2008

Analysis Of Kolmogorov's Superposition Theorem And Its Implementation In Applications With Low And High Dimensional Data.

Donald Bryant
University of Central Florida

ANALYSIS OF KOLMOGOROV'S SUPERPOSITION THEOREM AND ITS IMPLEMENTATION IN APPLICATIONS WITH LOW AND HIGH DIMENSIONAL DATA.

by

DONALD W. BRYANT
B.S. Massachusetts Institute of Technology, 1999
M.S. University of Central Florida, 2005

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Mathematics
in the College of Science
at the University of Central Florida
Orlando, Florida

Summer Term
2008

Major Professors: Xin Li and Mubarak Shah

ABSTRACT

In this dissertation, we analyze Kolmogorov's superposition theorem for high dimensions. Our main goal is to explore and demonstrate the feasibility of an accurate implementation of Kolmogorov's theorem. First, based on Lorentz's ideas, we provide a thorough discussion on the proof and its numerical implementation of the theorem in dimension two. We present computational experiments which prove the feasibility of the theorem in applications of low dimensions (namely, dimensions two and three). Next, we present high dimensional extensions with complete and detailed proofs and provide the implementation that aims at applications with high dimensionality. The amalgamation of these ideas is evidenced by applications in image (two dimensional) and video (three dimensional) representations, the content based image retrieval, video retrieval, de-noising and in-painting, and Bayesian prior estimation of high dimensional data from the fields of computer vision and image processing.

ACKNOWLEDGMENTS

I would like to thank all the people who helped me navigate through the dissertation process.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

The phrase "curse of dimensionality" was coined by Richard Bellman [3], in connection with the difficulty of optimization by exhaustive enumeration on product spaces. Bellman considered a cartesian grid of spacing 1/100 on the unit cube in 10 dimensions, where there are 10^{20} points; if the unit cube in 20 dimensions was considered, there would be 10^{40} points. His interpretation: if our goal is to optimize a function over a continuous product domain of a few dozen variables by exhaustively searching a discrete search space defined by a crude discretization, we could easily be faced with the problem of making tens of trillions of evaluations of the function. Bellman argued that this curse precluded, under almost any computational scheme then foreseeable, the use of exhaustive enumeration strategies.

No other century has been as inundated with data more than this one. This is the age where society has invested massively in the collection and processing of data of all kinds, on scales unimaginable. Hyperspectral Imagery, Internet Portals, Financial tick-by-tick data, and DNA microarrays are just a few of the better-known sources. Today, the trend is towards more observations as well as to radically larger numbers of variables. We are seeing examples where the observations gathered on individual data are curves, spectra, images, or even movies, so that a single observation has dimensions in the thousands or billions. Classical methods cannot cope with the explosive growth of the dimensionality of the observation vector. Mathematicians ideally desire the ability to find patterns in such high-dimensional data. In the context of the problems Bellman discussed, there are two

new influential principles: the **blessings** and the **curse** of dimensionality. This dissertation addresses the curse of dimensionality.

At the beginning of last century, David Hilbert [8] compiled a set of twenty three problems as a challenge to mathematicians in his celebrated address delivered to the Second International Congress of Mathematicians in Paris. The 13th problem entails proving whether or not a solution exists for all 7-th degree equations using functions of two arguments. It was first presented in the context of nomography, and in particular "nomographic construction" a process whereby a function of several variables is constructed using functions of two variables. The actual question is more easily posed, however, in terms of the representation of continuous functions.

Hilbert asked whether it was possible to construct the solution of the general seventh degree equation $x^7 + ax^3 + bx^2 + cx + 1 = 0$ using a finite number of two-variable functions. The complete answer was given by Vladimir Arnold in 1957, then only nineteen years old and a student of Andrey Kolmogorov. Kolmogorov had shown in the previous year that any continuous function of several variables can be constructed with a finite number of three-variable functions. Arnold then expanded on this work to show that in fact only two-variable functions were required, thus answering Hilbert's question. This formulation is further generalized to allow almost any multivariate function (there are some exceptions [16]) to be represented by not just a two dimensional function, but a one dimensional continuous function. This generalized version is usually referred to as Kolmogorov's superposition theorem.

1.1 Overview

This dissertation will attempt to address the representation of multivariate functions through the novel theorem developed by Andrey Kolmogorov. This theorem states that a continuous function of several variables can be represented by a composition and superposition of functions of one variable.

1.1.1 Dissertation Outline

In Chapter 2, we give background information in two areas, Approximation Theory and Probability and Statistics. This will give the reader the ability to reference particular topics that they may require some background.

In Chapter 3, Kolmogorov's theorem for dimension two is studied. Here we follow the outline given by Lorentz [16] and provide additional explanations for particular portions. This theorem will be used to re-examine a new way to deal with high dimensional data. Some examples of results obtained from our implementation are presented. Next, we present some properties of the representing one-dimensional functions. We will explore what effect the operations on the one dimensional functions have on the corresponding high dimensional function. It is hoped that by completely understanding the relationship between different sets of high dimensional data and the associated composition to one dimension, there will be more efficient processing of the data to be analyzed.

In Chapter 4, we present a high dimensional extension of Lorentz's proof . Lorentz's proof is specifically for a two dimensional function $f(x, y)$ but is easily extended to higher

dimensions. Here we try to present a complete proof for the high dimensional case so that it may be read independently. The purpose of presenting a complete proof of the high dimensional case is twofold. First, we avoid referring to other parts of the dissertation which may interrupt the continuity of the reading of the proof. Second, it allows us to write out all of the details in the proof that will be useful for our implementation. The last topic covered in this chapter is a detailed discussion of our algorithmic interpretation of Kolmogorov's theorem.

In Chapter 5, we will discuss the issues with the implementations and give some results from experiments conducted with our image retrieval system. We will discuss the obstacles and suggest some strategies for improving the implementation of the algorithm.

In Chapter 6, we discuss a few applications of the algorithm. There are many areas where this algorithm can be applied but we focus primarily on digital image retrieval. We explain in depth the manner in which a search engine uses the similarity and other properties between g-functions of images as a matching criteria. The idea is that two similar images (2-D functions) should have similar g-functions. We also present a framework for developing high dimensional PDF's in close relation to those presented by Roth and Black [4]. Lastly our extension and modifications to that method will be discussed.

To make our algorithm useful for others, we will provide our MATLAB©codes and the instructions on how to use them in the Appendices.

CHAPTER TWO: BACKGROUND

In order to facilitate an adequate discussion in later chapters on Kolmogorov's theorem and its applications, we present some background material here. In the first section, we present the important concepts from approximation theory which will be required for the discussions and extensions of Kolmogorov's theorem in Chapter 4 and Chapter 5. We follow the discussion of approximation theory with a short introduction to concepts in probability and statistics. This will be helpful for the reader to refer to when we discuss applications in Chapter 6.

2.1 Approximation Theory

One of the main tasks in Approximation Theory is the task of finding a linear combination of functions $\phi_i \epsilon \Phi$ such that for $P = a_1\phi_1 + \cdots + a_n\phi_n$, P is close to a given function f, which is defined on a fixed space A. This involves selecting the set Φ and deciding how to measure the deviation of P from f.

In order to measure the deviation of P from f, let A be a compact metric space, and let $C = C[A]$ be the set of continuous real functions on A. C is a linear space with the usual properties: if $f, g \in C$ then $af + bg \in C$ for $a, b \in \mathbb{R}$.

The supremum

$$\|f\| = \sup_{x \in A} |f(x)| \tag{2.1}$$

is attained for all functions $f \in C$; thus

$$\|f\| = \max_{x \in A} |f|. \tag{2.2}$$

This defines a norm on $C[A]$ which has the following properties:

$$\begin{aligned} \|f\| \geq 0; \quad \|f\| = 0 \text{ if and only if } \quad f = 0 \\ \|af\| = |a| \cdot \|f\| \\ \|f + g\| \leq \|f\| + \|g\|. \end{aligned} \tag{2.3}$$

Thus, C is a normed linear space associated with all continuos real functions over $\mathbb{R}$. The convergence $f_n \to f$ in the norm of C, $\|f_n - f\| \to 0$ as $n \to \infty$, is equivalent to the uniform convergence of $f_n(x)$ to $f(x)$ for all $x \in A$. The space C is a complete normed linear space. If f_n is a Cauchy sequence, then f_n converges to some element, f, of C:

$$\|f_n - f\| \to 0 \tag{2.4}$$

Complete normed spaces are called *Banach spaces.*

The following definitions apply to any Banach space X with elements f and a subset $\Phi \subseteq X$.

Definition. A function f is called *approximable* by linear combinations

$$P = a_1\phi_1 + a_2\phi_2 + \cdots + a_n\phi_n; \ \phi_i \in \Phi, \ a_i \text{ real}, \tag{2.5}$$

if for each $\epsilon > 0$ there is a P with $\|f - P\| < \epsilon$. Often, Φ is a sequence: $\phi_1, \phi_2, \cdots, \phi_n, \cdots$. Then

$$E_n(f) = E_n^{\Phi}(f) = \inf_{a_1,\ldots,a_n} \|f - (a_1\phi_1 + \cdots + a_n\phi_n)\| \tag{2.6}$$

is the nth *degree of approximation* of f by the ϕ_i. If the infimum is obtained for some P, then this P is called a *best approximation*.

Another useful definition is related to the continuity of functions.

Definition. To measure the continuity of a function $f \in C([a,b])$, consider the *first difference* with step t,

$$\Delta_t f(x) = f(x+t) - f(x) \tag{2.7}$$

of the function f and put

$$\omega(f,h) = \omega(h) = \max_{\substack{x, x+t \in [a,b] \\ |t| \le h}} |f(x+t) - f(x)|. \tag{2.8}$$

The function $\omega(h)$, called the modulus of continuity of f, is defined for $0 \le h < l$, where $l = b - a$.

Lastly we look at a way to group functions into different classes. The Lipshitz condition is one way to accomplish this.

Definition. A function f defined on $A = [a,b]$, satisfies a *Lipshitz condition* with constant M and exponent α, or belongs to the class $Lip_M\alpha$, $M > 0$, $0 < \alpha \le 1$ if

$$|f(x') - f(x)| \le M|x' - x|^\alpha, \ x, x' \in A \ . \tag{2.9}$$

This is equivalent to the inequality $\omega(f,h) \le Mh^\alpha$, for $0 \le h < l$. As an example, if f has a derivative that satisfies $|f'(x)| \le M$, $x \in A$, then $f \in Lip_M 1$. It is know that a function satisfying a Lipshitz condition on $[a,b]$ is continuous $[a,b]$. Define $\omega(f,A) = \sup_{0 \le h < l} \omega(f,h) = \omega(f,l)$. This can be extended to 2-D or higher dimensional cases.

Weierstrass Comparison Test. If $\|f_k\|_A \leq M_k$, for all k, and $\sum_1^\infty M_k$ converges, then $\sum_1^\infty f_k$ converges uniformly on A.

Geometric Series. The geometric series $\sum_0^\infty x^n$ converges to $\frac{1}{1-x}$ for $|x| < 1$ and diverges when $|x| \geq 1$.

Any continuous function f defined on a closed and bounded (compact) set D attains a maximum (and minimum) value at some point in D.

2.2 Probability and Statistics

Probability is the mathematical language used to quantify uncertainty. Statistical inference is the process of using data to infer the distribution that generated the data. In this section we introduce some basic concepts from probability theory and statistical inference. This will allow the reader to recall important concepts while reviewing the applications related to these fields.

2.2.1 Probability

A function P which assigns a real number $P(A)$ to each event A is a **probability distribution** or a **probability measure** if it satisfies the following three axioms:

1. **Axiom 1**: $P(A) \geq 0$ for every event A

2. **Axiom 2**: $P(\Omega) = 1$, where Ω is the complete sample space

3. **Axiom 3**: If $A_1, A_2, \ldots$ are disjoint then

$$P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i)$$

- Two events A and B are **independent** if

$$P(AB) = P(A)P(B)$$

- A set of events $\{A_i : i \in I\}$ are independent if

$$P\left(\bigcap_{i \in J} A_i\right) = \prod_{i \in J} P(A_i)$$

for every finite subset J of I.

- If $P(B) > 0$ then the **conditional probability** of A given B is

$$P(A|B) = \frac{P(AB)}{P(B)}$$

- (Bayes' Theorem) Let $A_1, \ldots, A_k$ be partitions of of the sample space Ω such that $P(A_i) > 0$ for each i. If $P(B) > 0$, then for each $i = 1, \ldots, k$,

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_j P(B|A_j)P(A_j)} \ .$$

2.2.2 Statistics

- A random variable is a mapping $X : \Omega \to \mathbb{R}$ that assigns a real number $X(\omega)$ to each outcome ω and satisfies the condition: $(X \leq x) = \{\omega : X(\omega) < x\}$ is an event for every $x \in \mathbb{R}$.

- Given a random variable the **cumulative distribution function**, or CDF, is the function $F_X : \mathbb{R} \to [0, 1]$ defined by

$$F_X(x) = P(X \leq x)$$

- When X is discrete, a **probability function** or **probability mass function** for X is defined by $f_X(x) = P(X = x)$.

- A random variable X has a probability density function (PDF) if there exists a function f_X such that $f_X(x) \geq 0$ for all x, $\int_{-\infty}^{\infty} f_X(x)dx = 1$ and for every $a \leq b$,

$$P(a < X < b) = \int_a^b f_X(x)dx \ .$$

- If **probability distribution function** f_X exists, then

$$F_X(x) = \int_{-\infty}^{x} f_X(t)dt$$

and $f_X(x) = F_X'(x)$ at almost all points x.

- The **expected value**, or **mean**, or **first moment**, of X is defined to be

$$E(X) = \begin{cases} \sum_x x f_X(x) & \text{if X is discrete} \\ \int x f_X(x)dx & \text{if X is continuous and } f_X \text{ exists} \end{cases}$$

assuming the integral (or sum) is well defined. We use the following notation to denote the expected value of X

$$E(X) = \mu \ .$$

CHAPTER THREE: KOLMOGOROV'S THEOREM

3.1 Kolmogorov's Theorem

Kolmogorov's Theorem states that a function of several variables may be represented as a superposition and composition of one dimensional functions. Here we present a version of the proof for Kolmogorov's theorem under the guidance of the proof by G. G. Lorentz [16].

3.1.1 Theorem Statement

Theorem 1 ***Kolmogorov's Theorem*** **(*1957*)** *Let $I = [0,1]$ and let S be the two dimensional square, $0 \leq x \leq 1$, $0 \leq y \leq 1$. There exists constants λ_i, $i = 1, 2$, and five functions $\varphi_q(x)$ for $q = 0, \ldots, 4$ defined on I with values in I,which have the following properties: The φ_q's are strictly increasing and belong to a class Lip α, $\alpha > 0$. For any continuous function f of two variables x, and y, on S, there exists a function g defined over the interval $[0,2]$ such that*

$$f(x,y) = \sum_{q=0}^{4} g\left(\lambda_1 \varphi_q(x) + \lambda_2 \varphi_q(y)\right) \tag{3.1}$$

A detailed proof will be presented here which is developed very closely to that of Lorentz.

3.1.2 Details required for Proof

There are several tasks which must be completed in order to develop the functions described in the statement of the theorem. They are as follows:

1. Determine values of λ_i such that they are rationally linearly independent

   (i.e., $\sum_i a_i\lambda_i = 0$, $a_i \in \mathbb{Q}$ when and only when all $a_i = 0$)

2. Construct the φ_q $(q = 0, \ldots, 4)$ functions used to map a point in two dimensional space S to a point on the real interval $[0, 2]$ via $\lambda_1\varphi_q(x) + \lambda_2\varphi_q(y)$.

3. Construct the g function used to represent the function f.

In the 2-D case, and similarly for higher dimensions, the first observation is that each point (x, y) can be mapped to disjoint intervals on the real line through the mapping $u_q = \lambda_1\varphi_q(x) + \lambda_2\varphi_q(y)$. (Over these intervals we shall define the function values for g.) This mapping is continuous and is also one-to-one. The one-to-one property is essential since otherwise there would be two different points (x_1, y_1) and (x_2, y_2) which would map to the same point on the real line. As a result, a problem would occur in the reconstruction given by the right side of equation (3.1). For example, if we define over the points (x_1, y_1) and (x_2, y_2) different values $f(x_1, y_1)$, $f(x_2, y_2)$, respectively, we would have the same approximate value from the reconstruction for both points. This is what we do not want.

Let us start the proof. Choose the λ_i constants such that for rational numbers a_i the following holds true:

$$\sum a_i\lambda_i \neq 0 \quad \text{if not all } a_\text{i} = 0 \tag{3.2}$$

For the two dimensional case we can let $\lambda_1 = 1$ and $\lambda_2 = \lambda$ where we choose λ to be an irrational number. In this case it is certain that 1 and λ are linearly independent over the rational numbers.

Next comes the construction of the φ-functions. We begin the process with some observations. There are four properties that we wish the five φ_q functions to possess. These will insure that the intervals created by the points (x, y) will be disjoint under the transformation $u_q = \varphi_q(x) + \lambda\varphi_q(y)$ on S into the interval [0,2]. The four criteria are:

1. The function $\varphi_q(x)$ will be strictly increasing.

2. The $\varphi_q(x)$ will first be defined over a set of points where the values of the φ_q function will be rational and distinct (and all the φ_q will be extended by continuity).

3. The slope of the line between any two segments connecting the points $(\alpha_i, \varphi_q(\alpha_i))$,$(\beta_i, \varphi_q(\beta_i))$ is less than 5^k, where k is called the rank. This will be explained more clearly later.

4. The intervals $[\varphi_q(\alpha_i) + \lambda\varphi_q(\alpha_j), \varphi_q(\beta_i) + \lambda\varphi_q(\beta_j)]$ will be disjoint for different $q = 0, \ldots, 4$.

In order to accomplish this we define the φ_q as piecewise linear functions over disjoint intervals of $I = [0, 1]$. Those intervals are defined by

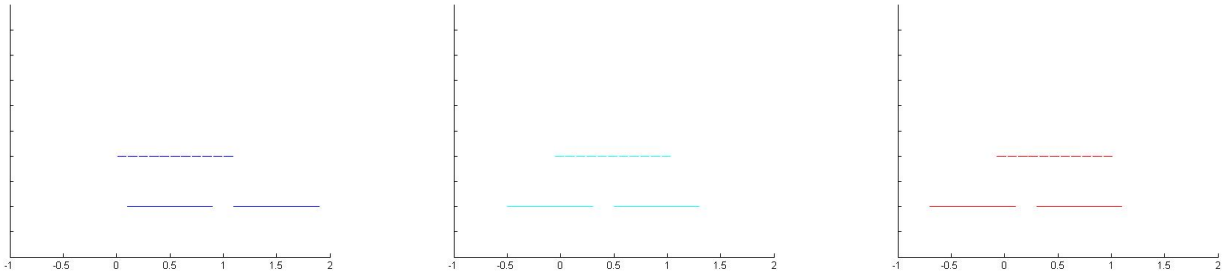
$$I_i^k = [i \cdot 10^{-k+1} + 10^{-k}, i \cdot 10^{-k+1} + 9 \cdot 10^{-k}] \tag{3.3}$$

$$i = 0, 1, 2, \ldots, 10^{k-1}; k = 1, 2, 3, \ldots$$

The integer k, as stated earlier, is called the rank of the this interval. All I_i^k are contained in $I = [0, 1]$, except when $i = 10^{k-1}$. Figure 3.1 shows intervals for up to $k = 2$.

The functions φ_q are associated with intervals I_{qi}^k, which are shifts of the intervals I_i^k. The relation is as follows:

$$I_{qi}^k = I_i^k - 2q \cdot 10^{-k} \tag{3.4}$$



(a) $q = 0$ (b) $q = 3$ (c) $q = 4$

Figure 3.1: Example of intervals i, k up to $k = 2$

$i = 0, 1, 2, \ldots, 10^{k-1}; k = 1, 2, 3, \ldots$

For intervals not contained entirely in I, we replace them by $I \cap I_{qi}^k$. Figure 3.2 demonstrates this property.

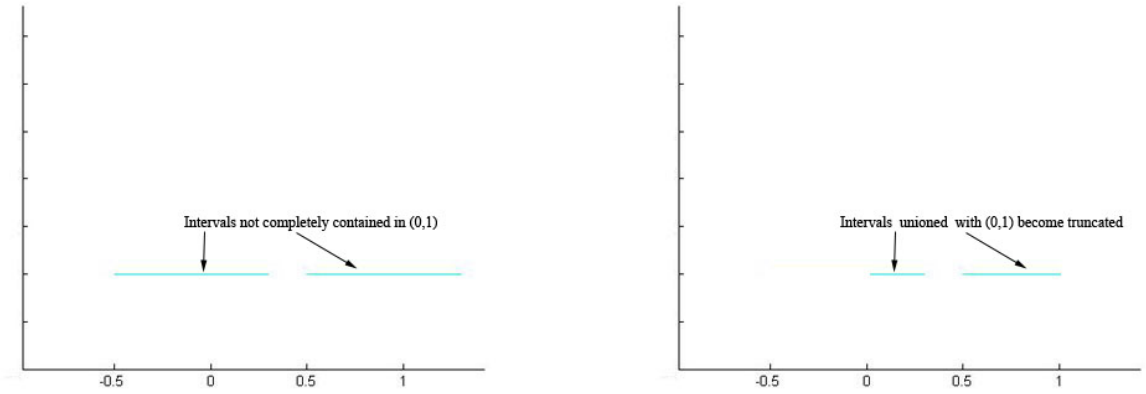


Figure 3.2: Intervals I_{3i}^1 not contained completely in $I = (0, 1)$

Here we specify the exact definition of the α, β points. Defining $\alpha_{qi}^k = i \cdot 10^{-k+1} + 10^{-k} - 2q \cdot 10^{-k}$ and $\beta_{qi}^k = i \cdot 10^{-k+1} + 9 \cdot 10^{-k} - 2q \cdot 10^{-k}$, equation (3.4) becomes,

$$I_{qi}^k = I_i^k - 2q \cdot 10^{-k} = [\alpha_{qi}^k, \beta_{qi}^k] \tag{3.5}$$

for $\qquad q = 0, 1, \ldots, 4;\ i = 0, 1, 2, \ldots, 10^{k-1};\ k = 1, 2, 3, \ldots$

If the rank k is fixed, the intervals in (3.5) form five families, corresponding to $q = 0, \cdots, 4$. No one of the families covers I. However, each point $x \epsilon I$ is covered by at least *four* of the *five* families, $q = 0, \cdots, 4$. The functions φ_q will be strictly increasing, which is sufficient for a one-to-one mapping. The most important property is that the image of each interval I_{qi}^k of each given rank under the φ_q will be disjoint, not only for a fixed q, but for all q and i (and a fixed rank k). Actually, more properties will need to be clarified.

The exact formulation of the required properties is given by Lemma 1 .

Lemma 1 *There exist five strictly increasing functions, φ_q, $q = 0, \cdots, 4$, defined on $[0, 1]$, with values in $[0, 1]$, and belonging to a class Lip α, $\alpha > 0$ such that for each fixed rank $k = 1, 2, \ldots$, the intervals*

$$\Delta_{qij}^k = [\varphi(\alpha_{qi}^k) + \lambda\varphi(\alpha_{qj}^k), \varphi(\beta_{qi}^k) + \lambda\varphi(\beta_{qj}^k)] \tag{3.6}$$

where $q = 0, \ldots, 4$ and $i, j = 0, \ldots, 10^{k-1}$ are all disjoint.

The proof of Lemma 1 will outline the construction of the φ_q functions. The functions are constructed by induction on the rank k.

1. $k = 0$: define $\alpha_{q0}^0 = 0$, $\beta_{q0}^0 = 1$ and define $0 \leq \varphi_q(0) < \varphi_q(1) < 1$, $q = 0, \ldots, 4$.

2. Fix q for the steps that follow

   (a) There exists two types of I_{qi}^k intervals for rank $k > 0$

      i. Intervals of the first kind: These intervals contain one point from α_{qi}^l, or β_{qi}^l from a lower rank $l < k$.

ii. Intervals of the second kind: these intervals do not contain any points α_{qi}^l, or β_{qi}^l from a lower rank $l < k$.

See Figures 3.3 and 3.4 for examples of each. For short, the points $\alpha_{qi}^l, \beta_{qi}^l$ will be designated as γ, where γ is either α_{qi}^l or β_{qi}^l.

(b) For intervals I_{qi}^k, we first assign φ_q at the points α_{qi}^k and β_{qi}^k the same value: $\varphi_q(\alpha_{qi}^k) = \varphi(\beta_{qi}^k) =: \varphi_{qi}^k$, where φ_{qi}^k is determined as follows

i. For intervals of the first kind, it must be determined which point $\alpha_{qi'}^l$ or $\beta_{qi'}^l$ from a lower rank ($l < k$) falls into I_{qi}^k. We call this point γ_{qi}^k. Then we assign $\varphi_{qi}^k = \varphi(\gamma)$.

ii. For intervals of the second kind, each point $\alpha_{qi}^k, \beta_{qi}^k$ of I_{qi}^k lie in some interval (γ, γ'), where γ, γ' are two adjacent points $\alpha_{qi'}^l, \beta_{qi'}^l$ from a lower rank. To assign values to $\varphi_q(\alpha_{qi}^k)$ and $\varphi(\beta_{qi}^k)$, it is sufficient to determine the increase over the gap between the intervals I_{qi}^k which fall into (γ, γ'). Let two points in the gap be designated (x,x'). One criteria of Lemma 1 is that the slope of each polygonal line connecting the points $(\alpha_{qi'}^l, \varphi_q(\alpha_{qi'}^l))$ and $(\beta_{qi'}^l, \varphi_q(\beta_{qi'}^l))$, $l \leq k$ is strictly smaller than 5^k. This requirement, along with the observation that each gap between the I_{qi}^k is $\frac{1}{5}(\gamma, \gamma')$, leads to the following formula for determining the value for $\varphi_q(x)$ and $\varphi_q(x')$.

$$\varphi_q(x') - \varphi_q(x) = 5\frac{x' - x}{\gamma' - \gamma}[\varphi_q(\gamma') - \varphi_q(\gamma)] \tag{3.7}$$

3. Now the values of φ_q need to be modified so that φ_q is strictly increasing for all q.

(a) Take all the values φ_{qi}^k for rank k and form the values

$$\varphi_{qi}^k + \lambda\varphi_{qj}^k, \quad q = 0, \cdots, 4; \quad i, j = 0, \cdots, 10^{k-1} \tag{3.8}$$

and order them on the real number line.

(b) choose an $\epsilon > 0$ so small that the 2ϵ neighborhood of the points in (3.8) are disjoint. We choose values in the ϵ neighborhood of φ_{qi}^k and define the rational values of $\varphi_q^k(\alpha_{qi}^k)$ and $\varphi_q^k(\beta_{qi}^k)$ such that

$$\varphi_q^k(\alpha_{qi}^k) < \varphi_{qi}^k < \varphi_q^k(\beta_{qi}^k) \tag{3.9}$$

If there is an endpoint at 0 or 1, we modify the inequality such that, for example if $\beta_{qi}^k = 1$, it should read $\varphi_q(\alpha_{qi}^k) < \varphi_{qi}^k = \varphi_q(\beta_{qi}^k)$.

4. Proceed to step (2) with $k = k + 1$;

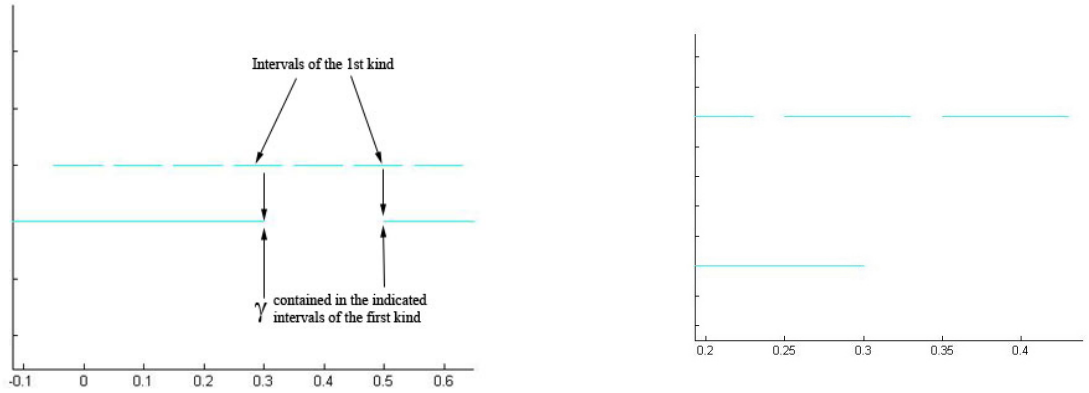
Once the preceding steps are completed the four criteria which are necessary for the φ functions stated at the beginning will be satisfied. Just for the purpose of recollection they are again stated using new notation as follows:

1. The functions φ_q are strictly increasing on the set of all the points $\alpha_{qi}^l, \beta_{qi}^l$, $0 \le i \le 10^{l-1}$, $l \le k$.

2. The slope of each segment of the polygonal line connecting the neighboring points $(x, \varphi_q(x))$, $x = \alpha_{qi}^l, \beta_{qi}^l$, $l \le k$, is strictly smaller than 5^k.

3. All values of $\varphi_q(\alpha_{qi}^l), \varphi_q(\beta_{qi}^l)$, $q = 0, \cdots, 4$, $0 \le i \le 10^{k-1}$, $l \le k$, are rational and distinct.

4. The intervals (3.6) are disjoint.



(a) Intervals of 1st kind

(b) close up of intervals in *(a)*

Figure 3.3: Example of intervals I_{3i}^2 of the first kind

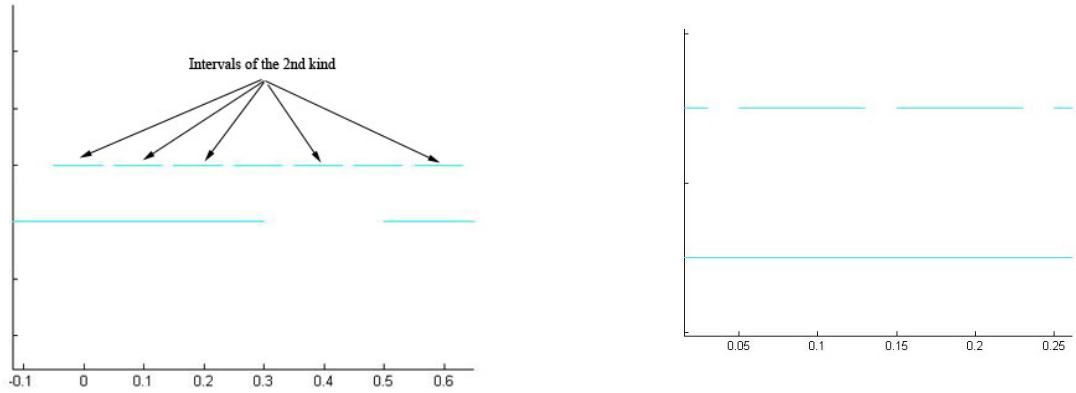
The last task is to construct the g-function. This is done through a series approximation to f. Define, for each function g on $[0, 2]$,

$$h(x, y) = \sum_{q=0}^{4} g(\varphi_q(x) + \lambda\varphi_q(y)) \tag{3.10}$$

For a function of two variables $f(x, y)$, the approximating function $h(x, y) \approx f(x, y)$ is formed over little squares S_{qij}^k, where

$$S_{qij}^k = I_{qi}^k \times I_{qj}^k, \tag{3.11}$$

for $q = 0, \cdots, 4; i, j = 0, \cdots, 10^{k-1}$



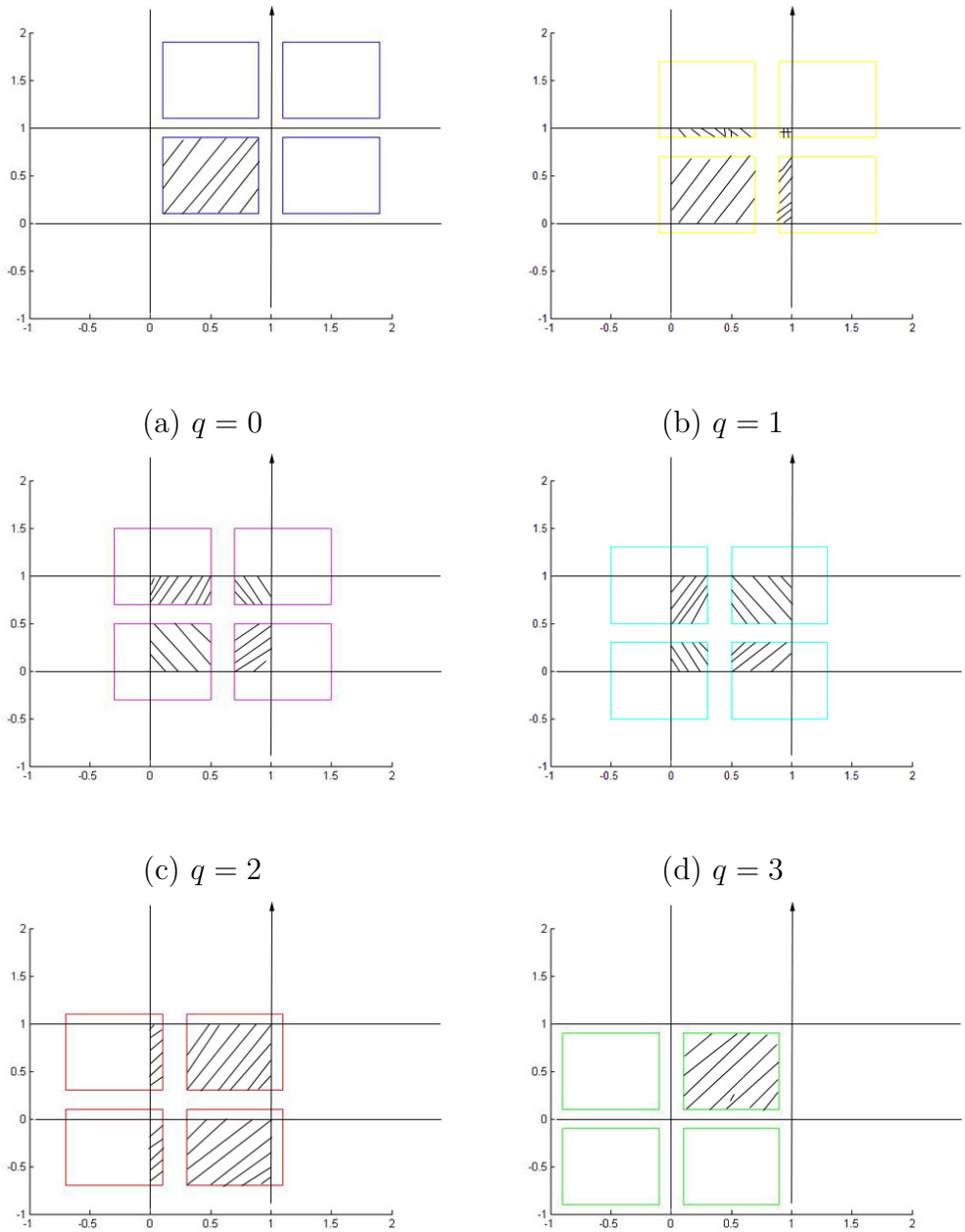
(a) Intervals of the 2nd kind (b) close up of intervals

Figure 3.4: Example of intervals I_{3i}^2 of the second kind

The image of squares (3.11) under the map $(x, y) \longrightarrow \varphi_q(x) + \lambda\varphi_q(y)$ are the intervals Δ_{qij}^k of (3.6). This is illustrated in Figure 3.5. For each fixed k, the functions $\varphi_q(x) + \lambda\varphi_q(y)$ maps squares (3.11) with corresponding q onto disjoint intervals.

The intervals I_{qi}^k form five families, $q = 0, \cdots, 4$. An interesting property to notice is that for fixed rank k, each point x in [0,1] can fail to be covered only by one of the five families. This also applies to y in $[0, 1]$ and the intervals I_{qj}^k. Therefore each point (x, y) of S can fail to be covered by the squares (3.11) only for at most two values of q. There are at least three hits at (x, y) out of five tries, $q = 0, \cdots, 4$.

The following result, Lemma 2, provides the means and the criteria by which the values for $g(u)$ are assigned over the disjoint intervals (3.6).



(a) $q = 0$ (b) $q = 1$

(c) $q = 2$ (d) $q = 3$

(e) $q = 4$ (f) $q = 5$

Figure 3.5: Example squares S for $k = 1$

Lemma 2 . *Let* $\frac{2}{3} < \theta < 1$. *Let* $S = I_{qi}^k \times I_{qj}^k$. *For each* $f \in C[S]$, *there is a function*

$g \in C[0,2]$ *such that*

$$\|f-h\| \leq \theta\|f\| \text{ and } \|\mathrm{g}\| \leq \frac{1}{3}\|\mathrm{f}\| \tag{3.12}$$

Proof. Take $\epsilon \in (0, \theta - \frac{2}{3})$. Choose k large enough so that $\omega(f, S_{qij}^k) < \epsilon\|f\|$. Let (x,y) be an arbitrary point in S. There are at least three values of q, where $(x,y) \in S_{qij}^k$, for some i,j. The corresponding terms of the sum in (3.10) will be assigned the values $\frac{1}{3}f_{qij}^k$, each of which is $\frac{1}{3}f(x,y)$, with an error less than $\frac{1}{3}\epsilon\|f\|$. The two remaining terms of (3.10) are each $\leq \frac{1}{3}\|f\|$ in absolute value. It follows then that

$$|f(x,y) - h(x,y)| \leq 3\frac{1}{3}\epsilon\|f\| + \frac{2}{3}\|f\| \leq \theta\|f\|. \tag{3.13}$$

This completes the proof of Lemma 2.

3.1.3 Proof of Theorem

Let $f \in C[S]$ be given. We define a sequence of functions $g_r \in C[0,2]$ with the corresponding functions

$$h_r(x,y) = \sum_{q=0}^{4} g_r(\varphi_q(x) + \lambda\varphi_q(y)) \tag{3.14}$$

as follows: First let g_1, h_1 be given by Lemma 2 with

$$\|f-h_1\| \leq \theta\|f\|,\ \|g_1\| \leq \frac{1}{3}\|f\|\ . \tag{3.15}$$

Applying Lemma 2 again and obtain functions g_2, h_2, while using $(f-h_1)$ instead of f, for which

$$\|(f-h_1)-h_2\| \leq \theta\|f-h_1\| \leq \theta^2\|f\|, \|g_2\| \leq \frac{1}{3}\|f-h_1\| \leq \frac{1}{3}\theta\|f\|\ . \tag{3.16}$$

In general the functions g_r, h_r will satisfy

$$\|f - h_1 \cdots - h_r\| \leq \theta^r \|f\|, \ \|g_r\| \leq \frac{1}{3}\theta^{r-1}\|f\|, \ r = 1, 2, \cdots \tag{3.17}$$

Let $r \to \infty$ in the relation (3.17). Then, the series $\sum_1^\infty h_r$ converges uniformly to f. Note that $\sum_1^\infty g_r$ is dominated by $\sum \frac{1}{3}\theta^{r-1}\|f\|$ which is convergent. So by Weierstrass theorem, $\sum_1^\infty g_r$ converges uniformly. Let g be the sum. This gives the desired representation (3.1).

3.2 Properties of f and its g function

Kolmogorov's theorem defines a mapping of a function f of 2 (or n in general) variables into a function of a single variable, g. One may ask several questions about the properties of the function g and the effect of operations of f on the function g. For example, given two multidimensional functions f_1 and f_2 and their sum $f = f_1 + f_2$, can any useful relation be obtained between their corresponding g-functions, g_1, g_2, and g_f? What about the relation between the derivatives of f and its g-function g_f? Or what about the relationship between the integral of f over its domain and the integral of the g-function?

The usefulness of the answers to such questions lies in the ability to analyze multidimensional functions with knowledge of only their one-dimensional g-functions. If these properties can be formally established then one may be able to perform traditional 2-D or 3-D image processing tasks by means of the 1-D g-function. The ultimate goal would be to have the ability to operate on very high dimensional functions by manipulation of the 1-D g-functions.

3.2.1 Addition and Multiplication

We start this section with the simple properties of addition and multiplication. Each is shown in a short proof and then demonstrated through a few examples.

Addition. For two multidimensional functions f_1, f_2 , let $f = f_1 + f_2$, then given their respective g-functions, g_1, g_2, g_f, we have $g_f = g_1 + g_2$.

Proof. This is done for the dimension $n = 2$ case. The higher dimensional cases have a similar argument. Let $u_q = \varphi_q(x) + \lambda\varphi_q(y)$ be the variable in the domain of g. Let $f = f_1 + f_2$ and let each f, f_1, f_2 have g-functions , g_f, g_1, g_2, respectively. On each interval Δ^k_{qij}, $g_f(u_q) = \frac{1}{3}f^k_{qij}$, $g_1(u_q) = \frac{1}{3}f^k_{1,qij}$, and $g_2(u_q) = \frac{1}{3}f^k_{2,qij}$. Now on each such interval, $f^k_{qij} = f^k_{1,qij} + f^k_{2,qij}$ and finally,

$$\begin{aligned} g_f(u_q) &= \frac{1}{3}f^k_{qij} \\ &= \frac{1}{3}\left(f^k_{1,qij} + f^k_{2,qij}\right) \\ &= \frac{1}{3}f^k_{1,qij} + \frac{1}{3}f^k_{2,qij} \\ &= g_1(u_q) + g_2(u_q) \ . \end{aligned} \tag{3.18}$$

In Figure 3.6 , we show two functions and the sum of their g-functions along with the reconstruction.

Multiplication. For two multi-dimensional functions f_1, f_2 , let $f = f_1 \cdot f_2$, then given their respective g-functions, g_f, g_1, g_2, we have $g_f = (n+1)\cdot g_1 \cdot g_2$, where n is the number of dimensions.

Proof. This is done for the dimension $n = 2$ case. The higher dimensional cases have a similar argument. Let $u_q = \varphi_q(x) + \lambda\varphi_q(y)$ be the variable in the domain of g-functions. Let

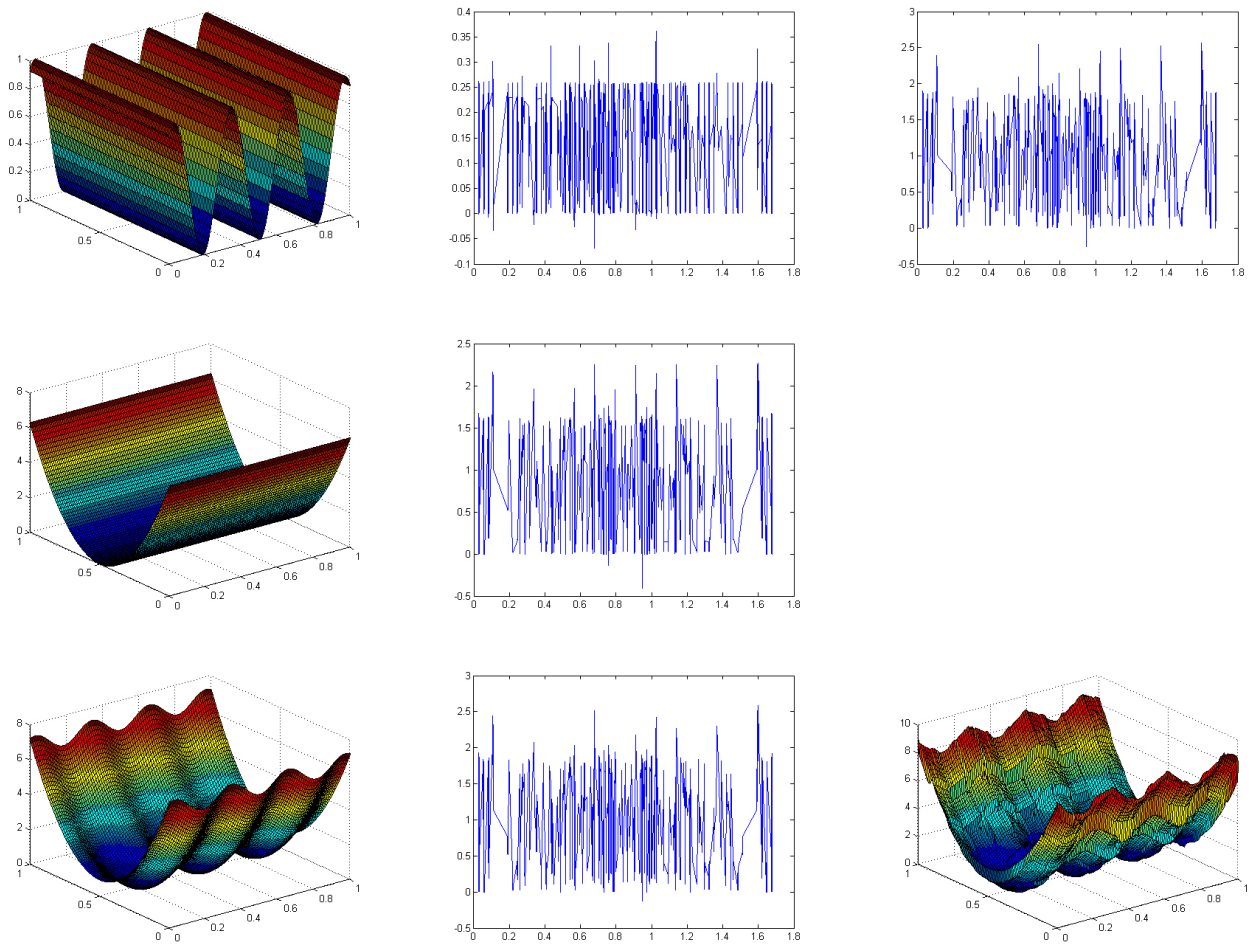


Figure 3.6: Addition

$f = f_1 \cdot f_2$ and let each f, f_1, f_2 have g-functions , g_f, g_1, g_2, respectively. On each interval Δ^k_{qij}, $g_f(u_q) = \frac{1}{3} f^k_{qij}$, $g_1(u_q) = \frac{1}{3} f^k_{1,qij}$, and $g_2(u_q) = \frac{1}{3} f^k_{2,qij}$. Now on each interval, $f^k_{qij} = f^k_{1,qij} \cdot f^k_{2,qij}$

and finally,

$$\begin{aligned} g_f(u_q) &= \frac{1}{3} f^k_{qij} \\ &= \frac{1}{3}\left(f^k_{1,qij} \cdot f^k_{2,qij}\right) \\ &= 3{\cdot}\frac{1}{3} f^k_{1,qij} \cdot \frac{1}{3} f^k_{2,qij} \\ &= 3 g_1(u_q){\cdot} g_2(u_q)\ . \end{aligned}$$

In Figure 3.7 , we show two functions and the product of their g-functions along with the reconstruction.

3.2.2 Shifting

For a multi-dimensional function f_1 with g-function g_1, the function $f = f_1 + C$ and has g-function g_f which has the relation, $g_f = g_1 + \frac{C}{n+1}$.

In order to shift along a dimension, we first need to define some universal functions dU_q.

Definition. There are n universal dU-functions, dU_{x_i}, and constants δx_i, $i = 1 \ldots n$, such that on Δ^k_q we define $dU(u_q) = \varphi_q(M_i + \delta x_i) - \varphi(M_i)$ for all $u_q \in \Delta^k_q$. $M_i = \frac{\alpha^k_{qi} + \beta^k_{qi}}{2}$ is the midpoint of interval I^k_{qi}. Also, $\delta x_i = (2s+1)^{-k+1}$, where $s = 2n+1$

For the two dimensional case we have, $\delta x = \delta y = 10^{-2}$

$$\begin{aligned} &x_1 = x; \quad dU_x(u_q) = \varphi_q(\tfrac{\alpha^k_{qi}+\beta^k_{qi}}{2} + \delta x) - \varphi_q(\tfrac{\alpha^k_{qi}+\beta^k_{qi}}{2}) \quad \text{for } u_{\mathrm{q}} \in \Delta^{\mathrm{k}}_{\mathrm{qij}} \\ &x_2 = y; \quad dU_y(u_q) = \varphi_q(\tfrac{\alpha^k_{qj}+\beta^k_{qj}}{2} + \delta y) - \varphi_q(\tfrac{\alpha^k_{qj}+\beta^k_{qj}}{2}) \quad \text{for } u_{\mathrm{q}} \in \Delta^{\mathrm{k}}_{\mathrm{qij}} \end{aligned} \tag{3.19}$$

For a function f_1 , with g-function g_1, which is the shift of another function f , with g-function g_f, along one dimension by an amount δx, the following is their relationship:

$$g_1(u_q) = g_f(u_q + dU_{x_i}(u_q)) \tag{3.20}$$

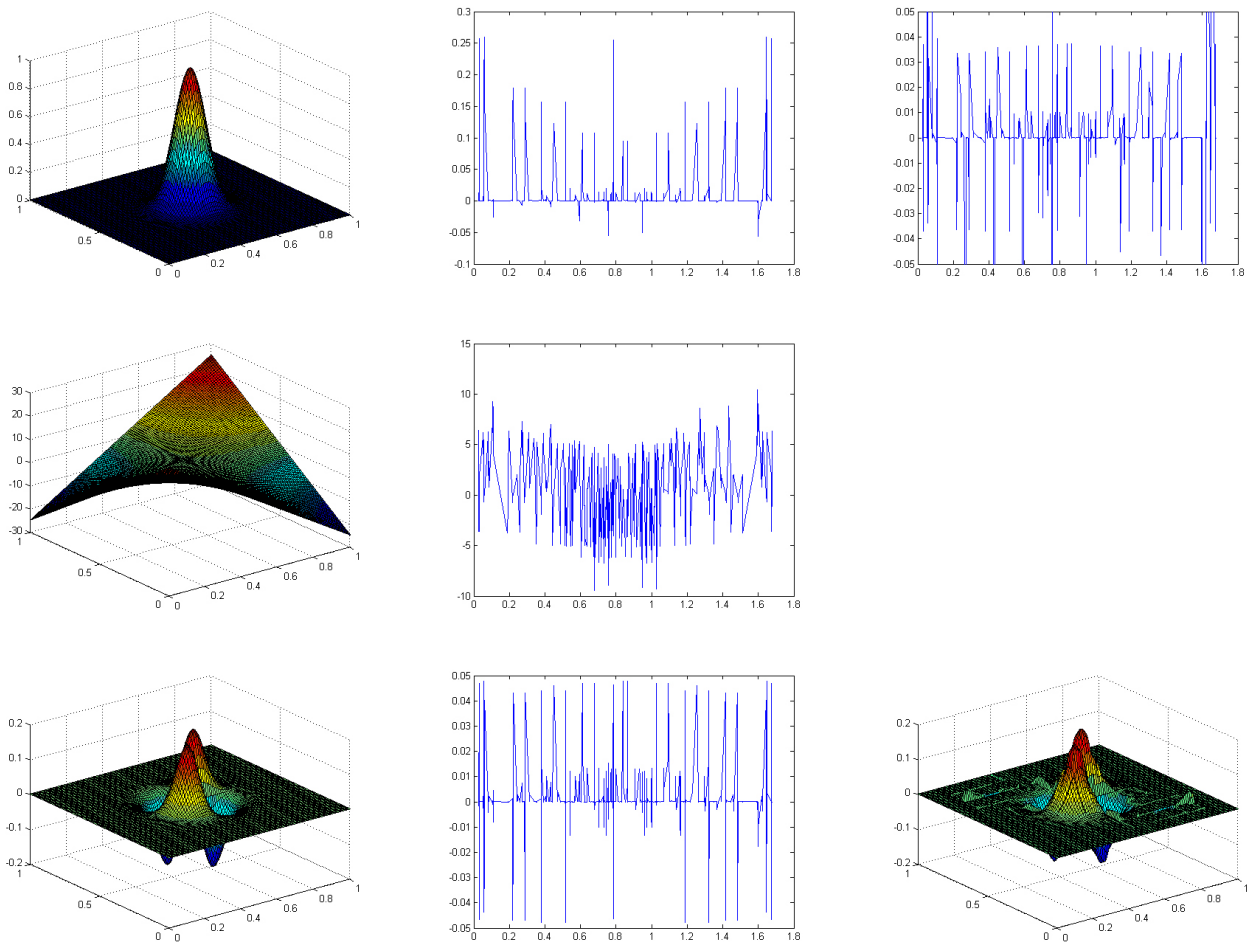


Figure 3.7: Multiplication

In order to shift a function f with g-function, g_f, a multiple of δx times along a chosen dimension, the following compositions are performed. Let $g_0 = g_f$, then

$$g_{N+1}(u_q) = g_N(u_q + dU_x(u_q)),\ N = 0, 1, 2, \ldots \tag{3.21}$$

Each composition shifts the graph by δx_i along x_i- axis. See Figure 3.9.

Proof. Given function f, its shifted version, f_1, along a given dimension x_i is written $f_1(x_1, \cdots, x_i + \delta x_i, \cdots) = f(x_1, \cdots, x_i, \cdots, x_n)$. Without loss of generality, let's look along

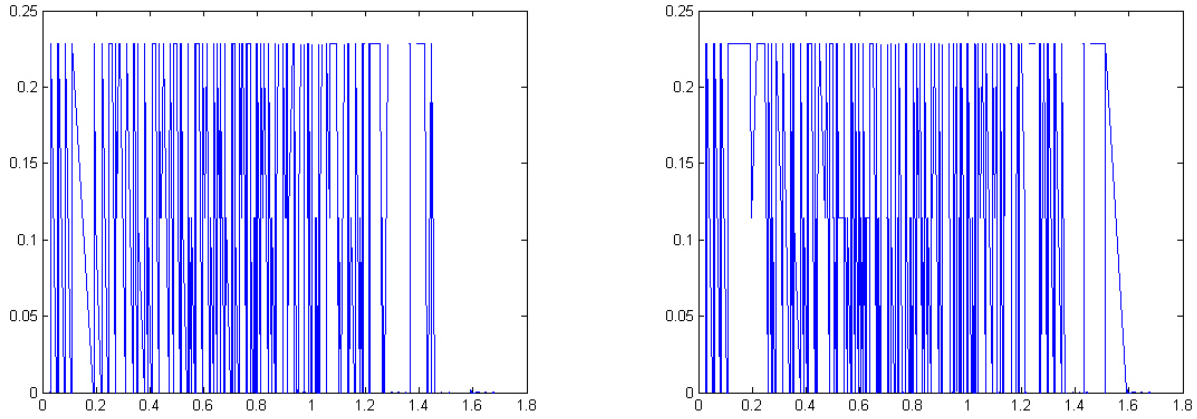


Figure 3.8: Shifting functions dU_x (left) and dU_y (right)

dimension x_1. We then look at the shift $f_1(x_1 + \delta x_1, \cdots, x_n) = f(x_1, \cdots, x_n)$.

For a chosen rank k, let f and f_1 have g-functions g_1 and g_f, respectively. Let the distance between the center of adjacent squares along dimension x_1 be δx_1. Let the left square be the jth square and the square to its right be $j+1$. The location of the center of square j is $M_j = \frac{\alpha^k_{qj} + \beta^k_{qj}}{2}$ for $q = 0, 1, \ldots, 2n$. And the center for square $j+1$ is $M_{j+1} = \frac{\alpha^k_{qj+1} + \beta^k_{qj+1}}{2}$. We assign $\delta x_1 = M_{j+1} - M_j = (2n+1)^{-k}$.

This definition of δx_1 is then the process of moving the value at the center of square $j+1$ to the center of square j. Actually this definition moves all the values over points in square $j+1$ to those in square j, but we only focus on the centers.

Now the compositions of the points at the center of squares j and $j+1$ are $u_{q,j} = \sum_{l,l\neq i} \lambda_l \varphi_q(x_l) + \lambda_1 \varphi_q(M_j)$ and $u_{q,j+1} = \sum_{l,l\neq i} \lambda_l \varphi_q(x_l) + \lambda_i \varphi_q(M_{j+1})$, respectively. Now

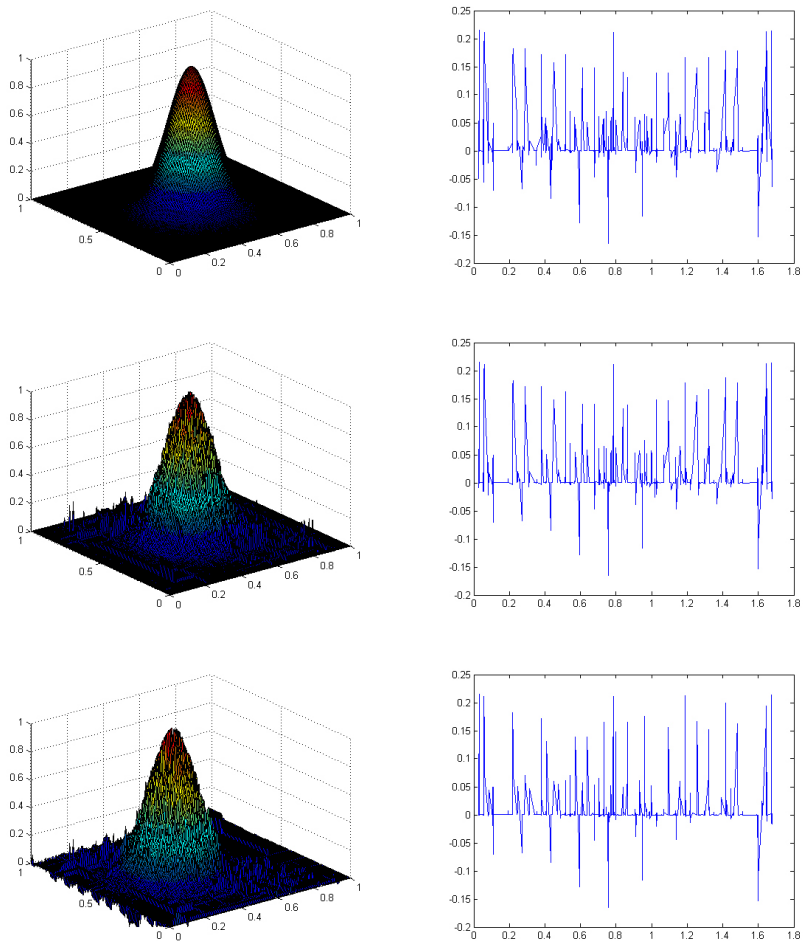


Figure 3.9: Shifting along x dimension: Top: (left) $f(x,y)$, (right) g_f Middle: (left) $f(x+0.01,y)$ (right) g_1 Bottom:(left) $f(x+0.1,y)$ (right) g_2

define $dU = u_{q,j+1} - u_{q,j}$, for $q = 0, 1, \ldots, 2n$. Note $dU = dU(u_{q,j})$. Thus

$$\begin{aligned} dU &= u_{q,j+1} - u_{q,j} \\ &= \left[\sum_{l,l\neq i} \lambda_l \varphi_q(x_l) + \lambda_1 \varphi_q(M_{j+1})\right] - \left[\sum_{l,l\neq i} \lambda_l \varphi_q(x_l) + \lambda_i \varphi_q(M_j)\right] \\ &= \lambda_i \varphi_q(M_{j+1}) - \lambda_i \varphi_q(M_j) \\ &= \lambda_i \left[\varphi_q(M_{j+1}) - \varphi_q(M_j)\right] \end{aligned} \tag{3.22}$$

Now, we want to assign the value at location M_{j+1} to location M_j. Thus, $f_1(M_j, \cdots, x_n) = f(M_{j+1}, \cdots, x_n)$. For their respective g-functions the relation is

$$g_1(u_{q,j}) = g_f(u_{q,j+1}) \tag{3.23}$$

for $q = 0, 1, \ldots, 2n$. From equation 3.22, we have $u_{q,j+1} = u_{q,j} + dU$. Substitution into equation 3.23, gives

$$g_1(u_{q,j}) = g_f(u_{q,j} + dU) \tag{3.24}$$

dropping j from the subscripts gives equation 3.20. We do this for each dimension giving n dU functions total.

3.2.3 Scaling

For two multidimensional functions f, f_1, we have for $f_1 = C \cdot f$, $g_1 = C \cdot g_f$ for $C > 0$. If $C > 1$, then the function is stretched. If $0 < C < 1$, the function is shrunk.

For scaling a function along a dimension x_i, we need to define another set of universal functions dV_q and dW_q.

There are n universal dV-functions, dV_{x_i}, n universal dW-functions dW_{x_i}, $i = 1 \ldots n$, such that for a variable $u_q \in \Delta_q^k$ we define $dV(u_q) = \varphi_q(\frac{1}{2}M_i) - \varphi_q(M_i)$ and $dW(u_q) = \varphi_q(2M_i) - \varphi(M_i)$. $M_i = \frac{\alpha_{qi}^k + \beta_{qi}^k}{2}$ is the midpoint of interval I_{qi}^k. dV is a stretching function and dW is a shrinking function.

For the two dimensional case we have,

$$\begin{aligned}
x_1 = x \quad & dV_x(u_q) = \varphi_q(\tfrac{1}{2}\tfrac{\alpha^k_{qi}+\beta^k_{qi}}{2}) - \varphi_q(\tfrac{\alpha^k_{qi}+\beta^k_{qi}}{2}) \\
x_2 = y \quad & dV_y(u_q) = \varphi_q(\tfrac{1}{2}\tfrac{\alpha^k_{qj}+\beta^k_{qj}}{2}) - \varphi_q(\tfrac{\alpha^k_{qj}+\beta^k_{qj}}{2}) \quad , \text{ for } u_q \in \Delta^k_{qij} \\
x_1 = x \quad & dW_x(u_q) = \varphi_q(2\tfrac{\alpha^k_{qi}+\beta^k_{qi}}{2}) - \varphi_q(\tfrac{\alpha^k_{qi}+\beta^k_{qi}}{2}) \\
x_2 = y \quad & dW_y(u_q) = \varphi_q(2\tfrac{\alpha^k_{qj}+\beta^k_{qj}}{2}) - \varphi_q(\tfrac{\alpha^k_{qj}+\beta^k_{qj}}{2})
\end{aligned} \tag{3.25}$$

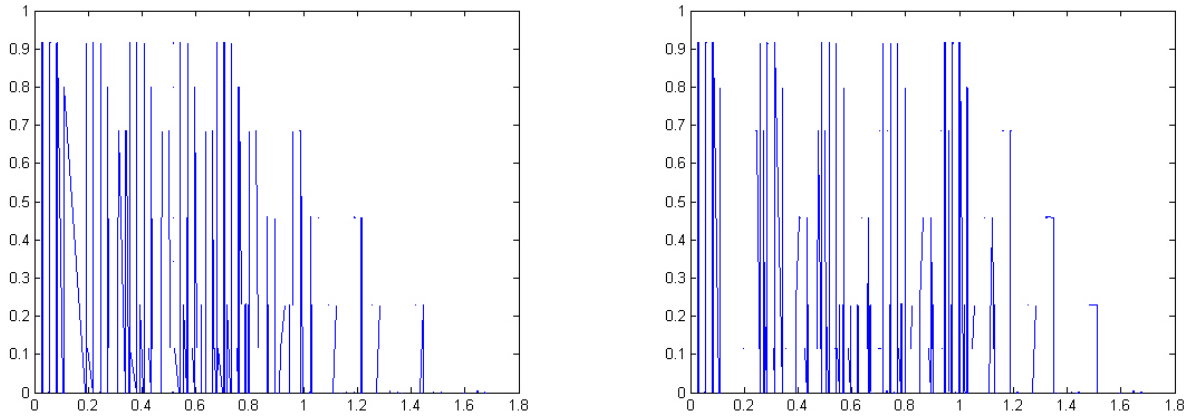


Figure 3.10: Scaling functions dW_x (left) and dW_y (right)

For a function f_1, f_2 , with g-functions g_1 and g_2 respectively, which are the scaled version of another function f , with g-function g_f, along one dimension by the amounts 2 and $\frac{1}{2}$, the following are their relationships respectively:

$$g_1(u_q) = g_f(u_q + dW_{x_i}(u_q)) \tag{3.26}$$

$$g_2(u_q) = g_f(u_q + dV_{x_i}(u_q)) \tag{3.27}$$

In order to scale a function f with g-function, g_f, a power of 2 along a chosen dimension,

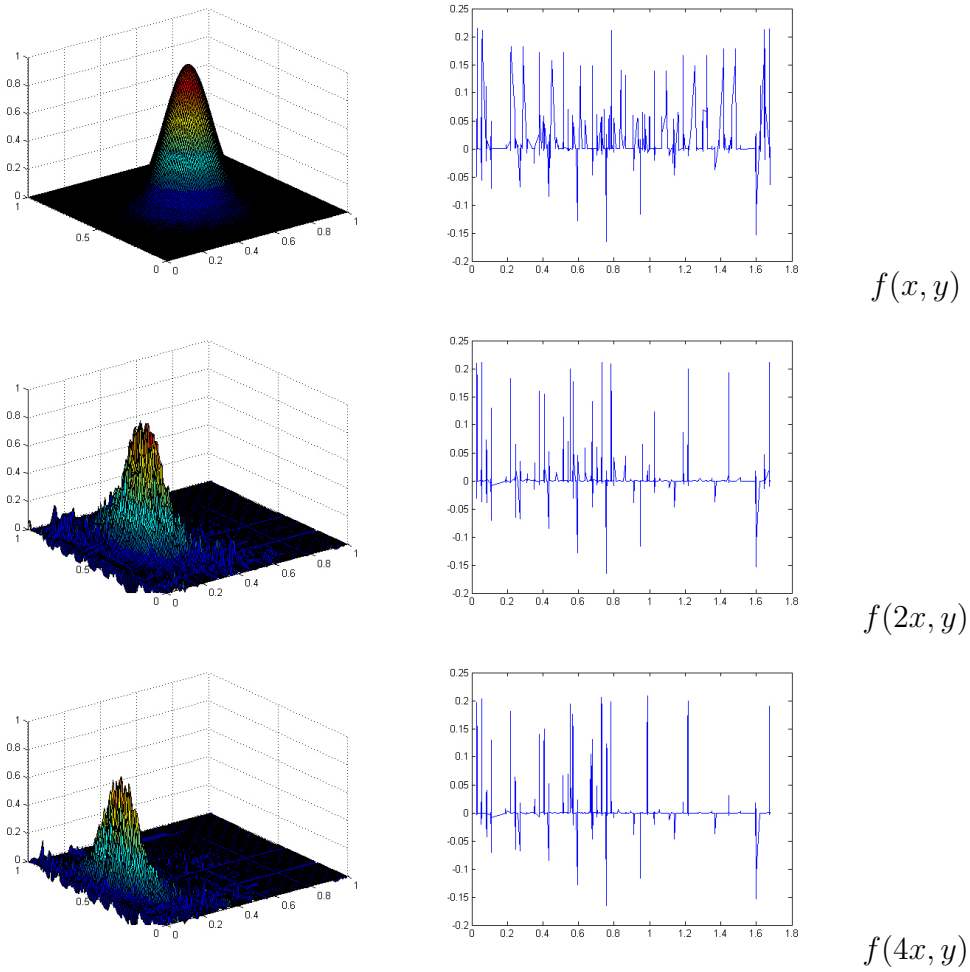


Figure 3.11: Scaling along x dimension

the following compositions are performed. Let $g_0 = g_f$, and let

$$g_{N+1}(u_q) = g_N(u_q + dW_x(u_q)),\ N = 0, 1, 2, \ldots \tag{3.28}$$

Each composition scales the graph by 2 along x_i-axis. For scaling the graph by powers of $\frac{1}{2}$ replace dW with dV. Scaling of a function is shown in Figure 3.11.

Proof. Given function f, its scaled version, f_1, along a given dimension x_i is written $f_1(x_1, \cdots, x_i + \delta x_i, \cdots) = f(x_1, \cdots, A{\cdot}x_i, \cdots, x_n)$, where $A = 2$ or $A = \frac{1}{2}$. Without loss

of generality, let's look along dimension x_1. We then look at the scaling $f_1(x_1, \cdots, x_n) = f(A{\cdot}x_1, \cdots, x_n)$.

For a chosen rank k, let f and f_1 have g-functions g_1 and g_f, respectively. Let the distance between the center of two squares along dimension x_1 be δx_1. Let the left square be the jth square and the square to its right be Aj. The location of the center of square j is $M_j = \frac{\alpha^k_{qj} + \beta^k_{qj}}{2}$ for $q = 0, 1, \ldots, 2n$. And the center for square Aj is $M_{Aj} = A{\cdot}\frac{\alpha^k_{qj+1} + \beta^k_{qj+1}}{2}$. We assign $\delta x_1 = M_{Aj} - M_j = (A-1){\cdot}M_j$.

Now the compositions of the points at the center of squares j and Aj are $u_{q,j} = \sum_{l,l\neq 1} \lambda_l \varphi_q(x_l) + \lambda_1 \varphi_q(M_j)$ and $u_{q,Aj} = \sum_{l,l\neq 1} \lambda_l \varphi_q(x_l) + \lambda_1 \varphi_q(M_{Aj})$, respectively. Now define $dU = u_{q,Aj} - u_{q,j}$, for $q = 0, 1, \ldots, 2n$. Note $dU = dU(u_{q,j})$. Thus

$$\begin{aligned} dU &= u_{q,Aj} - u_{q,j} \\ &= \left[\sum_{l,l\neq 1} \lambda_l \varphi_q(x_l) + \lambda_1 \varphi_q(M_{Aj})\right] - \left[\sum_{l,l\neq 1} \lambda_l \varphi_q(x_l) + \lambda_1 \varphi_q(M_j)\right] \\ &= \lambda_1 \varphi_q(M_{Aj}) - \lambda_1 \varphi_q(M_j) \\ &= \lambda_1 \left[\varphi_q(M_{Aj}) - \varphi_q(M_j)\right] \end{aligned} \tag{3.29}$$

Now, we want to assign the value at location M_{Aj} to location M_j. Thus, $f_1(M_j, \cdots, x_n) = f(M_{Aj}, \cdots, x_n)$. For their respective g-functions the relation is

$$g_1(u_{q,j}) = g_f(u_{q,Aj}) \tag{3.30}$$

for $q = 0, 1, \ldots, 2n$. From equation 3.29, we have $u_{q,Aj} = u_{q,j} + dU$. Substitution into equation 3.30, gives

$$g_1(u_{q,j}) = g_f(u_{q,j} + dU) \tag{3.31}$$

dropping j from the subscripts, letting $dU = dV$ for $A = 1/2$, and $dU = dW$ for $A = 2$ gives equation 3.26. We do this for each dimension giving n sets of dV, dW functions total.

3.2.4 Partial Differentiation

An approximation of the partial derivative of a multidimensional function in one variable, say x_i, is given as follows

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_1, \ldots, x_i + \delta x_i, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{\delta x_i} \tag{3.32}$$

Using the definitions of the shifting function explained in the previous section, we can define the one dimensional equivalent derivative. For a multidimensional function f, with g-function, g_f, the partial derivative is defined as follows:

$$g_1(u_q) = \frac{1}{\delta x_i} \left[g_f(u_q + dU_{x_i}(u_q)) - g_f(u_q) \right] \tag{3.33}$$

For the two dimensional case,

$$\begin{aligned} x_1 = x; \quad & g_1(u_q) = \tfrac{1}{\delta x} \left[g_f(u_q + dU_x(u_q)) - g_f(u_q) \right] \\ x_2 = y; \quad & g_1(u_q) = \tfrac{1}{\delta y} \left[g_f(u_q + dU_y(u_q)) - g_f(u_q) \right] \end{aligned} \tag{3.34}$$

Figure 3.12 shows the derivatives of f with respect to x and y.

We comment that the g-functions are in general of a very oscillating nature. They are not differentiable at all. Based on our numerical computations, divided differences of the g-functions are highly noised.

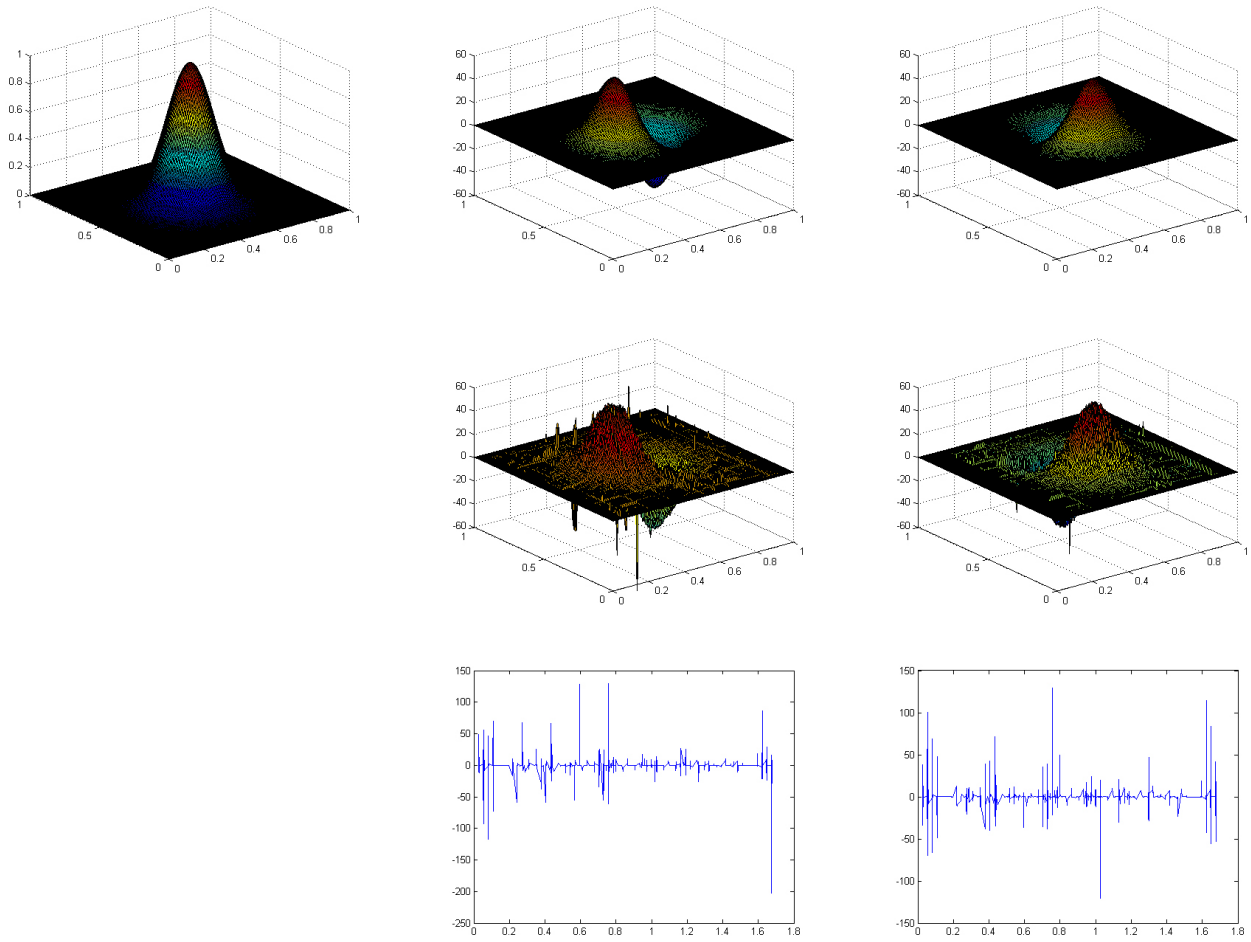


Figure 3.12: Partial Derivatives with respect to x y

In order to perform successive derivative approximations in one variable, the following compositions are carried out: Let $g_0 = g_f$, then

$$g_{N+1}(u_q) = \frac{1}{\delta x}\left[g_N(u_q + dU_x(u_q)) - g_N(u_q)\right], \; N = 0, 1, 2, \ldots \tag{3.35}$$

Extremum. For a multidimensional function f, the absolute maximum is given by $(n+1)\cdot \max(g_f)$. For the absolute minimum, the value is given by $(n+1)\cdot \min(g_f)$.

Proof. The n-dimensional cube S is a closed and bounded set, thus a function f defined on S attains a maximum (and minimum)value at some point in S. Let the maximum value be f_{max} at $(x_1, \cdots, x_n)$ and the minimum f_{min} at $(y_1, \cdots, y_n)$. The value $\frac{1}{n+1} f_{max}$ will be assigned to intervals containing $u_q = \sum_i \lambda_i \varphi_q(x_i)$ and the value $\frac{1}{n+1} f_{min}$ will be assigned to intervals containing $u_q = \sum_i \lambda_i \varphi_q(y_i)$. Now these values can be directly read from the range of g giving the maximum and minimum of f.

CHAPTER FOUR: KOLMOGOROV'S THEOREM FOR n DIMENSIONS ($n \geq 2$)

In order to deal with with functions of more than two variables, the proof by Lorentz [16] must be extended. In this chapter, some preliminary work and observations are provided which should lead to a useful implementation of Kolgomorov's Theorem in problems of high dimension. As indicated by Lorentz, the high dimensional case can be handled as in the 2-D case. Here we try to accomplish two objectives: 1. present a complete proof by working out all the needed modifications for the high dimensional case so that this chapter can be read independently. 2. Second, it allows us to write out all of the details in the proof that will be useful for our high dimensional problem's implementation.

4.1 Kolmogorov's Theorem for High Dimensions

For dimension $n \geq 2$, let $s = 2n + 1$. The statement of Kolmogorov's theorem is as follows:

Theorem 2 *[16] Let $I = [0, 1]$ and let S be the n-dimensional cube, $0 \leq x_p \leq 1$, $p = 1, \cdots, n$. There exists n constants $0 < \lambda_p \leq 1$, $p = 1, \cdots, n$, and $2n+1$ functions $\varphi_q(x), q = 0, \cdots, 2n$, defined on I and with values in I, which have the following properties: The φ_q are strictly increasing and belong to a class Lip α, $\alpha > 0$. For each continuous function f on S, one can find a continuous function $g(u)$, $0 \leq u \leq n$ such that*

$$f(x_1, \cdots, x_n) = \sum_{q=0}^{2n} g(\lambda_1 \varphi_q(x_1) + \cdots + \lambda_n \varphi_q(x_n)) \tag{4.1}$$

Let us first figure out the required modifications for the proof. Using the examples from the 2-D and 3-D cases, we see the following

- 2-D case:

$$\begin{aligned} I_{qi}^k &= I_i^k - 2q \cdot 10^{-k} = [\alpha_{qi}^k, \beta_{qi}^k] \\ S_{qij}^k &= I_{qi}^k \times I_{qj}^k, \end{aligned} \tag{4.2}$$

$q = 0, \ldots, 4;\ i = 0, 1, \ldots, 10^{k-1};\ k = 1, 2, 3, \ldots$

- 3-D case:

$$\begin{aligned} I_{qi}^k &= I_i^k - 2q \cdot 14^{-k} = [\alpha_{qi}^k, \beta_{qi}^k] \\ S_{qijl}^k &= I_{qi}^k \times I_{qj}^k \times I_{ql}^k, \end{aligned} \tag{4.3}$$

$q = 0, \ldots, 6 \quad i, j = 0, 1, \ldots, 14^{k-1}; \quad k = 1, 2, 3, \ldots$

The number intervals are related to the number of φ_q functions of which there are $s = 2n + 1$. There is also a pattern concerning the shifting property of the intervals. In the 2-D case the intervals are designed in such a way that when one shifts to say $q = 5$, these would be exactly the same intervals for $q = 0$. The same happens for the 3-D case. When one shifts the intervals for a function φ_q where $q = 7$, these would be exactly the same intervals as $q = 0$. Thus the following generalization for higher dimensional cases is proposed:

$$\begin{aligned} I_i^k &= \left[i \cdot (2s)^{-k+1} + (2s)^{-k}; i \cdot (2s)^{-k+1} + (2s-1) \cdot (2s)^{-k} \right] \\ \text{for} \quad k &= 1, 2, \ldots; \\ \text{and} \quad i &= 0, \ldots, (2s)^{k-1}; \end{aligned} \tag{4.4}$$

In order to maintain this cyclic behavior of shifting the intervals, one must also shift the intervals in such a way that the $(2n+1)^{th}$ φ_q function is the same as the φ_0. For the different q, the modification of intervals becomes:

$$I_{qi}^k = I_i^k - (s)^{-1}q(2s)^{-k+1} = I_i^k - 2q(2s)^{-k}$$

The intervals are shifted by : $2q(2s)^{-k}$

This formulation is consistent with the cases for $n = 2$ and $n = 3$. We also need to take note of the gap length. This is crucial in the construction of intervals of the second kind. In the 2-D case, the gap length between intervals is of length $2 \cdot 10^{-k}$. In the 3-D case the gap is of length $2 \cdot 14^{-k}$. So the gap is of length proportional to s^{-k}. The generalization of the gap is of length $2 \cdot s^{-k}$

4.1.1 Set up for the Proof

With the observations above and some to be presented in this section, we begin by defining the closed intervals

$$I_i^k = \left[i \cdot (2s)^{-k+1} + (2s)^{-k}; i \cdot (2s)^{-k+1} + (2s-1) \cdot (2s)^{-k}\right] \tag{4.5}$$

$$\text{for} \quad k = 1, 2, \ldots;$$

$$\text{and} \quad i = 0, \ldots, (2s)^{k-1};$$

The variable k will be called the rank of these intervals. All intervals I_i^k are contained in $I = [0, 1]$, except when $i = (2s)^{k-1}$. The functions φ_q are associated with the intervals

$$I_{qi}^k = [\alpha_{qi}^k, \beta_{qi}^k] = I_i^k - 2q \cdot (2s)^{-k} \text{ with } s = 2n + 1 \tag{4.6}$$

$$\text{for} \quad k = 1, 2, \ldots;$$

$$\text{and} \quad i = 0, \ldots, (2s)^{k-1};$$

which are obtained from the I_i^k by translation to the left by the distance $2q(2s)^{-k}$. (This distance depends on the rank k). For the intervals not entirely contained in $I = [0, 1]$, we replace them with $I \bigcap I_i^k$.

The intervals of rank k are of length $(2s)^{-k}(2s-2)$, and the gaps between them are $2(2s)^{-k}$. If we fix k, the intervals (4.6) form $(2n+1)$ families, corresponding to $q = 0, \ldots, 2n$. No one of these families covers I. However, each point $x \in I$ is covered by at least $2n$ of the $2n+1$ families, $q = 0, \ldots, 2n$.

Another useful remark is that the endpoints $\alpha_{qi}^l, \beta_{qi}^l$ of ranks $l < k$ are among the points of the form $m \cdot 2s^{-k+1}$; and that no end points of rank k (except 0, 1) are of this form. Therefore, none of the $\alpha_{qi}^k, \beta_{qi}^k$ different from $0, 1$ coincides with an endpoint of lower rank.

The φ_q will be strictly increasing. The most important property will be that of mapping the intervals I_{qi}^k of each given rank into small sets. It is not the smallness of measure that is important, but rather the fact that the images of the I_{qi}^k under the φ_q will be disjoint, not only for a fixed q, but for all q and all i (and a fixed rank k).

The exact formulation of the required property is given in the following lemma. Let $0 < \lambda_i < 1$ be linearly independent irrational numbers (over the rational field).

Lemma 3 *There exists $2s+1$ strictly increasing functions φ_q, $q = 0, \cdots, 2n$, defined on $[0, 1]$ and belonging to a class Lip α, $\alpha > 0$ such that for each fixed rank $k = 1, 2, \cdots$, the*

intervals

$$\Delta_q^k = [\sum_{i=1}^{n} \lambda_i \varphi_q(\alpha_{qj_i}^k), \sum_{i=1}^{n} \lambda_i \varphi_q(\beta_{qj_i}^k)] \text{ for } q = 0, \cdots, 2n; \tag{4.7}$$

$$\text{and} \quad j_i \in \{0, \cdots, (2s)^{k-1}\}; \quad \text{for} \quad i = 1, 2, \cdots, n$$

all are disjoint.

Proof. The functions φ_q will be constructed iteratively. We put $\alpha_{q0}^0 = 0$, $\beta_{q0}^0 = 1$, and define $0 \leq \varphi_q(0) < \varphi_q(1) < 1$, $q = 0, \cdots, 2n$ in such a way that all these values are rational and distinct. We continue this construction by induction on k. At the k^{th} step, we shall define each φ_q, $q = 0, \cdots, 2n$, at the end points $\alpha_{qi}^k, \beta_{qi}^k$ of all intervals (4.6) with the same q. For each $k = 1, 2, \cdots$ we shall take care to satisfy the following conditions:

i The function φ_q, $q = 0, \cdots, 2n$, is strictly increasing on the set of all points $\alpha_{qi}^l, \beta_{qi}^l$, $0 \leq i \leq (2s)^{l-1}$, $l \leq k$.

ii The slope of each segment of the polygonal line connecting the points $(x, \varphi_q(x))$, $x = \alpha_{qi}^l, \beta_{qi}^l$, $l \leq k$, is strictly smaller than s^k.

The proceeding properties involve each single function φ_q; the following properties involve all $2n + 1$ functions.

iii All values $\varphi_q(\alpha_{qi}^l), \varphi_q(\beta_{qi}^l)$, $q = 0, \cdots, 2n$, $0 \leq i \leq (2s)^{k-1}$, $l \leq k$, are rational and distinct.

iv The intervals (4.7) are disjoint.

Assume that the φ_q are defined at all endpoints $\alpha_{qi}^l, \beta_{qi}^l$ of ranks $l < k$ in such a way that conditions (i) to (iv) are satisfied. We shall explain how this definition can be extended to all points $\alpha_{qi}^k, \beta_{qi}^k$ in $(0,1)$ of rank k. First we construct a (not strictly) increasing extension of φ_q that satisfies (ii). Later we shall change this extension slightly so as to satisfy all the requirements (i) to (iv). The first part of the construction can be carried out independently for each q.

Let q be fixed. We denote by $\gamma = \gamma_q$ the points $\alpha_{qi}^l, \beta_{qi}^l$, $l < k$, where the function φ_q is already known. We have observed that each "new" point $\alpha_{qi}^k, \beta_{qi}^k$ in $(0,1)$ is different from each of the old points γ. The intervals I_{qi}^k cover $\frac{s-1}{s}$ of the total length of each interval $(m(2s)^{-k+1}, (m+1)(2s)^{-k+1})$, and the gaps between the intervals I_{qi}^k cover the remaining $\frac{1}{s}$ of this length. The same applies to each interval (γ, γ'). The intervals I_{qi}^k are of two kinds. The intervals of the first kind contain one point γ; we call it γ_{qi}^k, the interval of the second kind contains no points γ.

For each interval I_{qi}^k, we shall take $\varphi_q(\alpha_{qi}^k) = \varphi_q(\beta_{qi}^k) = \varphi_{qi}^k$. For the intervals of the first kind, we select φ_{qi}^k equal to $\varphi_q(\gamma_{qi}^k)$. To define φ_{qi}^k for the intervals I_{qi}^k of the second kind, we select the values of φ_q at the endpoints $\alpha_{qi}^k, \beta_{qi}^k$. Let γ, γ' (i.e. α, α' or β, β') be two adjacent points γ_q. We define φ_q at the end points of the intervals I_{qi}^k of the second kind contained in (γ, γ'). It is sufficient to determine the increase of φ_q on each gap (or part of a gap)(x, x') we assign an increase of φ_q, proportional to the length of (x, x'). Since the total length of gaps is $\frac{1}{s}(\gamma', \gamma)$, we must take

$$\varphi_q(x') - \varphi_q(x) = s\frac{x' - x}{\gamma' - \gamma}[\varphi_q(\gamma') - \varphi_q(\gamma)] \tag{4.8}$$

Thus $\varphi_q(x') - \varphi_q(x) < s^k(x' - x)$, and φ_q satisfies (ii).

From now on the construction involves all values of q. Small changes of the value of the φ_{qi}^k will not disturb the condition (ii), or the monotonicity of φ_q. We change slightly the φ_{qi}^k for the intervals of the second kind in such a way that al values φ_{qi}^k, $q = 0, \cdots, 2n$, $i = 0, \cdots, (2s)^{k-1}$, become rational and distinct. This is possible because, in view of (iii), the values $\varphi_{qi}^k = \varphi_q(\gamma_{qi}^k)$ for the intervals of the first kind already have these properties.

The last part of the kth step is an amendment of φ_q at the end points $\alpha_{qi}^k, \beta_{qi}^k$ of rank k, which lie in $(0, 1)$. Since λ_i are irrational numbers and linearly independent over the rational field, the values

$$\sum_{i=1}^{n} \lambda_i \varphi_{qj_i}^k,\ q = 0, \cdots, 2n,\ j_i \in \{0, 1, \cdots, (2s)^{k-1}\} \tag{4.9}$$

are distinct. Let $\epsilon > 0$ be so small that the 2ϵ-neighborhoods of the points (4.9) are disjoint. This allows us to define the remaining values of the φ_q. For each interval I_{qi}^k, we select $\varphi_q(\alpha_{qi}^k), \varphi_q(\beta_{qi}^k)$ in the ϵ-neighborhood of φ_{qi}^k in such a way that $\varphi_q(\alpha_{qi}^k) < \varphi_{qi}^k < \varphi_q(\beta_{qi}^k)$. (If one of the endpoints is 0 or 1, this inequality has to be changed correspondingly; for example, if $\beta_{qi}^k = 1$, it should read $\varphi_q(\alpha_{qi}^k) < \varphi_{qi}^k = \varphi_q(\beta_{qi}^k)$. It is clear that these selections can be made so that (i), (ii), and (iii) will be satisfied; (iv) will also hold because each interval (4.7) is contained in the 2ϵ-neighborhood of the corresponding point (4.9). This completes the kth step.

By induction, each function φ_q, is defined on the set A_q of the points $\alpha_{qi}^k, \beta_{qi}^k$, $0 \le i \le (2s)^{k-1}$, $k = 1, 2, \cdots$. Moreover, on A_q,

$$\varphi_q \in Lip\ \alpha, \alpha = \log_{2s} 2 \tag{4.10}$$

Indeed, let $x, x+h, h>0$ be two points of A_q. We may assume $h < \frac{1}{s}$. Let k be the integer that satisfies $2 \cdot 2s^{-k-1} \leq h < 2 \cdot 2s^{-k}$. The interval $(x, x+h)$ can contain at most one of the points $\alpha_{qi}^k, \beta_{qi}^k$. Hence, it is contained in an interval of length $(2s)^{-k+1}$ between some two such points. therefore by (i) and (ii), $\varphi_q(x+h) - \varphi_q(x) < s^k \cdot (2s)^{-k+1} = (2s) \cdot 2^{-k} = (2s) \cdot (2s)^{-\alpha k} \leq (2s)(sh)^\alpha = (2s) \cdot s^\alpha h^\alpha$ if α is defined by $2 = (2s)^\alpha$. This proves (4.10).

4.1.2 Further Details required for the Proof

Let I_{qi}^k be the intervals (4.7); let S be hypercube $0 \leq x_1, x_2, \cdots, x_n \leq 1$; and let $S_{qj_1j_2\cdots j_n}^k$ be little hypercubes:

$$S_{qj_1j_2\cdots j_n}^k = \prod_{i=1}^{n} I_{qj_i}^k \tag{4.11}$$

$$q = 0, \cdots, 2n;\ j_i \in \{0, \cdots, (2s)^{k-1}\}$$

The image of (4.11) under the map $(x_1, \cdots, x_n) \longrightarrow \sum_i^n \lambda_i \varphi_q(x_i)$ is the interval Δ_q^k of 4.7. Lemma 3 means, therefore, that for each fixed k, the functions $\sum_{i=1}^n \lambda_i \varphi_q(x_i)$ map hypercubes (4.11) with corresponding q onto disjoint intervals.

For each fixed k, the intervals I_{qi}^k form $(2n+1)$ families, $q = 0, \cdots, 2n$. We know that each point x in $[0,1]$ can fail to be covered only by one of the $(2n+1)$ families. The same applies to all intervals $I_{qj_i}^k$. Therefore each point $(x_1, x_2, \cdots, x_n)$ of S can fail to be covered by the cubes (4.11) only for at most n values of q. We have "at least $(n+1)$ hits at $(x_1, x_2, \cdots, x_n)$ out of $2n+1$ tries, $q = 0, \cdots, 2n$."

For each continuous function $g(u)$, defined for $0, \leq u \leq n$, we put

$$h(x_1, x_2, \ldots, x_n) = \sum_{q=0}^{2n} g\left(\sum_{i=1}^{n} \lambda_i \varphi_q(x_i)\right) \tag{4.12}$$

Lemma 4 *Let* $\frac{n}{n+1} < \theta < 1$. *For each* $f \in C[S]$ *there is a function* $g \in C[0, s]$ *such that*

$$\|f - h\| \leq \theta \|f\| \text{ and } \|g\| \leq \frac{1}{n+1}\|f\| \tag{4.13}$$

First, take rank k so large that the oscillation of f on the cubes (4.11) does not exceed $\epsilon\|f\|$ for some $\epsilon > 0$. Choose ϵ so small that $\frac{n}{n+1} + \epsilon \leq \theta$. Let $f^k_{qj_1j_2\cdots j_n}$ represent the value of $f(x_1, \cdots, x_n)$ in the center of the cube $S^k_{qj_1j_2\cdots j_n}$. On each interval Δ^k_q take $g(u)$ constant and equal to $\frac{1}{n+1} f^k_{qj_1j_2\cdots j_n}$. Next, $g(u)$ can be extended linearly in the gaps between the Δ^k_q. This produces a continuous function on $[0, n]$ that satisfies the second relation in (4.13).

Letting $(x_1, \cdots, x_n)$ be an arbitrary point in S, there are at least $(n + 1)$ values of q, where $(x_1, \cdots, x_n) \in S^k_{qj_1j_2\cdots j_n}$, for some $j_1, j_2, \cdots, j_n$. The corresponding terms of the sum in (4.12) have the values $\frac{1}{n+1} f^k_{qj_1j_2\cdots j_n}$, each of which is in the neighborhood of $\frac{1}{n+1} f(x_1, \cdots, x_n)$, with an error less than $\frac{1}{n+1}\epsilon\|f\|$. The two remaining terms of (4.12) are each $\leq \frac{1}{n+1}\|f\|$ in absolute value. It follows then that

$$|f(h, y) - h(x, y)| \leq (n+1)\frac{1}{n+1}\epsilon\|f\| + \frac{n}{n+1}\|f\| \leq \theta\|f\| \tag{4.14}$$

4.1.3 Proof of Theorem

This proof is for the case of dimension n. Let $f \in C[S]$ be given. We define a sequence of functions $g_r \in C[0,n]$ with the corresponding functions

$$h_r(x,y) = \sum_{q=0}^{2n} g_r \left(\sum_{i=0}^{n} \lambda_i \varphi_q(x_i) \right) \tag{4.15}$$

as follows: First let g_1, h_1 be given by Lemma 4 with

$$\|f - h_1\| \leq \theta \|f\|, \ \|g_1\| \leq \frac{1}{n+1} \|f\| \tag{4.16}$$

Applying Lemma 4 again and obtain functions g_2, h_2, while using $(f - h_1)$ instead of f, for which

$$\|(f - h_1) - h_2\| \leq \theta \|f - h_1\| \leq \theta^2 \|f\|, \|g_2\| \leq \frac{1}{n+1} \|f - h_1\| \leq \frac{1}{n+1} \theta \|f\| \tag{4.17}$$

In general the functions g_r, h_r will satisfy

$$\|f - h_1 \cdots - h_r\| \leq \theta^r \|f\|, \|g_r\| \leq \frac{1}{n+1} \theta^{r-1} \|f\|, r = 1, 2, \cdots \tag{4.18}$$

The series $\sum_1^\infty g_r$ is dominated by a convergent geometric series. So by Weierstrass M-test, it converges uniformly. Let g be the limit. Take the summation on both sides of (4.15) with respect to r from 1 to N:

$$\sum_{r=1}^{N} h_r(x,y) = \sum_{r=1}^{N} \sum_{q=0}^{2n} g_r \left(\sum_{i=0}^{n} \lambda_i \varphi_q(x_i) \right) \tag{4.19}$$

$$= \sum_{q=0}^{2n} \sum_{r=1}^{N} g_r \left(\sum_{i=0}^{n} \lambda_i \varphi_q(x_i) \right) \tag{4.20}$$

Letting $N \to \infty$ gives us,

$$\sum_{r=1}^{\infty} h_r(x,y) = \sum_{q=0}^{2n} \sum_{r=1}^{\infty} g_r \left(\sum_{i=0}^{n} \lambda_i \varphi_q(x_i) \right). \tag{4.21}$$

Also, note that, the first inequality implies $\sum_{r=1}^\infty h_r = f$ uniformly.

CHAPTER FIVE: NUMERICAL IMPLEMENTATION DISCUSSION

5.1 Algorithmic Outline of Kolmogorov's Theorem

We now discuss more details on the implementation of Kolmogorov's Theorem. Our main ideas come from Lorentz's proof of the Kolmogorov's theorem in [16, Chapter 12]. For simplicity, we discuss the case $n = 2$ and for functions defined in the unit square $[0, 1] \times [0, 1]$, *i.e.* functions $f(x, y)$ for $0 \leq x \leq 1$ and $0 \leq y \leq 1$.

5.1.1 Computation of φ's in two dimensional case

It is unlikely that the φ functions in (3.1) are available in explicit formulas. In order to use them, we have to develop some efficient algorithms to compute their numerical values.

When $n = 2$, there are five strictly increasing functions φ_q, $q = 0, 1, \ldots, 4$, according to Kolmogorov's theorem. These functions are computed iteratively as follows.

Procedure for constructing $\varphi_0, \ldots, \varphi_4$

Initialization

- *For* $k = 1, 2, \ldots$ *and* $i = 0, 1, \ldots, 10^{k-1}$, *define*
  - $\alpha_{qi}^k = i \cdot 10^{-k+1} + 10^{-k} - 2q10^{-k}$
  - $\beta_{qi}^k = i \cdot 10^{-k+1} + 9 \cdot 10^{-k} - 2q10^{-k}$
- *Let* $\alpha_{q0} = 0$ *and* $\beta_{q0}^0 = 1$

- *Choose different values for* $\varphi_q(0)$ *and* $\varphi_q(1)$ *such that* $0 < \varphi_q(\alpha_{q0}^0) < \varphi_q(\beta_{q0}^0) < 1$

For $k = 1, 2, \ldots$ *do the following*

- φ_q*'s are defined on* $\Gamma = \{\alpha_{qi}^l, \beta_{qi}^l\}$ *for all* $l < k$*. These are the "old points".*
- *Need to define* φ_q *at "new points":* $\alpha_{qi}^k, \beta_{qi}^k$
- *Let* γ *denote an arbitrary point from* Γ*.*
- *For* $i = 0, 1, \ldots, 10^{k-1}$*, consider two cases.*

  if $\alpha_{qi}^k < \gamma < \beta_{qi}^k$ *for some* $\gamma \in \Gamma$ ***then***

  – $\varphi_q(\alpha_{qi}^k) = \varphi_q(\gamma)$

  – $\varphi_q(\beta_{qi}^k) = \varphi_q(\gamma)$

  else

  (there is no γ *such that* $\alpha_{qi}^k < \gamma < \beta_{qi}^k$*)*

  – *find the smallest* γ' *such that* $\beta_{qi}^k < \gamma'$

  – *find the largest* γ *such that* $\alpha_{qi}^k > \gamma$

  – *collect all new points* α *and* β *contained in the interval* $[\gamma, \gamma']$

  – *if* $[\gamma, \gamma']$ *is a gap between two neighboring* $[\alpha, \beta]$ *intervals in* $[\gamma, \gamma']$*, define* $\varphi_q(x')$ *according to*

  $$\varphi_q(x) - \varphi_q(x') = 5\frac{x' - x}{\gamma' - \gamma}\left[\varphi_q(\gamma') - \varphi_q(\gamma)\right]$$

  end if

- *Final step (making the φ's strictly increasing). Pick a value for λ (irrational in theory)*

  - *List $\varphi_{qi}^k + \lambda\varphi_{qj}^k$ for all q, i, j (where $\varphi_{qi}^k = \varphi_q(\alpha_{qi}^k) = \varphi_q(\beta_{qi}^k)$)*

  - *Let d =minimum distance between these points. Let $\varepsilon = d/4$*

  - *In $(\varphi_{qi}^k - \varepsilon, \varphi_{qi}^k + \varepsilon)$ choose the values for $\varphi_q(\alpha_{qi}^k) < \varphi_{qi}^k < \varphi_q(\beta_{qi}^k)$.*

According to Chapter 3 (and Chapter 4 for higher dimensional cases), after k iterations, these φ functions will satisfy the following conditions:

1. The function φ_q is strictly increasing on the set of all points $\alpha_{qi}^l, \beta_{qi}^l$, $0 \leq i \leq 10^{l-1}$, $l \leq k$.

2. The slope of each line segment connecting the points $(\alpha_{qi}^l, \varphi_q(\alpha_{qi}^l))$ and $(\beta_{qi}^l, \varphi_q(\beta_{qi}^l))$ is strictly smaller than 5^k.

3. All values $\varphi_q(\alpha_{qi}^l)$ and $\varphi_q(\beta_{qi}^l)$ are distinct.

4. The intervals

$$\Delta_{qi}^k = \left[\varphi_q(\alpha_{qi}^k) + \lambda\varphi_q(\alpha_{qj}^k), \varphi_q(\beta_{qi}^k) + \lambda\varphi_q(\beta_{qj}^k)\right] \tag{5.1}$$

   are disjoint.

The key roles played by the φ-functions in the representation given by Kolmogorov's theorem is that the mappings $u = \varphi_q(x) + \lambda\varphi_q(y)$ $(q = 0, 1, \ldots, 4)$ transform the unit square to the interval (5.1) in a highly nonlinear way.

5.1.2 Training the g-function

Given a function f of n variables, one can train the g function in the following way.

- *Fix a value of k (which will depend on the dimension space and machine precision)*
- *For $i, j = 0, 1, \ldots, 10^{k-1}$, define $\alpha_{qij} = (\alpha^k_{qi} + \alpha^k_{qj})/2$, $\beta_{qij} = (\beta^k_{qi} + \beta^k_{qj})/2$, and $f_{qij} = f(\alpha_{qij}, \beta_{qij})$.*
  - *Assign g values: $g(\varphi_q(\alpha^k_{qi}) + \lambda\varphi_q(\alpha^k_{qi})) = \frac{1}{3} f_{qij}$, $g(\varphi_q(\beta^k_{qi}) + \lambda\varphi_q(\beta^k_{qi})) = \frac{1}{3} f_{qij}$, and let $h_g(x, y) = \sum_{q=0}^{4} g(\varphi_q(x) + \lambda\varphi_q(y))$*
  - *Stop if $\|f - h_g\|$ is small enough, otherwise, repeat steps with $f = f - h_g$ to obtain a new g-function.*
- *Output the sum of all g-functions.*

The advantage here is that a single variable function is being trained as opposed to a multivariate function. The main cost of computation is in the construction of the φ-functions. Fortunately, they can be done off-line and once they are constructed, they can be used in all representations as long as the data spaces have the same dimension.

Although the algorithm above trains the g-function by piecewise linear interpolation, there are many other possible function forms: polynomials, splines, rational functions, trigonometric (Fourier) series, wavelets, etc.

5.2 Issues related to implementation

5.2.1 Assigning φ_q values for $k > 0$

One of the main difficulties in assigning values to φ_q occurs in determining the smallest 2ϵ neighborhood around each point $\sum_i \lambda_i \varphi_q(x_i)$, so that they are distinct. In our implementation, the values from $\sum_i \lambda_i \varphi_{qi}^k$ are ordered on a number line. Next the smallest distance between each point is calculated. Once this value ϵ is determined, the final value used to make $\varphi_q(\alpha)$ distinct from $\varphi_q(\beta)$ will be $\frac{\epsilon}{4}$.

When the number of data points u_q grows large, a considerable amount of time is required to sort them. The main sorting algorithm used was quick sort. This is fine for dimension $s = 2$ and rank k up to 3. But for dimension $s > 3$, this poses serious time issues. For even this size we are sorting a list of about 50 million values.

We investigated a way to overcome this difficulty. This is presented in the statement that follows:

Let $P_2^k > P_1^k > 0$ where $P_m^k = \sum_{j=1}^n \lambda_j \varphi_{q_j i_j}^k$ be two points for rank $k > 1$. Let $d_k = |P_2^k - P_1^k|$. Then a useful smallest distance between any two points is given by d_k^*.

$$d_k^* = \lambda^* n \varphi^* \left(1 - \frac{\sqrt{\alpha_1}}{\alpha_1}\right) \tag{5.2}$$

Proof. Let $P_2^k = \alpha_k P_1^k$; $\alpha_k > 1$ Now let $d_k^* =$ the distance between points P_2^{k*}, P_1^{k*} the two closest points on rank k. In this way $d_k \geq d_k^*$ for all d_k.

Note $P_2^{k*} = \alpha_k^* P_1^{k*}$, $\alpha_k^* > 1$

So we have,

$$
\begin{aligned}
d_k^* &= |P_2^{k*} - P_1^{k*}| \qquad (5.3)\\
&= \left(1 - \frac{1}{\alpha_k^*}\right) |P_2^{k*}| \\
&\geq \left(1 - \frac{1}{\xi_k}\right) |P_2^{k*}| \\
&= \left(1 - \frac{1}{\xi_k}\right) |\sum_{j=1}^{n} \lambda_j \varphi_{q_j i_j}^k| \\
&\geq \left(1 - \frac{1}{\xi_k}\right) \lambda^* |\sum_{j=1}^{n} \varphi^*| \\
&= \left(1 - \frac{1}{\xi_k}\right) \lambda^* |n\varphi^*| \qquad (5.4)
\end{aligned}
$$

where

$$
\begin{aligned}
\lambda^* &= \min\{\lambda_j\} \\
\varphi^* &= \min\{\varphi_{q_j i_j}^k\}
\end{aligned}
$$

Now choose

$$
\epsilon_k = \frac{n\lambda^*\varphi^*}{4}\left(1 - \frac{1}{\xi_k}\right)
$$

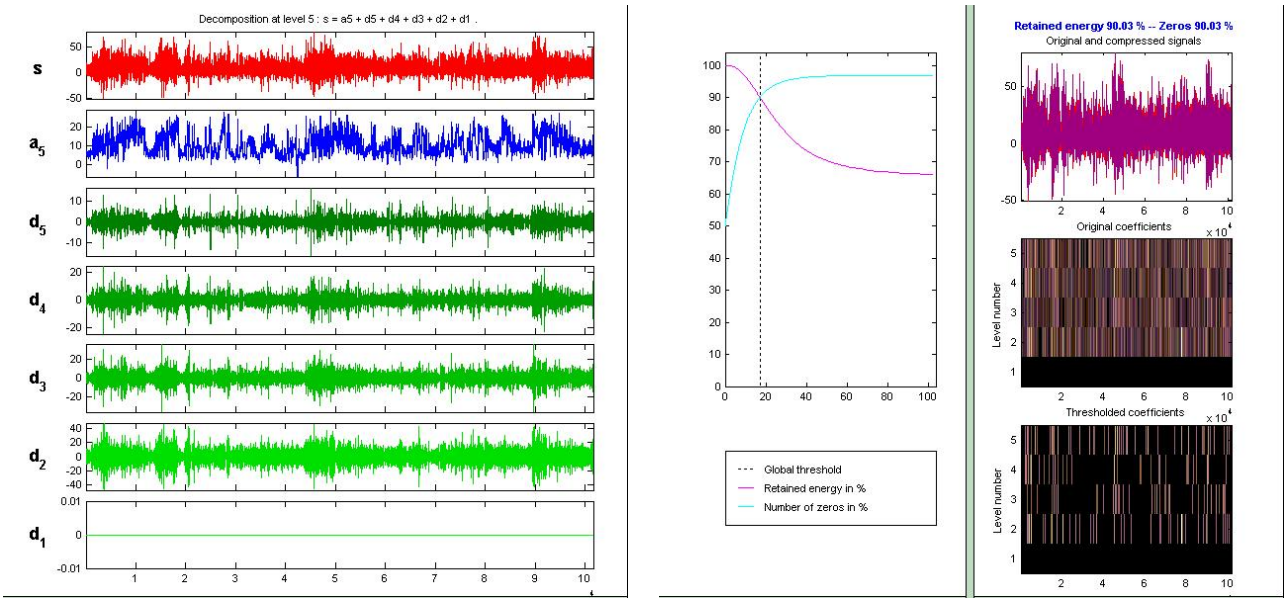
Note: For $w(\alpha_k) = \left(1 - \frac{1}{\alpha_k}\right)$, there exists a $\xi \epsilon [0, \alpha_k^*]$ such that $w'(\xi)\,(\alpha_k^* - 1) = w(\alpha_k^*) - w(1)$. So, $\xi = \sqrt{\alpha_k^*}$. Now choose ξ_k where $\xi = \sqrt{\alpha_k^*} \geq \sqrt{\frac{1}{\alpha_k^*}} > \sqrt{\frac{1}{\alpha_{k-1}^*}} = \xi_k$. ξ_k depends on the previous rank. So starting with α_1^* from rank $k = 1$, one can find a minimum distance for higher ranks and with the above choice of ϵ_k one can accomplish the task of making the u_q intervals disjoint without the necessity of sorting for each rank k.

5.2.2 Storing data points

Memory and storage requirements have also proven to be a hinderance to an efficient implementation of the algorithm. At one phase during experimentation, all data points were stored in type double format. This requires, depending on the machine, between 32 and 64 bits per data point. Our implementation utilized the 64 bit double type. The reason for doing so is that precision plays an important role in development of the g one dimensional function.

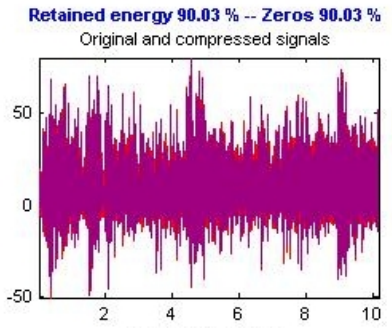
As per the theorem, one can easily calculate storage requirements for retaining the g function. Depending on dimension s and rank k the number of data points to store g goes as $(4s+2)^{3(k-1)} \times (2s+1)$. There are $2s+1$ q-functions and $(4s+2)^{3(k-1)}$ intervals. For dimension 3 and rank 3 one can already see that $14^6 \times 7 = 52706752$ data points are needed. With the double data type, this requires approximately 50 MB of storage. Essentially, the goal is to develop g functions over higher rank and higher dimension. In order to accomplish this a scheme must be developed such that all the data points for g will not be required.

We then turned to wavelet representation. Representation of the function as a wavelet allowed us to apply thresholds. With the thresholds we could reduce the number of coefficients required to represent the g-function but maintain a representation that is very similar to the original. This helped reduce the amount of data needed to store the function. As can be seen in figure 5.2, the reconstructed signal from the compressed wavelet is very similar to the original signal. Figure 5.1 shows the wavelet g-function decomposition and the wavelet signal compression interface screen.



(a) g-function decomposition (b) threshold applied to $g(u)$ function

Figure 5.1: tiger image



(a)superimposed g-function and its reconstruction from wavelet threshold function

Figure 5.2: tiger image

5.2.3 Computation times

There was little to be done with decreasing the required time for computation. This is the most significant issue to be addressed if there is to be the possibility of using this

algorithm in a realtime application. There are some authors [[1]] who proposed computing in a parallel computation environment as a solution. We have not been able to investigate this option extensively.

5.2.4 g-function representation

The format for storing the values are in Table (5.1). For the values listed in the following table, the parameter λ is equal to $\frac{1}{\sqrt{2}}$.

However, eventually we must deal with the truncation of the irrational values. This issue arises when using the values of ϕ_q and u_q in an application. Values are converted to a numerical data type such as double, which requires 64 bits. In this scenario, the values for ϕ_q and u_q are truncated. With the double data type, we can represent negative values from $-1.79769e{+}308$ to $-2.22507e{-}308$ and positive values from $2.22507e{-}308$ to $1.79769e{+}308$. This is tolerable since most of the data will not require this level of precision.

For example, in image analysis, we take the intensity of the image at pixel location (x, y) to be the value of $f(x, y)$. Also $f(x, y)$ would be considered piecewise linear between pixel locations. In images the pixel locations are integer pairs (x, y). All (x, y) pairs are re-normalized so that we are dealing with values of x and y where $0 \leq x, y \leq 1$. The composition $\phi_q(x) + \frac{1}{\sqrt{2}}\phi_q(y)$ will fall between two of the values u_q in the complete version of Table(5.2). During training of the g function, we focus only on the points (x, y) that fall into one of the S^k_{qij} cubes, then the g function value over that interval will be assigned the correct approximate (due to truncation) value, $\frac{1}{3}f(x, y)$. Since all values of g are constant over the interval u_q the table for the function g would be as in Table(5.3). This table will be used

γ	$\phi_q(\gamma)$
0	0
1/100	28302692683477/123629556035400
9/100	28303092025807/123629556035400
1/10	13249661/28937850
11/100	28319280710747/61814778017700
31/100	9461744492499/20604926005900
39/100	4730905524777/10302463002950
41/100	7104552465218/15453694504425
49/100	28418409532037/61814778017700
51/100	28451186244247/61814778017700
59/100	7112846478853/15453694504425
81/100	7137528848593/15453694504425
89/100	28550315065537/61814778017700
9/10	13373161/28937850
91/100	85436099526761/123629556035400
99/100	85436498869091/123629556035400
1	23/25

Table 5.1: Values for $\phi_0(x)$ function define over $[0, 1]$

u_{0ij}
$3864965751659/494518224141600 * 2^{1/2} + 3864965751659/247259112070800$
$3865764436319/494518224141600 * 2^{1/2} + 3865764436319/247259112070800$
$3996871285159/494518224141600 * 2^{1/2} + 3864965751659/247259112070800$
$3997669969819/494518224141600 * 2^{1/2} + 3865764436319/247259112070800$
$3864965751659/494518224141600 * 2^{1/2} + 3996871285159/247259112070800$
$3865764436319/494518224141600 * 2^{1/2} + 3997669969819/247259112070800$
$1376258939553/164839408047200 * 2^{1/2} + 3864965751659/247259112070800$
$1376525167773/164839408047200 * 2^{1/2} + 3865764436319/247259112070800$
$3996871285159/494518224141600 * 2^{1/2} + 3996871285159/247259112070800$
$3997669969819/494518224141600 * 2^{1/2} + 3997669969819/247259112070800$

Table 5.2: Sample points u_{qij} for $q = 0$ and $k = 2$ $\lambda_1 = 1$, $\lambda_2 = \frac{1}{\sqrt{2}}$

u_{0ij}	$g(u_{0ij})$
$3864965751659/494518224141600 * 2^{1/2} + 3864965751659/247259112070800$	$\frac{1}{3}f_{ij}^k$
$3865764436319/494518224141600 * 2^{1/2} + 3865764436319/247259112070800$	$\frac{1}{3}f_{qij}^k$
$3996871285159/494518224141600 * 2^{1/2} + 3864965751659/247259112070800$	$\frac{1}{3}f_{qij}^k$
$3997669969819/494518224141600 * 2^{1/2} + 3865764436319/247259112070800$	$\frac{1}{3}f_{qij}^k$
$3864965751659/494518224141600 * 2^{1/2} + 3996871285159/247259112070800$	$\frac{1}{3}f_{qij}^k$
$3865764436319/494518224141600 * 2^{1/2} + 3997669969819/247259112070800$	$\frac{1}{3}f_{qij}^k$
$1376258939553/164839408047200 * 2^{1/2} + 3864965751659/247259112070800$	$\frac{1}{3}f_{qij}^k$
$1376525167773/164839408047200 * 2^{1/2} + 3865764436319/247259112070800$	$\frac{1}{3}f_{qij}^k$
$3996871285159/494518224141600 * 2^{1/2} + 3996871285159/247259112070800$	$\frac{1}{3}f_{qij}^k$
$3997669969819/494518224141600 * 2^{1/2} + 3997669969819/247259112070800$	$\frac{1}{3}f_{qij}^k$

Table 5.3: Interpolation table for $(u_{qij}, g(u_{qij}))$, here the f_{qij}^k's are the values at the centers of the squares S_{qij}^k associated with u_{qij}^k

as an interpolation table when values from the g function are required for reconstruction or manipulation of the function $h(x, y) \approx f(x, y)$.

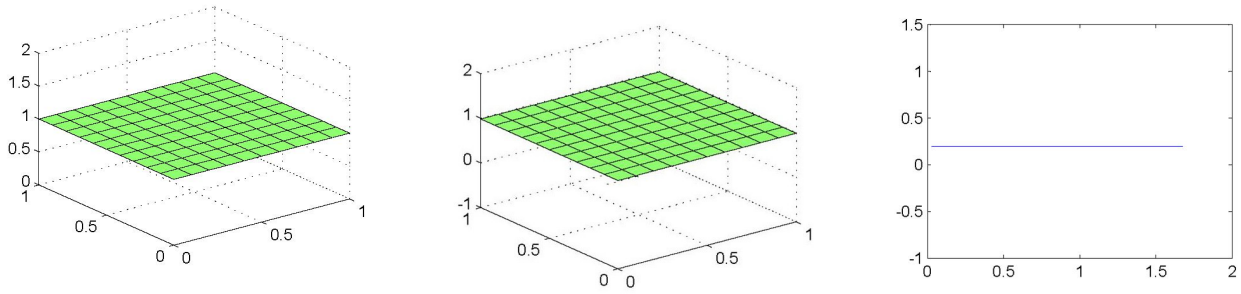
Table (5.3) was eventually converted into only double values. Possibly when using the table to interpolate $(u_q, g(u_q))$ some numerical approximation errors may be introduced. A more thorough investigation into the influence of changing from the exact value to double precision must be done.

CHAPTER SIX: APPLICATIONS

6.1 Image, Video, and MRI representation and reconstruction

Some simple experiments related to representation and compression are presented in this section. We use the basic formulation of Kolmogorov's theorem to represent images and video as one dimensional g- functions.

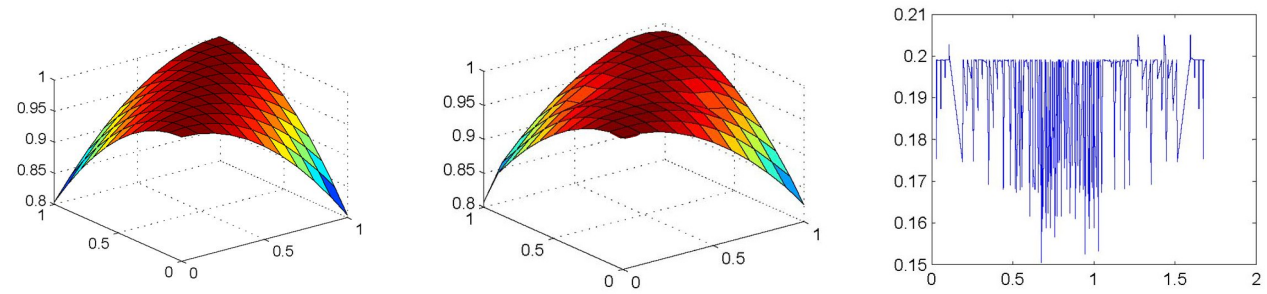
2-D Experiments. In this section, the application of Kolmogorov's theorem will be demonstrated. In Figures 6.1 to 6.3, there are three $f(x, y)$ functions presented, along with the associated $g(u)$ function, as well as the reconstructed approximation $h(x, y)$.



(a) original function (b) reconstruction (c) associated $g(u)$ function

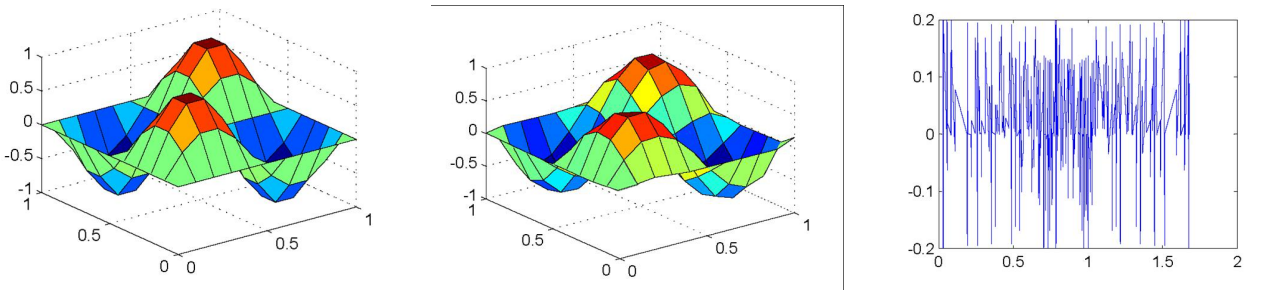
Figure 6.1: $f(x, y) = 1$

In another experiment, images are treated as functions of two variables. An image is a grid of $N \times M$ pixels grouped together. Each pixel contains some value which indicates the intensity (color) or some other property of the image. In this sense, images can be seen as a grid of (x, y) locations were the value at pixel (x, y) is its intensity, designated $f(x, y)$.



(a) original function (b) reconstruction (c) associated $g(u)$ function

Figure 6.2: $f(x, y) = \cos\left[\frac{2}{\pi}(x - y)\right]$

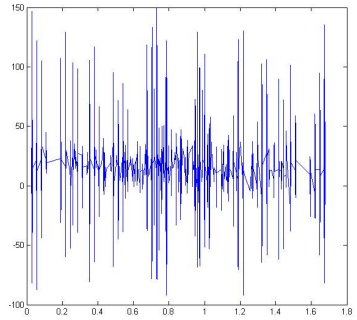


(a) original function (b) reconstruction (c) associated $g(u)$ function

Figure 6.3: $f(x, y) = \sin(2\pi x)\sin(2\pi y)$

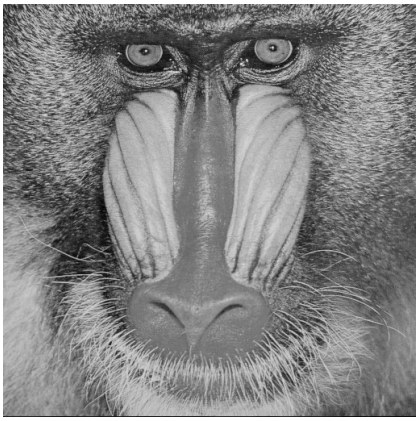
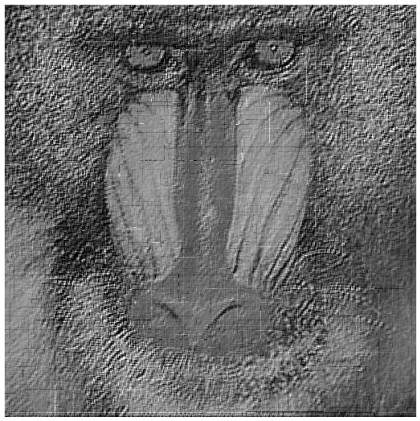
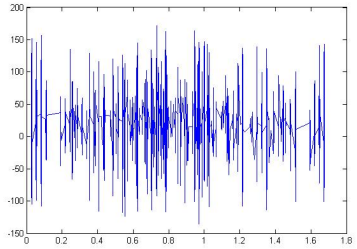
We can apply the Kolmogorov approximation to reconstruct an image from the trained g function. Using our computed φ functions, for each image f, we can construct its g–function. The φ for rank $k = 3$ are universal – once constructed they can be used over and over for any two dimensional functions. Our training of g–functions is very fast – each image takes seconds to train and even less time to be reconstructed. Figures 6.4 and 6.5 demonstrate an image on a x, y grid, along with the reconstruction using rank $k = 3$ and the associated g function .





(a) original image (b) reconstruction (c) associated $g(u)$ function

Figure 6.4: Image, g-function, reconstruction

(a) original image (b) reconstruction (c) associated $g(u)$ function

Figure 6.5: Image, g function, reconstruction

3-D Experiments Unlike in 2-D, verification by graphing functions is not straight forward in the 3-D case. However one may obtain some verification by using this method in applications involving three dimensional data. Two applications come to light when thinking about functions of three variables: MRI and video.

An MRI (or magnetic resonance imaging) scan is a radiology technique that uses magnetism, radio waves, and a computer to produce images of body structures. The MRI scanner is a tube surrounded by a giant circular magnet. The patient is placed on a moveable bed that is inserted into the magnet. The magnet creates a strong magnetic field that aligns the protons of hydrogen atoms, which are then exposed to a beam of radio waves. This spins the various protons of the body, and they produce a faint signal that is detected by the receiver portion of the MRI scanner. The receiver information is processed by a computer, and an image is produced.

In order to selectively image different voxels (volume picture elements) of the subject, orthogonal magnetic gradients are applied. Although it is relatively common to apply gradients in the principal axes of a patient (so that the patient is imaged in x, y, and z from head to toe), MRI allows completely flexible orientations for images. All spatial encoding is obtained by applying magnetic field gradients which encode position within the phase of the signal. In three dimensions (3D), a plane can be defined by "slice selection", in which an RF pulse of defined bandwidth is applied in the presence of a magnetic field gradient. Figure 6.6shows shows some perspective views of MRI images and the reconstruction of those perspective views. Figure 6.7 shows a portion of the associated g function.

One can think of video as a sequence of images. So the dimensions of the image gives two coordinates (x, y) while the third is given by time (t), incremented in units of 1. In this case, the value at a particular (x, y, t) coordinate represents intensity or color value at time t. The video is assumed to be a piecewise linear function, for simplicity. Figure 6.8 shows

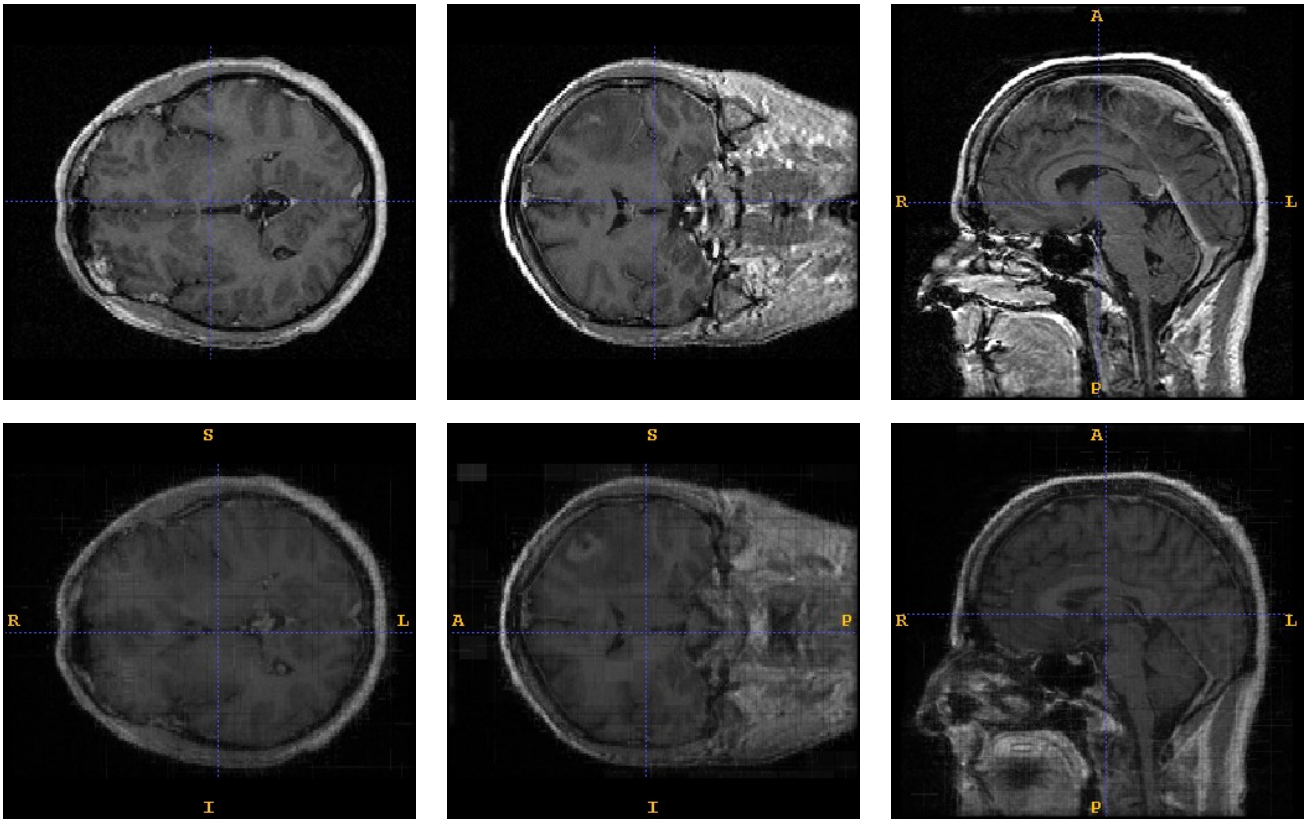


Figure 6.6: MRI image slices (top) and their reconstructions (bottom)

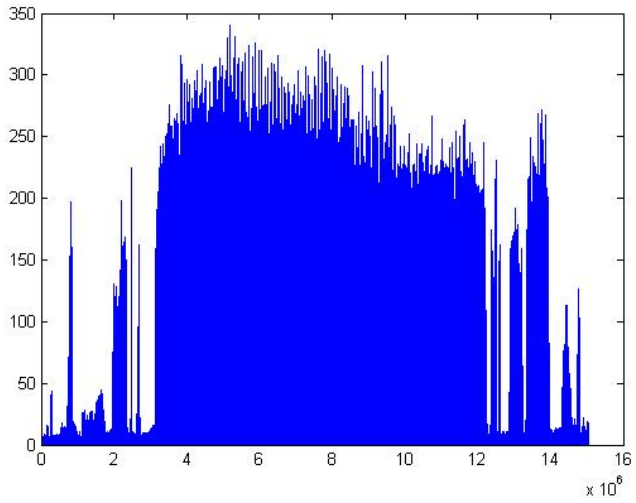


Figure 6.7: Portion of associated g function

the frames of an image sequence of a water drop falling and the reconstruction of the original water drop frames.

These experiments give preliminary verification of Kolmogorov's potential uses. Here we used a rank $k = 4$. Along with the properties of g-functions developed in an earlier chapter, one can potentially do high level analysis of multi-dimensional data via the one dimensional g-function.

6.2 Content Based Image Retrieval Application

Efficient ways for image and video retrieval has become a necessity for everyone who is dealing with the increasing quantities of digital media. Content-based image retrieval (CBIR), a technique for retrieving images from a database on the basis of automatically-derived features such as color, texture and shape, is becoming increasingly important in

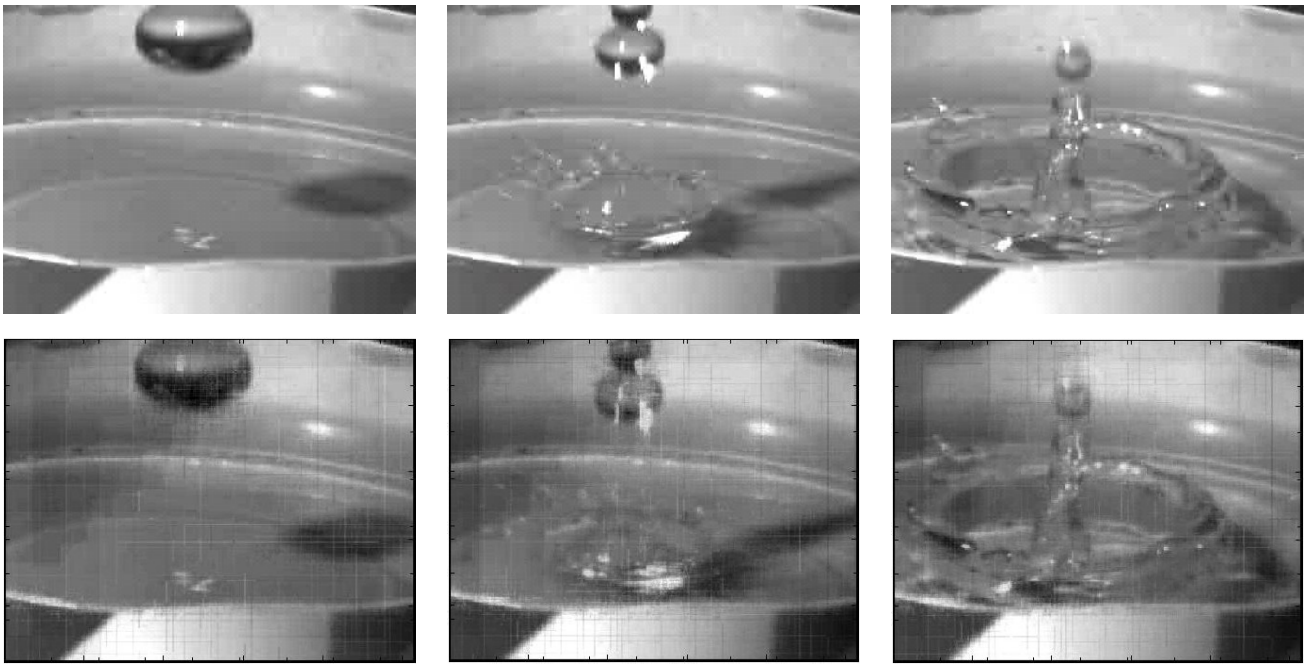


Figure 6.8: Frames in a video of water drop into a bucket and the reconstruction

visual information management systems [23]. Content-based image retrieval systems have traditionally used color and texture analysis. These analysis have not always achieved adequate level of performance and user satisfaction, particularly for the retrieval of images containing manmade objects. The growing need for robust image retrieval systems has led to a need for additional retrieval methodologies.

Subsets of this technique are collectively referred to as view-based approaches which tend to exploit the color, intensity or luminance information in an image. The basic idea is to use color or intensity information, which does not depend on the geometric shape of the objects. Most of the current view-based retrieval techniques analyze image data at a lower level on a strictly quantitative basis for color and texture features [18]. These techniques are geared towards retrieval on overall image similarity, especially for images containing natural objects

such as trees, vegetation, water, and sky.

An important issue in content-based image retrieval is effective indexing and fast searching of images based on visual features. Because the feature vectors of images tend to have high dimensionality and therefore are not well suited to traditional indexing structures, dimension reduction is usually used before setting up an efficient indexing scheme. One of the techniques commonly used for dimension reduction is principal component analysis (PCA). It is an optimal technique that linearly maps input data to a coordinate space such that the axes are aligned to reflect the maximum variations in the data. The QBIC system, one of the first content-based retrieval systems[6], uses PCA to reduce a 20-dimensional shape feature vector to two or three dimensions [20, 6]. In addition to PCA, many researchers have used Karhunen-Loeve (KL) transform to reduce the dimensions of the feature space. Although the KL transform has some useful properties such as the ability to locate the most important sub-space, the feature properties that are important for identifying the pattern similarity may be destroyed during blind dimensionality reduction [15]. Apart from PCA and KL transformation, neural network has also been demonstrated to be a useful tool for dimension reduction of features[5]. After dimension reduction, the multi-dimensional data are indexed. Most of these multi-dimensional indexing methods have reasonable performance for a small number of dimensions (up to 20), but explode exponentially with the increasing of the dimensionality and eventually reduce to sequential searching.

We also point out that content-based image retrieval methods themselves are a form of dimension reduction technique. Instead of directly dealing with the complete set of pixel

values in an image, content-based methods choose to concentrate on a smaller set of features. Our main idea is to first use the representation as guaranteed by the Kolmogorov Theorem to reduce the high dimensional objects into a function of a single variable and then apply the efficient 1-D methods on the 1-D representation. For example, the techniques used in the digital signal (or time series) of speech [19] can be applied to the image identification and retrieval problems once we transform the images into their 1-D representations using the Kolmogorov Theorem. Similarly, video sequences (3-D objects) can be transformed into their 1-D representation so that the techniques for 1-D sequences can be used.

To demonstrate our main ideas of using 1-D techniques to solve high dimensional problems, we apply the Kolmogorov theorem in constructing a simple image retrieval system. We want to show that our 1-D method as applied to the g-functions can yield comparable results as obtained using 2-D methods. We will not pay special attention in using more efficient search and matching techniques in our system for the moment.

The database retrieval system is implemented as follows.

Indexing and Storage

1. 2-D images are transformed to their 1-D representations
2. the 1-D representation is transformed by a 1-D wavelet transformation
3. the 1-D wavelet representation is compressed by threshholding.
4. the coefficients are stored in a vector as the image's index key

Retrieval

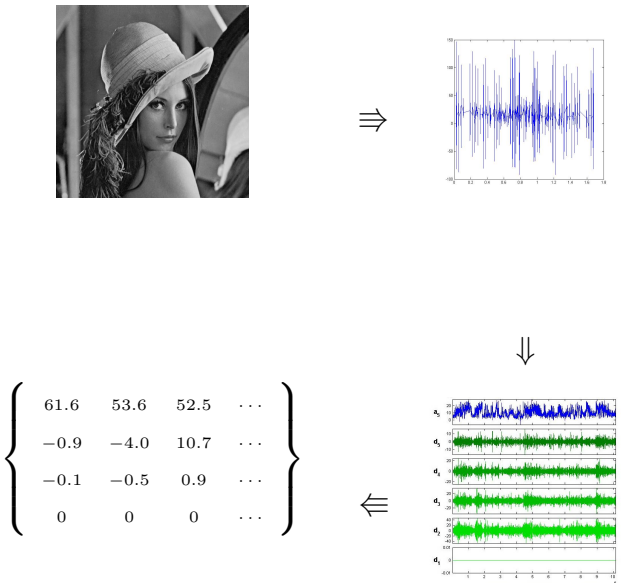


Figure 6.9: Indexing: All entries in the database will go through this step; a query image will go through the same step.

1. A test image is input as the key for the query
2. This 2-D image is converted to it's 1-D Kolmogorov representation
3. The 1-D representation is represented as a wavelet transformation
4. The wavelet coefficients are thresholded with the default threshhold value
5. Through a linear search, the coefficients are compared to entries in the database by a nearest neighbor match.

6. The nearest neighbors are returned as possible matches.

Again, our purpose is to show that 1-D methods can be directly used (to the Kolmogorov representation g function of the image in this case) to obtain comparable results. This opens the door for the applications of many well established techniques in 1-D domain in higher dimensional problems.

6.2.1 Results

We have performed some experiments using the above proposed simple scheme for image retrieval using the image database WANG ([2])that is created at the Pennsylvania State University. It contains about 1000 color images from the Corel database. The 1000 images are subdivided into 10 distinct classes: Africa, Beach, Horses, Dinosaurs, Flowers, Monuments, Food, Buildings, Buses, and Elephants. See Figure 6.10 for some samples of the various images from this database. For our experiments, we converted the images into black and

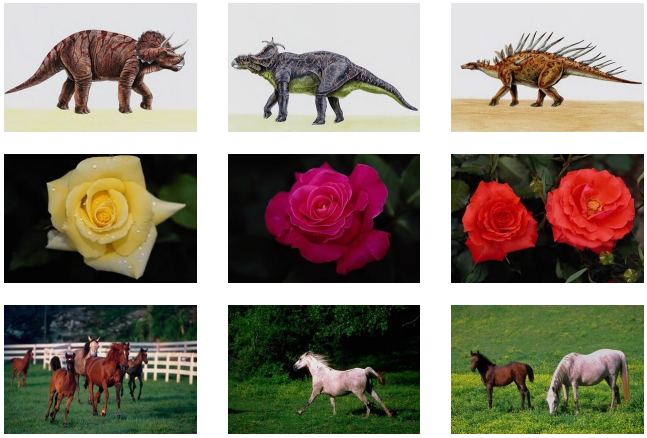


Figure 6.10: Various Images in WANG Database

white images for simplicity.

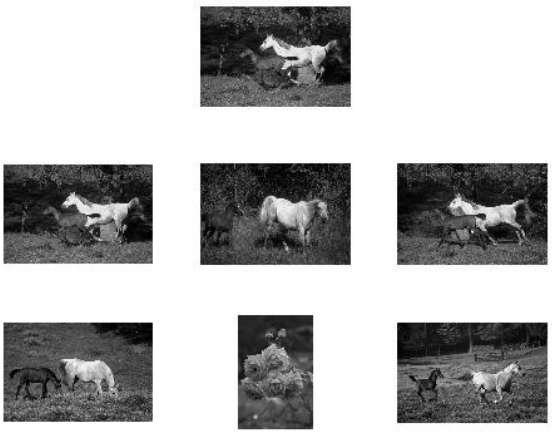


Figure 6.11: Top Image: Query Image. Six lower images are the best returned matches from the dataset.

Figure 6.12: Top Image: Query Image. Six lower images are the best returned matches from the dataset.

In Figure 6.11, our testing image is "horses". The returned images show the top six matches with one mismatch (a flower in the fifth match). In Figure 6.12, we copied the results from [12] that used the same query image for comparison (and there is a mismatch in the third returned image). We feel the results are comparable. Next, we show the results for other query images: one for "dinosaurs" (see Figure 6.13) and one for "flowers" (see Figure 6.14). Both return very good matches.

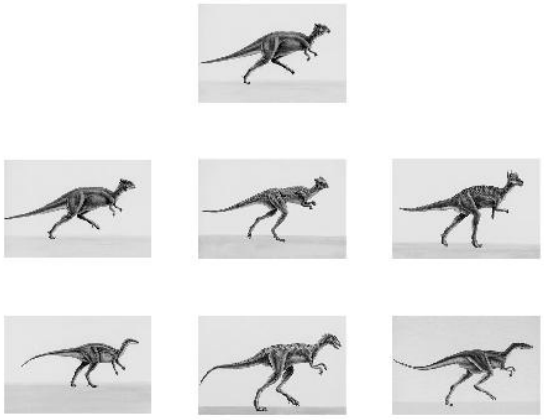


Figure 6.13: Top Image: Query Image (dinosaurs). Six lower images are the best returned matches from the dataset.

6.2.2 Video Retrievals

As one can see from the Kolmogorov theorem, if we transform each video sequence into its corresponding g function, we can then apply the same procedure to the g function as we did in the image retrieval. Our experiments for video retrieval is done on a subset of TRECVID database ([22]).

Figure 6.14: Top Image: Query Image (flowers). Six lower images are the best returned matches from the dataset.

6.2.3 Conclusion

Many image and video processing problems have been solved by developing certain extensions of the well established methods in the 1-D case. While in solving 1-D problems, we have powerful methods of time series analysis and digital signal processing. One challenging problem in dealing with images and videos is the high dimensionality. In the image and video retrieval problems, to avoid the high dimensionality, content-based methods are developed so that images are represented by a collection of their features that are enough to distinguish among themselves. But which features are important and must be included is still an open problem. In this application, we want to initiate the use of the Kolmogorov's theorem to solve high dimensional problems using 1-D method. We gave numerical implementation of the Kolmogorov's theorem that allowed us to represent any functions of higher

dimension by functions of one variable. We believe that the Kolmogorov's theorem can correctly "encode" and "automatically extract" the high dimensional information (features) in the "g functions". In other words, we are promoting the use of g functions as the features of images or videos. We used image and video retrieval as an example to show that 1-D methods can be used in dealing with 2-D and 3-D problems. Through a simple retrieval scheme, we demonstrated that our "1-D method" can produce comparable (if not better) results as the existing 2-D method. This proof-of-concept experiment may open the door for more advanced applications of the 1-D methods in solving higher dimensional problems in the future.

An immediate improvement we are working on is to add color in our treatment of the problems. We also need to find good examples of video sequences for our experiments.

6.3 Bayesian Prior Image Applications

Due to the uncertainty, many machine vision problems are solved based on some form of prior knowledge. In dealing with stereo, inpainting (*i.e.* filling in the missing image regions), denoising, and optical flow problems, for example, modeling image priors becomes necessary. This is challenging mainly due to the nature of high dimensionality of the data. One useful idea is to combine a large number of relatively simple probabilistic models in low dimensional subset of the data to obtain a global model. The simplest way of combining the simple models is by using summation.

This is exactly done in the approach of mixture of Gaussians in which a weighted sum of

individual Gaussians is employed. This is popular due to the fact that, in practice, it is easy to compute the sum by using the EM algorithm, and in theory, it can be proved that any distribution can be approximated as closely as possible by the sum of a sufficient number of Gaussians. But, in order to get a good fit, the number of Gaussians used in the mixture must be known and it is also well-known that the mixture of Gaussians method becomes very inefficient as the dimension of the data space gets larger. Another disadvantage of the mixture of Gaussians lies in the averaging and smoothing tendency of a weighted sum – the mixture cannot be sharper than any individual models [24].

An alternative way for combining individual models is by multiplication and renormalization. For example, high-dimensional distributions are approximated by the product of low-dimensional distributions. If the low-dimensional distributions are all Gaussians, then their product will be also a Gaussian. So, products of Gaussians cannot be used to approximate arbitrary non-Gaussian distributions, as noted by Hinton [10]. However, Hinton also found that by replacing the Gaussians by the so-called experts that are just a little bit more complicated with one or more additional hidden variables, multiplying the individual distributions together can still be a very powerful approach. This technique is called the Product of Experts (PoE).

One of the advantages of PoE is that it can produce much sharper distributions than the individual models in the product. Another feature of PoE is that the experts are obtained through training instead of being prescribed as in [7]. In order to deal with larger sized images, Roth and Black [4] applied the PoE technique in a framework of generic Markov

random field and developed a parametric representation that used data to train experts for local patches of images. The resulting approach is called the Field of Experts. Based on their FoE model, Roth and Black presented experimental results in image denoising and inpainting at the level of the current state-of-art.

However, both PoE and FoE methods face the difficulty of the computation of the partition function that is used for the renormalization of the product. It turns out that in solving the nonlinear problems where the gradient descent and quasi-Newton's methods are used, we need only to compute the average of the gradient with respect to the model distribution. Although the exact average of the gradient is hard to find, Hinton [11] shows that an approximate gradient of the contrastive divergence (CD) can be used as a safe substitute. Computer simulations show that this algorithm tends to converge, and to converge rapidly, although not always to the correct solution [26]. Yuille [25] related CD method to stochastic approximation and specified conditions under which the algorithm is guaranteed to converge.Futher reference on prior models can be found in [9][14][27].

In spite of the recent success in applications of PoE and FoE [21][13], so far, there is little theory to guide the construction and selection of the parametric representation of the experts. Why PoE and FoE work as seen in practice has not been theoretically justified. Without knowing the theoretical foundation of the methods, it is hard to come up with improvement. In this work, we want to provide a theoretical foundation for PoE and FoE and related mixture models methods using the function representation of Kolmogorov. Based on this observation, we want to propose a new method for estimation of the image prior.

Our proposed approach relies on the efficient numerical implementation.

6.3.1 Product of Experts

PoE uses a product to combine individual expert models as follows:

$$p(d|\theta_1 \cdots \theta_n) = \frac{\prod_m p_m(d|\theta_m)}{\sum_c \prod_m p_m(c|\theta_m)} \tag{6.1}$$

where d is a data vector, θ_m is the vector containing all parameters corresponding to the individual model m, $p_m(d|\theta_m)$ is the probability of d under model m, and the summation in the denominator runs c through all data vectors in the space. In order to fit a PoE model to a data set using an iterative method, one must compute the log likelihood of the observed vector d, under the PoE. This is given by :

$$\frac{\partial \log p(d|\theta_1 \cdots \theta n)}{\partial \theta_m} = \frac{\partial \log p_m(d|\theta_m)}{\partial \theta_m} - \sum_c p(c|\theta_1 \cdots \theta n) \frac{\partial \log p_m(c|\theta_m)}{\partial \theta_m}. \tag{6.2}$$

Unfortunately, even for small images d, the summation in (6.2) in this process is difficult to compute since it involves the unknown distribution p. To avoid this difficulty, Hinton [9] introduced a method of learning by minimizing contrastive divergence (the CD method). But the process still remains computationally expensive.

In image related learning, a choice of experts can be the combination of Student-t distribution and the linear filters that results in the Product of Student-t (PoT) of the form [17]

$$p(x) = \frac{1}{Z(\Theta)} \prod_{i=1}^{N} \Phi_i(J_i^T x; \alpha_i), \Theta = \{\theta_1, \theta_2, \cdots, \theta_n\} \tag{6.3}$$

where $\theta_i = \{J_i, \alpha_i\}$ and the experts have the form

$$\Phi_i(t, \alpha_i) = (1 + \frac{1}{2}t^2)^{-\alpha_i}, \tag{6.4}$$

where x represents the image, and J_i are the learned filters.

6.3.2 Field of Experts

Direct application of PoE to model image priors has the limitation on the image (or image patches) size. Black and Roth [4]-[21] overcame the problem by proposing the Field of Experts (FoE) method through combining ideas from sparse coding with Markov random field (MRF) models. The FoE frameworks breaks the image into overlapping cliques which has some probabilistic correlation then apply PoE to model the potentials of the MRF of image patches. The resulting formulation is as follows.

$$p(x) = \frac{1}{Z(\Theta)} \prod_k \prod_{i=1}^{N} \Phi_i(J_i^T x_k; \alpha_i), \tag{6.5}$$

where the parameters θ_i and the experts Φ_i are defined as before and x_k's represent the cliques in the image.

6.3.3 PoE and FoE revisted

We will place mixture and product models under the umbrella of Kolmogorov's theorem. If one considers $f(x_1, x_2, ..., x_n)$ as the prior probability distribution (or its logarithm) of interest, then the mixture and product models among others can be considered as very specific prior distribution (or simply function) approximations. There is a weaker version of Kolmogorov's theorem which is more suitable for explaining PoE and FoE models. Instead of

claiming the universality of the φ functions and constants λ's in (4.1), Kolmogorov's theorem implies the following: *For every function f of n variables, there are functions g_0, g_1, ..., g_N, and functions ψ_0, ψ_1, ...ψ_N, all of single variable, for some $N \leq 2n+1$, such that*

$$f(x) = \sum_{q=0}^{N} g_q \left(\sum_{i=1}^{n} \psi_q(x_i) \right). \tag{6.6}$$

This point of view is particularly useful if we only care about representing a special subset of functions (such as the PDF's in the cases of PoE and FoE) instead of the whole class of continuous functions.

Let us show how to cast PoE in the framework as given by the Kolmogorov's theorem. If p denotes the probability as given in (6.3) and (6.4), then define

$$f(x) := \log p(x) = \sum_{i=1}^{N} \log \Phi_i(J_i^T x; \alpha_i) - \log Z(\Theta). \tag{6.7}$$

So, (using index q to match the notation in (6.3))

$$f(x) = \sum_{q=1}^{N} \alpha_q \log(1 + \frac{1}{2}(J_q^T x; \alpha_i)^2) - \log Z(\Theta). \tag{6.8}$$

Comparing the right hand side with the representation in (6.6), we see that what the PoE tries to find is a special representation using linear ψ_q functions with $\psi_{iq}(x) = J_{iq}x$ and $g_q(u) = \alpha_q \log(1 + \frac{1}{2}u^2)$ for $q = 1, 2, \ldots, N$ and $g_0(u) = -\log Z(\Theta)$.

Similarly, taking the logarithm in (6.5), FoE model can be casted as a function representation model using linear functions for the φ functions in the representation (3.1) given in Kolmogorov's theorem:

$$\log p(x) = -\sum_{k} \sum_{i=1}^{N} \alpha_i \log(1 + \frac{1}{2}(J_i^T x_k; \alpha_i)^2) - \log Z(\Theta). \tag{6.9}$$

Finally, the mixture of Gaussians model is exactly when φ's are quadratic functions and g is a constant times a Gaussian of a single variable.

6.3.4 A new approach

Even though the above re-casting of PoE and FoE in the framework of Kolmogorov's theorem does not explain why the methods work, it does shed light on how to find alternative approaches. Let us start with the representation using the universal φ functions and constants λ's.

Let n be the dimension of the image size.

Step 1 Define n constants λ_i that are independent over the rational field $\mathbb{Q}$.

Step 2 Compute the $2n+1$ φ-functions.

Step 3 For every given image f, compute the g-function.

Step 1 can be accomplished easily in theory but in practice, due to the machine precision limitation, one can use carefully chosen rational numbers λ_i such that the relation

$$a_1\lambda_1 + a_2\lambda_2 + \cdots a_n\lambda_n = 0, \tag{6.10}$$

does not hold for any machine number unless all a_i's are equal to 0. Step 2 is most time consuming step and Step 3 is about evaluations of f which can be carried out in parallel if needed.

APPENDIX A: OVERVIEW OF FUNCTIONS

A.1 Function Overview

The following functions and the core functions for implementing and applying Kolmogorov's Theorem.

- **function build_phis_hp** - this function develops the φ_q functions needed to build the disjoint intervals on the real line. It calls the function find_first.

  Its inputs are:

  - kmax - the maximum rank for constructing the φ_q functions
  - ndimension - the dimensions space we are constructing the φ_q functions.
  - deltain - the user can set the spacing between $\varphi_q(0)$ and $\varphi_q(1)$ points
  - dprecin - allows the user to specify the number of decimals places to retain in calculations.
  - endpoints01 - the user can indicate the values at $\varphi_q(0)$ and $\varphi_q(1)$. This will override the values of deltain

  The outputs are:

  - phiq - contains all the $2n+1$ φ_q functions as an array of two column tables with columns γ and $\varphi_q(\gamma)$
  - u - contains the values for intervals which indicate the domain of g

- **function find_first** - this function determines which intervals are of the first or second kind. Refer to Lemma 1 in chapter 3, for a description of intervals of the first and second kind.

- **function train_gt** - this function builds the g- function and stores it as an interpolating table with columns u_q, $g(u_q)$.

- **function recon_ft** - this function is used to reconstruct the n-dimensional representation of function f according to the theorem. Its inputs are Its inputs are u and g and a vector representing the number of sample points for each dimensional direction. The output is a multidimensional array representing the reconstruction of the function f.

- **function eval_gt** - this function is used to determine f at one point from the g-function without having to reconstruct the entire f function first. Its inputs are u and g and vector $x = (x_1, \ldots, x_n)$. Its output is the value $f(x_1, \ldots, x_n)$.

APPENDIX B: FUNCTION CODE PRINTOUT

B.1 Code for building φ_q functions

B.1.1 build_phis_hp

```
0001 function [phiq, u ] = build_phis_hp2(kmax,ndimension,deltain,dprecin,endpoints01)
0002
0003 % hp stands for high precision .. using Symbolic toolbox to produce higher
0004 % k ranks
0005 %
0006 % input:
0007 %        required: kmax (scalar) = maximum rank to build functions
0008 %        optional: ndimension (scalar) = number of dimension variables
0009 %                                  ex n=2 f(x,y) //  n = 3 f(x,y,z)
0010 %                  delta (scalar) =  difference between phi_q endpoint
0011 %                                  values
0012 %                  dprec (scalar) = indicates how many decimal places
0013 %                                  to keep in
0014 %                           calculations
0015 %                  endpoints (matrix [2*ndimensions 2]) = predefined
0016 %                                     values for
0017 %                                     endpoints 0 & 1 . matrix must be
0018 %                                     given dimensions
0019 %
```

```
0020 % output:
0021 %        phi ([m,n,q] matrix array) = table of values used
0022 %               to interpolate values for phi(x)
0023 %               required to calculate  g(phi(x)+lamba* phi(y))
0024 %         u ([p,1] column vector)= table of values for which g will
0025 %               be defined. u specifies the domain of g
0026 % This function builds the phi_q functions and stores them in table phi
0027 % each phi will have the form
0028 % [point_value  interval_left_value ]
0029 % [ 0.1             7/32]
0030 % [ 0.9             9/32]
0031 %           .
0032 %           .
0033 %           .
0034 % [    etc             ]
0035 %
0036 % values will be in order sorted according to point_value;
0037 %
0038 % each u table will have the form
0039 % [ point_value    q i j]
0040 % [ phi(alpha_qik) + lamba* phi(alpha_qjk) ]
```

```
0041 % [ phi(beta_qik) + lamba* phi(beta_qjk)   ]
0042 %                    .
0043 %                    .
0044 %                    .
0045 % [             etc                  ]
0046 %
0047 % values will be in order sorted according to point_value;
0048
0049 %Initializations
0050 % construct non decreasing functions phi_q
0051 % define constants used in program
0052 %set the number of spacial dimensions to default 2-D f(x,y)
0053 if (nargin < 2)
0054     spacial_dimensions = 2; % number of dimensions for domain of function
0055 else
0056     spacial_dimensions = ndimension;
0057 end;
0058 qmax = 2*spacial_dimensions;
0059 % set value for delta
0060 if (nargin <3)
0061     delta = sym(1/64); % 32768);  % define small
```

```
0062 else
0063     delta = sym(deltain);
0064 end;
0065 % set maple precision to desired number of digits
0066 if (nargin < 4)
0067     dprec = 100;
0068     maple( ['Digits :=',char(sym(dprec))]);
0069 else
0070     dprec = dprecin;
0071     maple( ['Digits :=',char(sym(dprec))]);
0072 end;
0073 %define phi_q for k =0
0074 if (nargin < 5)
0075     phi = sym([0 92/100]);  % this is phi_0
0076     for q=1:qmax       % remember in this language indices start at 1
0077         phi(:,:,q+1) = phi(:,:,1) + (q)*delta;
0078     end;
0079 else
0080     % dimension should be  [2*ndimensions 2]; give error otherwise
0081     phi = sym(endpoints01);
0082 end;
```

```
0083
0084 final_value = sym(1);
0085
0086 lambda = sym([1 1/sqrt(2)]);
0087
0088 % now build phi_q  for all k according to theorem , one q at a time
0089 for k =1:kmax
0090     for q=0:qmax
0091         tble = findfirst_hp(spacial_dimensions,q,-k);
0092         tble=sortrowshp(tble,2);
0093         SIZETBLE= size(tble);
0094         temp = sym(zeros(1,2*(10^(k-1)+1))); %make the correct number
0095                                              %of endpoints
0096         % we need to assign values to intervals of the first kind
0097         if (SIZETBLE(1) > 0)
0098             for m=1:SIZETBLE(1)
0099                 if (tble(m,8) == 1) % this is an alpha endpoint
0100                                     %contained in I^k_i
0101                     pnt_k =double(2*(tble(m,2))); %this is done to handle
0102                                                   %k=0 gamma
0103                     pnt_l = double(2*(tble(m,6)));%points
```

```
0104                    l= double(tble(m,7));
0105                    temp(1, pnt_k+1) =phi(l+1,pnt_l+1,q+1);
0106                    temp(1, pnt_k+2) =phi(l+1,pnt_l+1,q+1);
0107                elseif (tble(m,8) == 9) %this is an beta endpoint
0108                                        % contained in I^k_i
0109                    pnt_k = double(2*(tble(m,2)));
0110                    pnt_l = double(2*(tble(m,6)));
0111                    l= double(tble(m,7));
0112                    temp(1, pnt_k+1) =phi(l+1,pnt_l+2,q+1);
0113                    temp(1, pnt_k+2) =phi(l+1,pnt_l+2,q+1);
0114                end;
0115            end;
0116        end;
0117        %now all intervals of the first kind have been assigned values
0118        % assign values to intervals of the second kind here
0119
0120        % create gammatable
0121        temp2 = sym([]);
0122        for lp=0:(k-1)
0123            for ip=0:(floor(10^(lp-1)))
0124                if (lp > 0)
```

```
0125                    alphaq = sym((10^(-lp))*(10*ip+1-2*q));
0126                else
0127                    alphaq = sym(0);
0128                end;
0129                temp2 = [temp2; alphaq ip lp 1 phi(lp+1,2*ip+1,q+1)];
0130                if (lp > 0)
0131                    betaq = sym((10^(-lp))*(10*ip+9-2*q));
0132                else
0133                    betaq = sym(1);
0134                end;
0135                temp2 = [temp2; betaq ip lp 9 phi(lp+1,2*ip+2,q+1)];
0136            end;
0137        end;
0138        gammatble= sortrowshp(temp2,1);
0139        SIZEGTBLE= size(gammatble);
0140
0141        %now assign values to interval i=0
0142        i=0;
0143        % check to see if interval is of the first kind
0144        found = false;
0145        if (SIZETBLE(1) > 0)
```

```
0146            w =1;
0147            while ((w <= SIZETBLE(1)) && (~found))
0148                if (tble(w,1) == q) && (tble(w,2) ==i) && (tble(w,3)==k)
0149                    found = true;
0150                end;
0151                w = w+1;
0152            end;
0153        end;
0154        if (~found) % it's not first kind, so assign value to i=0 interval
0155                    %according to theorem.
0156            j=1;    % it's second kind
0157            foundg = false;
0158            while ((j< SIZEGTBLE(1)) && (~foundg)) %look for gamma
0159                                                   %gamm_prime that
0160                gammaqil = gammatble(j,1);         %contains Iq0k
0161                gammapqil = gammatble(j+1,1);
0162                alphaqik = sym((10^(-k))*(10*i+1-2*q));
0163                betaqik = sym((10^(-k))*(10*i+9-2*q));
0164                expr1 = strcmpi(maple('evalb',[char(gammaqil), ' < ', ...
0165                        char(alphaqik)]), 'true');
0166                expr2 = strcmpi(maple('evalb',[char(alphaqik), ' <  ', ...
```

```
0167                          char(gammapqil)]), 'true');
0168                  expr3 = strcmpi(maple('evalb',[char(gammaqil), ' < ' , ...
0169                          char(betaqik)]), 'true');
0170                  expr4 = strcmpi(maple('evalb',[char(betaqik), ' < ', ...
0171                          char(gammapqil)]), 'true');
0172                  if ((expr1 && expr2) && (expr3 && expr4))
0173 %                 if ((( gammaqil < alphaqik) && (alphaqik < gammapqil)) &&
0174  %                        (( gammaqil < betaqik) && (betaqik < gammapqil)))
0175                      if (gammatble(j,4) ==1 )
0176                          phi_gamma =phi(double(gammatble(j,3))+1,2*...
0177                              double(gammatble(j,2))+1,q+1);
0178                      else
0179                          phi_gamma =phi(double(gammatble(j,3))+1,2*...
0180                              double(gammatble(j,2))+2,q+1);
0181                      end;
0182                      if (gammatble(j+1,4) ==1 )
0183                          phi_gammap = phi(double(gammatble(j+1,3))+...
0184                              1,2*double(gammatble(j+1,2))+1,q+1);
0185                      else
0186                          phi_gammap = phi(double(gammatble(j+1,3))+...
0187                              1,2*double(gammatble(j+1,2))+2,q+1);
```

```
0188                    end;
0189                    x = gammatble(j,1);
0190                    xp = alphaqik;
0191                    phi_x = phi(double(gammatble(j,3))+...
0192                        1,2*double(gammatble(j,2))+1,q+1);
0193                    temp(1,2*i+1)= phi_x +5 *...
0194                        ((xp-x)/(gammapqil -gammaqil))*...
0195                        (phi_gammap -phi_gamma);
0196                    temp(1,2*i+2) = temp(1,2*i+1);
0197                    foundg = true;
0198                end;
0199                if (~foundg)
0200                    j=j+1;
0201                end;
0202            end;
0203            if (~foundg) % if no interval was found then something is
0204                         %wrong
0205                disp('Interval gammas was not found');
0206            end;
0207        end;
0208        % assign values for remaining intervals i>0 according to theorem
```

```
0209        i =i+1;  % go to the next interval i, which is i =1
0210        move_on = false; % indicates whether to move to the next
0211                         %gamma, gammap interval
0212        j=1;
0213        while (i <= (10^(k-1)))
0214            if (j < SIZEGTBLE(1))        % this is mainly for
0215                                         %q=0 k =1 i =1 interval
0216                gammaqil = gammatble(j,1);
0217                gammapqil = gammatble(j+1,1);
0218            else
0219                gammaqil = gammatble(j,1);
0220                gammapqil = sym(2);
0221            end;
0222            alphaqik = sym((10^(-k))*(10*i+1-2*q));
0223            betaqik = sym((10^(-k))*(10*i+9-2*q));
0224            expr1 = strcmpi(maple('evalb', [char(gammaqil), ' < ', ...
0225                      char(alphaqik)]),'true');
0226            expr2 = strcmpi(maple('evalb', [char(alphaqik), ' < ', ...
0227                      char(gammapqil)]),'true');
0228            expr3 = strcmpi(maple('evalb', [char(gammaqil), ' < ', ...
0229                       char(betaqik)]),'true');
```

```
0230              expr4 = strcmpi(maple('evalb', [char(betaqik), ' <  ', ...
0231                          char(gammapqil)]),'true');
0232              if ((expr1 && expr2) && (expr3 && expr4))
0233 %              if ((( gammaqil < alphaqik) && (alphaqik < gammapqil)) &&
0234 %                      (( gammaqil < betaqik) && (betaqik < gammapqil)))
0235                  if (j < SIZEGTBLE(1))        % this is mainly for
0236                                               %q=0 k =1 i =1 interval
0237                      if (gammatble(j,4) ==1) % test to see if it's
0238                                              %alpha or beta
0239                          phi_gamma =phi(double(gammatble(j,3))+...
0240                              1,2*double(gammatble(j,2))+1,q+1);
0241                      else
0242                          phi_gamma =phi(double(gammatble(j,3))+...
0243                              1,2*double(gammatble(j,2))+2,q+1);
0244                      end;
0245                      if (gammatble(j+1,4) ==1) % test to see if
0246                                                %it's alpha or beta
0247                          phi_gammap = phi(double(gammatble(j+1,3))+...
0248                              1,2*double(gammatble(j+1,2))+1,q+1);
0249                      else
0250                          phi_gammap = phi(double(gammatble(j+1,3))+...
```

```
0251                            1,2*double(gammatble(j+1,2))+2,q+1);
0252                        end;
0253                    else
0254                        if (gammatble(j,4) ==1) % test to see if
0255                                                % it's alpha or beta
0256                            phi_gamma =phi(double(gammatble(j,3))+...
0257                                1,2*double(gammatble(j,2))+1,q+1);
0258                        else
0259                            phi_gamma =phi(double(gammatble(j,3))+...
0260                                1,2*double(gammatble(j,2))+2,q+1);
0261                        end;
0262                        phi_gammap = final_value; % can change to suit needs
0263                    end;
0264                    alphaqi_1k = sym((10^(-k))*(10*(i-1)+1-2*q));
0265                    betaqi_1k = sym((10^(-k))*(10*(i-1)+9-2*q));
0266                    % check to see which point is closer beta_i-1 or gamma
0267                    expr1 = strcmpi(maple('evalb', [char(gammaqil), ' <  ', ...
0268                                    char(betaqi_1k)]),'true');
0269                     if (expr1)
0270 %                    if (( gammaqil < betaqi_1k))
0271                        x = betaqi_1k; % beta_q(i-1)k
```

```
0272                    phi_x = temp(1,2*(i-1)+2);
0273                else
0274                    x = gammatble(j,1);
0275                    phi_x = phi_gamma;
0276                end;
0277                xp = alphaqik;
0278                temp(1,2*i+1)= phi_x +5 *((xp-x)/(gammapqil ...
0279                    -gammaqil))*(phi_gammap -phi_gamma);
0280                temp(1,2*i+2) = temp(1,2*i+1);
0281                move_on = false;
0282                i = i+1;
0283            else
0284                move_on = true;
0285                %check to see if it's an interval of the 1st kind
0286                w =1;
0287                found = false;
0288                while ((w <= SIZETBLE(1)) && (~found))
0289                    if (tble(w,1) == q) && (tble(w,2) ==i) && ...
0290                            (tble(w,3)==k)
0291                        found = true;
0292                    end;
```

```
0293                    w = w+1;
0294                end;
0295                if (found)
0296                    i= i+1;
0297                end;
0298
0299            end;
0300            if ((j < SIZEGTBLE(1)) && (move_on))
0301                j = j+1;
0302                move_on = false;
0303            end;
0304        end;
0305        % attach newly constructed rank to phi
0306        if (q==0) % only need to expand matrix once for q=0 for q> 0
0307                  %elements
0308            SIZEPHI = size(phi); % already exist
0309            SIZETEMP = size(temp);
0310            phi(k+1,SIZETEMP(2),q+1)= 0;
0311            phi(k+1,:,q+1) = temp;
0312        else
0313            SIZETEMP = size(temp);
```

```
0314            SIZEPHI = size(phi);
0315            if (SIZEPHI(2) > SIZETEMP(2)) % add space  if temp has
0316                                          %smaller dimensions
0317                temp(1,SIZEPHI(2)) = 0;
0318            end;
0319            phi(k+1,:,q+1) = temp;
0320        end;
0321        % create temporary u_q = sum (lambda_i *phi_q(x_i)) intervals
0322        %in order to determine epsilon
0323        Onesh = ones(1, (10^(k-1))+1);
0324        SIZEL = size(lambda);
0325        Top = phi(k+1,:,q+1);
0326        SIZETOP= size(Top);
0327        Z = zeros((SIZETOP(2)/2), SIZETOP(2));
0328        SIZEZ = size(Z);
0329        for z =1:SIZEZ(1)
0330            Z(z,2*z) = 1;
0331        end;
0332        Z = Z';
0333        Top = Top*Z;
0334        Side= Top';
```

```
0335         Sab = lambda(2)*Side*Onesh;
0336         SIZESAB = size(Sab);
0337         Tab = lambda(1)*ones(SIZESAB(1),1)*Top;
0338         Summ = Sab +Tab;
0339         SIZESUMM = size(Summ);
0340         NEWSIZE = SIZESUMM(1)*SIZESUMM(2);
0341         Side = reshape(Summ, NEWSIZE,1);
0342         if (SIZEL(2) >= 3)
0343             for l = 3:SIZEL(2)
0344                 Sab = Side*Onesh;
0345                 SIZESAB = size(Sab);
0346                 Tab = lambda(l)*ones(SIZESAB(1),1)*Top;
0347                 Summ = Sab +Tab;
0348                 SIZESUMM = size(Summ);
0349                 NEWSIZE = SIZESUMM(1)*SIZESUMM(2);
0350                 Side = reshape(Summ, NEWSIZE,1);
0351             end;
0352     end;
0353
0354
0355
```

```
0356     end;
0357
0358     T1 = sort(vpa(Side));
0359     disp(Side);
0360     disp(T1);
0361     T2 = circshift(T1,-1);
0362     dvec = sort(abs(T2-T1));
0363     dmin = dvec(1);
0364     disp(dvec);
0365     % assign epsilon to dmin/4 so that the 2-epsilon neighborhoods of the
0366     % points are disjoint
0367     epsilon = sym(maple('convert', ['evalf(', char(dmin/4),')' ] ,...
0368         'rational','10'));
0369 %    epsilon = vpa(dmin/4,WDTH);
0370     disp(['epsilon =' char(epsilon)]);
0371     for ip =0:(10^(k-1))
0372         phi(k+1,2*ip+1,:) = phi(k+1,2*ip+1,:) - epsilon;
0373         phi(k+1,2*ip+2,:) = phi(k+1,2*ip+2,:) + epsilon;
0374     end;
0375
0376
```

```
    clear maplemex;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           retrun phiq and u tables         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%create phi_q in a structure table
% put k =0 and k =1 points in table
for q =0:qmax
    for k=0:kmax
        if (k ==0)
            phiq{q+1}.table = sym([]);
            phiq{q+1}.table(1,1) = 0;
            phiq{q+1}.table(1,2) = phi(1,1,q+1);
            phiq{q+1}.table(2,1) = 1;
            phiq{q+1}.table(2,2) = phi(1,2,q+1);
        else
            %put k >0  into values into table
            for i =0:(10^(k-1))
                alphaqik = sym((10^(-k))*(10*i+1-2*q));
                betaqik = sym((10^(-k))*(10*i+9-2*q));
                expr1 = strcmpi(maple('evalb',[char(sym(0)), ' <=  ', ...
```

```
0398                            char(alphaqik)]), 'true');
0399                 expr2 = strcmpi(maple('evalb',[char(alphaqik), ' <= ', ...
0400                            char(sym(1))]), 'true');
0401                 if (expr1 && expr2)
0402                 %if ((0<= alphaqik) && (alphaqik <= 1))
0403                     phia = phi(k+1,2*i+1,q+1);
0404                     phiq{q+1}.table = [phiq{q+1}.table ; alphaqik  phia];
0405                 end;
0406                 expr1 = strcmpi(maple('evalb',[char(sym(0)), ' <= ', ...
0407                            char(betaqik)]), 'true');
0408                 expr2 = strcmpi(maple('evalb',[char(betaqik), ' <= ', ...
0409                            char(sym(1))]), 'true');
0410                 if (expr1 && expr2)
0411                 %if ((0<= betaqik) && (betaqik <= 1))
0412                     phib = phi(k+1,2*i+2,q+1);
0413                     phiq{q+1}.table = [phiq{q+1}.table ; betaqik  phib];
0414                 end;
0415             end;
0416         end;
0417     end;
0418     temp = sortrowshp(phiq{q+1}.table,1);
```

```
0419     phiq{q+1}.table = temp;
0420     temp= [];
0421 end;
0422 % build u table
0423 k = kmax;
0424 u=sym([]);
0425 for q=0:qmax
0426     for i =0:(10^(k-1))
0427         alphaqik = sym((10^(-k))*(10*i+1-2*q));
0428         betaqik =  sym((10^(-k))*(10*i+9-2*q));
0429         for j =0:(10^(k-1))
0430             alphaqjk = sym((10^(-k))*(10*j+1-2*q));
0431             betaqjk =  sym((10^(-k))*(10*j+9-2*q));
0432             phia =0;
0433             phib = 0;
0434             expr1 = strcmpi(maple('evalb',[char(betaqik),' <  ', ...
0435                         char(sym(0))]), 'true');
0436             expr2 = strcmpi(maple('evalb',[char(alphaqik), ' > ' , ...
0437                         char(sym(1))]), 'true');
0438             expr3 = strcmpi(maple('evalb',[char(betaqjk), ' < ', ...
0439                         char(sym(0))]), 'true');
```

```
0440            expr4 = strcmpi(maple('evalb',[char(alphaqjk), ' > ', ...
0441                        char(sym(1))]), 'true');
0442            expr5 = strcmpi(maple('evalb',[char(sym(0)), ' <  ' ,  ...
0443                        char(alphaqik)]), 'true');
0444            expr6 = strcmpi(maple('evalb',[char(alphaqik), ' <  ', ...
0445                        char(sym(1))]), 'true');
0446            expr7 = strcmpi(maple('evalb',[char(sym(0)), ' < ', ...
0447                           char(betaqik)]), 'true');
0448            expr8 = strcmpi(maple('evalb',[char(betaqik), ' <  ' , ...
0449                           char(sym(1))]), 'true');
0450            expr9 = strcmpi(maple('evalb',[char(sym(0)),' < ', ...
0451                           char(alphaqjk)]), 'true');
0452            expr10 = strcmpi(maple('evalb',[char(alphaqjk), ' < ' , ...
0453                           char(sym(1))]), 'true');
0454            expr11 = strcmpi(maple('evalb',[char(sym(0)), ' < ', ...
0455                           char(betaqjk)]), 'true');
0456            expr12 = strcmpi(maple('evalb',[char(betaqjk), ' < ', ...
0457                         char(sym(1))]), 'true');
0458            if ((expr1) || (expr2) || (expr3) || (expr4))
0459 %            if ((double(betaqik) < 0) || (double(alphaqik) > 1) || ...
0460 %                   (double(betaqjk) < 0) || (double(alphaqjk) > 1))
```

```
0461                %do nothing ; this is case were square is
0462                %completely outside
0463                %(0,1) x (0,1)
0464            elseif (((expr5) && (expr6)) && ((expr7) && (expr8)) && ...
0465                    ((expr9) && (expr10)) && ((expr11) && (expr12)))
0466 %                ((((0< double(alphaqik)) && (double(alphaqik) < 1)) ...
0467 %                    && ((0< betaqik) && (betaqik < 1))...
0468 %                    && ((0< alphaqjk) && (alphaqjk < 1))...
0469 %                    && ((0< betaqjk) && (betaqjk < 1)))
0470                % completely inside (0,1) x (0,1)
0471                phiai = phi(k+1, 2*i+1, q+1);
0472                phiaj = phi(k+1, 2*j+1, q+1);
0473                phibi = phi(k+1, 2*i+2, q+1);
0474                phibj = phi(k+1, 2*j+2, q+1);
0475                phia = phiai + lambda(2)* phiaj;
0476                phib = phibi + lambda(2)*phibj;
0477                u = [u; phia q i j];
0478                u = [u; phib q i j];
0479            else
0480                % partially inside (0,1) x (0,1)
0481                phiai = phi(k+1, 2*i+1, q+1);
```

```
0482                phiaj = phi(k+1, 2*j+1, q+1);
0483                phibi = phi(k+1, 2*i+2, q+1);
0484                phibj = phi(k+1, 2*j+2, q+1);
0485                expr1 = strcmpi(maple('evalb', [char(alphaqik), ...
0486                                ' <  ', char(sym(0))]), 'true');
0487                expr2 = strcmpi(maple('evalb', [char(sym(0)), ...
0488                                ' <  ', char(betaqik)]), 'true');
0489                expr3 = strcmpi(maple('evalb', [char(betaqik), ...
0490                                ' < ', char(sym(1))]), 'true');
0491                if ((expr1) && (expr2 && expr3))
0492 %                 if ((alphaqik < 0 ) && ((0 < betaqik) &&(betaqik < 1)))
0493                    phiai =phi(1,2*0+1,q+1);
0494                    phibi =phi(k+1,2*i+2, q+1);
0495                end;
0496                expr1 = strcmpi(maple('evalb', [char(sym(0)), ...
0497                                ' < ', char(alphaqik)]), 'true');
0498                expr2 = strcmpi(maple('evalb', [char(alphaqik), ...
0499                                ' < ', char(sym(1))]), 'true');
0500                expr3 = strcmpi(maple('evalb', [char(betaqik), ...
0501                                 ' > ', char(sym(1))]), 'true');
0502                if ((expr1 && expr2) && (expr3))
```

```
0503 %                  if (((0 <alphaqik) && (alphaqik <1)) && (betaqik > 1))
0504                      phiai = phi(k+1,2*i+1,q+1);
0505                      phibi = phi(1, 2*0+2, q+1);
0506                  end;
0507                  expr1 = strcmpi(maple('evalb', [char(alphaqjk), ...
0508                              ' < ', char(sym(0))]), 'true');
0509                  expr2 = strcmpi(maple('evalb', [char(sym(0)), ...
0510                              ' < ', char(betaqjk)]), 'true');
0511                  expr3 = strcmpi(maple('evalb', [char(betaqjk), ...
0512                              ' < ', char(sym(1))]), 'true');
0513                  if ((expr1) && (expr2 && expr3))
0514 %                  if ((alphaqjk < 0 ) && ((0 < betaqjk) &&(betaqjk < 1)))
0515                      phiaj =phi(1,2*0+1,q+1);
0516                      phibj =phi(k+1,2*j+2, q+1);
0517                  end;
0518                  expr1 = strcmpi(maple('evalb', [char(sym(0)), ...
0519                                  ' < ', char(alphaqjk)]), 'true');
0520                  expr2 = strcmpi(maple('evalb', [char(alphaqjk),  ...
0521                                  ' < ' , char(sym(1))]), 'true');
0522                  expr3 = strcmpi(maple('evalb', [char(betaqjk), ...
0523                                  ' > ', char(sym(1))]), 'true');
```

```
0524                 if ((expr1 && expr2) && (expr3))
0525 %                 if (((0 <alphaqjk) && (alphaqjk <1)) && (betaqjk > 1))
0526                     phiaj = phi(k+1,2*j+1,q+1);
0527                     phibj = phi(1, 2*0+2, q+1);
0528                 end;
0529                 phia = phiai + lambda(2)* phiaj;
0530                 phib = phibi + lambda(2)*phibj;
0531                 u = [u; phia q i j];
0532                 u = [u; phib q i j];
0533             end;
0534         end;
0535     end;
0536 end;
0537 temp = u;
0538 u = sortrowshp(temp,1);
```

B.1.2 findfirst_hp

```
0001 function [tble] = findfirst_hp(n,qwanted,kmax)
0002
0003 % findfirst(n,qwanted,kmax)
0004 %
0005 % input : qwanted= which phi_q for which to find intervals of first find
```

```
0006 %                  set qwanted = -1 to find for all phi_q
0007 %         kmax =   find intervals of the first kind up to rank kmax
0008 %                   set kmax to negative value to get just the intervals of
0009 %                   the first kind only for rank kmax. abs(kmax) >=1.
0010 %         n    =  the dimension space
0011 % output: tble = contains all the points which are of the first kind
0012 %                 each row has the form
0013 %                      [q i k alphaqik betaqik j l 9 betaqjl] or
0014 %                      [q i k alphaqik betaqik j l 1 alphaqjl]
0015
0016
0017 % finds the intervals of the first kind
0018 ptable = sym([]);
0019 qmin =0;
0020 qmax =0;
0021 if (qwanted == -1)
0022    qmin = 0;
0023    qmax = 4;
0024 elseif (qwanted > 0)
0025     qmin = qwanted;
0026     qmax = qwanted;
```

```
0027 end;
0028 mykmin = 2;
0029 if (kmax < 0)
0030    mykmin = abs(kmax);
0031    mykmax = abs(kmax);
0032 elseif (kmax > 0)
0033     mykmin = 1;
0034     mykmax = kmax;
0035 end;
0036 for q=qmin:qmax
0037     for k=mykmin:mykmax
0038         for i =0:(10^(k-1))
0039             alphaqik = sym((10^(-k))*(10*i+1 - 2*q));
0040             betaqik = sym((10^(-k))*(10*i+9 - 2*q));
0041             for l =0:(k-1)
0042                 for j = 0:(floor(10^(l-1)))
0043                     if (l > 0)
0044                         alphaqjl = sym((10^(-l))*(10*j+1 - 2*q));
0045                         betaqjl = sym((10^(-l))*(10*j+9 - 2*q));
0046                     else
0047                         alphaqjl = sym(0);
```

```
0048                        betaqjl =  sym(1);
0049                    end;
0050                    expr1 = strcmpi(maple('evalb', [char(alphaqik), ...
0051                              ' <  ', char(alphaqjl)]), 'true');
0052                    expr2 = strcmpi(maple('evalb', [char(alphaqjl), ...
0053                              ' <  ', char(betaqik)]), 'true');
0054                    if(expr1 && expr2)
0055 %                    if ((double(alphaqik) < double(alphaqjl)) && ...
0056 %                            (double(alphaqjl) < double(betaqik)))
0057                        ptable = [ptable; [q i k alphaqik betaqik j ...
0058                                    l 1 alphaqjl]];
0059                    end;
0060                    expr1 = strcmpi(maple('evalb', [char(alphaqik), ...
0061                              ' <  ', char(betaqjl)]), 'true');
0062                    expr2 = strcmpi(maple('evalb', [char(betaqjl), ...
0063                              ' <  ', char(betaqik)]), 'true');
0064                    if(expr1 && expr2)
0065 %                    if ((double(alphaqik) < double(betaqjl)) && ...
0066 %                            (double(betaqjl) < double(betaqik)))
0067                        ptable = [ptable; [q i k alphaqik betaqik j ...
0068                                       l 9 betaqjl]];
```

```
0069                     end;
0070                 end;
0071             end;
0072         end;
0073     end;
0074 end;
0075
0076 tble = ptable;
```

B.1.3 train_gt

```
0001 function [H2D,G1D] = train_gt(k,phihp,uhp, myiter, funin)
0002 %
0003 % this program takes an image and sees what image is reproduced after
0004 % myiter iterations
0005
0006 % Setup the function f (an image) for processing
0007 global fun;
0008 global xi;
0009 global yi;
0010 fun = funin;
0011 SIZEFUN =size(fun);
0012  mx =1:SIZEFUN(2);
```

```
0013  nx =1:SIZEFUN(1);
0014 % mxl=SIZEFUN(1);
0015 % nxl = SIZEFUN(1);
0016  xii=(1/(SIZEFUN(2)-1))*(mx-1);
0017  yii=(1/(SIZEFUN(1)-1))*(nx-1);
0018 [xi,yi] = meshgrid(xii,yii);
0019 disp([size(xi) size(yi)]);
0020 % Setup the function phi,u  for processing
0021 disp('initializing.........');
0022 if (~isnumeric(uhp))
0023     disp('converting u')
0024     SIZEU = size(uhp);
0025     u = zeros(SIZEU);
0026     tic;
0027     disp([' time start=' num2str(0)]);
0028     for i =1:SIZEU(1)
0029         u(i,:)= double(uhp(i,:));
0030         if (mod(i,5000) ==0 )
0031             TOCTIC = toc;
0032             disp(['i =' num2str(i) ' time end=' num2str(TOCTIC)]);
0033             tic;
```

```
0034             disp([' time start=' num2str(0)]);
0035         end;
0036     end;
0037     disp('done converting u');
0038 else
0039     disp('u is already type double');
0040     u = uhp;
0041 end;
0042 if (~isnumeric(phihp{1}.table))
0043     disp('converting phi_q');
0044     for q=0:4
0045         phi{q+1}.table = double(phihp{q+1}.table);
0046         disp(['done converting phi_' num2str(q)]);
0047     end;
0048     disp('done converting phi_q');
0049 else
0050     disp('phi_q is already type double');
0051     phi = phihp;
0052 end;
0053 disp('done initializing....');
0054
```

```
0055 SIZEU = size(u);

0056 global lambda;

0057

0058 lambda = 1/sqrt(2);

0059 maxiter =myiter;

0060

0061

0062 % preallocate memory

0063 gr = zeros([SIZEU(1) maxiter]);

0064 G = zeros(SIZEU(1),1);

0065

0066 % X=mx-1;

0067 % Y=nx-1;

0068 % [x,y] = meshgrid(X,Y);

0069 % SIZEX = size(x);

0070 % SIZEY =size(y);

0071 % xv = reshape(x,SIZEX(1)*SIZEX(2),1); %make x a long vector

0072 % yv = reshape(y,SIZEY(1)*SIZEY(2),1); %make y a long vector..

0073 % has same dimensions as x

0074 % Plot f(x,y)

0075 %z = f(x,y);
```

```
0076 %z = reshape(zv,SIZEX(1),SIZEX(2));
0077 % subplot(2,2,1);
0078 % surf(x,y,z);
0079 % reply = input('Pausing. Type 1 to loop without pausing: ');
0080 %assign values to f from f(x,y) at centers of squares
0081 alphaqik = (10^(-k))*(10*u(:,3)+1-2*u(:,2));
0082 betaqik = (10^(-k))*(10*u(:,3)+9-2*u(:,2));
0083 alphaqjk = (10^(-k))*(10*u(:,4)+1-2*u(:,2));
0084 betaqjk = (10^(-k))*(10*u(:,4)+9-2*u(:,2));
0085 %get rid of squares outside of [0,1] x [0,1]
0086 temp = find((alphaqik > 1) | (alphaqjk > 1) | ...
0087                                  (betaqik < 0) | (betaqjk < 0) );
0088 if (~isempty(temp))
0089     alphaqik(temp) = -10;
0090 end;
0091 temp = find(alphaqik ~= -10);
0092 if (~isempty(temp))
0093     alphaqik = alphaqik(temp);
0094     alphaqjk = alphaqjk(temp);
0095     betaqik = betaqik(temp);
0096     betaqjk = betaqjk(temp);
```

```
0097 end;
0098 %truncate squares partially in [0,1] x [0,1]
0099 temp = find(alphaqik <0);
0100 if (~isempty(temp))
0101     alphaqik(temp) = 0;
0102 end;
0103 temp = find(betaqik >1);
0104 if (~isempty(temp))
0105     betaqik(temp) = 1;
0106 end;
0107 temp = find(alphaqjk <0);
0108 if (~isempty(temp))
0109     alphaqjk(temp) = 0;
0110 end;
0111 temp = find(betaqjk > 1);
0112 if (~isempty(temp))
0113     betaqjk(temp) = 1;
0114 end;
0115 m1 = (alphaqik + betaqik)/2;
0116 m2 = (alphaqjk + betaqjk)/2;
0117 clear alphaqik betaqik alphaqjik betaqjk ;
```

```
0118
0119
0120
0121 %disp([size(m1) size(m2)]);
0122 % calculate h(x,y) from f(x,y);
0123 %iterate
0124 for iter=1:maxiter
0125     i= iter;
0126 %    disp(['iteration: ' num2str(iter)]);
0127     fxy = f(m1, m2) - h(m1, m2,u,gr,phi,1);
0128     gr(:,i) = (1/3)*fxy;
0129
0130 %     if(reply ~= 1)
0131 %         %plot h(x,y), gr(u), G(u)
0132 %         % gr(u)
0133 %         subplot(2,2,3);
0134 %         plot(u(:,1),gr(:,i));
0135 %
0136 %         % G(u) = sum_{1}^(iter} gr(u)
0137          G = G + gr(:,i);
0138 %         subplot(2,2,4);
```

```
0139 %         plot(u(:,1),G);
0140 %
0141 %         %h(x,y)
0142 %         zv = h(xv,yv,u,gr,phi,1);
0143 %         z = reshape(zv,SIZEX(1),SIZEX(2));
0144 %         subplot(2,2,2);
0145 %         surf(x,y,z);
0146 %         % pause to view plot
0147 %         reply = input('Pausing. Type 1 to loop without pausing: ');
0148 %     end;
0149 end;
0150
0151 %plot h(x,y), gr(u), g(u) for last time
0152 % gr(u)
0153 % subplot(2,2,3);
0154 % plot(u(:,1),gr(:,maxiter));
0155
0156 % G(u) = sum_{1}^(iter} gr(u)
0157 % subplot(2,2,4);
0158 % plot(u(:,1),G);
0159
```

```
0160 %h(x,y)
0161 % subplot(2,2,2);
0162 % surf(x,y,z);
0163 SIZEXI =size(xi);
0164 SIZEYI = size(yi);
0165 zv = h(reshape(xi,SIZEXI(1)*SIZEXI(2),1),reshape(yi, ...
0166             SIZEYI(1)*SIZEYI(2),1),u,gr,phi,1);
0167 disp(size(zv));
0168 z = reshape(zv,SIZEFUN(1),SIZEFUN(2));
0169 H2D =z;
0170 G1D = G;
0171 %save('myz','z', 'fun');
0172 %%%%% end main function app8_hp
0173 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0174 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0175
0176 %define f(x,y) here
0177 function [fvalue] = f(x,y)
0178     global fun;
0179     global xi;
0180     global yi;
```

```
0181     fvalue = interp2(xi,yi,fun,x,y);
0182
0183 %define h(x,y) here
0184 function[hvalue] = h(x,y,u,g,phi,tosum)
0185 global lambda;
0186 hsum =0;
0187 SIZEG = size(g);
0188 if (tosum == 0) % sum only the latest g_r; for h_r(,y)
0189     for q =0:4
0190         uc = interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),x) + ...
0191             lambda * interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),y);
0192         hsum = hsum + interp1(u(:,1),g(:,SIZEG(2)), uc);
0193     end;
0194 else  % sum all the g_r ; for h(x,y)
0195     for i=1:SIZEG(2)
0196         for q =0:4
0197             uc = interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),x) + ...
0198                 lambda * interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),y);
0199             hsum = hsum + interp1(u(:,1),g(:,i), uc);
0200         end;
0201     end;
```

```
0202 end;

0203 hvalue = hsum;
```

B.1.4 recon_ft

```
0001 function [f2D] = recon_ft(u,g, imageSize, phi)

0002 %

0003

0004

0005 global xi;

0006 global yi;

0007

0008 SIZEFUN = imageSize;

0009 mx =1:SIZEFUN(2);

0010 nx =1:SIZEFUN(1);

0011 xii=(1/(SIZEFUN(2)-1))*(mx-1);

0012 yii=(1/(SIZEFUN(1)-1))*(nx-1);

0013 [xi,yi] = meshgrid(xii,yii);

0014

0015 global lambda;

0016

0017 lambda = 1/sqrt(2);

0018
```

```
0019 SIZEXI =size(xi);
0020 SIZEYI = size(yi);
0021 zv = h(reshape(xi,SIZEXI(1)*SIZEXI(2),1),reshape(yi,...
0022            SIZEYI(1)*SIZEYI(2),1),u,g,phi,0);
0023 disp(size(zv));
0024 z = reshape(zv,SIZEFUN(1),SIZEFUN(2));
0025 f2D =z;
0026 %save('myz','z', 'fun');
0027 %%%%% end main function recon_f
0028 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0029 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0030
0031 %define h(x,y) here
0032 function[hvalue] = h(x,y,u,g,phi,tosum)
0033 global lambda;
0034 hsum =0;
0035 SIZEG = size(g);
0036 if (tosum == 0) % sum only the latest g_r; for h_r(,y)
0037     for q =0:4
0038         uc = interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),x) + ...
0039             lambda * interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),y);
```

```
0040         hsum = hsum + interp1(u(:,1),g(:,SIZEG(2)), uc);
0041     end;
0042 else  % sum all the g_r ; for h(x,y)
0043     for i=1:SIZEG(2)
0044         for q =0:4
0045             uc = interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),x) + ...
0046                 lambda * interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),y);
0047             hsum = hsum + interp1(u(:,1),g(:,i), uc);
0048         end;
0049     end;
0050 end;
0051 hvalue = hsum;
```

B.1.5 eval_gt

```
0001 function [fx] = eval_gt(x, phi, u,g)
0002 %
0003
0004
0005 global lambda;
0006
0007 lambda = 1/sqrt(2);
0008
```

```
0009 zv = h(x(:,1),x(:,2),u,g,phi,0);
0010 fx =zv;
0011
0012
0013 %%%%% end main function eval_g
0014 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0015 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
0016
0017 %define h(x,y) here
0018 function[hvalue] = h(x,y,u,g,phi,tosum)
0019 global lambda;
0020 hsum =0;
0021 SIZEG = size(g);
0022 if (tosum == 0) % sum only the latest g_r; for h_r(,y)
0023     for q =0:4
0024         uc = interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),x) + ...
0025             lambda * interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),y);
0026         hsum = hsum + interp1(u(:,1),g(:,SIZEG(2)), uc);
0027     end;
0028 else  % sum all the g_r ; for h(x,y)
0029     for i=1:SIZEG(2)
```

```
0030         for q =0:4
0031             uc = interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),x) + ...
0032                 lambda * interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),y);
0033             hsum = hsum + interp1(u(:,1),g(:,i), uc);
0034         end;
0035     end;
0036 end;
0037 hvalue = hsum;
```

LIST OF REFERENCES

[1]

[2] http://wang.ist.psu.edu.

[3] Richard Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press., 1961.

[4] M.J. Black and S. Roth. Fields of experts: A framework for learning image priors. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 860–867, 2005.

[5] J.S. Catalan, J.A.; Jin. Dimension reduction of texture features for image retrieval using hybrid associative neural networks. *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, 2:1211–1214 vol.2, 2000.

[6] Myron Flickner, Harpreet S. Sawhney, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The qbic system. *IEEE Computer*, 28(9):23–32, 1995.

[7] Tom Heskes. Selecting weighting factors in logarithmic opinion pools. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[8] D. Hilbert. Mathematical problems. *Bull. Amer. Math. Soc.*

[9] G. Hinton, S. Osindero, and K. Bao. Learning causally linked markov random fields, 2005.

[10] Geoffry E. Hinton. Product of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*, volume 1, pages 1–6, 1999.

[11] Geoffry E. Hinton. Training product of experts by minimizing contrastive divergence. Technical Report 004, Gatsby Computational Neuroscience Unit, 2000.

[12] Qasim Iqbal and J.K. Aggarwal. Feature integration, multi-image queries and relevance feedback in image retrieval.

[13] A.J. Smola J.J. McAuley, T.S. Caetano and M.O. Franz. Learning high order mrf priors of color images.

[14] M.I. Jordan. *Learning in graphical models*. MIT Press Cambridge, MA USA, 1993.

[15] W. J. Krzanowski. Recent advances in descriptive multivariate analysis. 1995.

[16] G.G. Lorentz. *Approximation of Functions*. Holt, Rinehart, and Winston, Inc., 1966.

[17] G. Hinton M. Welling and S. Osindero.

[18] W.Y. Manjunath, B.S.; Ma. Texture features for browsing and retrieval of image data. *Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, Aug 1996.

[19] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1994.

[20] Wayne Niblack, Ron Barber, William Equitz, Myron Flickner, Eduardo H. Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos, and Gabriel Taubin. The qbic project: Querying images by content, using color, texture, and shape. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 173–187, 1993.

[21] S. Roth and M.J. Black. On the spatial statistics of optical flow. In *Proceeedings of the Tenth IEEE International Conference on Computer Vision*, volume 1, pages 42–49, 2005.

[22] Alan F. Smeaton, Paul Over, and Wessel Kraaij. Evaluation campaigns and trecvid. In *MIR '06: Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval*, pages 321–330, New York, NY, USA, 2006. ACM Press.

[23] A. M. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.

[24] Ce; Adelson Edward H.; Freeman William T. Tappen, Marshall F.; Liu. Learning gaussian conditional random fields for low-level vision. *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 17–22 June 2007.

[25] A.L. Yuille. The convergence of contrastive divergences. *Advances in Information Processing Systems*, 17:1593–1600, 2005.

[26] S. Osindero Y.W. Teh, M. Welling and G.E. Hinton.

[27] S.C. Zhu and X. Liu. Learning in gibbsian fields: How accurate and how fast can it be? *IEEE Trans. Pattern Anal. Mach. Intell*, 7:1001–1006, 2002.