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

2008

Donald Bryant
University of Central Florida

Find similar works at: http://stars.library.ucf.edu/etd
University of Central Florida Libraries http://library.ucf.edu

Part of the Mathematics Commons

STARS Citation

Bryant, Donald, "Analysis Of Kolmogorov's Superposition Theorem And Its Implementation In Applications With Low And High Dimensional Data." (2008). Electronic Theses and Dissertations. 3689.
http://stars.library.ucf.edu/etd/3689

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER ONE: INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>3</td>
</tr>
<tr>
<td>1.1.1 Dissertation Outline</td>
<td>3</td>
</tr>
<tr>
<td>CHAPTER TWO: BACKGROUND</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Approximation Theory</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Probability and Statistics</td>
<td>8</td>
</tr>
<tr>
<td>2.2.1 Probability</td>
<td>8</td>
</tr>
<tr>
<td>2.2.2 Statistics</td>
<td>9</td>
</tr>
<tr>
<td>CHAPTER THREE: KOLMOGOROV’S THEOREM</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Kolmogorov’s Theorem</td>
<td>11</td>
</tr>
<tr>
<td>3.1.1 Theorem Statement</td>
<td>11</td>
</tr>
<tr>
<td>3.1.2 Details required for Proof</td>
<td>11</td>
</tr>
<tr>
<td>3.1.3 Proof of Theorem</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Properties of ( f ) and its ( g ) function</td>
<td>22</td>
</tr>
<tr>
<td>3.2.1 Addition and Multiplication</td>
<td>23</td>
</tr>
<tr>
<td>3.2.2 Shifting</td>
<td>25</td>
</tr>
<tr>
<td>3.2.3 Scaling</td>
<td>29</td>
</tr>
</tbody>
</table>
6.3 Bayesian Prior Image Applications ........................................ 72
  6.3.1 Product of Experts .................................................... 75
  6.3.2 Field of Experts ....................................................... 76
  6.3.3 PoE and FoE revisited ................................................ 76
  6.3.4 A new approach ....................................................... 78

APPENDIX A: OVERVIEW OF FUNCTIONS ...................................... 79
  A.1 Function Overview ...................................................... 80

APPENDIX B: FUNCTION CODE PRINTOUT .................................... 82
  B.1 Code for building $\varphi_q$ functions ................................. 83
     B.1.1 buildphis_hp ...................................................... 83
     B.1.2 findfirst_hp .................................................... 108
     B.1.3 train_gt .......................................................... 112
     B.1.4 recon_ft ......................................................... 122
     B.1.5 eval_gt ........................................................... 124

LIST OF REFERENCES .......................................................... 127
LIST OF FIGURES

3.1 Example of intervals $i, k$ up to $k = 2$ ........................................ 14
3.2 Intervals $I_{3i}$ not contained completely in $I = (0, 1)$ ................. 14
3.3 Example of intervals $I_{3i}^2$ of the first kind ................................... 18
3.4 Example of intervals $I_{3i}^2$ of the second kind ............................... 19
3.5 Example squares $S$ for $k = 1$ ..................................................... 20
3.6 Addition ........................................................................ 24
3.7 Multiplication ................................................................. 26
3.8 Shifting functions $dU_x$ (left) and $dU_y$ (right) ............................ 27
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) $g_2$ .... 28
3.10 Scaling functions $dW_x$ (left) and $dW_y$ (right) ............................ 30
3.11 Scaling along $x$ dimension ..................................................... 31
3.12 Partial Derivatives with respect to $x \ y$ ...................................... 34
5.1 tiger image ............................................................................. 53
5.2 tiger image ............................................................................. 53
6.1 $f(x, y) = 1$ ........................................................................ 58
6.2 $f(x, y) = \cos \left[ \frac{2}{\pi}(x - y) \right]$ ........................................... 59
6.3 $f(x, y) = \sin(2\pi x) \sin(2\pi y)$ ................................................ 59
6.4 Image, $g$-function, reconstruction ............................................. 60
6.5 Image, g function, reconstruction ........................................ 60
6.6 MRI image slices (top) and their reconstructions (bottom) ........ 62
6.7 Portion of associated g function ........................................... 63
6.8 Frames in a video of water drop into a bucket and the reconstruction .... 64
6.9 Indexing: All entries in the database will go through this step; a query image
will go through the same step. ............................................... 67
6.10 Various Images in WANG Database ..................................... 68
6.11 Top Image: Query Image. Six lower images are the best returned matches
from the dataset. .............................................................. 69
6.12 Top Image: Query Image. Six lower images are the best returned matches
from the dataset. .............................................................. 69
6.13 Top Image: Query Image (dinosaurs). Six lower images are the best returned
matches from the dataset. ................................................... 70
6.14 Top Image: Query Image (flowers). Six lower images are the best returned
matches from the dataset. ................................................... 71
LIST OF TABLES

5.1 Values for $\phi_0(x)$ function define over $[0, 1]$ ........................................ 55
5.2 Sample points $u_{qij}$ for $q = 0$ and $k = 2$ $\lambda_1 = 1$, $\lambda_2 = \frac{1}{\sqrt{2}}$ ......... 56
5.3 Interpolation table for $(u_{qij}, g(u_{qij}))$, here the $f_{qij}^k$’s are the values at the cen-
ters of the squares $S_{qij}^k$ associated with $u_{qij}^k$ ........................................ 57
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 \( \phi_i \in \Phi \) 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 \( \Phi \) 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

\[
\|f\| = \sup_{x \in A} |f(x)|
\]

(2.1)
is attained for all functions \( f \in C \); thus

\[
\|f\| = \max_{x \in A} |f|.
\]  

(2.2)

This defines a norm on \( C[A] \) which has the following properties:

\[
\|f\| \geq 0; \quad \|f\| = 0 \text{ if and only if } f = 0
\]

\[
\|af\| = |a| \cdot \|f\|
\]

\[
\|f + g\| \leq \|f\| + \|g\|.
\]  

(2.3)

Thus, \( C \) is a normed linear space associated with all continuous 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 \( C \):

\[
\|f_n - f\| \to 0
\]  

(2.4)

Complete normed spaces are called \textit{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 \textit{approximable} by linear combinations

\[
P = a_1 \phi_1 + a_2 \phi_2 + \cdots + a_n \phi_n; \quad \phi_i \in \Phi, \ a_i \text{ real},
\]  

(2.5)

if for each \( \epsilon > 0 \) there is a \( P \) with \( \|f - P\| < \epsilon \). Often, \( \Phi \) is a sequence: \( \phi_1, \phi_2, \ldots, \phi_n, \ldots \).

Then

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

(2.6)
is the $n$th degree of approximation of $f$ by the $\phi_i$. 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 $t$,

$$\Delta_t f(x) = f(x + t) - f(x)$$  \hspace{1cm} (2.7)

of the function $f$ and put

$$\omega(f, h) = \omega(h) = \max_{x,x+t \in [a,b]} \frac{1}{|t| \leq h} |f(x + t) - f(x)|.$$ \hspace{1cm} (2.8)

The function $\omega(h)$, called the modulus of continuity of $f$, is defined for $0 \leq h < l$, 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 $\alpha$, or belongs to the class $\text{Lip}_M \alpha$, $M > 0$, $0 < \alpha \leq 1$ if

$$|f(x') - f(x)| \leq M|x' - x|^{\alpha}, \quad x, x' \in A.$$ \hspace{1cm} (2.9)

This is equivalent to the inequality $\omega(f, h) \leq M h^{\alpha}$, for $0 \leq h < l$. As an example, if $f$ has a derivative that satisfies $|f'(x)| \leq M$, $x \in A$, then $f \in \text{Lip}_M 1$. It is know that a function satisfying a Lipshitz condition on $[a,b]$ is continuous $[a,b]$. Define $\omega(f, A) = \sup_{0 \leq h < l} \omega(f, h) = \omega(f, l)$. 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 \( \Omega \) 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 \( \Omega \) 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 \( \omega \) 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

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

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 $\lambda_i, i = 1, 2,$ and five functions $\varphi_q(x)$ for $q = 0, \ldots, 4$ defined on $I$ with values in $I$, which have the following properties: The $\varphi_q$’s are strictly increasing and belong to a class Lip $\alpha$, $\alpha > 0$. 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

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

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 $\lambda_i$ 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 $\varphi_q$ ($q = 0, \ldots, 4$) 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 $\lambda_i$ constants such that for rational numbers $a_i$ the following holds true:

$$\sum a_i \lambda_i \neq 0 \quad \text{if not all } a_i = 0 \quad (3.2)$$

For the two dimensional case we can let $\lambda_1 = 1$ and $\lambda_2 = \lambda$ where we choose $\lambda$ to be an irrational number. In this case it is certain that 1 and $\lambda$ are linearly independent over the rational numbers.
Next comes the construction of the \( \varphi \)-functions. We begin the process with some observations. There are four properties that we wish the five \( \varphi_q \) 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 \( \varphi_q \) function will be rational and distinct (and all the \( \varphi_q \) 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 \( \varphi_q \) as piecewise linear functions over disjoint intervals of \( I = [0, 1] \). Those intervals are defined by

\[
I^k_i = [i \cdot 10^{-k+1} + 10^{-k}, i \cdot 10^{-k+1} + 9 \cdot 10^{-k}]
\]  
(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 \( I^k_i \) are contained in \( I = [0, 1] \), except when \( i = 10^{k-1} \). Figure 3.1 shows intervals for up to \( k = 2 \).

The functions \( \varphi_q \) are associated with intervals \( I^k_{qi} \), which are shifts of the intervals \( I^k_i \). The relation is as follows:

\[
I^k_{qi} = I^k_i - 2q \cdot 10^{-k}
\]  
(3.4)
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 $I \cap I^k_q$. 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 $\alpha, \beta$ points. Defining $\alpha^k_q = i \cdot 10^{-k+1} + 10^{-k} - 2q \cdot 10^{-k}$ and $\beta^k_q = i \cdot 10^{-k+1} + 9 \cdot 10^{-k} - 2q \cdot 10^{-k}$, equation (3.4) becomes,

$$I_{qi}^k = I_i^k - 2q \cdot 10^{-k} = [\alpha^k_q, \beta^k_q]$$

(3.5)
for \( q = 0, 1, \ldots, 4; \ i = 0, 1, 2, \ldots, 10^{k-1}; \ k = 1, 2, 3, \ldots \)

If the rank \( k \) is fixed, the intervals in (3.5) form five families, corresponding to \( q = 0, \ldots, 4 \). No one of the families covers \( I \). However, each point \( x \in I \) is covered by at least four of the five families, \( q = 0, \ldots, 4 \). The functions \( \varphi_q \) 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 \( \varphi_q \) 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, \( \varphi_q, \ q = 0, \ldots, 4 \), defined on \([0, 1]\), with values in \([0, 1]\), and belonging to a class \( \text{Lip} \alpha, \alpha > 0 \) such that for each fixed rank \( k = 1, 2, \ldots \), the intervals

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

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 \( \varphi_q \) 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 \( \alpha_{qi}^l \), or \( \beta_{qi}^l \) from a lower rank \( l < k \).
ii. Intervals of the second kind: these intervals do not contain any points $\alpha_{qi}^l$, or $\beta_{qi}^l$ 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 $\gamma$, where $\gamma$ is either $\alpha_{qi}^l$ or $\beta_{qi}^l$.

(b) For intervals $I_{qi}^k$, we first assign $\varphi_q$ at the points $\alpha_{qi}^k$ and $\beta_{qi}^k$ the same value:

$$\varphi_q(\alpha_{qi}^k) = \varphi(\beta_{qi}^k) =: \varphi_{qi}^k,$$
where $\varphi_{qi}^k$ 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 $\gamma_{qi}^k$. 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 $(\gamma, \gamma')$, where $\gamma, \gamma'$ 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 $(\gamma, \gamma')$. 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 $(\beta_{qi}^l, \varphi(\beta_{qi}^l))$, $l \leq k$ 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 $\varphi_q(x')$.

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

(3.7)

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

16
(a) Take all the values \( \varphi_{q_i}^k \) for rank \( k \) and form the values

\[
\varphi_{q_i}^k + \lambda \varphi_{q_j}^k, \quad q = 0, \ldots, 4; \quad i, j = 0, \ldots, 10^{k-1}
\]  

and order them on the real number line.

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

\[
\varphi_q^k(\alpha_{q_i}^k) < \varphi_{q_i}^k < \varphi_q^k(\beta_{q_i}^k)
\]

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

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

Once the preceding steps are completed the four criteria which are necessary for the \( \varphi \) 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 \( \varphi_q \) are strictly increasing on the set of all the points \( \alpha_{q_i}^l, \beta_{q_i}^l, 0 \leq i \leq 10^{l-1}, \ l \leq k \).

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

3. All values of \( \varphi_q(\alpha_{q_i}^l), \varphi_q(\beta_{q_i}^l), \ q = 0, \ldots, 4, \ 0 \leq i \leq 10^{k-1}, \ l \leq k, \) 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 $[0, 2]$,

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

(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 $S_{qi}^k$, where

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

(3.11)

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

18
The image of squares (3.11) under the map \((x, y) \rightarrow \varphi_q(x) + \lambda \varphi_q(y)\) are the intervals \(\Delta^k_{qij}\) 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 \(I^k_{qi}\) 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 \(I^k_{qj}\). 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).
Lemma 2. Let $\frac{2}{3} < \theta < 1$. Let $S = I_{q_i}^k \times I_{q_j}^k$. For each $f \in C[S]$, there is a function
$g \in C[0, 2]$ such that
\[
\|f - h\| \leq \theta \|f\| \quad \text{and} \quad \|g\| \leq \frac{1}{3} \|f\| \tag{3.12}
\]

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

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
\[
\|f - h_1\| \leq \theta \|f\|, \quad \|g_1\| \leq \frac{1}{3} \|f\|. \tag{3.15}
\]
Applying Lemma 2 again and obtain functions $g_2, h_2$, while using $(f - h_1)$ instead of $f$, for which
\[
\|(f - h_1) - h_2\| \leq \theta \|f - h_1\| \leq \theta^2 \|f\|, \quad \|g_2\| \leq \frac{1}{3} \|f - h_1\| \leq \frac{1}{3} \theta \|f\|. \tag{3.16}
\]
In general the functions \( g_r, h_r \) will satisfy

\[
\|f - h_1 \cdots - h_r\| \leq \theta^r \|f\|, \quad \|g_r\| \leq \frac{1}{3} \theta^{r-1} \|f\|, \quad r = 1, 2, \ldots
\]  

(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 \( \Delta^k_{qij}, g_f(u_q) = \frac{1}{3} f^k_{qij}, g_1(u_q) = \frac{1}{3} f^k_{1qij}, \) and \( g_2(u_q) = \frac{1}{3} f^k_{2qij} \). Now on each such interval, \( f^k_{qij} = f^k_{1qij} + f^k_{2qij} \) and finally,

\[
g_f(u_q) = \frac{1}{3} f^k_{qij} = \frac{1}{3} (f^k_{1qij} + f^k_{2qij}) = \frac{1}{3} f^k_{1qij} + \frac{1}{3} f^k_{2qij} = g_1(u_q) + g_2(u_q).
\] (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 \( \Delta_{qij}^k \),

\[ g_f(u_q) = \frac{1}{3} f_{qij}^k, \quad g_1(u_q) = \frac{1}{3} f_{1,qij}^k, \] and \( g_2(u_q) = \frac{1}{3} f_{2,qij}^k \). Now on each interval, \( f_{qij}^k = f_{1,qij}^k \cdot f_{2,qij}^k \)
and finally,
\[
g_f(u_q) = \frac{1}{3} f_{qij}^k
\]
\[
= \frac{1}{3} (f_{1,qij}^k \cdot f_{2,qij}^k)
\]
\[
= 3 \cdot \frac{1}{3} f_{1,qij}^k \cdot \frac{1}{3} f_{2,qij}^k
\]
\[
= 3 g_1(u_q) \cdot g_2(u_q).
\]

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, \(dU_{xi}\), and constants \(\delta x_i\), \(i = 1 \ldots n\), such that on \(\Delta^k_q\) we define \(dU(u_q) = \varphi_q(M_i + \delta x_i) - \varphi(M_i)\) for all \(u_q \in \Delta^k_q\). \(M_i = \frac{\alpha^k_i + \beta^k_i}{2}\) 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, \(\delta x = \delta y = 10^{-2}\)

\[
x_1 = x; \quad dU_x(u_q) = \varphi_q(\frac{\alpha^k_{qi} + \beta^k_{qi}}{2} + \delta x) - \varphi_q(\frac{\alpha^k_{qi} + \beta^k_{qi}}{2}) \quad \text{for } u_q \in \Delta^k_{qij}
\]
\[
x_2 = y; \quad dU_y(u_q) = \varphi_q(\frac{\alpha^k_{qi} + \beta^k_{qi}}{2} + \delta y) - \varphi_q(\frac{\alpha^k_{qi} + \beta^k_{qi}}{2}) \quad \text{for } u_q \in \Delta^k_{qij}
\]

(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 \(\delta x\), the following is their relationship:

\[
g_1(u_q) = g_f(u_q + dU_{x_i}(u_q))
\]

(3.20)
In order to shift a function \( f \) with \( g \)-function, \( g_f \), a multiple of \( \delta x \) times along a chosen dimension, the following compositions are performed. Let \( g_0 = g_f \), then

\[
g_{N+1}(u_q) = g_N(u_q + dU_x(u_q)), \quad N = 0, 1, 2, \ldots
\]  

(3.21)

Each composition shifts the graph by \( \delta x_i \) 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). \quad \text{Without loss of generality, let’s look along} \\

26
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 $\delta x_1$. Let the left square be the $j$th square and the square to its right be $j + 1$. The location of the center of square $j$ is $M_j = \frac{\alpha_k q_j + \beta_k q_j^2}{2}$ for $q = 0, 1, \ldots, 2n$. And the center for square $j + 1$ is $M_{j+1} = \frac{\alpha_k q_{j+1} + \beta_k q_{j+1}}{2}$. We assign $\delta x_1 = M_{j+1} - M_j = (2n + 1)^{-k}$.

This definition of $\delta x_1$ 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

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

\[
= \lambda_i \varphi_q(M_{j+1}) - \lambda_i \varphi_q(M_j)
\]

\[
= \lambda_i [\varphi_q(M_{j+1}) - \varphi_q(M_j)]
\]
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

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

(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

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

(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, $dV_{x_1}$, $n$ universal $dW$-functions $dW_{x_i}$, $i = 1 \ldots n$, such that for a variable $u_q \in \Delta^k_q$ we define $dV(u_q) = \varphi_q(\frac{1}{2}M_i) - \varphi_q(M_i)$ and $dW(u_q) = \varphi_q(2M_i) - \varphi(M_i)$. $M_i = \frac{\alpha^{k_i} + \beta^{k_i}}{2}$ is the midpoint of interval $I^k_{q_i}$. $dV$ is a stretching function and $dW$ is a shrinking function.
For the two dimensional case we have,

\[
\begin{align*}
    x_1 &= x \quad dV_x(u_q) = \varphi_q(\frac{1}{2} \alpha_{kq}^k + \beta_{ki}^k) - \varphi_q(\frac{\alpha_{kq}^k + \beta_{ki}^k}{2})
    \\
    x_2 &= y \quad dV_y(u_q) = \varphi_q(\frac{1}{2} \alpha_{kq}^j + \beta_{kj}^j) - \varphi_q(\frac{\alpha_{kq}^j + \beta_{kj}^j}{2}), \text{ for } u_q \in \Delta^k_{qij} \\
    x_1 &= x \quad dW_x(u_q) = \varphi_q(2 \frac{\alpha_{kq}^k + \beta_{ki}^k}{2}) - \varphi_q(\frac{\alpha_{kq}^k + \beta_{ki}^k}{2})
    \\
    x_2 &= y \quad dW_y(u_q) = \varphi_q(2 \frac{\alpha_{kq}^j + \beta_{kj}^j}{2}) - \varphi_q(\frac{\alpha_{kq}^j + \beta_{kj}^j}{2})
\end{align*}
\]

(3.25)

![Figure 3.10: Scaling functions $dW_x$ (left) and $dW_y$ (right)](image)

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:

\[
\begin{align*}
    g_1(u_q) &= g_f(u_q + dW_x(u_q)) \\
    g_2(u_q) &= g_f(u_q + dW_x(u_q))
\end{align*}
\]

(3.26) (3.27)

In order to scale a function $f$ with $g$-function, $g_f$, a power of 2 along a chosen dimension,
the following compositions are performed. Let \( g_0 = g_f \), and let

\[
g_{N+1}(u_q) = g_N(u_q + dW_x(u_q)), \quad N = 0, 1, 2, \ldots
\]  

(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

\[
f_1(x_1, \ldots, x_i + \delta x_i, \ldots) = f(x_1, \ldots, A \cdot x_i, \ldots, x_n), \quad \text{where} \quad A = 2 \text{ or } A = \frac{1}{2}.
\]

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 $\delta x_1$. Let the left square be the $j$th square and the square to its right be $A_j$. The location of the center of square $j$ is $M_j = \alpha_k^q + \beta_k^q$, for $q = 0, 1, \ldots, 2n$. And the center for square $A_j$ is $M_{A_j} = A \cdot (\alpha_k^q + \beta_k^q)$ for $q = 0, 1, \ldots, 2n$. We assign $\delta x_1 = M_{A_j} - M_j = (A - 1) \cdot M_j$.

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

$$dU = u_{q,A_j} - u_{q,j}$$

$$= \left[ \sum_{l, l \neq 1} \lambda_l \varphi_q(x_l) + \lambda_1 \varphi_q(M_{A_j}) \right] - \left[ \sum_{l, l \neq 1} \lambda_l \varphi_q(x_l) + \lambda_1 \varphi_q(M_j) \right]$$

$$= \lambda_1 \varphi_q(M_{A_j}) - \lambda_1 \varphi_q(M_j)$$

$$= \lambda_1 [\varphi_q(M_{A_j}) - \varphi_q(M_j)]$$

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

$$g_1(u_{q,j}) = g_f(u_{q,A_j})$$

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

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

(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

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

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:

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

For the two dimensional case,

$$x_1 = x; \quad g_1(u_q) = \frac{1}{\delta x} [g_f(u_q + dU_x(u_q)) - g_f(u_q)]$$
$$x_2 = y; \quad g_1(u_q) = \frac{1}{\delta y} [g_f(u_q + dU_y(u_q)) - g_f(u_q)] \quad (3.34)$$

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

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

(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 $(n + 1) \cdot \min(g_f)$. 

34
**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 $(x_1, \ldots, x_n)$ and the minimum $f_{\min}$ at $(y_1, \ldots, y_n)$. 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 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 $\varphi_q$ are strictly increasing and belong to a class $\text{Lip} \alpha$, $\alpha > 0$. For each continuous function $f$ on $S$, one can find a continuous function $g(u), 0 \leq u \leq n$ such that

$$f(x_1, \cdots, x_n) = \sum_{q=0}^{2n} g(\lambda_1 \varphi_q(x_1) + \cdots + \lambda_n \varphi_q(x_n)) \quad (4.1)$$
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:**
  \[
  I^k_{qi} = I^k_i - 2q \cdot 10^{-k} = [\alpha^k_{qi}, \beta^k_{qi}] \tag{4.2}
  \]
  \[
  S^k_{qij} = I^k_{qi} \times I^k_{qj},
  \]
  
  \[q = 0, \ldots, 4; \ i = 0, 1, \ldots, 10^{k-1}; \ k = 1, 2, 3, \ldots\]

- **3-D case:**
  \[
  I^k_{qi} = I^k_i - 2q \cdot 14^{-k} = [\alpha^k_{qi}, \beta^k_{qi}] \tag{4.3}
  \]
  \[
  S^k_{qijl} = I^k_{qi} \times I^k_{qj} \times I^k_{ql},
  \]
  
  \[q = 0, \ldots, 6 \ i, j = 0, 1, \ldots, 14^{k-1}; \ k = 1, 2, 3, \ldots\]

The number intervals are related to the number of \(\varphi_q\) 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 \(\varphi_q\) where \(q = 7\), these would be exactly the same intervals as \(q = 0\). Thus the following generalization for higher dimensional cases is proposed:

\[
I^k_i = \left[ i \cdot (2s)^{-k+1} + (2s)^{-k}; i \cdot (2s)^{-k+1} + (2s - 1) \cdot (2s)^{-k} \right] \tag{4.4}
\]

for \(k = 1, 2, \ldots\);

and \(i = 0, \ldots, (2s)^{k-1}\);
In order to maintain this cyclic behavior of shifting the intervals, one must also shift the
intervals in such a way that the \((2n+1)^{th}\) \(\varphi_q\) function is the same as the \(\varphi_0\). 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

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

for \(k = 1, 2, \ldots\);

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

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 \(\varphi_q\) are associated with the intervals

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

38
for \( k = 1, 2, \ldots; \)
and \( i = 0, \ldots, (2s)^{k-1}; \)
which are obtained from the \( I^k_i \) 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 \( I \cap I^k_i. \)

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 \( \alpha^l_{qi}, \beta^l_{qi} \) 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 \( \alpha^k_{qi}, \beta^k_{qi} \) different from 0, 1 coincides with an endpoint of lower rank.

The \( \varphi_q \) will be strictly increasing. The most important property will be that of mapping the intervals \( I^k_i \) 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 \( I^k_i \) under the \( \varphi_q \) 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 \( \varphi_q, q = 0, \ldots, 2n, \) defined on \([0, 1]\) and belonging to a class \( \text{Lip} \alpha, \alpha > 0 \) such that for each fixed rank \( k = 1, 2, \ldots, \) the
\[ \Delta^k_q = \left[ \sum_{i=1}^{n} \lambda_i \varphi_q(\alpha^k_{qj_i}), \sum_{i=1}^{n} \lambda_i \varphi_q(\beta^k_{qj_i}) \right] \text{ for } q = 0, \cdots, 2n; \] (4.7)

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

all are disjoint.

**Proof.** The functions \( \varphi_q \) will be constructed iteratively. We put \( \alpha^0_{q0} = 0, \beta^0_{q0} = 1 \), 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 \( \varphi_q, q = 0, \cdots, 2n \), at the end points \( \alpha^k_{qj_i}, \beta^k_{qj_i} \) 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 \( \varphi_q, q = 0, \cdots, 2n \), is strictly increasing on the set of all points \( \alpha^l_{qj_i}, \beta^l_{qj_i}, 0 \leq i \leq (2s)^{l-1}, l \leq k \).

- **ii** The slope of each segment of the polygonal line connecting the points \( (x, \varphi_q(x)), x = \alpha^l_{qj_i}, \beta^l_{qj_i}, l \leq k \), is strictly smaller than \( s^k \).

The proceeding properties involve each single function \( \varphi_q \); the following properties involve all \( 2n + 1 \) functions.

- **iii** All values \( \varphi_q(\alpha^l_{qj_i}), \varphi_q(\beta^l_{qj_i}), q = 0, \cdots, 2n, 0 \leq i \leq (2s)^{k-1}, l \leq k \), are rational and distinct.

- **iv** The intervals (4.7) are disjoint.
Assume that the \( \varphi_q \) are defined at all endpoints \( \alpha^l_{q_i}, \beta^l_{q_i} \) 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 \( \alpha^k_{q_i}, \beta^k_{q_i} \) in \((0, 1)\) of rank \( k \). First we construct a (not strictly) increasing extension of \( \varphi_q \) 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 \( \alpha^l_{q_i}, \beta^l_{q_i}, l < k \), where the function \( \varphi_q \) is already known. We have observed that each “new” point \( \alpha^k_{q_i}, \beta^k_{q_i} \) in \((0, 1)\) is different from each of the old points \( \gamma \). The intervals \( I^k_{q_1} \) 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 \( I^k_{q_i} \) cover the remaining \( \frac{1}{s} \) of this length. The same applies to each interval \((\gamma, \gamma')\). The intervals \( I^k_{q_i} \) are of two kinds. The intervals of the first kind contain one point \( \gamma \); we call it \( \gamma^k_{q_i} \); the interval of the second kind contains no points \( \gamma \).

For each interval \( I^k_{q_i} \), we shall take \( \varphi_q(\alpha^k_{q_i}) = \varphi_q(\beta^k_{q_i}) = \varphi^k_{q_i} \). For the intervals of the first kind, we select \( \varphi^k_{q_i} \) equal to \( \varphi_q(\gamma^k_{q_i}) \). To define \( \varphi^k_{q_i} \) for the intervals \( I^k_{q_i} \) of the second kind, we select the values of \( \varphi_q \) at the endpoints \( \alpha^k_{q_i}, \beta^k_{q_i} \). Let \( \gamma, \gamma' \) (i.e. \( \alpha, \alpha' \) or \( \beta, \beta' \)) be two adjacent points \( \gamma_q \). We define \( \varphi_q \) at the end points of the intervals \( I^k_{q_i} \) of the second kind contained in \((\gamma, \gamma')\). It is sufficient to determine the increase of \( \varphi_q \) on each gap (or part of a gap) \((x, x')\) we assign an increase of \( \varphi_q \), 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 $\varphi_q$ satisfies (ii).

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

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

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

are distinct. Let $\epsilon > 0$ be so small that the $2\epsilon$-neighborhoods of the points (4.9) are disjoint. This allows us to define the remaining values of the $\varphi_q$. For each interval $I_{q_i}^k$, we select $\varphi_q(\alpha_{q_i}^k), \varphi_q(\beta_{q_i}^k)$ in the $\epsilon$-neighborhood of $\varphi_{q_i}^k$ in such a way that $\varphi_q(\alpha_{q_i}^k) < \varphi_{q_i}^k < \varphi_q(\beta_{q_i}^k)$. (If one of the endpoints is 0 or 1, this inequality has to be changed correspondingly; for example, if $\beta_{q_i}^k = 1$, it should read $\varphi_q(\alpha_{q_i}^k) < \varphi_{q_i}^k = \varphi_q(\beta_{q_i}^k)$. 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 $2\epsilon$-neighborhood of the corresponding point (4.9). This completes the $k$th step.

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

$$
\varphi_q \in \text{Lip } \alpha, \alpha = \log_{2s} 2 \tag{4.10}
$$
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 \( \alpha^k_{qi}, \beta^k_{qi} \). Hence, it is contained in an interval of length \((2s)^{-k+1}\) between some two such points. Therefore by (i) and (ii), \( \varphi(x + h) - \varphi(x) < s^k \cdot (2s)^{-k+1} = (2s) \cdot 2^{-k} = (2s) \cdot (2s)^{-ak} \leq (2s)(sh)^\alpha = (2s) \cdot s^\alpha h^\alpha \) if \( \alpha \) is defined by \( 2 = (2s)^\alpha \). This proves (4.10).

### 4.1.2 Further Details required for the Proof

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

\[
S^k_{qj_1j_2\cdots j_n} = \prod_{i=1}^{n} I^k_{qji}
\]  

(4.11)

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

The image of (4.11) under the map \((x_1, \ldots, x_n) \longrightarrow \sum_{i=1}^{n} \lambda_i \varphi_q(x_i)\) is the interval \( \Delta^k_q \) 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 \( I^k_{qi} \) form \((2n+1)\) families, \( q = 0, \ldots, 2n \). 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 \( I^k_{qi} \). Therefore each point \((x_1, x_2, \ldots, x_n)\) 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 \((x_1, x_2, \ldots, x_n)\) out of \( 2n + 1 \) tries, \( q = 0, \ldots, 2n \).”
For each continuous function $g(u)$, defined for $0 \leq u \leq n$, we put

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

**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

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

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 $\epsilon$ so small that $\frac{n}{n+1} + \epsilon \leq \theta$. Let $f^k_{q_1 j_2 \cdots j_n}$ represent the value of $f(x_1, \cdots, x_n)$ in the center of the cube $S^k_{q_1 j_2 \cdots j_n}$. On each interval $\Delta^k_q$ take $g(u)$ constant and equal to $\frac{1}{n+1} f^k_{q_1 j_2 \cdots j_n}$. Next, $g(u)$ can be extended linearly in the gaps between the $\Delta^k_q$. 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 $(x_1, \cdots, x_n) \in S^k_{q_1 j_2 \cdots j_n}$, for some $j_1, j_2, \cdots, j_n$. The corresponding terms of the sum in (4.12) have the values $\frac{1}{n+1} f^k_{q_1 j_2 \cdots j_n}$, 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

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

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

(4.15)

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

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

(4.16)

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

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

(4.17)

In general the functions $g_r, h_r$ will satisfy

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

(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 $N$:

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

(4.19)

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

(4.20)

Letting $N \to \infty$ gives us,

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

(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 $\varphi$’s in two dimensional case

It is unlikely that the $\varphi$ 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 $\varphi_q$, $q = 0, 1, \ldots, 4$, 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 $0 < \varphi_q(\alpha_{q0}) < \varphi_q(\beta_{q0}) < 1$.

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

- $\varphi_q$'s are defined on $\Gamma = \{\alpha_{qi}, \beta_{qi}\}$ for all $l < k$. These are the “old points”.

- Need to define $\varphi_q$ at “new points”: $\alpha_{ki}, \beta_{ki}$.

- Let $\gamma$ denote an arbitrary point from $\Gamma$.

- For $i = 0, 1, \ldots, 10^{k-1}$, consider two cases.

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

  - $\varphi_q(\alpha_{qi}) = \varphi_q(\gamma)$

  - $\varphi_q(\beta_{qi}) = \varphi_q(\gamma)$

  **else**

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

  - find the smallest $\gamma'$ such that $\beta_{qi} < \gamma'$

  - find the largest $\gamma$ such that $\alpha_{qi} > \gamma$

  - collect all new points $\alpha$ and $\beta$ 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

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

  **end if**
• Final step (making the \( \varphi \)'s strictly increasing). Pick a value for \( \lambda \) (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 \( d = \text{minimum distance between these points} \). 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 \( \varphi \) functions will satisfy the following conditions:

1. The function \( \varphi_q \) 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

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

are disjoint.

The key roles played by the \( \varphi \)-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 $\varphi$-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 $\varphi_q$ values for $k > 0$

One of the main difficulties in assigning values to $\varphi_q$ occurs in determining the smallest $2\epsilon$ neighborhood around each point $\sum_i \lambda_i \varphi_q(x_i)$, so that they are distinct. In our implementation, the values from $\sum_i \lambda_i \varphi^{k}_{qi}$ are ordered on a number line. Next the smallest distance between each point is calculated. Once this value $\epsilon$ 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 $P^k_2 > P^k_1 > 0$ where $P^k_m = \sum_{j=1}^n \lambda_j \varphi^{k}_{q_i j}$ be two points for rank $k > 1$. Let $d_k = |P^k_2 - P^k_1|$. Then a useful smallest distance between any two points is given by $d^*_k$.

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

**Proof.** Let $P^k_2 = \alpha^*_k P^k_1$, $\alpha_k > 1$ Now let $d^*_k = \text{the distance between points } P^k_2, \ P^k_1 \text{ the two closest points on rank k}$. In this way $d_k \geq d^*_k$ for all $d_k$.

Note $P^k_2 = \alpha^*_k P^k_1$, $\alpha^*_k > 1$

So we have,
\[ d_k^* = \vert P_{2k}^* - P_{1k}^* \vert \]  
\[ = \left( 1 - \frac{1}{\alpha_k^*} \right) \vert P_{2}^* \vert \]  
\[ \geq \left( 1 - \frac{1}{\xi_k} \right) \vert P_{2k}^* \vert \]  
\[ = \left( 1 - \frac{1}{\xi_k} \right) \left\vert \sum_{j=1}^{n} \lambda_j \varphi_{q,j}^k \right\vert \]  
\[ \geq \left( 1 - \frac{1}{\xi_k} \right) \lambda^* \left\vert \sum_{j=1}^{n} \varphi^* \right\vert \]  
\[ = \left( 1 - \frac{1}{\xi_k} \right) \lambda^* n \varphi^* \]  
\[ \text{(5.4)} \]

where

\[ \lambda^* = \min \{ \lambda_j \} \]

\[ \varphi^* = \min \{ \varphi_{q,j}^k \} \]

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 \( \xi_k \) where \( \xi = \sqrt{\alpha_k^*} \geq \sqrt{\frac{1}{\alpha_k}} > \sqrt{\frac{1}{\alpha_{k-1}}} = \xi_k \). \( \xi_k \) depends on the previous rank. So starting with \( \alpha_1^* \) from rank \( k = 1 \), one can find a minimum distance for higher ranks and with the above choice of \( \epsilon_k \) 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.
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 \textit{g-function representation}

The format for storing the values are in Table (5.1). For the values listed in the following table, the parameter $\lambda$ 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 $\phi_q$ 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 $\phi_q$ 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 $S_{qij}^k$ 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
<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>$\phi_q(\gamma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/100</td>
<td>28302692683477/123629556035400</td>
</tr>
<tr>
<td>9/100</td>
<td>28303092025807/123629556035400</td>
</tr>
<tr>
<td>1/10</td>
<td>13249661/28937850</td>
</tr>
<tr>
<td>11/100</td>
<td>28319280710747/61814778017700</td>
</tr>
<tr>
<td>31/100</td>
<td>9461744492499/20604926005900</td>
</tr>
<tr>
<td>39/100</td>
<td>4730905524777/10302463002950</td>
</tr>
<tr>
<td>41/100</td>
<td>7104552465218/15453694504425</td>
</tr>
<tr>
<td>49/100</td>
<td>28418409532037/61814778017700</td>
</tr>
<tr>
<td>51/100</td>
<td>28451186244247/61814778017700</td>
</tr>
<tr>
<td>59/100</td>
<td>7112846478853/15453694504425</td>
</tr>
<tr>
<td>81/100</td>
<td>7137528848593/15453694504425</td>
</tr>
<tr>
<td>89/100</td>
<td>28550315065537/61814778017700</td>
</tr>
<tr>
<td>9/10</td>
<td>13373161/28937850</td>
</tr>
<tr>
<td>91/100</td>
<td>85436099526761/123629556035400</td>
</tr>
<tr>
<td>99/100</td>
<td>85436498869091/123629556035400</td>
</tr>
<tr>
<td>1</td>
<td>23/25</td>
</tr>
</tbody>
</table>

Table 5.1: Values for $\phi_0(x)$ function define over $[0, 1]$
<table>
<thead>
<tr>
<th>$u_{ij}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3864965751659/494518224141600 * $2^{1/2}$ + 3864965751659/247259112070800</td>
</tr>
<tr>
<td>3865764436319/494518224141600 * $2^{1/2}$ + 3865764436319/247259112070800</td>
</tr>
<tr>
<td>3996871285159/494518224141600 * $2^{1/2}$ + 3864965751659/247259112070800</td>
</tr>
<tr>
<td>3997669969819/494518224141600 * $2^{1/2}$ + 3865764436319/247259112070800</td>
</tr>
<tr>
<td>3864965751659/494518224141600 * $2^{1/2}$ + 3996871285159/247259112070800</td>
</tr>
<tr>
<td>3865764436319/494518224141600 * $2^{1/2}$ + 3997669969819/247259112070800</td>
</tr>
<tr>
<td>1376258939553/164839408047200 * $2^{1/2}$ + 3864965751659/247259112070800</td>
</tr>
<tr>
<td>1376525167773/164839408047200 * $2^{1/2}$ + 3865764436319/247259112070800</td>
</tr>
<tr>
<td>3996871285159/494518224141600 * $2^{1/2}$ + 3996871285159/247259112070800</td>
</tr>
<tr>
<td>3997669969819/494518224141600 * $2^{1/2}$ + 3997669969819/247259112070800</td>
</tr>
</tbody>
</table>

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

---
Table 5.3: Interpolation table for \((u_{qij}, g(u_{qij}))\), here the \(f_k^{qij}\)'s are the values at the centers of the squares \(S_k^{qij}\) associated with \(u_k^{qij}\) 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 $h(x, y)$.

![Figure 6.1: $f(x, y) = 1$](image)

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 $f(x, y)$. 

58
We can apply the Kolmogorov approximation to reconstruct an image from the trained $g$ function. Using our computed $\varphi$ functions, for each image $f$, we can construct its $g$-function. The $\varphi$ 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.
3-D Experiments  Unlike in 2-D, verification by graphing functions is not straightforward 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 6.6 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)
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 thresholding.
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 its 1-D Kolmogorov representation

3. The 1-D representation is represented as a wavelet transformation

4. The wavelet coefficients are thresholded with the default threshold 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 white.

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.

![Dinosaur images](image)

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]).
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. Further 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:

\[ p(d|\theta_1 \cdots \theta_n) = \frac{\prod_m p_m(d|\theta_m)}{\sum_c \prod_m p_m(c|\theta_m)} \]  

(6.1)

where \( d \) is a data vector, \( \theta_m \) 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:

\[ \frac{\partial \log p(d|\theta_1 \cdots \theta_n)}{\partial \theta_m} = \frac{\partial \log p_m(d|\theta_m)}{\partial \theta_m} - \sum_c p(c|\theta_1 \cdots \theta_n) \frac{\partial \log p_m(c|\theta_m)}{\partial \theta_m}. \]  

(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]

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

(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 $\theta_i$ and the experts $\Phi_i$ are defined as before and $x_k$’s represent the cliques in the image.

### 6.3.3 PoE and FoE revisited

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 $\varphi$ functions and constants $\lambda$’s in (4.1), Kolmogorov’s theorem implies the following: For every function $f$ of $n$ variables, there are functions $g_0, g_1, \ldots, g_N$, and functions $\psi_0, \psi_1, \ldots, \psi_N$, all of single variable, for some $N \leq 2n + 1$, such that

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

(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

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

(6.7)

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

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

(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 $\psi_q$ 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 $\varphi$ functions in the representation (3.1) given in Kolmogorov’s theorem:

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

(6.9)
Finally, the mixture of Gaussians model is exactly when \( \varphi \)'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 \( \varphi \) functions and constants \( \lambda \)'s.

Let \( n \) be the dimension of the image size.

**Step 1** Define \( n \) constants \( \lambda_i \) that are independent over the rational field \( \mathbb{Q} \).

**Step 2** Compute the \( 2n + 1 \) \( \varphi \)-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 \( \lambda_i \) such that the relation

\[
a_1 \lambda_1 + a_2 \lambda_2 + \cdots + a_n \lambda_n = 0,
\]

(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 $\varphi_q$ 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 $\varphi_q$ functions
- ndimension - the dimensions space we are constructing the $\varphi_q$ 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 $2n + 1 \varphi_q$ functions as an array of two column tables with columns $\gamma$ 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 $u$ 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 $\varphi_q$ functions

B.1.1 buildphis_hp

0001 function [phiq, u ] = buildphis_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 %
0012 % dprec (scalar) = indicates how many decimal places
0013 % to keep in
0014 %
0015 % endpoints (matrix [2*ndimensions 2]) = predefined
0016 %
0017 % endpoints 0 & 1 . matrix must be
0018 %
0019 %
% output:

phi ([m,n,q] matrix array) = table of values used to interpolate values for phi(x)

required to calculate \( g(\phi(x) + \lambda \phi(y)) \)

\( u ([p,1] column vector)=\) table of values for which \( g \) will be defined. \( u \) specifies the domain of \( g \)

This function builds the \( \phi_q \) functions and stores them in table \( \phi \)

each \( \phi \) will have the form:

\[
\begin{array}{ccc}
\text{point value} & \text{interval}\_\text{left}\_\text{value} \\
0.1 & 7/32 \\
0.9 & 9/32 \\
\end{array}
\]

values will be in order sorted according to point_value;

each \( u \) table will have the form:

\[
\begin{array}{ccc}
\text{point value} & q & i & j \\
\phi(\alpha\_qik) + \lambda \phi(\alpha\_qjk) \\
\end{array}
\]
\[ \phi(\beta_{qik}) + \lambda \phi(\beta_{qjk}) \]

values will be in order sorted according to point_value;

%Initializations
% construct non decreasing functions \( \phi_q \)
% define constants used in program
% set the number of spacial dimensions to default 2-D \( f(x,y) \)
if (nargin < 2)
    spacial_dimensions = 2; \% number of dimensions for domain of function
else
    spacial_dimensions = ndimension;
end;
qmax = 2*spacial_dimensions;
% set value for \( \delta \)
if (nargin <3)
    delta = sym(1/64); \% 32768); \% define small
else

delta = sym(deltain);
end;

% set maple precision to desired number of digits
if (nargin < 4)
dprec = 100;
maple(['Digits :=',char(sym(dprec))]);
else
dprec = dprecin;
maple(['Digits :=',char(sym(dprec))]);
end;

%define phi_q for k =0
if (nargin < 5)
phi = sym([0 92/100]); % this is phi_0
for q=1:qmax % remember in this language indices start at 1
    phi(:,:,q+1) = phi(:,:,1) + (q)*delta;
end;
else
    % dimension should be [2*ndimensions 2]; give error otherwise
    phi = sym(endpoints01);
end;
final_value = sym(1);

lambda = sym([1 1/sqrt(2)]);

% now build phi_q for all k according to theorem, one q at a time

for k =1:kmax
    for q=0:qmax
        tble = findfirst_hp(spacial_dimensions,q,-k);
        tble = sortrowshp(tble,2);
        SIZETBLE= size(tble);
        temp = sym(zeros(1,2*(10^(k-1)+1))); %make the correct number of endpoints

        if (SIZETBLE(1) > 0)
            for m=1:SIZETBLE(1)
                if (tble(m,8) == 1) % this is an alpha endpoint
                    % contained in I^k_i
                    pnt_k =double(2*(tble(m,2))); % this is done to handle %k=0 gamma
                end
            end
        end
    end
end

87
l = double(tble(m,7));

temp(1, pnt_k+1) =phi(l+1,pnt_l+1,q+1);

temp(1, pnt_k+2) =phi(l+1,pnt_l+1,q+1);

elseif (tble(m,8) == 9) %this is an beta endpoint

  % contained in I_k_i

  pnt_k = double(2*(tble(m,2))); 

  pnt_l = double(2*(tble(m,6))); 

  l = double(tble(m,7));

  temp(1, pnt_k+1) =phi(l+1,pnt_l+2,q+1);

  temp(1, pnt_k+2) =phi(l+1,pnt_l+2,q+1);

end;

end;

division of the first kind have been assigned values

% assign values to intervals of the second kind here

% create gammatable

temp2 = sym([]);

for lp=0:(k-1)

  for ip=0:(floor(10^(lp-1)))

    if (lp > 0)
alphaq = sym((10^(-lp))*(10*ip+1-2*q));

else
alphaq = sym(0);
end;

temp2 = [temp2; alphaq ip lp 1 phi(lp+1,2*ip+1,q+1)];

if (lp > 0)
betaq = sym((10^(-lp))*(10*ip+9-2*q));
else
betaq = sym(1);
end;

temp2 = [temp2; betaq ip lp 9 phi(lp+1,2*ip+2,q+1)];
end;

gammatble = sortrowshp(temp2,1);
SIZEGTBLE = size(gammatble);

% now assign values to interval i=0
i = 0;

% check to see if interval is of the first kind
found = false;

if (SIZEGTBLE(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       gammaprime = gammatble(j,1);  % contains I_q_0_k
0160       gammapqil = gammatble(j+1,1);
0161       alphaqik = sym((10^(-k))*((10*i+1)-2*q));
0162       betaqik = sym((10^(-k))*((10*i+9)-2*q));
0163       expr1 = strcmpi(maple('evalb', [char(gammaqil), ' < ', ...
0164           char(alphaqik)]), 'true');
0165       expr2 = strcmpi(maple('evalb', [char(alphaqik), ' < ', ...
0166       ...
char(gammapqil)), 'true');
expr3 = strncmpi(maple('evalb', [char(gammaqil), ' < ', ...
char(betaqik)), 'true');
expr4 = strncmpi(maple('evalb', [char(betaqik), ' < ', ...
char(gammapqil)), 'true');
if ((expr1 && expr2) && (expr3 && expr4))
  if ((( gammaqil < alphaqik) && (alphaqik < gammapqil)) &&
  (( gammaqil < betaqik) && (betaqik < gammapqil)))
    if (gammatble(j,4) ==1 )
      phi_gammas =phi(double(gammatble(j,3))+1,2*...
      double(gammatble(j,2))+1,q+1);
    else
      phi_gammas =phi(double(gammatble(j,3))+1,2*...
      double(gammatble(j,2))+2,q+1);
  end;
if (gammatble(j+1,4) ==1 )
  phi_gammap = phi(double(gammatble(j+1,3))+...
  1,2*double(gammatble(j+1,2))+1,q+1);
else
  phi_gammap = phi(double(gammatble(j+1,3))+...
  1,2*double(gammatble(j+1,2))+2,q+1);
x = gammatble(j,1);
xp = alphaqik;

phi_x = phi(double(gammatble(j,3))+...
1,2*double(gammatble(j,2))+1,q+1);
temp(1,2*i+1)= phi_x +5 *...
((xp-x)/(gammapqil -gammaqil))*...
(phi.gammap -phi.gamma);
temp(1,2*i+2) = temp(1,2*i+1);
foundg = true;

if (~foundg)
    j=j+1;
end;
if (~foundg) % if no interval was found then something is
    % wrong
    disp('Interval gammas was not found');
end;

% assign values for remaining intervals i>0 according to theorem
i = i+1; % go to the next interval i, which is i = 1
move_on = false; % indicates whether to move to the next
            %gamma, gammap interval
j=1;
while (i <= (10^(-k-1)))
    if (j < SIZEGTBLE(1)) % this is mainly for
        %q=0 k =1 i =1 interval
        gammaqil = gammatble(j,1);
        gammadqil = gammatble(j+1,1);
    else
        gammaqil = gammatble(j,1);
        gammadqil = sym(2);
    end;
alphaqik = sym((10^(-k))*10*i+1-2*q);
betaqik = sym((10^(-k))*10*i+9-2*q);
expr1 = strcmpi(maple('evalb', [char(gammaqil), ' < ', ...
    char(alphaqik)]), 'true');
expr2 = strcmpi(maple('evalb', [char(alphaqik), ' < ', ...
    char(gammadqil)]), 'true');
expr3 = strcmpi(maple('evalb', [char(gammaqil), ' < ', ...
    char(betaqik)]), 'true');
expr4 = strcmpi(maple('evalb', [char(betaqik), ' < ', ...

char(gammapqil)]), 'true');

if ((expr1 && expr2) && (expr3 && expr4))

if ((gammaqil < alphaqik) && (alphaqik < gammapqil)) &&

(( gammaqil < betaqik) && (betaqik < gammapqil))

if (j < SIZEGTBLE(1)) % this is mainly for

if (gammatble(j,4) ==1) % test to see if it's

%alpha or beta

phi_gamma = phi(double(gammatble(j,3))+...

1,2*double(gammatble(j,2))+1,q+1);

else

phi_gamma = phi(double(gammatble(j,3))+...

1,2*double(gammatble(j,2))+2,q+1);

end;

if (gammatble(j+1,4) ==1) % test to see if

%it's alpha or beta

phi_gammap = phi(double(gammatble(j+1,3))+...

1,2*double(gammatble(j+1,2))+1,q+1);

else

phi_gammap = phi(double(gammatble(j+1,3))+...
1,2*double(gammatble(j+1,2))+2,q+1);

end;

else

if (gammatble(j,4) == 1) % test to see if
% it's alpha or beta

phi.gamma = phi(double(gammatble(j,3))+
1,2*double(gammatble(j,2))+1,q+1);

else

phi.gamma = phi(double(gammatble(j,3))+
1,2*double(gammatble(j,2))+2,q+1);

end;

phi.gamma = final_value; % can change to suit needs

end;

alphaq_i_k = sym((10^(-k))*(10*(i-1)+1-2*q));

betaq_i_k = sym((10^(-k))*(10*(i-1)+9-2*q));

% check to see which point is closer beta_i-1 or gamma

expr1 = strcmpi(maple('evalb', [char(gammaqil), ' < ',
char(betaq_i_k)]),'true');

if (expr1)

if (( gammaqil < betaq_i_k))

x = betaq_i_k; % beta_q(i-1)k

phi_x = temp(1,2*(i-1)+2);

else

    x = gammatble(j,1);
    phi_x = phi_gamma;
end;

xp = alphaqik;

temp(1,2*i+1)= phi_x +5 *((xp-x)/(gammapqil -gammaqil))*(phi_gammap -phi_gammap);
temp(1,2*i+2) = temp(1,2*i+1);
move_on = false;
i = i+1;

else

    move_on = true;

%check to see if it’s an interval of the 1st kind

w =1;
found = false;

while ((w <= SIZETBLE(1)) && (~found))

    if (tble(w,1) == q) && (tble(w,2) ==i) && ...
        (tble(w,3)==k)
        found = true;
end;
w = w+1;
end;
if (found)
i= i+1;
end;
end;
if ((j < SIZEGTBLE(1)) && (move_on))
j = j+1;
move_on = false;
end;
end;
% attach newly constructed rank to phi
if (q==0) % only need to expand matrix once for q=0 for q> 0
%elements
SIZEPHI = size(phi); % already exist
SIZETEMP = size(temp);
phi(k+1,SIZETEMP(2),q+1)= 0;
phi(k+1,:,q+1) = temp;
else
SIZETEMP = size(temp);
SIZEPHI = size(phi);

if (SIZEPHI(2) > SIZETEMP(2)) % add space if temp has
    % smaller dimensions
    temp(1,SIZEPHI(2)) = 0;
end;

phi(k+1,:,q+1) = temp;
end;

% create temporary u_q = sum (lambda_i * phi_q(x_i)) intervals
% in order to determine epsilon
Onesh = ones(1, (10^(k-1))+1);
SIZEL = size(lambda);
Top = phi(k+1,:,q+1);
SIZETOP= size(Top);
Z = zeros((SIZETOP(2)/2), SIZETOP(2));
SIZEZ = size(Z);
for z =1:SIZEZ(1)
    Z(z,2*z) = 1;
end;
Z = Z';
Top = Top*Z;
Side= Top';
Sab = lambda(2)*Side*Onesh;

SIZESAB = size(Sab);

Tab = lambda(1)*ones(SIZESAB(1),1)*Top;

Summ = Sab +Tab;

SIZESUMM = size(Summ);

NEWSIZE = SIZESUMM(1)*SIZESUMM(2);

Side = reshape(Summ, NEWSIZE,1);

if (SIZEL(2) >= 3)
    for l = 3:SIZEL(2)
        Sab = Side*Onesh;
        SIZESAB = size(Sab);
        Tab = lambda(l)*ones(SIZESAB(1),1)*Top;
        Summ = Sab +Tab;
        SIZESUMM = size(Summ);
        NEWSIZE = SIZESUMM(1)*SIZESUMM(2);
        Side = reshape(Summ, NEWSIZE,1);
    end;
end;
T1 = sort(vpa(Side));
disp(Side);
disp(T1);
T2 = circshift(T1,-1);
dvec = sort(abs(T2-T1));
dmin = dvec(1);
disp(dvec);

% assign epsilon to dmin/4 so that the 2-epsilon neighborhoods of the
% points are disjoint
epsilon = sym(maple('convert', ['evalf(', char(dmin/4),')', ')'],
               'rational','10'));
epsilon = vpa(dmin/4,WDTH);
disp(['epsilon = ' char(epsilon)]);
for ip =0:(10^(k-1))
phi(k+1,2*ip+1,:) = phi(k+1,2*ip+1,:) - epsilon;
phi(k+1,2*ip+2,:) = phi(k+1,2*ip+2,:) + epsilon;
end;
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)), ', <= ', 0]));
      end
    end
  end
end
char(alphaqik]), 'true');
expr2 = strcmpi(maple('evalb', [char(alphaqik), ' <= ', sym(1)]), 'true');
if (expr1 && expr2)
%if ((0<= alphaqik) && (alphaqik <= 1))
phia = phi(k+1,2*i+1,q+1);
phia{q+1}.table = [phia{q+1}.table ; alphaqik phia];
end;
expr1 = strcmpi(maple('evalb', [char(sym(0)), ' <= ', sym(1)]), 'true');
expr2 = strcmpi(maple('evalb', [char(betaqik), ' <= ', sym(1)]), 'true');
if (expr1 && expr2)
%if ((0<= betaqik) && (betaqik <= 1))
phib = phi(k+1,2*i+2,q+1);
phia{q+1}.table = [phia{q+1}.table ; betaqik phib];
end;
end;
end;
end;
temp = sortrowshp(phia{q+1}.table,1);
0419 phi{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),' < ', ... char(sym(0)))], 'true');
0435         expr2 = strcmpi(maple('evalb',[char(alphaqik),' > ', ... char(sym(1)))], 'true');
0436         expr3 = strcmpi(maple('evalb',[char(betaqjk),' < ', ... char(sym(0)))], 'true');

103
expr4 = strcmpi(maple('evalb', [char(alphaqjk), ' > ', char(sym(1))]), 'true');
expr5 = strcmpi(maple('evalb', [char(sym(0)), ' < ', char(alphaqik)]), 'true');
expr6 = strcmpi(maple('evalb', [char(alphaqik), ' < ', char(sym(1))]), 'true');
expr7 = strcmpi(maple('evalb', [char(sym(0)), ' < ', char(betaqik)]), 'true');
expr8 = strcmpi(maple('evalb', [char(betaqik), ' < ', char(sym(1))]), 'true');
expr9 = strcmpi(maple('evalb', [char(sym(0)), ' < ', char(alphaqjk)]), 'true');
expr10 = strcmpi(maple('evalb', [char(alphaqjk), ' < ', char(sym(1))]), 'true');
expr11 = strcmpi(maple('evalb', [char(sym(0)), ' < ', char(betaqjk)]), 'true');
expr12 = strcmpi(maple('evalb', [char(betaqjk), ' < ', char(sym(1))]), 'true');
if ((expr1) || (expr2) || (expr3) || (expr4))
if ((double(betaqik) < 0) || (double(alphaqik) > 1) || ...
if ((double(betaqjk) < 0) || (double(alphaqjk) > 1))
%do nothing ; this is case were square is completely outside
%(0,1) x (0,1)

elseif (((expr5) && (expr6)) && ((expr7) && (expr8)) && ...
((expr9) && (expr10)) && ((expr11) && (expr12)))

% (((0< double(alphaqik)) && (double(alphaqik) < 1)) ...
% && ((0< betaqik) && (betaqik < 1))...
% && ((0< alphaqjk) && (alphaqjk < 1))...
% && ((0< betaqjk) && (betaqjk < 1)))

% completely inside (0,1) x (0,1)

phiai = phi(k+1, 2*i+1, q+1);
phiaj = phi(k+1, 2*j+1, q+1);
phibi = phi(k+1, 2*i+2, q+1);
phibj = phi(k+1, 2*j+2, q+1);
phia = phiai + lambda(2)*phiaj;
phib = phibi + lambda(2)*phibj;
u = [u; phia q i j];
u = [u; phib q i j];

else

% partially inside (0,1) x (0,1)

phiai = phi(k+1, 2*i+1, q+1);
phiaj = phi(k+1, 2*j+1, q+1);
phibi = phi(k+1, 2*i+2, q+1);
phibj = phi(k+1, 2*j+2, q+1);
expr1 = strcmpi(maple('evalb', [char(alphaqik), ... ', char(sym(0))]), 'true');
expr2 = strcmpi(maple('evalb', [char(sym(0)), ... ', char(betaqik)], 'true');
expr3 = strcmpi(maple('evalb', [char(betaqik), ... ', char(sym(1))]], 'true');
if ((expr1) && (expr2 && expr3))
  if ((alphaqik < 0) && ((0 < betaqik) && (betaqik < 1))
    phiai = phi(1,2*i+1,q+1);
    phibi = phi(k+1,2*i+2,q+1);
  end;
expr1 = strcmpi(maple('evalb', [char(sym(0))], ... ', char(alphaqik)], 'true');
expr2 = strcmpi(maple('evalb', [char(alphaqik), ... ', char(sym(1))]], 'true');
expr3 = strcmpi(maple('evalb', [char(betaqik), ... ', char(sym(1))]], 'true');
if ((expr1 && expr2) && (expr3))
if (((0 < alphaqik) && (alphaqik < 1)) && (betaqik > 1))

phiai = phi(k+1,2*i+1,q+1);

phibi = phi(1, 2*0+2, q+1);

end;

expr1 = strcmpi(maple('evalb', [char(alphaqjk), ...
                      ', char(sym(0))]), 'true');

expr2 = strcmpi(maple('evalb', [char(sym(0)), ...
                      ', char(betaqjk)]), 'true');

expr3 = strcmpi(maple('evalb', [char(betaqjk), ...
                      ', char(sym(1))]), 'true');

if ((expr1) && (expr2 && expr3))

if ((alphaqjk < 0 ) && ((0 < betaqjk) &&(betaqjk < 1))

phiaj =phi(1,2*0+1,q+1);

phibj =phi(k+1,2*j+2, q+1);

end;

expr1 = strcmpi(maple('evalb', [char(sym(0)), ...
                      ', char(alphaqjk)]), 'true');

expr2 = strcmpi(maple('evalb', [char(alphaqjk), ...
                      ', char(sym(1))]), 'true');

expr3 = strcmpi(maple('evalb', [char(betaqjk), ...
                      ', char(sym(1))]), 'true');
if ((expr1 && expr2) && (expr3))

% if (((0 < alphaqjk) && (alphaqjk < 1)) && (betaqjk > 1))

phiaj = phi(k+1, 2*j+1, q+1);
phibj = phi(1, 2*0+2, q+1);
end;

phia = phiai + lambda(2)*phiaj;
phib = phibi + lambda(2)*phibj;

u = [u; phia q i j];
u = [u; phib q i j];
end;
end;
end;
end;
temp = u;
u = sortrowshp(temp, 1);

B.1.2 findfirst_hp

function [tble] = findfirst_hp(n, qwanted, kmax)

% findfirst(n, qwanted, kmax)

% input: qwanted= which phi_q for which to find intervals of first find
set qwanted = -1 to find for all phi_q

kmax = find intervals of the first kind up to rank kmax

set kmax to negative value to get just the intervals of
the first kind only for rank kmax. abs(kmax) >=1.

n = the dimension space

output: tble = contains all the points which are of the first kind

each row has the form

[q i k alphaqik betaqik j l 9 betaqjl] or

[q i k alphaqik betaqik j l 1 alphaqjl]

finds the intervals of the first kind

ptable = sym([]);

qmin =0;

qmax =0;

if (qwanted == -1)

qmin = 0;

qmax = 4;

elseif (qwanted > 0)

qmin = qwanted;

qmax = qwanted;
end;

mykmin = 2;

if (kmax < 0)
    mykmin = abs(kmax);
    mykmax = abs(kmax);
elseif (kmax > 0)
    mykmin = 1;
    mykmax = kmax;
end;

for q=qmin:qmax
    for k=mykmin:mykmax
        for i =0:(10^(k-1))
            alphaqik = sym((10^(-k))*(10*i+1 - 2*q));
            betaqik = sym((10^(-k))*(10*i+9 - 2*q));
        for l =0:(k-1)
            for j = 0:(floor(10^(l-1)))
                if (l > 0)
                    alphaqjl = sym((10^(-l))*(10*j+1 - 2*q));
                    betaqjl = sym((10^(-l))*(10*j+9 - 2*q));
                else
                    alphaqjl = sym(0);
                endif
            endfor
        endfor
    endfor
endfor
betaqjl = sym(1);
end;

expr1 = strcmpi(maple('evalb', [char(alphaqik), '<', char(alphaqjl)]), 'true');
expr2 = strcmpi(maple('evalb', [char(alphaqjl), '<', char(betaqik)]), 'true');
if(expr1 && expr2)
    ptable = [ptable; [q i k alphaqik betaqik j l 1 alphaqjl]];
end;

expr1 = strcmpi(maple('evalb', [char(alphaqik), '<', char(betaqjl)]), 'true');
expr2 = strcmpi(maple('evalb', [char(betaqjl), '<', char(betaqik)]), 'true');
if(expr1 && expr2)
    ptable = [ptable; [q i k alphaqik betaqik j l 9 betaqjl]];
end;
B.1.3 train_gt

function [H2D,G1D] = train_gt(k,phihp,uhp, myiter, funin)

% this program takes an image and sees what image is reproduced after
% myiter iterations

% Setup the function f (an image) for processing

global fun;
global xi;
global yi;
fun = funin;
SIZEFUN =size(fun);
mx =1:SIZEFUN(2);
nx =1:SIZEFUN(1);

% mxl=SIZEFUN(1);

% nxl = SIZEFUN(1);

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

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

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

disp([size(xi) size(yi)]);

% Setup the function phi,u for processing

disp('initializing.........');

if (~isnumeric(uhp))
    disp('converting u')
    SIZEU = size(uhp);
    u = zeros(SIZEU);
    tic;
    disp([ ' time start=' num2str(0)]);
    for i =1:SIZEU(1)
        u(i,:)= double(uhp(i,:));
        if (mod(i,5000) == 0 )
            TOCTIC = toc;
            disp([ 'i =' num2str(i) ' time end=' num2str(TOCTIC)]);
            tic;
        end
    end
else
    disp('data already numeric')
end

disp('processing is complete')

disp('finished processing');
disp([' time start=' num2str(0)));
end;
end;
disp('done converting u');
else

disp('u is already type double');
end;

if (~isnumeric(phihp{1}.table))
disp('converting phi_q');
for q=0:4
phi{q+1}.table = double(phihp{q+1}.table);
disp(['done converting phi_' num2str(q)]);
end;
disp('done converting phi_q');
else

disp('phi_q is already type double');
phi = phihp;
end;
disp('done initializing....');
SIZEU = size(u);

global lambda;

lambda = 1/sqrt(2);

maxiter = myiter;

% preallocate memory
gr = zeros([SIZEU(1) maxiter]);
G = zeros(SIZEU(1),1);

% X=mx-1;
% Y=nx-1;
% [x,y] = meshgrid(X,Y);
% SIZEX = size(x);
% SIZEY = size(y);
% xv = reshape(x,SIZEX(1)*SIZEX(2),1); % make x a long vector
% yv = reshape(y,SIZEY(1)*SIZEY(2),1); % make y a long vector..
% has same dimensions as x
% Plot f(x,y)
% z = f(x,y);
%z = reshape(zv,SIZEX(1),SIZEX(2));

subplot(2,2,1);
surf(x,y,z);
reply = input('Pausing. Type 1 to loop without pausing: ');

% assign values to f from \( f(x,y) \) at centers of squares
alphaqik = (10^(-k))*(10*u(:,3)+1-2*u(:,2));
betaqik = (10^(-k))*(10*u(:,3)+9-2*u(:,2));
alphaqjk = (10^(-k))*(10*u(:,4)+1-2*u(:,2));
betaqjk = (10^(-k))*(10*u(:,4)+9-2*u(:,2));

% get rid of squares outside of \([0,1] \times [0,1]\)
temp = find((alphaqik > 1) | (alphaqjk > 1) | ...
\quad (betaqik < 0) | (betaqjk < 0) );

if (~isempty(temp))
    alphaqik(temp) = -10;
end;

temp = find(alphaqik ~= -10);
if (~isempty(temp))
    alphaqik = alphaqik(temp);
    alphaqjk = alphaqjk(temp);
    betaqik = betaqik(temp);
    betaqjk = betaqjk(temp);
end;

%truncates squares partially in [0,1] x [0,1]

temp = find(alphaqik < 0);

if (~isempty(temp))
    alphaqik(temp) = 0;
end;

temp = find(betaqik > 1);

if (~isempty(temp))
    betaqik(temp) = 1;
end;

temp = find(alphaqjk < 0);

if (~isempty(temp))
    alphaqjk(temp) = 0;
end;

temp = find(betaqjk > 1);

if (~isempty(temp))
    betaqjk(temp) = 1;
end;

m1 = (alphaqik + betaqik)/2;

m2 = (alphaqjk + betaqjk)/2;

clear alphaqik betaqik alphaqjk betaqjk;
%disp([size(m1) size(m2)]);

% calculate h(x,y) from f(x,y);

%iterate

for iter=1:maxiter

    i = iter;

    % disp(['iteration: ' num2str(iter)]);

    fxy = f(m1, m2) - h(m1, m2,u,gr,phi,1);

    gr(:,i) = (1/3)*fxy;

    if(reply ~= 1)

        % plot h(x,y), gr(u), G(u)

        % gr(u)

        subplot(2,2,3);

        plot(u(:,1),gr(:,i));

        %

        % G(u) = sum_{1}^{iter} gr(u)

        G = G + gr(:,i);

        subplot(2,2,4);

    end

end
plot(u(:,1),G);

% h(x,y)
zhv = h(xv,yv,u,gr,phi,1);
z = reshape(zv,SIZEX(1),SIZEX(2));
subplot(2,2,2);
surf(x,y,z);

% pause to view plot
reply = input('Pausing. Type 1 to loop without pausing: ');
end;

end;

%plot h(x,y), gr(u), g(u) for last time

gr(u)
subplot(2,2,3);
plot(u(:,1),gr(:,maxiter));

G(u) = sum{1}^(iter) gr(u)
subplot(2,2,4);
plot(u(:,1),G);
% $h(x,y)$

% subplot(2,2,2);
% surf(x,y,z);

SIZEXI = size(xi);

SIZEYI = size(yi);

zv = h(reshape(xi,SIZEXI(1)*SIZEXI(2),1),reshape(yi,...
    SIZEYI(1)*SIZEYI(2),1),u,gr,phi,1);

disp(size(zv));

z = reshape(zv,SIZEFUN(1),SIZEFUN(2));

H2D = z;

G1D = G;

% save('myz','z', 'fun');

%%%%% end main function app8_hp

% define f(x,y) here

function [fvalue] = f(x,y)

global fun;

global xi;

global yi;
fvalue = interp2(xi,yi,fun,x,y);

%define h(x,y) here

function[ hvalue ] = h( x,y,u,g,phi,tosum)

global lambda;

hsum = 0;

SIZEG = size(g);

if (tosum == 0) % sum only the latest g;r; for h_r(,y)

for q = 0:4
    uc = interp1( phi{q+1}.table(:,1), phi{q+1}.table(:,2), x ) + lambda * interp1( phi{q+1}.table(:,1), phi{q+1}.table(:,2), y );
    hsum = hsum + interp1( u(:,1), g(:, SIZEG(2)), uc );
end;
else % sum all the g;r; for h(x,y)

for i=1:SIZEG(2)
    for q = 0:4
        uc = interp1( phi{q+1}.table(:,1), phi{q+1}.table(:,2), x ) + lambda * interp1( phi{q+1}.table(:,1), phi{q+1}.table(:,2), y );
        hsum = hsum + interp1( u(:,1), g(:, i), uc );
    end;
end;
end;
end;

hvalue = hsum;

B.1.4  recon_ft

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

% global xi;
% global yi;

SIZEFUN = imageSize;

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

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

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

global lambda;

lambda = 1/sqrt(2);
SIZEXI = size(xi);

SIZEYI = size(yi);

zv = h(reshape(xi,SIZEXI(1)*SIZEXI(2),1),reshape(yi,...

SIZEYI(1)*SIZEYI(2),1),u,g,phi,0);

disp(size(zv));

z = reshape(zv,SIZEFUN(1),SIZEFUN(2));

f2D = z;

% save('myz','z', 'fun');

% end main function recon_f

% define h(x,y) here

function [hvalue] = h(x,y,u,g,phi,tosum)

global lambda;

hsum = 0;

SIZEG = size(g);

if (tosum == 0) % sum only the latest g_r; for h_r(,y)

for q = 0:4

uc = interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),x) + ...

lambda * interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),y);
hsum = hsum + interp1(u(:,1),g(:,SIZEG(2)), uc);

end;

else % sum all the g_r ; for h(x,y)
for i=1:SIZEG(2)
for q =0:4
uc = interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),x) + 
lambda * interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),y);
hsum = hsum + interp1(u(:,1),g(:,i), uc);
end;
end;
end;

hvalue = hsum;

B.1.5 eval_gt

function [fx] = eval_gt(x, phi, u,g)

% 

global lambda;

lambda = 1/sqrt(2);
zv = h(x(:,1),x(:,2),u,g,phi,0);
fx =zv;

%define h(x,y) here
function[hvalue] = h(x,y,u,g,phi,tosum)

global lambda;
hsum =0;
SIZEG = size(g);

if (tosum == 0) % sum only the latest g.r; for h_r,(y)
    for q=0:4
        uc = interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),x) + ...
            lambda * interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),y);
        hsum = hsum + interp1(u(:,1),g(:,SIZEG(2)), uc);
    end;
else % sum all the g.r ; for h(x,y)
    for i=1:SIZEG(2)
for q = 0:4
    uc = interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),x) + lambda * interp1(phi{q+1}.table(:,1),phi{q+1}.table(:,2),y);
    hsum = hsum + interp1(u(:,1),g(:,i), uc);
end;
end;
end;
hvalue = hsum;
LIST OF REFERENCES

[1]


129


[27] S.C. Zhu and X. Liu. Learning in gibbsian fields: How accurate and how fast can it be?