\theta, \phi) \text{ for } (\theta, \phi) \in \mathbf{G}_0\right\}, \tag{2.9}$$

where $\tau : \mathbf{S}^2 \to \mathbb{R}^3$ is the conversion from spherical to Cartesian coordinates that maps $\Omega_{\theta\phi} \to \mathbb{R}^3$.

Let $\mathbf{Z}_i = Z(\mathbf{G}^{\tilde{g}_i})$ denote the *ground-truth* spherical function $Z : \Omega_{\theta\phi} \to \mathbb{R}$ sampled on grid $\mathbf{G}^{\tilde{g}_i}$ that has been rotated by some unknown *ground-truth* rotation $\tilde{g}_i$. Note that $\mathbf{Z}_i$ can be represented by an $M \times M$ matrix of pixel values. Given a subset $\mathcal{I} \subseteq \{1 : N\}$ of the low-resolution images, let $\mathbf{Z}_\mathcal{I} = \{\mathbf{Z}_i\}_{i\in\mathcal{I}}$ and let $g_\mathcal{I} = \{g_i\}_{i\in\mathcal{I}}$ be a corresponding set of rotations.

For an estimated spherical function $\hat{Z} : \Omega_{\theta\phi} \to \mathbb{R}$, let $\hat{\mathbf{Z}}_{g_i} = \hat{Z}\big(\mathbf{G}^{g_i}\big)$ denote the reconstruction of the i^{th} image under rotation g_i for $i = 1, \ldots, N$. The error in the estimate of the i^{th} low-resolution image is then given by the Frobenius norm

$$||\mathbf{Z}_i - \hat{\mathbf{Z}}_{g_i}||_F = \text{tr}\left[(\mathbf{Z}_i - \hat{\mathbf{Z}}_{g_i})^T(\mathbf{Z}_i - \hat{\mathbf{Z}}_{g_i})\right], \tag{2.10}$$

which is the total squared pixel-wise error between the two images. This is the same cost function as used in [3, 40].

The objective in joint-registration and super-resolution of omni-directional images is to estimate the function Z on a $L \times L$ higher resolution spherical grid from multiple $M \times M$ low-resolution images without any prior information about the corresponding rotations of images. We are given N low-resolution omni-directional images $\mathbf{Z}_i, i = 1, \ldots, N$, each of size $M \times M$ on grid-locations $\mathbf{G}^{\tilde{g}_i}$ that are unknown.

2.4.2 Super-Resolution using Spherical Harmonics

Here we describe the state-of-the-art method for super-resolution with omni-directional images presented by Arican and Frossard [1, 2, 3] which we use as benchmark. The estimate of the true spherical function is given by the spherical harmonic series,

$$\hat{Z}(\theta,\phi) = \sum_{l=0}^{B} \sum_{m=-l}^{l} \hat{a}_{l,m}\ Y_{l,m}(\theta,\phi), \tag{2.11}$$

where $Y_{l,m}$ are associated Legendre functions [13, 19], B is the assumed bandwidth of the spherical function Z [2], and $\hat{a}_{l,m}$ are spherical Fourier coefficients estimated from the unregistered low-resolution images as described below.

Let $\mathbf{Y}_0(\theta,\phi)$ denote the $B^2 \times 1$ vector of spherical harmonics (of all degrees and order up to B) evaluated at a point (θ,ϕ) on the unit sphere. Let $\mathbf{Y}_0 = [\mathbf{Y}_0(\theta_1,\phi_1) \cdots \mathbf{Y}_0(\theta_{M^2},\phi_{M^2})]$ denote the $B^2 \times M^2$ matrix of spherical harmonics evaluated at all points on the reference canonical grid $\mathbf{G}_0$. Similarly, let $\mathbf{Y}_k$ denote the matrix of spherical harmonics evaluated on the rotated grid $\mathbf{G}^{g_k}$. A closer examination of the matrices $\mathbf{Y}_0$ and $\mathbf{Y}_k$ reveals the following linear relationship (see [13]),

$$\mathbf{Y}_k^{\mathbf{T}} = \mathbf{U}_k(\mathbf{R}(g_k))\ \mathbf{Y}_0^{\mathbf{T}}, \tag{2.12}$$

where $\mathbf{U}_k(\mathbf{R}(g_k))$ is a $B^2 \times B^2$ matrix which depends only on the rotation matrix connecting the two grids. Consequently, the pixel values of given low-resolution images can be described by a system of linear equations,

$$z = \mathbf{Y}\ \mathbf{U}\ a, \tag{2.13}$$

where z is the vector of all pixel values from all low-resolution images, a are spherical Fourier coefficients, matrix $\mathbf{U}$ is built from matrices $\mathbf{U}_k(\mathbf{R}(g_k))$ and depends on rotations of spherical images and matrix $\mathbf{Y}$ is a block-diagonal matrix comprising $\mathbf{Y}_0$. Please refer to [3] for more details.

The system of linear equations $z = \mathbf{Y}\ \mathbf{U}\ a$ needs to be solved for spherical Fourier coefficients a as well as unknown rotations that determine the matrix $\mathbf{U}$. This is accomplished

via an iterative conjugate gradient approach [3]. The vector a can be used in (2.11) to compute the estimate of the true spherical function on a higher resolution grid.

Chapter 3

ENCLOSING NEIGHBORHOODS

Linear regression and local linear regression are standard methods for estimating a function given sample input/output pairs (see section 2.1). Here, we consider the question: *Under what conditions are linear regression and local linear regression guaranteed to work well?* Clearly, if the true underlying function f is linear or locally linear then these methods are well-suited. For analysis, we consider the slightly more general model that the underlying function f is locally linear but we are given measurements of the output values corrupted by iid noise. The local linear approximation is equivalent to a first-order Taylor series expansion [61] and although in practice we do not expect the error of this approximation (applied to an arbitrary continuous function) to be zero, it will be related to the size of the neighborhood to which the linear approximation is fit. It is posited that, given sufficient training samples, the error incurred by this first-order approximation can be arbitrarily reduced by reducing the size of the neighborhood used in estimation.

Given a test point g and a neighborhood $\mathcal{J}_g$, assume that the true function f is such that for $x \in g \cup \mathcal{X}_{\mathcal{J}_g}$, we have $f(x) = \beta^T \begin{bmatrix} x \\ 1 \end{bmatrix}$ for $\beta \in \mathbb{R}^{d+1}$. Recall that $\mathcal{J}_g \subseteq \{1, \ldots, n\}$ and $\mathcal{X}_{\mathcal{J}_g} = \{x_j \mid j \in \mathcal{J}_g\}$ as discussed in section 2.1. We will consider local linear estimates of f that are of the form $\hat{f}(g) = \hat{\beta}^T \begin{bmatrix} g \\ 1 \end{bmatrix}$ where $\hat{\beta}$ is determined by local linear regression (2.1). In evaluating the quality of the estimate $\hat{f}$, the overall squared-error can be decomposed into estimation variance and squared estimation bias [39] (as discussed in section 1),

$$\begin{aligned} E\big[f(x) - \hat{f}(x)\big]^2 &= E\Big[\big(\hat{f}(x) - E\big[\hat{f}(x)\big]\big)^2\Big] + \Big(f(x) - E\big[\hat{f}(x)\big]\Big)^2 \\ &= \quad \operatorname{var}\big(\hat{f}(x)\big) \quad + \quad \operatorname{bias}^2\big(\hat{f}(x)\big), \end{aligned}$$

and it is useful to analyze each independently. Under this noisy-linear model, the bias of the linear regression is determined by the average of the regression coefficients with respect to the noise. If the noise is zero-mean, then the expected least-squares regression coefficients

are equal to the true hyperplane coefficients, and therefore there is no bias. However, in general, the overall estimation error can still be unbounded due to estimation variance. We show that for the noisy-linear model, the estimation variance is in fact bounded by the noise power for those test samples that lie in the closure of the convex hull of the training samples and a corollary establishes which test sample faces the minimum estimation variance.

3.1 Bounding the Estimation Variance

This section provides the main result motivating our approach to constructing an adaptive neighborhood for local linear regression: For local linear regression – in which the estimated output for a test point g is constructed from the neighborhood $\mathcal{J}_g$ – the *estimation variance* is bounded if g lies within the convex hull of the neighborhood $\mathcal{J}_g$.

For the following analysis, we will assume that the underlying true function f is locally linear over the neighborhood of the test sample g. That is, there exists some $\beta \in \mathbb{R}^{d+1}$ such that $f(x) = \beta^T \begin{bmatrix} x \\ 1 \end{bmatrix}$ for $x \in g \cup \mathcal{X}_{\mathcal{J}_g}$. Under these assumptions, we establish that the estimation variance is bounded if $g \in \mathbf{conv}(\mathcal{X}_{\mathcal{J}_g})$, where $\mathbf{conv}(\mathcal{X}_{\mathcal{J}_g})$ denotes the closure of the convex hull of $\mathcal{X}_{\mathcal{J}_g}$:

Theorem 1. *Let f be a locally linear function such that there exists $\beta \in \mathbb{R}^{d+1}$ for which $f(x) = \beta^T \begin{bmatrix} x \\ 1 \end{bmatrix}$ for all $x \in \left\{g, \mathcal{X}_{\mathcal{J}_g}\right\}$. Let the points $g \cup \mathcal{X}_{\mathcal{J}_g}$ be in general position and suppose that the training labels $\{y_i\}_{i=1:n}$ are corrupted by independent additive noise such that $y_i = f(x_i) + \omega_i$ where $\omega = [\omega_1, \omega_2, \ldots, \omega_m]^T$ is drawn from a distribution with finite mean and covariance $\mathbf{\Sigma}_\omega = \sigma^2 \mathbf{I}$. If $g \in \mathbf{conv}(\mathcal{X}_{\mathcal{J}_g})$ then, for the estimate $\hat{f}(g) = \hat{\beta}^T \begin{bmatrix} g \\ 1 \end{bmatrix}$ where $\hat{\beta}$ are given by linear least-squares regression* (2.1)*, the estimation variance is bounded by σ^2:*

$$\mathrm{var}(\hat{f}(g)) = E_\omega \left[\left(\hat{f}(g) - E_\omega[\hat{f}(g)] \right)^2 \right] \leq \sigma^2. \tag{3.1}$$

Proof of Theorem 1:

Let y be the $1 \times k$ vector with elements $\mathcal{Y}_{\mathcal{J}_g}$, let $\mathbf{X}$ be the $d \times k$ matrix with columns $\mathcal{X}_{\mathcal{J}_g}$

and let $\tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{X} \\ \mathbf{1}^T \end{bmatrix}$. The least-squares regression coefficients which solve (2.1) are,

$$\begin{aligned} \hat{\beta} &= (\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T)^{-1}\tilde{\mathbf{X}}y \\ &= (\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T)^{-1}\tilde{\mathbf{X}}(\tilde{\mathbf{X}}^T\beta + \omega) \\ &= \beta + (\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T)^{-1}\tilde{\mathbf{X}}\,\omega\,. \end{aligned}$$

If we define $\mathbf{A} = (\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T)^{-1}\tilde{\mathbf{X}}$, the covariance matrix of the regression coefficients can be expressed as

$$\begin{aligned} \mathrm{cov}(\hat{\beta}) &= E_\omega\big[(\hat{\beta} - E_\omega[\hat{\beta}])(\hat{\beta} - E_\omega[\hat{\beta}])^T\big] \\ &= E_\omega\big[(\mathbf{A}\omega - \mathbf{A}E_\omega[\omega])(\mathbf{A}\omega - \mathbf{A}E_\omega[\omega])^T\big] \\ &= \mathbf{A}E_\omega\big[(\omega - E_\omega[\omega])(\omega - E_\omega[\omega])^T\big]\mathbf{A}^T \\ &= (\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T)^{-1}\tilde{\mathbf{X}}\;\sigma^2\mathbf{I}\;\tilde{\mathbf{X}}^T(\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T)^{-1} \\ &= \sigma^2(\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T)^{-1}. \end{aligned}$$

Define $\tilde{g} = \begin{bmatrix} g \\ 1 \end{bmatrix}$. The estimation variance is

$$\begin{aligned} \mathrm{var}(\hat{f}(g)) &= E_\omega\Big[\big(\hat{\beta}^T\tilde{g} - E_\omega\big[\hat{\beta}^T\tilde{g}\big]\big)^2\Big] \\ &= E_\omega\Big[\tilde{g}^T\big(\hat{\beta} - E_\omega[\hat{\beta}]\big)\big(\hat{\beta} - E_\omega[\hat{\beta}]\big)^T\tilde{g}\Big] \\ &= \tilde{g}^T\,\mathrm{cov}(\hat{\beta})\tilde{g} \\ &= \sigma^2\tilde{g}^T(\mathbf{X}\mathbf{X}^T)^{-1}\tilde{g}. \end{aligned} \tag{3.2}$$

Thus, the proof is finished by showing that $\tilde{g}^T(\mathbf{X}\mathbf{X}^T)^{-1}\tilde{g} \leq 1$ if $\tilde{g} \in \mathbf{conv}(\mathcal{X}_{\mathcal{J}_g})$.

Define $\mathcal{W} = \{v \in \mathbb{R}^k \mid \mathbf{X}v = g\}$, $\widetilde{\mathcal{W}} = \{v \in \mathbb{R}^k \mid \tilde{\mathbf{X}}v = \tilde{g}\}$ and the set of convex weights $\mathcal{C} = \{v \in [0,1]^k \mid v^T\mathbf{1} = 1\}$. Note that $v \in \mathcal{W} \cap \mathcal{C} \Rightarrow v \in \widetilde{\mathcal{W}} \cap \mathcal{C}$. To see this, consider that if $v \in \mathcal{W} \cap \mathcal{C}$ it holds that $\mathbf{X}v = g$ and $\mathbf{1}^Tv = 1$ and therefore

$$\tilde{\mathbf{X}}v = \begin{bmatrix} \mathbf{X} \\ \mathbf{1}^T \end{bmatrix}v = \begin{bmatrix} \mathbf{X}v \\ \mathbf{1}^Tv \end{bmatrix} = \begin{bmatrix} g \\ 1 \end{bmatrix} = \tilde{g}.$$

Also, for any $v \in \mathcal{C}$, it holds that

$$0 \leq v^T v \leq v^T \mathbf{1} = 1.$$

Since $g \in \mathbf{conv}(\mathcal{X}_{\mathcal{J}_g})$, the set $\mathcal{C} \cap \mathcal{W}$ is non-empty and thus $\widetilde{\mathcal{W}} \cap \mathcal{C}$ is non-empty as well. Therefore, there exists some $\tilde{v} \in \widetilde{\mathcal{W}}$ such that $\tilde{v}^T \tilde{v} \leq 1$. However, of all elements of $\widetilde{\mathcal{W}}$, the least-squares solution

$$\hat{v} = \tilde{\mathbf{X}}^T \left(\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \right)^{-1} g \tag{3.3}$$

has the minimum norm [9]. Thus, it follows that

$$\begin{aligned} \hat{v}^T \hat{v} &= g^T \left(\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \right)^{-1} \tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \left(\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \right)^{-1} g \\ &= g^T \left(\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \right)^{-1} g \\ &\leq \tilde{v}^T \tilde{v} \\ &\leq 1, \end{aligned}$$

completing the proof. □

Corollary 2. *Assume the same conditions of Theorem 1. The estimation variance is minimized for the test point that is the mean of the training samples:*

$$g = \bar{x}_{\mathcal{J}_g} = \frac{1}{k} \sum_{j \in \mathcal{J}_g} x_j. \tag{3.4}$$

Proof of Corollary 2:

Let $\mathbf{X}$ be the $d \times k$ matrix with columns $\mathcal{X}_{\mathcal{J}_g}$. We can rewrite (3.2) as

$$\mathrm{cov}(\hat{\beta}) = \sigma^2 \begin{bmatrix} \mathbf{X}\mathbf{X}^T & \mathbf{X}\mathbf{1} \\ \mathbf{1}^T \mathbf{X}^T & \mathbf{1}^T \mathbf{1} \end{bmatrix}^{-1}.$$

By defining the matrix,

$$\begin{bmatrix} \mathbf{V}_{11} & \mathbf{V}_{12} \\ \mathbf{V}_{12}^T & \mathbf{V}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{X}\mathbf{X}^T & \mathbf{X}\mathbf{1} \\ \mathbf{1}^T\mathbf{X}^T & \mathbf{1}^T\mathbf{1} \end{bmatrix}^{-1},$$

the estimation variance can be expressed as

$$\begin{aligned} \mathrm{var}(\hat{f}(g)) &= \sigma^2 \begin{bmatrix} g \\ 1 \end{bmatrix}^T \begin{bmatrix} \mathbf{V}_{11} & \mathbf{V}_{12} \\ \mathbf{V}_{12}^T & \mathbf{V}_{22} \end{bmatrix} \begin{bmatrix} g \\ 1 \end{bmatrix} \\ &= \sigma^2(g^T\mathbf{V}_{11}g + 2g^T\mathbf{V}_{12} + \mathbf{V}_{22}). \end{aligned}$$

The point of minimum variance will be the g^* that satisfies

$$\begin{aligned} 0 &= \frac{\partial \hat{f}(g)}{\partial g} \\ &= \sigma^2(2\mathbf{V}_{11}g + 2\mathbf{V}_{12}), \end{aligned}$$

and therefore $g^* = -\mathbf{V}_{11}^{-1}\mathbf{V}_{12}$.

Applying the block matrix inverse [58]:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{12}^T & \mathbf{A}_{22} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{C}_1^{-1} & -\mathbf{C}_1^{-1}\mathbf{A}_{12}\mathbf{A}_{22}^{-1} \\ -\mathbf{A}_{22}^{-1}\mathbf{A}_{12}^T\mathbf{C}_1^{-1} & \mathbf{C}_2^{-1} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_{11} & \mathbf{V}_{12} \\ \mathbf{V}_{12}^T & \mathbf{V}_{22} \end{bmatrix}$$

where

$$\mathbf{C}_1 = \mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{12}^T \quad \text{and} \quad \mathbf{C}_2 = \mathbf{A}_{22} - \mathbf{A}_{12}^T\mathbf{A}_{11}^{-1}\mathbf{A}_{12},$$

we find that

$$-\mathbf{V}_{11}^{-1}\mathbf{V}_{12} = \mathbf{C}_1\mathbf{C}_1^{-1}\mathbf{A}_{12}\mathbf{A}_{22}^{-1} = \mathbf{A}_{12}\mathbf{A}_{22}^{-1}.$$

Thus,

$$g^* = \mathbf{X}\mathbf{1}(\mathbf{1}^T\mathbf{1})^{-1} = \frac{\mathbf{X}\mathbf{1}}{k} = \frac{1}{k}\sum_{j\in\mathcal{J}_g} x_j,$$

completing the proof. □

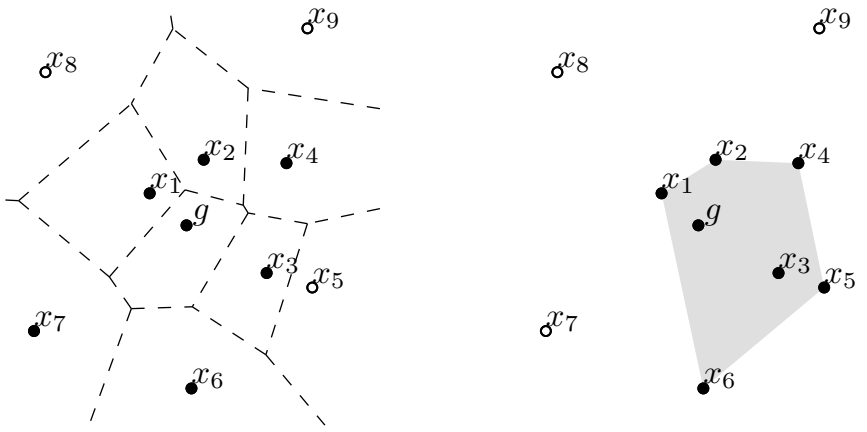
In practice, regularized estimates of the regression coefficients are often used. If regularization results in lower (or equal) estimation variance for all g than the unregularized coefficients, then the bounded variance result of Theorem 1 will hold. This variance reduction should occur with any regularization that penalizes the regression coefficients for deviating from the zero vector (as in lasso and ridge regression).

3.2 Enclosing Neighborhood Definitions

As a formal criteria for a bounded-variance neighborhood, we define an *enclosing neighborhood* to be a set of training samples $\mathcal{J}_g$ such that *the test sample g is contained in the convex hull of its neighborhood.* That is, $\mathcal{J}_g \subseteq \{1, \ldots, n\}$ is an enclosing neighborhood if $g \in \mathbf{conv}(\mathcal{X}_{\mathcal{J}_g})$.

This property formalizes previous researchers' intuitive goal of "a neighborhood that would surround the test point," as discussed in Section 2.1.1. For instance Sanchez et al. noted that although it was intuitive to them to choose a neighborhood that surrounds a test point, they knew of no theoretical justification for this [62]. Theorem 1 provides a first theoretical justification by showing that enclosing neighborhoods have bounded estimation variance under a noisy-linear model. In preliminary experiments, we also considered minimizing (rather than bounding) the estimation variance by forming a neighborhood $\mathcal{J}_g$ for which g is near the mean. We found however, that this often induces significant bias in the estimate as the diameter of $\mathcal{J}_g$ increases to accommodate this goal. Thus, we limit the scope of this work to bounded variance, aiming to achieve a better balance of the error contributions of bias and variance.

In the next few sections we consider two constructive neighborhood definitions that yield the enclosing neighborhood property for $g \in \mathbf{conv}(\mathcal{X})$. We note that such a construction is not possible in situations where the test sample is outside the convex hull of the entire training set, $g \notin \mathbf{conv}(\mathcal{X})$, and while the theoretical results presented do not hold for such g, the following neighborhoods are still well-defined when $g \notin \mathbf{conv}(\mathcal{X})$.



$\mathcal{J}_g^{\mathrm{NN}} = \{x_1, x_2, x_3, x_4, x_6, x_7\}$ $\mathcal{J}_g^{\mathrm{ekNN}} = \{x_1, \ldots, x_6\}$

(a) Natural Neighbors (b) Enclosing k-NN

Figure 3.1: In the left figure, the *natural neighbors* neighborhood $\mathcal{J}_g^{\mathrm{NN}}$ is marked with solid circles. For reference, the Voronoi diagram of this set is dashed. In the right figure, the *enclosing k-NN* neighborhood $\mathcal{J}_g^{\mathrm{ekNN}}$ is marked with solid circles.

3.2.1 Natural Neighbors

Natural neighbors form enclosing neighborhoods when possible [56, 67]. The natural neighbors are defined by the Voronoi tessellation $\mathcal{V}$ [4] of the training set and the test point $g \cup \mathcal{X}$. Given $\mathcal{V}$, the natural neighbors of g are defined to be those training points x_j whose Voronoi cells are adjacent to the cell containing g. An example of the natural neighbors is shown in Fig. 3.1.

The *local coordinates property* of the natural neighbors can be used to prove that the natural neighbors form an enclosing neighborhood when $g \in \mathbf{conv}(\mathcal{X})$ [67]. Although the natural neighbors were designed for use with a specific generalized linear interpolation method called *natural neighbors interpolation* [67], Theorem 1 suggests that this neighborhood may be successful when applied to linear regression as well.

One issue with natural neighbors for general learning tasks is that the complexity of computing the Voronoi tessellation of n points in d dimensions is $O(n \log n)$ when $d < 3$ and $O((n/d)^{d/2})$ when $d \geq 3$ [7]. Another concern is whether the natural neighbors are

sufficiently "local" as they do not form a *radial* neighborhood. That is, the natural neighbors neighborhood can include training points that are quite far (in Euclidean distance) from the test point while excluding others that are closer. This leads to a growth in the diameter of the neighborhood and thus an increase in the bias incurred from the local linear approxiamtion.

3.2.2 Enclosing k-NN Neighborhood

The bias incurred from the linear approximation in local linear regression will grow with the diameter of the neighborhood $\mathcal{J}_g$ (i.e. as neighbors become far from the test point g). To reduce this risk, we propose – and this is the main contribution of this section – choosing the k nearest neighbors with the smallest k such that $g \in \mathbf{conv}(\{\mathcal{X}_g\}_{1:k})$, where $\{\mathcal{X}_g\}_{1:k} \subseteq \mathcal{X}$ denotes the k nearest neighbors of g in the training set $\mathcal{X}$. Note that this is ill-defined when $g \notin \mathbf{conv}(\mathcal{X})$ is not in the convex hull of the training set; a more complete definition requires a bit more care in its exposition. To this end, denote the distance from g to the convex hull of the k nearest neighbors in $\mathcal{X}$ as

$$D_{\mathcal{X}}(g,k) = \min_{x \in \mathbf{conv}(\{\mathcal{X}_g\}_{1:k})} \|g - x\|_2. \tag{3.5}$$

Note that $D_{\mathcal{X}}(g,k) = 0$ if $g \in \mathbf{conv}(\{\mathcal{X}_g\}_{1:k})$.

The *enclosing k-NN neighborhood* of g is defined as $\mathcal{J}_g^{\mathrm{eNN}} = \{\mathcal{X}_g\}_{1:k^*}$ where

$$k^* = \min_k \left\{ k \;\middle|\; D_{\mathcal{X}}(g,k) = D_{\mathcal{X}}(g,n) \right\}. \tag{3.6}$$

This can be interpreted as follows: For any $g \in \mathbf{conv}(\mathcal{X})$, k^* is the smallest $k \in \{1,2,\ldots,n\}$ such that $g \in \mathbf{conv}(\{\mathcal{X}_g\}_{1:k})$, while if $g \notin \mathbf{conv}(\mathcal{X})$ then k^* is the smallest $k \in \{1,2,\ldots,n\}$ such that g is *as close as possible* to the convex hull of $\{\mathcal{X}_g\}_{1:k}$. An example of the enclosing k-NN neighborhood is shown in Fig. 3.1 and an algorithm for its computation is provided below.

Algorithm for Computing the Enclosing k-NN Neighborhood

We include a slack term ϵ for which a test point g within distance ϵ of the convex hull of $\mathcal{X}_{\mathcal{J}_g}$ is considered inside the hull. Setting $\epsilon = 0$ corresponds to the strict definition of the enclosing k-NN neighborhood. We found that, in practice, using non-zero values of ϵ led to increased performance by allowing smaller neighborhoods (i.e. less bias) at a marginal cost to variance. However, setting ϵ to an appropriate value is highly dependent on the application.

Let us begin the description of the algorithm by defining a few variables. Let $\mathcal{S} \subseteq \mathcal{X}$ be a temporary set of neighbors and denote its complement by $\mathcal{S}^c = \mathcal{X} \backslash \mathcal{S}$. We will assume that the complement set $\mathcal{S}^c$ is updated automatically without directly stating so in the algorithm.

1) Initialize: Let $x_{\min} = \arg\min_{x \in \mathcal{X}} \|g - x\|_2$ and set $\mathcal{S} = x_{\min}$.

2) Project onto the Neighbors: Let $g_{\mathcal{S}} = \arg\min_{x \in \mathbf{conv}(\mathcal{S})} \|g - x\|_2$ be the projection of g onto the convex hull of S. If $\|g_{\mathcal{S}} - g\|_2 < \epsilon$ go to Step 4.

3) Partition the Training Set: Let $d = \|g_{\mathcal{S}} - g\|_2^2$ and partition $\mathcal{S}^c$ into the following two sets:

$$\mathcal{S}_0^c = \left\{x \in \mathcal{S}^c \mid (g_{\mathcal{S}} - g)^T x \leq d\right\} \quad \text{and} \quad \mathcal{S}_1^c = \left\{x \in \mathcal{S}^c \mid (g_{\mathcal{S}} - g)^T x > d\right\}.$$

Note that adding points in $\mathcal{S}_1^c$ to $\mathcal{S}$ will *not* bring the convex hull of $\mathcal{S}$ any closer to g. If $\mathcal{S}_0^c = \emptyset$ go to Step 4.

3) Add a Neighbor and Iterate: Add to the set $\mathcal{S}$ the training point $x^* \in \mathcal{S}_0^c$ nearest to g. That is, let

$$x^* = \arg\min_{x \in \mathcal{S}_0^c} \|g - x\|_2,$$

and set $\mathcal{S} = \mathcal{S} \cup x^*$. Go to Step 2.

4) Add all Radial Neighbors: Let $x_{\max} = \arg\max_{x \in \mathcal{S}} \|g - x\|_2$ and set

$$\mathcal{J}_g^{\text{e}k\text{NN}} = \left\{x \in \mathcal{X} \mid \|g - x\|_2 \leq \|g - x_{\max}\|_2\right\}.$$

3.3 Sizes of Enclosing Neighborhoods

Enclosing k-NN and natural neighbors adapt the size of the neighborhood to the local spatial distribution of the training and test sample. Smooth nonlinear functions may be effectively modeled as locally linear, where the approximation will tend to be better the more local the neighborhood choice. Thus, in general the more local the neighborhood choice, the lower the model bias will be. As a proxy for analyzing the expected bias for each of the three enclosing neighborhood constructions, we will consider the expected size of the neighborhood.

3.3.1 Expected Neighborhood Size

Asymptotically, the expected number of natural neighbors is equal to the expected number of edges of a Delaunay triangulation [56]. A common stochastic spatial model for analyzing Delaunay triangulations is the Poisson point process, which assumes that points are drawn randomly and uniformly such that the average density λ is m points per volume S. Given this model, the expected number of natural neighbors is known for low dimensions: 6 neighbors for two dimensions, $48\pi^2/35 + 2 \approx 15.5$ neighbors for three dimensions, and $340/9 \approx 37.7$ neighbors for four dimensions [56].

The following theorem establishes that the expected number of neighbors in the enclosing k-NN neighborhood is $2d+1$ if the training samples are sampled from a uniform distribution within a hypersphere centered on the test sample, or in fact from any distribution that is invariant to rotations about the test sample.

Theorem 3. *Suppose n training samples are sampled independently and identically from a distribution that is symmetric around a test sample in $\mathbb{R}^d$. Then, in the limit that $n \to \infty$, the expected number of neighbors in the enclosing k-NN neighborhood is $2d + 1$.*

Proof of Theorem 3:

The proof requires the following lemma:

Lemma 4. *Let* $\mathbf{X} \in \mathbb{R}^{d \times n}$ *have ith column* x_i*, and denote the convex hull of the columns of* $\mathbf{X}$ *by* $\mathbf{conv}(\mathbf{X})$*, then scaling the data* $\mathbf{X}$ *will not affect the event* $\mathbf{0} \in \mathbf{conv}(\mathbf{X})$*. That is, if and only if the origin* $\mathbf{0} \in \mathbf{conv}(\mathbf{X})$*, then* $\mathbf{0} \in \mathbf{conv}(\mathbf{XA})$ *for any positive definite diagonal* $n \times n$ *matrix* $\mathbf{A}$*.*

Proof. Suppose $\mathbf{0} \in \mathbf{conv}(\mathbf{X})$. By definition, there exists a weight vector w such that $\mathbf{1}^T w = 1$, $w \succeq 0$, and $\mathbf{X}w = \mathbf{0}$. If $\mathbf{X}$ is scaled by the positive definite diagonal matrix $\mathbf{A}$, then it must be shown that there exist a set of weights w' with the properties that $\mathbf{1}^T w' = 1$, $w' \succeq 0$, and $\mathbf{XA}w' = \mathbf{0}$. Denote the normalization scalar $z = \mathbf{1}^T \mathbf{A}^{-1} w$, then it can be seen that one such weight vector that satisfies this condition is $w' = (\mathbf{A}^{-1} w)/z$ and we conclude that $\mathbf{0} \in \mathbf{conv}(\mathbf{XA})$. Next, suppose that $\mathbf{0} \notin \mathbf{conv}(\mathbf{X})$, then it must be shown that scaling $\mathbf{X}$ by any positive definite diagonal matrix $\mathbf{A}$ does not form a convex hull that contains the origin. The proof is by contradiction: assume that $\mathbf{0} \in \mathbf{conv}(\mathbf{XA})$ but $\mathbf{0} \notin \mathbf{conv}(\mathbf{X})$. The first part of this proof could be applied, scaling $\mathbf{XA}$ by $\mathbf{A}^{-1}$, which would lead to the conclusion that $\mathbf{0} \in \mathbf{conv}(\mathbf{X})$, thus forming a contradiction. □

Without loss of generality, assume the test point is the origin $g = \mathbf{0}$. Let $\mathbf{X}$ be the random $d \times n$ matrix with columns $\{x_i\}_{i=1:n}$ drawn independently and identically from a symmetric distribution over $\mathbb{R}^d$ centered a the origin. Rearrange the columns of $\mathbf{X}$ so that they are sorted such that $||x_{k-1}||_2 \leq ||x_k||_2 \leq ||x_{k+1}||_2$ for all k. As established in the lemma, without a loss of generality with respect to the event $\mathbf{0} \in \mathbf{conv}(\mathbf{X})$, scale all columns such that $||x_j||_2 = 1$ for all j. Then $\mathbf{0} \in \mathbf{conv}(\mathbf{X})$ if and only if the column vectors are *not* all contained in some hemisphere [42].

Let H_n indicate the event that n vectors lie on the same hemisphere, and let $\bar{H}_n$ denote the complement of H_n. Wendel [72] showed that for n points chosen uniformly on a the surface of a hypersphere in $\mathbb{R}^d$,

$$\mathrm{P}(H_n) = 2^{-n+1} \sum_{k=0}^{d-1} \binom{n-1}{k} \quad \forall\, n \geq 1. \tag{3.7}$$

Let F_n be the event that: the first n ordered points enclose the origin, but the first $n-1$ ordered points do not enclose the origin. The probability of the event F_n is

$$\begin{aligned} \mathrm{P}(F_n) &= \mathrm{P}(\bar{H}_n, H_{n-1}) \\ &= \mathrm{P}(\bar{H}_n | H_{n-1}) \mathrm{P}(H_{n-1}) \\ &= (1 - \mathrm{P}(H_n | H_{n-1}))\, \mathrm{P}(H_{n-1}) \\ &= \mathrm{P}(H_{n-1}) - \mathrm{P}(H_n, H_{n-1}) \\ &= \mathrm{P}(H_{n-1}) - \mathrm{P}(H_n). \end{aligned} \tag{3.8}$$

Because one or zero points cannot complete a convex hull around the origin, $\mathrm{P}(F_0) = 0$ and $\mathrm{P}(F_1) = 0$. Combining (3.7) and (3.8), and using the recurrence relation of the binomial coefficient

$$\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}, \tag{3.9}$$

$$\begin{aligned} \mathrm{P}(F_n) &= 2^{-n+2} \sum_{k=0}^{d-1} \binom{n-2}{k} - 2^{-n+1} \sum_{k=0}^{d-1} \binom{n-1}{k} \\ &= 2^{-n+1} \sum_{k=0}^{d-1} \left[2\binom{n-2}{k} - \binom{n-1}{k} \right] \\ &= 2^{-n+1} \sum_{k=0}^{d-1} \left[2\binom{n-2}{k} - \binom{n-2}{k-1} - \binom{n-2}{k} \right] \\ &= 2^{-n+1} \left[\binom{n-2}{d-1} - \binom{n-2}{d-2} + \binom{n-2}{d-2} \right. \\ &\qquad \left. - \binom{n-2}{d-3} + \ldots + \binom{n-2}{0} - \binom{n-2}{-1} \right] \\ &= 2^{-n+1} \left[\binom{n-2}{d-1} - \binom{n-2}{-1} \right] \\ &= 2^{-n+1} \binom{n-2}{d-1} \quad \text{for all } n \geq 2, \end{aligned}$$

where the last line follows because $\binom{r}{-1} = 0$ for all r [32, p. 154].

Then the expectation $\mathrm{E}[F_a]$ is given by

$$\mathrm{E}[F_a] = \sum_{a=2}^{\infty} a\mathrm{P}(F_a) = \sum_{a=2}^{\infty} a2^{-a+1}\binom{a-2}{d-1}.$$

To simplify, change variables to $b = a - 2$,

$$\begin{aligned}
\mathrm{E}[F_a] &= \sum_{b=0}^{\infty} 2^{-b-1}\left[(b+1)\binom{b}{d-1} + \binom{b}{d-1}\right] \\
&= \sum_{b=0}^{\infty} 2^{-b-1}\left[d\binom{b+1}{d} + \binom{b}{d-1}\right] \\
&= \sum_{b=0}^{\infty} 2^{-b-1}\left[d\binom{b+1}{d} + \binom{b+1}{d} - \binom{b}{d}\right] \qquad (3.10) \\
&= \sum_{b=0}^{\infty} 2^{-b-1}\left[(d+1)\binom{b+1}{d} - \binom{b}{d}\right]
\end{aligned}$$

where (3.10) is an application of (3.9).

Expanding the summation,

$$\begin{aligned}
\mathrm{E}[F_a] &= \lim_{n\to\infty}\left(2^{-1}(d+1)\binom{1}{d} - 2^{-1}\binom{0}{d} + 2^{-2}(d+1)\binom{2}{d} - 2^{-2}\binom{1}{d}\dots\right. \\
&\quad \left. + 2^{-n-2}(d+1)\binom{n}{d} - 2^{-n-2}\binom{n-1}{d} + 2^{-n-1}(d+1)\binom{n+1}{d} - 2^{-n-1}\binom{n}{d}\right) \\
&= \lim_{n\to\infty}\left(2^{-1}\binom{0}{d} + 2^{-2}\big(2(d+1)-1\big)\binom{1}{d}\dots\right. \\
&\quad \left. + 2^{-n-1}\big(2(d+1)-1\big)\binom{n}{d} + 2^{-n-1}(d+1)\binom{n+1}{d}\right) \\
&= \lim_{n\to\infty}\left(2^{-1}\binom{0}{d} + \sum_{i=1}^{n} 2^{-i-1}(2d+1)\binom{i}{d} + 2^{-n-1}(d+1)\binom{n+1}{d}\right).
\end{aligned}$$

The first term in the last line is zero from the identity $\binom{0}{d} = 0$ [32, p. 155]. The third term converges to zero as $n \to \infty$. The remaining term can be re-written,

$$\mathrm{E}[F_a] = (2d+1)(.5)\sum_{i=1}^{\infty}\binom{i}{d}(.5)^i.$$

Using the summation [32, p. 199],

$$\sum_{i=0}^{\infty}\binom{i}{d}z^i = \frac{z^d}{(1-z)^{d+1}}$$

with $z = .5$, establishes the result: $\mathrm{E}[F_a] = 2d+1$. □

3.3.2 Simulated Neighborhood Sizes

To gain intuition about the distribution of neighborhood sizes, we simulated drawing training samples uniformly, calculated the neighborhood sizes, and plotted them in Fig. 3.3. For each run of the simulation, three hundred points were drawn independently and uniformly over the interior of the unit hypersphere. The enclosing k-NN and natural neighbors were calculated for a test point at the origin. For each dimension the simulation was run twenty times, and each neighborhood size is marked on the plot of Fig. 3.3. The natural neighbors simulations were run only up to six dimensions due to memory restrictions (run with 2 GB of RAM) for the larger number of total training samples needed to estimate neighborhood sizes for higher dimensions.

One sees that the enclosing neighborhood sizes do increase linearly as roughly $2d+1$, and that the distribution of neighborhood sizes does not appear to broaden. The average natural neighbors' sizes also appear to match the known low-dimensional analytical expected mean sizes, but the distribution of sizes broadens as the dimension increases, and appears to increase roughly exponentially as the dimension increases.

3.4 Enclosing Neighborhood Experiments

In this section we compare the performance of local linear regression on an enclosing neighborhood to local linear regression on a fixed neighborhood size and to other regression

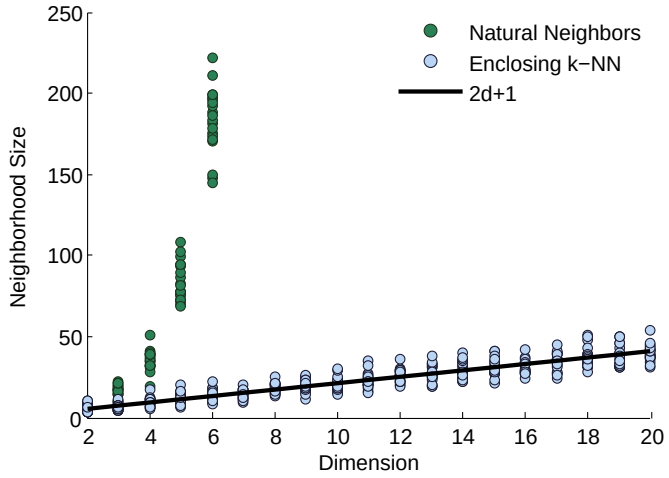


Figure 3.2: Shown are the neighborhood sizes for the natural neighbors and the enclosing k-NN neighborhoods over dimension.

techniques as well. Section 3.4.1 presents experiments on simulated data that are aimed at evaluating the theoretical claims presented in section 3.1. This is followed in section 3.4.2 by a similar set of experiments on a real geospatial interpolation problem (see section 2.3). Finally, in section 3.4.3, we apply local linear regression using enclosing neighborhoods to the task of estimating color transformations for the color management of digital printers. For a complete description of this application, see section 2.2.

3.4.1 Simulated Data

We consider approximating a two-dimensional sinusoidal function by local linear regression given uniformly drawn random samples with additive Gaussian noise. The simulation is designed to match the conditions of Theorem 1. Since Theorem 1 is only applicable for test samples that lie in the convex hull of the set of all training samples, the test points are regularly spaced over the unit square while the training samples are uniformly and randomly drawn over the square $[-.5, 1.5]^2$. For each of one hundred runs of the simulation, independent and identically drawn Gaussian noise with mean zero and variance .01 is drawn and added to the sinusoidal function values of the fixed training samples.

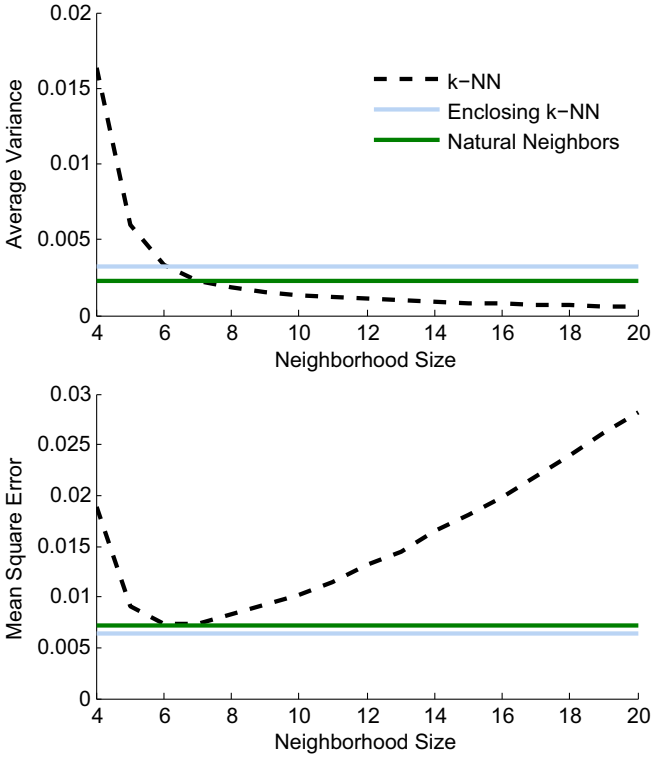


Figure 3.3: Shown are the average variance and mean squared error of local linear regression vs. neighborhood for different sizes of fixed k, enclosing k-NN, and natural neighbors.

Results are shown in Fig. 3.3. In the top plot of Fig. 3.3, note that the variance of the local linear estimate decreases as a function of k. In fact, as k increases the k-NN neighborhood becomes an enclosing neighborhood, and can achieve lower variance than that of our adaptive enclosing neighborhoods because it is larger. However, note that in the bottom plot of Figure 3.3, the mean square error for the k-NN estimate begins to increase after $k = 7$, as the bias incurred by using a such a large neighborhood begins to affect the error. We see that the adaptive size of the enclosing neighborhoods manages a nice balance between this estimation bias and variance, and achieves a lower average error than the k-NN estimate for any fixed choice of k.

3.4.2 Geospatial Interpolation

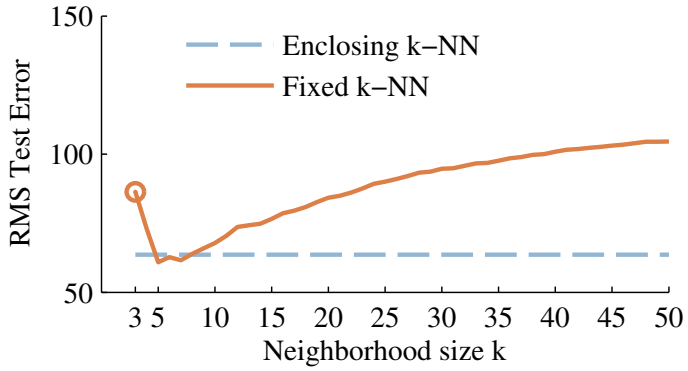
Geospatial interpolation refers to the regression of measurements that are tied to geographic coordinates such as elevation, rainfall, forest cover, wind speed, etc (see section 2.3). For a small enough area – that is, if the curvature of the earth can be ignored – this amounts to a two-dimensional regression problem; thus, suitable for enclosing k-NN method. As a representative example from this class of data, we applied the proposed technique on the Spatial Interpolation Comparison 97 (SIC97) dataset [20] from the Journal of Geographic Information and Decision Analysis. This dataset is composed of 467 rainfall measurements made at distinct locations across Switzerland. Of these, 100 randomly chosen sites were designated as training to predict the rainfall at the remaining 367 sites.

We test three local regression techniques: local Tikhonov regression, local ridge regression, and local linear regression (see section 2.1) each with two neighborhoods: enclosing k-NN and a fixed radial neighborhood of varying size k. The regularization parameter λ of Tikhonov and ridge regression was chosen from the set $\{10^{-6}, 10^{-5}, \ldots, 10^{1}\}$ via ten-fold cross-validation on the $n = 100$ training points. Shown in Fig. 3.4 are the RMS error of the predictions at the 367 test sites.

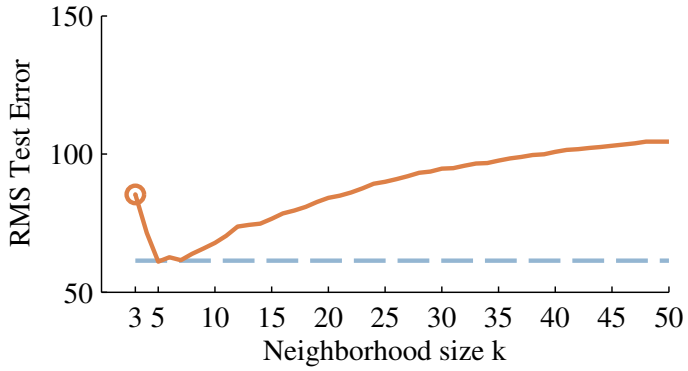
One can see that enclosing k-NN does a reasonable job of matching the *lowest* test error produced by a fixed neighborhood size. Importantly, for a fixed neighborhood size, computing the value k that produces the lowest test error is not possible in practice as one does not typically have access to the true measurements at the test locations. To see just how well enclosing k-NN matched these optimistic results for the fixed neighborhood size, we computed the statistical significance of the individual test errors via a one-sided Wilcoxon significance test (p=0.05). The results of this test showed no significant difference between enclosing k-NN and the *best possible* fixed k for any of the regression methods.

3.4.3 Color Management Experiments

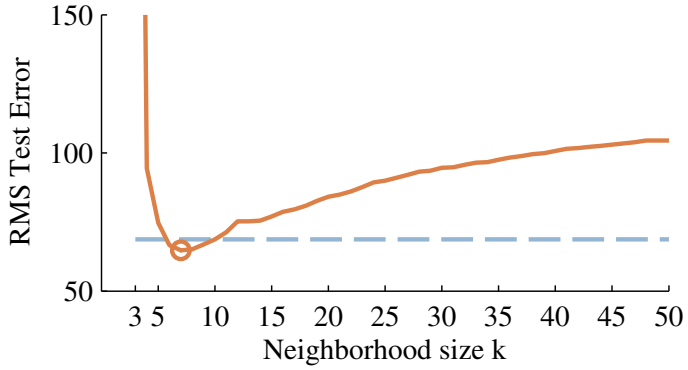
As discussed in section 2.2, color management refers to the task of controlling how colors are rendered across diverse image display and capture devices, such as scanners, monitors, and printers. The goal is to ensure that the displayed (or captured) colors in the image are



(a) Local Tikhonov Regression



(b) Local Ridge Regression



(c) Local Linear Regression

Figure 3.4: Shown is the RMS test error of enclosing k-NN and fixed k-NN for three local regression methods: (a) local Tikhonov regression,(b) local ridge regression, and (c) local linear regression. Enclosing k-NN is shown as a horizontal dashed line while the performance of fixed k-NN is shown for a sequence of neighborhood sizes k; the value of k chosen by ten-fold cross-validation for fixed k-NN is circled.

perceptually consistent across color representations (RGB, CMYK) and technologies (LCD, CRT, Inkjet, Laserjet, etc). The color management of printers is especially difficult because the color response is nonlinear, and the colors they produce depend on the printer hardware, the halftoning method, the ink or toner, paper type, humidity, and temperature [5, 24].

In estimating color transformations for printers from empirical data, local linear regression has been shown to work better than neural networks, polynomial regression, and splines [5], and better than color management techniques using models of ink-substrate interactions [65]. Thus, this application provides a useful and relevant testbed for local linear regression on enclosing neighborhoods.

The proposed technique of local linear regression on enclosing neighborhoods was tested for color accuracy on three printers: an Epson Stylus Photo 2200 (ink jet) with Epson Matte Heavyweight Paper and Epson inks, an Epson Stylus Photo R300 (ink jet) with Epson Matte Heavyweight Paper and third-party ink from Premium Imaging, and a Ricoh Aficio 1232C (laser engine) with generic laser copy paper. Color measurements of the printed patches were done with a GretagMacbeth Spectrolino spectrophotometer at a 2° observer angle with D50 illumination.

Both linear and ridge regression were tested in order to assess the compatibility of the variance reduction provided by the enclosing neighborhood with that provided by a more standard regularization technique. The ridge regression regularization parameter λ (see Eqn. (2.5)) was fixed at $\lambda = 0.1$ for all the experiments. This value was chosen based on a small set of preliminary experiments which suggested that values of λ from $\lambda = .001$ to $\lambda = 2$ would produce similar results.[1]

The accuracy of each method was tested on reproducing 918 new randomly-chosen in-gamut[2] test Lab colors. All errors are computed via the ΔE^*_{94} standard[3]. Tables 3.1,3.2 and 3.3 show the average ΔE^*_{94} error and 95th-percentile error for the three printers for

[1]It is common wisdom that a small amount of regularization can be very helpful in reducing estimation variance, but larger amounts of regularization can cause unwanted bias, resulting in oversmoothing [39].

[2]We drew 918 samples iid uniformly over the RGB cube and printed these, measuring the resulting Lab values; these Lab values were as test samples. This is a standard approach to assuring that the test samples are Lab colors that are in the achievable color gamut of the printer [35].

[3]The ΔE^*_{94} error metric is one standard way to measure color management error [5].

each neighborhood definition with local linear regression (2.1) and local ridge regression (2.5). As a baseline, we compared to a fixed neighborhood size of $k = 15$ neighbors with local linear regression which was found to work well in practice [34]. Small errors may not be noticeable; although actual human perception of color differences varies throughout the color space and between people, errors under $2\Delta E^*_{94}$ are generally not noticeable.

On the Ricoh laser printer, both local ridge and local linear regression on the enclosing neighborhoods produce lower average and lower 95th-percentile error than the corresponding baseline regression on $k = 15$ neighbors. The case is strengthened with the results from the Epson 2200, where results produced using the enclosing neighborhoods have lower average and 95th-percentile error than any of the baseline of $k = 15$ neighbors.

However, the outlier in this set of experiments is the Epson R300 inkjet printer. For un-regularized linear regression, there is increased error when using the enclosing neighborhoods. Combined with ridge regression, there is little difference in performance, save a lower 95th-percentile error with natural neighbors. Since the enclosing neighborhoods in three dimensions typically have fewer than 15 points (see Section 3.3), it is not surprising that ridge regression provides more of a benefit when used with these neighborhoods than on the baseline.

Overall, the results indicate that, when used in conjunction with regularized local linear regression, an enclosing neighborhood can provide device characterizations that are often more accurate but certainly no worse than than using a regularized local linear regression on a fixed neighborhood size. Thus showing local linear regression on enclosing neighborhoods to be a promising contender in estimating inverse device characterizations for color management.

3.5 Conclusions

In this section, we have proposed a strategy for determining locality in local linear regression. Namely, to choose a neighborhood that encloses the test sample within its convex hull. We showed that, under a true linear model with additive zero-mean iid noise, such a neighborhood will produce an estimate with bounded variance. In order to keep the bias low as well, we proposed the *enclosing k-NN* neighborhood, defined to be the smallest *ra-*

Table 3.1: Ricoh Aficio 1232C

Neighborhood	Regression	ΔE^*_{94} Error	
		Mean	95 %-ile
Enclosing k-NN	Linear	4.27	8.47
	Ridge	3.66	7.38
Natural Neighbors	Linear	3.74	7.55
	Ridge	3.69	7.10
15 Neighbors	Linear	4.41	9.84
	Ridge	4.16	8.61

Table 3.2: Epson Photo Stylus 2200

Neighborhood	Regression	ΔE^*_{94} Error	
		Mean	95 %-ile
Enclosing k-NN	Linear	2.32	5.01
	Ridge	2.20	5.03
Natural Neighbors	Linear	2.40	5.48
	Ridge	2.20	5.16
15 Neighbors	Linear	2.44	6.52
	Ridge	2.43	6.46

Table 3.3: Epson Photo Stylus R300

Neighborhood	Regression	ΔE^*_{94} Error	
		Mean	95 %-ile
Enclosing k-NN	Linear	1.67	3.65
	Ridge	1.55	3.32
Natural Neighbors	Linear	1.71	3.49
	Ridge	1.54	2.87
15 Neighbors	Linear	1.55	3.32
	Ridge	1.55	3.34

dial neighborhood that encloses the test sample within its convex hull. Experiments on simulated and real data demonstrated that enclosing neighborhoods achieve similar performance to the minimum possible error achieved by a fixed neighborhood size. That is, even if one were to choose a fixed neighborhood size for local linear regression that minimizes the *test error*, enclosing k-NN can match this performance. This is an impressive result as the neighborhood size is typically chosen less optimistically by minimizing the cross-validation error on the training set.

Chapter 4

LATTICE REGRESSION

For high-throughput regression applications, one is concerned with the computational time required to evaluate an estimated function as well as its accuracy. However, most regression techniques do not produce models that have an efficient implementation, particularly in hardware. For example, kernel-based methods such as Gaussian process regression [60] and support vector regression require kernel computations between each test sample and a subset of training examples, and local smoothing techniques such as weighted nearest neighbors [39] require a search for the nearest neighbors.

For functions with a known and bounded domain, a standard *efficient* approach to function approximation is to store a regular lattice of function values spanning the domain, then linearly interpolate[1] each test sample from the lattice nodes surrounding it. Evaluating the lattice is independent of the size of any original training set, but exponential in the dimension of the input space making it best-suited to low-dimensional applications. For instance, this approach is used ubiquitously in color management where real-time performance often requires millions of evaluations every second and it has been standardized by the International Color Consortium (ICC) with a file format called an ICC profile [70].

A lattice $\{a_i, b_i\}_{i=1:m}$ consists of m lattice nodes $a_i \in \mathbb{R}^d$ and m corresponding lattice outputs $b_i \in \mathbb{R}$ (see Fig. 4.1 for an example). The node locations are typically chosen a priori in a regular grid (though this need not be the case) and the final result of a learning procedure will be the estimation of the lattice outputs b_i. For applications where one begins with a set of training data $\{x_i, y_i\}_{i=1:n}$, the standard approach is to first estimate a function $\hat{f}$ that fits the training data, then evaluate $\hat{f}$ at the lattice points. That is, the estimated output value for a_i is $\hat{b}_i = \hat{f}(a_i)$. Mathematically, the estimated function $\hat{f}$ is chosen to

[1] Here, linear means that the interpolation is a linear combination of the outputs at lattice nodes; the result can be a nonlinear function of the input variables as is the case for tetrahedral or trilinear interpolation.

minimize the sum of some loss function $L : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$:

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} L(f(x_i) - y_i), \tag{4.1}$$

where $\mathcal{F}$ is a set of allowed functions. For example, standard least-squares linear regression is (4.1) with $\mathcal{F}$ being the set of all linear functions $f(x) = \beta^T x + \beta_0$ and $L(\cdot) = (\cdot)^2$. Neural nets, decision trees, and support vector machine regression and other standard approaches to regression can be written as (4.1) for different choices of $\mathcal{F}$ and L, and sometimes with additional regularization terms that are independent of the data but add a preference for smoother functions in the function class $\mathcal{F}$ (see subsection 4.1).

However, this approach is suboptimal because the effect of interpolation from the lattice is not considered when estimating the function in (4.1). For example, if one used the lattice to estimate the appropriate output for a training sample x_i, the output is not $\hat{f}(x_i)$, rather it is the interpolation of x_i from the subset of $\left\{\left(a_i, \hat{f}(a_i)\right)\right\}$ that surround x_i. Thus, the error being minimized in (4.1) is not the error that we wish to minimize; we would like to directly minimize the error of $\hat{f}$ on the training data. If one knows a priori what interpolation technique will be applied to the lattice, one could instead learn lattice outputs that – upon interpolation – minimize the error on the training data.

This is precisely what the proposed *lattice regression* [28, 29, 30] aims to do by estimating lattice outputs that minimize the regularized *interpolation* error on the training data. The key to this estimation is that the linear interpolation operation can be inverted to solve for the node outputs that minimize the squared error of the training data. However, unless there is ample training data, the solution will not necessarily be unique. Also, to lower estimation variance it may be beneficial to avoid fitting the training data exactly. For these reasons, we additionally maximize the *smoothness* of the estimated function via regularization; a collection of techniques for quantifying smoothness on a lattice are explored.

4.1 Lattice Regression Formulation

A widely used principle in machine learning is that of *regularized empirical risk minimization* (also known as structural risk minimization). For instance, support vector machines (and

support vector regression), Tikhonov regression, splines, and Gaussian process regression all apply this principle. Regularized empirical risk minimization is one approach to solving a basic paradox of machine learning: one should approximate a function $\hat{f}$ that accurately predicts the training data $\{x_i, y_i\}_{i=1:n}$ but, at the the same time, does not *overfit* (i.e. merely memorize) the training data. Instead, it should *generalize* to unseen data drawn from the same distribution. To achieve this balance, empirical risk algorithms optimize the following cost function

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} L(f(x_i), y_i) + \lambda J(f), \tag{4.2}$$

where $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$ is a loss function that penalizes differences between the estimated outputs $f(x_i)$ and the known outputs y_i and $J : \mathcal{F} \to \mathbb{R}$ is a regularization function that typically measures the 'smoothness' of the function f. Adjusting the trade-off parameter λ balances the empirical risk (accuracy on the training data) with the 'smoothness' of estimate. The usage of the term 'smoothness' is purposefully vague because it is highly context dependent. In the context of regression (learning a real valued function from discrete samples), smoothness in the Lipschitz sense is typically appropriate. In section 4.1.2 we will investigate a number of regularization functions and their effects on estimation.

4.1.1 Lattice Regression: The Empirical Risk Term

The lattice regression *empirical risk* term solves for lattice outputs that accurately interpolate the training data. For a bounded input space $D \subset \mathbb{R}^d$, let $\{a_j \in \mathbb{R}^d\}$ for $j = 1, \ldots, m$ be the nodes of a regular lattice in $\mathbb{R}^d$ such that D lies within the convex hull of the lattice $\{a_j\}$. Let the training data be a $n \times d$ matrix of inputs $\mathbf{X} = \begin{bmatrix} x_1, \ldots, x_n \end{bmatrix}^T$ where $x_j \in D$ and a $n \times 1$ matrix of outputs $y_= \begin{bmatrix} y_1, \ldots, y_n \end{bmatrix}^T$ where $y_j \in \mathbb{R}$. A training point $x_i \in \mathbb{R}^d$ falls in a cell of the lattice with 2^d vertices; the jth vertex in the lattice is given a linear interpolation weight $w_{ij} \geq 0$, where $w_{ij} = 0$ if a_j is not a vertex of the cell containing x_i, and otherwise w_{ij} is set so that linear interpolation equations hold: $\sum_j w_{ij} a_j = x_i$, and $\sum_j w_{ij} = 1$. The input value x_i is then interpolated as $\hat{y}_i = \sum_j w_{ij} b_j$.

To minimize the post-interpolation error on the training data, the empirical risk term chooses the $m \times 1$ vector of output values $\hat{b}$ that solve,

$$\begin{aligned}\hat{b} &= \arg\min_{b \in \mathbb{R}^m} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 . \\ &= \arg\min_{b \in \mathbb{R}^m} \sum_{i=1}^{n} \left(\left(\sum_j w_{ij} b_j \right) - y_i \right)^2 . \end{aligned} \tag{4.3}$$

To write (4.3) more compactly, let $\mathbf{W} \in [0,1]^{n \times m}$ denote the matrix with ith-jth element w_{ij}. Then given n training inputs $\{x_i\}$ and their corresponding linear interpolation weights $\mathbf{W}$, the lattice interpolates a vector of n output values $\hat{y} = \mathbf{W}b$. The empirical risk objective (4.3) can be succinctly expressed as

$$\begin{aligned}\hat{b} &= \arg\min_{b \in \mathbb{R}^m} \|\hat{y} - y\|_2^2. \\ &= \arg\min_{b \in \mathbb{R}^m} \|\mathbf{W}b - y\|_2^2. \end{aligned} \tag{4.4}$$

Conveniently, (4.4) has a closed-form solution:

$$\hat{b} = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T y, \tag{4.5}$$

and because $\mathbf{W}$ is sparse, the matrix inversion can be computed efficiently by sparse Cholesky factorization (for instance the `ldivide` command in Matlab).

Though conceptually straightforward, deriving the matrix $\mathbf{W}$ for a particular interpolation technique is is somewhat involved. Here we provide the details via a set of functions that annotate lattice operations. A guiding example that illustrates some of the necessary notation is shown in Fig. 4.1.

Without loss of generality, assume that the domain is scaled and translated such that the lattice sits at the origin with nodes at integer coordinates in $\mathbb{R}^d$. Further, let the d-dimensional vector $\tilde{m} = [\tilde{m}_1, \tilde{m}_2, \ldots, \tilde{m}_d]^T$ denote the number of nodes in each dimension where $m = \prod_{k=1}^{d} \tilde{m}_k$. In Fig. 4.1 $\tilde{m} = [3, 3]$.

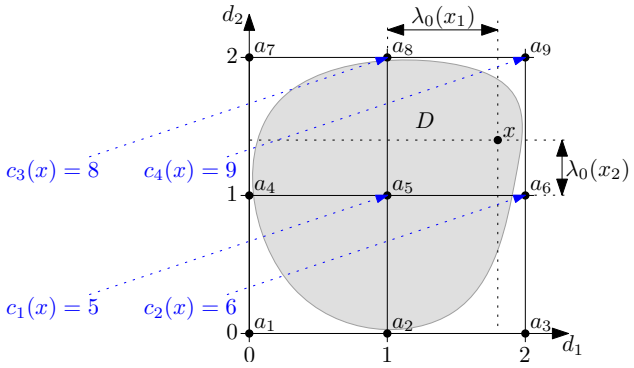


Figure 4.1: The 3×3 lattice with test point $x = [1.8, 1.4]^T$. The shaded area shows the function domain D, which is contained in the convex hull of the lattice.

Now, let us define the function $N_{\tilde{m}} : \mathbb{N} \to \mathbf{Z}^d$ that returns the d-dimensional coordinates of the jth node in a lattice a_j. The ℓth element of $N_{\tilde{m}}(j)$ is

$$N_{\tilde{m}}(j)_\ell = \left\lfloor \frac{j-1}{\prod_{k=1}^{\ell-1} \tilde{m}_k} \right\rfloor \bmod \tilde{m}_\ell \quad \text{for } \ell = 1, \ldots, d.$$

For example, on the lattice shown in Fig. 4.1,

$$N_{[3,3]}(2) = \left[\lfloor 1 \rfloor \bmod 3,\ \lfloor 1/3 \rfloor \bmod 3\right]^T = [1, 0]^T$$

and

$$N_{[3,3]}(6) = \left[\lfloor 5 \rfloor \bmod 3,\ \lfloor 5/3 \rfloor \bmod 3\right]^T = [2, 1]^T.$$

Checking Fig. 4.1, one sees that these are indeed the coordinates of a_2 and a_6, respectively. As shorthand, when all entries of the $\tilde{m}$ are identical (i.e. $\tilde{m} = \alpha\mathbf{1}$ where $\mathbf{1}$ is a d-dimensional vector of ones) let $N_\alpha(\cdot) = N_{\alpha\mathbf{1}}(\cdot)$ for $\alpha \in \mathbf{Z}^+$.

Next, interpolating a test point $x \in D$ requires the ability to index the nodes of the cell in which it is contained. To that end, define the function $c_j(x) : \mathbb{R}^d \to \mathbb{N}$ that returns the index of the jth vertex ($j = 1, \ldots, 2^d$) of the lattice cell that contains x. The function $c_j(x)$

can be computed as follows:

$$c_j(x) = 1 + \sum_{k=1}^{d} \big(\lfloor x_k \rfloor + N_2(j)_k\big)\big(\prod_{i=0}^{k-1} \tilde{m}_k\big) \tag{4.6}$$

where we define $\tilde{m}_0 = 1$ for notational convenience. For example, in Fig. 4.1, we have

$$\begin{aligned}
c_1(x) &= 1 + (\lfloor 1.8 \rfloor + 0)(1) + (\lfloor 1.4 \rfloor + 0)(3) &= 5 \\
c_2(x) &= 1 + (\lfloor 1.8 \rfloor + 1)(1) + (\lfloor 1.4 \rfloor + 0)(3) &= 6 \\
c_3(x) &= 1 + (\lfloor 1.8 \rfloor + 0)(1) + (\lfloor 1.4 \rfloor + 1)(3) &= 8 \\
c_4(x) &= 1 + (\lfloor 1.8 \rfloor + 1)(1) + (\lfloor 1.4 \rfloor + 1)(3) &= 9,
\end{aligned}$$

which are indeed the indices of the four lattice nodes surrounding x.

Given the nodes that enclose x, there are a number of linear interpolation methods corresponding to weights on these nodes. For example, in three dimensions, trilinear, pyramidal, and tetrahedral interpolation are all linear interpolations that result in different weights [6]. Lattice regression can accommodate all of these interpolation techniques, but here we present only the case of *d-linear interpolation* (e.g. bilinear/trilinear interpolation) because it is arguably the most popular variant of linear interpolation, can be implemented efficiently, and has the theoretical support of being the maximum entropy solution to the underdetermined linear interpolation equations [36].

Computing the d-Linear Interpolation Weights

Let $w_j(x)$ be the weight associated with the jth vertex $a_{c_j(x)}$ (for $j = 1, \ldots, 2^d$) of the lattice cell that contains x. For d-linear interpolation, $w_j(x)$ can be computed as

$$w_j(x) = \prod_{k=1}^{d} \lambda(x)_k^{N_2(j)_k} \big(1 - \lambda(x)_k\big)^{1 - N_2(j)_k},$$

where $N_2(j)_k$ denotes the kth component of $N_2(j)$, $\lambda(x) = x - \lfloor x \rfloor$, and $\lfloor \cdot \rfloor$ is performed component-wise. Note that in the above weight equation $N_2(j)_k$ is either 1 or 0, and thus

acts like a selector of either the $\lambda(x)_k$ or $(1-\lambda(x)_k)$ term, automatically selecting whichever of the two is positive.

Let $W(x)$ be the $1 \times m$ sparse vector with kth element

$$W(x)_k = \begin{cases} w_j(x) & \text{if } k = c_j(x) \text{ for } j = 1, \ldots, 2^d \\ 0 & \text{otherwise} \end{cases}$$

and for the $m \times 1$ matrix of lattice outputs b, the function value at x is interpolated as $W(x)b$. Likewise, given the matrix of n training vectors $\mathbf{X} = \big[x_1, \ldots, x_n\big]$, let $\mathbf{W}$ be the $n \times m$ matrix $\mathbf{W} = \big[W(x_1), \ldots, W(x_n)\big]$.

4.1.2 Lattice Regression: The Regularization Term

Minimizing the empirical risk term given in (4.4) does not dictate how to set the value of the lattice nodes that do not contribute to the interpolation of training samples. Mathematically, one can say that the empirical risk objective is *underdetermined*. That is, any choice of values for those lattice nodes will result in the same objective value. Prior knowledge about the nature of the underlying function should be added in order to set the value for such nodes. Specifically, one expects the function to be somewhat smooth, and we encode this information as a regularization term that prefers to give lattice nodes that are near one another output values that are also close, in some sense. Adding such a regularization term will also reduce the probability of over-fitting any noise in the training data measurements.

But how exactly should one measure the 'smoothness' of a function that is to be interpolated from a lattice? The answer to this question is likely to be highly application-dependent and thus the choice for the regularization term J in (4.2) may require careful consideration the function that is to be estimated. Here, we investigate three possibilities for regularization. The first, *Laplacian regularization*, is a first-order measure of smoothness in the sense that it penalizes absolute differences in output values for adjacent lattice nodes. The second, *thin-plate regularization*, is a second-order method that penalizes the integral of the second derivative of the estimated function (and penalizing deviations from linearity). The

third, *global bias regularization*, measures the adherence of the estimated function to a *bias function* supplied by some side information about the application.

Laplacian Regularization

A lattice can be represented as a graph with edges connecting adjacent nodes in the lattice. A standard approach to enforcing smoothness on the nodes of a graph is to minimize the graph Laplacian [50, 8], which can be expressed as the sum of squared differences between the values at adjacent nodes in a graph, that is:

$$\begin{aligned} J_{\mathbf{L}}(b) &= \sum_{\text{adjacent } a_i, a_j} (b_i - b_j)^2 \\ &= b^T \mathbf{L} b, \end{aligned} \tag{4.7}$$

where the normalized graph Laplacian [15] L is defined as follows. Given the $m \times m$ lattice adjacency matrix $\mathbf{A}$ where $\mathbf{A}_{ij} = 1$ for nodes directly adjacent to one another and 0 otherwise,

$$\mathbf{L} = 2\frac{\text{diag}(\mathbf{1}^T\mathbf{A}) - \mathbf{A}}{\mathbf{1}^T\mathbf{A}\mathbf{1}}$$

where $\mathbf{1}$ is the $m^d \times 1$ all-ones vector and $\text{diag}(\cdot)$ maps a vector to the diagonal of a square all-zeros matrix.

Solving for a lattice that minimizes the empirical risk (4.3) and also minimizes the Laplacian given by (4.7) forces the values chosen for adjacent nodes in the lattice to be close, and is expressed as:

$$\hat{b} = \arg\min_{b \in \mathbb{R}^m} \|\mathbf{W}b - y\|_2^2 + \lambda J_{\mathbf{L}}(b), \tag{4.8}$$

where the regularization parameter $\lambda > 0$ trades-off the two goals. Just as with (4.5), the solution to (4.8) has a closed form that can be efficiently computed via sparse Cholesky factorization

$$\hat{b} = (\mathbf{W}^T\mathbf{W} + \lambda\mathbf{L})^{-1}\mathbf{W}^T y. \tag{4.9}$$

Discrete Second-order Regularization

As mentioned in the introduction of this subsection, thin-plate regularization is a second-order method that operates on the second derivative of the estimated function. Before going into the details of this regularization technique, we address the natural question that arises from considering Laplacian regularization: Why not penalize the *discrete* second-order differences of the lattice values, analogous to a higher-order form of the Laplacian regularizer? Let us consider this approach briefly.

Consider the case of a one-dimensional lattice with outputs $b = [b_1, b_2, \ldots, b_m]$ and inputs $a_i = i$ for $i = 1, \ldots, m$ such that lattice nodes are spaced at integer distances in the domain. An un-normalized Laplacian regularization applies the following penalty on the lattice outputs:

$$b^T \mathbf{L} b = \sum_{i=1}^{m-1} (b_i - b_{i+1})^2. \tag{4.10}$$

This is the sum of the squared discrete first-order differences. The Laplacian matrix $\mathbf{L}$ is positive definite and thus lends itself naturally as a regularizer.

Let us now consider the natural extension of this regularization to the second-order. That is,

$$b^T \mathbf{M} b = \sum_{i=2}^{m-1} (b_{i-1} - 2b_i + b_{i+1})^2. \tag{4.11}$$

for some matrix $\mathbf{M}$. We can solve for the matrix $\mathbf{M}$ that satisfies this relationship by decomposing the summation.

$$\begin{aligned}
b^T \mathbf{M} b &= \sum_{i=2}^{m-1} (b_{i-1}^2 - 2b_i + b_{i+1})^2 \\
&= \sum_{i=2}^{m-1} b_{i-1}^2 - 4b_{i-1}b_i + 2b_{i-1}b_{i+1} + 4b_i^2 - 4b_i b_{i+1} + b_{i+1} \\
&= b_1^2 + 5b_2^2 + 6 \sum_{i=3}^{m-2} b_i^2 + 5b_{m-1}^2 + b_m^2 \; + \; \sum_{i=2}^{m-1} b_{i-1}b_{i+1} - 4b_{i-1}b_i - 4b_i b_{i+1} + b_{i-1}b_{i+1}
\end{aligned}$$

Rewriting this relationship, one sees that $\mathbf{M}$ has the following structure

$$\mathbf{M} = \begin{bmatrix} 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & 5 & -4 & 1 & \ddots & \ddots & \ddots & \ddots & 0 \\ 1 & -4 & 6 & -4 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 1 & -4 & 6 & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & 6 & -4 & 1 & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & -4 & 6 & -4 & 1 \\ 0 & \ddots & \ddots & \ddots & \ddots & 1 & -4 & 5 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \end{bmatrix},$$

and is ill-defined when $m < 5$. Inspection of the eigenstructure of this matrix shows that it is not positive definite (it has two eigenvalues that are zero). Since positive definiteness is a prerequisite for regularization this approach was not explored further.

Thin-plate regularization

The *thin-plate regularizer* [33] borrows its name from an analogous physical process. Imagine you were to take an thin, flat plate of aluminum and bend it into a wiggly shape. The thin-plate regularizer captures the amount of 'bending energy' required to keep the plate in this wiggly position. The tighter the kinks in the plate, the more energy needed to hold the position. Holding the plate flat and rotating it in space requires no energy at all. This is mathematically identical to a form of second-order regularization. Note that the mathematical description of the thin-plate regularizer is relatively simple to grasp but the precise form as used in this work (the details of the matrix $\mathbf{K}$ to be used as $b^T\mathbf{K}b$) is difficult to arrive at. Therefore, we will begin by focusing on the definition before providing the implementation details.

Let $\hat{f}$ be the function which results interpolating the lattice $\{a_j, b_j\}_{j=1:m}$, let $c_1, c_2, \ldots, c_d$ denote the unit vectors of $\mathbb{R}^d$, and let $\Omega \subset \mathbb{R}^d$ denote the convex hull of the lattice in $\mathbb{R}^d$.

The thin-plate regularizer $J_{\mathbf{K}}$ can be written as

$$J_{\mathbf{K}}(\hat{f}) = \int_{\Omega} \left(\sum_{i=1}^{d} \sum_{j=1}^{d} \frac{\partial}{\partial c_i} \frac{\partial}{\partial c_j} \right) \hat{f}^2(x)\, dx \tag{4.12}$$

$$= \int_{\Omega} \mathbf{1}^T \big(\nabla \hat{f}(x)\big) \big(\nabla \hat{f}(x)\big)^T \mathbf{1}\, dx \tag{4.13}$$

This can be thought of as the "roughness" of the function in the domain Ω [33].

In order to incorporate such a penalty into the lattice regression framework, we would like to rewrite $J_{\mathbf{K}}$ as

$$J_{\mathbf{K}}(b) = b^T \mathbf{K} b \tag{4.14}$$

for some (sparse) matrix $\mathbf{K}$, where b is the vector of $\{b_j\}$s from which $\hat{f}$ emerges. Luckily, such a form exists. The rest of this section is devoted to the computation of $\mathbf{K}$.

In order for (4.12) to be well-defined, the function $\hat{f}$ must be twice-differentiable. This precludes the computation of (4.12) for the case of d-linear interpolation (which we would like to employ for its computational efficiency). However, we posit that penalizing of (4.12) for d-cubic interpolation (the lowest order interpolation that produces an $\hat{f}$ that is twice-differentiable) will result in a function that exhibits the desired smoothness properties regardless of whether d-cubic or d-linear interpolation is used to interpolate the lattice.

The transformation of (4.12) into (4.14) relies on the fact that $\hat{f}$ has a basis representation. That is, for an arbitrary point $x \in \Omega$,

$$\hat{f}(x;\, \{a_j, b_j\}_{j=1:m}) = \sum_{j=1}^{m} k(x, a_j)\, b_j \tag{4.15}$$

where $k : \Omega \times \Omega \to \mathbb{R}$ is the basis function associated with the function class of $\hat{f}$ (see Fig. 4.3). For d-cubic interpolation, this basis function can be expressed as the tensor product of d one-dimensional basis functions centered at each lattice node:

$$k(x, a_j) = \prod_{i=1}^{d} \delta_{ij}\Big((x)_i\Big) \tag{4.16}$$

where $(\cdot)_i$ extracts the ith dimension of a vector and with

$$\delta_{ij}(u) = \tilde{\delta}\left(\frac{|u - (a_j)_i|}{h_i}\right) \tag{4.17}$$

for $u \in \mathbb{R}$ where $h_i \in \mathbb{R}$ is the spacing of the lattice in the ith dimension. This is the centering and scaling of the following one-dimensional cubic interpolation basis function about the jth lattice node:

$$\tilde{\delta}(u) = \begin{cases} 1.5u^3 - 2.5u^2 + 1 & \text{if } 0 \le u < 1 \\ -.5u^3 + 2.5u^2 - 4u + 2 & \text{if } 1 \le u < 2 \\ 0 & \text{otherwise,} \end{cases} \tag{4.18}$$

where, again, $u \in \mathbb{R}$.

Now, with the basis functions defined, the (i, j)-entry of the matrix $\mathbf{K}$ can be computed as follows

$$\mathbf{K}_{i,j} = \sum_{k=1}^{d} D_{k\,i,j}^{(2)} \left(\prod_{r \neq k} D_{r\,i,j}^{(0)} \right) \; + \; D_{k\,i,j}^{(1)} \sum_{\ell \neq k} D_{\ell\,i,j}^{(1)} \left(\prod_{r \neq k,\ell} D_{r\,i,j}^{(0)} \right) \tag{4.19}$$

$$= \begin{array}{ccccccccc} D_{1\,i,j}^{(2)} D_{2\,i,j}^{(0)} \dots D_{d\,i,j}^{(0)} & + & D_{1\,i,j}^{(1)} D_{2\,i,j}^{(1)} \dots D_{d\,i,j}^{(0)} & + & \cdots & + & D_{1\,i,j}^{(1)} D_{2\,i,j}^{(0)} \dots D_{d\,i,j}^{(1)} & + \cdots \\ D_{1\,i,j}^{(1)} D_{2\,i,j}^{(1)} \dots D_{d\,i,j}^{(0)} & + & D_{1\,i,j}^{(0)} D_{2\,i,j}^{(2)} \dots D_{d\,i,j}^{(0)} & + & \cdots & + & D_{1\,i,j}^{(0)} D_{2\,i,j}^{(1)} \dots D_{d\,i,j}^{(1)} & + \cdots \\ \vdots & & \vdots & & \ddots & & \vdots & \\ D_{1\,i,j}^{(1)} D_{2\,i,j}^{(0)} \dots D_{d\,i,j}^{(1)} & + & D_{1\,i,j}^{(0)} D_{2\,i,j}^{(1)} \dots D_{d\,i,j}^{(1)} & + & \cdots & + & D_{1\,i,j}^{(0)} D_{2\,i,j}^{(0)} \dots D_{d\,i,j}^{(2)} & \end{array}$$

where

$$D_{k\,i,j}^{(s)} = \int \delta_{ki}^{(s)}(u) \delta_{kj}^{(s)}(u) du \tag{4.20}$$

with $f^{(s)}(u)$ denoting the sth derivative of the function $f(u)$.

Because δ is piecewise-polynomial, (4.20) can be computed algebraically. Furthermore, the computation of $\mathbf{K}$ is decoupled from the lattice values b, and can thus be pre-computed once for a particular combination of lattice size, dimension, and spacing. Once computed, it can be stored and reused indefinitely.

Global Bias

So far, we have focused on regularization techniques that penalize the "smoothness" of the function $\hat{f}$ as measured by local differences or derivatives. Here, we offer the alternative (and at times complementary) approach of regularizing the estimated function toward a *bias function* $\tilde{g} : \mathbb{R}^d \to \mathbb{R}$. Ideally, such a bias function is derived from the data $\{x_i, y_i\}$ and some prior knowledge about the application.

Given a bias function $\tilde{g}$, the simplest way to regularize the lattice output toward this function is to penalize the divergence of b_i from $\tilde{g}(a_i)$ for $i = 1, \ldots, m$. Letting $g = [\tilde{g}(a_1), \tilde{g}(a_2), \ldots, \tilde{g}(a_m)]^T$, we may write

$$J_g(b) = \sum_{i=1}^{m} \big(b_i - g_i\big)^2 \tag{4.21}$$

$$= (b - g)^T (b - g). \tag{4.22}$$

This technique can be used to adapt an existing lattice to new data in an incremental way, or as we will see in the experiments section, it can be used to correct for negative effects of other regularizers. For instance, Laplacian regularization rewards smooth transitions between adjacent lattice outputs but there is no incentive to extrapolate the function beyond the boundary of the cells that contain training samples. This can result in clipping of the output function (see Fig. 4.2 left) which can be corrected somewhat by applying Laplacian regularization along with a bias regularization toward a global trilinear fit of the data (see Fig. 4.2 center). Further, we see that the second-order nature of the thin-plate regularizer naturally encourages extrapolation (see Fig. 4.2 right).

4.2 Choice of Interpolation Function

When the thin-plate regularizer is used with lattice regression, it bears a striking resemblance to a smoothing tensor cubic b-spline with fixed knot locations. In this section, we explore the similarities and differences between these two approaches. In the interest of brevity, the term **splines** herein will refer to **smoothing tensor cubic b-splines** with **fixed knot locations**. Note that the knot locations for a tensor b-spline consist of the

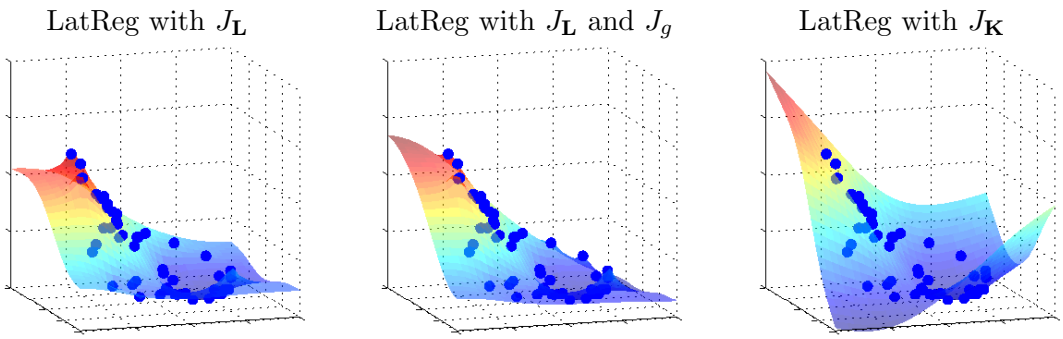


Figure 4.2: Figure illustrates the extrapolation effect of the different regularizers when combined with the lattice regression empirical risk minimization. The training data (blue dots) used to fit each of the models was the same in all cases, and the true function is a sum of Gaussian pdfs.

tensor product of (not necessarily uniform) knots along each dimension, producing a (not necessarily uniform) grid. This is in contrast to the more common thin-plate spline which can have arbitrary knot locations (usually coincident with the training data) within the domain.

The connection between lattice regression and splines is best exposed via the *basis function interpretation* of these regression techniques. That is, each method can be interpreted as fitting a function that can be represented as the linear combination of a basis function k that is repeated and shifted to the lattice node (knot) locations:

$$\hat{f}(x) = \sum_{i=1}^{m} w_i \, k(x, x_i) \tag{4.23}$$

where w_i is the weight given to the ith basis function centered at the lattice node x_i.

The relevant basis functions for the one dimensional domain are shown in the top row of Fig. 4.3. These basis functions can be expanded to an arbitrary dimension via the tensor product of multiple 1-d bases. That is, a d-dimensional basis k^d can be constructed as

$$k^d(x, x_i) = \prod_{j=1}^{d} k\big((x)_j, (x_i)_j\big) \tag{4.24}$$

where the operator $(\cdot)_j : \mathbb{R}^d \to \mathbb{R}$ takes a vector and returns the jth element. The corresponding bases in two dimensions are shown in the bottom row of Fig. 4.3.

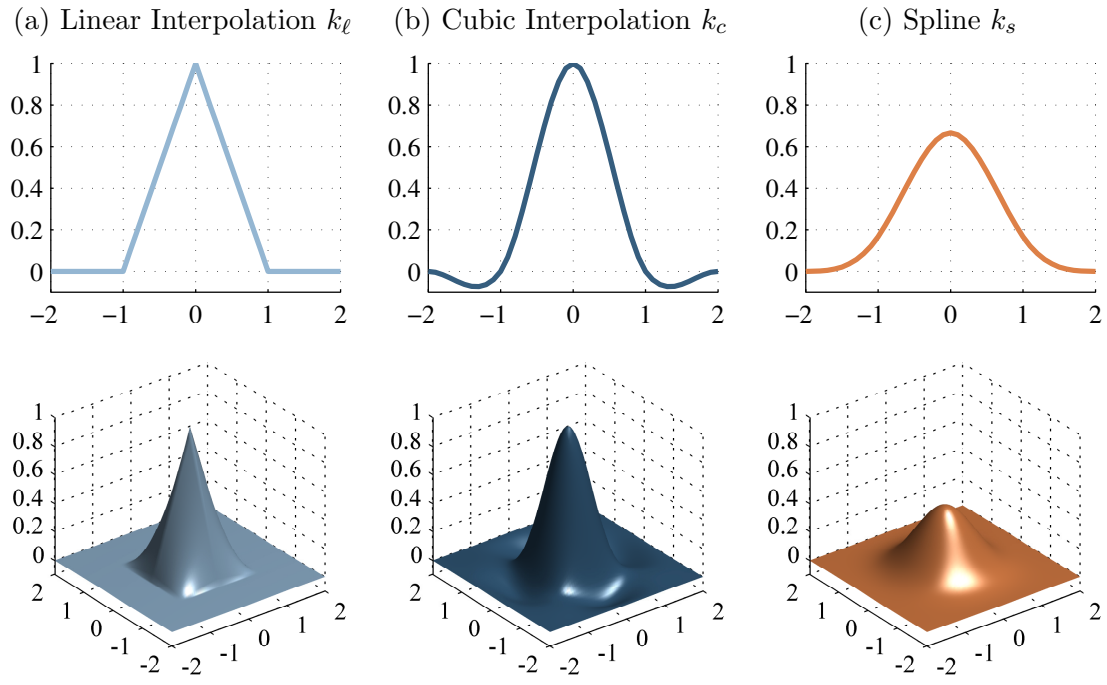


Figure 4.3: Linear interpolation, cubic interpolation and spline basis functions. Each basis function is plotted with one argument set at the origin $k(\cdot, \mathbf{0})$.

The primary benefit of lattice regression is the speed of test evaluations provided by linear interpolation on the lattice. Due to the local support of the linear interpolation basis function, in d dimensions only 2^d lattice points are involved in calculating the response for a given test point. The larger support of the cubic interpolation basis function and the b-spline basis function result in longer test evaluation. Further efficiency can be gained by exploiting the arithmetic involved in the calculation of linear interpolation in fixed point computations [70]. That is, linear interpolation weights for a test point can be computed by binary operations such as bit-shifting on fixed-point numbers; this is not possible for the computation of cubic interpolation and spline weights. This property is particularly beneficial in applications such as printer color management where dedicated hardware is used to perform test evaluations. A comparison of the test-evaluation speeds of these basis

functions in two and three dimensions is shown in Fig. 4.4. All three methods were implemented in the same code and no hardware speedups were exploited, so this is a somewhat pessimistic comparison for linear interpolation.

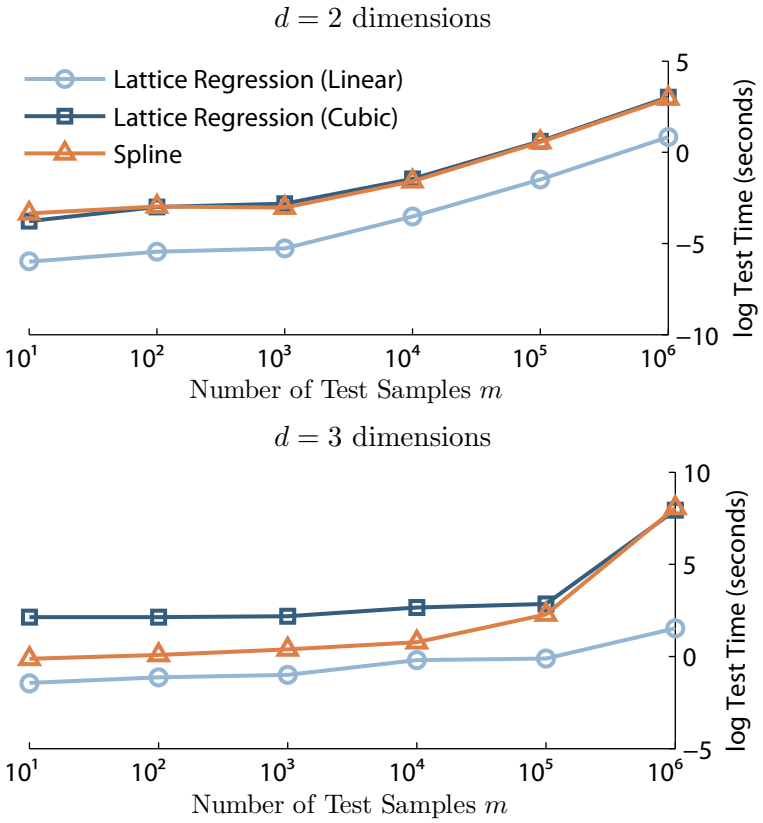


Figure 4.4: Shown is the test-evaluation time vs. the number of test samples on a log-scale for Lattice Regression optimized for d-linear interpolation (Lattice Regression (linear), Lattice Regression optimized for d-cubic interpolation (Lattice Regression (cubic), and Splines.

In the proposed lattice regression we use the linear interpolation in order to optimize the test-evaluation speed. One might expect greater accuracy from applying the cubic interpolation basis function or the spline basis function (at the aforementioned cost of reduced test-evaluation speed due to the larger support of the basis) and this is indeed the case in many situations. We now examine two questions that probe the nature of this trade-off. First, if one were willing to compromise the test-evaluation time, how much accuracy can be gained by cubic interpolation or splines compared to the proposed lattice

regression? Second, if one learns the lattice by optimizing a larger basis (such as cubic interpolation or splines) but then simply linearly interpolates this lattice to reduce the test-evaluation time, what is the loss in accuracy that is incurred compared to the proposed lattice regression? Since in practice it is far more common to linearly interpolate a lattice the practitioner might take the following perspective on these questions: The latter question illuminates what we gain by learning the lattice with the linear-interpolation basis function (that is, using lattice regression) compared to learning the lattice with cubic convolution or splines while the former question provides an optimistic performance baseline (in terms of estimation accuracy) that would be unacceptable to implement in practice.

The following set of experiments were performed on the domain $[0,1]^d$ using a real-valued function $f(x) = \sum_{i=1}^{10} f_i(x)$ where each f_i is a Gaussian pdf with uniformly drawn mean $\mu_i \sim U([-.5, 1.5]^d)$ and variance $\sigma_i^2 = .1$. In all experiments, a training sample of size n was drawn such that $x_i \sim U([0,1]^d)$ and $y_i = f(x_i)$ for $i = 1, \ldots, n$. Lattices with $m = \tilde{m}^d$ total nodes were constructed with inputs $a_j \in \prod_d \left[0, \frac{1}{\tilde{m}-1}, \frac{2}{\tilde{m}-1}, \ldots, 1\right]$ and outputs $b_j \in \mathbb{R}$ estimated by each algorithm; here $\prod_d X = X \times X \times \ldots \times X$ denotes the d-dimensional Cartesian product. The root mean square error (RMSE) between the true function f and the estimated $\hat{f}$ (interpolated from the lattice) was then computed for each method at k uniformly drawn locations in $[0,1]^d$.

4.2.1 Matched Interpolation and Estimation

In this subsection, we address the first of the questions posed: What is the difference in accuracy between (i) optimizing the lattice nodes for linear interpolation and using linear interpolation at test time, and (ii) optimizing the lattice nodes for a higher-order basis function method like splines and using that same higher-order basis function method at test time? Regardless of the basis-function used, all three methods share the same regularization parameter λ (c.f. (4.2)) which is typically set via cross-validation on the training data (because an appropriate value for this will depend on the smoothness of the true function f that is to be estimated). To reduce the number of variables for the comparison, we chose to fix λ to a 'reasonable' value for these experiments. In order to find a suitable value, we

analyzed the test error (for k=10,000 locations) in the estimation of the Gaussian mixture function; the results are shown in Fig. 4.5. Please note that in the following figures, the solid lines correspond to accuracy (left axes) and dotted lines correspond to test-evaluation times (right axes).

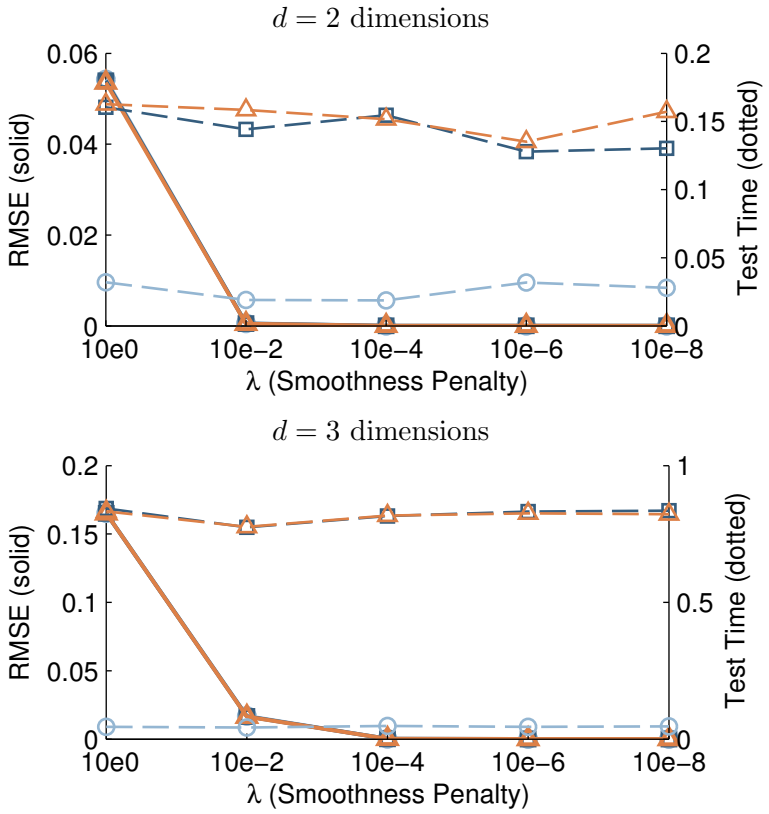
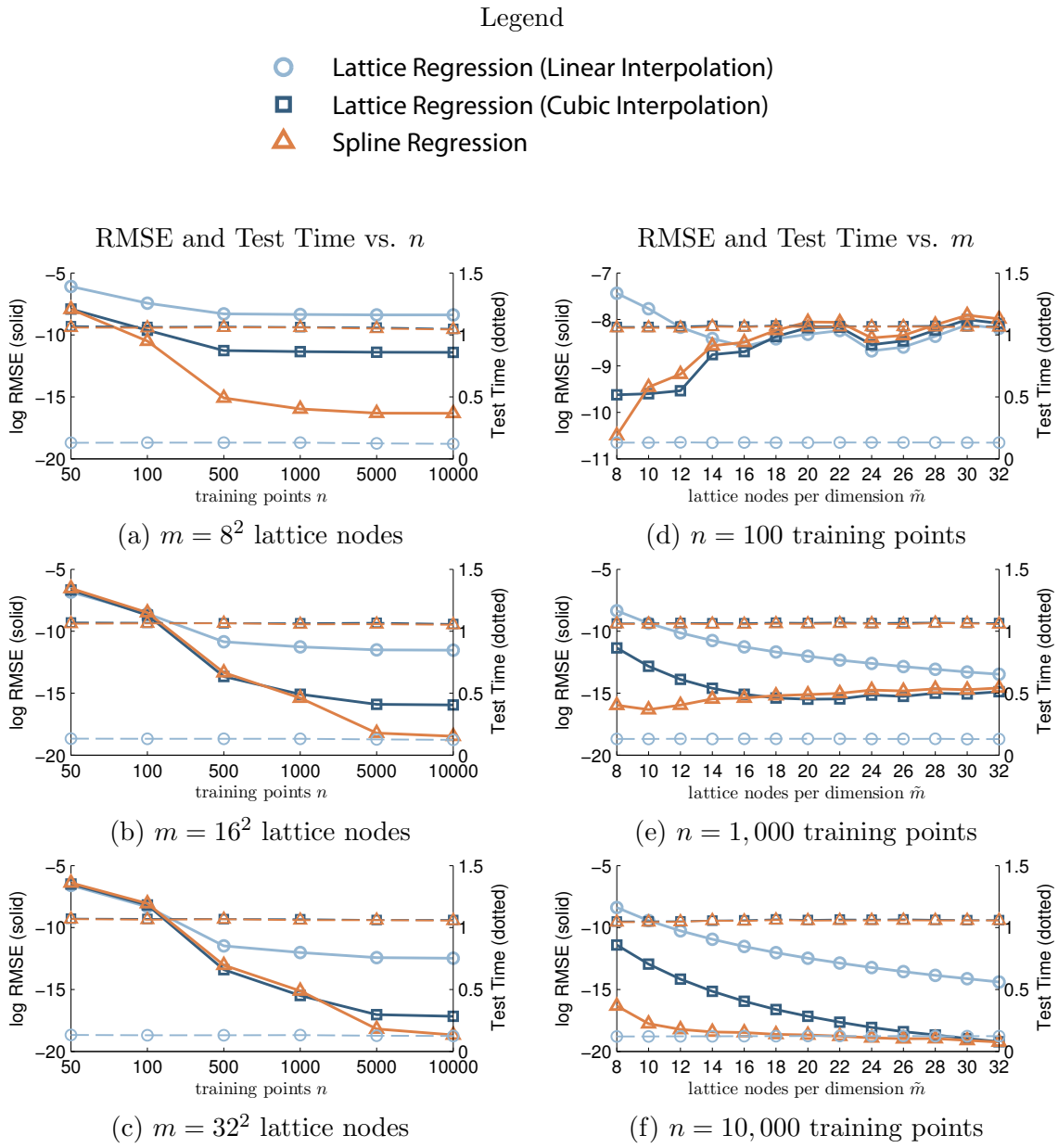


Figure 4.5: Typical plot of RMSE vs. Smoothness Parameter λ. Computed from $n = 1,000$ training points with $m = 16^d$ lattice nodes. Though the specific results vary with choice of n and m, the value $\lambda = 10^{-6}$ gives reasonable performance for the ranges investigated in two and three dimensions.

With the value of λ fixed, a comparison of the performance of the algorithms for a range of training set sizes n and lattice sizes m is made. A comparison on $d = 2$ dimensions is shown in Fig. 4.6 and a comparison on $d = 3$ dimensions is shown in Fig. 4.7.

As the figures demonstrate, all three methods perform comparably in accuracy when the number of training samples n is small and the number of lattice nodes m is large. The



(a) $m = 8^2$ lattice nodes

(b) $m = 16^2$ lattice nodes

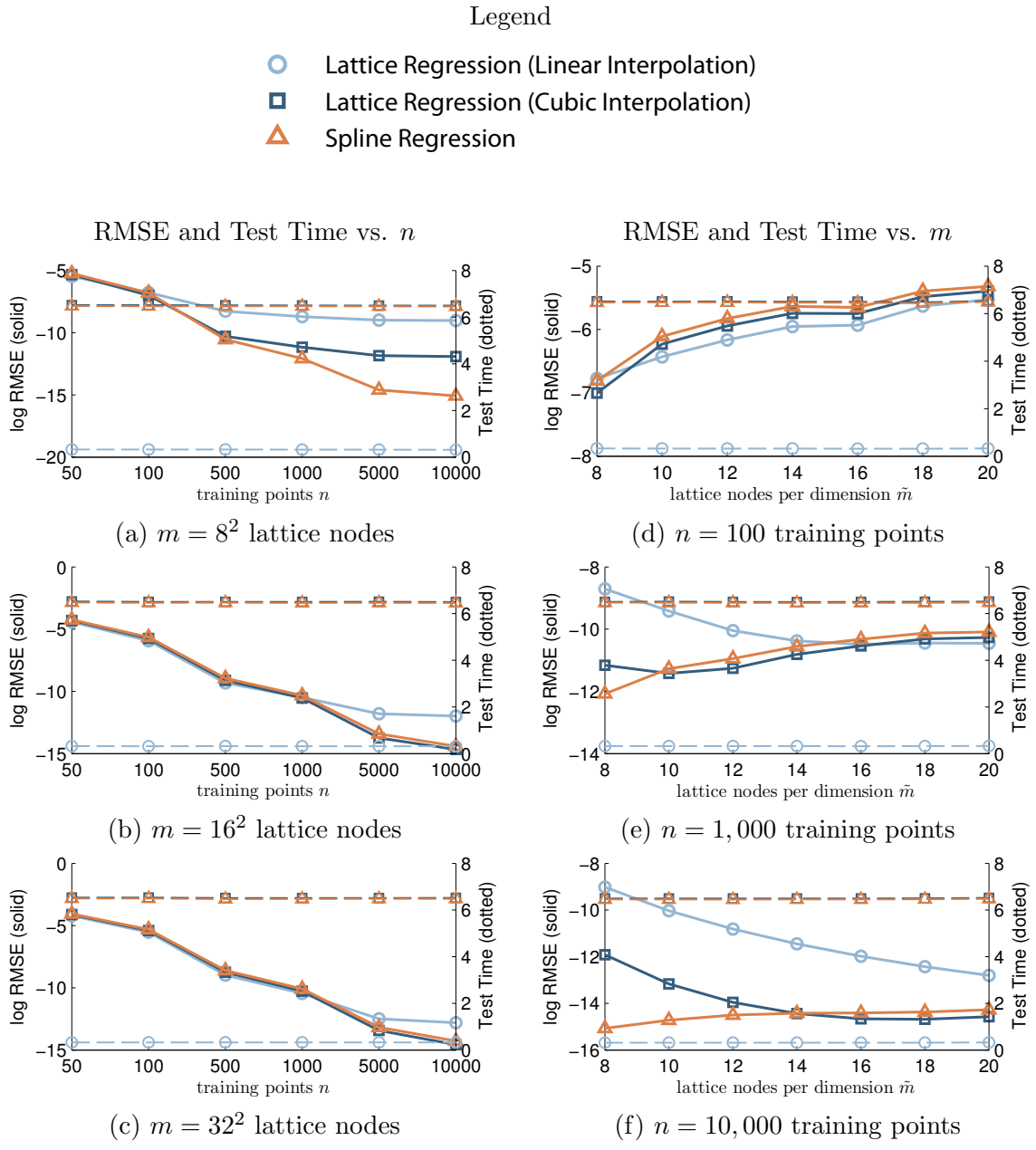
(c) $m = 32^2$ lattice nodes

(d) $n = 100$ training points

(e) $n = 1,000$ training points

(f) $n = 10,000$ training points

Figure 4.6: Matched Interpolation and Estimation. Estimation of a $d = 2$ dimensional Gaussian mixture. Shown are RMS test error for $k = 10,000$ uniformly drawn locations in $[0, 1]^2$ and the test-evaluation time (seconds) for varying lattice sizes m and training sample sizes n.



(a) $m = 8^2$ lattice nodes

(d) $n = 100$ training points

(b) $m = 16^2$ lattice nodes

(e) $n = 1,000$ training points

(c) $m = 32^2$ lattice nodes

(f) $n = 10,000$ training points

Figure 4.7: Matched Interpolation and Estimation. Estimation of a $d = 3$ dimensional Gaussian mixture. Shown are RMS test error for $k = 10,000$ uniformly drawn locations in $[0, 1]^3$ and the test-evaluation time (seconds) for varying lattice sizes m and training sample sizes n.

latter is to be expected because, as the spacing of lattice nodes decreases, the fit provided by linear interpolation becomes more flexible.

The left-most columns of Fig. 4.6 and Fig. 4.7 show the effect that the number of training samples n has on test-error and test-evaluation time. In all cases, one sees a monotonic downward trend in test-error. This is to be expected because an estimate based on more samples of the true function will naturally be more accurate. The three methods perform comparably in test error for the smaller training sets and diverge as the set grows with splines besting cubic interpolation besting linear interpolation. However, for larger lattice sizes, it takes more training samples to see a difference between the methods. We see in all cases that the linear interpolation takes much less time to evaluate than cubic interpolation or splines.

The right-most columns of Fig. 4.6 and Fig. 4.7 show the effect that lattice size m has on test-error and test-evaluation time. On all three methods, we see negligible difference in test-evaluation time for increasing grid sizes. The test error, however, exhibits a few interesting trends to be examined. First, when the number of training samples n is small (Fig. 4.6d, and Fig. 4.7d,e), increasing the resolution of the lattice does not reduce the test error. This is most likely due to over-fitting on a small training sample and could be mitigated in practice by adjusting the smoothness parameter λ. On the larger training sets, linear interpolation performs worse than the other two methods, but seems to converge in performance as the grid size m is increased.

4.2.2 Mismatched Interpolation and Estimation

In this subsection, we investigate the second question posed: What is the difference in accuracy between optimizing the lattice for spline, cubic, or linear interpolation, if at test-time – for efficiency – the lattice is interpolated with linear interpolation?

It should be noted that a lattice optimized for the spline basis function (see Fig. 4.3c) requires a pre-filtering step before applying linear interpolation. This is because the basis function is not 'interpolating' in the sense that the value at a lattice node is not identical to the value of the estimated function at that location. The actual value of the function

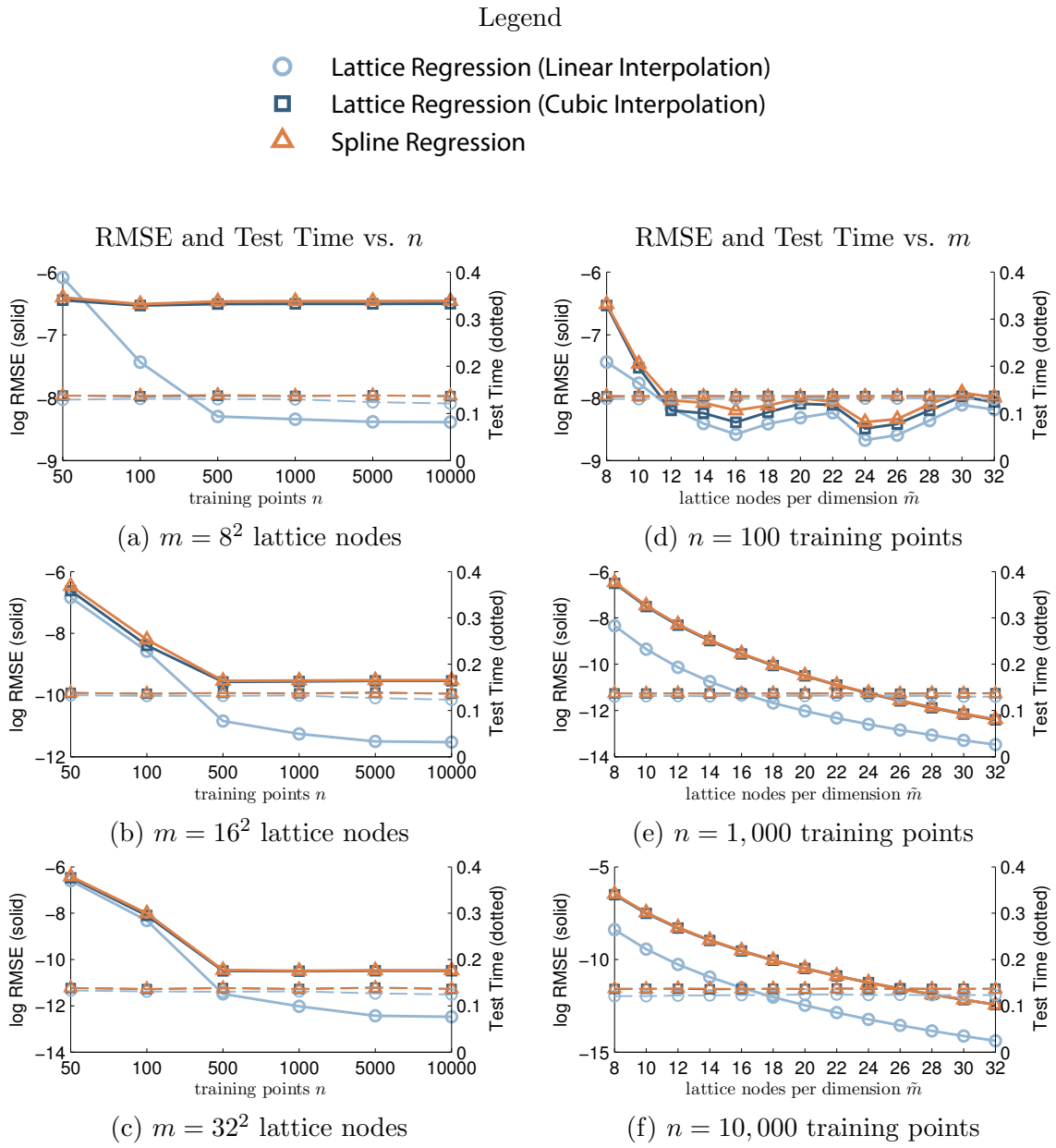
will be a linear combination of the lattice value and the adjacent lattice values (weighted according to Fig. 4.3c). This is not the case for the linear and cubic interpolation basis functions. This fact can be seen graphically by noting that the interpolation basis functions are identically zero at integer distances from the center while the spline basis function is not (c.f. Fig. 4.3). All evaluations with splines in this subsection have been pre-filtered to arrive at the estimated function values at the lattice nodes prior to interpolation and the time required to do so was not counted against the method in the test-time evaluation.

We fix λ for each of the regression problems and compare of the performance in both test-evaluation time and accuracy on the same Gaussian mixture regression problem of the previous subsection. Again, a range of training set sizes n and lattice sizes m are tested to explore the behavior of each of the basis functions. A comparison on $d = 2$ dimensions is shown in Fig. 4.8 and a comparison on $d = 3$ dimensions is shown in Fig. 4.9.

As expected, the test times for all three methods are comparable, since they all use linear interpolation on the lattice to evaluate test samples. We see similar trends in which a difference in accuracy is notable when the training set size is large relative to the lattice size. Again, the methods converge for larger lattice sizes. In most of the cases evaluated, we see that lattice regression optimized for linear interpolation outperforms the regressions optimized for spline and cubic interpolation basis functions. This is exactly as expected.

4.3 Lattice Regression Experiments

In this section, we test lattice regression first on simulated data and then in three diverse application domains. In the first application, functions of the form $\mathbb{R}^3 \rightarrow \mathbb{R}$ are estimated in the color management of digital printers (as outlined in Section 2.2). In this domain, linear interpolation from a lattice is the preferred function representation because the function is typically implemented in hardware and millions of pixels must be processed for each image sent to the printer. The second application is the regression of annual rainfall measurements over a geographic area. This was chosen as a typical example of the $\mathbb{R}^2 \rightarrow \mathbb{R}$ functions learned for geospatial interpolation applications (elevation, rainfall, wind speed, temperature, etc). In this domain, look-up speed is typically not as important but it is nonetheless standard in geographic information systems to express the estimated geospa-



(a) $m = 8^2$ lattice nodes

(b) $m = 16^2$ lattice nodes

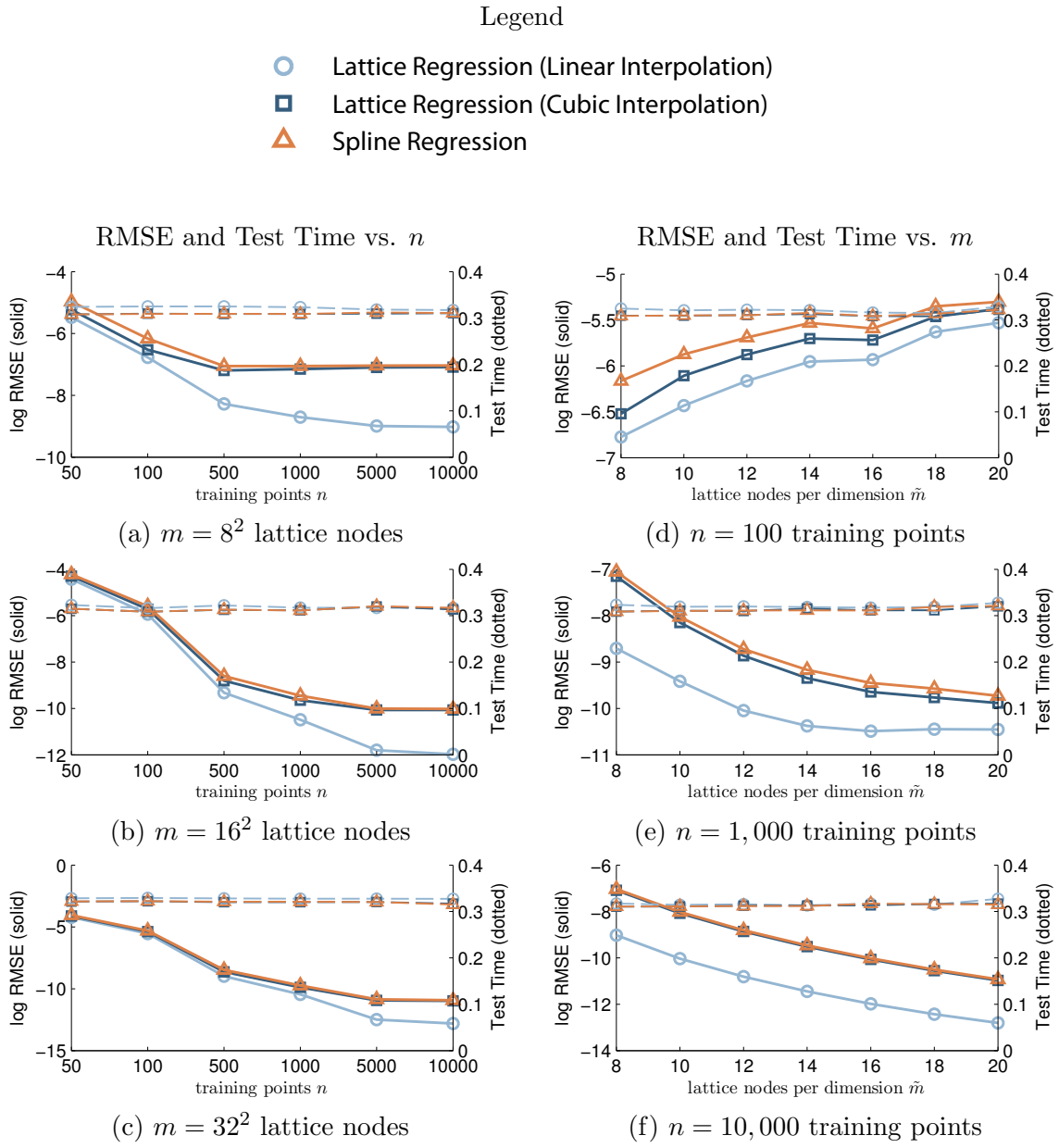
(c) $m = 32^2$ lattice nodes

(d) $n = 100$ training points

(e) $n = 1,000$ training points

(f) $n = 10,000$ training points

Figure 4.8: Mismatched Interpolation and Estimation. Estimation of a $d = 2$ dimensional Gaussian mixture. Shown are RMS test error for $k = 10,000$ uniformly drawn locations in $[0, 1]^2$ and the test-evaluation time (seconds) for varying lattice sizes m and training sample sizes n.



(a) $m = 8^2$ lattice nodes

(d) $n = 100$ training points

(b) $m = 16^2$ lattice nodes

(e) $n = 1,000$ training points

(c) $m = 32^2$ lattice nodes

(f) $n = 10,000$ training points

Figure 4.9: Mismatched Interpolation and Estimation. Estimation of a $d = 3$ dimensional Gaussian mixture. Shown are RMS test error for $k = 10,000$ uniformly drawn locations in $[0, 1]^3$ and the test-evaluation time (seconds) for varying lattice sizes m and training sample sizes n.

tial function as a regular grid. The final application is the registration of omni-directional images for super-resolution (as outlined in Section 2.4). In this application, we use lattice regression to learn a real-valued function on the surface of a sphere $[0, \pi] \times [-\pi, \pi] \to \mathbb{R}$. This function is used in the inner loop of a greedy optimization procedure, placing a premium on test-evaluation performance and motivating the choice of lattice regression for its estimation.

In many of the experiments, Laplacian regularization $J_{\mathbf{L}}$ is used in conjunction with global bias regularization in order to provide extrapolation outside of the convex hull of the training data $\mathbf{conv}(\{x_i\}_{i=1:m})$. For this, we modify (4.2) to be

$$\hat{b} = \arg\min_{b \in \mathbb{R}^m} \|\mathbf{W}b - y\|_2^2 + \lambda_{\mathbf{L}} J_{\mathbf{L}}(b) + \lambda_g J_g(b), \tag{4.25}$$

now with two regularization parameters $\lambda_{\mathbf{L}}$ and λ_g that must be trained by cross-validation.

Many experiments compare to Gaussian process regression (GPR). To implement GPR, we used the MATLAB code provided by Rasmussen and Williams at http://www.GaussianProcess.org/gpml. The covariance function was set as the sum of a squared-exponential with an independent Gaussian noise contribution and all data were demeaned by the mean of the training outputs before applying GPR. The hyperparameters for GPR were set by maximizing the marginal likelihood of the training data (for details, see Rasmussen and Williams [60]). To mitigate the problem of choosing a poor local maxima, gradient descent was performed from 20 random starting log-hyperparameter values drawn uniformly from $[-10, 10]^3$ and the maximal solution was chosen. The parameters for all other algorithms were set by minimizing the 10-fold cross-validation error using the Nelder-Mead simplex method, bounded to values in the range $[10^{-3}, 10^3]$. The starting point for this search was set at the default parameter setting for each algorithm: $\lambda = 1$ for local ridge regression[2] and $\lambda_{\mathbf{L}} = 1, \lambda_g = 1$ for lattice regression. Experiments on the simulated dataset comparing this approach to the standard cross-validation over a grid of values

[2]Note that no locality parameter is needed for this local ridge regression as the neighborhood size is automatically determined by enclosing k-NN [35].

$[10^{-3}, 10^{-2}, \ldots, 10^{3}] \times [10^{-3}, 10^{-2}, \ldots, 10^{3}]$ showed no difference in performance, and the former was nearly 50% faster.

4.3.1 Simulated Data

To begin, the effectiveness of the proposed method was analyzed with simulations on randomly-generated functions. We compared the proposed method to Gaussian process regression (GPR) applied directly to the final test points (no lattice), and to estimating test points by interpolating a lattice where the lattice nodes are learned by the same GPR. The first-order Laplacian regularization $J_{\mathbf{L}}$ was evaluated in conjunction with three different types of global bias to test the effectiveness of this strategy: 1) An "accurate" bias $\tilde{g}$ was learned by GPR on the training samples; 2) An "inaccurate" bias $\tilde{g}$ was learned by a global d-linear interpolation[3]; and 3) The no bias case (setting $\lambda_g = 0$ in (4.25)).

We analyzed the proposed method with simulations on randomly-generated piecewise-polynomial functions $f : \mathbb{R}^d \to \mathbb{R}$ constructed from the sum of one-dimensional splines. These functions are smooth but have features that occur at different length-scales; two-dimensional examples are shown in Fig. 4.10. To construct each function, we first drew ten iid random points $\{s_i\}$ from the uniform distribution on $[0,1]^d$, and ten iid random points $\{t_i\}$ from the uniform distribution on $[0,1]$. Then for each of the d dimensions we first fit a one-dimensional spline $\tilde{f}_k : \mathbb{R} \to \mathbb{R}$ to the pairs $\{\big((s_i)_k, t_i\big)\}$, where $(s_i)_k$ denotes the kth component of s_i. We then combined the d one-dimensional splines to form the d-dimensional function $\tilde{f}(x) = \sum_{k=1}^{d} \tilde{f}_k\big((x)_k\big)$, which was then scaled and shifted to have range spanning $[0, 100]$:

$$f(x) = 100\left(\frac{\tilde{f}(x) - \min_{z\in[0,1]^d} \tilde{f}(z)}{\max_{z\in[0,1]^d} \tilde{f}(z)}\right).$$

For input dimensions $d \in \{2, 3\}$, a set of 100 functions $\{f_1, \ldots, f_{100}\}$ were randomly generated as described above and a set of $n \in \{50, 1000\}$ randomly chosen training inputs $\{x_1, \ldots, x_n\}$ were fit by each regression method. A set of $\ell = 10,000$ randomly chosen test

[3]We considered the very coarse $m = 2^d$ lattice formed by the corner vertices of the original lattice and solved (4.5) for this one-cell lattice, using the result to interpolate the full set of lattice nodes, forming $g_i = \tilde{g}(a_i)$ for $i = 1, \ldots, m$

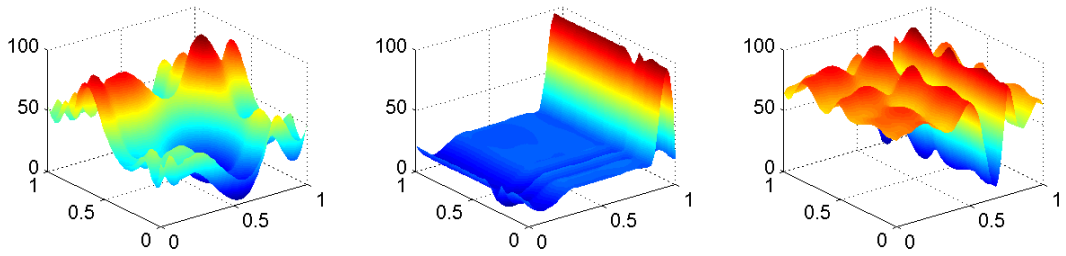


Figure 4.10: Example random piecewise-polynomial functions created by the sum of one-dimensional splines fit to ten uniformly drawn points in each dimension.

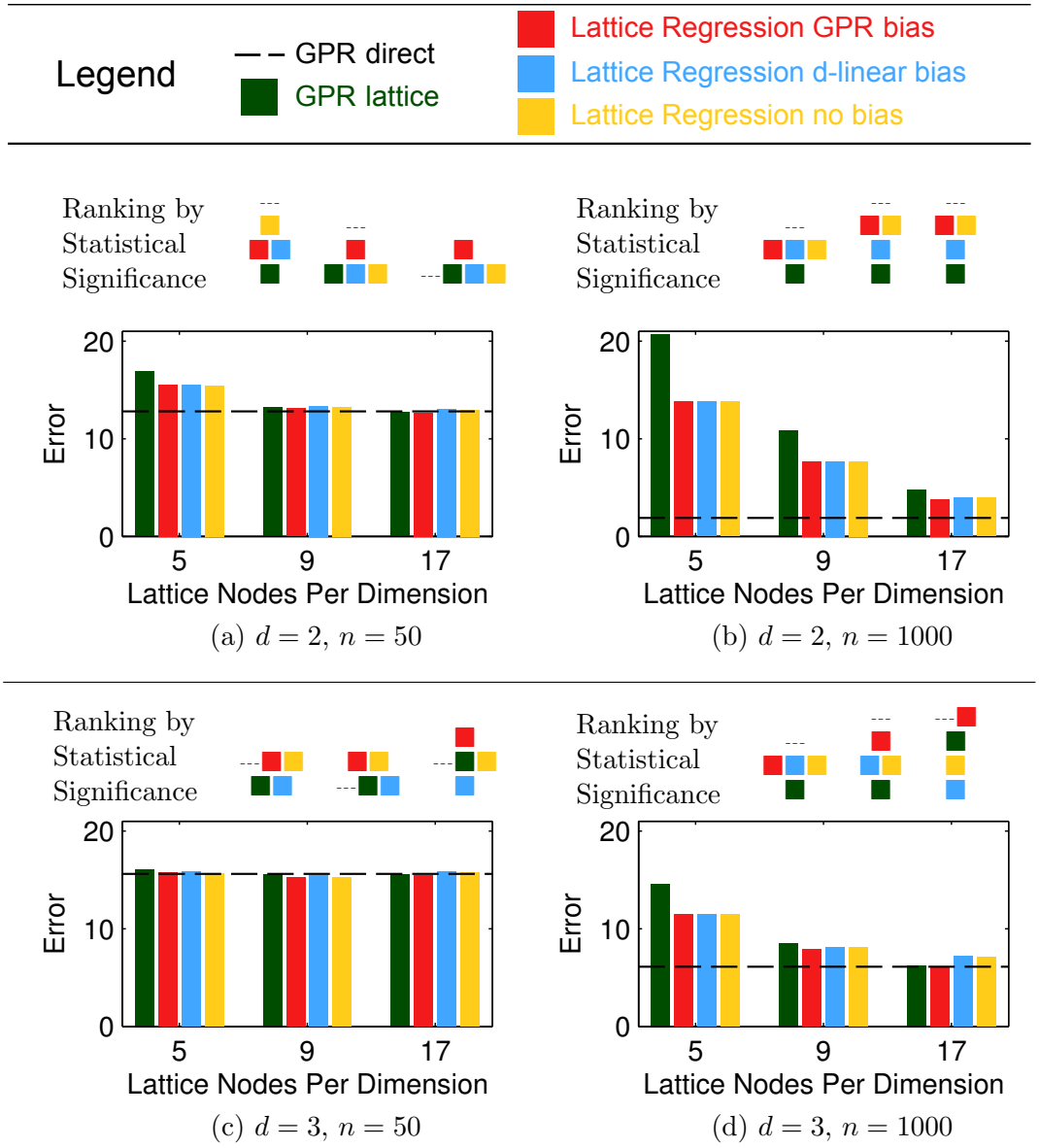
inputs $\{z_1, \ldots, z_\ell\}$ were used to evaluate the accuracy of each regression method in fitting these functions. For the rth randomly-generated function f_r, denote the estimate of the jth test sample by a regression method as $(\hat{y}_j)_r$. For each of the 100 functions and each regression method we computed the root mean-squared errors (RMSE) where the mean is over the $m = 10,000$ test samples:

$$e_r = \left(\frac{1}{\ell} \sum_{j=1}^{\ell} \left(f_r(z_j) - (\hat{y}_j)_r \right)^2 \right)^{1/2}.$$

The mean and statistical significance (as judged by a one-sided Wilcoxon with $p = 0.05$) of $\{e_r\}$ for $r = 1, \ldots, 100$ is shown in Fig. 4.11 for lattice resolutions of $5, 9$ and 17 nodes per dimension.

In interpreting the results of Fig. 4.11, it is important to note that the statistical significance test compares the ordering of relative errors between each pair of methods *across the random functions*. That is, it indicates whether one method consistently outperforms another in RMSE when fitting the randomly drawn functions.

Consistently across the random functions, and in all 12 experiments, lattice regression with a GPR bias performs better than applying GPR to the nodes of the lattice. At coarser lattice resolutions, the choice of bias function does not appear to be as important: in 7 of the 12 experiments (all at the low end of grid resolution) lattice regression using no bias does as well or better than that using a GPR bias.



(a) $d = 2$, $n = 50$

(b) $d = 2$, $n = 1000$

(c) $d = 3$, $n = 50$

(d) $d = 3$, $n = 1000$

Figure 4.11: Shown is the average RMSE of the estimates given by each regression method on the simulated dataset. As summarized in the legend, shown is GPR applied directly to the test samples (dotted line) and the bars are (from left to right) GPR applied to the nodes of a lattice which is then used to interpolate the test samples, lattice regression with a GPR bias, lattice regression with a d-linear regression bias, and lattice regression with no bias. The statistical significance corresponding to each group is shown as a hierarchy above each plot: method A is shown as stacked above method B if A performed statistically significantly better than B.

Interestingly, in 3 of the 12 experiments, lattice regression with a GPR bias achieves statistically significantly lower errors (albeit by a marginal average amount) than applying GPR directly to the random functions. This surprising behavior is also demonstrated on the real-world datasets in the following sections and is likely due to large extrapolations made by GPR and in contrast, interpolation from the lattice regularizes the estimate which reduces the overall error in these cases.

4.3.2 Geospatial Interpolation

Interpolation from a lattice is a common representation for storing geospatial data (measurements tied to geographic coordinates) such as elevation, rainfall, forest cover, wind speed, etc (see section 2.3). To investigate the effectiveness of the proposed technique in this domain, we tested it on the Spatial Interpolation Comparison 97 (SIC97) dataset [20] from the Journal of Geographic Information and Decision Analysis. This dataset is composed of 467 rainfall measurements made at distinct locations across Switzerland. Of these, 100 randomly chosen sites were designated as training to predict the rainfall at the remaining 367 sites. The RMSE of the predictions made by GPR and variants of the proposed method are presented in Fig. 4.12. Additionally, the statistical significance (as judged by a one-sided Wilcoxon with $p = 0.05$) of the differences in squared error on the 367 test samples was computed for each pair of techniques. In contrast to the previous section in which significance was computed on the RMSE across 100 randomly drawn functions, significance in this section indicates that one technique produced consistently lower squared error across the individual test samples.

Compared with GPR applied to a lattice, lattice regression with a GPR bias again produces a lower RMSE on all five lattice resolutions. However, for four of the five lattice resolutions, there is no performance improvement as judged by the statistical significance of the individual test errors. In comparing the effectiveness of the bias term, we see that on four of five lattice resolutions, using no bias and using the d-linear bias produce consistently lower errors than both the GPR bias and GPR applied to a lattice.

Additionally, for finer lattice resolutions ($\geq$ 17 nodes per dimension) lattice regression either outperforms or is not significantly worse than GPR applied directly to the test points. Inspection of the maximal errors confirms the behavior posited in the previous section: that interpolation from the lattice imposes a helpful regularization. The range of values produced by applying GPR directly lies within $[1, 552]$ while those produced by lattice regression (regardless of bias) lie in the range $[3, 521]$; the actual values at the test samples lie in the range $[0, 517]$.

4.3.3 Color Management Experiments

Although the proposed lattice regression is applicable in a wide variety of applications, the color management of printers has been the major driving application for its development. As such, we present two sets of color management experiments here focused on slightly different aspects of the color management problem. The first compares the color accuracy of lattice regression with a Laplacian and global bias regularizer c.f. (4.25) to GPR and local ridge regression using the enclosing k-NN neighborhood [35] (see also section 3.2.2). The

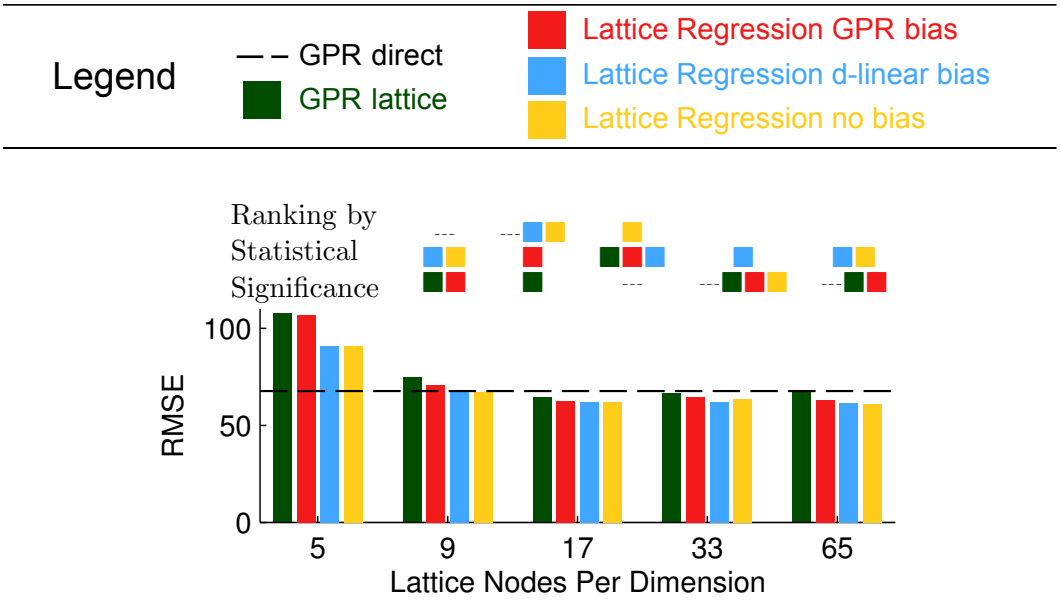


Figure 4.12: Shown is the RMSE of the estimates given by each method for the SIC97 test samples. The hierarchy of statistical significance is presented as in Fig. 4.11.

second experiment compares both color accuracy and color smoothness (see section 2.2) of three methods: lattice regression with the thin-plate regularizer $J_{\mathbf{K}}$, lattice regression with the Laplacian and global bias regularizers ($J_{\mathbf{L}}$,J_g), and local Tikhonov regression (a state-of-the-art technique previously analyzed in color smoothness and accuracy [28]).

Lattice Regression with $J_{\mathbf{L}}$ and J_g

The following printers were used for these experiments: an HP Photosmart D7260 ink jet printer and a Samsung CLP-300 laser printer. As a baseline, we compared to a state-of-the-art color regression technique used previously in this application [35]: local ridge regression (LRR) using the enclosing k-NN neighborhood (see section 3.2.2). Training samples were created by printing the Gretag MacBeth TC9.18 RGB image (Fig. 2.2), which consists of 918 color patches, of which 789 form a $9 \times 9 \times 9$ grid in the RGB space, and the remaining 189 samples are neutral ramps and bright saturated colors. We then measured the printed color patches with an X-Rite iSis spectrophotometer using D50 illuminant at a 2° observer angle and UV filter. As shown in Fig. 2.1 and as is standard practice for this application, the data for each printer is first gray-balanced using 1D calibration look-up-tables (1D LUTs) for each color channel (see [35, 5] for details). We use the same 1D LUTs for all the methods compared in the experiment and these were learned for each printer using direct GPR on the training data.

We tested each method's accuracy on reproducing 918 new randomly-chosen in-gamut[4] test Lab colors. The test errors for the regression methods the two printers are reported in Tables 1 and 2. As is common in color management, we report ΔE_{76} error, which is the Euclidean distance between the desired test Lab color and the Lab color that results from printing the estimated RGB output of the regression (see Fig. 2.1).

For both printers, the lattice regression methods performed best in terms of mean, median and 95 %-ile error. Additionally, according to a one-sided Wilcoxon test of statistical significance with $p = 0.05$, all of the lattice regressions (regardless of the choice of bias) were

[4]We drew 918 samples iid uniformly over the RGB cube, printed these, and measured the resulting Lab values; these Lab values were used as test samples. This is a standard approach to assuring that the test samples are Lab colors that are in the achievable color gamut of the printer [35].

Table 4.1: Samsung CLP-300 laser printer

	Euclidean Lab Error			
	Mean	Median	95 %-ile	Max
Local Ridge Regression (to fit lattice nodes)	4.59	4.10	9.80	14.59
GPR (direct)	4.54	4.22	9.33	17.36
GPR (to fit lattice nodes)	4.54	4.17	9.62	15.95
Lattice Regression (GPR bias)	**4.31**	**3.95**	**9.08**	**15.11**
Lattice Regression (Trilinear bias)	**4.14**	**3.75**	**8.39**	**15.59**
Lattice Regression (no bias)	**4.08**	**3.72**	**8.00**	**17.45**

Table 4.2: HP Photosmart D7260 inkjet printer

	Euclidean Lab Error			
	Mean	Median	95 %-ile	Max
Local Ridge Regression (to fit lattice nodes)	3.34	2.84	7.70	14.77
GPR (direct)	2.79	2.45	6.36	11.08
GPR (to fit lattice nodes)	2.76	2.36	6.36	11.79
Lattice Regression (GPR bias)	2.53	2.17	5.96	10.25
Lattice Regression (Trilinear bias)	2.34	1.84	5.89	12.48
Lattice Regression (no bias)	**2.07**	**1.75**	**4.89**	**10.51**

The bold face indicates that the individual errors are statistically significantly lower than the others as judged by a one-sided Wilcoxon significance test (p=0.05). Multiple bold lines indicate that there was no statistically significant difference in the bolded errors.

statistically significantly better than the other methods for both printers; on the Samsung, there was no significant difference between the choice of bias, and on the HP using the using no bias produced consistently lower errors. These results are surprising for three reasons. First, the two printers have rather different nonlinearities because the underlying physical mechanisms differ substantially (one is a laser printer and the other is an inkjet printer), so it is a nod towards the generality of the lattice regression that it performs best in both cases. Second, the lattice is used for computationally efficiency, and we were surprised to see it perform better than directly estimating the test samples using the function estimated with

GPR directly (no lattice). Third, we hypothesized (incorrectly) that better performance would result from using the more accurate global bias term formed by GPR than using the very coarse fit provided by the global trilinear bias or no bias at all.

Lattice Regression with $J_{\mathbf{K}}$

Since GPR and LRR with enclosing neighborhoods did not perform well in color accuracy, we omitted these techniques from the following experiments with color accuracy and color smoothness. Instead, accuracy and smoothness experiments were performed comparing the following three techniques: lattice regression with J_K regularization, lattice regression with $J_L + J_g$ regularization, and Tikhonov-regularized local linear regression with enclosing neighborhoods [35, 43] (see section 3.4.3).

Each of the three methods has a regularization parameter that must be trained to determine how much the regularizer is weighted compared to the empirical error. Choices for the cross-validation parameters were based on previously work and preliminary experiments, and are detailed in Table 4.3.

Table 4.3: Cross-validation Parameter Choices

	λ	λ_g
Lattice J_K	{1e-8,1e-7,...,1e0}	n/a
Lattice J_L+bias	{1e-6,1e-5,...,1e2}	× {1e-8,1e-7,...,1e0}
Tikhonov	{1e-4,1e-3,...,1e4}	n/a

All lattices were $17 \times 17 \times 17$ ranging from $L \in [0, 100]$, $a \in [-100, 100]$, and $b \in [-100, 100]$. Training data was produced for each printer by printing the Gretag MacBeth TC9.18 RGB target (Fig. 2.2). All prints were measured with an X-Rite iSis spectrophotometer, using D50 illuminant at a 2^o observer angle and UV filter.

All the 3D LUTS were preceded by the 1D gray-calibration LUTS for each color channel, as described in section 2.2.1. The 1D LUTS were the same for all experiments for each printer, and were estimated using Tikhonov-regularized regression method [35, 43] with a regularization parameter $\lambda = 1$.

For each printer, a randomly generated set of RGB values was printed and measured, forming an independent set of RGB $\rightarrow$ CIELAB pairs. This set provided a dual purpose in our experiments. The first role is as a test set; since the CIELAB values are guaranteed to be within the gamut of each printer, we use the set to assess the color accuracy of the constructed LUTs. That is, the LUT is applied to the CIELAB values, producing $\widehat{\text{RGB}}$ which is sent to the printer and measured, producing $\widehat{\text{CIELAB}}$ which can be compared in ΔE_{2000} to the original CIELAB values. Second, these set of values provided an independent set on which the RGB error of the LUT built by an algorithm could be compared across different parameter settings. Since cross-validation of the parameters for each algorithm in terms of the actual CIELAB error is intractable in a realistic workflow, the absolute RGB error was used as a proxy in order to find an appropriate parameter setting for the specific device. Table 4.3 shows the compared parameter settings and those that were chosen as optimal in terms of absolute RGB error are shown in Table 4.4.

Table 4.4: RGB-error Cross-validated Parameters

		λ	λ_g
Brother	Lattice (J_K)	1e-5	n/a
	Lattice (J_L+bias)	1e0	1e-2
	Tikhonov	1e2	n/a
HP	Lattice (J_K)	1e-5	n/a
	Lattice (J_L+bias)	1e0	1e-2
	Tikhonov	1e1	n/a

Starting with the parameters in Table 4.4, a range centered at these values was logarithmically swept by two values in each direction in order to compare how the smoothness and accuracy change with respect to the parameters. As described above, accuracy was measured in the ΔE_{2000} error between the desired and measured CIELAB values. Additionally, the smoothness of each algorithm was assessed by perceptual evaluation of a target consisting of circular ramps on a neutral ($L^* = 50, a^* = 0, b^* = 0$) background. Examples of the smoothness target are shown in Fig. 4.13, which shows that discontinuities in the color show up as circular artifacts. We found in previous experiments that such circular artifacts

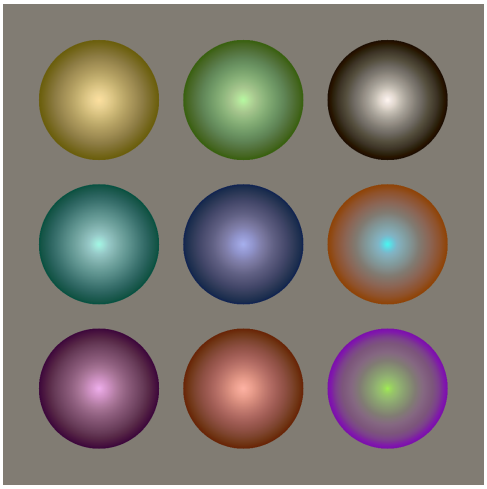
are generally more noticeable than the banded artifacts produced by analogous rectangular color ramps. All of the CIELAB ramps were adjusted to lie within the gamut of the intended printer and this gamut was estimated by the alpha-shape [14] (with $\alpha = 2000$) applied to the measured values from the TC9.18 chart for that printer. For seven of the ramps, a constant chroma was fixed at values lying at a constant radius in a^*b^* (including one at $a^* = 0, b^* = 0$) and the L^* value was swept within the range of the gamut at this chroma. The remaining two ramps were the lines (in $[L^*, a^*, b^*]$) {[100,-100,-100],. . . ,[0,100,100]} and {[100,-100,100],. . . ,[0,100,-100]} clipped to the gamut and these provide an example of smoothness in both lightness and hue.

Twenty adults with normal color vision subjectively ranked the smoothness of the printed targets. Each person ranked fifteen targets separately for the inkjet and laser printer, with ties not allowed. The lighting was produced by two 4700K Sollux lamps from Tailored Lighting approximating D50 illumination.

The ranked data was treated as pairwise preference information, and Thurstone's Law (Case V) was applied for analysis [22], producing a smoothness scale value for each print along with a increment Δ_s by which values can be judged significantly different.

In Fig. 4.14, the performance of each algorithm was evaluated in both smoothness and median ΔE_{2000} accuracy over a range of parameter settings. Here we see similar performance between the two regularization techniques for lattice regression with no clear winner between the two. However, we see that lattice regression outperforms local-linear Tikhonov regression, as the results for the latter are to the lower-right. Additionally, a comparison of the median and 95^{th}-percentile ΔE_{2000} errors and smoothness in terms of Δ_s, the smallest significant smoothness difference, is shown in Table 4.5 for the cross-validated parameter settings. Here, we see comparable errors but increased smoothness for lattice regression (vs. Tikhovnov) on the Brother 4040CDN laser printer. For the HP Photosmart D7260 inkjet printer, the errors are again comparable, and lattice regression with the $J_{\mathbf{L}} + J_g$ produces the smoothest result while the results produced by lattice regression with J_K and Tikhonov are not significantly different in smoothness.

LR with J_K HP Photosmart D7260 inkjet



LR with J_K Brother 4040CDN laser

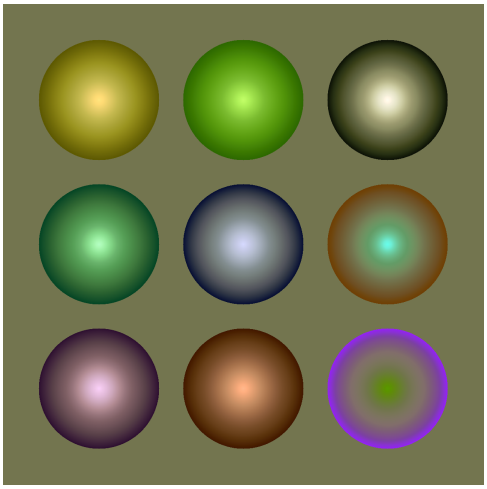


Figure 4.13: Figure shows examples of the color-managed prints that subjects ranked to evaluate smoothness. Since the displayed RGB values are intended to be printed by the respective printers, the actual smoothness and color is not reproduced accurately but, when treated as SRGB, the images do effectively illustrate the kind of contours seen in the prints.

4.3.4 Omni-directional Image Super-resolution

Omni-directional image super-resolution requires the ability to register (align) a number of unregistered low-resolution images prior to constructing a single high-resolution image from this data. In this section, we propose a novel optimization strategy that uses lattice regres-

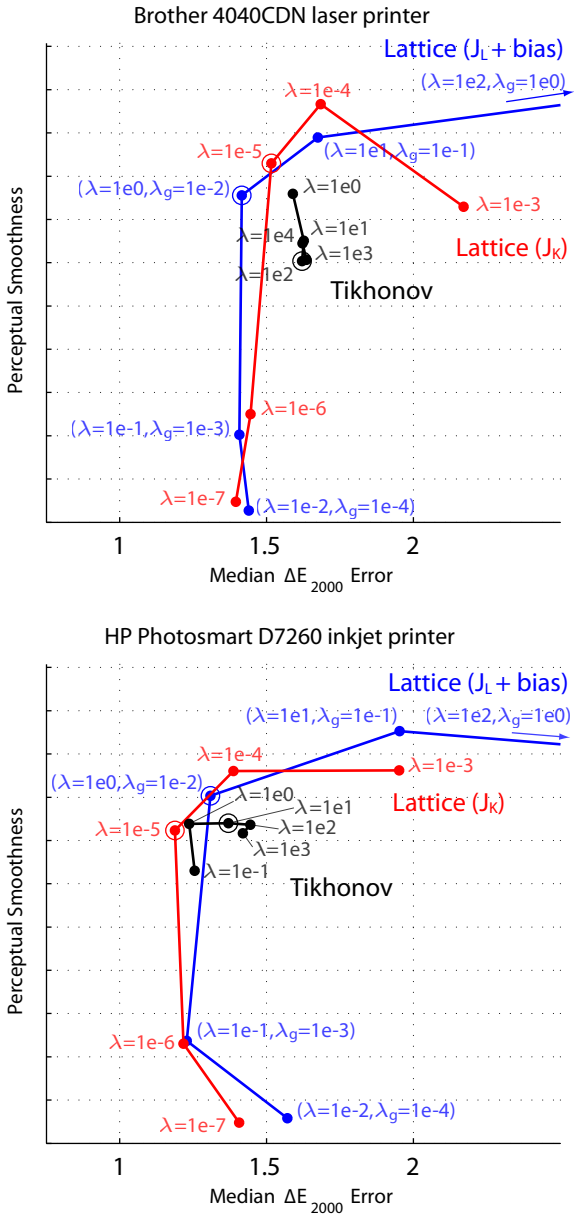


Figure 4.14: Figure shows the psychometrically scaled smoothness obtained from the ranking experiment plotted against median ΔE_{2000} error for varying algorithm parameters. The "best" algorithms will have values in the upper-left (high smoothness and low error). The data point indicated by the circle is the cross-validated algorithm setting. Legend: Red=LR (J_K), Blue=LR (J_L+bias), Black=Tikhonov.

Table 4.5: Cross-validated Algorithm Comparison

		ΔE^*_{2000} Error		
		Median	95%-ile	Smoothness*
Brother	Lattice (J_L+bias)	1.42	3.48	$4.03\,\Delta_s$
	Lattice (J_K)	1.52	3.92	$6.00\,\Delta_s$
	Tikhonov	1.62	3.37	$0.00\,\Delta_s$
HP	Lattice (J_L+bias)	1.31	2.53	$2.12\,\Delta_s$
	Lattice (J_K)	1.19	2.21	$0.00\,\Delta_s$
	Tikhonov	1.37	2.90	$0.44\,\Delta_s$

Table 4.6: *Note the smoothness scores should not be compared across printers, as the scores are based on relative assessments made for each printer separately.

sion at its heart to solve this registration problem. For notation, background information, and related work on this topic, please refer to section 2.4.

Like previous research [3, 40], our approach is to jointly minimize the interpolation error on the low-resolution images with respect to the registration parameters and the estimated high resolution image. This requires a global optimization, and in order to make even an approximate global optimization computationally feasible, the reconstruction must be fast to learn and especially fast to evaluate. Since lattice regression is designed to yield efficient and accurate low-dimensional regression, we adapt it in this application for learning images on the sphere. Lattice regression learns a lattice of values that can be interpolated to estimate the image intensity at any location on the sphere, and the lattice is chosen to minimize the regularized error when interpolating the low-resolution images under a putative set of registration parameters.

Proposed Greedy Registration

Our registration strategy is to learn the set of image registration parameters $g \in \Omega_g^N$ that lead to the best reconstruction of the low-resolution images given a model fit by the other $N-1$ low-resolution images.

Let $\mathbf{G}^{g_i} \in \Omega_{\theta\phi}^{M\times M}$ denote an arbitrarily rotated (by the Euler rotation $g_i \in \Omega_g$) spherical grid on which we will map the i^{th} image $\mathbf{Z}_i \in \mathbb{R}^{M\times M}$. Given a subset of the N indices of images $\mathcal{I} \subseteq \{1:N\}$, let $\mathcal{I}^C = \{1:N\}\backslash\mathcal{I}$ denote its complement. Suppose that one has trained an estimate of the underlying true spherical function using a set $\mathcal{I}$ of low-resolution images $\mathbf{Z}_\mathcal{I} = \{\mathbf{Z}_i \mid i \in \mathcal{I}\}$ under rotations $g'_\mathcal{I} = \{g'_i \mid i \in \mathcal{I}\}$; denote this estimate by $\hat{Z}(\,\cdot\,|\,\{\mathbf{G}^{g'_\mathcal{I}}, \mathbf{Z}_\mathcal{I}\}\,) : \Omega_{\theta\phi}^{M\times M} \rightarrow \mathbb{R}$, where $\hat{Z}$ is deterministically estimated via some regression technique given the input/output pairs $\{\mathbf{G}^{g'_\mathcal{I}}, \mathbf{Z}_\mathcal{I}\}$. Let

$$\hat{\mathbf{Z}}_{g_i|(g',\mathcal{I})} \triangleq \hat{Z}(\mathbf{G}^{g_i}|\{\mathbf{G}^{g'_\mathcal{I}}, \mathbf{Z}_\mathcal{I}\}) \tag{4.26}$$

be the reconstructed low-resolution image at $\mathbf{G}^{g_i}$.

We propose estimating the registration parameters $\hat{g}$ that minimize the leave-one-out reconstruction error over all N images. That is,

$$\hat{g} = \underset{g\in\Omega_g^N}{\arg\min} \sum_{i=1}^{N} \left\|\mathbf{Z}_i - \hat{\mathbf{Z}}_{g_i|(g,i^C)}\right\|_F. \tag{4.27}$$

This empirical risk minimization approach to learning the registration angles will work with any method to estimate $\hat{Z}$, though in this paper we focus on the lattice regression estimate because it enables fast creation and evaluation of $\hat{Z}$, which makes optimizing (4.27) more computationally feasible.

The registration cost function given in (4.27) has many local minima and requires searching a $3(N-1)$ dimensional space. While full global optimization of (4.27) would be ideal, the dimension of the search space is too high to confidently find a quality local optima in finite time with present-day computational power. Thus, we present an efficient (and approximate) greedy method for breaking the optimization into smaller problems:

1) Initialize: Let the iteration index $t = 1$ and let the training image set $\mathcal{I}_0 = 1$ be initialized with the first image $\mathbf{Z}_1$ and initialize the registration parameter $\hat{g}_1 = (0, 0, 0)$.

2) Register: For each image not in the current training set, find the registration parameter $\hat{g}_i$ that minimizes the reconstruction of image $\mathbf{Z}_i$ as learned from the current training image set $\mathcal{I}_{t-1}$ with previously estimated rotations $\hat{g}_{\mathcal{I}_{t-1}}$. That is, for $i \in \mathcal{I}_{t-1}^C$ set

$$\hat{g}_i = \underset{g_i \in \Omega_g}{\arg\min} \quad \left\| \mathbf{Z}_i - \hat{\mathbf{Z}}_{g_i|(\hat{g}, \mathcal{I}_{t-1})} \right\|_F. \tag{4.28}$$

3) Add an Image: Add the image that was reconstructed with the least error in Step 2 to the training set. That is, set $\mathcal{I}_t = \mathcal{I}_{t-1} \bigcup i^*$ where

$$i^* = \underset{i \in \mathcal{I}_{t-1}^C}{\arg\min} \quad \left\| \mathbf{Z}_i - \hat{\mathbf{Z}}_{\hat{g}_i|(\hat{g}, \mathcal{I}_{t-1})} \right\|_F. \tag{4.29}$$

4) Swap an Image: If there exists an image in $\mathcal{I}_t$ that *at the time it was added to the training set* had a larger reconstruction error than the image with the lowest reconstruction error in $\mathcal{I}_t^C$, swap these two images. That is, if $j \in \mathcal{I}_t$ has a larger reconstruction error than $k \in \mathcal{I}_t^C$, let $\mathcal{I}_t = \left(\mathcal{I}_t \backslash j\right) \bigcup k$.

5) Repeat: Augment the iteration counter: $t = t + 1$. Go to Step 2 until $\mathcal{I}_t = \{1 : N\}$.

Note that, the greedy registration algorithm described above takes $N - 1$ iterations to complete. To improve results, it can be run multiple times by using a subset of the previously registered images as the seeding set $\mathcal{I}_0$. In the experiments, we perform a second pass, setting $\mathcal{I}_0$ to the set of images with reconstruction error lower than the median; this heuristic improved the resulting registration in most cases. If the global optimization of (4.29) is allowed V guesses, then there will be $O(VM^2N^2)$ total evaluations of the reconstruction function. Thus, it is helpful if evaluating the estimate is efficient, as is the case with lattice regression. Still, the above greedy registration can be too slow. To speed it up, we keep track of the reconstruction error of the last image added to the training set. If *at any time* an image has a lower reconstruction error than that of the last image added,

it is immediately added to the training set and this threshold is updated; the order in which the images in $\mathcal{I}_{t-1}^{C}$ are traversed in Step 2 is randomized on each iteration.

To perform each global optimization in (4.29), we used a state-of-the-art metaheuristic termed *fully-informed particle swarm* (FIPS) [52], which is a variant of the original particle swarm optimizer (PSO) [21]. PSO begins with a number of random guesses (usually, equal to the number of search dimensions squared) called birds. At each iteration, the optimizer moves each bird to a new point in the domain, where each bird is moved in part towards the best location it has ever seen, in part towards the best location any of the birds has ever seen, and its movement has an inertial term that encourages exploration. FIPS imposes further connections between the birds movements; we used the connecting ring neighborhood variant.

Experimental Details

In this section we show that our method outperforms the state-of-the-art method [3] in terms of the resulting PSNR for the super-resolved image as well as in terms of the computational requirements and scalability of method to larger problems. Tested are two real omni-directional images (image '*graves*' shown in Fig. 4.18(a) and image '*street*' shown in Fig. 4.19(a)) from a public domain image database [48] and one synthetically generated omni-directional image (image '*room*' shown in Fig. 4.20(a) which was the image used in reporting results with the benchmark method in [1, 2]). The low-resolution images were generated by downsampling in the SFT domain (as done in [1, 2, 3]) for all experimental results except results in Figs. 4.18, 4.19, and 4.20 which for computational feasibility were downsampled using the Matlab in-built function `TriScatteredInterp` with natural neighbor interpolation. The low resolution images were captured on uniformly random rotations of the canonical grid G_0. The reconstruction algorithms are given the low-resolution images and the grids they were captured on within a registration error of $[-\psi^\circ,\psi^\circ]$ where the maximum registration error ψ is known. The setting $\psi = 90^\circ$ corresponds to complete ignorance of the orientations.

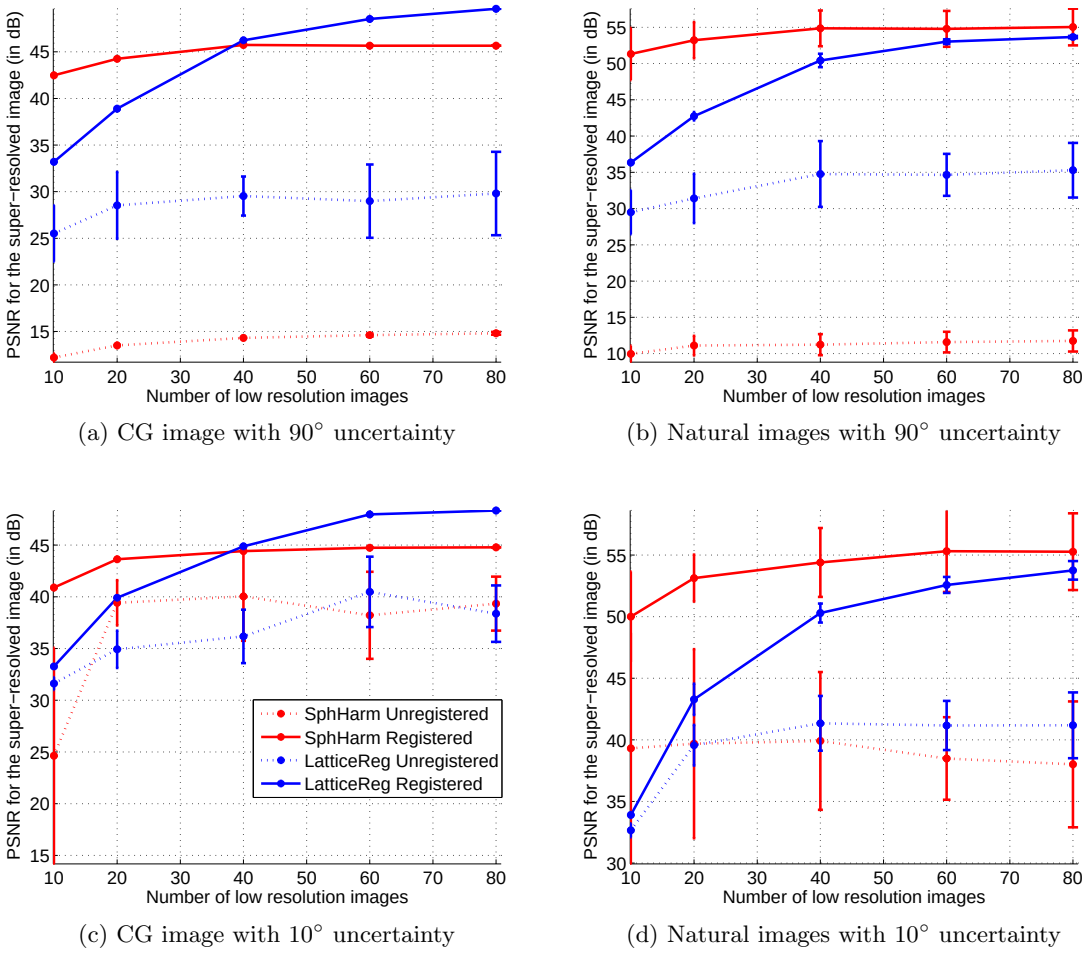
The experiments were run for three different settings (1) reconstructing a (64×64) image from (16×16) images - this was the only setting we could extensively collect results for because the benchmark [3] computation time explodes exponentially with the number of low-res images and the size of the high-res image, (2) reconstructing a (128×128) image from (16×16) images and (3) reconstructing a (256×256) image from (64×64) images. The results are given below Figs. 1-3 in terms of PSNR values and runtime as a function of the number of low resolution images and degree of registration uncertainty. Each set of experiments reported in this paper was run with 10 random realizations of unknown registration parameters.

Lattice regression was run for the reconstruction of a 64×64 high-resolution image from 16×16 low-resolution images, using a lattice of size 128×128 as an underlying representation. A 128×128 lattice was also used for the reconstruction of a 128×128 high-resolution image using a set of 16×16 low-resolution images. For the reconstruction of a 256×256 high-resolution image from 64×64 images, a 256×256 lattice was used.

Results

Fig. 4.15 shows the PSNR of reconstructing a high-resolution image from multiple low-resolution images that were downsampled at arbitrarily rotated grids. In the left column are synthetic results (‘*room*’) and the right column is the average of the real-image results (‘*street*’ and ‘*graves*’). Note that the solid lines in these plots are the reconstruction given perfect registration and the dotted lines are the reconstruction given the registration estimated by each algorithm. Fig. 4.15(a,b) show that the proposed method outperforms the benchmark by an order of magnitude (15 to 20 dB) when the registration is completely unknown for reconstructing a (64×64) image from (16×16) images. If the registration is known within 10°, Figs. 4.15(c,d) show that the two methods are comparable in performance.

For reconstructing a (128×128) image from (16×16) images with large degree of registration uncertainty, Figs. 4.16(a,b) show that the proposed method performs much better than the benchmark. Our approach is consistent across the number of low-resolution images used and provides a 20 dB gain over the benchmark.



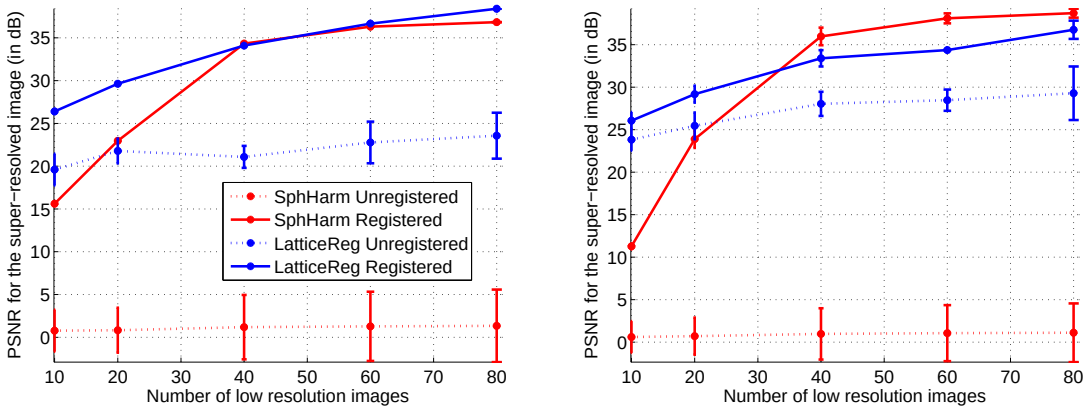
(a) CG image with 90° uncertainty

(b) Natural images with 90° uncertainty

(c) CG image with 10° uncertainty

(d) Natural images with 10° uncertainty

Figure 4.15: Reconstruction PSNR (in dB) for (64×64) image vs number of low-resolution (16×16) images. The registration uncertainty in Euler angle parameters for each rotated image was 5°. Results with (a,c) '*room*', (b,d) '*street*' and '*graves*'. The error bars represent the standard deviation of the resulting PSNR values.



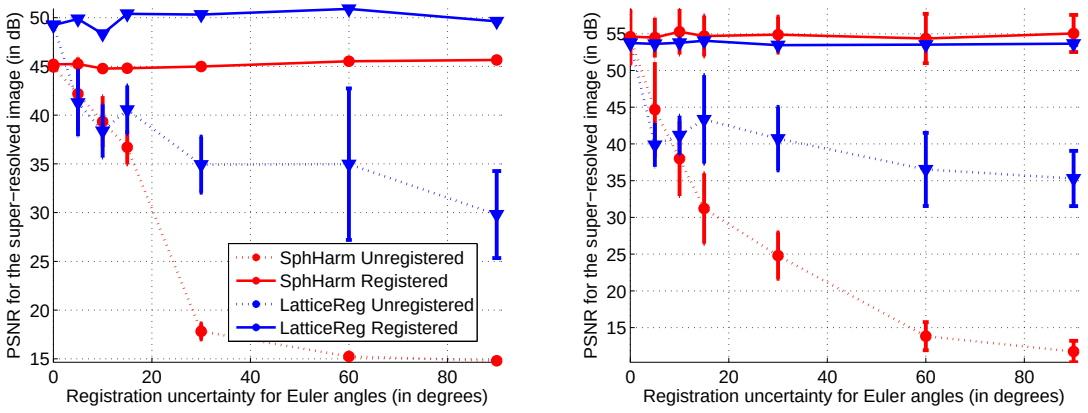
(a) CG image with 90° uncertainty

(b) Natural images with 90° uncertainty

Figure 4.16: Reconstruction PSNR of (128×128) image vs number of low-resolution (16×16) images of real images '*graves*', '*street*'. The registration uncertainty was 90°. Results with image (a) '*room*', (b) '*street*' and '*graves*'

To illuminate another angle on performance we show, for a fixed number of (80) low-resolution images, the performance of the method over varying registration uncertainties. Fig. 4.17 illustrates that the proposed approach outperforms the benchmark by a margin of $15-20$ dBs in terms of PSNR when reconstructing (64×64) image from (16×16) images for conditions with large degree of registration uncertainty while, as we saw before, for the smaller amount of uncertainty, the methods are comparable.

Shifting now to computational efficiency, Table 4.7 lists runtime (in mins) for registration and reconstruction of omni-directional images for various settings. All the experiments reported in this paper were run on an Intel quad-core 3.20GHz machine with 12 Gigabyte memory. It is evident from the table that the benchmark method [3] does not scale efficiently to larger resolutions. The computation of the fast spherical transform requires $\mathcal{O}(B^2(\log(B))^2)$ operations where B is typically chosen to correspond with the size of the high-resolution image that needs to be reconstructed. Furthermore, this operation needs to be repeated several times at each iteration of the gradient descent approach [1]. This polynomial increase in computation time with the size of the reconstructed image severely limits the scalability to image sizes larger than 128×128. Our approach on the other hand is more computationally efficient and allows for scaling-up without significant increase in



(a) CG image (b) Natural images

Figure 4.17: Reconstruction PSNR (in dB) of (64×64) image from 80 low-resolution (16×16) images vs degree of uncertainty. Results with images (a) '*room*', (b) '*street*' and '*graves*'.

Table 4.7: Comparing runtime-per-super-resolved-image (in mins) for Lattice regression with the benchmark [3]

Number of Images	10	20	40	60	80
Benchmark (128×128)	13.4113	24.8804	40.8226	49.3601	70.8547
Lattice Regression (128×128)	0.0673	0.1785	0.5720	1.3148	3.4343
Lattice Regression (256×256)	0.1588	0.4287	1.1683	2.5282	5.6448

computational requirements. We show reconstruction results with lattice regression for images of size (256×256) from 80 (64×64) images for image '*graves*' in Fig. 4.18, for '*street*' in Fig. 4.19 and image '*monkey*' in Fig. 4.20. These conditions were computationally infeasible with the benchmark due to memory and runtime constraints.

4.4 Conclusions

In this chapter we have presented *lattice regression*, a method for estimating the outputs of a linearly-interpolated look-up table (lattice) from a set of training examples. The approach we adopt is to minimize the *regularized* training error of the look-up table, directly taking into account the effect of interpolation from the nodes. We hypothesized at the outset that such an approach would outperform the naive approach of estimating a function $\hat{f}$ independent of the look-up table and predicting the look-up table node outputs from $\hat{f}$.

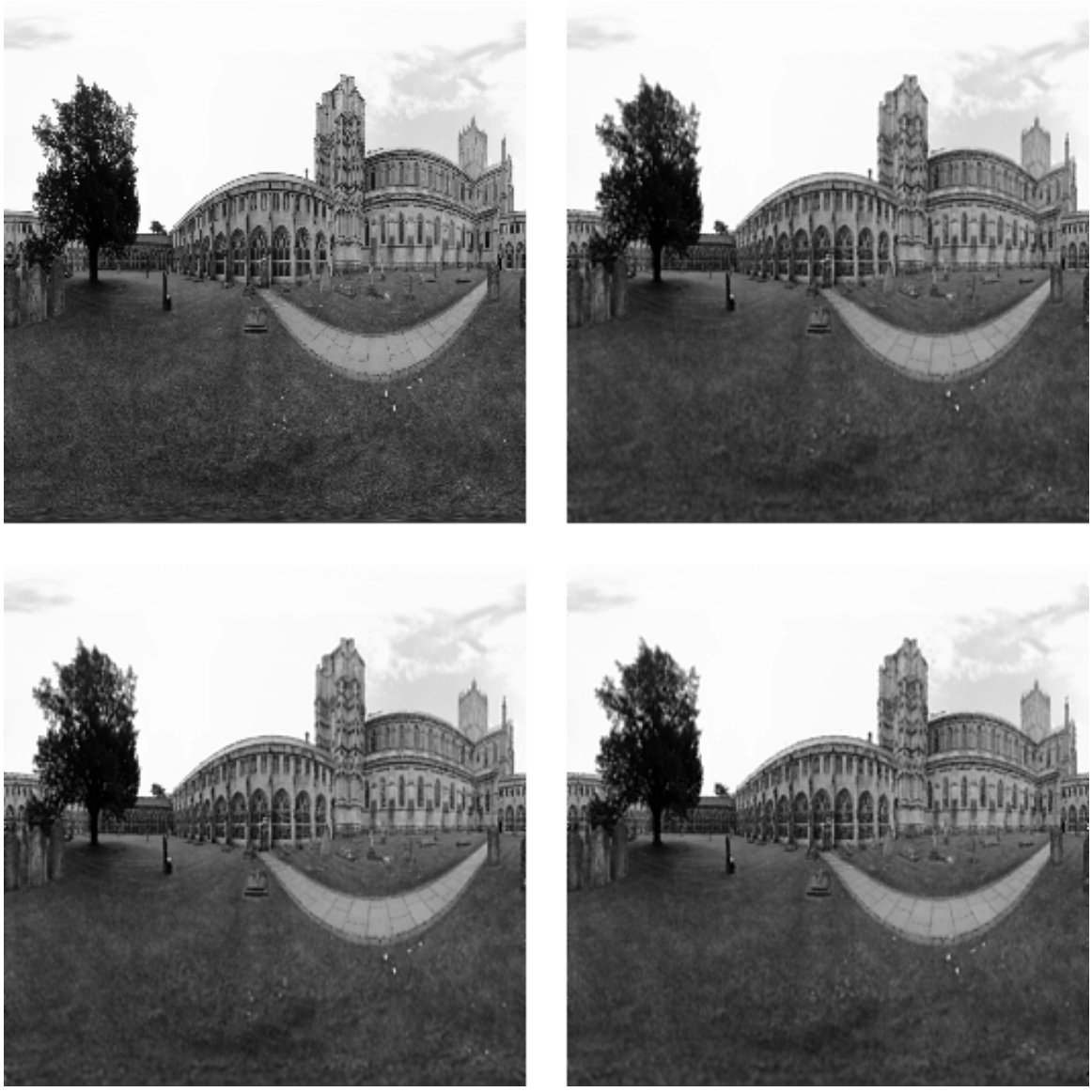


Figure 4.18: Reconstructing (256 × 256) omni-directional image from 80 (64 × 64) low resolution omni-directional images for the real image '*graves*': (a) Top-Left: Original (256 × 256) image, (b) Top-Right: reconstruction from perfectly registered images (resulting PSNR = 26.8021 dB), (c) Bottom-Left: reconstruction from unregistered images with registration uncertainty of 5° (resulting PSNR = 26.4469 dB), (d) Bottom-Right: reconstruction from unregistered images with registration uncertainty of 90° (resulting PSNR = 26.1720 dB).

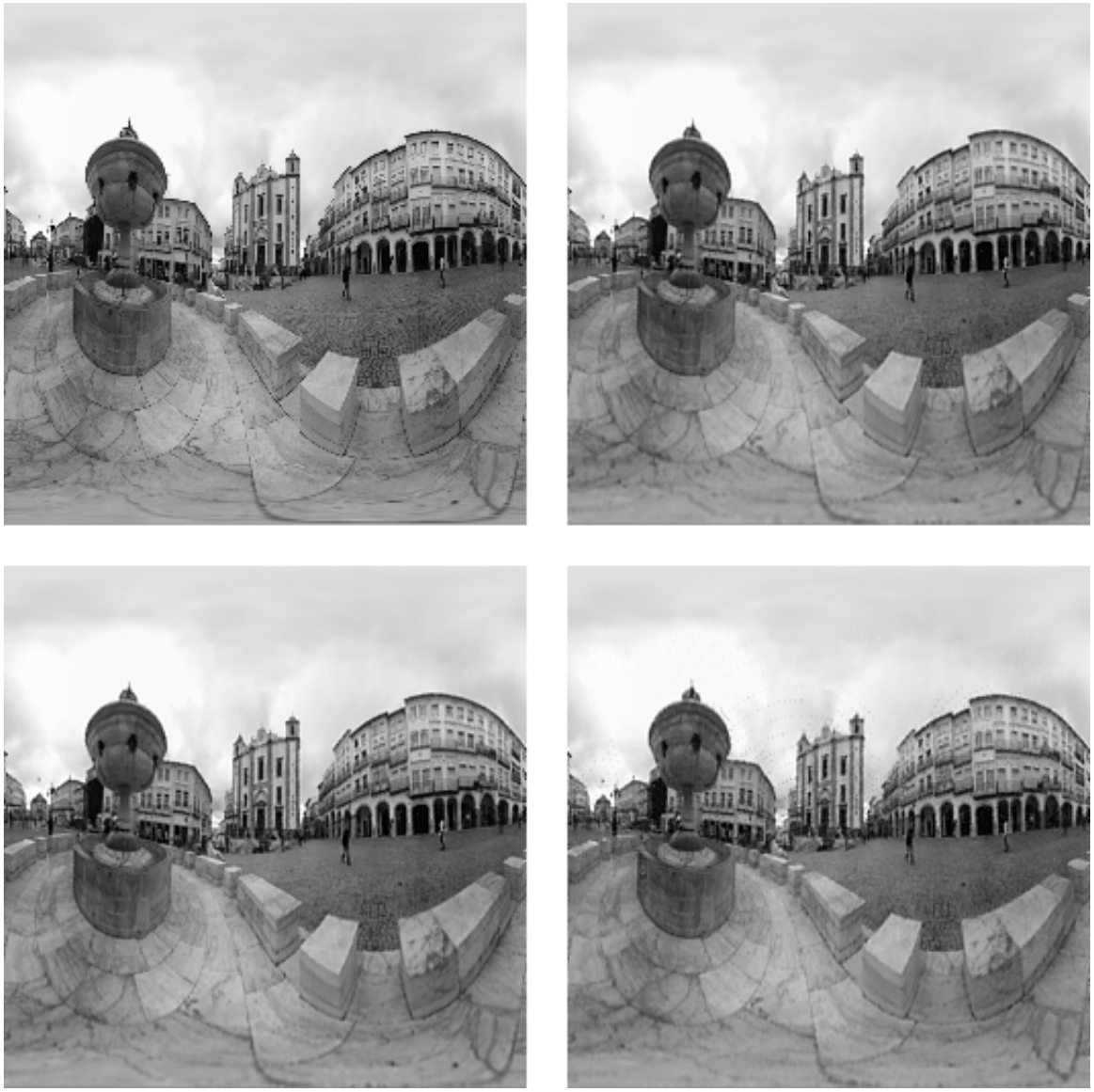


Figure 4.19: Reconstructing (256 × 256) omni-directional image from 80 (64 × 64) low resolution omni-directional images for the real image '*street*': (a) Original (256 × 256) image, (b) reconstruction from perfectly registered images (resulting PSNR = 27.8758 dB), (c) reconstruction from unregistered images with registration uncertainty of 5° (resulting PSNR = 27.3273 dB), (d) reconstruction from unregistered images with registration uncertainty of 90° (resulting PSNR = 24.2951 dB).

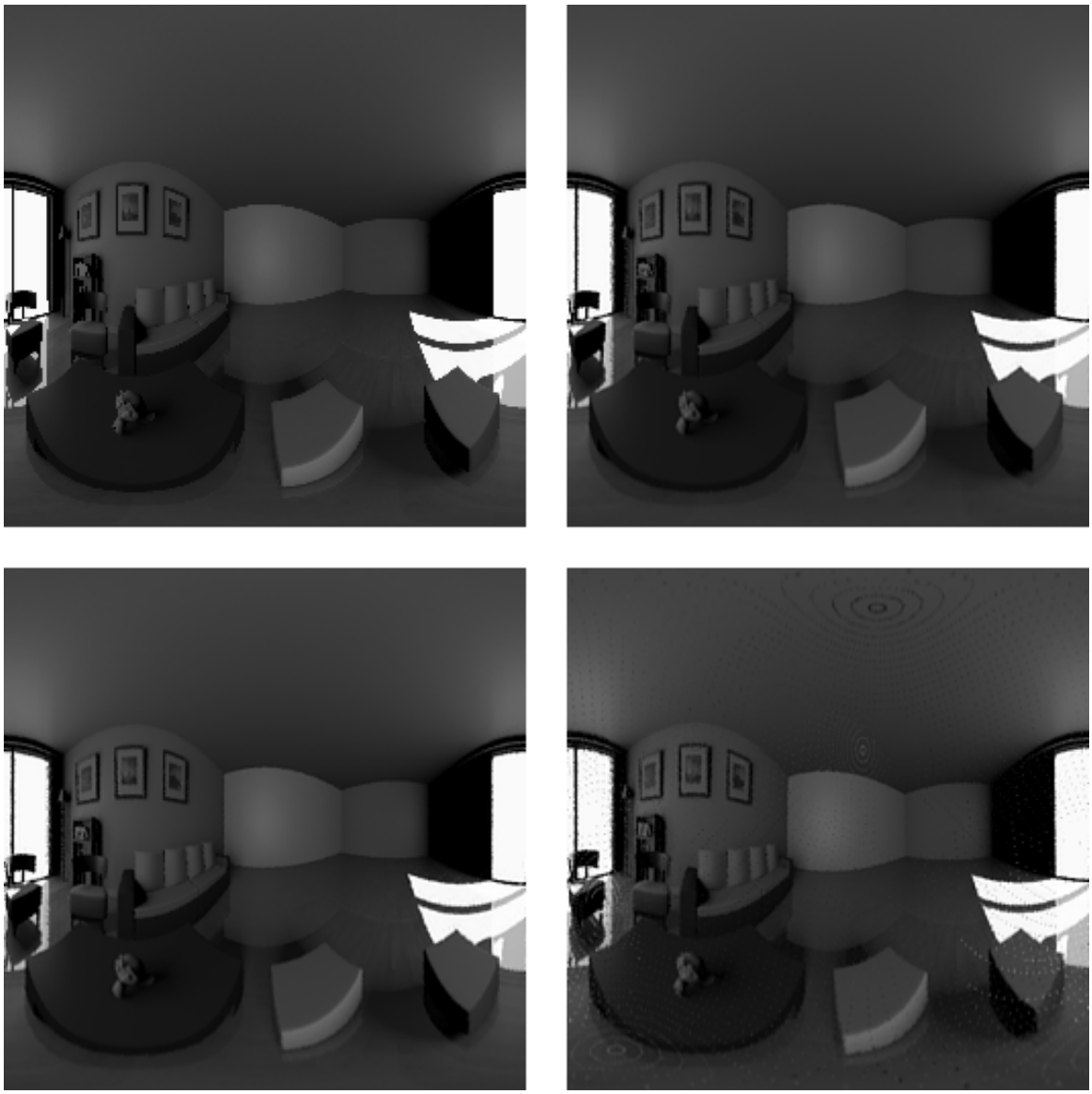


Figure 4.20: Reconstructing (256 × 256) image from 80 (64 × 64) images for the image '*room*': (a) Original (256 × 256) image, (b) reconstruction from perfectly registered images (resulting PSNR = 29.0318 dB), (c) reconstruction from unregistered images with registration uncertainty of 5° (resulting PSNR = 27.6772 dB), (d) reconstruction from unregistered images with registration uncertainty of 90° (resulting PSNR = 22.8049 dB).

This hypothesis was tested on a wide variety of applications leading to some expected but also a few counter-intuitive results.

First, we have observed that when applying lattice regression with Laplacian and global bias regularization – where the global bias f is the result of a sophisticated regression estimate (such as GPR) – lattice regression outperforms the naive approach of setting the nodes look-up table by simply evaluating the same f. This was an expected result which was validated in the simulated data of section 4.3.1, the geospatial interpolation application of section 4.3.2, and the color management application in section 4.3.3. In this context, lattice regression amounts to *tweaking* an existing estimate to be smoother (in a first-order sense) and to more closely match the training data. This could be appropriate when one has an existing look-up table that must be adapted to new data or when one has some side information about the underlying function.

In a few instances, we also observed that lattice regression with Laplacian and global bias regularization actually outperforms a regression that sidesteps using a look-up table at all. This is a surprising result as lattice regression was designed to overcome the inherent limitations of discretizing the function to a lattice (which is appropriate only when one is interested in optimizing test-evaluation performance). Offhand, one would expect to do better with a regression that is unencumbered by the lattice, but we see in the geospatial interpolation and color management applications that lattice regression using no bias at all actually outperforms the application of GPR directly to the data. This behavior is observed only at the large lattice sizes and is likely due to the ability of lattice regression to fit varying amounts of roughness in the estimate throughout the domain while GPR attempts to fit one length-scale to the entire domain, over-smoothing in some areas in order to fit others.

The second-order thin-plate regularization was tested in the color management experiments of section 4.3.3 and in the omni-directional super-resolution problem of section 4.3.4. In this set of color management experiments, we found no significant difference in color accuracy or color smoothness when comparing lattice regression with the two types of regularization: thin-plate vs. Laplacian + global trilinear bias. However, the thin plate regularization provides a more concise solution and fewer regularization parameters to train. In the omni-directional super-resolution problem, we demonstrate an application that takes

advantage of the test-evaluation speed of lattice regression in an interesting way. By using the lattice as an intermediate function representation in the inner loop of a global optimization procedure, one continually tests the accuracy of a series of possible registrations of a single image against this model. Although the metric for evaluation in this application confounds the registration process with the accuracy of the regressed estimate (the final super-resolved image), the technique involving lattice regression is shown to have higher accuracy at a lower computational cost than state-of-the-art for this application.

Overall, the proposed lattice regression performed quite well in comparison to other standard techniques in the tested applications. Compared to the standard approach of estimating a look-up table, lattice regression optimizes the *correct* training objective: the regularized error of the *estimated function* with respect to the training data. Further, we have explored a few approaches to regularization, with the thin-plate approach being the most theoretically sound in terms of characterizing roughness. For practitioners, we suggest using lattice regression with the thin-plate regularizer unless one has some side information about the underlying function to be estimated or an existing estimate one would like to update. In these cases, one might attempt the combination of Laplacian and global bias regularizers.

Chapter 5

EXTENSIONS AND CONCLUSIONS

The contributions of this thesis are summarized in section 5.1. Section 5.2 provides a discussion of the limitations of the proposed topics as well as recommendations for how they might be applied. Finally, the thesis is concluded with a discussion of future work in section 5.3

5.1 Contributions

In this thesis, we have presented two new concepts in regression that – due to computational considerations – each happen to be applicable only in low-dimensional domains. First, we have proven that *enclosing neighborhoods* for local linear regression provide estimates with bounded variance and proposed the *enclosing kNN* neighborhood as the smallest (and thus lowest bias) radial neighborhood exhibiting this property along with an algorithm for its construction. Second, we have presented a new technique, *lattice regression*, for estimating look-up tables (suitable for applications where fast test throughput is required) with the proper training objective (minimizing the training error of the *overall* estimated function). Although each contribution was developed with estimation of color transformations in mind, we have shown their applicability to a wide variety of low-dimensional problems as well.

5.2 Limitations

As mentioned at the outset, the proposed methods are applicable only in low-dimensional settings due to computational restrictions. In each case, the restrictions emerge from the so-called *curse of dimensionality*, a general purpose term for the various unfortunate side-effects caused by the exponential increase in volume of a space with its dimension. The nature of this *curse* is best demonstrated by comparing the volume of the unit sphere with that of the unit hypercube with respect to dimension. For an even dimension d, the volume

of the unit sphere is $V_s = \frac{\pi^{d/2}}{(d/2)!}$ while that of the unit hypercube is $V_c = 2^d$ [39]. The ratio of the two volumes is

$$\frac{V_s}{V_c} = \frac{(\pi/4)^{d/2}}{(d/2)!},$$

this is plotted against dimension in Fig. 5.1 One important consequence of this fact is that

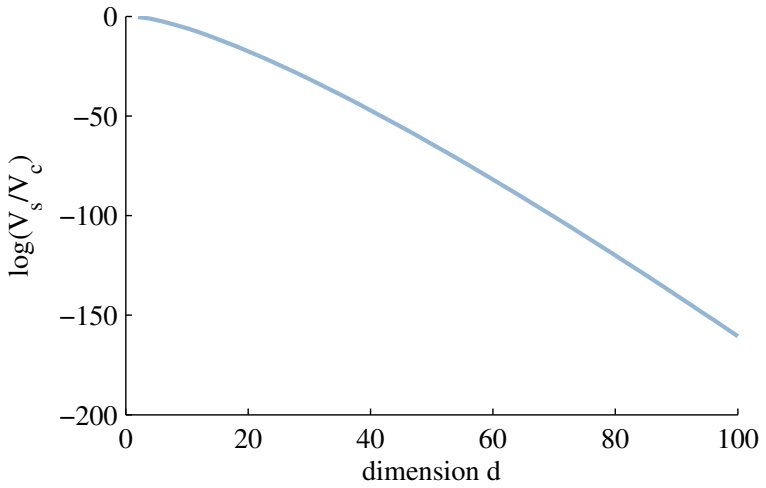


Figure 5.1: The ratio of the volume of the unit sphere V_s to that of the unit hypercube V_c vs dimension. Plotted on a logarithmic scale.

points in high dimensional space become far from one another; this presents a problem for learning.

For the enclosing neighborhoods, one is relying on there being some subset of the training data that indeed encloses the test point within its convex hull. This objective is stifled in high dimensions because for most distributions of data,[1] the likelihood of finding a test point inside the convex hull of a fixed-size training set decreases exponentially with dimension. This problem could be mitigated by exponentially increasing the size of the training set for higher dimensional problems, but in many applications this is simply not possible. Furthermore, increasing the size of the training set increases the amount of time required to seek out the nearest neighbors which adds to the computational burden.

[1] All but pathological cases such as finding training points at the corners of a hypercube.

For lattice regression, the detrimental effects of dimension are a bit more straightforward. Lattice regression requires one to construct a lattice in the domain of interest, but the number of points m required to construct such a lattice will grow exponentially with dimension. Since the solution to lattice regression is requires the inversion of a (sparse) $m \times m$ matrix, the size of this problem will grow exponentially with dimension. This can be mitigated somewhat by reducing the number of lattice nodes in each dimension, but this detracts from the flexibility of the estimated function. In the limit of $m = 2^d$, this reduces to fitting a global trilinear function, but even this approach quickly becomes intractable as by $d = 20$ dimensions, this very rough fit will require more than $m > 10^6$ nodes.

5.3 Future Work

The bounded estimation variance result for regression on an enclosing neighborhood is promising, but the result only holds for iid noise and un-regularized linear regression. Similar results might be shown in the case of non-iid noise, Tikhonov or ridge regression, and non-linear interpolation. Additionally, given some smoothness quantification (such as a representation that is a truncated Taylor series) one might be able to quantify the trade-off between linear extrapolation from a small neighborhood and linear interpolation from a larger neighborhood. Such a result would attempt to formalize the intuition that the accuracy of linear extrapolation degrades as functions become increasingly non-linear.

From a computational standpoint, local linear regression is encumbered by exhaustively searching for near-neighbors in a large training set. For general learning problems, this burden is lifted somewhat by seeking only *approximate* neighbors from an optimized data structure known as a k-d tree [11]. The searching algorithm used in kd-trees is markedly similar to the algorithm used to construct the enclosing kNN in that it uses rules out entire half-spaces of possible neighbors at a time. The major difference is that the half-spaces in the kd-tree search are axis-aligned whereas those in the enclosing kNN algorithm are unconstrained. Thus, a fast approximation to the enclosing kNN neighborhood could be easily built in to the kd-tree search. The efficiency gained from this approximation would allow local linear regression on enclosing neighborhoods to be run on larger data sets.

The bottleneck for computation in lattice regression is the inversion of a sparse $m \times m$ matrix where m is the total number of nodes in the desired look-up table. A promising approach to improving both the performance and parallelization ability of lattice regression is to decouple the lattice into sub-lattices. That is, to partition the lattice along each dimension forming many small lattices on which one may apply lattice regression. The delicate part to this approach lies in how one 'stitches' the multiple lattices at the seams in some optimal way. A promising approach to this open question is to overlap each sub-lattice by a full cell and to average the overlapping node outputs of this cell although one can imagine various iterative approaches as well.

BIBLIOGRAPHY

[1] Zafer Arican and Pascal Frossard. l_1 regularized super-resolution from unregistered omnidirectional images. In *Proc. Int. Conf. Acoust. Speech and Sig. Proc.*, 2008.

[2] Zafer Arican and Pascal Frossard. Super-resolution from unregistered omnidirectional images. In *Proc. Int. Conf. Pattern Recog.*, 2008.

[3] Zafer Arican and Pascal Frossard. Joint registration and super-resolution with omnidirectional images. `http://lts4www.epfl.ch/~frossard/publications/pdfs/SSROmni-TR.pdf`, 2009. Technical report TR-LTS-2009-014. Ecole Polytechnique Federale de Lausanne (EPFL), Submitted for journal publication to IEEE Trans. Img. Proc.

[4] F. Aurenhammer. Voronoi diagrams-a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23:345–405, 1991.

[5] R. Bala. *Digital Color Handbook*, chapter 5: Device Characterization, pages 269–384. CRC Press, 2003.

[6] R. Bala and R. V. Klassen. *Digital Color Handbook*, chapter 11: Efficient Color Transformation Implementation. CRC Press, 2003.

[7] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[8] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal Machine Learning Research*, pages 2399–2434, 2006.

[9] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2006.

[10] L. Cazzanti and M. R. Gupta. Local similarity discriminant analysis. In *Proc. of the 24th Intl. Conf. on Machine Learning*, 2007.

[11] S. Chandran. Introduction to kd-trees. Technical report, University of Maryland Department of Computer Science, 2002.

[12] B. B. Chaudhuri. A new definition of a neighbourhood of a point in multi-dimensional space. *Pattern Recognition Letters*, pages 11–17, 1996.

[13] Gregory S. Chirikjian and Alexander B. Kyatkin. *Engineering Applications of Noncommutative Harmonic Analysis*. CRC, 2000.

[14] T. J. Cholewo and S. Love. Gamut boundary determination using alpha-shapes. *Proc. 7th IS&T Color Imaging Conference*, 1999.

[15] F. Chung. *Spectral Graph Theory*. Number 92 in Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[16] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. *Proc. Intl. Conf. on Machine Learning*, 2007.

[17] L. Devroye, L. Gyorfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag Inc., New York, 1996.

[18] C. Domeniconi, J. Peng, and D. Gunopulos. Locally adaptive metric nearest neighbor classification. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(9):1281–1285, 2002.

[19] J. Driscoll and D. Healy. Computing fourier transforms and convolutions on the 2-sphere. *Advances in Applied Mathematics*, 15:202–250, 1994.

[20] G. Dubois. Spatial interpolation comparison 1997: Foreword and introduction. *Special Issue of the Journal of Geographic Information and Descision Analysis*, 2:1–10, 1998.

[21] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Sixth Int. Symposium on Micro Machine and Human Science*, pages 39–43. IEEE, October 1995.

[22] P. Engeldrum. *Psychometric Scaling*, chapter chapter 8: Indirect Interval Scaling - Case V and Paired Comparison, pages 93–108. Imcotek Press, 2000.

[23] S. Farsiu, D. Robinson, M. Elad, and P. Milanfar. Advances and challenges in super-resolution. *Int. J. of Imaging Sys. and Tech.*, 14:47–57, 2004.

[24] B. Fraser, C. Murphy, and F. Bunting. *Real World Color Management*. Peachpit Press, Berkeley, CA, 2003.

[25] J. Friedman. Flexible metric nearest neighbor classification. *Technical Report, Stanford University, CA*, 1994.

[26] K. Fukunaga and T. Flick. An optimal global nearest neighbor metric. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 6:314–318, 1984.

[27] K. Fukunaga and L. Hostetler. Optimization of k-nearest neighbor density estimates. *IEEE Trans. on Information Theory*, 19:320–326, 1973.

[28] E. K. Garcia and M. R. Gupta. Building accurate and smooth icc profiles by lattice regression. *Proc. 17th IS&T Color Imaging Conference*, pages 101–106, 2009.

[29] E. K. Garcia and M. R. Gupta. Lattice regression. *Advances in Neural Information Processing Systems (NIPS)*, pages 1–9, 2009.

[30] E. K. Garcia and M. R. Gupta. Optimized construction of icc profiles by lattice regression. *Proc. 18th IS&T Color Imaging Conference*, 2010.

[31] K. C. Gowda and G. Krishna. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. *IEEE Trans. on Information Theory*, 25(4):488–490, 1979.

[32] R. L. Graham, D. E. Knuth, and O. Pastashnik. *Concrete Mathematics*. Addison-Wesley, New York, 1989.

[33] P.J. Green and B.W. Silverman. *Nonparametric Regression and Generalized Linear Models: A roughness penalty approach*. Chapman & Hall, 1994.

[34] M. R. Gupta and R. Bala. Personal communication with Raja Bala, June 21 2006.

[35] M. R. Gupta, E. K. Garcia, and E. M. Chin. Adaptive local linear regression with application to printer color management. *IEEE Trans. on Image Processing*, 17(6):936–945, 2008.

[36] M. R. Gupta, R. M. Gray, and R. A. Olshen. Nonparametric supervised learning by linear interpolation with maximum entropy. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 28(5):766–781, 2006.

[37] T. Hastie and C. Loader. Local regression: automatic kernel carpentry. *Statistical Science*, 8(2):120–143, 1993.

[38] T. Hastie and R. Tibshirani. Discriminative adaptive nearest neighbour classification. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(6):607–615, 1996.

[39] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, New York, 2001.

[40] Y. He, K. Yap, L. Chen, and L. Chau. A nonlinear least square technique for simultaneous image registration and super-resolution. *IEEE Trans. Image Proc.*, 16:2830–2841, 2007.

[41] A. E. Hoerl and R. Kennard. Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

[42] R. Howard and P. Sisson. Capturing the origin with random points: Generalizations of a Putnam problem. *College Mathematics Journal*, 27(3):186–192, May 1996.

[43] N. Hrustemovic and M. R. Gupta. Multiresolutional regularization of local linear regression over adaptive neighborhoods for color management. In *Proc. of the Intl. Conf. on Image Processing*, 2008.

[44] H. Kang. *Color Technology for Electronic Imaging Devices*. SPIE Press, United States of America, 1997.

[45] H. Kawasaki, K. Ikeuchi, and M. Sakauchi. Super- resolution omnidirectional camera images using spatio-temporal analysis. *Electronics and Communications in Japan Part 3 Fundamental Electronic Science*, 89:47–59, 2006.

[46] F. Li, J. Yang, and J. Wang. A transductive framework of distance metric learning by spectral dimensionality reduction. *Proc. Intl. Conf. on Machine Learning*, 2007.

[47] Shigang Li. Full-view spherical image camera. In *IEEE Int. Conf. Pattern Recog.*, Aug 2006.

[48] Digital Image Library. Omnidirectional images. `http://www.artstor.org/`, 2009.

[49] Clive Loader. *Local Regression and Likelihood*. Springer, New York, 1999.

[50] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, pages 395–416, 2007.

[51] M. Mahy and P. Delabastita. Inversion of the neugebauer equations. *Color Res. Appl.*, 21:404–411, 1996.

[52] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. *IEEE Trans. on Evolutionary Computation*, 8(3):204–210, 2004.

[53] J. Myles and D. Hand. The multi-class metric problem in nearest neighbour discrimination rules. *Pattern Recognition*, 23:1291–1297, 1990.

[54] H. Nagahara, Y. Yagi, and M. Yachida. Super-resolution from an omnidirectional image sequence. In *Proc. 26th conf. IEEE Ind. Electronics Soc. (IECON)*, 2000.

[55] R. Nock, M. Sebban, and D. Bernard. A simple locally adaptive nearest neighbor rule with application to pollution forecasting. *Intl. Journal of Pattern Recognition and Artificial Intelligence*, 17(8):1–14, 2003.

[56] A. Okabe, B. Boots, K. Sugihara, and S. Chiu. *Spatial Tessellations*, chapter 6, pages 418–421. John Wiley and Sons, Ltd., Chichester, England, 2000.

[57] S. Park, M. Park, and M. Kang. Super-resolution image reconstruction: a technical overview. *IEEE Sig. Proc. Mag.*, 20:21–36, 2003.

[58] Kaare B. Petersen and Michael S. Pedersen. The matrix cookbook, February 2008.

[59] I. Pobboravsky and M. Pearson. Computation of dot areas required to match a colorimetrically specified color using the modified neugebauer equations. In *Proc. TAGA*, pages 65–77, 1972.

[60] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[61] W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill Publishing Co., 1976.

[62] J. S. Sánchez, F. Pla, and F. J. Ferri. On the use of neighbourhood-based non-parametric classifiers. *Pattern Recognition Letters*, pages 1179–1186, 1997.

[63] J. S. Sánchez, F. Pla, and F. J. Ferri. Improving the k-NCN classification rule through heuristic modifications. *Pattern Recognition Letters*, pages 1165–1170, 1998.

[64] G. Sharma. *Digital Color Handbook*, chapter 1: Color Fundamentals for Digital Imaging, pages 1–114. CRC Press, 2003.

[65] Mark Shaw, Gaurav Sharma, Raja Bala, and Edul N. Dalal. Color printer characterization adjustment for different substrates. *COLOR Research and Application*, 28(6):454–467, December 2003.

[66] R. Short and K. Fukunaga. The optimal distance measure for nearest neighbor classification. *IEEE Trans. on Information Theory*, 27(5):622–627, 1981.

[67] R. Sibson. *Interpreting multivariate data*, chapter A brief description of natural neighbour interpolation, pages 21–36. John Wiley, 1981.

[68] C. J. Stone. Consistent nonparametric regression. *The Annals of Statistics*, 5(4):595–645, 1977.

[69] P. Vandewalle, L. Sbaiz, J. Vandewalle, and M. Vetterli. Super-resolution from unregistered and totally aliased signals using subspace methods. *IEEE Trans. Sig. Proc.*, 55:3687–3703, 2007.

[70] D. Wallner. *Building ICC Profiles - the Mechanics and Engineering*, chapter 4: ICC Profile Processing Models, pages 150–167. International Color Consortium, 2000.

[71] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *NIPS*, pages 480–483, 2006.

[72] J. Wendel. A problem in geometric probability. *Math. Scand.*, 11:109–111, 1962.

[73] Y. Yagi. Omnidirectional sensing and applications. *IEICE Trans. Info. Sys.*, Mar 1999.

[74] H. Zhang, A. C. Berg, M. Maire, and J. Malik. SVM-KNN: discriminative nearest neighbor classification for visual category recognition. *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2:2126–2136, 2006.

[75] J. Zhang, Y.-S. Yim, and J. Yang. Intelligent selection of instances for prediction functions in lazy learning algorithms. *Artificial Intelligence Review*, 11:175–191, 1997.