ABSTRACT

Title of dissertation: EFFICIENT ALGORITHMS FOR CLUSTERING AND INTERPOLATION OF LARGE SPATIAL DATA SETS

Nargess Memarsadeghi
Doctor of Philosophy, 2007

Dissertation directed by: Professor David M. Mount
Department of Computer Science

Categorizing, analyzing, and integrating large spatial data sets are of great importance in various areas such as image processing, pattern recognition, remote sensing, and life sciences. For example, NASA alone is faced with huge data sets gathered from around the globe on a daily basis to help scientists better understand our planet. Many approaches for accurately clustering, interpolating, and integrating these data sets are very computationally expensive. The focus of my PhD thesis is on the development of efficient implementations of data clustering and interpolation methods for large spatial data sets, and the application of these methods to geostatistics and remote sensing. In particular, I have developed fast implementations of ISODATA clustering and kriging interpolation algorithms. These implementations derive their efficiency through the use of both exact and approximate computational techniques from computational geometry and scientific computing.

My work on the ISODATA clustering algorithm employs the kd-tree data structure and the filtering algorithm to speed up distance and nearest neighbor calcula-

tions. In the case of kriging interpolation, I applied techniques from scientific computing including iterative methods, tapering, fast multipole methods, and nearest neighbor searching techniques. I also present an application of kriging interpolation method to the problem of data fusion of remotely sensed data.

EFFICIENT ALGORITHMS FOR CLUSTERING AND INTERPOLATION OF LARGE SPATIAL DATA SETS

by

Nargess Memarsadeghi

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

Advisory Committee:

Professor David M. Mount, Chair
Associate Professor Ramani Duraiswami
Assistant Professor Lise Getoor
Doctor Jacqueline Le Moigne
Professor Dianne P. O'Leary
Professor John Townshend

Dedication

To my mother, Sedigheh Rastegaralam

and

To my father, Alinaghi Memarsadeghi.

Acknowledgments

My graduate study years at University of Maryland, College Park, have been one of the best periods of my life. I found many good friends with whom unforgettable memories were created. I had the opportunity of knowing and working with world-class researchers. I learned many lessons and survived some difficult times. The completion of this thesis marks the commencement of a new period of my life. This thesis would not have been possible without help and good intentions of many people whom I would like to thank.

First and foremost, I would like to thank Professor David M. Mount for being an extremely kind, patient, wise, and knowledgeable advisor. He demonstrates being an excellent researcher and educator on a daily basis. I am deeply honored and grateful for being his student and for having my share of his valuable time. He helped me through various stages of my research, from providing insights for design and implementation of different algorithms to editing my writings. I enjoyed our meetings, in most of which Dr. Mount would enlighten me by associating physical and geometric meanings to abstract ideas. I will always cherish these good memories and lessons I have learned from him. I strive to be one of his good students.

I have also been fortunate to have Dr. Jacqueline Le Moigne as my supervisor and advisor at work. She introduced me to the fields of remote sensing and image processing, and patiently taught me various concepts in these areas. She gracefully handles various crises, be it dealing with family emergencies or finding projects for her employees after an extreme budget cut. I thank her for being such a great role model, who makes many tasks seem easy and manageable.

I would like to thank other members of my examining committee, Dr. Ramani Duraiswuami, Dr. Lise Getoor, Dr. Dianne P. O'Leary, and Dr. John Townshend for their time during both my preliminary oral exam and my final defense exam as well as for their feedback which greatly improved this thesis. In particular, Professor O'Leary was always available and responsive to my questions, even during her sabbatical leave.

Special thanks to my collaborators Ramani Duraiswami, Jacqueline Le Moigne, Jeffrey T. Morisette, David M. Mount, Nathan S. Netanyahu, Dianne P. O'Leary, and Vikas C. Raykar for working with me. This thesis would not have been possible without their insights, contributions, and support.

I would like to thank the Computer Science Department of the University of Maryland, College Park, for a great educational environment, for gathering bright students from all around the world, for wonderful faculty and staff, and for supporting my education during the first year of my graduate studies.

I started graduate school while I was already an employee of NASA GSFC. I am grateful to NASA GSFC's Advanced Architectures and Automation Branch (code 588) and NASA GSFC's Cooperative Education Office for supporting my research and education for the past four years. Maintaining my employment and full time graduate student status as well as having support for my research would not have been possible without the continuous support of various supervisors and managers at NASA GSFC's code 580. In particular, I am thankful to Joseph Hennessy, Jacqueline Le Moigne, Julie Loftis (formerly Breed), and Barbara Medina for their encouragements and for allowing me work on the topics presented in this thesis.

Special thanks to my first supervisor at NASA, Ms. Julie Loftis, for hiring me, for her constant support ever since, and for her innovative ideas that give rise to development of new technologies in our division. I also thank my colleagues at NASA GSFC, code 588, for their kindness, friendship, and encouragements.

I am grateful for many caring friends who constantly contributed to my well-being and happiness during my graduate study years. Two of them were with me everyday at work for the past four years: Peyush Jain and Lisa Kane. A big Thank You to my dear friends Sima Asgari, Maliheh Poorfarhani, Shabnam Tafreshi, Indrajit Bhattacharya, Houman Alborzi, Jennifer Geiger, and many others for all the good times and for providing me a safety net I could always rely on. I also thank my uncle Heidarali Memarsadeghi for always being there for me.

I would like to thank my mother Sedigheh Rastegaralam, my father Alinaghi Memarsadeghi, and my bothers Mohammad and Hossein Memarsadeghi for their unconditional love and support throughout my life. Regardless of the situation, getting a good education was always prioritized in order of matters by my parents. For this reason, among countless others, I thank them. This thesis is dedicated to them.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

NASA	National Aeronautics and Space Administration
GSFC	Goddard Space Flight Center
UMD	University of Maryland
ALI	Advanced Land Imager
CWL	Central WaveLength
DEM	Digital Elevation Model
DTC	Decision Tree Classifier
EO-1	Earth Observing-1
FGT	Fast Gauss Transform
FMM	Fast Multipole Methods
GIS	Geographic Information System
GP	Gaussian Process
GT	Gauss Transform
HMM	Hidden Markov Model
HPF	High Pass Filter
IFGT	Improved Fast Gauss Transform
IHS	Intensity Hue Saturation
ISODATA	Iterative Self-Organizing Data Analysis Techniques
ISFS	Invasive Species Forecasting System
LS	Least Squares
MLS	Moving Least Squares
MS	Multispectral
NDVI	Normalized Difference Vegetation Index
PAN	Panchromatic
PCA	Principal Component Analysis
RGB	Red Green Blue
SAR	Synthetic Aperture Radar
SPOT	System d'Observation de la Terr
SVM	Support Vector Machines
TM	(Landsat) Thematic Mapper
ETM	(Landsat) Enhanced Thematic Mapper
WLS	Weighted Least Squares
YIQ	Luminance Chrominance color space

Chapter 1

Introduction

Scientists in various disciplines are faced with large amounts of data that need to be studied and analyzed. For example, NASA continuously collects data regarding the earth, the solar system, and the universe beyond. Scientists analyze these data to study and understand the earth, sun, oceans, and issues such as climate change. Categorizing, analyzing, and integrating these data are vital to advancements in various areas of science and to our understanding of our planet and the universe. Clustering, interpolation, and multi-sensor data fusion are examples of important computational tools for developing a better understanding of the mass of data being gathered from numerous sensors. These tools are widely used in *remote sensing* and *geostatistics*. Remote sensing in broad terms is defined as collection of information about an object without being in physical contact with it [145]. In the case of earth and space observation, this is done through the use of of aircraft, satellites, and so on. The term "geostatistics" was first coined by G. Matheron (1962), who defined it as the application of the formalism of random functions to the reconnaissance and estimation of natural phenomena [85].

Given a large data set, *clustering* is the process of grouping together data objects with similar properties [81]. For example, clustering is used in geosciences and remote sensing applications to obtain vegetation maps of a given region. *In-*

terpolation of data values is of interest where gathering data at specific locations is either impossible or impractical. This arises, for example, in geological and mining applications [80, 85]. Scientists often integrate data covering the same geographic region from various sensors for their studies. This is called *data fusion* [67, 129]. In the context of data fusion, interpolation techniques can compensate for missing or noisy data. Data fusion is also an important step in filling the gaps where data are gathered from multiple sources having different sampling rates. The application under study can be earth science-related, e.g. preparing a vegetation map, performing ecological modeling, weather forecasting, or wild fire prediction [67,129]. In all cases, scientists would like to take advantage of complementary information provided by multiple sources such as various sensors or in-situ measurements. These multiple data sets representing different spatial, spectral, and temporal resolutions, or gathered from multiple viewpoints, can then be used to supplement missing or spurious data, and to generate new improved data sets. Both clustering and interpolation methods can be used at various stages of the data fusion process.

Thus, clustering and interpolation are significant tools in the areas of remote sensing, geostatistics, and earth sciences. However, they can be quite computationally expensive. Many formulations of clustering as an optimization problem are known to be NP-hard [78]. This issue is particularly relevant when dealing with large data sets. For example, most center-based clustering algorithms, such as k-means [81], require the repeated calculation of the distances from each point to its closest cluster center. This can be very time consuming when very large data sets are involved. Depending on the method used, interpolation can also be computationally

expensive.

In this dissertation we will consider a popular clustering method called ISODATA. The ISODATA clustering algorithm is very similar to the k-means algorithm in nature [16, 17, 81]. Unlike k-means, where the user specifies a fixed number of initial clusters k, ISODATA has various heuristics through which it adaptively determines the final number of clusters. As with the k-means algorithm, ISODATA is computationally very expensive for large data sets.

The interpolation method that we will focus on throughout is called kriging. Kriging is an important interpolation method that is widely used in geostatistics applications [60, 80]. It is often referred to as the "gold standard" in interpolation because it possesses a number of desirable statistical properties [111]. In particular, it is a best linear unbiased estimator [60, 80]. There are a number of variants and generalizations to kriging. Unfortunately, this general class of interpolation techniques can be computationally very expensive when large data sets are involved, since computing the value of the interpolant exactly for a single point might generally involve solving a dense linear system of size $O(n^2)$, where n is the number of data points.

Clearly, the availability of efficient implementations of clustering and interpolation algorithms for large data sets is essential for helping scientists analyze and integrate their data in a timely and efficient manner. The focus of my thesis is on the efficient implementations of computational tools for clustering and interpolation, particularly for large spatial data sets. We will also consider their application to the task of data fusion of remotely sensed data. As mentioned, the clustering algorithm

we will focus on is ISODATA, and the interpolation method is kriging. These implementations derive their efficiencies through the application of methods from both computational geometry and scientific computing.

First, we will present a more efficient approach to ISODATA clustering, which achieves better running times by storing the points in a common spatial data structure, the kd-tree [20], and through a modification of the way in which the algorithm estimates the dispersion of each cluster. We also present an approximate version of the algorithm, which allows the user to further improve the running time at the expense of lower fidelity in computing the nearest cluster center to each point. We provide both theoretical and empirical justification that our modified approach produces clusterings that are very similar to those produced by the standard ISODATA approach. We also provide empirical studies on both synthetic data and remotely sensed Landsat and MODIS images that show that our approach has significantly lower running times.

We also present implementations of efficient kriging interpolation algorithms, which have been specially designed for handling large data sets efficiently. Our algorithms are based on the use of iterative methods for solving the linear systems involved combined with a number of additional approximation techniques. These include tapering [55], fast multipole methods [61], and nearest neighbor searching methods [114]. We will present extensive experimental evidence that these approaches result in significant memory and running time savings without significantly compromising the quality of the results. Finally, we present a study that shows how kriging can be applied to the problem of data fusion involving remotely sensed data.

The remainder of this dissertation is organized as follows. First, a survey of clustering techniques is presented in Chapter 2. One of these techniques is the ISODATA clustering algorithm. In Chapter 3, our work on efficient implementation of a variant of the ISODATA clustering algorithm is presented. In Chapter 4, we present a survey of approaches to scattered data interpolation, with a particular focus on interpolation techniques traditionally used in geostatistics applications. Following this is a presentation of our work on efficient implementation of kriging interpolation via two different approaches. The first approach, which is based on tapering and iterative methods, is described in Chapter 5. Chapter 6 contains a presentation of our work on efficient implementation of the kriging algorithm by using iterative methods with ideas based on the fast multipole methods. We then proceed by introducing data fusion, its applications and objectives in Chapter 7. In Chapter 8, we present results of our prior work on image fusion of remotely sensed data and present an application of the kriging interpolation to this problem. Chapter 9 concludes this dissertation and presents some open research problems related to the work presented in this thesis. Details of the kriging interpolation methods are presented in Appendix A. Specifications of the satellite data that we have used in Chapter 8 are available in Appendix B.

Chapter 2

Survey of Clustering Algorithms

Clustering can be loosely defined as the unsupervised classification of patterns into groups [82]. Clustering has various applications in areas such as pattern recognition, land cover and land use, remote sensing, machine learning, data mining, data compression, image processing, document analysis, and life sciences [9, 21, 44, 82, 84, 112, 143, 156, 165, 166]. Clustering algorithms are generally used in a *learning process* where one wants to learn properties of a given data. For clustering, this learning process is *unsupervised*. That is, the user does not have any information about properties of data and their similarities. Thus, clustering can be defined as the grouping of points with similar properties into the same class, without knowing the class's properties or its label.

Different clustering algorithms use different measures of similarity. In most algorithms, points are considered to be similar if they are close to each other according to some notion of spatial distance. Definition of closeness depends on the distance function being considered. In this survey, we will present a number of clustering algorithms, focusing on those that are most closely related to this dissertation.

There are two general families of clustering algorithms [21, 82, 161]: *partition-based* and *hierarchical* clustering algorithms. The focus of this survey is on partition-based clustering methods. I will present an overview of past work done for these

methods as well as explaining my prior work related to this area. See [32,82,128,161] to learn more about work done in the area of hierarchial clustering algorithms. There are variants and subcategories of the above mentioned families of clustering algorithms. A few of the issues which can affect taxonomy of clustering algorithms are as follows [82]:

Agglomerative versus Divisive: An agglomerative approach considers each point to be a member of a singleton cluster and starts to merge points to form a final clustering. On the other hand a divisive approach considers all points to be in one big cluster and then splits them until some criteria are met.

Hard versus Fuzzy: Hard clustering techniques assign each point to a single cluster. Fuzzy clustering algorithms, on the other hand, assign a degree of membership in each cluster j, f_{ij}, to each data point x_i. A fuzzy clustering can be mapped to a hard one by assigning each point x_i to the cluster j for which f_{ij} is maximum [82].

Deterministic versus Stochastic: This implementation issue mostly shows itself for partition-based algorithms where an optimization problem can be solved either through traditional techniques or through a random search of the state space of all possible clusterings.

These implementation issues (and a few others) result in various categories of clustering algorithms, a few of which are presented below. See [21, 82, 161] for further information. Examples of such categories include the following:

- Probabilistic Clustering (e.g. fuzzy clustering, see [21, 94, 112].)
- Model-based clustering (e.g. mixture models, Gaussian mixture models, expectation maximization (EM), see [11, 31, 36].)
- Graph Theoretic Methods (e.g. spectral clustering, see [42, 64, 86, 154].)

2.1 Partition-based Methods

Given a set P of n points in $\mathbb{R}^d$ and an integer $k > 0$, a typical clustering problem asks for partitioning P into k subsets called *clusters* so that a certain objective function (usually a function of distance) is minimized. These clustering algorithms are also referred to as *center-based clustering algorithms*. That is, clusters are represented by a set of k centers, and the goal of the clustering algorithm is to assign each point to a cluster such that its objective function is minimized. The objective function of center-based algorithms is to minimize a function of distance from every point in P to its closest cluster center.

There are two types of objective functions: *centered* and *summed* [3]. That is, given a function μ which measures the *extent* of a cluster, the objective function being minimized in a centered clustering algorithm is of the form $\max_{1 \leq i \leq k} \mu(P_i)$ and in a summed clustering algorithm is of the form $\sum_{i=1}^{k} \mu(P_i)$. For example, *k-center* is a centered clustering algorithms while *k-median* and *k-means* are summed clustering algorithms.

Currently, there are no efficient exact solutions to any of the above mentioned problems for general k and some formulations are known to be NP-hard [56]. Thus,

researchers have been working on efficient approximate implementations for these cases. We will present a number of partition/center-based algorithms and the work that has been completed in this area. Note that depending on which distance measure is being considered in the objective function of these clustering algorithms we are dealing with different optimization problems. Throughout this survey Euclidean distance will be assumed.

2.1.1 K-center

The objective of the k-center problem is to minimize the maximum distance of points within a cluster from their cluster center (*min-max radius clustering*) [59]. In other words, given a set of data points, we would like to cover them with k-balls where radius of the largest ball is minimized. This problem has two versions: *continuous* and *discrete*. In continuous k-center, centers can be located anywhere in the underlying space while in the discrete case centers must be selected from input data points. Unless otherwise specified, we will assume the discrete case.

A naive exact solution to the discrete k-center problem would require $\Omega(n^k)$ time at least to generate all k-element subsets of the point set. This requires exponential time, if k is a function of n. Hochbaum and Shmoys introduced a graph algorithm which provides a factor-2 approximation for the k-center and runs in $O(n^2 \log n)$ time [76, 77]. Independently, and around the same time, Gonzalez presented another factor-2 algorithm for the k-center problem [59]. He used a simple greedy approach that runs in $O(nk)$ time. This algorithm repeatedly selects the

farthest point from current set of centers as the next center to be added. Later, Feder and Greene [47] presented a different implementation of this greedy approach to achieve a factor-2 approximation to the k-center problem that runs in $\Theta(n \log k)$ time. They also showed that computing a c-approximation is NP hard for $c \leq 1.822$ under the Euclidean norm and for $c \leq 2$ for L_∞ metric.

Agarwal and Procopiuc in [4] presented an $(1+\epsilon)$-approximation algorithm for the Euclidean k-center problem with a running time of $O(n \log k) + (\frac{k}{\epsilon})^{O(k^{1-\frac{1}{d}})}$. Har-Peled introduced the first approximate k-center algorithm for a set of moving data points [68, 69]. He used a static clustering approach for the moving points. That is, at any given time, he selects a small subset of data points such that performing exact clustering on them yields a good approximation to the optimal k-center clustering at any time. Such a set is called a *coreset*. He also improved the Feder and Greene algorithm to run in $O(n)$ time.

2.1.2 K-median and k-means

The objective of the k-median problem is to minimize the sum of distances to the nearest center [78]. For $k = 1$ this problem is known as the Fermat-Weber problem [3]. The goal of the k-means clustering is to minimize the sum of squared distances [57, 81, 100, 102]. We define the *neighborhood* of a center point to be the set of data points for which this center is the closest. It is easy to prove that any locally minimal solution to the k-means problem lies at the centroid of its neighborhood [43, 45]. It is known that k-means has a local optimum equal to the

centroid of the points in each cluster, and that the k-means algorithm also called Lloyd's algorithm converges to this local centroid [24, 72, 104, 130, 147].

There is a $(1+\epsilon)$-approximation for the k-median problem which runs in $O(2^{\frac{1}{\epsilon^d} n \log n \log k})$ time assuming that dimension d is fixed [93]. This work, similar to [4] mentioned for the k-center problem, is based on applying dynamic programming to an adaptive hierarchial decomposition of space.

Jain and Vazirani gave a 6-approximation algorithm for the k-median problem with $O(n^2(\log n)(L + \log n))$ running time, where L denotes the number of bits required to store the longest edge or largest connection cost [83]. This approximation was later improved by Charikar and Guha to a factor-4 approximation algorithm that runs in $O(n^3)$ time [25].

This approximation factor was further improved by Arya *et al.* to a 3-approximation algorithm through a local search approach [14,15]. They show that when performing a local search where up to p centers can be exchanged (swapped) in each step, an improved approximation factor of $3 + \frac{2}{p}$ with a running time of $O(n^p)$ can be obtained. That is, the algorithm starts with k initial centers and at each step, it proceeds by removing p centers and adding p others until the overall objective function cannot be improved.

For the k-means problem, Matoušek made a breakthrough by introducing the first $(1+\epsilon)$-approximation algorithm for this problem [106]. Matoušek showed that his algorithm has a running time of $O(n(\log^k n)\epsilon^{-2k^2 d})$ for fixed d and k. Unfortunately, this approach is impractical unless k is a small constant. Kanungo *et al.* gave a practical local search approximation for the k-means algorithm yielding

a $(9+\epsilon)$-approximation algorithm [88]. Later, this implementation was improved by Frahling and Sohler [51]. The first linear time $(1+\epsilon)$-approximation algorithm (when treating k and ϵ as constants) for the k-means problem was introduced by A. Kumar *et al.* [96]. Their algorithm uses random sampling and is very simple.

Har-Peled and Mazumdar showed that given any data set P, there exists a nonempty subset $Q \subseteq P$ of size $O(k\epsilon^{-d} \log n)$, such that one can compute the k-median/means clustering on Q instead of P to obtain an $(1+\epsilon)$-approximation to the optimal solution to the problem on P [71]. They called such a set Q, a (k, ϵ) *coreset* of P. As a result, they presented a $(1+\epsilon)$-approximation algorithm for the k-means and k-median problem with *linear* running time for fixed k and ϵ. Har-Peled and Kushal showed how to construct a smaller coreset of size $O\left(\frac{k^2}{\epsilon^d}\right)$ and $O\left(\frac{k^3}{\epsilon^{(d+1)}}\right)$ for k-median and k-means problems respectively, which are independent in size of n, number of data points [70].

2.1.3 ISODATA

The ISODATA algorithm was first introduced in [16–18], and it is also described in [59, 81, 156]. ISODATA is a clustering algorithm similar to k-means but it changes the number of clusters by merging and splitting clusters. It has various heuristics to determine when to merge a pair of clusters or when to split a cluster. This algorithm is widely used in remote sensing where the user is dealing with large data sets and would like to perform unsupervised clustering without having to know how many distinct classes exist within the data [58,84,143]. Fast implementations of this

algorithm are presented in [109, 110]. Detailed description of the original algorithm is mentioned in [59, 81, 156]. Chapter 3 also provides the description of the ISODATA algorithm as well as my work on its efficient implementations.

Chapter 3

A Fast Implementation of the ISODATA Clustering Algorithm

Unsupervised clustering is a fundamental tool in image processing for geoscience and remote sensing applications. For example, unsupervised clustering is often used to obtain vegetation maps of an area of interest. This approach is useful when reliable training data are either scarce or expensive, and when relatively little *a priori* information about the data is available. Unsupervised clustering methods play a significant role in the pursuit of unsupervised classification [143].

The problem of clustering points in multidimensional space can be posed formally as one of a number of well-known optimization problems, such as the Euclidean *k-median problem* [78], in which the objective is to minimize the sum of distances to the nearest center, the Euclidean *k-center problem* [59], in which the objective is to minimize the maximum distance, and the *k-means problem*, in which the objective is to minimize the sum of squared distances [57, 81, 100, 102]. Efficient solutions are known to exist only in special cases such as the planar 2-center problem [5, 148]. There are no efficient exact solutions known to any of these problems for general k, and some formulations are known to be NP-hard [56]. Efficient approximation algorithms have been developed in some cases. These include constant factor approximations for the k-center problem [47, 59], the k-median problem [15, 25, 83], and the k-means problem [88]. There are also ϵ-approximation algorithms for the

k-median [12, 93] and k-means [96, 106] problems, including improvements based on coresets [70, 71]. Work on the k-center algorithm for moving data points, as well as a linear time implementation of a 2-factor approximation of the k-center problem have also been introduced [68, 69].

In spite of progress on theoretical bounds, ϵ-approximation algorithms for these clustering problems are still not suitable for practical implementation in multidimensional spaces, when k is not a small constant. This is due to very fast growing dependencies in the asymptotic running times on the dimension and/or on k. In practice, it is common to use heuristic approaches, which seek to find a reasonably good clustering, but do not provide guarantees on the quality of the results. This includes randomized approaches, such as CLARA [89] and CLARANS [119], and methods based on neural networks [92]. One of the most popular and widely used clustering heuristics in remote sensing is ISODATA [16, 81, 84, 156]. A set of n data points in d-dimensional space is given along with an integer k indicating the initial number of clusters and a number of additional parameters. The general goal is to compute a set of cluster centers in d-space. Although there is no specific optimization criterion, the algorithm is similar in spirit to the well-known k-means clustering method [81], in which the objective is to minimize the average squared distance of each point to its nearest center, called the *average distortion*. One significant advantage of ISODATA over k-means is that the user need only provide an initial estimate of the number of clusters, and based on various heuristics the algorithm may alter the number of clusters by either deleting small clusters, merging nearby clusters, or splitting large diffuse clusters. The algorithm will be described in Section 3.1.

As currently implemented, ISODATA can run very slowly, particularly on large data sets. Given its wide use in remote sensing, its efficient computation is an important goal. Our objective here is not to provide a new or better clustering algorithm, but rather show how computational geometry methods can be applied to produce a faster implementation of ISODATA clustering. There are a number of minor variations of ISODATA that appear in the literature. These variations involve issues such as termination conditions, but they are equivalent in terms of their overall structure. We focus on a widely used version, called ISOCLUS [124], which will be presented in Section 3.1.

The running times of ISODATA and ISOCLUS are dominated by the time needed to compute the nearest among the k cluster centers to each of the n points. This can be reduced to the problem of answering n nearest-neighbor queries over a set of size k, which naively would involve $O(kn)$ time. To improve the running time, an obvious alternative would be to store the k centers in a spatial index such as a kd-tree [20]. However, this is not the best approach, because k is typically much smaller than n, and the center points are constantly changing, requiring the tree to be constantly updated. Kanungo *et al.* [87] proposed a more efficient and practical approach by storing the points, rather than the cluster centers, in a kd-tree. The tree is then used to solve the *reverse nearest neighbor problem*, that is, for each center we compute the set of points for which this center is the closest. This method is called the *filtering algorithm*.

We show how to modify this approach for ISOCLUS. The modifications are not trivial. First, in order to perform the sort of aggregate processing that the

filtering algorithm employs, it was necessary to modify the way in which the ISOCLUS algorithm computes the degree of dispersion within each cluster. In Section 3.4 and Section 3.5 we present, respectively, empirical and theoretical justification that this modification does not significantly alter the nature of the clusters that the algorithm produces. In order to further improve execution times, we have also introduced an approximate version of the filtering algorithm. A user-supplied approximation error bound $\epsilon > 0$ is provided to the algorithm, and each point is associated with a center whose distance from the point is not farther than $(1 + \epsilon)$ times the distance to its true nearest neighbor. This result may be of independent interest because it can be applied to k-means clustering as well. It is presented in Section 3.2.5.

The running time of the filtering algorithm is a subtle function of the structure of the clusters and centers, and so rather than presenting a worst-case asymptotic analysis, we present an empirical analysis of its efficiency based on both synthetically generated data sets, and actual data sets from a common application in remote sensing and geostatistics. These results are presented in Section 3.4. As the experiments show, depending on the various input parameters (that is, dimension, data size, number of centers, etc.), the algorithm presented runs faster than a straightforward implementation of ISOCLUS by factors ranging from 1.3 to over 50. In particular, the improvements are very good for typical applications in geostatistics, where the data size n and the number of centers k are large, and the dimension d is relatively small. Thus, we feel that this algorithm can play an important role in the analysis of geostatistical data analysis and other applications of data clustering.

The remainder of this chapter is organized as follows. In Section 3.1 we de-

scribe a variant of ISODATA, called ISOCLUS, whose modification is the focus of this chapter. In Section 3.2 we provide background, concerning basic tools such as the kd-tree data structure and the filtering algorithm, that will be needed in our efficient implementation of ISOCLUS. We present, in Section 3.3, our improved variants of the ISOCLUS algorithm and, in Section 3.4, the experimental results for these variants. In Section 3.5 we provide a theoretical justification of our cluster dispersion measure, which formed the basis of our efficient implementation. Finally, Section 3.6 contains concluding remarks.

3.1 The ISOCLUS Algorithm

We begin by presenting the particular variant of ISODATA, called ISOCLUS, [124] whose modification will be presented later. Although our description is not exhaustive, it contains enough information to understand our various modifications. The algorithm tries to find the best cluster centers through an iterative approach. It also uses a number of different heuristics to determine whether to merge or split clusters.

At a high level, the following tasks are performed in each iteration of the algorithm: Points are assigned to their closest cluster centers, cluster centers are updated to be the centroid of their associated points, clusters with very few points are deleted, large clusters satisfying some conditions are split, and small clusters satisfying other conditions are merged. The algorithm continues until the number of iterations exceeds a user-supplied value.

Let us present the algorithm in more detail. There are a number of user-supplied parameters. These include the following. (In parentheses we give the variable name of the parameter used in [124].)

k_{init}: initial number of clusters (NUMCLUS)

$n_{\min}$: minimum number of points that can form a cluster (SAMPRM)

$I_{\max}$: maximum number of iterations (MAXITER)

$\sigma_{\max}$: maximum standard deviation of points from their cluster center along each axis (STDV)

$L_{\min}$: minimum required distance between two cluster centers (LUMP)

$P_{\max}$: maximum number of cluster pairs that can be merged per iteration (MAXPAIR)

Here is an overview of the algorithm. (See [124] for details.) Let $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ denote the set of points to be clustered. Each point $\mathbf{x}_j = (x_{j1}, \ldots, x_{jd})$ is treated as a vector in real d-dimensional space, $\mathbb{R}^d$. Let n denote the number of points. If the original set is too large, all of the iterations of the algorithm, except the last, can be performed on a random subset of S of an appropriate size. Throughout, let $\|\mathbf{x}\|$ denote the Euclidean length of the vector $\mathbf{x}$.

(1) Letting $k = k_{\text{init}}$, randomly sample k cluster initial centers $Z = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_k\}$ from S.

(2) Assign each point to its closest cluster center. For $1 \le i \le k$, let $S_i \subseteq S$ be the subset of points that are closer to $\mathbf{z}_i$ than to any other cluster center of Z. That is, for any $\mathbf{x} \in S$,

$$\mathbf{x} \in S_j \quad \text{if} \quad \|\mathbf{x} - \mathbf{z}_j\| < \|\mathbf{x} - \mathbf{z}_i\|, \quad \forall i \neq j.$$

(Ties for the closest center are broken arbitrarily.) Let n_j denote the number of points of S_j.

(3) Remove cluster centers with fewer than $n_{\min}$ points. (The associated points of S are not deleted, but are ignored for the remainder of the iteration.) Adjust the value of k and relabel the remaining clusters $S_1 \ldots, S_k$ accordingly.

(4) Move each cluster center to the centroid of the associated set of points. That is,

$$\mathbf{z}_j \leftarrow \frac{1}{n_j} \sum_{\mathbf{x} \in S_j} \mathbf{x}, \qquad \text{for } 1 \le j \le k.$$

If any clusters were deleted in Step 3, then the algorithm goes back to Step 2.

(5) Let Δ_j be the average distance of points of S_j to the associated cluster center $\mathbf{z}_j$, and let Δ be the overall average of these distances.

$$\Delta_j \leftarrow \frac{1}{n_j} \sum_{\mathbf{x} \in S_j} \|\mathbf{x} - \mathbf{z}_j\|, \quad \text{for } 1 \le j \le k. \qquad\qquad \Delta \leftarrow \frac{1}{n} \sum_{j=1}^{k} n_j \Delta_j.$$

(6) If this is the last iteration, then set $L_{\min} = 0$ and go to Step 9. Also, if $2k > k_{\text{init}}$ and it is either an even numbered iteration or $k \ge 2k_{\text{init}}$, then go to Step 9.

(7) For each cluster S_j, compute a vector $\mathbf{v}_j = (v_1, \ldots, v_d)$ whose ith coordinate is the standard deviation of the ith coordinates of the vectors directed from $\mathbf{z}_j$ to every point of S_j. That is,

$$v_{ji} \leftarrow \left(\frac{1}{n_j} \sum_{\mathbf{x} \in S_j} (x_i - z_{ji})^2 \right)^{1/2} \qquad \text{for } 1 \le j \le k \text{ and } 1 \le i \le d.$$

Let $v_{j,\max}$ denote the largest coordinate of $\mathbf{v}_j$.

(8) For each cluster S_j, if $v_{j,\max} > \sigma_{\max}$ and either

$$((\Delta_j > \Delta) \text{ and } (n_j > 2(n_{\min} + 1))) \text{ or } k \leq \frac{k_{\text{init}}}{2},$$

then increment k and split S_j into two clusters by replacing its center with two cluster centers centered around $\mathbf{z}_j$ and separated by an amount and direction that depends on $v_{j,\max}$ [124]. If any clusters are split in this step, then go to Step 2.

(9) Compute the pairwise *intercluster distances* between all distinct pairs of cluster centers

$$d_{ij} \leftarrow \|\mathbf{z}_i - \mathbf{z}_j\|, \qquad \text{for } 1 \leq i < j \leq k.$$

(10) Sort the intercluster distances of Step 9 in increasing order, and select a subset of at most $P_{\max}$ of the closest such pairs of clusters, such that each pair has an intercluster distance of at most $L_{\min}$. For each such pair (i, j), if neither S_i nor S_j has been involved in a merger in this iteration, replace the two clusters S_i and S_j with a merged cluster $S_i \cup S_j$, whose associated cluster center is their weighted average

$$\mathbf{z}_{ij} \leftarrow \frac{1}{n_i + n_j}(n_i \mathbf{z}_i + n_j \mathbf{z}_j).$$

Relabel the remaining clusters and decrease k accordingly.

(11) If the number of iterations is less than $I_{\max}$, then go to Step 2.

If the algorithm is implemented in the most straightforward manner, and if it is assumed that the number of clusters, k, is much smaller than the total number of points, n, then the most time-consuming stage of the algorithm is Step 2. Computing

naively the distances from each of the n points of S to each of the k centers for a total of $O(kn)$ time (assuming a fixed dimension d).

Our approach for improving the algorithm's running time is to speed up Step 2 through the use of an appropriate spatial data structure. Note that the algorithm does not need to explicitly compute the closest center to each point. What is needed is the centroid of the points that are closest to each center. Our approach is to compute this quantity directly. Before describing how to do this, we provide some background on a related clustering algorithm, called *Lloyd's algorithm*, and its fast implementation by a method called the *filtering algorithm*.

3.2 The Filtering Algorithm

At its heart, the ISOCLUS algorithm is based on an enhancement of a simple and widely used heuristic for k-means clustering, often called *Lloyd's algorithm* or the *k-means algorithm* [49, 100, 102]. It iteratively repeats the following two steps until convergence. First, for each cluster center, it computes the set of points for which this center is the closest. Next, it moves each center to the centroid of its associated set. It can be shown that with each step the average distortion decreases and that the algorithm converges to a local minimum [147]. See Refs. [[24, 72, 104, 130]] for further discussion on the statistical properties and convergence conditions of Lloyd's algorithm and other related procedures. The ISOCLUS algorithm combines Lloyd's algorithm with additional mechanisms for eliminating very small clusters (Step 3), splitting large clusters (Steps 7–8), and merging nearby clusters (Steps 9–10).

As with ISOCLUS, the running time of Lloyd's algorithm is dominated by the time to compute the nearest cluster center to each data point. Naively, this would require $O(kn)$ time. Kanungo *et al.* [87] presented a more efficient implementation of Lloyd's algorithm, called the *filtering algorithm*. Although its worst-case asymptotic running time is not better than the naive algorithm, this approach was shown to be quite efficient in practice. In this section we present a high-level description of the filtering algorithm. We also introduce an approximate version of this algorithm, in which points may be assigned, not to their nearest neighbor, but to an approximate nearest neighbor.

3.2.1 The kd-tree

If considered at a high level, the filtering process implicitly involves computing, for each of the k centers, some aggregate information for all the points that are closer to this center than any other. In particular, it computes the centroid of these points and some other statistical information that is used by the ISOCLUS algorithm. Thus, the process can be viewed very abstractly as answering a number of range queries involving k disjoint ranges, each being the Voronoi cell of some cluster center. As such, an approach based on hierarchical spatial subdivisions is natural.

The filtering algorithm builds a standard kd-tree [20], augmented with additional statistical information, which will be discussed. A *kd-tree* is a hierarchical decomposition of space into axis-aligned hyperrectangles called *cells*. Each node of the tree is implicitly associated with a unique cell and the subset of the points

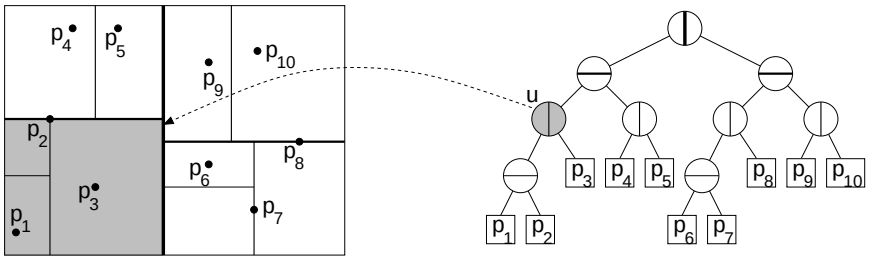


Fig. 3.1: An example of a kd-tree of a set of points in the plane, showing both the associated spatial subdivision (left) and the binary tree structure (right).

that lie within this cell. Each *internal node* of the kd-tree stores an axis-orthogonal *splitting hyperplane*. This hyperplane subdivides the cell into two subcells, which are associated with the left and right subtrees of the node. Nodes holding a single point are declared to be *leaves* of the tree. In Fig. 3.1, the highlighted node u of the tree is associated with the shaded rectangular cell shown on the left side of the figure and the subset $\{p_1, p_2, p_3\}$ of points. It is well known that a kd-tree on n points can be constructed in $O(n \log n)$ time in any fixed dimension [53].

3.2.2 The filtering process

We provide an overview of how the filtering algorithm is used to perform one iteration of Lloyd's algorithm. (See [87] for details.) Given a kd-tree for the data points S and the current set of k center points, the algorithm processes the nodes of the kd-tree in a top-down recursive manner, starting at the root. Consider some node u of the tree. Let $S(u)$ denote the subset of points S that are associated with this node. If it can be inferred that all the points of $S(u)$ are closer to some center $\mathbf{z}_j$ than to any other center (that is, the node's associated rectangular cell lies entirely within the Voronoi cell of $\mathbf{z}_j$), then we may *assign* u to cluster S_j. Every point

associated with u is thus *implicitly assigned* to this cluster. (For example, this is the case for the node associated with cell a as shown in Fig. 3.2.) If this cannot be inferred, then the cell is split, and we apply the process recursively to its two children. (This is the case for the node associated with cell b in the figure, which is split and whose two children are b_1 and b_2.) Finally, if the process arrives at a leaf node, which contains a single point, then we determine which center is closest to the point, and assign its associated node to this center. (This is the case for the node associated with cell c of the figure.)

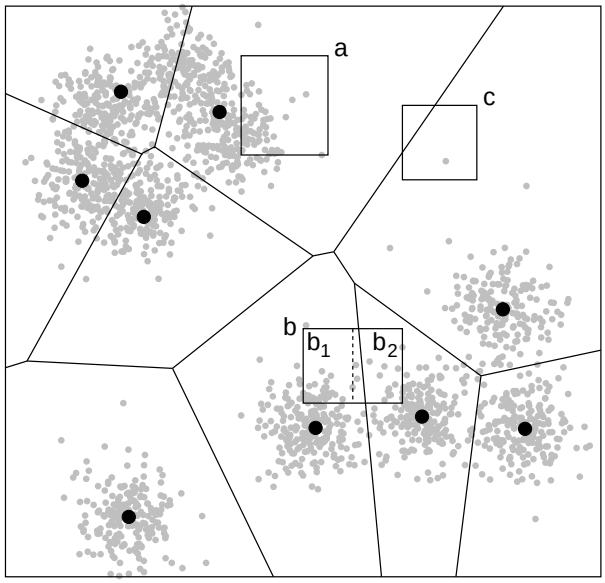


Fig. 3.2: Classifying nodes in the filtering algorithm. The subdivision is the Voronoi diagram of the centers, which indicates the neighborhood regions of each center.

At the conclusion of the process, the filtering algorithm assigns the nodes of the kd-tree to clusters in such a manner that every point of S is implicitly assigned to its closest cluster center. Furthermore, this is done so that the sets $S(u)$ assigned to a given cluster form a disjoint union of the associated cluster. There are two issues to be considered: (1) How to determine whether one center is closer to every

point of a node's cell than all other centers, and (2) when this occurs, how to assign *en masse* the points of the node to this center. We address these issues in reverse order in the following two sections.

3.2.3 Additional statistical information

As previously mentioned, the k-means algorithm seeks a placement of the centers that minimizes the average squared distance of each point to its nearest center. Formally, for each cluster S_j, we recall that $n_j = |S_j|$, and define the *average distortion* of the jth cluster, denoted $\Delta_j^{(2)}$, to be the average squared distance of each point in cluster S_j to its cluster center, that is,

$$\Delta_j^{(2)} = \frac{1}{n_j} \sum_{\mathbf{x} \in S_j} \|\mathbf{x} - \mathbf{z}_j\|^2.$$

(Contrast this quantity with the average distance Δ_j, computed in Step 5 of the ISOCLUS algorithm.) The overall distortion of the entire data set is the weighted average distortion among all clusters, where the weight factor for the jth cluster is n_j/n, that is, the fraction of points in this cluster.

In order to compute this information efficiently for each cluster, we store the following statistical information with each node u of the kd-tree. (Recall that each point of the data set is represented as a coordinate vector in $\mathbb{R}^d$.)

$\mathbf{s}(u)$: *weighted centroid*; contains the vector sum of the points associated with this node.

$ss(u)$: *sum of squares*; contains the sum of the dot products $(\mathbf{x} \cdot \mathbf{x})$ for all points $\mathbf{x}$ associated with this node.

$w(u)$: *weight*; contains the number of points associated with this node.

The above quantities can be computed in $O(dn)$ time by a simple postorder traversal of the kd-tree. We omit the straightforward details. The following lemma shows that once the set of nodes associated with a given center is known, the centroid of the set and the distortion of the resulting cluster can then be computed.

Lemma 3.2.1 *Consider a fixed cluster S_j, and let $U = \{u_1, u_2, \ldots, u_m\}$ be a set of nodes that are assigned to this cluster, so that S_j is the disjoint union of $S(u_i)$, for $1 \leq i \leq m$. Consider the following sums of the above quantities associated with the nodes in U:*

$$\mathbf{s}_j \;=\; \sum_{i=1}^{m} \mathbf{s}(u_i), \qquad ss_j \;=\; \sum_{i=1}^{m} ss(u_i), \qquad w_j \;=\; \sum_{i=1}^{m} w(u_i).$$

Then the size of the cluster is $n_j = w_j$, the centroid of the cluster is $(1/n_j)\mathbf{s}_j$, and the average distortion of the cluster is

$$\Delta_j^{(2)} \;=\; \frac{1}{w_j} ss_j - \frac{2}{w_j}(\mathbf{z}_j \cdot \mathbf{s}_j) + (\mathbf{z}_j \cdot \mathbf{z}_j).$$

Proof: Because $\bigcup_{i=1}^{m} S(u_i)$ is a disjoint partition of S_j the following identities hold:

$$\mathbf{s}_j \;=\; \sum_{\mathbf{x} \in S_j} \mathbf{x}, \qquad ss_j \;=\; \sum_{\mathbf{x} \in S_j} (\mathbf{x} \cdot \mathbf{x}), \qquad w_j \;=\; \sum_{\mathbf{x} \in S_j} 1 \;=\; |S_j| = n_j.$$

The first two claims follow directly from these identities, leaving only the expression of the average distortion to prove. In a slight abuse of notation, for two vectors $\mathbf{x}$ and $\mathbf{z}$, we express their dot products as $\mathbf{x}^2 = (\mathbf{x} \cdot \mathbf{x})$ and $\mathbf{xz} = (\mathbf{x} \cdot \mathbf{z})$. Then we can

express the total distortion for the jth cluster as:

$$\begin{aligned} n_j\Delta_j^{(2)} &= \sum_{\mathbf{x}\in S_j}(\mathbf{x}-\mathbf{z}_j)^2 = \sum_{\mathbf{x}\in S_j}(\mathbf{x}^2 - 2\mathbf{x}\mathbf{z}_j + \mathbf{z}_j^2) = \sum_{\mathbf{x}\in S_j}\mathbf{x}^2 - \sum_{\mathbf{x}\in S_j}2\mathbf{x}\mathbf{z}_j + \sum_{\mathbf{x}\in S_j}\mathbf{z}_j^2 \\ &= \sum_{\mathbf{x}\in S_j}(\mathbf{x}\cdot\mathbf{x}) - 2\left(\mathbf{z}_j\cdot\sum_{\mathbf{x}\in S_j}\mathbf{x}\right) + w_j(\mathbf{z}_j\cdot\mathbf{z}_j) \\ &= ss_j - 2(\mathbf{z}_j\cdot\mathbf{s}_j) + w_j(\mathbf{z}_j\cdot\mathbf{z}_j). \end{aligned}$$

The final result follows by dividing by $n_j = w_j$. □

3.2.4 Assigning nodes to centers

All that remains is to explain how the filtering algorithm assigns nodes to each of the cluster centers. Recall that the input to the algorithm is the set S given in the form of a kd-tree, the statistical quantities $\mathbf{s}(u)$, $ss(u)$, and $w(u)$ for each node u of the kd-tree, and the locations of the cluster centers $\mathbf{z}_j$. As the algorithm assigns a node u to a center $\mathbf{z}_j$, it adds these three quantities to the associated sums $\mathbf{s}_j$, ss_j, and w_j, as defined in the proof of Lemma 3.2.1. Upon termination of the algorithm, each center $\mathbf{z}_j$ is associated with the sum of these quantities for all the points S_j.

As previously mentioned, the filtering algorithm visits the nodes of the tree in a recursive top-down manner. For each node it visits, it maintains the subset of centers, called *candidates*, such that the closest center to any point in the node's cell is one of these candidate centers. Thus, for each node we keep track of a subset of centers that may serve as the nearest center for any point within the cell. Unfortunately, we know of no sufficiently efficient test to determine the set of true candidates (which involves determining the set of Voronoi cells overlapped by an

axis-aligned rectangle). Instead, we will describe a simple procedure that associates each node with a superset of its true candidates.

To start the process, the candidates for the root node of the kd-tree consists of all k centers. The centers are then filtered through the kd-tree as follows. Let C be the cell associated with the current node u, and let Z be the set of the candidate centers associated with C. First, the closest center $\mathbf{z}^* \in Z$ to the midpoint of C is computed. Then, for the rest of the candidates $\mathbf{z} \in Z \backslash \mathbf{z}^*$, if all parts of C are farther from $\mathbf{z}$ than they are from $\mathbf{z}^*$, we may conclude that $\mathbf{z}$ cannot serve as the nearest center for any point in u. So we can eliminate, or *filter*, $\mathbf{z}$ from the set of candidates. If there is only one candidate center (that is, $|Z| = 1$), then the node in question is assigned to this center. In particular, this means that the quantities $\mathbf{s}$, ss, and w for node u are added to the corresponding sums for this center. Otherwise, for an internal node, we pass the surviving set of candidates to its two children, and repeat the process recursively. If the algorithm reaches a leaf node having two or more candidates, the distances from all centers of Z to the node's data point are calculated, and this data point is assigned to the nearest candidate center.

In order to determine whether any part of C is closer to candidate $\mathbf{z}$ than to $\mathbf{z}^*$ we proceed as follows. Let H be the hyperplane bisecting the line segment $\overline{\mathbf{z}\mathbf{z}^*}$ (see Fig. 3.3). We can filter $\mathbf{z}$ if C is entirely on the same side of H as $\mathbf{z}^*$. This condition is tested through the use of a vector $\mathbf{w} = \mathbf{z} - \mathbf{z}^*$, from $\mathbf{z}^*$ to $\mathbf{z}$. Let $\mathbf{v}$ be the vertex of C that maximizes the dot product $(\mathbf{v} \cdot \mathbf{w})$, that is $\mathbf{v}$ is the farthest vertex in C in the direction of $\mathbf{w}$. If $\text{dist}(\mathbf{z}, \mathbf{v}) \geq \text{dist}(\mathbf{z}^*, \mathbf{v})$, then $\mathbf{z}$ is pruned. The choice of the vertex $\mathbf{v}$ can be determined simply by the signs of the individual coordinates of $\mathbf{w}$.

(See [87] for details.) The process requires $O(d)$ time for each center tested.

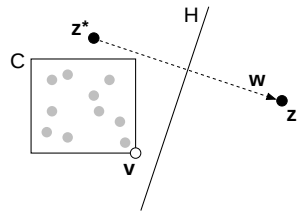


Fig. 3.3: Filtering process where **z** is pruned.

The filtering algorithm achieves its efficiency by assigning many points at once to each center. A straightforward implementation of Lloyd's algorithm requires $O(kn)$ time to compute the distance from each of the n points to each of the k centers. The corresponding measure of complexity for the filtering algorithm is the number of interactions between nodes and candidates. Kanungo *et al.* [87] have shown experimentally that this number is smaller by factors ranging from 10 to 200 for low dimensional clustered data sets. Even with the additional preprocessing time and overhead, the speed-ups in actual CPU time can be quite significant.

3.2.5 Approximate filtering

As with many approaches based on spatial subdivision methods, the filtering algorithm suffers from the so-called "curse of dimensionality," which in our context means that as the dimension increases the algorithm's running time increases exponentially as a function of the dimension. This was observed by Kanungo *et al.* in their analysis of the filtering algorithm [87]. The problem with high dimensions stems from the fact that any approach based on kd-trees relies on the hypothesis that the rectangular cell associated with each node is a good approximation to the

extent of the subset of points of S that lie within the cell. This is true when the dimension is low. As the dimension increases, however, the cell progressively becomes a poorer approximation to the set of points lying within it. As a result, the pruning process is less efficient, and more nodes need to be visited by the filtering algorithm before termination.

Our approach for dealing with this problem is to apply filtering in an approximate manner, and so to trade accuracy for speed. In our case, we allow the user to provide a parameter $\epsilon > 0$, and the filtering algorithm is permitted to assign each point of S to any center point that is within a distance of up to $(1 + \epsilon)$ times the distance to the closest center. This makes it easier to prune a cell from further consideration, and thus ameliorates the adverse effects arising in high dimensions.

This can be incorporated into the filtering process as follows. We recall the notation from the previous section, where u is the current node being processed, C and Z denote, respectively, the cell and set of candidate centers associated with u, and $\mathbf{z}^* \in Z$ is the closest center in Z to the midpoint of C. Given two vectors $\mathbf{x}$ and $\mathbf{z}$, we used the notation $\|\mathbf{xz}\|$ to denote the Euclidean length of the vector $\mathbf{z} - \mathbf{x}$. Our goal is to determine those centers $\mathbf{z} \in Z \backslash \{\mathbf{z}^*\}$, such that for every center $\mathbf{x} \in C$ we have $\|\mathbf{xz}^*\| \leq (1 + \epsilon)\|\mathbf{xz}\|$. All such center points $\mathbf{z}$ can be filtered. In geometric terms, this is equivalent to replacing the bisector test used in the exact algorithm with a test involving an approximate bisector, denoted $H_\epsilon(\mathbf{z}, \mathbf{z}^*)$. The latter is defined to be the set of points $\mathbf{x}$, such that $\|\mathbf{xz}^*\| = (1 + \epsilon)\|\mathbf{xz}\|$. (See Fig. 3.4.)

The hyperplane bisector test of the previous section must be adapted to deter-

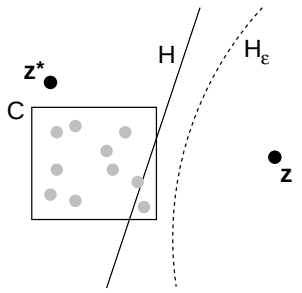


Fig. 3.4: Approximate filtering, where $\mathbf{z}$ is pruned.

mine whether C is stabbed by $H_\epsilon(\mathbf{z}, \mathbf{z}^*)$. At first, this seems to be a much harder test to perform. For example, it is no longer sufficient to merely test an appropriate vertex of C, since it is possible that the approximate bisector intersects the interior of a facet of C, while all the vertices lie to one side of the approximate bisector. What saves the day is the fact that the approximate bisector is a hypersphere, and hence the problem reduces to computing the distance between an axis-aligned rectangle and the center of this hypersphere, which can be computed easily. For completeness, we present the following two technical lemmas, which provide the necessary groundwork.

Lemma 3.2.2 *Given $\epsilon > 0$, and two points $\mathbf{z}$ and $\mathbf{z}^*$ in d-space, $H_\epsilon(\mathbf{z}, \mathbf{z}^*)$ is a $(d-1)$-sphere of radius r_ϵ centered at the point $\mathbf{c}_\epsilon$, where*

$$r_\epsilon = \frac{1+\epsilon}{\gamma - 1}\|\mathbf{z}\mathbf{z}^*\| \qquad \textit{and} \qquad \mathbf{c}_\epsilon = \frac{1}{\gamma - 1}(\gamma\mathbf{z} - \mathbf{z}^*), \qquad \textit{where} \qquad \gamma = (1+\epsilon)^2.$$

Proof: A point $\mathbf{x}$ lies on H_ϵ if and only if

$$\|\mathbf{x}\mathbf{z}^*\|^2 = (1+\epsilon)^2\|\mathbf{x}\mathbf{z}\|^2.$$

As before, it will be convenient to express dot products using $\mathbf{x}^2 = (\mathbf{x} \cdot \mathbf{x})$ and

$\mathbf{xz} = (\mathbf{x} \cdot \mathbf{z})$. The above is equivalent to

$$\begin{aligned}(\mathbf{x} - \mathbf{z}^*)^2 &= (1+\epsilon)^2(\mathbf{x} - \mathbf{z})^2 \\ \mathbf{x}^2 - 2\mathbf{x}\mathbf{z}^* + \mathbf{z}^{*2} &= \gamma(\mathbf{x}^2 - 2\mathbf{x}\mathbf{z} + \mathbf{z}^2).\end{aligned}$$

Expanding and completing the square yields

$$\begin{aligned}(\gamma-1)\mathbf{x}^2 - 2(\gamma\mathbf{z} - \mathbf{z}^*)\mathbf{x} + (\gamma\mathbf{z}^2 - \mathbf{z}^{*2}) &= 0 \\ \mathbf{x}^2 - \frac{2}{\gamma-1}(\gamma\mathbf{z} - \mathbf{z}^*)\mathbf{x} + \frac{1}{(\gamma-1)^2}(\gamma\mathbf{z} - \mathbf{z}^*)^2 &= \\ \frac{1}{(\gamma-1)^2}(\gamma\mathbf{z} - \mathbf{z}^*)^2 - \frac{1}{\gamma-1}(\gamma\mathbf{z}^2 - \mathbf{z}^{*2}) &= \\ \left(\mathbf{x} - \frac{1}{\gamma-1}(\gamma\mathbf{z} - \mathbf{z}^*)\right)^2 &= \\ \frac{1}{(\gamma-1)^2}(\gamma\mathbf{z} - \mathbf{z}^*)^2 - \frac{1}{\gamma-1}(\gamma\mathbf{z}^2 - \mathbf{z}^{*2}).\end{aligned}$$

The left-hand side is $(\mathbf{x} - c_\epsilon)^2$. Expanding the right-hand side gives

$$\begin{aligned}(\mathbf{x} - c_\epsilon)^2 &= \frac{1}{(\gamma-1)^2}((\gamma\mathbf{z} - \mathbf{z}^*)^2 - (\gamma-1)(\gamma\mathbf{z}^2 - \mathbf{z}^{*2})) \\ &= \frac{1}{(\gamma-1)^2}((\gamma^2\mathbf{z}^2 - 2\gamma\mathbf{z}\mathbf{z}^* + \mathbf{z}^{*2}) - (\gamma^2\mathbf{z}^2 - \gamma\mathbf{z}^{*2} - \gamma\mathbf{z}^2 + \mathbf{z}^{*2})) \\ &= \frac{1}{(\gamma-1)^2}(\gamma\mathbf{z}^2 - 2\gamma\mathbf{z}\mathbf{z}^* + \gamma\mathbf{z}^{*2}) \\ &= \frac{\gamma}{(\gamma-1)^2}(\mathbf{z} - \mathbf{z}^*)^2 = \left(\frac{1+\epsilon}{\gamma-1}\|\mathbf{z}\mathbf{z}^*\|\right)^2 = r_\epsilon^2.\end{aligned}$$

This is the equation of the desired hypersphere. □

Lemma 3.2.3 *The closest (Euclidean) distance between an axis-aligned hyperrectangle in $\mathbb{R}^d$ and any point $\mathbf{c} \in \mathbb{R}^d$ can be computed in $O(d)$ time.*

Proof: Let $\mathbf{v} = (v_1, \ldots, v_d)$ and $\mathbf{w} = (w_1, \ldots, w_d)$ be the rectangle vertices with the lowest and highest coordinate values, respectively. (For example, these would

be the lower left and upper right vertices in the planar case.) The rectangle is just the d-fold intersection of axis-orthogonal strips

$$\{(x_1, \ldots, x_d) \mid v_i \leq x_i \leq w_i\}.$$

Based on the location of $\mathbf{c}$ relative to each of these strips, we can compute the squared distance from $\mathbf{c} = (c_1, \ldots, c_d)$ to the rectangle as $\sum_{i=1}^{d} \delta_i^2$, where

$$\delta_i = \begin{cases} v_i - c_i & \text{if } c_i < v_i \\ 0 & \text{if } v_i \leq c_i \leq w_i \\ c_i - w_i & \text{if } w_i < c_i. \end{cases}$$

The final distance is the square root of this sum. □

Using these two lemmas, it is now easy to see how to replace the exact filtering step described in the previous section with an approximate filtering test, which also runs in $O(d)$ time. Given candidate centers $\mathbf{z}$ and $\mathbf{z}^*$, we apply Lemma 3.2.2 to compute r_ϵ and $\mathbf{c}_\epsilon$. We then apply Lemma 3.2.3 to compute the closest distance between the cell C and $\mathbf{c}_\epsilon$. If this distance is greater than r_ϵ, then $\mathbf{z}$ is pruned. The remainder of the algorithm is the same. In Section 3.4.3 below, we present experimental evidence for the benefits of using approximate filtering.

Although points are assigned to cluster centers that are ϵ-nearest neighbors, it does not follow that the result produced by the approximate version of the ISOCLUS algorithm results in an ϵ-approximation in the sense of distortion. The reason is that ISOCLUS is a heuristic and does not provide any guarantees on the resulting distortion. It follows some path in the space of possible solutions to some local minimum. Even a minor change to the algorithm's definition can alter this path, and may lead to a local minimum of a significantly different value, either larger or smaller.

3.3 Our Modifications and Improvements

As mentioned earlier, most of the computational effort in the ISOCLUS algorithm is spent calculating and updating distances and distortions in Steps 2–5. These steps take $O(kn)$ time, whereas all the other steps can be performed in $O(k)$ time, where k is the current number of centers. Our improvement is achieved by adapting the filtering algorithm to compute the desired information. This is the reason for computing the additional statistical information, which was described in the previous section.

There is one wrinkle, however. The filtering algorithm achieves its efficiency by processing points in groups, rather than individually. This works fine as long as the statistical quantities being used by the algorithm can be computed in an aggregated manner. This is true for the centroid, as shown in Lemma 3.2.1, since it involves the sum of coordinates. Generally, the filtering method can be applied to any polynomial function of the point and center coordinates. However, there is one statistical quantity computed by the ISOCLUS algorithm that does not satisfy this property. In particular, Step 5 of the ISOCLUS algorithm involves computing the sum of Euclidean distances from each point to its closest center as a measure of the dispersion of the cluster. This information is used later in Step 8 to determine whether to split the cluster. This involves computing the sum of square roots, and we know of no way to aggregate this processing.

Rather than implementing ISOCLUS exactly as described in [124], we modified Step 5 as follows. For each cluster j, instead of computing the average Euclidean

distance of each point to its center, Δ_j, we compute the average *squared Euclidean distance*, denoted $\Delta_j^{(2)}$. In order to preserve the metric units, we use the square root of this quantity, denoted Δ'_j. In short, we modified the definitions of Step 5 by computing the following quantities:

$$\begin{aligned}
\Delta_j^{(2)} &= \frac{1}{n_j} \sum_{\mathbf{x} \in S_j} \|\mathbf{x} - \mathbf{z}_j\|^2, \qquad \text{for } \ 1 \leq j \leq k \\
\Delta'_j &= \sqrt{\Delta_j^{(2)}}, \qquad \text{for } \ 1 \leq j \leq k \\
\Delta' &= \frac{1}{n} \sum_{j=1}^{k} n_j \Delta'_j.
\end{aligned}$$

The decision as to whether to split a cluster in Step 8 depends on the relative sizes of Δ'_j and Δ', rather than Δ_j and Δ. Note that this can produce different results. Nonetheless, having experimented with both synthetically generated data and real images, we observed that the actual performance of our algorithm was quite similar to that of ISOCLUS, in terms of the number of clusters obtained and the positions of their centers. This will be demonstrated in the next section. Thus, we believe that this modification does not significantly alter the nature of the algorithm, and has the benefit of running significantly faster. The value Δ'_j can be computed as outlined in Lemma 3.2.1. In the next section we present the experimental results obtained using our convention, and in Section 3.5 we provide theoretical justification for the modifications made.

3.4 Experiments

In order to establish the efficiency of both our new exact and approximate algorithmic versions, and to determine the degree of similarity in clustering perfor-

mance with the existing ISOCLUS algorithm, we ran a number of experiments on synthetic data, as well as remotely sensed images. Our modified algorithm involves changing both the functionality and computational approaches. To make the comparisons clearer, we implemented an intermediate, or *hybrid*, algorithm, which is functionally equivalent to one variant but uses the same computational approach as the other.

Standard version (Std): The straightforward implementation of ISOCLUS as described in [124], which uses average Euclidean distances in Step 5 and Step 8.

Hybrid version (Hyb): A modification of the standard version using Δ'_j and Δ' rather than Δ_j and Δ in Step 5 and Step 8, but without using the filtering algorithm.

Filtering version (Fil): The same modification, but using the filtering algorithm for greater efficiency.

The Hybrid and Filtering versions are functionally equivalent, but use different computational approaches. The Standard and Hybrid versions are roughly equivalent in terms of the computational methods, but are functionally distinct. Our goal is to show that the Standard and Hybrid versions are nearly functionally equivalent, and that in many instances the Filtering version is significantly more efficient. All experiments were run on a SUN Blade 100 running Solaris 2.8, using the g++ compiler (version 2.95.3).

We mention for completeness that we also implemented and tested a fourth version, the results of which are not reported, as they were not competitive with

the filtering algorithm. The latter variant stores the k center points in a kd-tree, as implemented in the ANN library [13]. The nearest center to each data point is then computed by a search of this tree. This approach proved to be consistently slower than the filtering algorithm for two reasons. First, there are significantly fewer center points than query points ($k \ll n$). Thus, there are lower savings in running time that would result by storing the k center points in a tree as compared to the savings that result by storing the n data points in a tree. Second, the center points change with each iteration, and so the tree would need to be rebuilt constantly. We also compared the performance of our code with a similar software called *gmeans* [38, 39]. This software is efficient for high dimensional sparse data. For low dimensional dense data, however, gmeans calculates distances in a brute-force manner. Our software was faster by roughly an order of magnitude, with lower or comparable final distortion values.

The remainder of this chapter is devoted to presenting the results of the various experiments we ran. Section 3.4.1 presents the performance of these algorithms on synthetically generated clustered data sets of various sizes and in various dimensions. In Section 3.4.2 we present experiments on data sets generated from an application in remote sensing, in which ISOCLUS is regularly used. Next, in Section 3.4.3 we investigate the performance of the approximate version of the filtering algorithm. Finally, in Section 3.4.4 we consider the effect of increasing the dimension of the data set on the running time and speed-up, for both the exact and approximate versions.

3.4.1 Synthetic data

We ran the following three sets of experiments on synthetically generated data sets to analyze the performance of our algorithm. All experiments were run in dimensions 3, 5, and 7. (This choice of dimensions was guided by the fact that many applications of ISOCLUS in remote sensing involve Landsat satellite image data. Raw Landsat data contains 7 spectral bands, and reductions to dimensions 3 and 5 are quite common.)

(1) For the first set of experiments we generated $n = 10,000$ data points. In each case the points were sampled with equal probability from a variable number of Gaussian clusters ranging from 10 to 100, by a method described below.

(2) In the second set of experiments five data sets were considered, containing 100, 500, 1000, 5000, and $10,000$ points, respectively. In each case the points were distributed evenly among 20 Gaussian clusters.

(3) In the third set of experiments, we varied both the number of randomly generated points and the number of clusters. Specifically, we considered data sets containing 100, 500, 1000, 5000, and $10,000$ points. For each data set, the points were distributed evenly among 5, 10, 20, 40, and 80 Gaussian clusters.

All of the above experiments involved points drawn from a collection of some number k of Gaussian clusters. This was done as follows. Cluster centers were sampled uniformly at random from the hypercube $[-1, 1]^d$ of side length 2. In order to generate a point for each cluster, a vector was generated, such that each of its

coordinates was drawn from a Gaussian distribution with a given standard deviation $\sigma = (1/k)^{1/d}$.

The value of σ was derived by the following reasoning. In order for the results to be comparable across different dimensions and with different numbers of clusters, it is desirable that clusters have comparable degrees of overlap. In low dimensions, a significant amount of the probability mass of a Gaussian cluster lies within a region whose volume is proportional to $(2\sigma)^d$. We wish to subdivide a cube of unit volume uniformly into k clusters, which suggests that each cluster should cover $1/k$-th of the total volume, and hence σ should be chosen such that $(2\sigma)^d = 2^d/k$, from which the above value of σ was obtained.

We ran the ISOCLUS algorithms for a maximum of 20 iterations ($I_{\max} = 20$). In each case the initial number of clusters was set to the actual number of clusters generated ($k_{\text{init}} = k$), the maximum cluster standard deviation was set to twice the standard deviation of the distribution ($\sigma_{\max} = 2\sigma$), and the minimum cluster separation was set to 0.001 ($L_{\min} = 0.001$). We decided to remove a cluster if it contained fewer than 1/5 of the average cluster size, and so set $n_{\min} = n/(5k_{\text{init}})$. For the first set of experiments where $n = 10,000$ was fixed, we set the initial number of clusters to 10, 20, 40, 80, and 100, in accordance with the respective number of actual clusters generated. In each case, the results were averaged over five runs. The results of the above 3 sets of experiments are shown in Table 3.1, Table 3.2, and Table 3.3.

For each run, we computed the running time in CPU seconds, the final number of centers, and the final average distortion. Not surprisingly, since the hybrid and

Table 3.1: Results for synthetic data with n = 10,000

Dim	k_{init}	Final Centers		Avg. Distortion		CPU Seconds			Speed-
		Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	up
3	10	10	10	0.385	0.385	5.00	5.14	1.420	3.62
	20	20	20	0.286	0.286	8.74	9.07	1.788	5.07
	40	40	40	0.120	0.120	16.39	17.20	2.022	8.50
	80	78	78	0.077	0.077	33.87	34.99	2.626	13.32
	100	100	100	0.061	0.061	43.34	44.95	2.956	15.21
5	10	9	9	2.108	2.108	6.86	6.77	3.092	2.189
	20	19	19	1.184	1.184	12.58	13.20	3.880	3.403
	40	36	36	0.819	0.819	21.79	22.83	5.372	4.249
	80	79	79	0.490	0.490	48.69	50.01	7.998	6.253
	100	93	93	0.478	0.478	57.67	59.30	9.172	6.465
7	10	10	10	4.062	4.062	9.18	9.38	5.29	1.771
	20	20	20	1.971	1.971	16.31	16.82	7.40	2.274
	40	32	32	2.303	2.303	26.11	26.59	11.50	2.312
	80	74	74	1.447	1.447	59.27	60.29	22.07	2.732
	100	93	93	1.242	1.242	75.27	76.55	26.02	2.942

Table 3.2: Results for synthetic data with $k_{\text{init}} = 20$

Dim	n	Final Centers		Avg. Distortion		CPU Seconds			Speed-
		Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	up
3	100	20	20	0.164	0.164	0.100	0.088	0.048	1.833
	500	20	20	0.278	0.278	0.446	0.428	0.148	2.892
	1000	20	20	0.265	0.265	0.902	0.876	0.256	3.422
	5000	20	20	0.288	0.288	4.512	4.232	0.960	4.408
	10000	20	20	0.286	0.286	9.290	8.804	1.770	4.974
5	100	20	20	0.828	0.828	0.138	0.116	0.092	1.261
	500	17	17	1.095	1.095	0.560	0.556	0.286	1.944
	1000	20	20	1.074	1.074	1.368	1.300	0.600	2.167
	5000	19	19	1.188	1.188	6.304	6.130	2.234	2.744
	10000	19	19	1.184	1.184	12.958	12.812	3.868	3.312
7	100	20	20	1.349	1.349	0.168	0.158	0.112	1.411
	500	17	17	1.957	1.957	0.690	0.692	0.478	1.450
	1000	18	18	1.990	1.990	1.546	1.526	0.924	1.652
	5000	19	19	1.990	1.990	8.078	7.994	4.146	1.928
	10000	20	20	1.971	1.971	16.740	16.604	7.472	2.222

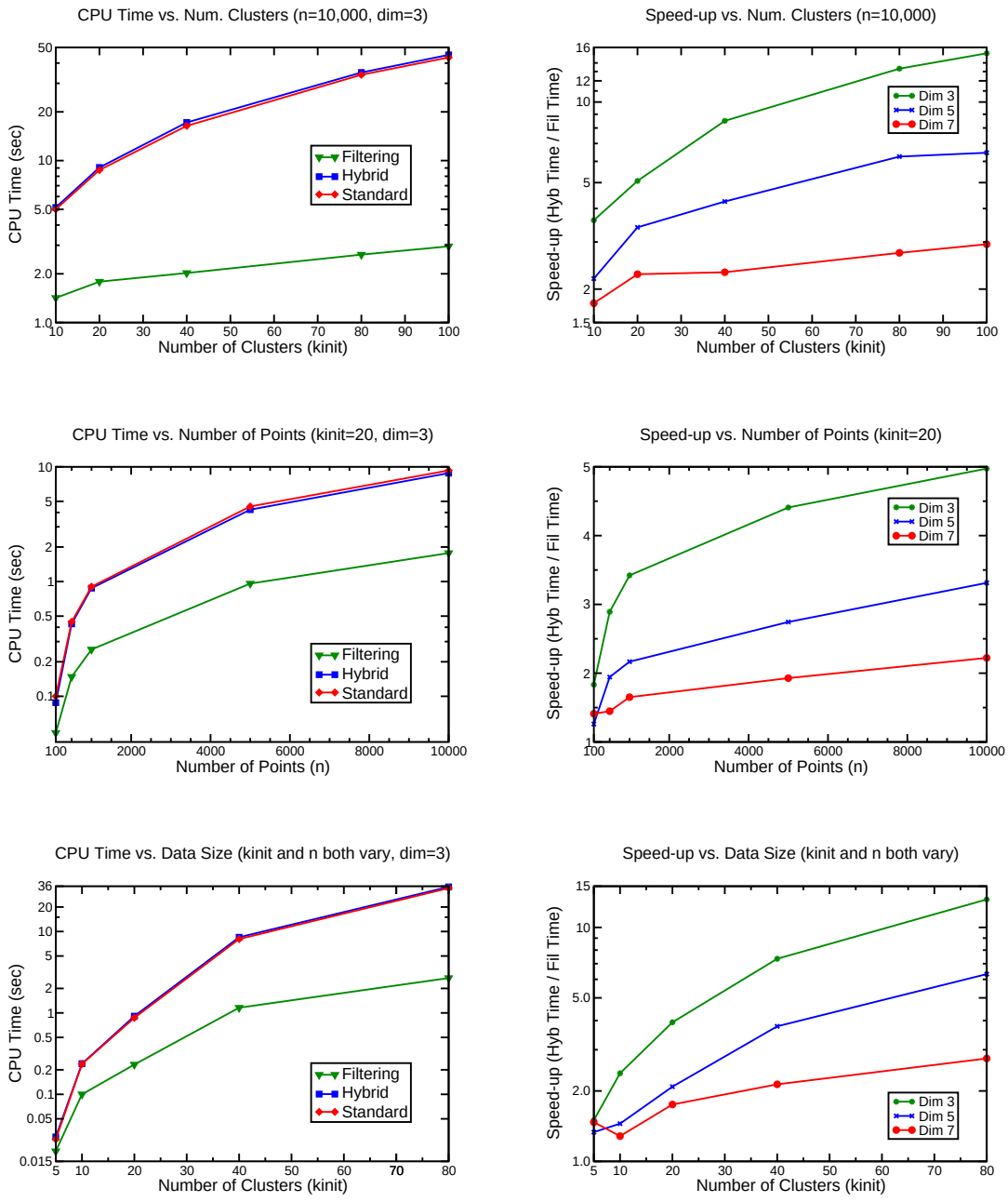


Fig. 3.5: CPU times and speed-ups for the various algorithms run on synthetic data. (Note that the x and y axes do not intersect at the origin.) For the bottom pair of plots, note that n also varies with k_{init} as indicated in Table 3.3.

Table 3.3: Results for synthetic data where both n and k_{init} vary

Dim	n	k_{init}	Final Centers		Avg. Distortion		CPU Seconds			Speed-
			Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	up
3	100	5	5	5	0.480	0.480	0.028	0.03	0.020	1.500
	500	10	10	10	0.357	0.357	0.236	0.24	0.100	2.380
	1000	20	20	20	0.265	0.265	0.870	0.91	0.232	3.931
	5000	40	40	40	0.120	0.120	8.082	8.50	1.158	7.344
	10000	80	78	78	0.077	0.077	34.074	35.33	2.682	13.174
5	100	5	5	5	2.350	2.350	0.036	0.04	0.030	1.334
	500	10	8	8	2.095	2.095	0.280	0.29	0.200	1.450
	1000	20	20	20	1.074	1.074	1.280	1.27	0.608	2.086
	5000	40	39	39	0.797	0.797	11.904	12.12	3.208	3.778
	10000	80	79	79	0.490	0.490	48.470	50.49	7.994	6.316
7	100	5	4	4	4.417	4.417	0.042	0.05	0.03	1.471
	500	10	9	9	4.321	4.321	0.398	0.42	0.33	1.282
	1000	20	18	18	1.990	1.990	1.550	1.58	0.90	1.750
	5000	40	36	36	2.201	2.201	14.782	15.07	7.06	2.135
	10000	80	74	74	1.447	1.447	46.966	60.69	22.04	2.753

filtering versions implement the same functional specifications, the final numbers of centers and final distortions obtained were almost identical. (Small differences were observed due to floating point round-off errors.) Thus, we listed together the corresponding results in the tables (under "Hyb/Fil"). We also computed the *speed-up*, which is defined as the ratio between the CPU time of the hybrid version and that of the filtering version.

In support of our claim that using squared distances does not significantly change the algorithm's clustering performance, observe that both algorithms performed virtually identically with respect to average distortions and the final number of centers. Also observe that the standard and hybrid versions ran in roughly the same time, whereas the filtering version ran around 1.3 to 15.2 times faster than the other two. Fig. 3.5 shows our experimental results on the synthetic data sets.

We can see that for a fixed number of points, increasing the number of clusters increases both the CPU time and speed-up. The same result holds when we increase the number of points and fix the other parameters.

3.4.2 Image data

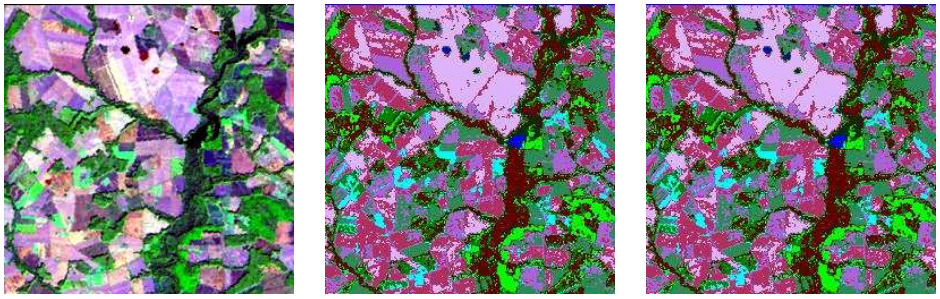
For image data we used two different data sets from remotely sensed imagery: A Landsat data set and a MODIS scene. For the Landsat data we ran nine tests on a 256×256 image of Ridgely, Maryland ($n = 65,536$). The first set of experiments involved three tests on 3-dimensional data using spectral bands 3, 4, and 5. The initial number of clusters was set to 10, 50, and 100. This choice covers the range of values used in typical remote sensing applications. The second set of experiments was performed in 5-dimensional space using spectral bands 3 through 7, and the third set was carried out in 7-dimensional space using all seven bands. The tests in dimensions 5 and 7 were performed with 10, 50, and 100 initial centers (k_{init}), as well. We ran all nine tests with the three versions of ISOCLUS, each for 20 iterations, $\sigma_{\max} = 15$, $L_{\min} = 10$, and $n_{\min} = n/(5k_{\text{init}})$ (approximately), and k_{init} of 10, 50, and 100. Each experiment was run 10 times, invoking every time the algorithmic version in question with a different set of initial random centers. The results obtained were averaged over these 10 runs. (This accounts for the noninteger number of "Final Centers" reported in the tables.)

The results are summarized in Table 3.4. As with the tests on synthetic data, all versions performed essentially equivalently with respect to the number of centers

and final distortions. The filtering version was faster by a factor of roughly 4 to 30. Fig. 3.6 shows the original data and the clustered images obtained due to the standard and filtering ISOCLUS in 3-dimensional space. (As indicated, the clusters for the two versions were essentially identical.)

Table 3.4: Results for Landsat data set

Dim	k_{init}	Final Centers		Avg. Distortion		CPU Seconds			Speed-
		Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	up
3	10	6.3	6.3	67.92	67.86	28.109	27.370	5.838	4.688
	50	10.1	9.9	43.49	44.11	84.729	82.213	7.182	11.447
	100	22.1	22.9	25.31	24.55	290.110	280.470	9.117	30.763
5	10	5.9	5.9	144.04	144.04	43.989	43.169	10.352	4.170
	50	15.6	15.7	85.50	91.02	174.590	171.160	17.778	9.628
	100	23.9	22.7	33.93	35.12	367.130	359.200	18.748	19.159
7	10	7.3	7.3	169.17	169.17	62.214	61.277	15.788	3.881
	50	15.8	15.8	107.68	107.68	206.720	203.610	26.659	7.638
	100	22.1	22.1	46.21	46.21	442.650	430.860	31.655	13.611



(a) (b) (c)

Fig. 3.6: A Landsat scene and its clustered images: (a) 256×256 Landsat image of Ridgely, Maryland (bands 3, 4, and 5), (b) clustered image due to standard ISOCLUS, and (c) clustered image due to the Filtering variant.

For the MODIS data set we repeated the above three sets of experiments on a 128×128 ($n = 16,384$) subimage acquired over an agricultural area from the Konza Prairie in Kansas. The results are summarized in Table 3.5. As with the Landsat data set, we experimented in dimensions 3, 5, and 7, only that here the spectral

bands were selected through principal component analysis (PCA) by the standard approach based on the Karhunen-Loéve transformation [54].

The initial number of clusters experimented with in each case was 10, 50, and 100. The remaining parameters used were essentially the same as those for the Landsat data set, except for $n_{\min} = 45$. As before, each experiment was repeated 10 times, invoking the algorithm in question every time with a different set of initial random centers. The results reported were averaged over these 10 runs.

Table 3.5: Results for MODIS data set

Dim	k_{init}	Final Centers		Avg. Distortion		CPU Seconds			Speed-
		Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	up
3	10	17.2	17.8	389.69	383.46	14.515	14.577	2.231	6.534
	50	51.7	51.7	177.95	177.94	38.562	37.209	2.784	13.365
	100	98.4	98.4	114.86	114.87	83.172	80.444	3.600	22.345
5	10	20.3	21.0	970.00	946.91	22.761	21.901	4.345	5.041
	50	69.5	69.5	478.09	478.09	82.020	79.916	7.450	10.727
	100	116	116.0	372.55	372.56	143.360	139.910	9.805	14.269
7	10	20.8	20.8	1437.30	1443.00	28.506	28.360	6.454	4.394
	50	79.5	81.2	728.53	722.13	143.950	144.740	16.950	8.539
	100	134.9	134.5	564.94	565.80	255.950	252.320	24.393	10.344

The final results in dimensions 3 and 5 were identical with respect to both the final number of clusters and the final distortions. In dimension 7, while all versions of the algorithm resulted in an (almost) identical final number of clusters, their distortions were slightly different. The filtering version was faster by factors ranging from roughly 4 to 22. The speed-ups were most dramatic for the cases involving a large numbers of clusters. This is to be expected because the filtering algorithm achieves its improvement by eliminating unpromising candidate centers from consideration.

3.4.3 Experiments with approximate filtering

In order to better understand the effect of approximation, we experimented with the approximate version of the filter-based algorithm. Recall that the algorithm differs from the exact algorithm in how candidate centers are pruned from each node in the process of determining which center is closest to the points of a node. The user supplies a value $\epsilon > 0$, and the algorithm may assign a point to a center whose distance (from the point) is (up to) $(1 + \epsilon)$ times the point's distance to its true nearest center. We performed experiments on both synthetic and satellite image data. In all cases, we ran experiments with approximation parameter $\epsilon \in \{0.1, 0.2, 0.5, 1.0, 1.5\}$, and compared the results against the exact ($\epsilon = 0$) case. Note that approximation was used in all but the last iteration of the algorithm, in which case exact pruning was performed. The reason is that when the algorithm terminates, we want all the points to be assigned to their true closest center.

The use of ϵ values greater than 1 may seem to be unreasonably large for practical purposes, since this allows for more than 100% relative error. But note that the ϵ value is merely an upper bound on the error committed for each individual point-to-center assignment, and the aggregated effect of these errors is subject to cancelation and may be much smaller. As we shall see below, even for fairly large values of ϵ, the observed distortions relative to the exact version of ISOCLUS were almost always less than 5%.

As mentioned at the end of Section 3.2.5, ISOCLUS is a heuristic and not an optimization algorithm. Thus, minor changes to the algorithm can result in

convergence to local minima with significantly different average distortions. This can happen even when $\epsilon = 0$, because the algorithm is invoked with random initial center points. For this reason, all of the results were averaged over the number of invocations of the algorithm.

For synthetic data, we generated five random sets of $n = 10,000$ points in dimensions 3, 5, and 7. Points were sampled with equal probability from 100 Gaussian clusters with uniformly distributed centers. The distributions and program parameter settings were the same as for the experiments on synthetic data of Section 3.4.1. We measured the CPU time, the final distortion, and the final number of clusters in each experiment. Finally, we evaluated the algorithm's relative performance with respect to the standard version of ISOCLUS (by invoking the latter on the same data sets). We computed (average) speed-ups, as well as relative distortion errors with respect to the standard version. These results are summarized in Table 3.6.

For the satellite image data, we used the same Landsat and MODIS data sets and the same parameter settings described in Section 3.4.2. Also, we used the same experimental setup described above (for the approximate version). The results are shown in Table 3.7 and Table 3.8 for the Landsat and MODIS data sets, respectively.

The results demonstrate that approximation can result in significant speed-ups. In spite of the relatively large values of ϵ supplied, it is noteworthy that the average error in the final distortion relative to the exact case ("Rel Dist Err %") was dramatically smaller. It never exceeded 8% and was usually less than 3%. The phenomenon of a geometric approximation algorithm performing significantly better on average than the allowable error bound has been observed elsewhere [13]. Since

Table 3.6: Results for synthetic data with Approx. Filtering, $n = 10,000$, and $k_{\text{init}} = 100$

Dim	ϵ	Final Centers		Avg. Dist.×100		CPU Seconds		Speed-up	Rel Dist
		Std	Fil	Std	Fil	Std	Fil		Err %
3	0.0	98.2	98.2	6.09	6.09	43.12	2.72	15.85	0.00
	0.1		98.2		6.09		4.36	9.89	0.00
	0.2		98.4		6.09		3.90	11.06	0.00
	0.5		98.6		6.11		2.88	14.97	0.33
	1.0		98.0		6.35		1.99	21.67	4.27
	1.5		97.4		6.57		1.61	26.78	7.88
5	0.0	94.6	94.6	47.20	47.20	58.44	8.84	6.61	0.00
	0.1		94.6		47.20		17.40	3.36	0.00
	0.2		94.8		47.11		14.39	4.06	-0.19
	0.5		95.6		47.10		9.40	6.22	-0.21
	1.0		96.8		47.58		5.89	9.92	0.81
	1.5		96.6		48.59		4.20	13.91	2.94
7	0.0	93.2	93.2	124.76	124.76	73.70	24.64	2.99	0.00
	0.1		93.2		124.76		48.63	1.52	0.00
	0.2		93.2		124.76		39.79	1.85	0.00
	0.5		93.6		124.62		23.45	3.14	-0.11
	1.0		93.0		126.36		12.19	6.05	1.28
	1.5		93.4		129.04		7.48	9.85	3.43

Table 3.7: Results for Landsat data set with Approx. Filtering, $k_{\text{init}} = 25$

Dim	ϵ	Final Centers		Avg. Distortion		CPU Seconds		Speed-up	Rel Dist
		Std	Fil	Std	Fil	Std	Fil		Err %
3	0.0	7.9	7.9	56.58	56.85	47.31	6.61	7.16	0.47
	0.1		7.9		58.03		6.54	7.23	2.56
	0.2		7.8		58.05		5.93	7.98	2.60
	0.5		7.9		57.28		5.35	8.84	1.24
	1.0		8.3		54.87		5.12	9.24	-3.02
	1.5		8.5		55.14		5.06	9.35	-2.55
5	0.0	9.7	9.7	114.32	114.43	88.82	14.17	6.27	0.10
	0.1		9.4		115.26		16.45	5.40	0.82
	0.2		9.3		116.82		13.96	6.36	2.19
	0.5		9.5		109.81		9.45	9.40	-3.95
	1.0		9.2		114.52		7.91	11.23	0.17
	1.5		8.5		116.87		7.79	11.40	2.23
7	0.0	10.8	10.9	139.85	137.75	115.51	20.06	5.76	-1.50
	0.1		11.0		135.88		28.36	4.07	-2.83
	0.2		10.8		137.38		24.36	4.74	-1.77
	0.5		11.2		135.47		17.00	6.79	-3.13
	1.0		10.3		137.34		10.86	10.64	-1.79
	1.5		9.0		145.46		9.64	11.98	4.01

Table 3.8: Results for MODIS data set with Approx. Filtering, $k_{\text{init}} = 75$

Dim	ϵ	Final Centers		Avg. Distortion		CPU Seconds		Speed-up	Rel Dist
		Std	Fil	Std	Fil	Std	Fil		Err %
3	0.0	74.9	74.9	137.93	138.04	58.83	3.18	18.50	0.08
	0.1		74.6		138.59		4.44	13.25	0.48
	0.2		74.5		138.74		3.86	15.24	0.59
	0.5		74.3		141.23		2.74	21.47	2.39
	1.0		75.6		143.55		1.90	30.96	4.07
	1.5		76.8		145.28		1.71	34.40	5.33
5	0.0	93.5	93.5	412.43	412.80	112.62	8.76	12.86	0.09
	0.1		93.3		413.88		15.39	7.32	0.35
	0.2		93.2		414.00		12.69	8.87	0.38
	0.5		94.1		412.80		8.39	13.42	0.09
	1.0		98.7		410.71		5.76	19.55	-0.42
	1.5		104.9		401.72		4.59	24.54	-2.60
7	0.0	106.3	106.1	633.54	635.31	180.77	18.63	9.70	0.28
	0.1		105.8		635.66		32.81	5.51	0.33
	0.2		106.3		634.93		27.71	6.52	0.22
	0.5		105.5		636.08		17.52	10.32	0.40
	1.0		110.0		631.15		10.73	16.85	-0.38
	1.5		118.4		618.51		8.60	21.02	-2.37

ISOCLUS is a heuristic, it is possible for the approximate version to converge on a better local minimum, and so in some cases the distortion error is actually negative.

It is also noteworthy that the approximate algorithm achieved speed-ups of up to one order of magnitude with low average distortion errors throughout the range of parameter values. Note that increasing ϵ did not always lead to a decrease in execution time. This is because of the sensitivity of ISOCLUS to its starting configuration, which further affects the number of iterations and the number of clusters and their structure.

3.4.4 Dependence on the dimension

In this section we study the effect of the dimension of the data set on the running times for various versions of our algorithm. Because of their greater sensitivity to the dimension, the filtering and approximate filtering algorithms exhibit poorer speed-ups as the dimension of the data set increases. To investigate this phenomenon more thoroughly, we generated a synthetic data set of 50,000 points (as described in Section 3.4.1). We ran experiments in various dimensions for the standard, hybrid, filtering, and approximate filtering algorithms. For the approximate version we considered ϵ values ranging from 0 (equivalent to pure filtering) to 2. The dimensions considered range from 2 to 35. Each experiment was run 5 times, invoking each run with a different set of 100 randomly selected centers ($k_{\text{init}} = 100$). The final number of clusters, distortions, and running times were measured and averaged over these five runs.

The results for the standard, hybrid, and (exact) filtering algorithms are presented in Table 3.9 and Fig. 3.7(a). We see that while filtering yields identical performance to the standard and hybrid versions, in terms of the final number of clusters and distortions, the speed-ups diminish rapidly with the dimension. Nonetheless, it is interesting to note that speed-ups greater than 1 are obtained even for dimensions as high as 35.

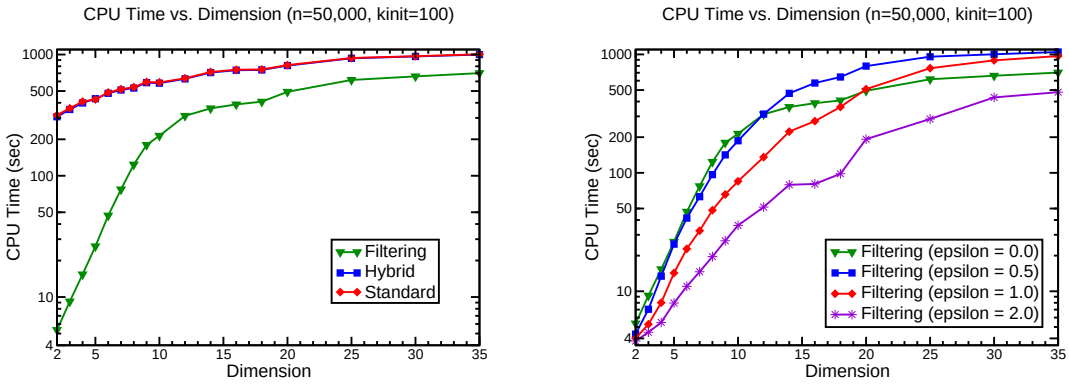
Although the exact version of the algorithm exhibited modest speed-ups in higher dimensions, we wanted to find out whether the approximate version could produce still better speed-ups. We repeated the same experiments for the approximate version of the filtering algorithm with ϵ values of 0.5, 1.0, and 2.0. The results show the expected tradeoff, that is, as ϵ increases, the running time tends to decrease while the distortion errors tend to increase. As the dimension increases, nodes are pruned with lower efficiency, and so the algorithm's running time tends to approach that of the exact algorithm. In some cases the running time of the approximate version is even higher than the exact filtering algorithm. This is because the pruning test for the approximate version is computationally more complicated than the pruning test for the exact version. As shown in Fig. 3.7(b), the approximate filtering algorithm with $\epsilon = 0.5$ is slightly faster than the exact filtering algorithm up to dimension 12. As ϵ increases, the running times improve. For $\epsilon = 1$, the approximate filtering is faster than the exact filtering algorithm up to dimension 20, and for $\epsilon = 2$, the approximate filtering is faster in all of the dimensions tested.

Of course, $\epsilon = 1$ and $\epsilon = 2$ are quite large approximation bounds (allowing for 100% and 200% errors, respectively). For this reason we computed the actual error

committed by the algorithm by comparing it with the exact version. Fig. 3.7(b) and Fig. 3.8 show that for higher values of ϵ (for example, $\epsilon = 2$) the average distortion errors are very small, while the speed-ups are quite significant. Remarkably, as the dimension increases, the distortion error becomes successively smaller. Thus, the algorithm obtains significant speed-ups in almost all the dimensions tested with very small actual distortion errors. Unfortunately, the algorithm cannot guarantee small distortion errors for all inputs, and it seems to be difficult to characterize the class of inputs for which this would be the case.

Table 3.9: Dependence on dimension for synthetic data, $n = 50,000, k_{\text{init}} = 100$

Dim	Final Centers		Avg. Distortion		CPU Seconds			Speed-up
	Std	Hyb/Fil	Std	Hyb/Fil	Std	Hyb	Fil	
2	97.0	97.0	0.007	0.007	316.38	306.74	5.35	57.31
3	98.0	98.0	0.064	0.064	360.30	352.26	9.19	38.33
4	97.8	97.8	0.210	0.210	409.12	398.98	15.36	25.97
5	95.6	95.6	0.483	0.483	426.48	432.40	26.18	16.52
6	96.0	96.0	0.887	0.887	485.88	479.02	46.97	10.20
7	93.8	93.8	1.275	1.275	517.72	510.22	77.18	6.61
8	91.4	91.4	1.985	1.985	537.64	528.96	123.80	4.27
9	92.2	92.2	2.531	2.531	593.90	586.20	179.70	3.26
10	87.2	87.2	2.909	2.909	588.84	580.44	213.80	2.71
12	83.8	83.8	5.208	5.208	636.58	630.06	312.14	2.02
14	84.2	84.2	6.590	6.590	717.56	710.48	359.72	1.98
16	79.4	79.4	8.660	8.660	749.32	742.48	387.52	1.92
18	73.2	73.2	10.892	10.892	753.06	746.50	408.54	1.83
20	73.0	73.0	13.278	13.278	817.90	810.50	492.86	1.64
25	68.8	68.8	19.278	19.278	936.04	931.08	616.90	1.51
30	60.6	60.6	25.650	25.650	968.98	964.26	660.56	1.46
35	54.4	54.4	30.998	30.998	1004.26	998.70	702.30	1.42



(a) (b)

Fig. 3.7: CPU times for the various algorithmic versions as a function of the dimension: (a) Standard, hybrid, and exact filtering, and (b) approximate filtering for various ϵ's.

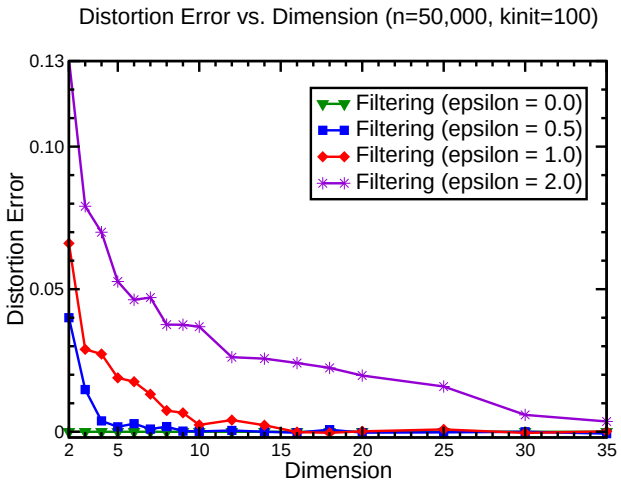


Fig. 3.8: Average distortion error (relative to standard version) for the approximate filtering algorithm (with various ϵ's) as a function of the dimension.

3.5 Average Distance and Average Distortion

As mentioned, our use of the square root of the average distortion as a measure of cluster dispersion is different from the average distance used in the standard ISOCLUS algorithm. Our experiments suggest that this modification does not make a significant difference in the quality of the resulting clustering. ISOCLUS uses the value of Δ_j in determining whether or not to split a cluster in Step 8. In particular, the jth cluster is split if $\Delta_j > \Delta$. Thus, it would be of interest to establish the conditions for which the following equivalence holds:

$$\Delta_j > \Delta \text{ (in standard ISOCLUS)} \quad \Longleftrightarrow \quad \Delta'_j > \Delta' \text{ (in filtering ISOCLUS).}$$

This raises an important question as to whether our modification is justifiable, in some sense. To further motivate this question, note that there are other reasonable generalizations of the dispersion that could produce substantially different results.

Had we not considered the square root of the distortion, large distortions would have had a disproportionately greater influence on the average dispersion, which would have resulted in different clusters being split in Step 8 of the algorithm. To see this, consider the following simple 1-dimensional example. We are given three well-separated clusters, each consisting of an equal number of points. The points are drawn from three normal distributions of standard deviations 1, 6, and 9, respectively. Suppose further that the algorithm places three centers, one at the mean of each cluster. If the number of points is large, then the three average Euclidean distances, as computed by the standard version of ISOCLUS, would be close to 1, 6, and 9, respectively. Thus, the overall average would be roughly $\Delta =$

$(1 + 6 + 9)/3 \approx 5.333$, implying that the two clusters with standard deviations of 6 and 9 would be eligible for a split in Step 8 of the algorithm. If squared distances were used instead, however, then the average of the squared distances for each cluster would be very close to 1, 36, and 81, respectively. The overall average would then be $(1 + 36 + 81)/3 \approx 39.333$, implying that only the cluster of standard deviation 9 would be eligible for a split.

An alternative approach involves taking the square root of the average distortion for each cluster (as we do in the filtering algorithm), and then taking the overall average dispersion as the square root of the weighted average of the squared distortions over all the clusters. (This is in contrast to the filtering algorithm, which takes square roots before averaging.) However, this alternative suffers from the same problem as the above approach.

Although it does not seem to be possible to make any worst-case theoretical assertions about the similarity between the results of the standard ISOCLUS algorithm and our modified version, we will endeavor to show that, in the limit, the approach taken in the filtering algorithm does not suffer from the biases of the above alternatives. Our analysis is based on the statistical assumption that points are drawn independently from a number of well separated cluster distributions that are identical up to translation and uniform scaling. This assumption is satisfied in the above examples, where the alternative definitions are shown to fail.

More specifically, we assume that the point set S is drawn from k distinct cluster distributions in $\mathbb{R}^d$. We assume that all the cluster distributions are statistically identical up to a translation and uniform scaling. In particular, let $f(x_1, \ldots, x_d)$ be

a d-variate probability density function [48] of the *base cluster distribution*, and let $\mathbf{X}$ denote a random vector sampled from this distribution. Without loss of generality, we may assume that its expected value, $E[\mathbf{X}]$, is the origin. Let $Y = \|\mathbf{X}\|$ be a random variable whose value is the Euclidean length of a vector drawn from this distribution. For the purposes of our analysis, we do not need to make any more specific assumptions about the base distribution. For example, the distribution could be a Gaussian distribution centered about the origin with an arbitrary covariance matrix.

For $1 \leq j \leq k$, we assume that the points of the jth cluster are sampled from a distribution that arises by uniformly scaling all the coordinates of $\mathbf{X}$ by some positive scale factor $a_i \in \mathbb{R}^+$ and translated by some vector $\mathbf{t}_j \in \mathbb{R}^d$. Thus, a point of the jth cluster is generated by a random vector $\mathbf{X}_j = a_i\mathbf{X} + \mathbf{t}_j$. Since the origin is the mean of the base distribution, $\mathbf{t}_j$ is the mean of the jth cluster, which we will call the *distribution center*. Let $Y_j = \|\mathbf{X}_j - \mathbf{t}_j\|$ be the random variable that represents the Euclidean distance from a point of the jth cluster to $\mathbf{t}_j$. Because this is a uniform scaling of the base distribution by a_i and translation by $\mathbf{t}_j$, it is easily verified that $E[Y_j] = a_i E[Y]$ and $E[Y_j^2] = a_i^2 E[Y^2]$.

We make the following additional assumptions about the clusters and the current state of the algorithm's execution:

(1) The clusters are well-separated, that is, the probability that a point belonging to one cluster is closer to the center of another cluster than to its own cluster center is negligible.

(2) The number of points n_j in each cluster is sufficiently large, that is, the law of large numbers can be applied to each cluster. (We do not assume that the clusters have equal numbers of points.)

(3) The algorithm is near convergence, in the sense that the difference between the current location of cluster center $\mathbf{z}_j$ and the actual cluster center $\mathbf{t}_j$ is negligible.

Theorem 3.5.1 *Subject to Assumptions* (1)–(3) *above, standard* ISOCLUS *and the filtering variant behave identically.*

Proof: As mentioned earlier, the only differences between the two algorithms are in the computations of the individual and average cluster dispersion in Step 5 and their use in determining whether to split a cluster in Step 8. Consider a cluster center j, for $1 \leq j \leq k$. Recall that to establish the equivalence of the two algorithms it suffices to show that

$$\Delta_j > \Delta \text{ (in standard ISOCLUS)} \quad \Longleftrightarrow \quad \Delta'_j > \Delta' \text{ (in filtering ISOCLUS)}.$$

First let us consider the average Euclidean distance of the standard algorithm. Recall that n_j denotes the number of points in a cluster. From the definitions of the cluster distributions and Assumption (3) we have

$$\Delta_j \;=\; \frac{1}{n_j} \sum_{\mathbf{x} \in S_j} \|\mathbf{x} - \mathbf{z}_j\| \;\approx\; \frac{1}{n_j} \sum_{\mathbf{x} \in S_j} \|\mathbf{x} - \mathbf{t}_j\|,$$

where $\approx$ denotes approximate equality (subject to the degree to which Assumption (3) is satisfied).

S_j consists of the points that are closer to $\mathbf{z}_j$ than to any other cluster center. By Assumptions (1) and (3) it follows that the contribution to the dispersion of S_j that arises due to points from other clusters is negligible. From Assumption (2) it follows from the law of large numbers that this quantity, which is just a sample mean of a large number of independent and identically distributed random variables, will be arbitrarily close to the expected value for the cluster distribution. Thus we have

$$\Delta_j \;\approx\; E[\,\|\mathbf{X_j} - \mathbf{t}_j\|\,] \;=\; E[Y_j] \;=\; a_j E[Y].$$

Next, consider the average squared distance of the filtering algorithm. From Assumption (3), the corresponding quantity in this case is

$$\Delta'_j \;=\; \sqrt{\Delta_j^{(2)}} \;=\; \left(\frac{1}{n_j}\sum_{\mathbf{x}\in S_j}\|\mathbf{x}-\mathbf{z}_j\|^2\right)^{1/2} \;\approx\; \left(\frac{1}{n_j}\sum_{\mathbf{x}\in S_j}\|\mathbf{x}-\mathbf{t}_j\|^2\right)^{1/2}.$$

As before, from our assumptions we may approximate this sample mean with the expected value for the cluster distribution, from which we obtain

$$\Delta'_j \;\approx\; \sqrt{E[\,\|\mathbf{X}_j - \mathbf{t}_j\|^2\,]} \;=\; \sqrt{E[Y_j^2]} \;=\; \sqrt{a_j^2 E[Y^2]} \;=\; a_j\sqrt{E[Y^2]}.$$

Now, let us consider the average dispersions computed by the two algorithms. Let $w_j = n_j/n$ denote the fraction of points of S that are in cluster S_j. By the definitions of Δ and Δ' we have

$$\Delta \;=\; \frac{1}{n}\sum_{i=1}^{k} n_i\Delta_i \;=\; \sum_{i=1}^{k} w_i\Delta_i \qquad \text{and} \qquad \Delta' \;=\; \frac{1}{n}\sum_{i=1}^{k} n_i\Delta'_i \;=\; \sum_{i=1}^{k} w_i\Delta'_i.$$

Finally, we combine all of this to obtain the desired conclusion. Observe that the implications are not absolute, but hold in the limit as Assumptions (1)–(3) are

satisfied:

$$\begin{array}{ccccccc}
\Delta_j > \Delta & \iff & \Delta_j > \sum_{i=1}^{k} w_i \Delta_i & \iff & a_j E[Y] > \sum_{i=1}^{k} w_i a_i E[Y] \\
 & & & & \Updownarrow & & \\
 & & & & a_j > \sum_{i=1}^{k} w_i a_i & & \\
 & & & & \Updownarrow & & \\
\Delta'_j > \Delta' & \iff & \Delta'_j > \sum_{i=1}^{k} w_i \Delta'_i & \iff & a_j \sqrt{E[Y^2]} > \sum_{i=1}^{k} w_i a_i \sqrt{E[Y^2]} &
\end{array}$$

This completes the proof. □

3.6 Summary

We have demonstrated the efficiency of a new implementation of the ISOCLUS algorithm, based on the use of the kd-tree data structure and the filtering algorithm. Our algorithm is a slight modification of the original ISOCLUS algorithm, because it uses squared distances, rather than Euclidean distances as a measure of cluster dispersion in determining whether to split clusters. We have provided both theoretical and experimental justification that the use of squared distances yields essentially the same results. The experiments on synthetic clustered data showed speed-ups in running times ranging from 1.3 to 57, while the experiments on Landsat and MODIS satellite image data showed speed-ups of 4 to 30 and 4 to 22, respectively.

We also presented an approximate version of the algorithm which allows the user to further improve the running time at the expense of lower fidelity in computing the nearest cluster center to each point. We showed that with relatively small distortion errors, significant additional speed-ups can be achieved by this ap-

proximate version. The software is freely available, and can be downloaded from `http://www.cs.umd.edu/~mount/Projects/ISODATA`.

One possible direction for future research involves sensitivity to the input parameters. The running times for the standard and hybrid versions increase linearly with the number of points n, the number of centers k, and the dimension d. For the inputs we tested, however, the running time of the filtering version increases sublinearly in n and k, but superlinearly in the dimension d. Thus, the filtering version is most appropriate when n and k are large and the dimension is fairly small.[1]

[1]The results of this chapter are based on joint work with David M. Mount, Nathan S. Netanyahu, and Jacqueline Le Moigne [110]. We would like to thank Jeffery Morisette of NASA/GSFC and the EOS Validation Core Sites project for providing us with the MODIS data, and In-Joon Chu for his contributions to the geometrical analysis used in approximate filtering.

Chapter 4

Survey of Interpolation Methods

Interpolation is an important computational tool, which is widely used throughout science and engineering. Interpolation is a method of estimating new intermediate values or points from a discrete set of surrounding known values or points [159]. In this chapter we will present a survey of a number of interpolation methods. Our focus is mainly on interpolation methods used for scattered data points. These interpolation methods include Shepard's interpolation [149], Nadaraya-Watson estimator [116, 117], moving least squares [97], natural neighbors [152], Gaussian processes [133], and kriging [80]. Many interpolation methods for scattered-data utilize spline functions [19, 75, 101, 163], radial basis functions [22], and triangulation methods [10, 34]. We discuss few of these interpolation methods below.

4.1 Shepard's Interpolation Methods

Shepard's interpolation methods, also known as inverse distance weighted methods, were initially introduced by Donald Shepard in 1968 [149]. The simplest interpretation of this method is that of a weighted inverse distance interpolation:

$$f(P) = \sum_{i=1}^{n} w_i f_i,$$

where n is the number of scattered data points in the set, the f_i are function values at known scattered data points, and the w_i are the weights assigned to each scatter

point. The *classical* form of the weight function is

$$w_i = \frac{h_i^{-p}}{\sum_{j=1}^{n} h_j^{-p}},$$

where p is an arbitrary positive real number (power parameter), and h_i is the distance from the scatter point to the interpolation point. Since Shepard's method is a global technique, the above weight assignments allow too much influence by distant points [10]. Franke and Nielson showed that the following inverse distance relative weight function yields smoother results while making Shepard's interpolation a local method so that points outside a radius, R, will not have any effect on the interpolation results [52].

$$w_i = \frac{\left[\frac{R-h_i}{R \cdot h_i}\right]^2}{\sum_{j=1}^{n} \left[\frac{R-h_j}{R \cdot h_j}\right]^2}.$$

In his paper [149] Shepard observed that the shortcomings of this method include the following:

- High computational costs for large n.
- Dependance of the interpolant value only on its distance from the query point.
- Representation of an arbitrary and undesirable constraint on the interpolated surface.

He also suggests approaches one can take to alleviate these problems by suggesting ways of selecting nearby points, incorporating direction in function evaluations, and so on.

Renka developed a modified quadratic power Shepard's method. Without sacrificing the advantages of Shepard's method, it also has accuracy comparable to

other local methods and improved computational efficiency [139, 140]. This computational efficiency is achieved by using a cell method for nearest neighbor searching. While this method is less efficient and less accurate than some triangle-based methods, it is a better candidate for extensions to three or more independent variables because triangulation-based methods have greater complexity and storage requirements. Note that Shepard's method requires $O(n)$ storage as opposed to $O(n^2)$ for tetrahedron-based methods based on 3-dimensional Delaunay triangulations. Also, the linear preprocessing time of Shepard's method is faster than triangulation time of $O(n \log n)$. The main advantage of his modified version is that data is fit to a function of three or more independent variables. Later, Renka presented a slightly different implementation of his algorithm, which achieves cubic precision and second derivative continuity at little additional cost [141].

4.2 Cubic Hermite Interpolation Problem

Given n triplets $(x_1, y_1, s_1) \ldots (x_n, y_n, s_n)$, the goal of this interpolation method is to find a function $C(x)$ that is piecewise cubic on partition $x_1 \ldots x_n$ such that for $i = 1 \ldots n$ we have $C(x_i) = y_i$ and $C'(x_i) = s_i$ [101].

Interpolating given the data and requiring continuity of the first derivative imposes $(3n-4)$ constraints (n constrains for passing through data points and $2n-4$ constraints for continuity of first derivatives) [75, 101]. However, the total number of parameters that need to be calculated is $4(n-1)$ since there are $n-1$ intervals and each cubic function requires 4 parameters. Thus, the Hermite interpolation

problem still leaves n other free parameters. These n parameters can be calculated by imposing other constraints such as those due to convexity or monotonicity [75].

4.3 Spline Interpolants

A *spline* is a piecewise polynomial function of degree k that is continuously differentiable $k-1$ times [75]. For example, a linear spline is a piecewise-linear function that is continuous but not differentiable. A spline can have a very simple form locally but be very flexible and smooth globally. For this reason, splines are widely used for interpolation of data. Two of the most commonly used class of spline functions are *thin-plate splines* [163] and *polyharmonic splines* [19].

Given n triples $(x_1, y_1, s_1) \ldots (x_n, y_n, s_n)$, the goal of the *cubic spline interpolants* is to obtain a function $S(z)$ with the property that S, S', and S'' are continuous [101]. The requirements that the function and its first derivative are continuous imposes $3n-4$ constraints. Continuity of the second derivatives imposes $n-2$ additional constraints, giving a total of $4n-6$ constraints. We are left with two free parameters mainly due to having one less constraint on the end points of our data's interval (x_1 and x_n). Depending on what the two additional constraints are, different interpolation methods are defined. For example, the *complete spline interpolant* constrains the first derivative at the two end points by imposing $S'(x_1) = \mu_1$ and $S'(x_n) = \mu_2$, where μ_1 and μ_2 are the required end point values for the first derivative. Similarly, *natural splines* require the second derivatives at the two end points be constrained, and the *not-a-knot spline* interpolant requires third derivative

continuity at x_2 and x_{n-1}.

4.4 Natural Neighbors

Natural neighbors is a local interpolation method that was first introduced by Sibson [152]. Natural neighbors interpolation uses a weighted averaging method, where weights are area-based rather than distance-based. Area-based methods are more robust and perform equally well in clustered and sparse areas of given data set. However, they are more computationally intensive than other methods. Natural neighbors interpolation requires a preprocessing step that constructs the *Voronoi diagram* of the given point set [10]. The Voronoi diagram (also called the Dirichlet or Thiessen tessellation) partitions the plane into disjoint tiles; (or polytopes) where each tile T_i encloses one point x_i of the given point set, and the area of T_i is closer to the data point x_i than to any other data point.

Suppose we have n scatter data points $x_1 \dots x_n$. After the tessellation is constructed, we add the query point x_q to the data set and update the Voronoi diagram. Let's call the Voronoi cell of x_q, $T(x_q)$, and its intersections with the old Voronoi cells/tiles, $T_i(x_q)$, that is $T_i(x_q) = T(x_q) \cap T_i$. Note that intersection $T_i(x_q)$ is nonempty only for neighboring cells from which $T(x_q)$ has inherited its area. The actual calculations of these intersecting areas is a complicated but reasonably efficient geometric calculation. See [152] for more details. The natural neighbor interpolant at point x_q can then defined as

$$f(x_q) = \sum_i w_i(x_q) z_i, \tag{4.1}$$

where

$$\begin{aligned} w_i(x_q) &= \frac{Area[T_i(x_q)]}{Area[T(x_q)]}, \\ \sum_i w_i(x_q) &= 1, \\ 0 \leq w_i(x_q) &\leq 1. \end{aligned} \tag{4.2}$$

This method results in cone-like peaks at data points x_i which are natural neighbors of x_q. This means that the resulting interpolant is only C^0 continuous. Sibson [152] obtained a C^1 continuous surface by taking into account the gradient at the data points. That is, in equation 4.1, he replaced z_i by a first degree polynomial $g_i(x_q)$ where

$$g_i(x_q) = z_i + \nabla z(x_i)^T (x_q - x_i). \tag{4.3}$$

In order to guarantee the correct slopes at the points x_i, weights w_i are also replaced by the weights

$$w_i'(x_q) = \frac{w_i(x_q)\|x_q - x_i\|^{-1}}{\sum_i w_i(x_q)\|x_q - x_i\|^{-1}}. \tag{4.4}$$

Thus, Eq. 4.1 is modified to the following interpolant:

$$f(x_q) = \sum_i w_i'(x_q) g_i(x_q), \tag{4.5}$$

Note that if we replace $w_i'(x_q)$ by $\|x_q - x_i\|^{-1}$ we obtain a linearly precise version of Shepard's method [6]. Later, Farin [46] extended Sibson's work by giving a continuous parametrization of the Sibson's coordinates. He presented a C^1 continuous interpolation method based on Bernstein-Bézier techniques which exactly interpolates quadratic functions, assuming that the function gradient is known [46].

4.5 Triangulation (Tetrahedrization) Based Methods

The idea for this interpolation technique is based on *Delaunay triangulation* of points. Delaunay triangulation is a dual structure of the Voronoi diagram in which sites located at each three adjacent cells are connected and form a triangle.

Once a Delaunay triangulation of points is constructed, one can interpolate value at a query point q in two steps.

1. Find the triangle to which the query point belongs. This can be done through the following search procedure [10]:

   - Let p_i be a point belonging to a known tetrahedron T.
   - q belongs to the same tetrahedron if there is no intersection between segment pq and any surface of T. If this holds we have found the tetrahedron to which q belongs.
   - If we had not found the tetrahedron, a new point p_{i+1} is chosen in the tetrahedron adjacent to T, which shares with T its intersected surface. The procedure is repeated until the correct tetrahedron is found.

2. Use a linear, linear multi-valued triangular, or cubic triangular interpolation method to come up with the value for the query point. For details of each of these methods please see [10].

Triangulation based methods are local and thus capable of handling large data sets. Their only drawback is the need for calculating the triangulation. Once triangulation of points is calculated, interpolation method is very simple.

4.6 Moving Least Squares

Moving Least Squares (MLS) was first introduced in [97]. Before defining MLS, we need to first know what Least Squares (LS) and Weighted Least Squares (WLS) are.

Given n points at locations x_i in $\Re^d$ where $i \in \{1 \dots n\}$, the goal of LS problem is to find a global function $f(x)$ such that $f(x)$ approximates values f_i at points x_i such that it minimizes sum of squared errors, E. That is

$$f(x) = \min_{f \in \prod_m^d} \sum_i \parallel f(x_i) - f_i \parallel^2 .$$

In the WLS formulation, $f(x)$ is calculated as the solution to the minimization of sum of weighted squared errors. Weights are usually selected as a function,θ, of distances of points x_i to a fixed point $\overline{x}$. Thus, $f(x)$ is calculated as follows:

$$f(x) = \min_{f \in \prod_m^d} \sum_i \theta(\parallel \overline{x} - x_i \parallel) \parallel f(x_i) - f_i \parallel^2 .$$

In each case, partial derivatives of the error function we are minimizing with respect to the unknown coefficients are taken, set to zero, and solved for unknowns. In MLS formulation, one starts with the weighted least squares formulation for a fixed point in $\Re^d$. Then, this point is *moved* over the entire parameter domain, where a weighted least squares fit is evaluated for each point individually. Thus,

$$f(x) = f_x(x), \min_{f_x \in \prod_m^d} \sum_i \theta(\parallel x - x_i \parallel) \parallel f(x_i) - f_i \parallel^2 .$$

It can be shown that the global function $f(x)$ is continuously differentiable if and only if the weighting function is continuously differentiable [98].

4.7 Gaussian Process Regression

Gaussian process (GP) is defined as collection of random variables, any finite number of which have a joint Gaussian distribution [133]. Thus, Gaussian process is completely specified by its mean function and covariance function. Let $m(x)$ be the mean function of random variables (real process) $f(x)$, and $k(x, x')$ be the covariance function of a real process $f(x)$, then the Gaussian process $f(x)$ is represented as $f(x) \sim GP(m(x), k(x, x'))$, where

$$m(x) = E[f(x)]$$

and

$$k(\mathbf{x}, \mathbf{x}') = E[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))].$$

Thus, learning a Gaussian process is exactly the problem of finding suitable properties for the covariance function to produce a model of the data.

Regression is concerned with the prediction of continuous quantities. We focus on application of Gaussian Processes for regression problems [133]. In order to understand how GP is used for linear regression, we first review concepts of linear regression model and Bayesian inference.

A standard linear regression model with noise can be modeled by $f(\mathbf{x}) = \mathbf{x^T w}$ and $y = f(\mathbf{x}) + \epsilon$. In this model, the vector $\mathbf{x}$ is the input vector, $\mathbf{w}$ is a vector of weights (parameters) of the linear model, and f is the actual function value, y is the observed target value, and the observed values differ from the function values by an additive noise ϵ. Bayesian inference is based on maximizing a probability distribution. For regression, this probability is the posterior distribution over weights,

$P(\mathbf{w}|\mathbf{x}, y)$. This probability is calculated using Bayes' rule as follows.

$$P(\mathbf{w}|\mathbf{x}, y) = \frac{P(y|\mathbf{x}, \mathbf{w})P(\mathbf{w})}{P(y|\mathbf{x})}.$$

Performing regression using the above approach has the disadvantage that one needs to restrict the estimation to a particular class of functions. Another possible approach is to consider all possible functions and calculate the probabilities in the function space. Initially it appears that since we are dealing with an infinite number of possible functions, it would be impossible to calculate prior probabilities of every possible function. This is where we take advantage of the GP. A GP is generalization of a Gaussian distribution. A Gaussian distribution describes properties of data and variables whereas a GP governs properties of functions. One can think of a function as an infinite vector where each element of it is the function value $f(x)$ for a particular input x.

To use GP in Bayesian inference, inputs are projected into a high dimensional space. In particular, let $\phi(\mathbf{x})$ be the function that maps vector $\mathbf{x}$ from an D dimensional feature space to an N dimensional space. Then, the linear regression model will have the form $f(\mathbf{x}) = \phi(\mathbf{x})^T\mathbf{w}$. Let X be the aggregated n input column vectors in a matrix. Predictive distribution for f_*, $f(x_*)$, is given by averaging over the output of all possible linear models with respect to the Gaussian posterior $p(f_*|x_*, X, y)$. This Gaussian posterior is Gaussian as well.

It can be shown that $\mathbf{y} \sim N(\mathbf{0}, K + \sigma_n^2 I)$ where K is the covariance matrix of input X, and σ_n^2 is the noise variance to the predictive variance of f_* [133]. Given a value x_* one can calculate predictive mean $\overline{f}_*(x_*)$ as $\overline{f}_*(x_*) = k_*^T\alpha$, where $k_* = k(x_*)$

denotes the vector of covariances between the query point and n input data. α is calculated as $\alpha = (K + \sigma_n^2 I)^{-1}\mathbf{y}$. Another way of representing the same result is that $\overline{f}_*(x_*) = \sum_{i=1}^{n} \alpha_i k(x_i, x_*)$. For details of these derivations please see [133].

Note that in order to predict $f_*(x_*)$ one needs to solve a symmetric linear system to calculate α vector as mentioned above. For large input data, representing and solving this linear system can be very expensive (both in computation and memory).

In summary, a Gaussian process can be used as a prior probability distribution over functions in Bayesian inference. Inference of continuous values with a Gaussian process prior is known as Gaussian process regression, or *Kriging*. Gaussian process prediction is well known in the geostatistics field as kriging [85], which we will discuss in Section 4.8.

4.8 Kriging/Cokriging Interpolation Methods

Interpolating noisy or scattered data points is a problem of wide ranging interest, where gathering data at specific locations is either impractical or expensive. Because of the high cost of collecting measurements, high accuracy is required in the interpolants. A popular class of interpolation methods used in these situations is *kriging*. Kriging is a class of interpolation methods named after Danie Krige, a South African mining engineer, who pioneered in the field of geostatistics [60]. Kriging is also referred to as the *Gaussian process predictor* in the machine learning domain [133]. Kriging and its variants have been traditionally used in mining and

geostatistics applications [60, 80, 85]. The most commonly used variant is called *ordinary kriging*, which is often referred to as a BLUE method, that is a Best Linear Unbiased Estimator [60, 80, 85]. Ordinary kriging is considered to be *best* because it minimizes the variance of the estimation error. It is *linear* because estimates are weighted linear combination of available data, and is *unbiased* since it aims to have the mean error equal to zero [80]. A more generalized technique used for geostatistics and mining applications is *cokriging* [60, 80, 85]. A special characteristic of cokriging is that it can utilize data of different nature to model and interpolate a particular attribute [37, 60, 80].

As mentioned before, kriging is a Gaussian process predictor. In fact, kriging equations are equal to those obtained for GP if we do not assume any noise associated with our observations. However, we present equations from a geostatistics perspective rather than from a Bayesian inference point of view.

Cokriging is a generalization of kriging that minimizes the variance of the estimation error by taking into consideration the spatial correlations between the variables of interest and the secondary variables. In other words, a function U at location 0 is estimated as a *linear combination* of both the variable of interest and the secondary variable(s). That is, in the case where we have one secondary variable, the estimate of U at location 0, $\hat{u}_0$, using the two sets of variables as mentioned in [80], is given by $\hat{u}_0 = \sum_{i=1}^{n} a_i u_i + \sum_{j=1}^{m} b_j v_j$, where $u_1, u_2, \ldots, u_n$ are primary data at n nearby locations, $v_1, v_2, \ldots, v_n$ are secondary data at m nearby locations, and $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$ are cokriging weights which are needed to be found and calculated. The estimation error, R, is calculated as $R = \widehat{U}_0 - U_0 = w^T Z$, where

$w^T = (a_1, \ldots, a_n, b_1, \ldots, b_m, -1)$, and $Z^T = (U_1, \ldots, U_i, V_1, \ldots, V_m, U_0)$. The goal of cokriging is to find the weights $a_1, \ldots a_n$ and $b_1 \ldots b_m$ such that the variance of the error is minimized and the estimate for $\widehat{U}_0$ is *unbiased*, that is, the mean error residual is zero. In Appendix A we show conditions under which these requirements are satisfied. We also present necessary mathematical and geostatistical background needed to understand the kriging problem. We then derive mathematical formalization and algorithmic approaches for solving the kriging problem. We also introduce available software for kriging and mention this problem's computational issues in more detail. In summary, in Appendix A, we show that the generalized cokriging system is the following.

$$\begin{pmatrix} C & L \\ L^T & 0 \end{pmatrix} \begin{pmatrix} w \\ \mu \end{pmatrix} = \begin{pmatrix} C0 \\ I0 \end{pmatrix}, \tag{4.6}$$

where C is the pairwise covariances of all known variable values, and C_0 is the vector of pairwise covariances between the unknown variable U_0 and all other known variables. The μ entry is a vector of all Lagrange multipliers $\mu_1 \ldots \mu_s$. L is a vector of matrices $I_1 \ldots I_s$. Each matrix $I_i, i \in \{1 \ldots s\}$ is of size $(n_i \times s)$, where n_i is the number of points in the i^{th} variable set. All elements in the i^{th} column of I_i are one and all other entries are zero. The w entry is the vector of all coefficients, and I_0 is a column vector of size $s \times 1$ of all elements under C_0 on the right hand side of the equation. Similarly to ensure unbiasedness, this vector is made of a 1 on top and all zeros for the rest of entries. It can also be proven that in order for the above system to have a solution, we need C to be positive definite [28, 115] (see Appendix A for details). Note that in all the above mentioned equations, if we had only one type

of variable U, at n different locations, the problem reduces to the *ordinary kriging* problem. In this case, L would be a column of all 1s, and we solve for one set of coefficients $a_1, \ldots a_n, \mu_1$.

Thus, for a data set of size n the kriging problem involves solving a system of order $O(n \times n)$. For large data sets, solving this system using traditional methods such as Gaussian Elimination is impractical. We explored two approaches for efficiently solving these systems for very large spatial data sets. One is through introducing sparsity to the system, and the other is based on utilizing *Fast Multipole Method* (FMM) [61]. In the next two chapters, we describe each of these approaches along with computational issues involved in each case, our work, and results. For background information and details of the kriging interpolation methods please see Appendix A.

Chapter 5

Kriging via Covariance Tapering and Iterative Methods

Scattered data interpolation is a problem of interest in numerous areas such as electronic imaging, smooth surface modeling, and computational geometry [6, 10]. Our motivation arises from applications in geology and mining. In many instances data can be costly to compute and are available only at nonuniformly scattered positions. Because of the high cost of collecting measurements, high accuracy is required in the interpolants. The method of choice, sometimes called the "gold standard" in this area [111], is *ordinary kriging* (please see Section 4.8 and Appendix A for more details). This is because it is a best linear unbiased estimator [60, 80, 85]. The price for its statistical optimality is that the estimator is computationally very expensive. For n scattered data points, computing the value of a single interpolant involves solving a dense linear system of size roughly $n \times n$. This is infeasible for large n. In practice, kriging is solved approximately by local approaches that are based on considering only a relatively small number of points that lie close to the query point [60, 80]. There are many problems with this local approach, however. The first is that determining the proper neighborhood size is tricky, and is usually solved by *ad hoc* methods such as selecting a fixed number of nearest neighbors or all the points lying within a fixed radius. Such fixed neighborhood sizes may not work well for all query points, depending on local density of the point distribution [60]. Local

methods also suffer from the problem that the resulting interpolant is not continuous. Meyer showed that while kriging produces smooth continuous surfaces, it has zero order continuity along its borders [111]. Thus, at interface boundaries where the neighborhood changes, the interpolant behaves discontinuously. Therefore, it is important to consider and solve the global system for each interpolant. However, solving such large dense systems for each query point is impractical.

Recently a more principled approach to approximating kriging has been proposed based on a technique called *covariance tapering* [55]. We will discuss its computational issues in greater detail, but the problems arise from the fact that the covariance functions that are used in kriging have global support. In tapering these functions are approximated by functions that have only local support, and that possess certain necessary mathematical properties. This achieves greater efficiency by replacing large dense kriging systems with much sparser linear systems. Covariance tapering has been successfully applied to a restriction of our problem, called *simple kriging* [55]. Simple kriging is not an unbiased estimator for stationary data whose mean value differs from zero, however. In this chapter we generalize these results by showing how to apply covariance tapering to the more general problem of ordinary kriging. Ordinary kriging ensures unbiasedness for stationary data, whose mean can have any value.

Our implementations combine, use, and enhance a number of different approaches that have been introduced in literature for solving large linear systems for interpolation of scattered data points. For very large systems, exact methods such as Gaussian elimination are impractical since they require $O(n^3)$ time and $O(n^2)$

storage. As Billings *et al.* [22] suggested, we use an iterative approach. In particular we use the SYMMLQ method [122] for solving the large but sparse ordinary kriging systems that result from tapering. There are a number of technical issues that need to be overcome in our algorithmic solution. In particular, we describe in Section A.3.5 why the points' covariance matrix for kriging should be symmetric positive definite. The goal of tapering is to obtain a sparse approximate representation of the covariance matrix while maintaining its positive definiteness. Furrer *et al.* [55] used tapering to obtain a sparse linear system of the form $Ax = b$, where A is the tapered symmetric positive definite covariance matrix. Thus, Cholesky factorization [101] could be used to solve the linear system for their application. They further utilized the sparseness of A to implement an efficient sparse Cholesky decomposition method for solving the linear system. They also applied the Fast Fourier Transform in conjunction with their sparse implementation to obtain better efficiency in solving the system. In addition, they show if these tapers are used for a limited class of covariance models, the solution of the system converges to the solution of the original system. While their results show significant improvements over dense Cholesky factorization, their approach is not applicable to the ordinary kriging problem. This is mainly due to the fact that matrix A in the ordinary kriging linear system, while symmetric, is not positive definite. We will discuss details of the ordinary kriging system in Section A.3 of Appendix A.

In ordinary kriging, additional constraints are imposed on the interpolation coefficients to ensure the unbiasedness of the estimator. These constraints result in one or more additional rows and columns in matrix A. As we will see in Section A.3

of Appendix A, these constraints result in a matrix that fails to be positive definite. Thus, efficient implementations of Cholesky factorization (which require a positive definite matrix) are not applicable to the ordinary kriging problem. Therefore, we use tapering only to obtain a sparse linear system, and then use a sparse iterative method to solve our linear systems. In particular, we use SYMMLQ method which is an iterative method for solving symmetric but not positive definite systems [122].

We show that solving large kriging systems becomes practical via tapering and iterative methods, and results in lower estimation errors compared to traditional local approaches, and significant memory savings compared to the original global system. We achieve further significant speed-ups by introducing a variant of the global tapered system. This variant is obtained by projecting our global system to an appropriate lower dimensional system. This approach can be viewed as adaptively finding the correct local neighborhood for each query point in the ordinary kriging interpolation process. We compare both quality of our results and running times with those obtained using traditional approaches based on neighborhood sizes for solving large ordinary kriging systems.

The remainder of this chapter is organized as follows. In Section 5.1.2 we describe what is meant by tapering, and the effect that it has on the ordinary kriging system (see Appendix A for details). We proceed by introducing our approaches for solving the ordinary kriging problem in Section 5.2. Section 5.3 describes data sets used in this chapter. Then, we describe our experiments and results in Section 5.4. We conclude this chapter in Section 5.5.

5.1 Tapering Covariances

Tapering covariances for the kriging interpolation problem, as described in [55], is the process of obtaining a sparse representation of the points' pairwise covariances such that positive definiteness of the covariance matrix is preserved and the solution to the approximate (tapered) system is very close to the exact system. There are two approaches for obtaining such a sparse representation. One approach is to set all sufficiently small covariance values to zero. That is, we assign zero to pairwise covariances of points that are farther than a threshold distance, θ, from each other. We refer to this approach as *truncation*. The other approach used by Furrer *et. al.* [55] is to modify the covariance function via some *tapering functions* such that the above objectives are met. Next, we will describe each of these approaches and their computational issues involved.

5.1.1 Tapering via truncation

Consider the ordinary kriging system mentioned in Eq. (A.25) in Appendix A. For very large data sets of size $O(n)$, this system is of size $O(n^2)$. For large n it would be impractical to solve the system using traditional approaches. Let τ be a user-specified small value. Instead of solving this system, we consider solving a system in which the covariance values that are sufficiently small are set to zero. That is, we replace the kriging system of Eq. (A.25) in Appendix A with the following.

$$\begin{pmatrix} \widetilde{C} & L \\ L^T & 0 \end{pmatrix} \begin{pmatrix} T \\ \mu \end{pmatrix} = \begin{pmatrix} \widetilde{C}_0 \\ I_0 \end{pmatrix}, \tag{5.1}$$

where

$$\widetilde{c}_{ij} = \begin{cases} 0 & \text{if} \quad |c_{ij}| \leq \tau, \\ c_{ij} & \text{otherwise.} \end{cases} \tag{5.2}$$

and

$$\widetilde{c}_{0_{i1}} = \begin{cases} 0 & \text{if} \quad |c_{0_{i1}}| \leq \tau, \\ c_{0_{i1}} & \text{otherwise.} \end{cases} \tag{5.3}$$

We refer to the original linear system Eq. (A.25) in Appendix A as $Ax = b$ and the modified system Eq. (5.1) as $\widetilde{A}\tilde{x} = \tilde{b}$. There are several issues that need to be addressed. In particular, the matrix $\widetilde{A}$ should be nonsingular, $\widetilde{C}$ should be positive definite, and the solution of Eq. (5.1) should be approximately equal to that of Eq. (A.25) in Appendix A. Let us consider each of these individuals. First, we would like to determine conditions under which the truncated matrix $\widetilde{A}$ is nonsingular. Let matrix E and vector f be defined as follows:

$$e_{ij} = \begin{cases} -a_{ij} & \text{if} \quad |a_{ij}| \leq \tau, \\ 0 & \text{otherwise.} \end{cases} \tag{5.4}$$

and

$$f_{i1} = \begin{cases} -b_{i1} & \text{if} \quad |b_{i1}| \leq \tau, \\ 0 & \text{otherwise.} \end{cases} \tag{5.5}$$

Our modified system in terms of E and f is $(A+E)x = (b+f)$, where $\widetilde{A} = A+E$ and $\widetilde{b} = b+f$. We know from Theorem 5 in [101] (page 185) that if a matrix A is stored as $\widetilde{A} = A + E$, where $|e_{ij}| < \tau|a_{ij}|$ then $\|E\| \leq \tau\|A\|$ (considering the Frobenius norm). Similarly, we can conclude $\|f\| \leq \tau\|b\|$ in our problem formulation. Based on the Banach Lemma [121], also mentioned in Appendix A, $A + E$ is nonsingular if $\|A^{-1}\| \cdot \|E\| < 1$. Putting these together, yields the following.

$$\|A^{-1}\| \cdot \|E\| \;\leq\; \tau\|A^{-1}\| \cdot \|A\| \;\leq\; \tau\kappa, \tag{5.6}$$

where $\kappa = \|A\| \cdot \|A^{-1}\|$. Thus, in order for $A + E$ to be nonsingular we should select the threshold for the sparse representation of the matrix to be less than $\frac{1}{\kappa}$.

The error analysis presented in Section A.6 of Appendix A applies to our truncated system as well. Considering that discussion and $\alpha = \kappa \frac{\|E\|}{\|A\|}$, we have

$$\begin{aligned}
\left(\frac{\|E\|}{\|A\|} \ \leq \ \tau\right) &\iff \left(\kappa\frac{\|E\|}{\|A\|} \ \leq \ \kappa\tau\right) \\
&\iff \left(1 - \kappa \cdot \frac{\|E\|}{\|A\|} \ \geq \ 1 - \kappa\tau\right) \\
&\iff \left(\frac{1}{1 - \kappa\frac{\|E\|}{\|A\|}} \ \leq \ \frac{1}{1-\kappa\tau}\right) \\
&\iff \left(\frac{1}{1-\alpha} \ \leq \ \frac{1}{1-\kappa\tau}\right). \qquad (5.7)
\end{aligned}$$

Putting this together with Eq. (A.37) in Appendix A, implies that the relative error of the truncated systems' solution, $\widehat{x}$, with respect to the solution of the original system, x, can be bounded as follows.

$$\begin{aligned}
\frac{\|x - \widehat{x}\|}{\|x\|} &\leq \left(\frac{\kappa}{1-\kappa\tau}(\tau + \tau)\right) \\
&= \left(\frac{2\kappa\tau}{1-\kappa\tau}\right).
\end{aligned}$$

In order to make sure that the relative error in solving the system is less than a certain threshold, ϵ, we need to select τ such that $\frac{2\kappa\tau}{1-\kappa\tau} \leq \epsilon$. In other words, the value of τ for performing truncation should satisfy

$$\tau \leq \frac{\epsilon}{\kappa(2+\epsilon)}. \qquad (5.8)$$

Finally, assuming that C is positive definite, we would like to know whether or not its tapered form via truncation, $\widetilde{C}$, is also positive definite. Before presenting our analysis of the general case, it will be helpful to first consider a special case in

which positive definiteness is preserved. We consider two data points are *correlated* with each other if their pairwise covariance value is greater than or equal to τ. We then say that data points are *well-clustered* if it is possible to partition them into disjoint sets, or *clusters*, such that points lying within the same cluster are *correlated* with each other and points that are in different clusters are not. If points are well-clustered, then the tapered matrix, $\widetilde{C}$, can be expressed in block diagonal form, $\widetilde{C}'$. In other words, there is a permutation matrix P, such that $\widetilde{C}' = P^T \widetilde{C} P$. Since we have a similarity transformation, $P^{-1} = P^T$, thus the eigenvalues of $\widetilde{C}'$ are equal to the eigenvalues of $\widetilde{C}$ [121]. Therefore, it suffices to show that the permuted and tapered block diagonal covariance matrix is positive definite. Note that each matrix block on the diagonal is positive definite since it is the covariance matrix of the set of points in one of the clusters. Then, as a result of Fischer's inequality [79] we conclude that the eigenvalues of the whole block diagonal matrix $\widetilde{C}'$ is the union of all eigenvalues of each of the block matrices. Since all the eigenvalues are positive so is their union, and thus the tapered covariance matrix is positive definite.

For the general case, the clusters may overlap and thus this block diagonal representation may not exist. The tapered matrix $\widetilde{C}$ can be written as $\widetilde{C} = C + D$, where

$$\widetilde{d}_{ij} = \begin{cases} -c_{ij} & \text{if} \quad |c_{ij}| \leq \tau, \\ 0 & \text{otherwise.} \end{cases}$$

It can be shown that if $||D||$ is smaller than the smallest eigenvalue of the matrix C, then $\widetilde{C}$ is still positive definite. To see this, observe that since C is positive definite all its eigenvalues, $\lambda_1, \ldots, \lambda_n$, are strictly positive. Suppose we have labeled the

eigenvalues in decreasing order such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$. Now, consider the eigen-decomposition of matrix C. Since C is symmetric positive definite, there exists an orthogonal matrix Q such that $C = Q\Lambda Q^T$, where Λ is a diagonal matrix with eigenvalues of C on its diagonal, and Q is an orthogonal matrix. That is, $\Lambda = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$ and $QQ^T = I$. Now, suppose that the smallest eigenvalue of C, λ_n, is perturbed by an amount δ. Let $\widetilde{\Lambda} = diag(\lambda_1, \lambda_2, \ldots, (\lambda_n - \delta))$. In this case, the perturbed matrix $\widetilde{C}$ is $Q\widetilde{\Lambda}Q^T$. Let Δ be the norm of the change in matrix C (amount of perturbation). That is,

$$\Delta = ||\widetilde{C} - C|| = ||Q(\widetilde{\Lambda} - \Lambda)Q^T|| = \lambda_n - \delta.$$

Note that if $\delta \geq \lambda_n$, then $\widetilde{C}$ is not positive definite. Thus, in order to ensure that the perturbed matrix (tapered matrix) is positive definite, it is sufficient that the selected value τ for tapering the covariance matrix in the kriging system be smaller than λ_n, its smallest eigenvalue. In fact, τ should be smaller than λ_n by a constant factor roughly equal to the number of elements being truncated.

Based on the discussion presented here, the correct choice of τ for truncation is crucial to guaranteeing that the truncated system has a solution with relative error less than a desired error threshold, ϵ, and that positive definiteness of C is preserved. In order to make such a correct choice, one is required to know the κ (the condition number of matrix A) as well as λ_n (the smallest eigenvalue of C). However, solving for these values requires solving linear systems of the same size as our original problem, which were impractical to solve exactly (which we wanted to solve efficiently via truncation). In the next section, we present an alternative

way for introducing sparsity to the kriging linear system which preserves positive definiteness of the covariance matrix C. This approach is based on using tapering functions. Some of these tapering functions guarantee convergence to the optimal solution of the system for certain classes of the covariance functions as well [55].

5.1.2 Tapering via tapering functions

In the past section, we discussed limitation of introducing sparsity to the kriging system via truncation. In particular, we discussed that selecting a valid threshold for truncation, requires knowledge of the smallest eigenvalue of the points covariance matrix, as well as the condition number of our linear system's matrix. However,estimating these values for large matrices, reduces to solving large linear systems of the same size as the original problem. Instead, the sparse representation via tapering can be obtained through the Schur (or termwise) product of the original positive definite covariance matrix by another positive definite covariance matrix.

$$C_{tap}(h) \quad = \quad C(h) \times C_{\theta}(h). \tag{5.9}$$

The resulting tapered covariance matrix, C_{tap}, has zero values for points that are more than a certain distance apart from each other. It is also positive definite since the Schur product of two positive definite matrices are positive definite. A taper is considered valid for a covariance model if it preserves its positive definiteness property and makes the approximate system's solution converge to the original system's solution.

The authors of [55] mention few valid tapering functions. They also showed

that tapers need to be as smooth as the original covariance function to ensure convergence to the original system's solution. In this chapter, we used the following tapers [55].

$$Spherical = \left(1-\frac{h}{\theta}\right)_+^2 \left(1+\frac{h}{2\theta}\right), \tag{5.10}$$

$$Wendland_1 = \left(1-\frac{h}{\theta}\right)_+^4 \left(1+\frac{4h}{\theta}\right), \tag{5.11}$$

$$Wendland_2 = \left(1-\frac{h}{\theta}\right)_+^6 \left(1+\frac{6h}{\theta}+\frac{35h^2}{3\theta^2}\right), \tag{5.12}$$

$$TopHat = \left(1-\frac{h}{\theta}\right)_+, \tag{5.13}$$

where $x_+ = \max\{0, x\}$ and θ is chosen so that pairwise covariances can be supported in $[0, \theta)$. Note that the above tapers result in positive definite covariance functions in $\mathbb{R}^3$ and lower dimensions [55]. However, considering convergence to the optimal estimator, these tapers are not valid for all covariance functions. Tapers need to be as smooth as the original covariance function at origin to guarantee convergence to the optimal estimator [55]. Thus, for a Gaussian covariance function, which is infinitely differentiable, no taper exists that satisfies this smoothness requirement. However, since tapers proposed in [55] still maintain positive definiteness of the covariance matrices, we examined using these tapers for Gaussian covariance functions as well. We used these tapers mainly to build a sparse approximate system to our original global system even though these tapers do not guarantee convergence to the optimal solution of the original global dense system theoretically. Of the above mentioned tapering functions, the Top Hat taper is closest to the truncation idea while guaranteeing positive definiteness of the covariance matrix.

5.2 Our Approaches

We implemented and examined both local and global interpolation methods for the ordinary kriging interpolation problem as follows.

Local Methods: This is the traditional and the most common way of solving kriging systems. That is, instead of considering all known values in the interpolation process, points within a neighborhood of the query point are considered. Neighborhood sizes are defined either by a *fixed number* of points closest to the query point or by points within a *fixed radius* from the query point. Therefore, the problem is solved locally. We experimented our interpolations using both of these local approaches. We defined the fixed radius to be the distance beyond which correlation values are less than 10^{-6} of the maximum correlation. Similarly, for the fixed number approach, we used maximum connectivity degree of points' pairwise covariances, when covariance values are larger than 10^{-6} of the maximum covariance value. Gaussian elimination [118] was used for solving the local linear systems in both cases.

Global Tapered Methods: In global tapered methods we first redefine our points' covariance function to be the tapered covariance function obtained through Eq. (5.9), where $C(h)$ is the covariance function which was used (Eq. (A.13) or (A.14) in Appendix A), and $C_\theta(h)$ is one of the tapering functions defined in Section 5.1.2. We then solve the linear system using the SYMMLQ approach as mentioned in [122]. Note that, while one can use conjugate gradient method for solving symmetric sys-

tems, the method is guaranteed to converge only when the coefficient matrix is both symmetric and positive definite [150]. Since ordinary kriging systems are symmetric and not positive definite, we used SYMMLQ. We modified the original SYMMLQ implementation to take advantage of the sparseness of the matrix A, similar to the sparse conjugate gradient implementation mentioned in [131]. Note that in [131]'s implementation, matrix elements that are less than or equal to a *threshold* value are ignored. Since we obtain sparseness through tapering, this *threshold* value for our application is zero. One appealing approach would be to obtain a sparse system by having a small nonzero *threshold* value, instead of obtaining sparseness through tapering. However, as mentioned before, this approach does not necessarily result in a positive definite covariance matrix, and that is the main reason why tapering functions are of great value for kriging applications [55].

Global Tapered and Projected Methods: This implementation is motivated by numerous empirical results in geostatistics which show that interpolation weights associated with points that are very far from the query point tend to be very close to zero. That is, very far points do not seem to contribute much to the interpolation weights. This phenomenon is called the *screening effect* in the geostatistical literature [155]. Stein showed conditions under which the screening effect occurs for gridded data [155]. While the existence of the screening effect has been the basis for using local methods in the past, there is no proof of this empirically supported idea for scattered data points [55]. We use this conjecture for solving the global ordinary kriging system $Ax = b$, observing that many elements of b are zero after

tapering. This indicates that for each zero element b_i , representing the covariance between the query point and the i^{th} data point, we have $C_{i0} = 0$. Thus, we expect their associated interpolation weight, w_i, to be very close to zero. We assign zero to such w_i's, and consider solving a smaller system $A'x' = b'$, where b' consists of nonzero entries of b. We store indices of nonzero rows in b in a vector called *indices*. A' contains only those elements of A whose row and column indices both appear in the *indices* vector. Then, we solve the projected system $A'x' = b'$. This method is effectively the same as the fixed radius neighborhood size. The difference is that the local neighborhood is found adaptively by looking at covariance values in the global system for each query point. There are several differences between this approach and the local methods. One is that we build the global matrix A once, and use relevant parts of it, contributing to nonzero weights, for each query point. Second, for each query point, the local neighborhood is found adaptively by looking at covariance values in the global system. Third, the covariance values are modified through tapering.

5.3 Data Sets

We need large scattered data sets to test and evaluate performance of various approaches mentioned in Section 5.2. As mentioned before, we cannot solve the original global systems exactly for very large data sets, and thus cannot compare our solutions with respect to the original global systems. Therefore, we would need ground truth values for our data sets. Also, since performance of local approaches

can depend on data points' density around the query point, we would like our data sets to be scattered non-uniformly. Therefore, we create our scattered data sets by sampling points of a large dense grid from both uniform and Gaussian distributions. Values of the dense grid are either synthetically generated or are real measurements.

We generated our synthetic data sets using the sgems [138] software. We generated values on a (1000×1000) grid. Values were generated using the Sequential Gaussian Simulation (sgsim) algorithm of the sgems software (please see [137, 138] for more details). Points were simulated through ordinary kriging with a Gaussian covariance function Eq. (A.13) in Appendix A of range equal to 12. Each point was simulated using a maximum of 400 neighboring points within a 24 unit radius area. Figure 5.1 illustrates a small region of one of these dense terrains. Then, we created five *sparse* data sets by sampling 1% to 5% of the original simulated grid's points. This procedure resulted in sparse data sets of sizes ranging from over 9K to over 48K. The sampling was done so that the concentration of points in different locations vary. That is, for each data set, 5% of the points were sampled from ten randomly selected Gaussian distributions. The rest of the points were drawn from the uniform distribution. We then removed duplicates that were resulted from sampling in these two different manners.

We also used the exhaustive Walker Lake data set, which is described in [80]. This data set was originally derived from a digital elevation model from the Walker Lake area in Nevada, U.S. There are two variables measured at 78000 points on a 260×300 grid. These two variables are continuous and their values range from zero to several thousands. These variables, which we will refer to as U and V, are

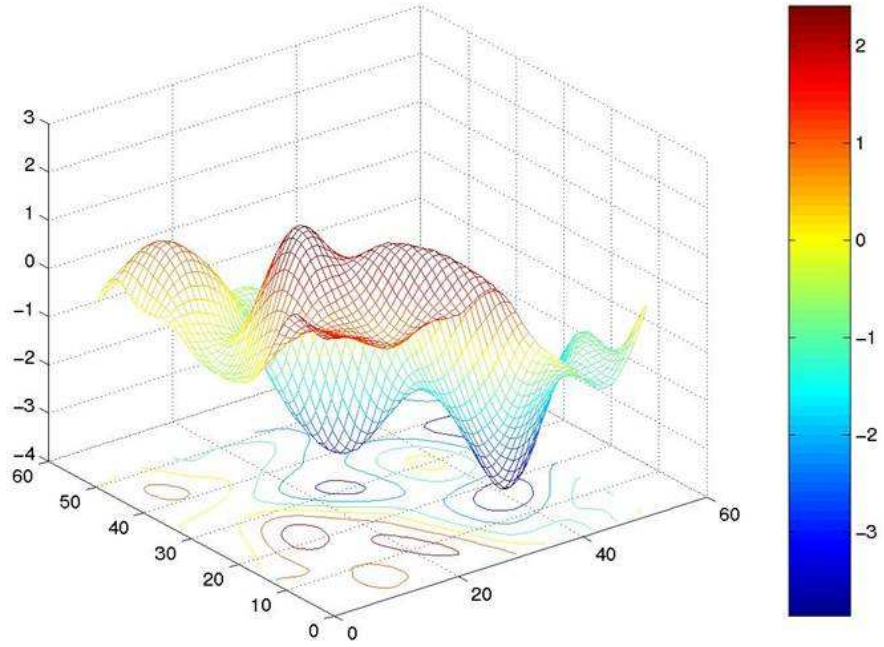


Fig. 5.1: A small region of one of the dense data sets that were generated

related to topographic features. Authors in [80] try to keep their book generic by mentioning that these variables can represent various features such as thickness of a geographic horizon, rainfall measurements, soil strength, etc. From the dense grid, we created two scattered data sets (one for U, and one for V). In each case we sampled less than 5% of the points from the grid. About 95% of the sampled points were from the uniform distribution while the rest were sampled from five Gaussian clusters.

5.4 Experiments

All experiments were run on a Sun Fire V20z running Red Hat Enterprise release 3, using the g++ compiler version 3.2.3. Our software is implemented in

C++ and uses the Geostatistical Template Library (GsTL) [137] and Approximate Nearest Neighbor library (ANN) [114]. GsTL is used for building and solving the ordinary kriging systems, and ANN is used for finding nearest neighbors for local approaches.

For each input data we examined various ordinary kriging interpolation methods on 200 query points which are missing in our sparse data sets. One hundred of these query points were sampled uniformly from the original grids. The other 100 query points were sampled from the same Gaussian distributions that were used in the generation of a small percentage of the sparse data. We used two classes of interpolation techniques: local and global methods. Local methods used Gaussian elimination for finding the solution of the linear system while global methods used a sparse SYMMLQ with $threshold = 0$ (see Section 5.2). All experiments' running times are averaged over five runs.

We examined methods mentioned in Section 5.2 for each query point. Global approaches require selection of a tapering function. Note that the covariance model for synthetic data is Gaussian, which is infinitely differentiable. Therefore, there is no function which is as smooth as the covariance model to guarantee convergence to the optimal solution. For synthetic data, we examined all tapers mentioned in Section 5.1.2 to introduce sparsity while maintaining positive definiteness of the covariance matrix. For real data, we used the spherical tapering function since the underlying covariance model was spherical as well, and thus we have a valid taper. The value for θ was chosen as the distance beyond which our data's covariance function, is less than 10^{-6}. After performing tapering and storing the global sparse

covariance matrix, we examined two approaches for solving the linear system. One approach solves the tapered global system using sparse SYMMLQ, and the other approach solves the tapered and projected global system as described in Section 5.2. Next, we analyze the quality of results, time spent solving the linear systems, and memory savings associated with our global approaches.

Table 5.1: Average absolute errors over 200 randomly selected query points.

n	Local		Tapered Global							
	Fixed Num	Fixed Radius	Top Hat	Top Hat Projected	Spherical	Spherical Projected	W_1	W_1 Projected	W_2	W_2 Projected
48513	0.416	0.414	0.333	0.334	0.336	0.337	0.278	0.279	0.276	0.284
39109	0.461	0.462	0.346	0.345	0.343	0.342	0.314	0.316	0.313	0.322
29487	0.504	0.498	0.429	0.430	0.430	0.430	0.384	0.384	0.372	0.382
19757	0.569	0.562	0.473	0.474	0.471	0.471	0.460	0.463	0.459	0.470
9951	0.749	0.756	0.604	0.605	0.602	0.603	0.608	0.610	0.619	0.637

Table 5.2: Average CPU times for solving the system over 200 random query points.

n	Local		Tapered Global							
	Fixed Num	Fixed Radius	Top Hat	Top Hat Projected	Spherical	Spherical Projected	W_1	W_1 Projected	W_2	W_2 Projected
48513	0.03278	0.00862	8.456	0.01519	7.006	0.01393	31.757	0.0444	57.199	0.04515
39109	0.01473	0.00414	4.991	0.00936	4.150	0.00827	17.859	0.0235	31.558	0.02370
29487	0.01527	0.00224	2.563	0.00604	2.103	0.00528	08.732	0.0139	15.171	0.01391
19757	0.00185	0.00046	0.954	0.00226	0.798	0.00193	02.851	0.0036	05.158	0.00396
9951	0.00034	0.00010	0.206	0.00045	0.169	0.00037	00.509	0.0005	00.726	0.00064

Table 5.3: Memory savings in the global tapered coefficient matrix

n	$(n+1)^2$ (Total Elements)	Stored Elements	% Stored	Savings Factor
48513	2,353,608,196	5,382,536	0.229	437.267
39109	1,529,592,100	3,516,756	0.230	434.944
29487	869,542,144	2,040,072	0.235	426.231
19757	390,378,564	934,468	0.239	417.755
9951	99,042,304	252,526	0.255	392.206

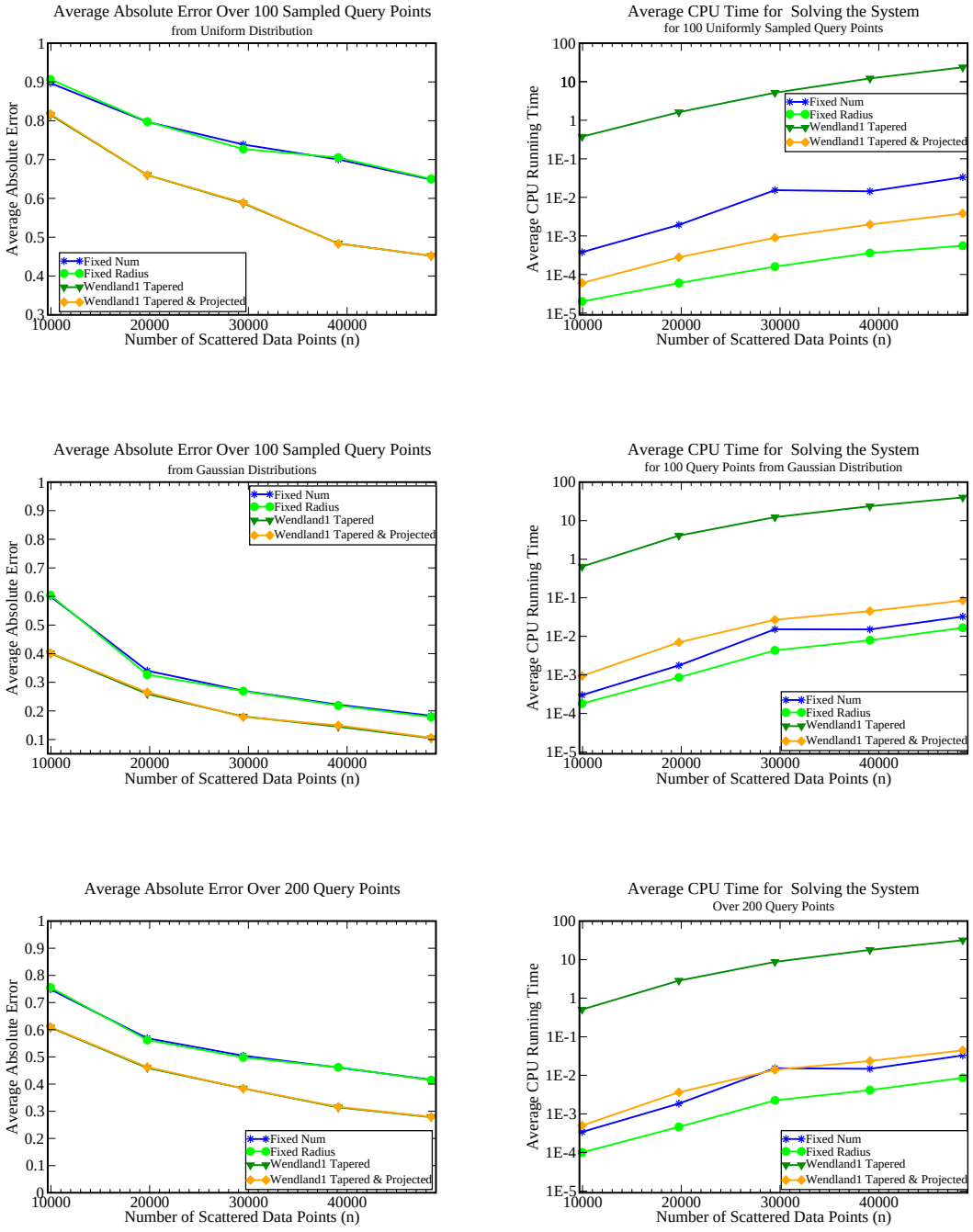


Fig. 5.2: Left: Average absolute errors. Right: Average CPU running times

5.4.1 Synthetic data

Table 5.1 gives the overall average absolute estimation errors over the 200 query points compared to the ground truth values generated on the original grid. Table 5.2 reports the corresponding average CPU running times for solving the linear systems involved. Even though there is no taper which is as smooth as the Gaussian model to guarantee convergence to the optimal estimates, in almost all cases, we obtained lower estimation errors when using global tapered approaches. As expected, smoother functions result in lower estimation errors. Also, results from *tapered and projected* cases are comparable to their corresponding *tapered global* approaches. In other words, projecting the global tapered system did not significantly affect the quality of results compared to the global tapered approach in our experiments. In most cases, Top Hat and Spherical tapers performed similar to each other with respect to the estimation error, and so did $Wendland_1$ and $Wendland_2$ tapers. Wendland tapers give the lowest overall estimation errors since they are smoother functions. Among Wendland tapers, $Wendland_1$ has lower CPU running times for solving the systems (Table 5.2). Thus, we plot the absolute errors and CPU running times for local approaches and global cases where $Wendland_1$ taper is being used (Figure 5.2). As seen in Table 5.2, global tapered and projected approaches are a factor of 2–3 orders of magnitude faster than the global tapered approaches, and are comparable to running times of the local approaches. Right column of Figure 5.2 displays these running times. The absolute estimation errors for global approaches, as seen on Table 5.1 and the left column of Figure 5.2, are lower than the local

approaches.

For local approaches, using fixed radius neighborhoods resulted in lower overall errors for query points from the Gaussian distribution. Using fixed number of neighbors seems more appropriate for query points from the uniform distribution, where not enough points may be within a fixed radius. Considering maximum degree of points' covariance connectivity as number of neighbors to use in the local approach requires extra work and longer running times compared to the fixed radius approach. The fixed radius local approach is faster than the fixed neighborhood approach by 1–2 orders of magnitude for the uniform query points, and is faster within a constant factor to one order of magnitude for query points from clusters, while giving better or very close by estimations compared to the results obtained when using fixed number of neighbors.

Tapering covariances, when used with sparse implementations for solving the linear systems, results in significant memory savings. Ordinary kriging of n data points involves a coefficient matrix of size $(n+1)^2$ (see Eq. (A.25) in Appendix A). Table 5.3 reports memory savings due to tapering. Memory needed after tapering is a factor of roughly 400 less than the original coefficient matrix's size.

5.4.2 Real data

As explained in Section 5.3, we have two real data sets, each representing a different measurement, called U and V. Since we know that the underlying covariance model for these data sets is a Spherical model, Eq. (A.14) in Appendix A, we only

applied the Spherical taper (Eq. 5.10). Table 5.4 presents the overall average normalized absolute error. As before, global approaches give better estimation errors than the local approaches, even though the difference in errors is subtler compared to the synthetic data.

Similarly, Table 5.5 present CPU running times for solving the ordinary kriging systems. The running times, in contrast to the estimation errors, show significant improvements, even when using the global tapered system without projection. This is due to two reasons. First, the data sets are denser than the synthetic data. For real data, we have sampled almost 5% of the original grid, while for synthetic data this ratio ranged from 1% to 5%. This makes the maximum number of neighbors to consider in local approaches much larger than number of neighbors that were considered in local approaches for synthetic data. Second, the original covariance model, Spherical model (Eq. (A.14) in Appendix A), is a tapered function itself (unlike the Gaussian model), even before more sparsity is introduced via tapering. This sparseness is not being taken advantage of in local approaches that use Gaussian elimination to solve the interpolation systems, and where the largest safest neighborhood is being used. Even though the tapered global systems are solved quite fast compared to the dense local systems, we still can improve their running times by an order of magnitude using the tapered and projected approach.

Table 5.6 indicates that global tapered approaches use a factor of 4-22 less memory compared the original global systems. Again, these factors are smaller compared to the synthetic data sets, since the sampled points are denser (higher percentage of the original grid).

Table 5.4: Average normalized absolute errors over 200 query points.

n	Variable	Local		Tapered Global	
		Fixed Num	Fixed Radius	Spherical	Spherical Projected
3720	u	0.380	0.379	0.364	0.360
3675	v	0.346	0.346	0.342	0.341

Table 5.5: Average CPU times over 200 query points.

n	Variable	Local		Tapered Global	
		Fixed Num	Fixed Radius	Spherical	Spherical Projected
3720	u	4.649	4.023	0.729	0.045
3675	v	4.649	4.650	1.642	0.119

5.5 Summary

Solving very large ordinary kriging systems via direct approaches is infeasible for large data sets. We implemented efficient ordinary kriging algorithms through covariance tapering [55] and iterative methods [118, 131]. Furrer *et al.* [55] used covariance tapering along with sparse Cholesky decomposition to solve simple kriging systems. We explained why Cholesky decomposition is not applicable to the ordinary kriging problem. We used tapering with sparse SYMMLQ method to solve large ordinary kriging systems. We also implemented a variant of the global tapered method through projecting the large global system onto an appropriate smaller system. Global tapered methods resulted in saving factors ranging from roughly 4 to 400 for the storage of the coefficient matrix of the ordinary kriging system compared

Table 5.6: Memory savings in the global tapered coefficient matrix

n	$(n+1)^2$ (Total Elements)	Stored Elements	% Stored	Savings Factor
3720	13,845,841	3,013,823	21.77	4.59
3675	13,512,976	2,973,046	22.00	4.54

to the original global system. Global tapered iterative methods gave better estimation errors compared to the local approaches. In all cases, the estimation results of the global tapered method were very close to the global tapered and projected method. This is while global tapered and projected method solves the linear systems within order(s) of magnitude faster than the global tapered method. This method can be viewed as a way of adaptively finding the correct neighbors to consider for the interpolation problem. Results of traditional local approaches depend on the underlying points' distribution, and whether or not enough points are included in the specified neighborhood.[1]

[1]I would like to thank Galen Balcom for his contributions to the C++ implementation of the SYMMLQ algorithm during his summer 2006 internship at NASA GSFC. I would also like to thank Dianne P. O'Leary for helpful discussions that greatly contributed to the work presented in this chapter, in particular to analysis presented in Section 5.1.1.

Chapter 6

Kriging via Fast Multipole and Iterative Methods

We mentioned computational challenges involved in solving large kriging systems in Chapter 5 and Appendix A. In particular, for very large systems, exact methods such as Gaussian elimination are impractical since they require $O(n^3)$ time and $O(n^2)$ storage. Iterative methods such as SYMMLQ reduce these requirements to $O(n^2)$ and $O(n)$ for solving time and storage, respectively. However, for very large data sets $O(n^2)$ solving time is still an unexpectedly high computational cost. In Chapter 5 we demonstrated efficient implementations based on iterative methods and tapering functions. We also mentioned that achieving high estimation accuracy is related to finding valid tapers [55]. Such valid tapers are sometimes hard to design for many covariance functions. In fact, for the Gaussian covariance function, which is indefinitely differentiable, such a valid taper does not even exist. Tapering is also most effective when covariance functions have small ranges.

In this chapter, we address the shortcomings of the previous approaches through an alternative based on Fast Multipole Methods (FMM). The FMM was first introduced by Greengard and Rokhlin for fast multiplication of a *structured* matrix by a vector [61,62]. An $(M \times N)$ matrix is structured if its elements depend on $O(M+N)$ parameters. For example, consider the covariance matrix of two data sets, where entries are calculated as a function of the points' pairwise distances. Such a matrix

is structured. If the Gaussian function is used for generating the matrix entries, the matrix-vector product is called the *Gauss transform*. In this chapter, we utilize efficient implementations of the Gauss transform based on the FMM approach (see [136, 162]) for solving the large ordinary kriging systems. As Billings *et al.* [22] suggested, we also use an iterative approach in combination with an FMM-based approach for solving large ordinary kriging systems. Similar to Chapter 5, we use the SYMMLQ method [122], for solving the large ordinary kriging systems.

The remainder of this chapter is organized as follows. We proceed by introducing the Gauss Transform in Section 6.1. We then describe two existing efficient implementations for the Gauss Transform in Sections 6.2 and 6.3. We explain the applicability of these approaches to solving the ordinary kriging system of Section 6.4. Section 6.5 describes data sets used for this chapter. Then, we describe our experiments and results in Section 6.6. Section 6.7 summarizes this chapter.

6.1 Gauss Transform

The *Gaussian kernel function* of two points x and y in d-dimensional space, denoted $ker(x, y)$, is defined as

$$ker(x, y) = e^{\frac{-||x-y||^2}{b^2}}, \tag{6.1}$$

where b is the *bandwidth* of the kernel. The sum of multivariate Gaussian kernels is known as the *discrete Gauss transform* in scientific computing. In other words, for

each *target* point $y_i \in \mathbb{R}^d$, $\{i = 1 \dots M\}$, the discrete Gauss transform is defined as

$$G(y_j) = \sum_{i=1}^{N} q_i e^{\frac{-||x_i - y_j||^2}{b^2}}, \tag{6.2}$$

where $\{x_i \in \mathbb{R}^d\}_{\{i=1,\dots N\}}$ are the *source points* (the center of their Gaussians), and $b \in \mathbb{R}^+$ is the *source scale* or *bandwidth* or *range*, and $\{q_i \in \mathbb{R}\}_{\{i=1\dots N\}}$ are the source weights.

The task of evaluating the discrete Gauss transform for M target points to N different source locations arises in many applications. A straightforward implementation would take $O(MN)$ time. This problem can also be viewed as calculating a matrix-vector product $g = Kq$, where K is an $(M \times N)$ matrix, q is an $(N \times 1)$ vector, and g is an $(M \times 1)$ vector. Considering the Gaussian kernel k, source weights, and source and target points sets, elements of K, q, and g are calculated as follows.

$$\begin{aligned} K(i,j) &= ker(x_i, y_j) \\ g(j) &= G(y_j) \\ q(i) &= q_i. \end{aligned}$$

In the remainder of this chapter, the *Gauss Transform* (GT) refers to calculating such matrix-vector products:

$$g = Kq. \tag{6.3}$$

Suppose we are interested in product of matrix K by different vectors. That is, we use matrix K repeatedly. There are two possible implementations. One approach it to calculate entries of matrix K once, store them, and use them every time needed. This approach avoids repeated calculations of entries of the K matrix. However, it requires $O(n^2)$ memory, which is a problem for very large data sets. The

other alternative is to generate entries of the matrix K on demand and avoid storing them explicitly. This approach, while memory-efficient, suffers from excessive computational cost. However, for very large data sets, this implementation can be carried out, although very slowly, by a computer of moderate resources. Thus, we use the memory-efficient approach as our reference implementation of the GT. We compare performances of this approach with other implementations that gain their efficiency by approximating the matrix-vector product.

6.2 Improved Fast Gauss Transform

The *Improved Fast Gauss Transform* (IFGT) is an efficient algorithm for approximating the Gauss transform (see Eq. (6.3)). For any $\epsilon > 0$, $\widehat{G}$ is an ϵ-*exact* approximation to G if the maximum absolute error relative to the total weight $Q = \sum_{i=1}^{N} |q_i|$ is upper bounded by ϵ:

$$\max_{y_j} \left[\frac{|\widehat{G}(y_j) - G(y_j)|}{Q} \right] \leq \epsilon.$$

The *fast Gauss transform*, first proposed by Greengard and Stein [63], is an ϵ-exact approximation algorithm for the Gauss transform. This algorithm reduces the Gauss transform's computational complexity from $O(MN)$ to $O(M + N)$. However, this algorithm's constant factor grows exponentially with dimension d. Later improvements, including the IFGT algorithm, reduced this constant factor to asymptotically polynomial order in terms of d. The IFGT algorithm was first introduced by Yang *et al.* [162]. Their implementation did not use a sufficiently tight error bound to be useful in practice. Also, they did not adaptively select the necessary parameters

to achieve the desired error bound. Raykar *et al.* later presented an approach that overcame these shortcomings [135, 136]. The efficiency of the IFGT algorithm, depends on the desired ϵ accuracy. The computational cost of IFGT increases as we decrease the desired ϵ value.

```
for j = 1 to M do
  g_j = 0;
  for i = 1 to N do
    g_j = g_j + φ(x_i, y_j)q_i;
  end for
end for
```

Fig. 6.1: Evaluating an exact discrete Gauss Transform

Suppose elements of an $(M \times N)$ matrix K are calculated via a function ϕ of N source points $x_1 \dots x_N$, and M target points $y_1, \dots y_M$. Then, the matrix-vector product $g = Kq$, is usually coded as shown in Figure 6.1. Thus, this product is calculated in $O(MN)$ time. The idea is to decompose this calculation into two steps. The process of representing an $O(MN)$ matrix-vector multiplication by two consecutive processes with total complexity of $O(M + N)$ is called *factorization*. Suppose such a factorization exists. That is, the function $\phi(x_i, y_j)$ can be calculated as an expansion around a center point $x_* \in \mathbb{R}^d$ as follows.

$$\phi(x_i, y_j) = \sum_{m=0}^{\infty} a_m(x_i - x_*) f_m(y_j - x_*). \tag{6.4}$$

Then, the vector g_j mentioned above can be closely approximated as follows.

$$\begin{aligned} g_j &= \sum_{i=1}^{N} q_i \phi(x_i, y_j) = \sum_{i=1}^{N} q_i \sum_{m=0}^{p-1} a_m(x_i - x_*) f_m(y_j - x_*) \\ &= \sum_{m=0}^{p-1} f_m(y_j - x_*) \sum_{i=1}^{N} q_i a_m(x_i - x_*). \end{aligned}$$

If we precalculate and store $c_m = \sum_{i=1}^{N} q_i a_m(x_i - x_*)$, for $m = 0 \ldots p-1$, the above approximation can be calculated as shown in Figure 6.2 (see [65] for details).

```
for m = 0 to p − 1 do
  c_m = 0;
  for i = 1 to N do
    c_m = c_m + a_m(x_i − x_*)q_i;
  end for
end for
for j = 1 to M do
  g_j = 0;
  for m = 0 to p − 1 do
    g_j = g_j + c_m f_m(y_j − x_*);
  end for
end for
```

Fig. 6.2: Evaluating an approximate discrete Gauss Transform via IFGT

These two steps are calculated in $O(Mp)$ and $O(Np)$ time respectively, where $p \ll \min(M, N)$, for a total time of $O(Mp + Np)$. Please note that if one center x_* is used for factorization, a relatively large values of p is required for the truncation to ensure that the matrix-vector approximation is sufficiently close to the exact one. Instead, multiple centers are used to reduce this computation complexity. This is usually done via clustering data points, and then using each point's corresponding cluster center as x_* in the mentioned expansion. We use an implementation due to Raykar *et al.* [135, 136]. This implementation works for points in 1, 2, or 3 dimensional spaces. The algorithm has four stages. The first stage involves determining the parameters of the algorithm based on specified error bounds, kernel bandwidth, and data distribution. Second, the d-dimensional space is subdivided using a k-center clustering (see Chapter 2). Next, a truncated representation of kernels inside each cluster is built using a set of decaying basis functions. Finally, the influence of

all the data in a neighborhood using coefficients at cluster centers are collected and the approximate GT is evaluated. Please see [135, 136] for more details.

6.3 Gauss Transform with Nearest Neighbor Searching (GTANN)

GTANN is another efficient algorithm for calculating matrix-vector products. This method was implemented by Raykar [134], where it is referred to as the FigTree method. This method is most effective when the Gaussian models being used have small ranges. Instead of converting a double for-loop into two consecutive for-loops as IFGT does (see Figure 6.2), GTANN reduces the complexity of the second inner loop in the exact method. Since the Gaussian function dissipates very rapidly, nearby points have the greatest influence. This method avoids multiplying each target point by distant source points. This method utilizes the kd-tree data structure and the nearest neighbor finding algorithms from the ANN library [114].

6.4 Our Approaches

The Gaussian covariance model used in geostatistics, Eq. (A.13) in Appendix A, differs slightly from the Gaussian kernel, Eq. (6.1). Thus, efficient Gauss Transform implementations can be applied to the kriging problems with a Gaussian covariance models. We applied all mentioned Gauss Transform implementations to the calculation of matrix-vector products involved in solving the ordinary kriging system, where the covariance model involved is Gaussian.

Gauss Transform (GT): This version solves the exact ordinary kriging system with a Gaussian covariance model, which generates all matrix entries on demand.

Improved Fast Gauss Transform (IFGT): This version approximates the matrix-vector multiplications involved in solving the kriging system via the IFGT method (see Section 6.2). We use this method for solving the kriging linear, system mentioned in Eq. (A.25) in Appendix A, via the SYMMLQ iterative method. We use the fact that a large portion of the matrix-vector product involves multiplication of the covariance matrix C by a vector, and that elements of matrix C are estimated via a modeling function, which we can factor. For example, consider the Gaussian covariance model, Eq. (A.13) in Appendix A, for points in $\mathbb{R}^1$. This covariance function for two points x_i and y_j can be represented and factorized as follows. (Please note that a Taylor expansion is used in the last step.)

$$\begin{aligned}
\phi(x_i, y_j) &= e^{\frac{-3\cdot(x_i-y_j)^2}{a^2}} \\
&= e^{\frac{-3\cdot((x_i-x_*)-(y_j-x_*))^2}{a^2}} \\
&= e^{\frac{-3\cdot(x_i-x_*)^2}{a^2}} e^{\frac{-3\cdot(y_j-x_*)^2}{a^2}} e^{\frac{6\cdot(x_i-x_*)(y_j-x_*)}{a^2}} \\
&= e^{\frac{-3\cdot(x_i-x_*)^2}{a^2}} e^{\frac{-3\cdot(y_j-x_*)^2}{a^2}} \sum_{m=0}^{\infty} \frac{6^m \cdot (x_i-x_*)^m (y_j-x_*)^m}{a^2 \cdot m!}.
\end{aligned}$$

Let

$$\begin{aligned}
a_m(x_i - x_*) &= \frac{1}{a} \cdot \sqrt{\frac{6^m}{m!}} \cdot e^{\frac{-3(x_i-x_*)^2}{a^2}} \cdot (x_i - x_*)^m, \\
f_m(y_j - x_*) &= \frac{1}{a} \cdot \sqrt{\frac{6^m}{m!}} \cdot e^{\frac{-3(y_j-x_*)^2}{a^2}} \cdot (y_j - x_*)^m,
\end{aligned}$$

for $m = 0 \ldots (p-1)$ in Eq. (6.4). Please note that the approximation and efficiency is introduced by summing the Taylor series expansion up to p elements only. We

use this factorization for the Gaussian covariance model, when solving the ordinary kriging system. Thus, whenever multiplication of matrix C by a vector is needed in the SYMMLQ method, we use the IFGT implementation for its calculation.

Gauss Transform with Nearest Neighbor Searching (GTANN): This version approximates the matrix-vector multiplications involved in solving the kriging system via GTANN method (see Section 6.3). First, based on the desired error bound, a search radius is calculated. Then, for each target point, source points within that radius are considered. These source points are calculated via fixed-radius nearest neighbor search routines of the ANN library [114]. Finally, for each target point, their nearest neighbor source points are calculated in matrix-vector product calculations, involving the covariance matrix C in the ordinary kriging system.

6.5 Data Sets

We use large scattered data sets to test and evaluate performance of the various approaches given in Section 6.4. As mentioned before, it would be infeasible to solve the original global systems exactly for very large data sets, and thus cannot compare our solutions with respect to the original global systems. Also, since performance of local approaches may depend on the density of the data points around the query point, we generated data sets that are not uniformly distributed to better model realistic input configurations. Therefore, we create our scattered data sets by sampling points of a large dense grid from both uniform and Gaussian distributions. The data generation process is very similar to the one explained in Section 5.3. This

time, 0.1%–0.5% of points from the dense grid were sampled. In order to evaluate the effect of data sizes and covariance function's ranges on the performance of the GTANN and IFGT approaches, we generated three sets of sparse data sets. For the first set, the number of sampled points varied from 1000 up to 5000, while their covariance model had a small range value of 12 compared to data points' maximum pairwise distances, where range corresponds to value a in Eq. (A.13) in Appendix A. For the second set of experiments, we varied the number of samples in the same manner, except that the points' covariance model had a larger range equal to 100. Finally, for the last set of experiments, we sampled 5000 points from dense grids, where points' covariance model had ranges equal to 12, 24, 100, 250, and 500. That is, for each data set, 5% of the points were sampled from ten randomly selected Gaussian distributions. The rest of the points were drawn from the uniform distribution. We removed duplicates that were resulted from sampling in these two different manners.

6.6 Experiments

All experiments were run on a Sun Fire V20z running Red Hat Enterprise release 3, using the g++ compiler version 3.2.3. Our software is implemented in C++ and uses the Geostatistical Template Library (GsTL) [137] and Approximate Nearest Neighbor library (ANN) [114]. GsTL is used for building and solving the ordinary kriging systems, and ANN is used for finding nearest neighbors when using the GTANN approach.

For each input data set we examined various ordinary kriging interpolation methods on 200 query points which are drawn from the same dense grid but are not present in the sampled data set. One hundred of these query points were sampled uniformly from the original grids. The other 100 query points were sampled from the same Gaussian distributions that were used in the generation of a small percentage of the sparse data. We used two classes of interpolation techniques: local and global methods. Local methods used Gaussian elimination for finding the solution of the linear system while global methods used a sparse SYMMLQ with $threshold = 0$ (see Section 5.2). All experiments' running times are averaged over five runs. In all cases, for the IFGT and GTANN approaches we required the approximate matrix-vector products to be evaluated within $\epsilon = 10^{-4}$ accuracy. We set the desired solutions' relative error, which is the convergence criteria for the SYMMLQ method, to 10^{-3}. Thus, if the system was solved exactly, we expect the relative error to be less than 10^{-3}. The exact error is likely to be higher than that. However, note that for the IFGT and GTANN approaches the system is not solved exactly, either. For these methods, at every iteration, the matrix-vector products are calculated approximately. Thus, when compared to the exact solution obtained from the GT method, our approximate approaches may result in both relative and absolute errors higher than 10^{-3}. Recent studies in linear algebra [153], also mentioned in Chapter 8 of [134], studies the effect of approximate matrix-vector multiplication for solving Krylov subspace methods for the solution of symmetric and nonsymmetric systems. Results indicate that as the number of iterations increases, one can increase the amount of error introduced for solving the system, and still expect convergence to

the solution [153].

We examined methods mentioned in Section 6.4 on the three data sets mentioned in Section 6.5. In the first experiment we examined the effect of the number of data points for each approach, when using a small Gaussian covariance range. Since IFGT and GTANN are expected to perform differently on different covariance ranges, we tried two sets of fixed range values when varying the number of data points. That is, we solved the ordinary kriging system for 200 query points on data sets of size ranging from 1000 up to 5000, with a small fixed Gaussian covariance model of range 12. Table 6.6 presents the running times for solving the linear systems for these data sets. Table 6.6 presents the error residuals of solving these systems, as well as the average number of iterations it took SYMMLQ in each case to converge. Figure 6.3 shows these results. As we can see, when utilizing approximate methods IFGT and GTANN the exact residuals are comparable. The IFGT approach gave speed-ups ranging from 1.3–7.6, while GTANN resulted in speed-ups ranging roughly from 50–150. This is mainly due to the fact that the covariance function's range is rather small. Since only a limited number of source points influences each target point, collecting and calculating the influence of all source points for each target point (the IFGT approach) is excessive. The GTANN approach works well for such cases by considering only the nearest source points to each target point, which are in fact the only points influencing the final result. In both cases, the speed-ups grow with number of data points. The algorithms' performances do not differ significantly for points from the Gaussian distribution compared to those from uniform distribution.

Table 6.1: Experiment 1: Average CPU time for solving the system for (a) 100 uniformly sampled query points, (b) 100 query points sampled from the Gaussian distribution, (c) all 200 query points.

	n	GT	IFGT	GTANN	IFGT (Speed-up)	GTANN (Speed-up)
(a)	1000	1.225	0.879	0.019	1.394	65.810
	2000	5.985	1.971	0.058	3.037	102.664
	3000	16.826	3.372	0.134	4.990	125.795
	4000	35.272	4.796	0.252	7.354	140.159
	5000	77.022	10.102	0.503	7.624	153.223
(b)	1000	1.716	1.222	0.025	1.404	68.404
	2000	11.584	3.827	0.114	3.027	101.598
	3000	35.201	7.053	0.280	4.991	125.879
	4000	92.232	12.499	0.657	7.379	140.396
	5000	253.728	35.379	1.669	7.172	151.982
(c)	1000	1.470	1.050	0.022	1.400	67.299
	2000	8.785	2.899	0.086	3.030	101.959
	3000	26.014	5.213	0.207	4.990	125.852
	4000	63.752	8.648	0.454	7.372	140.330
	5000	165.375	22.741	1.086	7.272	152.269

Table 6.2: Experiment 1: Average iterations and exact residuals for (a) 100 uniformly sampled query points, (b) 100 query points sampled from the Gaussian distribution, (c) all 200 query points.

		Iterations			Exact Residuals		
	n	GT	IFGT	GTANN	GT	IFGT	GTANN
(a)	1000	2.87	2.87	2.87	0.000756	0.000756	0.000756
	2000	4.51	4.46	4.46	0.000597	0.000608	0.000608
	3000	6.82	6.83	6.83	0.000719	0.000721	0.000721
	4000	8.99	9.02	9.02	0.000769	0.000773	0.000772
	5000	14.52	17.18	14.49	0.000839	0.000839	0.000841
(b)	1000	5.94	5.94	5.94	0.000678	0.000679	0.000679
	2000	13.45	13.40	13.40	0.000810	0.000836	0.000837
	3000	19.70	19.65	19.65	0.000986	0.000986	0.000983
	4000	31.60	31.47	31.54	0.001096	0.001109	0.001106
	5000	59.31	72.51	59.64	0.001300	0.001304	0.001312
(c)	1000	4.41	4.41	4.41	0.000717	0.000717	0.000717
	2000	8.98	8.93	8.93	0.000704	0.000722	0.000722
	3000	13.26	13.24	13.24	0.000853	0.000854	0.000852
	4000	20.30	20.25	20.28	0.000932	0.000941	0.000939
	5000	36.92	44.85	37.07	0.001069	0.001072	0.001076

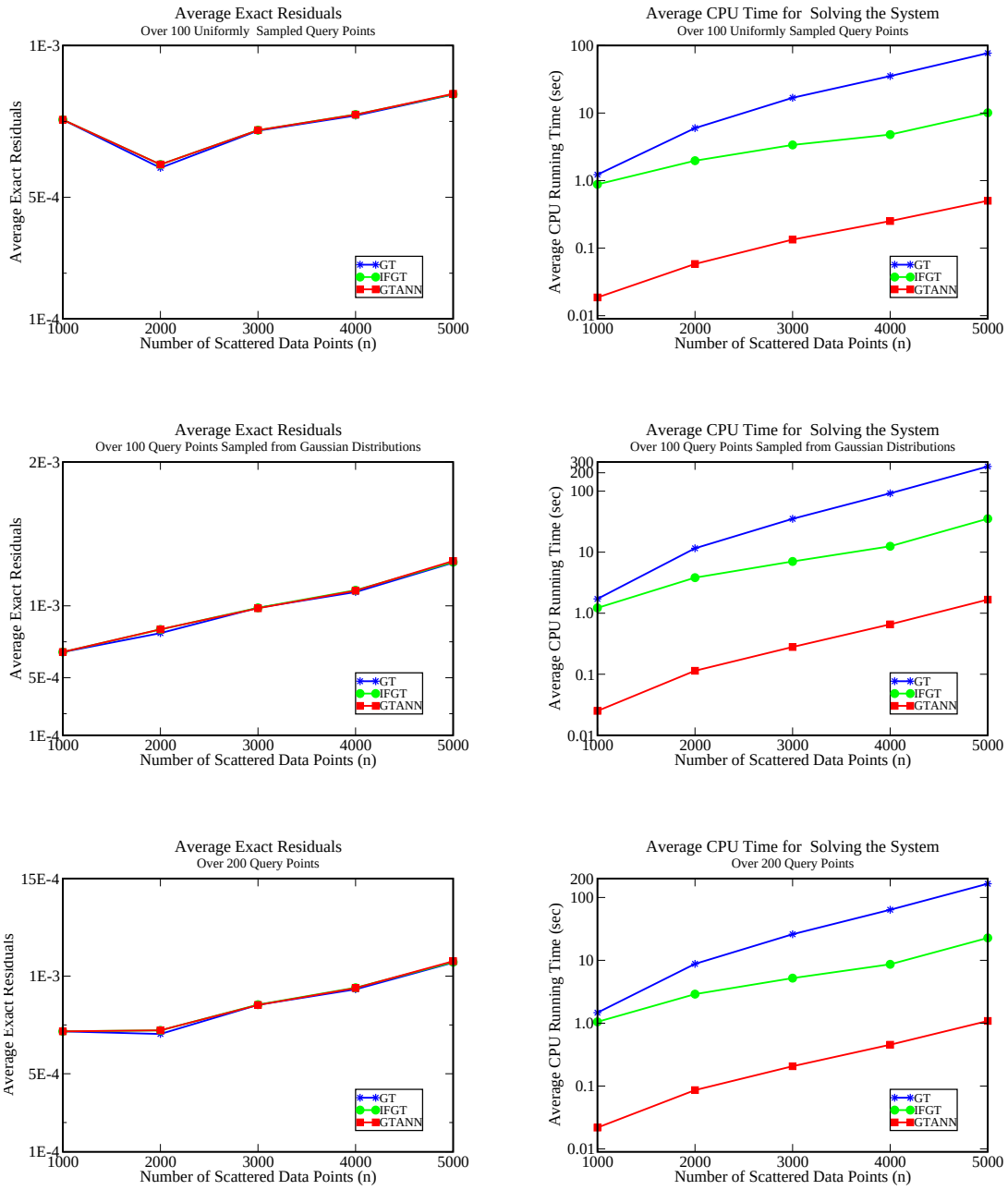


Fig. 6.3: Experiment 1, Left: Average exact residual. Right: Average CPU running times

Similarly, we repeated the experiments for our second data set. In this experiment, we again varied the number of sampled points. This time, we used a larger covariance range of 100 for the model. Table 6.6 presents running times of solving the linear systems for these data sets. Table 6.6 includes the exact error residuals of solving these systems, as well as average number of iterations used for solving the system. Figure 6.6 presents these results. While the GTANN approach did not result in significant speed-ups, the IFGT gave constant factor speed-ups ranging roughly from 1.5 to 7.5, as we increased the number of data points. The IFGT approach results in larger residuals for small data sets, and when solving the ordinary kriging system for query points from the uniform distribution. In particular, for $n = 1000$ and $n = 2000$, the performance of IFGT is not acceptable with respect to the exact residuals calculated. This poor overall result for these two cases is because the SYMMLQ method did not meet its convergence criteria before reaching its maximum number of iterations in few cases, raising the overall error average. For $n = 1000$, the iterative solver did not converge for 9 out of 100 random points from the uniform distribution. For $n = 2000$, the solver did not converge only for 3 out of 100 random points from the uniform distribution. Increasing the required accuracy for the IFGT algorithm, resolved this issue. Another possible approach to avoid such cases, and area for future work is to decrease the required accuracy as the number of iterations increases. As we mentioned before, one can increase the amount of error introduced for solving the system, and still expect convergence to the solution [153]. Our experiments show that as the number of data points increases, the quality of results approaches those of the exact methods. For the query points from the Gaussian

distribution, the quality of results are comparable to the exact method, when using the IFGT approach. The GTANN approach also results in comparable residuals to the exact methods in all cases.

Table 6.3: Experiment 2: Average CPU time for solving the system for (a) 100 uniformly sampled query points, (b) 100 query points sampled from the Gaussian distribution, (c) all 200 query points.

	n	GT	IFGT	GTANN	IFGT (Speed-up)	GTANN (Speed-up)
(a)	1000	23.643	14.131	15.148	1.673	1.561
	2000	50.464	16.497	48.971	3.059	1.030
	3000	105.296	23.605	136.793	4.461	0.770
	4000	174.982	29.660	169.391	5.900	1.033
	5000	277.039	37.545	408.058	7.379	0.679
(b)	1000	12.534	7.287	6.819	1.720	1.838
	2000	27.639	8.189	13.363	3.375	2.068
	3000	62.108	13.336	38.630	4.657	1.608
	4000	94.357	15.440	58.500	6.111	1.613
	5000	152.174	19.808	104.192	7.682	1.461
(c)	1000	18.088	10.709	10.983	1.689	1.647
	2000	39.051	12.343	31.167	3.164	1.253
	3000	83.702	18.471	87.711	4.532	0.954
	4000	134.669	22.550	113.945	5.972	1.182
	5000	214.607	28.676	256.125	7.484	0.838

Finally, we examined the effect of different covariance ranges on a fixed data set of size 5000. Table 6.6 presents running times of solving the linear systems for these data sets. Table 6.6 includes the exact error residuals for these systems, as well as the average number of iterations used for solving the system. Figure 6.6 presents these results. In all cases, the quality of results is comparable to those obtained from exact methods. The IFGT approach resulted in speed-ups of 7–15 in all cases. The GTANN approach is best when used for covariance functions with small range values of 12, and 25. While the GTANN approach is slower than the exact methods

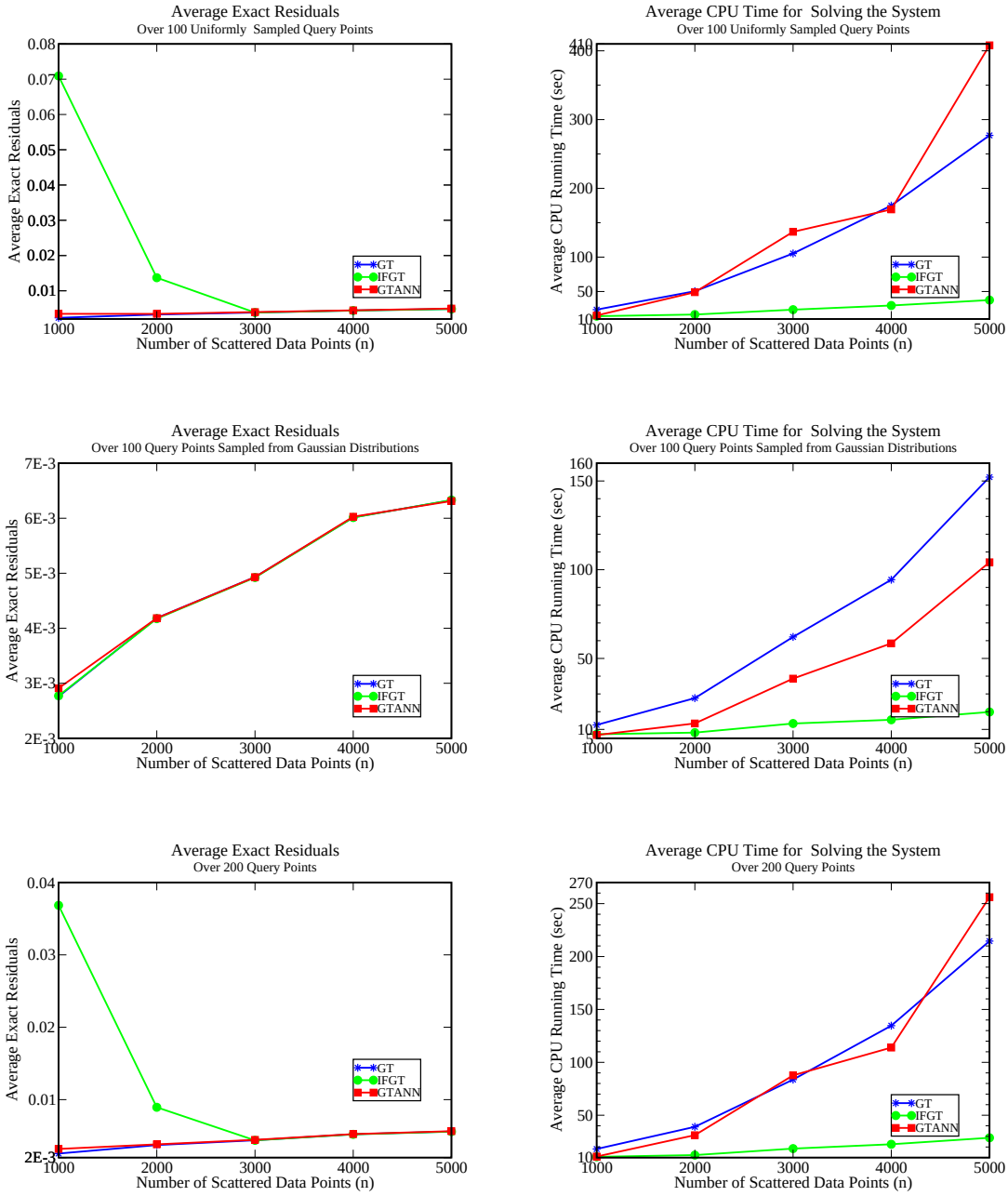


Fig. 6.4: Experiment 2, Left: Average exact residuals. Right: Average CPU running times

Table 6.4: Experiment 2: Average iterations and exact residuals for (a) 100 uniformly sampled query points, (b) 100 query points sampled from the Gaussian distribution, (c) all 200 query points.

		Iterations			Exact Residuals		
	n	GT	IFGT	GTANN	GT	IFGT	GTANN
(a)	1000	267.76	327.87	222.27	0.002346	0.070919	0.003468
	2000	154.00	184.02	163.44	0.003265	0.013714	0.003462
	3000	141.56	173.90	174.04	0.003843	0.003835	0.003968
	4000	131.74	158.67	131.44	0.004421	0.004393	0.004459
	5000	133.72	159.88	196.71	0.004868	0.004833	0.004973
(b)	1000	154.30	197.31	146.62	0.002760	0.002773	0.002910
	2000	82.05	98.14	82.39	0.004186	0.004174	0.004184
	3000	81.48	96.08	94.55	0.004931	0.004922	0.004930
	4000	68.74	80.20	68.76	0.006016	0.006012	0.006028
	5000	71.20	82.06	71.37	0.006327	0.006331	0.006312
(c)	1000	211.03	262.59	184.45	0.002553	0.036846	0.003189
	2000	118.03	141.08	122.92	0.003725	0.008944	0.003823
	3000	111.52	134.99	134.30	0.004387	0.004379	0.004449
	4000	100.24	119.44	100.10	0.005218	0.005203	0.005243
	5000	102.46	120.97	134.04	0.005597	0.005582	0.005643

for range values larger than 100, it results in speed-up factors of 151–153, and 47–49 for range values 12 and 25 respectively. Thus, as we can see in Figure 6.6, it is recommended to use the GTANN approach when the covariance function has small range values, and the IFGT approach when dealing with large ranges.

6.7 Summary

We adapted efficient implementations of the Gauss Transform for solving ordinary kriging systems. Please note that while IFGT was used for cases where the covariance function is Gaussian, a similar approach can be used for other covariance functions where a factorization, similar to the one described for the Gaussian function in Section 6.4, exists. The GTANN approach can also be used in other cases

Table 6.5: Experiment 3: Average CPU time for solving the system for (a) 100 uniformly sampled query points, (b) 100 query points sampled from the Gaussian distribution, (c) all 200 query points.

	Range	GT	IFGT	GTANN	IFGT (Speed-up)	GTANN (Speed-up)
(a)	12	77.022	10.102	0.503	7.624	153.223
	25	572.715	65.589	11.58776	8.732	49.424
	100	277.039	37.545	408.0578	7.379	0.679
	250	139.229	15.811	1320.934	8.806	0.105
	500	70.994	4.587	2406.208	15.478	0.030
(b)	12	253.728	35.379	1.669	7.172	151.982
	25	944.685	112.943	19.815	8.364	47.676
	100	152.174	19.808	104.192	7.682	1.461
	250	110.328	12.161	1040.044	9.073	0.106
	500	63.226	4.017	2167.496	15.741	0.029
(c)	12	165.375	22.741	1.086	7.272	152.269
	25	758.701	89.266	15.701	8.499	48.321
	100	214.607	28.676	256.125	7.484	0.838
	250	124.779	13.986	1180.490	8.922	0.106
	500	67.110	4.302	2286.852	15.601	0.029

Table 6.6: Experiment 3: Average iterations and exact residuals for (a) 100 uniformly sampled query points, (b) 100 query points sampled from the Gaussian distribution, (c) all 200 query points.

		Iterations			Exact Residuals		
	Range	GT	IFGT	GTANN	GT	IFGT	GTANN
(a)	12	14.52	17.18	14.49	0.000839	0.000839	0.000841
	25	163.36	199.58	164.58	0.001476	0.001476	0.001525
	100	133.72	159.88	196.71	0.004868	0.004833	0.004973
	250	64.72	85.52	65.30	0.010961	0.010849	0.011008
	500	30.54	42.48	30.36	0.017983	0.018409	0.018402
(b)	12	59.31	72.51	59.64	0.001300	0.001304	0.001312
	25	272.72	347.41	284.81	0.002227	0.002249	0.002711
	100	71.20	82.06	71.37	0.006327	0.006331	0.006312
	250	50.24	64.63	50.39	0.011910	0.011835	0.011931
	500	26.64	36.62	26.86	0.020036	0.019847	0.020152
(c)	12	36.92	44.85	37.07	0.001069	0.001072	0.001076
	25	218.04	273.49	224.70	0.001851	0.001862	0.002118
	100	102.46	120.97	134.04	0.005597	0.005582	0.005643
	250	57.48	75.08	57.85	0.011436	0.011342	0.011470
	500	28.59	39.55	28.61	0.019009	0.019128	0.019277

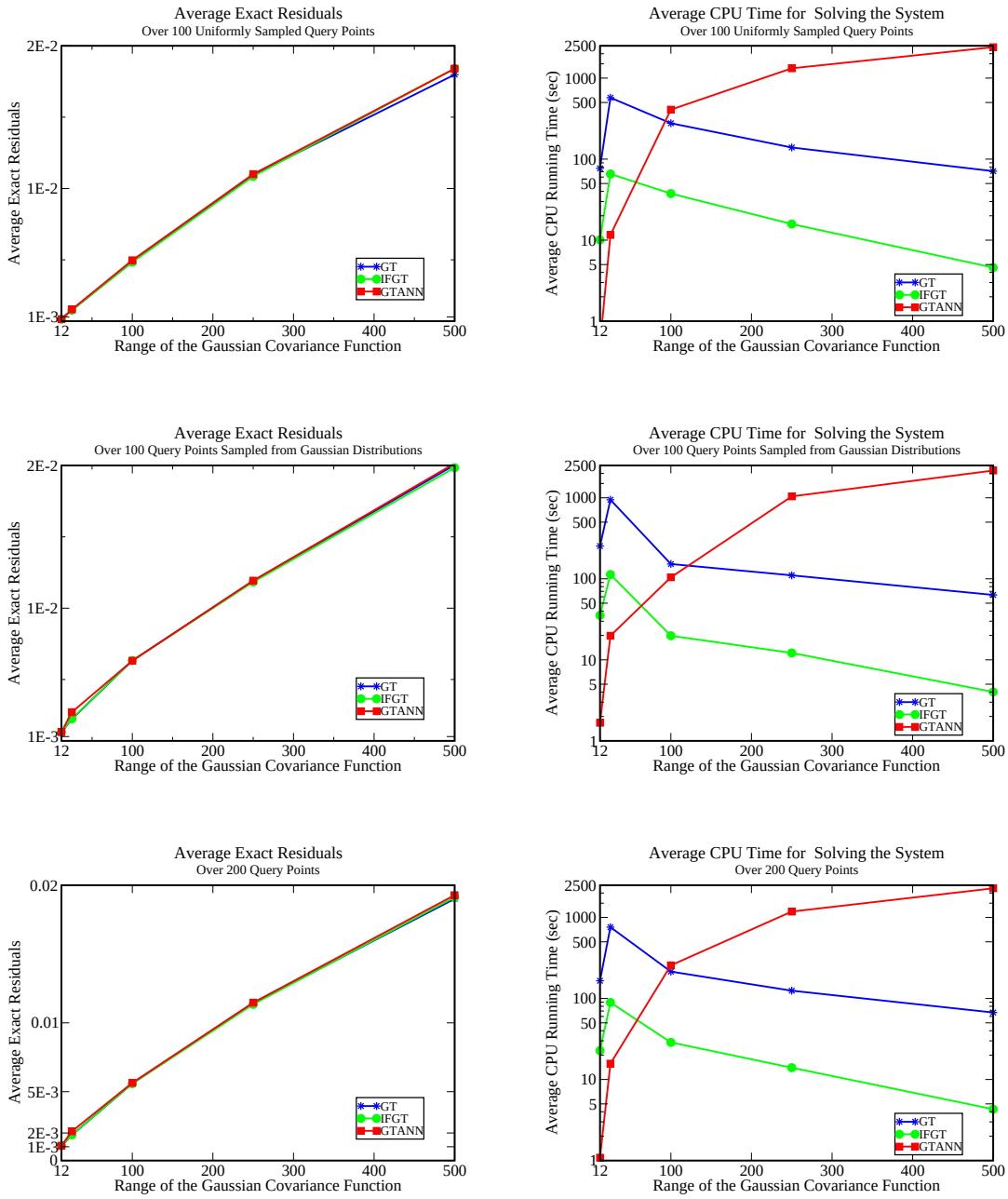


Fig. 6.5: Experiment 3, Left: Average exact residuals. Right: Average CPU running Times

regardless of the covariance function but it will be efficient for covariance functions with small ranges.

We examined the effect of number of points, query points' distribution, and the covariance functions' ranges on the running time for solving the system and the quality of results. The IFGT is more effective as number of data points increases. Our experiments using the IFGT approach for solving the ordinary kriging system demonstrated speed-up factors ranging from 7–15 when using 5000 points. Based on our tests on varying number of data points, we expect even higher speed-up factors compared to the exact method when using larger data sets. While for small data sets the IFGT method may not give high quality results compared to the exact methods, the quality of results are comparable to the exact methods for large data sets. The GTANN approach outperformed the IFGT method for small covariance range values of 12 and 25, resulting in speed-up factors as high as 153 and 49 respectively. The GTANN approach is slower than the exact methods for large ranges (100 and over in our experiments), and thus is not recommended to be used in these cases. A safe cutoff value for determining whether to use GTANN or IFGT is a covariance range value of 50. The quality of results when using GTANN was comparable to the exact methods in all cases.[1]

[1]I would like to thank Vikas Raykar for helpful discussions on integration and applicability of the IFGT and GTANN algorithms to the kriging problem. I would also like to thank Ramani Duraiswami for helpful discussions and his interest in this work.

Chapter 7

Data Fusion of Remotely Sensed Data

Data fusion has been defined in literature in different ways. Hall defined it as process of dealing with information from multiple sources to achieve refined and improved information for decision making [66]. Image fusion is a special case of the general data fusion problem where data being fused are images. The goal of performing image fusion is usually to increase either the spatial or spectral resolution of images involved.

One particular case of image fusion is *pan-sharpening*. Pan-sharpening is a technique that deals with limitations of sensors in capturing high resolution multispectal images [164]. That is, Panchromatic (Pan) images have high spatial resolution and low spectral resolution. On the other hand, multispectral images have high spectral resolution, since they cover a narrower wavelength, but have a lower spatial resolution. The goal of image fusion in this context is to create a high spatial and spectral resolution image given original images.

As in [129] we will present applications and objectives of data fusion in general and image fusion in particular and describe data types that are involved in remote sensing applications. We also review various fusion methods and the preprocessing or computational issues involved in each case. Another main area of research is assessing the quality of fusion performed. Thus, we also review various evaluation

techniques for image fusion.

7.1 Objectives and Applications

The objectives of fusion are well presented in a survey done by Pohl [129]. Some of the most commonly sought objectives of image fusion are improved registration accuracy, creation of stereo data sets, feature enhancement, improved classification, temporal aspects for change detection, and overcoming gaps. Please see [129] for descriptions of these objectives.

Data fusion has applications in various fields. Its many applications include topographic mapping and map updating, land use, forestry and agriculture, flood, ice, and snow monitoring, weather modeling, and study of invasive species.

All the mentioned applications are examples of what can be done through remote sensing. While we mention related work that has been done for other applications, we mainly focus on remote sensing application, data fusion methods in general and image fusion techniques in particular, and issues involved in evaluating the quality of fused data.

7.2 Data Sets Used in Remote Sensing Applications

In remote sensing applications the data that are used in fusion are usually from one or multiple sensors with different spatial, spectral, or temporal resolutions [129]. These data sets usually have a temporal aspect involved since one cannot assure that data were obtained exactly at the same time. Also in recent years, a lot of research

has been done in the area of sensor networks where one deals with large number of small scattered sensors [35, 74]. Integrating data received from each of these sources to model a physical phenomena is another source of data fusion applications.

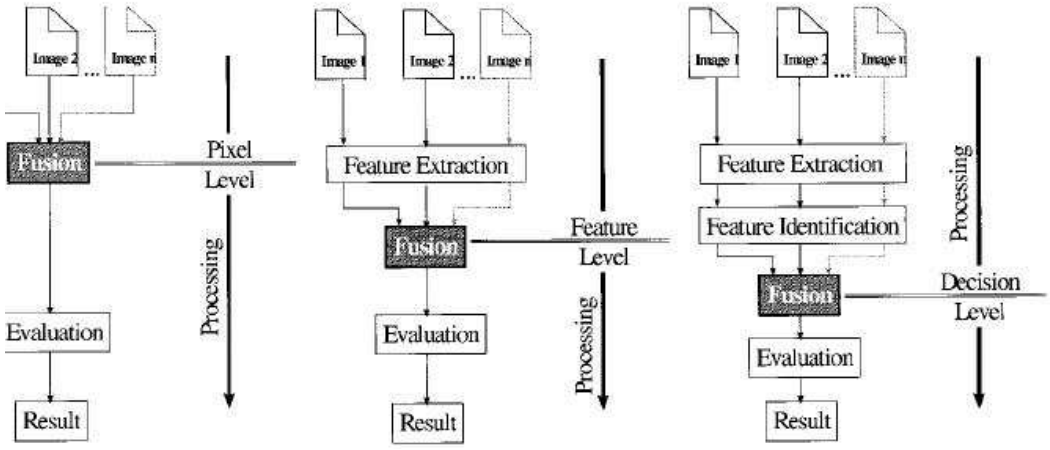


Fig. 7.1: Processing levels of fusion [129].

In this chapter, when talking about fusion, image fusion at its finest level of data (pixel) level is meant unless otherwise is specified. Fusion can also be performed after some processing is performed on the original data. Here are the three principal levels of fusion (see figure 7.1).

Pixel Level: This is fusion at the finest level which requires each sensor's data to be registered within sub-pixel accuracy to avoid fusing irrelevant pixels together [67].

Feature Level: This is fusion at an intermediate level which requires each data set to go through some process of object/feature recognition, and then fusion be performed on the extracted features using some statistical approach or neural networks [129].

Decision Level: Fusion at this level requires that decision making be first done though each data set separately, and then a final decision be made by using logical operators or by some heuristic enforcement [67].

While old surveys in the area mainly focus on pixel level fusion [129], recent works are mostly concerned with decision level fusion [27, 142]. This may be due to the fact that data of various natures (for example, temperature and humidity) need to be processed differently, fusing them at the pixel level may either not provide good results or simply not make sense due to different natures of data.

7.3 Fusion Methods

There are a number of methods used to perform data fusion. Here we review the most commonly used methods. Traditionally, methods used for data fusion were classified as one of the following two groups: color related methods, and numerical/statistical methods [129]. These methods will be discussed in greater detail below. In recent years, we see a third group of approaches to data fusion using machine learning approaches. When performing data fusion, one needs to consider the following two issues for the result: color distortion, and quality of fusion [164]. There are different approaches for each data fusion method mentioned below in order to reduce color distortions and improve quality of results.

First, we mention the preprocessing that needs to be performed on input data before being fused. Then, we mention a few image fusion methods as well as metrics that are used for assessing the quality of image fusion.

7.3.1 Preprocessing steps

Image fusion requires a few preprocessing steps. One needs to calculate and apply the correct translation, rotation, and scaling to map images to a common framework and resolution [26,129]. Calculating and applying transformations to an image so that it corresponds exactly to a particular reference image is called *image registration*. Please see [29] for more details. Results can be smoothed by a low pass filter to eliminate the block effects. Suppose each pixel of our high spatial resolution data (PAN) covers h units, and that each pixel of the low spatial resolution data (MS) covers l units. The first preprocessing step is to digitally enlarge the MS image by a factor $\frac{l}{h}$ in both directions so that we have pixel sizes the same as the PAN image [26, 129]. Results can be smoothed by a low pass filter to eliminate the block effects. Then, image-to-image points can be selected to register the MS images to the PAN image. Image registration is the process by which we determine a transformation that provides the most accurate match between two images [29]. After finding and applying such a transformation to the MS image to match our PAN image, we have registered images of the same size and from the same area, to which we can apply various fusion techniques. We would like the fused image to maintain high spectral accuracy of the MS image while containing high spatial resolution of the PAN image.

7.3.2 Fusion methods

There are various data fusion techniques mentioned in the literature that can perform fusion of data at either their lowest level (pixel for images) or at the decision level (label or class associated with data). We focus on image fusion rather than the broad data fusion problem. As mentioned above, there are three main categories of these methods.

Color related methods: These methods take advantage of various ways of displaying data in color. Some of these representations are RGB, IHS, YIQ. There are few variations in which these methods can be applied to data fusion [129]. In many cases the RGB technique is used with another fusion method such as IHS. IHS techniques can be applied to image fusion in different ways. In the *direct* approach, three image bands assigned to I, H, and S are transformed to RGB. In the *substitutional* approach, one of the I, H, or S bands, most commonly intensity band I, is replaced with another image with higher spatial resolution (Pan image). Then, the result is transformed back into the RGB space. In the *color contrast stretching* approach, hue and saturation bands are stretched before transforming them back to RGB space. This is usually done to enhance the fused image's color, and to reduce color distortion in result. For details of these display techniques and how they are applied to image fusion see [26, 50, 129]. These methods can fuse a limited number of images (usually three) after being registered and resampled.

Statistical/numerical methods: These methods deal with fusion of image data at the pixel level. They produce a fused image by applying some arithmetic or statistical operations on each input image and then assigning the result to the fused data set. These numerical methods vary depending on whether differences or similarities among data need to be captured. In some fusion methods, image data is treated as a signal from which one needs to extract either high or low frequency data. This class of fusion methods utilize techniques such as Brovey transform [129], Principal Component Analysis (PCA) [26, 129], arithmetic combination [129], High Pass Filters (HPF) [26, 129] and Wavelet Transformations [26, 113, 129, 167]. Most of these methods have the limitation that they require their input images be resampled and registered or that they can only fuse limited number of images at a time, or both.

Machine learning methods: These techniques are mostly applied when dealing with decision level fusion where a label need to be assigned to each pixel. Some of the machine learning tools used for image fusion are Support Vector Machines (SVM) [129], Hidden Markov Models (HMM) [91, 112, 129], Neural Networks [112, 129], and Decision Tree Classifiers (DTC) [112, 129].

7.4 Evaluation Methods

After performing data fusion, one needs to evaluate the quality of the results. There are various methods for evaluating the performance of a particular fusion technique. Some of these methods aim to measure spectral quality of the fused data

while others focus on spatial quality [157]. Some measures are based on human visual perception of the fused data [26]. In recent years such methods have been studied in conjunction with psychological issues involved in human visual perception of an image [40, 41]. We mention several image fusion evaluation methods.

We categorize image fusion evaluation techniques into three groups: *spatial quality metrics*, *spectral quality metrics*, and *joint spectral and spatial quality metrics*. Some spatial quality metrics are based on the edge analysis of the input and fused images. The idea is that edges that appear in the high spatial resolution image need to be present in the fused image. See [125] for more details.

Spectral quality metrics are based on various statistical measures which are calculated for the high spectral resolution input images and the fused result. Examples of these measures are correlation coefficient, relative shift in mean, variations in standard deviation, and (root) mean squared error [26, 157]. A fused image is considered to have a good spectral quality if relative to the image of high spectral resolution it is highly correlated, has relatively low shift in its mean value, and low (root) mean squared error.

Joint spectral and spatial quality metrics are a more active area of research. Obtaining an objective image fusion quality measure that considers both the spectral and spatial quality of the result is of great interest especially in the absence of the ground truth. When ground truth is present, one can evaluate the fusion method by comparing classification accuracy of the fused image and comparing it with that of each input image [157]. In the absence of ground truth, only a few statistical measures such as entropy and mutual information seem to be applicable. Entropy

measures the amount of increased information while mutual information represents a measure of relative entropy between the two sets [29, 132].

Wang and Bovik proposed a universal image quality index which later resulted in various image fusion measures both by themselves and other researchers [158]. They showed that their quality measure models distortion as combination of three factors: loss of correlation, luminance distortion, and contrast distortion. Wang and Bovid showed that their measure performs better than mean squared error since it also captures correlation between images in addition to their differences.

There are various fusion quality metrics introduced later based on the work of Wang and Bovik. Piella [126, 127] proposed a variant fusion quality metric which is a weighted average of image quality indices calculated for pairs of input and fused images. The weights for quality indices are chosen based on relevance of each image to the fused image. This relevance is calculated through a saliency measure such as contrast, variance, or amount of edges being transferred. Please see [126, 127] for more details.

While Piella's quality index was the first to measure both spectral and spatial quality of a fused image with respect to its two input images, it has its own limitations. First, it only can be used for grey level values. It is not clear how this measure can be used when more than two input images are used to obtain a fused image. Alparone *et al.* [8] modified this image quality index so that it can evaluate quality of four fused images with respect to a reference image. This partially alleviates problems mentioned above. They did so by using the concept of *hypercomplex correlation*. See [8] for more information.

Chapter 8

Cokriging as an Image Fusion Method

As mentioned in Chapter 7, the aim of image fusion is to integrate different data in order to obtain more information than can be derived from each of the single sensor data alone [129]. While methods like PCA and wavelet-based fusion have been traditionally used for image fusion, they have their own shortcomings. For example, PCA results are very sensitive to the selected area for fusion [129]. Both of these methods cannot handle fusion of scattered or rather sparse data, or fusion of images that differ greatly in either their spectral or spatial resolutions. In this chapter, we consider image fusion of remotely sensed data via cokriging as a solution to these limitations.

As we have seen earlier in this dissertation, cokriging is an interpolation method for scattered data (see Appendix A for details). Cokriging can integrate data of various sources with different spatial and spectral resolutions, or at arbitrary scattered locations. Unlike many other image fusion methods, cokriging does not require resampling of the data sets when registration of the remotely sensed data is being performed. This avoids introduction of errors due to rotation, translation, and interpolation of the data during the resampling process. Cokriging is also applicable to the vision of future sensor networks, where many small sensors are located at scattered locations. Using cokriging one can estimate sensor measurements for a

particular property at locations where those values are missing. For these reasons, we find cokriging suitable for addressing image fusion needs.

In this context, image fusion problem can be considered as an interpolation problem to estimate values at missing location for either low spatial or spectral resolution data. That is, we can perform fusion via cokriging either in the spatial or the spectral domain. We examine both problems of increasing the spatial and spectral resolution of images via cokriging. In Section 8.1 we perform fusion via cokriging in spatial domain. In particular, we increase the spatial resolution of Landsat's MS bands by fusing them with its PAN band. In Section 8.2 we demonstrate how cokriging can be utilized to improve the spectral resolution of the imagery data. In particular, we demonstrate increasing ALI's spectral resolution by fusing it with Hyperion.

8.1 Improving the Spatial Resolution Via Cokriging

We employed the cokriging interpolation method for image fusion of remotely sensed data [60, 80]. In particular, we show preliminary results on applying a variant called ordinary cokriging for pan-sharpening of multispectral images from the Landsat 7 sensor. We then evaluate both spectral and spatial quality of our fused images through a few quantitative measures. We also compare our results to those obtained from more traditional approaches based on principal component analysis and wavelets.

8.1.1 Data sets

We used Landsat 7 ETM data sets provided by the IEEE Data Fusion Committee, data set grss_dfc_0002 [1]. The images were taken over Hasselt (Belgium) in 1999. Landsat 7 ETM imagery has 8 bands. Landsat data specifications are presented in Table B.1 of Appendix B. Note that the spectral resolution of the panchromatic band 8 corresponds to MS bands 2, 3, and 4 combined. Thus, for our experiments, we used a 200×200 subset of multispectral bands 2, 3, and 4 and their corresponding 400×400 panchromatic band 8 which are shown in Figures 8.1 and 8.2 respectively.

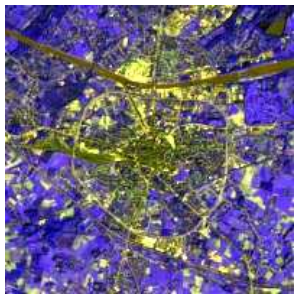


Fig. 8.1: Landsat 7 multispectral bands 2, 3, and 4. Landsat 7 image courtesy of ESA 1999 - distribution Eurimage.

8.1.2 Methods

We performed pan-sharpening of Landsat MS bands 2, 3, and 4 by fusing them with Pan band 8 using three different fusion methods: cokriging, principal component analysis (PCA), and wavelet-based fusion. In this section, we describe each method briefly.

Fig. 8.2: Landsat panchromatic band 8. Landsat 7 image courtesy ESA 1999 - distribution Eurimage.

8.1.2.1 Cokriging

In order to set up the ordinary kriging linear system, one needs to model pairwise covariances among available measurements. A requirement on these models is that they should generate a positive definite covariance matrix. A few covariance models are known to have this property (see [60, 80] for more details). We selected a few of these models with a limited number of parameters, and in each case we chose the one which best fit our data, which was spherical model with range 10. We performed our modeling and cokriging interpolation through a freely available software for interpolation of agro-climatic data [23]. For each query point, we considered its 32 nearest neighbors although different neighborhood sizes may result in better results. Cokriging interpolation and evaluation steps are computationally expensive tasks. For this reason, and because far points are expected to have less effect on interpolation weights, cokriging systems are traditionally solved over a lo-

cal neighborhood from the query point [60, 80]. Efficient implementations of these tasks will be the focus of our future research. Pan-sharpened MS bands 2, 3, and 4 (fused bands) by cokriging are shown in Figure 8.4.

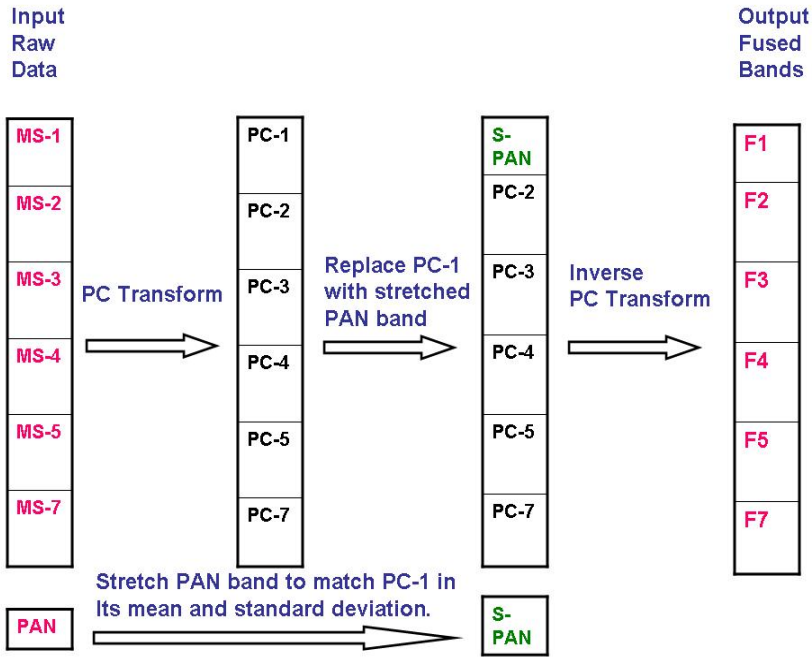
8.1.2.2 Principal component analysis (PCA)

We applied PCA for image fusion similarly to [26, 160]. First, we performed principal component transformation on Landsat multispectral bands. Then, the first principal component (PC) was replaced with the high resolution Pan band, which was scaled so that its mean and standard deviation match those of the first principal component of the MS bands. This scaling was performed to avoid distortion of the spectral information. Then, the first component was replaced by the stretched band. We then proceeded by performing inverse PCA on the stretched pan band and other PCs. Figure 8.3 demonstrates this method.

8.1.2.3 Wavelet-based fusion

A wavelet decomposition of any given signal (1-D or 2-D) is the process that provides a complete representation of the signal according to a well-chosen division of the time-frequency (1-D) or space-frequency (2-D) plane [33]. Through iterative filtering by low-pass and high-pass filters, it provides information about low- and high-frequencies of the signal at successive spatial scales. For fusion purposes, multi-resolution wavelet decomposition separates high- and low-frequency components of the two given data sets and these components are then recomposed differently in

Fig. 8.3: Pan-sharpening of Landsat MS bands with its PAN band through principal component analysis.



the reconstruction phase.

In our experiments, we are using a Daubechies filter [33] of size 4 and a Mallat Multi-Resolution Analysis (MRA) [103] decomposition and reconstruction scheme. Then, components from both decompositions are combined during the reconstruction phase to create the new fused data. In this scheme and similarly to [90], where different spatial resolution data are fused, we fuse the different spectral resolution data in the following manner: high-frequency information of the high spatial resolution data (e.g., Pan Landsat band 8) is combined with low-frequency information of the high spectral resolution data (e.g., Landsat MS bands). In our experiments, the same Daubechies filter of size 4 is used for both decomposition and reconstruction phases and for both types of data.

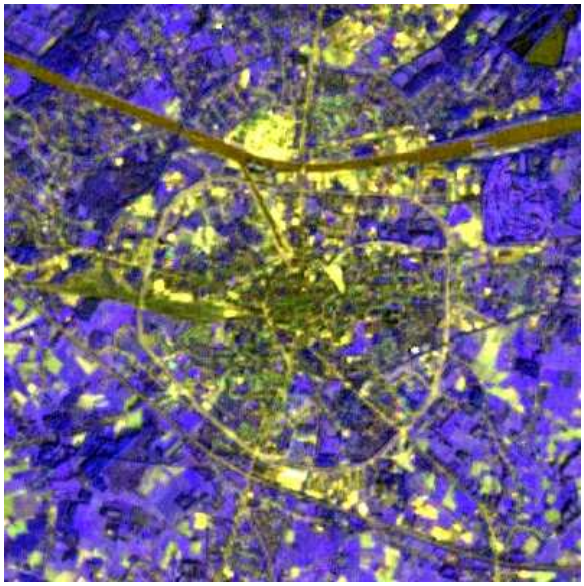


Fig. 8.4: Landsat Pan-sharpened MS bands 2, 3, and 4 through cokriging with Pan band 8

8.1.3 Evaluation

We increased the spatial resolution of Landsat ETM multispectral bands 2, 3, and 4 by fusing them with its panchromatic band 8. We performed fusion based on cokriging, PCA, and wavelets as described in the previous section. Next, we evaluated the quality of our results. Ideally, this evaluation would involve a comparison of the classification accuracy on ground-truth data. One could perform classification on input bands and fused bands respectively, assess the classification accuracy through ground truth in each case, and see which fused bands resulted in the most improvement of the classification accuracy. We evaluate our fusion methods through a few quantitative methods.

We evaluate both the spectral and spatial quality of our fused bands. The spectral quality was evaluated by calculating how highly each fused band is correlated with its corresponding input MS band. We expect the spectral quality of MS

bands to be preserved in the fused bands. Thus, the higher the correlation of the fused bands with their corresponding MS bands is, the better the spectral quality of the fusion. In order to evaluate the spatial quality of the fused bands we calculate the entropy of the multispectral input bands and their corresponding fused bands. The idea is that the fused images should have enhanced information content compared to their corresponding input MS bands. Because entropy is a measure of information content, the higher the entropy of the fused band as compared to its corresponding MS input band, the better the spatial quality of the fusion is.

In [108] we proposed using Haralick's texture quality metrics [73] as a fusion quality metric. The motivation for doing so is that an image with high textural information is more likely to result in better classification accuracy. Haralick [73] first proposed using a *co-occurrence matrix* to calculate various statistical texture properties for an image. A co-occurrence matrix calculates the number of occurrences of all pairs of gray level which are separated by a distance d along a given direction. From the co-occurrence matrix, several texture measurements can be computed among which are contrast, variance, and entropy. Usually co-occurrence matrices are calculated locally by considering a small window around each pixel. For each window, co-occurrence matrices are calculated along four directions. Then, a statistical measure (e.g., contrast, variance, entropy) is calculated for that local window. Then, the middle pixel of that window is replaced by the mean of the calculated statistical measure over all four directions. This is repeated for every pixel so that at the end of the process we have an image where each of its pixels is representing a statistical measure of its local neighborhood. We calculated entropy

images and then calculated the mean value of each of these images. Increase in mean of entropy images indicates increase in textural information contained in the image, which most likely causes better classification accuracy. However, the true evaluation criteria for our fusion methods would be through ground truth and comparing the classification results of original and fused bands against them.

8.1.4 Results

First we evaluate the spectral quality of fused images by calculating each fused band's correlation with its corresponding input MS band. Pairwise correlation of fused bands and their corresponding input MS bands are shown in Table 8.1. While PCA gives the best spectral quality results for bands 2 and 3, wavelet-based fusion performs best for band 4. However, we see that cokriging performs consistently for all bands and correlations of fused bands with all input MS bands exceeded 90% in all cases.

Table 8.1: Correlation of the fused bands with MS input bands

Bands	Wavelet	PCA	Cokriging
f_2, b_2	0.82	0.99	0.91
f_3, b_3	0.84	0.99	0.93
f_4, b_4	0.92	0.75	0.93
Average	0.86	0.91	0.92

As for spatial quality measures we considered both the overall entropy of images as well as the mean of entropy images calculated through local co-occurrence matrices [73]. The entropies of input MS bands and fused images are reported in Table 8.2, and the mean entropy of entropy images calculated through local co-

occurrence matrices are presented in Table 8.3. In both cases, cokriging results in increased spatial information compared to their corresponding MS bands. In all cases, cokriging performed better than wavelet-based fusion in increasing the spatial content of MS bands. PCA performed better in spatial domain for bands 3 and 4. However, cokriging performed more consistently overall in increasing spatial information of all MS bands. As we see in Table 8.2, cokriging resulted in higher average entropy of the fused bands compared to PCA and wavelet based fusion. Similarly, results in Table 8.3 indicate that PCA does not increase the textural information significantly for band 2. Cokriging performs more consistently in increasing the textural information across all bands. However, the overall textural information gained is comparable to that obtained from PCA.

Table 8.2: Entropy of the MS and fused bands

Original Bands		Fused Bands	Wavelet	PCA	Cokriging
b_2	2.68	f_2	3.12	2.69	3.23
b_3	3.01	f_3	3.28	3.72	3.64
b_4	3.44	f_4	3.93	5.21	4.90
Average	3.04		3.44	3.87	3.92

Table 8.3: Mean entropy of the entropy images obtained through co-occurrence matrices

Original Bands		Fused Bands	Wavelet	PCA	Cokriging
b_2	1.37	f_2	1.37	1.37	1.44
b_3	1.42	f_3	1.45	1.49	1.45
b_4	1.77	f_4	1.78	2.02	1.96
Average	1.52		1.53	1.63	1.62

8.1.5 Summary

Our experiments indicate that cokriging can be used as a fusion method for pan-sharpening of multispectral data. Methods like PCA or wavelet-based fusion are sensitive to particular wavelengths for preserving spectral resolution of MS bands or increasing their spatial information. Cokriging, on the other hand, performed consistently by producing fused bands that are more than 90% correlated with their corresponding MS input bands and that have significantly increased spatial information compared to their input MS bands. This effort only provides preliminary results on the applicability of cokriging to image fusion. There are various factors and parameters that can lead to better-quality fused images. These include having better models for pairwise covariances of data, and considering the best possible neighborhood size for interpolation of data. Evaluation of the results would also be more accurate if reference data were available. After submitting these results for publication [107], we learned that Pardo-Iguzquiza *et al.* independently and around the same time also used cokriging for the pan-sharpening of the remotely sensed imagery [123].

8.2 Improving the Spectral Resolution via Cokriging

We investigated the advantages of increasing ALI's spectral resolution via fusion with Hyperion. Our underlying motivation and application for this exercise involved the analysis of *invasive species* through a collaborative project among NASA Office of Earth Science and the US Geological Survey called Invasive Species Fore-

casting System (ISFS) [2, 146]. The data sets used in this section's experiments are from one of the four main Tamarisk study sites in Colorado.

While many vegetation types may appear to have the same color when viewed in the visible spectrum, they can be differentiated from each other when viewed in the infra-red [99]. Even when viewed in the non-visible spectrum, reflectance of these vegetation types may be of different degrees and from nearby portions of the spectrum. For this reason hyperspectral data is of great importance to the ISFS project. However, one would need to choose appropriate bands for a particular study based on the application.

We investigated candidate choices for Hyperion bands to be used for the ISFS project by learning the amount of detail that they introduce to the classification compared to their corresponding ALI bands. The fusion of ALI with Hyperion data was studied using PCA and wavelet-based fusion. We then utilized a geostatistical based interpolation method called cokriging for image fusion in spectral domain.

8.2.1 Data sets

Our data sets were acquired on July 5, 2004 from "Debeque" (near Grand Junction, Colorado) site which is one of the four study sites for ISFS's Tamarisk mapping effort [2]. The ALI and Hyperion are two instruments on the EO-1 platform. Hyperion is a hyperspectral instrument with 242 bands covering wavelengths ranging from 356 nm to 2577 nm at a spatial resolution of 30 meters per pixel. ALI on the other hand, has only 10 bands, one of which is panchromatic at 10 meters

spatial resolution and 9 of which are multispectral at a 30 meters spatial resolution, covering wavelengths ranging from 433 nm to 2350 nm. Thus, ALI data represent low spectral resolution data while Hyperion provides high spectral resolution images. ALI is considered a successor system to the Landsat Thematic Mapper series, and thus five of its multispectral bands' wavelengths correspond to that of Landsat 7. Hyperion is the only civilian hyperspectral instrument operating in space.

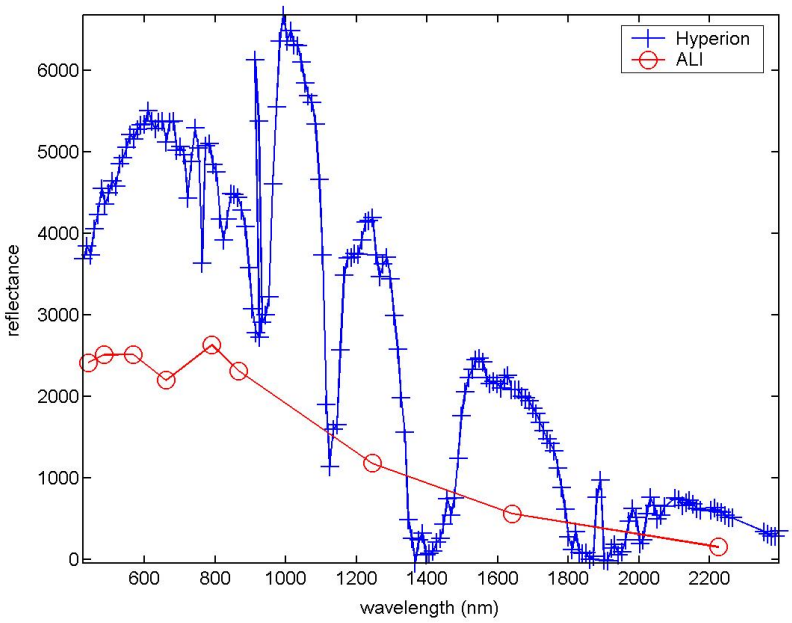
For this work, we used two data sets, one obtained from ALI and one from Hyperion instrument, containing approximately the same area. We had the 9 multispectral bands of ALI as well as all Hyperion bands for the region under study.

8.2.2 Experiments

We first performed fusion of ALI MS bands with Hyperion bands in spectral domain using PCA and wavelet-based fusion methods. Our objective for performing these two image fusion techniques on ALI and Hyperion images, one with low spectral resolution and one with high spectral resolution, was to study how much we can improve the quality of the classification performed on ALI MS bands using hyperspectral data for our application. Fusion results based on PCA and wavelets show that texture, measured through Haralick's texture quality metrics [73], can be improved through fusion, while preserving almost all the input original information. Haralick's texture quality metrics were used to validate the results and form the basis for a new fusion quality metric. Details of these experiments and their results are available in [108].

We then demonstrate how cokriging can be used to improve the spectral resolution of ALI. Looking at Table B.2 of Appendix B, we can see there are some wavelength ranges which are not covered by ALI data. In particular, considering only calibrated bands of Hyperion which did not seem visually corrupted, wavelengths covered by Hyperion bands 17, 26–27, 34–41, 46–48, 54–105, 116–140, 161–194, and bands 220–224 are not covered by ALI bands. Thus, one could create 8 new bands of ALI through cokriging, where each new ALI band will cover the missing intervals of the spectrum. Another fusion goal might also be to interpolate ALI only at a particular wavelength of interest, based on the application. Figure

Fig. 8.5: ALI and Hyperion reflectance in their spectral domain



8.5 shows the reflectance at one pixel both with the Hyperion and ALI sensors. Our experiments deal with estimating ALI values at missing intervals by using both ALI

and Hyperion, and investigate how we can mimic Hyperion's spectral signature at wavelengths of interest for ALI. This is done by first interpolating ALI at one wavelength location at each interval by estimating ALI values at wavelengths matching centers of wavelength ranges for Hyperion bands 17, 26, 37, 47, 77, 134, and 180. In the second experiment, we get a smoother ALI coverage, by estimating ALI at wavelength centers of Hyperion bands 17, 26, 37, 39, 47, 57, 77, 97, 130, 134, 138, 170, 180, and 190. Finally, we examine how cokriging would perform if we were to reconstruct ALI at centers of every single interval where we have Hyperion coverage. This demonstrates how cokriging performs for the spectral fusion of ALI with Hyperion. In practice, a user could specify an interval of interest for ALI coverage based on the application, and thus construct new fused bands for ALI.

In our preliminary results for the cokriging, Figures 8.6, 8.7, and 8.8 show that as we increase the number of wavelengths at which the ALI data is interpolated, we can construct ALI bands that have Hyperion's spectral signature while being within ALI's range of values. Of course, one will choose intervals of interest to perform this cokriging so that instead of dealing with 242 bands of Hyperion, or only nine bands of ALI, one can get a full spectrum coverage through about 17 ALI bands (nine original and about eight or more fused ALI bands).

8.2.3 Summary

Cokriging was used as an image fusion technique, and preliminary fusion experiments were performed with the intent of improving the spectral resolution of ALI

Fig. 8.6: Fusion by Cokriging: estimating one ALI value in center of each wavelength interval where ALI data is missing

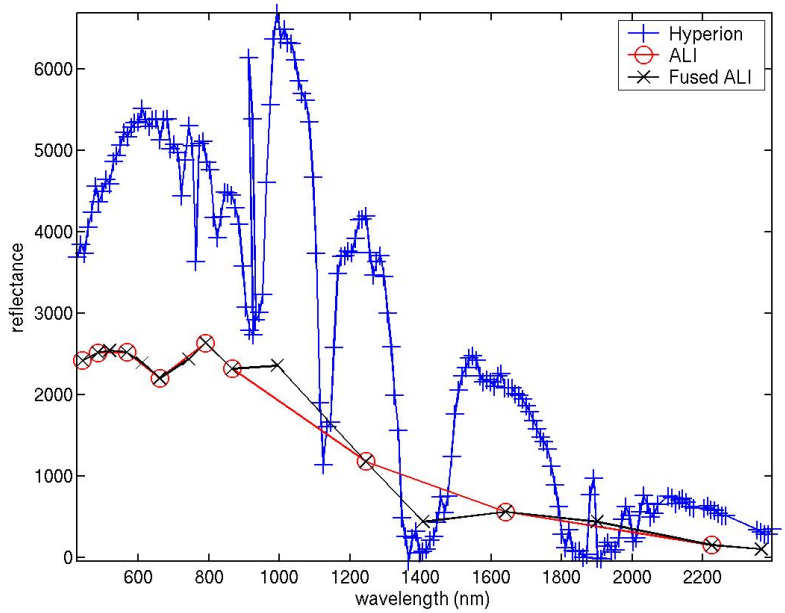


Fig. 8.7: Fusion by Cokriging: estimating up to three ALI values each wavelength interval where ALI data is missing

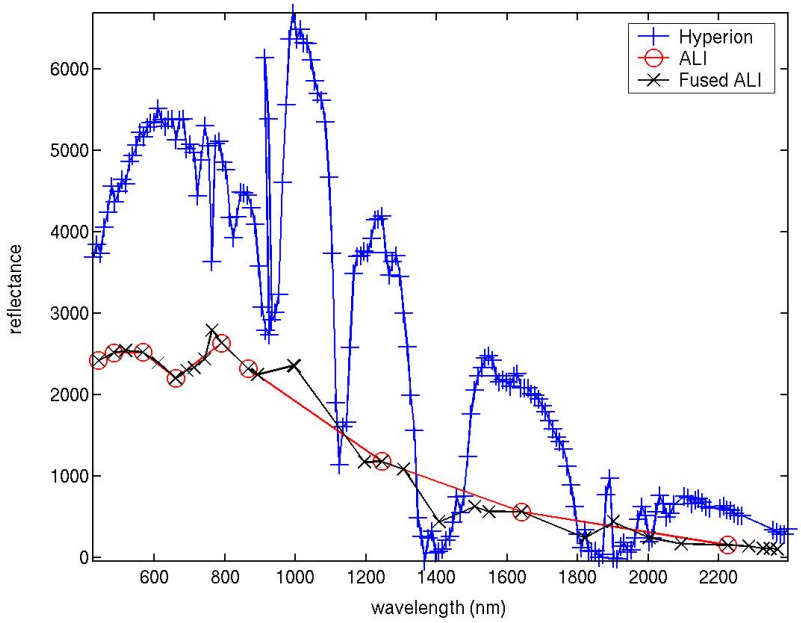
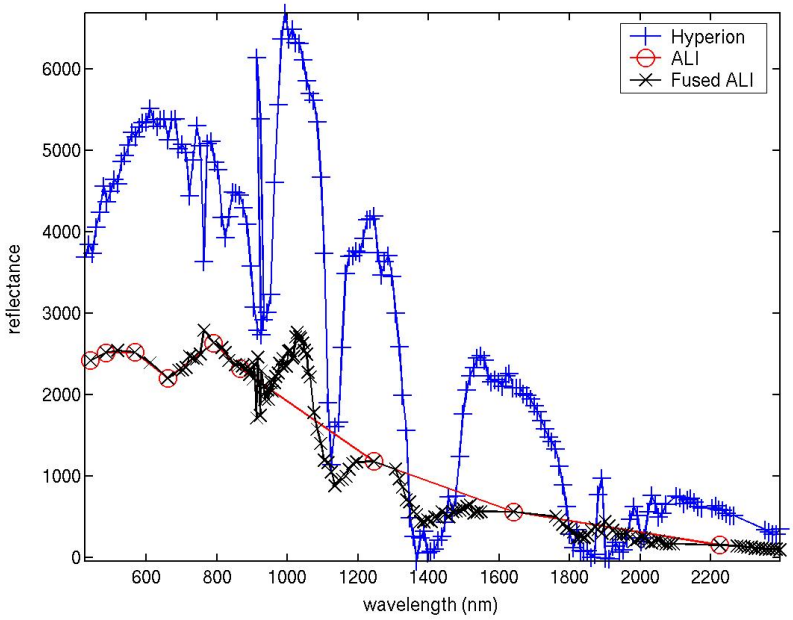


Fig. 8.8: Fusion by Cokriging: estimating ALI values in all Hyperion interval centers where ALI data is missing



data by fusing ALI with Hyperion data. Results show that new fused ALI bands can be created to have similar spectral pattern to that of the Hyperion's spectral signature. Estimated values were also within the range of ALI's measurements.[1]

[1]The results of this chapter are based on joint work with Jacqueline Le Moigne, David M. Mount, and Jeffrey T. Morisette [107, 108]. We thank the IEEE Data Fusion Committee, GRSS_DFC, for providing us the grss_dfc_0002 data set.

Chapter 9

Conclusions and Future Work

The focus of this dissertation has been on efficient implementations of computational tools for clustering and interpolation on large spatial data sets, and their application to the task of data fusion of remotely sensed data. Chapter 3 presented my work on efficient implementations of the ISODATA clustering algorithm, and Chapters 5 and 6 presented my work on efficient implementation of the ordinary kriging interpolation methods. A brief presentation of how the kriging interpolation method can be applied to image fusion problem of remotely sensed data was provided in Chapter 8. In this chapter, we will summarize these results and discuss future related work and open problems.

9.1 ISODATA Clustering Algorithm

In Chapter 3 we demonstrated the efficiency of a new implementation of the ISOCLUS algorithm based on the use of the kd-tree data structure and the filtering algorithm. This algorithm is a slight modification to the original ISOCLUS algorithm. We have provided both theoretical and experimental justification that our implementation yields essentially the same results. The experiments on synthetic clustered data showed speed-ups in running times up to roughly 60, while the experiments on Landsat and MODIS satellite image data showed speed-ups of roughly

4 to 30 and 4 to 20, respectively.

We also presented an approximate version of the algorithm which allows the user to further improve the running time at the expense of lower fidelity in computing the nearest cluster center to each point. We showed that with relatively small distortion errors, significant additional speed-ups can be achieved by this approximate version. The approximate version is most effective for data sets in high dimensions. The software is freely available, and can be downloaded from `http://www.cs.umd.edu/~mount/Projects/ISODATA`.

One possible direction for future work is analyzing the sensitivity of the algorithm to various parameters. While the kd-tree data structure with the filtering algorithm was first applied for efficient implementation of the k-means algorithm [87], the approximate filtering approach was first applied in our work. Thus, applying these methods, in particular the approximate filtering, for efficient implementations of other partitioning-based clustering algorithms (see Chapter 2) is another possible area of future work.

9.2 Kriging via Tapering

In Chapter 5 we implemented efficient ordinary kriging algorithms through utilizing covariance tapering [55] and iterative methods [118, 131]. We also implemented a variant of the global tapered method through projecting the large global system on to an appropriate smaller system. Global tapered methods resulted in memory saving factors ranging from 4 to 400 roughly for the storage of the coeffi-

cient matrix of the ordinary kriging system compared to the original global system. Global tapered iterative methods gave better estimation errors compared to the local approaches. In all cases, the estimation results of the global tapered method were very close to the global tapered and projected method. This is while global tapered and projected method solves the linear systems order(s) of magnitude faster than the global tapered method. This method can be viewed as a way of adaptively finding the correct neighbors to consider for the interpolation problem. Results of traditional local approaches depend on the underlying points' distribution, and whether or not enough points are included in the specified neighborhood.

This work has raised some open problems for possible future research. For the truncation approach, we showed the importance of knowing the smallest eigenvalue of two large matrices (see Section 5.1.1). The smallest eigenvalue of the covariance matrix is the upper bound for allowable change we can apply via truncation to this matrix to persevere the positive definiteness of the matrix. Also, the smallest eigenvalue of the system's coefficient matrix determines the maximum change we can introduce to the system to ensure the estimation error is below a desired threshold. Both of these matrices are very large, for which calculating the smallest eigenvalue involves solving a system as large as the original ordinary kriging system. However, the original kriging system was impractical to solve exactly to begin with. Finding the upper and lower bounds for the smallest eigenvalues of such large matrices is an open area of research. There exist methods for calculating or estimating the two largest eigenvalues of some graph's adjacency matrix [120], as well as estimating them via their upper and lower bounds [95]. While there has been some work on

bounding the smallest eigenvalue for some graphs [7], these bounds are not sufficiently tight nor do they apply to more general matrices. Therefore, this theoretical problem is an important area of research.

We also mentioned that empirical results support that the screening effect takes place (see Section 5.2). While there are some evidence of this hypothesis for gridded data [155], there are no theoretical results on this idea for general scattered data sets [55]. Formalizing and proving this hypothesis is an open mathematical question in the geostatistics domain.

9.3 Kriging via Fast Multipole Methods

Chapter 6 presented my work on efficient implementation of the ordinary kriging algorithm using a different approach. We integrated the efficient implementations of the Gauss Transform for solving ordinary kriging systems. Please note that while IFGT was used for cases where the covariance function is Gaussian, a similar approach can be used for other covariance functions where a factorization, similar to the one described for the Gaussian function in Section 6.4, exists. We also implemented another efficient variant, which we referred to as the GTANN approach. This method is effective when the covariance functions have small range values. As the number of data points increases, the IFGT approach yields higher speed-ups. Our experiments using the IFGT approach for solving the ordinary kriging system demonstrated speed-up factors ranging from 7-15 when using 5000 points. Based on our tests with varying number of data points, we expect even higher speed-ups

when using larger data sets. The IFGT method's estimations were comparable to the exact methods for large data sets. The GTANN approach outperformed the IFGT method in cases where the covariance functions used had very small values, resulting in speed-up factors of approximately 50–150. The GTANN approach is slower than the exact methods for large covariance ranges, and thus is not recommended to be used in these cases. Based on our experiments and those presented in [134], a safe cutoff value for determining whether to use GTANN or IFGT is a covariance range value of 50. The quality of results when using GTANN was comparable to the exact methods in all cases.

The IFGT approach was applicable to cases where the covariance function is Gaussian. Future work includes implementing the same approach for different covariance models (e.g. spherical and exponential). Also, the iterative method for solving the kriging system took more number of iterations to converge when using the IFGT approach compared to the GT approach. One can reduce the number of iterations and further improve the speed-ups obtained from the IFGT approach by first preconditioning [144] the linear systems. Preconditioning the kriging linear systems and studying the effect of various preconditioners for solving the system with different covariance models can be an immediate area for future work.

9.4 Image Fusion via Cokriging

Chapter 8 presented an application of the generalization of the kriging interpolation method, to the image fusion of remotely sensed data. Cokriging was applied

to address two problems in the image fusion domain. In one instance, cokriging was used for improving the spatial resolution of Landsat imagery (pan-sharpening). In the other case, the objective was to improve the spectral resolution of ALI data by fusing it with Hyperion via cokriging.

In Section 8.1, we performed pan-sharpening of the Landsat MS bands by fusing it with the Landsat Pan bands using cokriging. Methods like PCA or wavelet-based fusion are sensitive to particular wavelengths for preserving spectral resolution of MS bands or increasing their spatial information. Cokriging, on the other hand, performed consistently by producing fused bands that are more than 90% correlated with their corresponding MS input bands and that have significantly increased spatial information compared to their input MS bands. This effort only provides preliminary results on the applicability of cokriging to image fusion. Similarly in Section 8.2, we examined improving the spectral resolution of ALI data by fusing it with the Hyperion data. Preliminary fusion experiments were performed. Results show that new fused ALI bands can be created to have similar spectral pattern to that of the Hyperion spectral signature.

We demonstrated how kriging can be used for increasing the spatial or spectral resolution of satellite imagery. Future work includes applying kriging methods for image fusion of various sensors' data for different applications. Also, designing new quantitative metrics as well as improving the current measures for evaluating the quality of the fused image in the absence of a reference image is another possible direction for future work.

Appendix A

Cokriging and Kriging Interpolation Methods

In this Appendix we present necessary mathematical and statistical background for better understanding the cokriging interpolation method in general, and ordinary kriging in particular. Cokriging involves the solution of an optimization problem with an equality constraint. It also involves calculations of some statistical quantities such as covariance matrices, variograms, etc. This chapter is organized by first reviewing the necessary background material. Section A.1 reviews the mathematics involved in solving an optimization problem with equality constraints. Then, we present related statistical definitions and properties in Section A.2. In Section A.3, we explain what is meant by cokriging and present its derivation, assumptions, objectives, properties, and computational issues involved. Similarly in Section A.3.3, we describe the kriging problem in general, and the ordinary kriging problem. This appendix concludes by explaining why this problem is computationally expensive as well as introducing existing implementations of cokriging.

A.1 Mathematical Background for Solving Linear Systems

Let x be a vector in $\mathbb{R}^d$, and suppose we would like to minimize a function f subject to a linear constraint $Ax = b$, where x and b are d-vectors and A is a $n \times d$ matrix.

There are two approaches for solving this problem. First approach is as follows. Let Z be a basis for null space of A. Then, it is well known that if $\overline{x}$ is a solution to $Ax = b$, so is $\overline{x} + Zv$, where v is any d-vector. Thus, we can restate our optimization problem with an equality constraint as an unconstrained optimization problem as follows: $\min_v \ f(\overline{x} + Zv)$.

The second approach is that solving our minimization problem is equivalent to solving the following Lagrangian equation, $L(x, \lambda) = f(x) - \lambda^T c(x)$, where $c(x) = Ax - b$. Note that we are not minimizing the Lagrangian function. Rather we are finding a saddle point of this function. We will go over optimality conditions for a solution to our original problem and the meaning of Lagrange multipliers.

In terms of the first approach, necessary conditions for optimality are that the reduced gradient be zero and that the reduced Hessian be positive semidefinite [118]. That is: $Z^T \bigtriangledown f(x) = 0$ and $Z^T \bigtriangledown^2 f(x) Z$ is positive semidefinite. $Z^T \bigtriangledown f(x) = 0$ is also the sufficient condition for optimality.

In terms of the second approach, partial derivatives of the Lagrangian with respect to both x and λ must be zero. That is, first order necessary conditions for optimality are

$$\bigtriangledown_x L \ = \ \bigtriangledown f(x) - A^T \lambda \ = \ 0, \text{ and} \tag{A.1}$$

$$-\bigtriangledown_\lambda L \ = \ Ax - b \ = \ 0. \tag{A.2}$$

Suppose the solution to the above minimization problem is x^*. Also, suppose we have a point $\widehat{x}$ very close to x^* so that $||x^* - \widehat{x}|| \leq \epsilon$ and $A\widehat{x} = b + \delta$, where both ϵ and $||\delta||$ are very small. Then, we can approximate $f(\widehat{x})$ using Taylor series expansion.

Note that since $A\widehat{x} = b + \delta$ and $Ax^* = b$, we have $A(\widehat{x} - x^*) = \delta$.

$$\begin{aligned} f(\widehat{x}) &= f(x^*) + (\widehat{x} - x^*)^T g(x^*) + O(\epsilon^2) \\ &= f(x^*) + (\widehat{x} - x^*)^T A^T \lambda^* + O(\epsilon^2) \\ &= f(x^*) + \delta^T \lambda^* + O(\epsilon^2). \end{aligned} \tag{A.3}$$

This means that if we perturb b_i by δ_i, then optimal value is changed by $\delta_i \lambda_i^*$. Thus, λ_i is the change in the optimal objective per unit change in b_i. We say that, λ_i is the *sensitivity* of f to b_i [118]. For this reason Lagrange multipliers are also called *shadow prices* or *dual variables* [118].

A.2 Geostatistics Background

In this section we go over various statistical definitions. In order to do so, assume we are dealing with two random variables X and Y, such that X can take on the values $\{x_1, \ldots x_n\}$ and Y can take on the values $\{y_1 \ldots y_m\}$. Also let μ_x and μ_y denote the expected values of these variables.

A.2.1 Spatial analysis

When performing geostatistical modeling, some assumptions are usually made about the data. Some of the main assumptions made are defined below.

Stationarity Assumption means that the statistics of a random function are invariant under translation [60, 80, 151].

Isotropic Assumption means that data statistics are independent of direction.

Thus, for various statistics only the distance between the pairs of data points needs to be taken into account and not their orientation [60, 80, 151].

Anisotropic Assumption indicates that variability of data changes as a function of direction. Thus, for computation of data statistics both the distance and orientation between pairs of data points needs to be taken into account [60,80].

Intrinsic Hypothesis indicates that variance may be unbounded [151].

Quasi-stationarity implies that stationarity applies to a neighborhood of the data and not to the entire domain of data [151].

A.2.2 Covariance

Generally, the covariance between two random variables $x_i \in X$ and $y_j \in Y$ is defined as $Cov(X,Y) = E\left[(X-\mu_x)(Y-\mu_y)\right] = E(XY) - \mu_x\mu_y$. In geostatistics, two random variables z and z' of the same distribution are usually location dependent, that is, they are a function, let's say Z, of their locations. Let us denote these locations by u and u' respectively. Thus, $z = Z(u)$ and $z' = Z(u')$. In geostatistics, $C(u,u')$ is a shorthand for $Cov(Z(u), Z(u'))$, where $Z(u)$ and $Z(u')$ are values of a random function in locations u and u'. The covariance between two random variables $Z(u)$ and $Z(u')$ is defined as follows.

$$\begin{aligned} C(u,u') &= Cov(Z(u),Z(u')) = E\left[(Z(u)-E(Z(u)))(Z(u')-E(Z(u'))\right] \\ &= E(Z(u)Z(u')) - E(Z(u))E(Z(u')). \end{aligned} \tag{A.4}$$

The *stationary covariance*, $C(h)$, is defined as the covariance between two

random variables $Z(u)$ and $Z(u+h)$, separated in location by vector h:

$$\begin{aligned} C(h) &= Cov(Z(u+h), Z(u)) \\ &= E(Z(u+h)Z(u)) - (E(Z(u)))^2, \forall u, u+h \in A. \end{aligned} \tag{A.5}$$

A.2.3 Variance and standard deviation

For a variable X, variance is defined as follows:

$$Var(X) = Cov(X,X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)^2 = E\left[X - \mu_x\right]^2 = E(X^2) - \mu_x^2.$$

Recall the definition of stationary covariance function $C(h)$ from Section A.2.2. For $h = 0$, $C(0) = Cov(Z(u), Z(u)) = Var(Z(u))$. Thus, for a stationary random variable $Z(u)$, we have $C(0) = Var(Z(u))$. Standard deviation of a variable is defined as square root of its variance: $\delta(X) = \sqrt{Var(X)}$.

A.2.4 Correlation coefficient

Correlation coefficient, ρ, is a measure of linear relationship between two variables, or how close the values come to falling into a straight line. For two variables X and Y we have $\rho(x,y) = \frac{Cov(x,y)}{\delta(x)\delta(y)}$, where $\delta(x)$ and $\delta(y)$ are standard deviations of variable X and Y respectively. The *stationary* correlation coefficient, denoted $\rho(h)$, is defined as the correlation coefficient between values of random function Z at locations u and $u+h$, $\rho(h) = \rho(Z(u+h), Z(u)) \; \forall \; u, u+h$.

A.2.5 Variogram

Variogram is a measure of calculating spatial variability or dissimilarity between values of a random variables approximately separated by a vector h. This measure can be used as an alternative to measure $C(h)$, described above. For a set of points, variogram, or $2\gamma(h)$, is defined as follows ([37, 60, 80]):

$$2\gamma(h) = \frac{1}{N(h)} \sum_{h_{ij}=h} (x_i - x_j)^2, \tag{A.6}$$

where $N(h)$ is number of pairs separated by vector h, x_i and x_j are values of variables at two ends of the vector, and $h_{ij} = loc(x_i) - loc(x_j)$. Value of $loc(x_i)$ represents the location where x_i is measured, and $loc(x_j)$ indicates the location of x_j. Similarly, *semivariogram*, is defined as half of the average squared difference between two attributes separated by vector h:

$$\gamma(h) = \frac{1}{2N(h)} \sum_{h_{ij}=h} (x_i - x_j)^2. \tag{A.7}$$

We expect spatial variability of values of a random variable, or its variogram, to increase as the distance between locations of those values increase. However, after reaching a certain distance, this increase in variogram function stops. The distance at which variogram function stops increasing is called *range* of variogram, and the value that variogram has at distance equal to range is called the *sill*.

The sill value of the variogram is also the variance of the random function [80]. That is, $C(0) = \gamma(\infty)$. In other words the maximum variability of the random function values whose locations are far enough from each other is the same as maximum similarity among values of a random function evaluated at the same location. There are few points we need to consider when speaking of variogram and semivariogram.

1. Variogram and semivariogram are used interchangeably in practice. In fact, in most cases, a semivariogram as defined above is calculated, while for convenience it is referred to as variogram.

2. For a *stationary* random function variogram is defined as the variance of the increment between two random variables separated by vector h:

$$2\gamma(h) = Var(Z(u+h) - Z(u)), \quad \forall u, \tag{A.8}$$

where Z is a random function taking a location as its parameter.

A.2.6 Variogram modeling

As we will see in Section A.4, in order to perform cokriging, we need to model variograms. That is, we need to fit variogram values (see Section A.2.5) as function of distance h to a function which best fits it. Variograms are usually modeled so that we be able to model pairwise covariances as a function of distance h. For stationary data, having a variogram model $\gamma(h)$ allows us to come up with a covariance model $C(h)$, as a function of distance, using relation $C(h) = C(0) - \gamma(h)$ (see details in Section A.2.7, Property (3a)). In other words, once we have a model for variogram, the covariance model for distance h can be calculated by subtracting $\gamma(h)$ from variogram's sill value.

As we will see in Section A.3, to perform (co)kriging it is important that the involved covariance matrix C be positive definite. Thus, only models which will result in positive definite covariance matrices are considered for modeling variograms. Christakos showed necessary and sufficient conditions for permissible covariance

and variogram models [28]. Some of these permissible variogram models are as follows [30]:

Nugget effect model:

$$\gamma(h) = \begin{cases} 0 & \text{if} \quad h = 0, \\ c_0 + c(h) & \text{otherwise.} \end{cases} \tag{A.9}$$

Spherical model: This is the mostly used variogram model, where a represents the range of the variogram. This model is defined as follows:

$$\gamma(h) = \begin{cases} 0 & \text{if} \quad h = 0, \\ c_0 + c\{1.5\frac{h}{a} - 0.5(\frac{h}{a})^3\} & \text{if} \quad 0 < h \leq a, \\ c_0 + c & \text{if} \quad h \geq a. \end{cases} \tag{A.10}$$

Exponential model:

$$\gamma(h) = \begin{cases} 0 & \text{if} \quad h = 0, \\ c_0 + c\{1 - \exp(-\frac{3h}{a})\} & \text{otherwise.} \end{cases} \tag{A.11}$$

Gaussian model: This model is mostly used for extremely continuous values, and is defined as follows:

$$\gamma(h) = c_0 - \exp(-\frac{3h^2}{a^2}). \tag{A.12}$$

There are several issues that need to be considered when dealing with the mentioned models.

1. Usually the simplified version of above models are used where $c_0 = c = 1$, that is variogram values are normalized to have the sill value equal to one [60, 80].

2. Practical range a for Gaussian and exponential models is defined as the distance where variogram reaches 95% of its sill value [60, 80].

3. In some geostatistical literature, Gaussian and exponential models are defined without the factor 3, and thus in those cases variable a would be $\frac{1}{3}$ of the practical range [80].

4. Linear combination of acceptable variogram models is also an acceptable model.

A.2.7 Variogram properties

Above mentioned statistical quantities have several properties which we will review in this section [37, 60, 80].

1. Variance of a random variable created as linear combination of other random variables, $V_1 \dots V_n$, is estimated as follows (see [80], p. 216):

$$Var(\sum_{i=1}^{n} w_i V_i) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j Cov(V_i V_j),$$

where $w_1 \dots w_n$ are the weights associated with $V_1 \dots V_n$ respectively.

2. It is trivial to see for variograms that $\gamma(h) = \gamma(-h)$.

3. For a stationary random function, we have

(a) $\gamma(h) = C(0) - C(h)$.

$$\begin{aligned} 2\gamma(h) &= Var\left[Z(u+h) - Z(u)\right] \\ &= E\left[Z(u+h) - Z(u)\right]^2 - \left[E\left(Z(u+h) - Z(u)\right)\right]^2 \\ &= E\left[Z(u+h) - Z(u)\right]^2 - \left[E\left(Z(u+h)\right) - E\left(Z(u)\right)\right]^2 \\ &= E\left[Z(u+h) - Z(u)\right]^2 - 0 \qquad \text{(by stationarity)} \\ &= E\left(Z(u+h)\right)^2 + E\left(Z(u)\right)^2 - 2E\left(Z(u+h)\right)E\left(Z(u)\right) \\ &= 2E\left(Z(u)\right)^2 - 2E\left(Z(u+h)\right)E\left(Z(u)\right) \Longleftrightarrow \\ \gamma(h) &= E\left(Z(u)\right)^2 - E\left(Z(u+h)\right)E\left(Z(u)\right). \end{aligned}$$

We also have:

$$\begin{aligned} C(0) &= E(Z(u))^2 - \left[E(Z(u))\right]^2, \text{and} \\ C(h) &= E(Z(u+h)Z(u)) - \left[E(Z(u))\right]^2 \Longleftrightarrow \\ C(0) - C(h) &= E(Z(u))^2 - E(Z(u+h)Z(u)) \;=\; \gamma(h). \end{aligned}$$

Note that derivation of second definition for semivariogram requires Z be a stationary random function. That is, the mean, or expected value of its values are invariant under transformation of variables passed to it and is always constant.

This property allows one to easily derive the data's corresponding covariance function from its variogram function and vice versa. For example, the Spherical and Gaussian covariance functions (C_s and C_g respectively) are derived from Eq. (A.10) and (A.12), assuming $c_0 = 1$ as follows.

$$C_g(h) = \exp\left(-\frac{3h^2}{a^2}\right), \tag{A.13}$$

$$C_s(h) = \begin{cases} 1 & h = 0 \\ 1 - 1.5\frac{h}{a} + 0.5(\frac{h}{a})^3 & \text{if} \quad 0 < h \leq a, \\ 0 & \text{otherwise} \end{cases} \tag{A.14}$$

where a is the *range* for the covariance values, and h is the Euclidean distance of a pair of points. The range is the distance after which the covariance values remain constant at their lowest possible value. Please see [28, 60, 80] for other examples of permissible covariance functions.

(b) $\rho(h) = \dfrac{C(h)}{C(0)}$. Recall that

$$\begin{aligned} C(h) &= Cov(Z(u+h), Z(u)), \text{ and} \\ C(0) &= Cov(Z(u), Z(u)) = Var(Z(u)). \end{aligned}$$

Then, we have:

$$\begin{aligned} \rho(h) &= \rho(Z(u+h), Z(u)) = \frac{Cov(Z(u+h), Z(u))}{\delta(Z(u+h))\delta(Z(u))} \\ &= \frac{Cov(Z(u+h), Z(u))}{\delta(Z(u))^2} \quad \text{(by stationarity)} \\ &= \frac{Cov(Z(u+h), Z(u))}{Var(Z(u))} = \frac{C(h)}{C(0)}. \end{aligned}$$

(c) $\rho(h) = 1 - \dfrac{\gamma(h)}{C(0)}$. Using the previous property, we have

$$\rho(h) \;=\; \frac{C(h)}{C(0)} \;=\; \frac{C(0) - \gamma(h)}{C(0)} \;=\; 1 - \frac{\gamma(h)}{C(0)}.$$

A.3 Cokriging

Cokriging is multivariate version of kriging. A method for estimation that minimizes the variance of the estimation error by taking into consideration the spatial correlation between the variables of interest and the secondary variables [80]. In other words, a function U at location 0 is estimated as a *linear* combination of both the variable of interest and the secondary variables. That is, to estimate $\widehat{u}_0$, the estimate of U at location 0, as mentioned in [80], is given by

$$\widehat{u}_0 = \sum_{i=1}^{n} a_i u_i + \sum_{j=1}^{m} b_j v_j. \tag{A.15}$$

Note that $u_1, \ldots, u_n$ are primary data at n nearby locations, $v_1, \ldots, v_n$ are secondary data at m nearby locations, $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$ are cokriging weights which need to be calculated. Also, estimation error, R, can be calculated as

$$R = \widehat{U}_0 - U_0 = w^T Z, \tag{A.16}$$

where $w^T = (a_1, \ldots, a_n, b_1, \ldots, b_m, -1)$, and $Z^T = (U_1, \ldots, U_n, V_1, \ldots, V_m, U_0)$.

A.3.1 Mathematical formalization of the cokriging problem

The objective of cokriging is to find weights, vector w^T mentioned previously, such that the variance of the error be minimized and the estimate for $\widehat{U}_0$ be unbiased.

That is, the mean residual or error be equal to 0. Constraints that are imposed on the linear system ensure unbiasedness of the interpolant, and form various types of cokriging methods (see [60], page 204 and [80], chapter 17), few of which are the following.

Simple Cokriging: No constraints are imposed on the weights. Means of primary and secondary data are required. Simple cokriging considers that local means are known and constant through the study area.

Ordinary Cokriging: Imposes the following two constraints on coefficients: $\sum_{i=1}^{n} a_i = 1$ and $\sum_{j=1}^{m} b_j = 0$. This method limits the influence of the secondary variables greatly. As we will see, these conditions indicate that ordinary cokriging considers local means to be constant but unknown.

Standardized Ordinary Cokriging: is performed by creating new secondary variables so that they have the same mean as the primary variables. The constraint is that coefficients should add up to one: $\sum_{i=1}^{n} a_i + \sum_{j=1}^{m} b_j = 1$.

In Section A.3.4 we show how the above conditions on coefficients of the system ensures unbiasedness of the interpolant for each type of cokriging. Next, we describe how the weight coefficient for a cokriging system is found. We do so by forming and solving the ordinary cokriging problem. Setting up and solving other varieties of cokriging are then clear and very similar.

Variance of a random variable created as a linear combination of other random variables, $Z_1 \ldots Z_n$, is estimated as follows (see [80], p. 216):

$$Var\left(\sum_{i=1}^{n} w_i' Z_i\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_i' w_j' Cov\left(Z_i Z_j\right). \tag{A.17}$$

where $Z_1 \ldots Z_n$ are random variables at given locations, and $w_1' \ldots w_n'$ are the weights associated with them. Equations A.16 and A.17 imply the following objective function for minimizing the variance of the estimation error.

$$\begin{aligned} Var(R) &= w^T C_Z w \\ &= \sum_{i}^{n}\sum_{j}^{n} a_i a_j Cov(U_i U_j) + \sum_{i}^{m}\sum_{j}^{m} b_i b_j Cov(V_i V_j) \\ &+ 2\sum_{i}^{n}\sum_{j}^{m} a_i b_j Cov(U_i V_j) - 2\sum_{i}^{n} a_i Cov(U_i U_0) \\ &- 2\sum_{j}^{m} b_j Cov(V_j U_0) + Cov(U_0 U_0). \end{aligned} \tag{A.18}$$

One way of ensuring unbiasedness for our estimation $\widehat{U}_0$ is to require $\sum_{i=1}^{n} a_i = 1$ and $\sum_{j=1}^{m} b_j = 0$ (we will show in Section A.3.4). So now we have an optimization problem with two constraints. This is where we take advantage of Lagrange multipliers (see Section A.1). Let our Lagrange multipliers be μ_1 and μ_2. Then, we are trying minimize $Var(R)$ subject to two mentioned constraints by solving for coefficients $a_1 \ldots a_n, b_1 \ldots b_m, \mu_1, \mu_2$, where

$$Var(R) = w^T C_Z w + 2\mu_1(\sum_{i=1}^{n} a_i - 1) + 2\mu_2(\sum_{j=1}^{m} b_j). \tag{A.19}$$

The next step is taking partial derivatives of the above equation with respect to all $n + m$ cokriging variables and the two Lagrange multipliers and setting them to zero. Then, we will get the following $n + m + 2$ equations to solve:

$$\sum_{i=1}^{n} a_i Cov(U_i U_j) + \sum_{i=1}^{m} b_i Cov(V_i U_j) + \mu_1 = Cov(U_0 U_j) \quad (j = 1...n), \text{ (A.20)}$$

$$\sum_{i=1}^{n} a_i Cov(U_i V_j) + \sum_{i=1}^{m} b_i Cov(V_i V_j) + \mu_2 = Cov(U_0 V_j) \quad (j = 1...m), \text{ (A.21)}$$

$$\sum_{i=1}^{n} a_i = 1, \text{ and} \quad \text{(A.22)}$$

$$\sum_{i=1}^{m} b_i = 0. \quad \text{(A.23)}$$

Equivalently:

$$\begin{pmatrix} C_{u_1u_1} & \dots & C_{u_nu_1} & C_{v_1u_1} & \dots & C_{v_mu_1} & 1 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 1 & 0 \\ C_{u_1u_n} & \dots & C_{u_nu_n} & C_{v_1u_n} & \dots & C_{v_mu_n} & 1 & 0 \\ C_{u_1v_1} & \dots & C_{u_nv_1} & C_{v_1v_1} & \dots & C_{v_mv_1} & 0 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 & 1 \\ C_{u_1v_m} & \dots & C_{u_nv_m} & C_{v_1v_m} & \dots & C_{v_mv_m} & 0 & 1 \\ 1 & \dots & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & \dots & 0 & 1 & \dots & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_1 \\ \dots \\ a_n \\ b_1 \\ \dots \\ b_m \\ \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} C_{u_0u_1} \\ \dots \\ C_{u_0u_n} \\ C_{u_0v_1} \\ \dots \\ C_{u_0v_m} \\ 1 \\ 0 \end{pmatrix} \quad \text{(A.24)}$$

Once the above system of equations is solved, we have necessary coefficients $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m$ to estimate function U at location 0.

A.3.2 Generalized cokriging system

One can see that instead of having one set of secondary variables $V_1 \dots V_m$, we may use multiple sets of secondary variables. Each additional set of secondary variables $W_1 \dots W_k$ will introduce a new set of coefficients $c_1 \dots c_k$ and a new Lagrange multiplier μ_w.

For the general case where we have s set of variables (as opposed to just two sets, one primary and one secondary), our linear system will be as follows:

$$\begin{pmatrix} C & L \\ L^T & 0 \end{pmatrix} \begin{pmatrix} T \\ \mu \end{pmatrix} = \begin{pmatrix} C_0 \\ I_0 \end{pmatrix}. \quad \text{(A.25)}$$

Where C is the covariance (or its estimate) matrix of all known variables' pair, and C_0 is the vector of pairwise covariances between the unknown variable U_0 and all other known variables. The μ entry is the vector of all Lagrange multipliers $\mu_1 \dots \mu_s$. L is a vector of matrices $I_1 \dots I_s$. Each matrix $I_i, i \in \{1 \dots s\}$ is of size $n_i \times s$, where n_i is the number of points in i^{th} variable set . All elements in the i^{th} column of I_i are one and all other entries are zero. The T entry is the vector of all coefficients, and I_0 is a column vector of size $s \times 1$ of all elements under C_0 on the right hand side of the equation. Similarly to ensure unbiasedness, this vector is made of a 1 on top and all zeros for the rest of entries. It can also be proven that in order for the system to have a solution, we need matrix C to be positive definite [60, 80].

A.3.3 Mathematical formalization of the kriging problem

Kriging is a special case of cokriging where we estimate value of a variable at a location using only values of the same variable at scattered points around it. That is, to estimate the value of a random function U at location 0, u_0, using values of U at n other locations $(u_1, \dots u_n)$ we need to calculate the coefficients $a_1, \dots a_n$ such that $\widehat{u}_0 = \sum_{i=1}^{n} a_i u_i$ and variance of the error be minimized.

Similarly we have *Simple* and *Ordinary Kriging*, depending on whether or not local means are known and/or constant or not (see Section A.3). Simple kriging does not require any constraints on coefficients while ordinary kriging requires sum of coefficients to add up to one for insuring unbiasedness (see Section A.3.4 for details). Thus, for ordinary kriging, Eq. (A.3.1) simplifies as follows:

$$\sum_{i=1}^{n} a_i Cov(U_i U_j) + \mu_1 = Cov(U_0 U_j) \quad (j = 1...n),$$

$$\sum_{i=1}^{n} a_i = 1.$$

Equivalently:

$$\begin{pmatrix} C_{u_1u_1} & \dots & C_{u_nu_1} & 1 \\ \dots & \dots & \dots & 1 \\ C_{u_1u_n} & \dots & C_{u_nu_n} & 1 \\ 1 & \dots & 1 & 0 \end{pmatrix} \begin{pmatrix} a_1 \\ \dots \\ a_n \\ \mu_1 \end{pmatrix} = \begin{pmatrix} C_{u_0u_1} \\ \dots \\ C_{u_0u_n} \\ 1 \end{pmatrix}. \tag{A.26}$$

A.3.4 Unbiasedness condition

For kriging and cokriging interpolation methods, it is often required that estimates obtained via interpolation be *unbiased*. Unbiasedness of the estimator means that expected value of error should be zero.

Lemma A.3.1 (Isaaks and Srivastava [80]) *A kriging estimate of an stationary variable is unbiased iff sum of its kriging weights is 1.*

Suppose we have a random function $V(x)$, where x is a location. Assume that for points x_1 to x_n we know value of function V, and we would like to estimate V at an unknown location x_0 as a linear combination of n known function values so that our estimate be unbiased (this is the case in kriging).

Let $\widehat{V}$ be our estimate function, R the error associated with our estimate, and $v_1 \dots v_n$ be weights given to each known function value in estimation. Then we have

$$\begin{aligned} \widehat{V}(x_0) &= \sum_{i=1}^{n} v_i V(x_i), \text{ and} \\ R(x_0) &= \widehat{V}(x_0) - V(x_0) = \sum_{i=1}^{n} v_i V(x_i) - V(x_0). \end{aligned}$$

Unbiasedness condition states

$$\begin{aligned} E\left(R(x_0)\right) &= 0 \iff \\ E\left(\sum_{i=1}^{n} v_i V(x_i) - V(x_0)\right) &= 0 \iff \\ E(V)\sum_{i=1}^{n} v_i - E(V) &= 0 \iff \\ E(V)\left[\sum_{i=1}^{n} v_i - 1\right] &= 0. \end{aligned}$$

This indicates that we need to have $\sum_{i=1}^{n} v_i = 1$ to assure unbiasedness of our estimate, and this is often one of the constraints in optimization problems that we end up solving for our interpolation methods [80]. It is easy to similarly derive the necessary conditions for ensuring unbiasedness in ordinary cokriging. In this case, we estimate a random function value as a linear combination of values of more than one random function at each point. Let our secondary random function be W. Also, assume we know values of W at points $x_1 \dots x_m$. Then,

$$\begin{aligned} \widehat{V}(x_0) &= \sum_{i=1}^{n} v_i V(x_i) + \sum_{i=1}^{m} w_i W(x_i), \text{ and} \\ R(x_0) &= \widehat{V}(x_0) - V(x_0) = \sum_{i=1}^{n} v_i V(x_i) + \sum_{i=1}^{m} w_i W(x_i) - V(x_0). \end{aligned}$$

Unbiasedness condition states

$$
\begin{aligned}
E\left(R(x_0)\right) &= 0 \Longleftrightarrow \\
E\left(\sum_{i=1}^{n} v_i V(x_i) + \sum_{i=1}^{m} w_i W(x_i) - V(x_0)\right) &= 0 \Longleftrightarrow \\
E(V)\sum_{i=1}^{n} v_i + E(W)\sum_{i=1}^{m} w_i - E(V) &= 0 \Longleftrightarrow \\
E(V)\left[\sum_{i=1}^{n} v_i - 1\right] + E(W)\sum_{i=1}^{m} w_i &= 0.
\end{aligned}
$$

By requiring $\sum_{i=1}^{n} v_i = 1$ and $\sum_{i=1}^{m} w_i = 0$ we ensure that the above equation and thus our unbiasedness condition holds. Similarly, if we use more additional functions in ordinary cokriging, we need to require sum of coefficients of values in our linear combination obtained from each particular additional function to be equal to zero. Standardized cokriging assumes that all random functions used in our estimation process have the same mean. That is, in above equation we have $E(V) = E(W)$. This condition results in reducing our number of constraints from s to 1, where s is the number of random functions used in our linear interpolation.

$$
\begin{aligned}
E(V)\left[\sum_{i=1}^{n} v_i - 1\right] + E(W)\sum_{i=1}^{m} w_i &= 0 \Longleftrightarrow \\
E(V)\left[\sum_{i=1}^{n} v_i - 1\right] + E(V)\sum_{i=1}^{m} w_i &= 0 \Longleftrightarrow \\
E(V)\left[\sum_{i=1}^{n} v_i + \sum_{i=1}^{m} w_i - 1\right] &= 0.
\end{aligned}
$$

Thus, in standardized cokriging we require $\sum_{i=1}^{n} v_i + \sum_{i=1}^{m} w_i = 1$. Similarly, if more random functions are involved, we require sum of all coefficients in our linear combination be one.

A.3.5 Positive definiteness condition

A requirement for cokriging system to have a solution is that its covariance/variogram matrix, lets call it K, needs to be positive definite.

Lemma A.3.2 (Isaaks and Srivastava [80]) *The variance of the ordinary kriging estimation error is positive if C is positive definite.*

Lemma A.3.3 (Myers [115]) *The ordinary kriging system described in Eq. (A.25) has a solution if C is positive definite.*

The first lemma is easy to show. As seen above, the error involved in cokriging result can be written as follows:

$$R \;=\; \widehat{U}_0 - U_0 \;=\; w^T Z$$

$$Var(R) \;=\; w^T C_Z w.$$

Requiring $Var(R)$ to be positive is the same as requiring $C = C_z$ to be positive definite: $w^T C_z w > 0$.

Recall the generalized cokriging system from Eq. (A.25). Correctness of Lemma A.3.2 can be shown by proving its contrapositive statement: If the system does not have a solution, then C not positive definite [115]. Let

$$G = \begin{pmatrix} C & L \\ L^T & 0 \end{pmatrix}. \tag{A.27}$$

If G is not invertible, it means that there exist non-zero U and V such that

$$\begin{pmatrix} C & L \\ L^T & 0 \end{pmatrix} \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{A.28}$$

Thus,

1. $CU + LV = 0 \implies U^T CU + U^T LV = 0$

2. $L^T U = 0 \implies U^T L = 0$

(1) and (2) $\implies U^T CU = 0 \implies C$ is not positive definite. Notice that this positive definiteness condition on C is different from positive definiteness condition for insuring optimality of a minimization/optimization problem. Here we are just insuring that the coefficient matrix is invertible while in an optimization problem we require Hessian of the system to be positive definite as described in Section A.1.

A.4 Algorithmic Approach

Estimating an unknown variable via (co)kriging involves the following steps.

1. Setting up the linear system as mentioned in its most general form in Section A.3.2 satisfying conditions mentioned in Sections A.3.4 and A.3.5.

2. Solving the linear system for coefficients.

3. Evaluating the estimation for the unknown variable.

The main task in the first step is calculating elements of C and C_0. Elements of C are pairwise covariances between random variables for which we have only one value. That is, we know values of a primary variable at n points $(u_1, \ldots u_n)$, and we are treating each of these values as an instance of a random variable U_1. Thus, for C_{ij} we need covariance functions of the two random variables U_i and U_j that generated values u_i and u_j at i^{th} and j^{th} points respectively. Here is where variograms come into the picture. Variograms between pairwise variables are functions of distances

between the two particular samples. Notice that for a *stationary* random variable, once we know variogram values between variables at a pair of points separated by distance h, it is easy to calculate their pairwise covariances as well (see Section A.2.7, Property (3a)). Also, one can transform anisotropic data to data which is isotropic (stationary) [60, 80].

Usually variograms are modeled as a function of distance between points, and so are the covariances. After fitting various possible models to calculated variograms, the best one which gives the least error is picked and used to model the pairwise variograms. Having a general variogram function $\gamma(h)$, we can obtain a covariance function $C(h)$ using equation in Section A.2.7, Property (3a).

At this point, we have a modeled variogram and covariance, $\gamma(h)$ and $C(h)$ as a function of distance between two samples of random variables. Then, matrices C and $C0$ in Section A.3.2 are generated as follows: for every element c_{ij} in C or $C0$, calculate distance h_{ij} between point labeled i and the one labeled j. Then simply calculate $c_{ij} = C(h_{ij})$.

In addition to just calculating matrix C, we need to make sure that it is positive definite (see Section A.3.5). Instead of checking for this condition every time a model is picked, we limit our selection of variogram models to only those functions which will lead to a positive definite matrix C (see Section A.2.6). Also, it is important that the model we choose for our variogram be bounded so that we be able to calculate $C(h)$ given $\gamma(h)$ for a given distance h using equation in Section A.2.7, Property (3a).

Now that we are done with first step, and have our linear system set up, we can solve it for the coefficients. There are a variety of well-known methods for solving a linear system of equations [118], and once the solution to the system in Section A.3.2 is known, evaluating the unknown variable is just a matter of substituting values and calculating $\widehat{u}_0$ as mentioned in Section A.3.

Another advantage of kriging methods over other interpolation methods is that one can calculate the variance of the estimation error without having the true values. We mentioned that the variance of the error for ordinary cokriging is Eq. (A.18). Using Eq. (A.20–A.23), we have:

$$\begin{aligned}
\sum_{i=1}^{n} a_i Cov(U_i U_j) + \sum_{i=1}^{m} b_i Cov(V_i U_j) &= -\mu_1 + Cov(U_0 U_j) \\
\sum_{i=1}^{n} a_i Cov(U_i V_j) + \sum_{i=1}^{m} b_i Cov(V_i V_j) &= -\mu_2 + Cov(U_0 V_j)
\end{aligned}$$

Substituting the above in Eq. (A.18) gives us the following simplified version for variance of error in ordinary cokriging:

$$\begin{aligned}
Var(R) &= \sum_{i}^{n} a_i Cov(U_i U_0) - \mu_1 + \sum_{j}^{m} b_j Cov(U_0 V_j) \\
&- 2\sum_{i}^{n} a_i Cov(U_i U_0) - 2\sum_{j}^{m} b_j Cov(V_j U_0) + Cov(U_0 U_0) \\
&= Cov(U_0 U_0) - \mu_1 - \sum_{i}^{n} a_i Cov(U_i U_0) - \sum_{j}^{m} b_j Cov(V_j U_0).
\end{aligned}$$

If we do not consider constraints imposed in Eq. (A.25), we obtain the following general equation for variance of the error for the cokriging system which can then be simplified based on conditions imposed on coefficients. $Var(R) = Cov(U_0 U_0) - \sum_i^n a_i \mu_1 - \sum_j^m b_j \mu_2 - \sum_i^n a_i Cov(U_i U_0) - \sum_j^m b_j Cov(V_j U_0)$.

A.5 Past Implementations

There are various software which support kriging and/or cokriging interpolation methods. Here we list the most commonly used ones.

GSLIB: Geostatistical Software Library [37]: This library is written in Fortran by Deutsch and Journal, and has support for both kriging and cokriging.

Cokriging in Matlab [105]: This program supports cokriging in two and three dimensional space. The program makes calls to Fortran executables.

C Implementation [23]: This software supports variogram modeling and cokriging of data in two dimensional space only.

GsTL: Geostatistical Template Library [137]: This is a library written in C++ which supports various geostatistical algorithms including kriging and cokriging.

S-GeMS: The Stanford Geostatistical Modeling Software [138]: This software is implemented in C++, and utilizes the GsTL library. This software can perform three dimensional geostatistical modeling, and has a graphical user interface as well.

A.6 Error Analysis

There are three different sources of error that need to be considered in kriging problems using iterative methods.

- ϵ_1: the kriging error, which is the difference between the estimated value and the true value, Eq. (A.16).

- ϵ_2: the absolute or relative residual that the iterative solver calculates. This value is usually used as the convergence criteria for iterative methods.

- ϵ_3: error involved in solving the kriging system Eq. (A.25).

First error indicates the best that kriging (or any other interpolation method) can do if the system is solved exactly. The second error represents the solution's accuracy level that we are requiring, either for the absolute or residual error. The third error represents the limitations of the machine precision and errors that arise in solving an approximate system. Suppose we are solving a linear system

$$Ax = b, \tag{A.29}$$

and instead we end up finding an approximate solution, $\widehat{x}$, which is the answer to a system of the form

$$(A + E)\widehat{x} = b + f. \tag{A.30}$$

We would like to analyze the relative error of the approximate solution $\widehat{x}$ satisfying Eq. (A.30) with respect to the exact solution x satisfying Eq. (A.29). This error can be introduced either due to limitations of the machine precision or the way we set up the linear system (as we described in Section 5.1.1). To analyze ϵ_3, we first need to introduce the notion of the *condition number* for a matrix A.

Definition: Let A be an $n \times m$ matrix. Then, the condition number of A with respect to the particular norm used, $\kappa(A)$, is defined as follows [121]:

$$\kappa(A) = \begin{cases} \|A\| \cdot \|A^{-1}\| & \text{if } A \text{ is nonsingular,} \\ +\infty & \text{otherwise.} \end{cases} \tag{A.31}$$

According to the Perturbation Lemma, also known as Banach Lemma [121], if $\|A^{-1}\|\|E\| < 1$, then $A + E$ is nonsingular and

$$\| (A+E)^{-1} \| \leq \left(\frac{\|A^{-1}\|}{1 - \|A^{-1}\|\|E\|} \right). \tag{A.32}$$

Assuming that both A and $A+E$ are nonsingular, and that x and $\widehat{x}$ satisfy Equations A.29 and A.30 respectively, we have

$$\begin{aligned} x - \widehat{x} &= A^{-1}b - (A+E)^{-1}(b+f) \\ &= (A+E)^{-1}\left((A+E)A^{-1}b - (b+f)\right) \\ &= (A+E)^{-1}\left((b + EA^{-1}b - b - f\right) \\ &= (A+E)^{-1}(Ex - f). \end{aligned} \tag{A.33}$$

From Eq. (A.29) we have:

$$\|b\| \leq \|A\|\|x\| \Longrightarrow \|x\| \geq \|A\|^{-1}\|b\|.$$

Equations (A.32) and (A.33), give the following expression for the relative error of our estimated solution:

$$\begin{aligned} \frac{\|x - \widehat{x}\|}{\|x\|} &\leq \| (A+E)^{-1} \| (\|Ex\| + \|f\|) \frac{1}{\|x\|} \\ &\leq \| (A+E)^{-1} \| \left(\frac{\|E\|\|x\|}{\|x\|} + \frac{\|f\|}{\|x\|} \right) \\ &\leq \| (A+E)^{-1} \| \left(\|E\| + \|A\| \frac{\|f\|}{\|b\|} \right) \end{aligned} \tag{A.34}$$

Using the above equation and Eq. (A.32), we get

$$\begin{aligned}\frac{\|x-\widehat{x}\|}{\|x\|} &\leq \left(\frac{\|A^{-1}\|\|A\|}{1-\|A^{-1}\|\|E\|}\right)\left(\frac{\|E\|}{\|A\|}+\frac{\|f\|}{\|b\|}\right) \\ &\leq \left(\frac{\kappa(A)}{1-\kappa(A)\frac{\|E\|}{\|A\|}}\right)\left(\frac{\|E\|}{\|A\|}+\frac{\|f\|}{\|b\|}\right). \end{aligned} \tag{A.35}$$

Let

$$\alpha = \kappa(A)\left(\frac{\|E\|}{\|A\|}\right) = \|A^{-1}\|\|E\| < 1. \tag{A.36}$$

Thus, the relative error estimate reduces to

$$\frac{\|x-\widehat{x}\|}{\|x\|} \leq \frac{\kappa(A)}{1-\alpha}\left(\frac{\|E\|}{\|A\|}+\frac{\|f\|}{\|b\|}\right). \tag{A.37}$$

A.7 Computational Challenges and Solutions

As we discussed in this Appendix, estimating an unknown value using (co)kriging requires us to first set up the (co)kriging linear system (see Section A.3.2). This includes the problem of fitting data values to acceptable variogram models, which results in a positive definite covariance matrix for the linear system we need to solve. Then, we solve a linear system. Finally, the unknown value is estimated as the linear combination of known values by using the weights that were obtained from solving the linear system (see Section A.3).Factors that make (co)kriging computationally expensive are large sizes of the linear systems involved for large data sets and the fact that we cannot reuse weights obtained for one query point for another estimation.

Appendix B

Satellite Data Specification

Table B.1: Landsat 7 ETM data specification

Band	Resolution	
	Spatial (meters)	Spectral (μm)
1	30	0.45–0.52
2	30	0.53–0.61
3	30	0.63–0.69
4	30	0.78–0.90
5	30	1.55–1.75
6	60	10.4–12.5
7	30	2.09–2.35
8	15	0.52–0.90

Table B.2: ALI bands and the corresponding calibrated and not corrupted Hyperion bands used

ALI MS Bands	Spectral Range (nm)	CWL (nm)	Matching Used Hyperion Bands	CWL (nm)
1 (MS-1')	433–453	441.6	9 **10**	436.99 447.17
2 (MS-1)	450–515	484.8	11 ... **14** 15 16	457.34 ... 487.87 498.04 508.22
3 (MS-2)	525–605	567.2	18 ... **22** 23 ... 25	528.57 ... 569.27 579.45 ... 599.80
4 (MS-3)	630–690	660	28 ... **31** ... 33	630.32 ... 660.85 ... 681.200
5 (MS-4)	775–805	790	42 43 **44** 45	772.78 782.95 793.13 803.30
6 (MS-4')	845–890	865.6	49 50 **51** ... 53	844.00 854.18 864.35 ... 884.70
7 (MS-5')	1200–1300	1244.4	106 ... **110** ... 115	1205.07 ... 1245.36 ... 1295.86
8 (MS-5)	1550–1750	1640.1	141 ... **149** ... 160	1558.12 ... 1638.81 ... 1749.70
9 (MS-7)	2080–2350	2225.7	195 ... 202 204 ... **207** 208 211	2102.94 ... 2173.53 2193.73 ... 2224.03 2234.12 2264.32

Bibliography

[1] Data fusion community. `http://www.dfc-grss.org`. January 2006.

[2] Invasive species forecasting system. `http://bp.gsfc.nasa.gov/isfs.html`. April 2007.

[3] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. A survey: `http://valis.cs.uiuc.edu/~sariel/research/papers/04/survey`, February 2005.

[4] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 658–667, San Francisco, CA, January 1998.

[5] P. K. Agarwal, M. Sharir, and E. Welzl. The discrete 2-center problem. *Discrete & Computational Geometry.*, 20(3):287–305, 1998.

[6] P. Alfeld. Scattered data interpolation in three or more variables. *Mathematical methods in computer aided geometric design*, pages 1–33, 1989.

[7] N. Alon and B. Sudakov. Bipartite subgraphs and the smallest eigenvalue. *Combinatorics, Probability and Computing*, 9:1–12, 2000.

[8] L. Alparone, S. Baronti, A. Garzelli, and F. Nencini. A global quality measurement of pan-sharpened multispectral imagery. *IEEE Geoscience and Remote Sensing Letters*, 1(4):313–317, October 2004.

[9] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, October 2004.

[10] I. Amidror. Scattered data interpolation methods for electronic imaging systems: A survey. *Journal of Electronic Imaging*, 11(2):157–176, April 2002.

[11] S. Arora and R. Kannan. Learning mixtures of arbitrary Gaussians. In *Proceedings of the 33rd ACM Symposium on Theory of Computation (STOC)*, pages 247–257, 2001.

[12] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-median and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 106–113, Dallas, TX, May 1998.

[13] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.

[14] V. Arya, N. Garg, R. Khandekar, V. Pandit, A. Meyerson, and K. Munagala. Local search heuristics for k-median and facility location problems. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 21–29, Crete, Greece, 2001.

[15] V. Arya, N. Garg, R. Khandekar, V. Pandit, A. Meyerson, and K. Munagala. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[16] G. H. Ball and D. J. Hall. Some fundamental concepts and synthesis procedures for pattern recognition preprocessors. In *International Conference on Microwaves, Circuit Theory, and Information Theory*, pages 281–297, Tokyo, Japan, September 1964.

[17] G. H. Ball and D. J. Hall. Isodata, a novel method of data analysis and pattern classification. Technical Report AD 699616, Stanford Research Institute, Menlo Park, CA, 1965.

[18] G. H. Ball and D. J. Hall. Research on isodata techniques. Technical Report AD 744337, Stanford Research Institute, Menlo Park, CA, 1971.

[19] R. K. Beatson, J. B. Cherrie, and D. L. Ragozin. *Curve and Surface Fitting: San Malo 1999*, chapter Polyharmonic Splines in R^d: Tools for Fast Evaluation, pages 47–56. Vanderbilt University Press, Nashville,, 2000.

[20] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[21] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[22] S. D. Billings, R. K. Beatson, and G. N. Newsam. Interpolation of geophysical data using continuous global surfaces. *Geophysics*, 67(6):1810–1822, November-December 2002.

[23] P. Bogaert, P. Mahau, and F. Beckers. *Cokriging Software: The Spatial Interpolation of Agro-Climatic Data*, November 1995. `http://metart.fao.org/T_I/GBR/Tools/Ecokrig/Man0.htm`, January 2005.

[24] L. Bottou and Y. Bengio. Convergence properties of the k-means algorithms. In G. Tesauro and D. Touretzky, editors, *Advances in Neural Information Processing Systems 7*, pages 585–592. MIT Press, 1995.

[25] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceeding of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 378–388, New York, NY, USA, October 1999.

[26] P. S. Chavez, S. C. Slides, and J. A. Anderson. Comparison of three different methods to merge multiresolution and multispectral data: Landsat TM and SPOT panchromatic. *Photogrammetric Engineering and Remote Sensing*, 57(3):295–303, March 1991.

[27] A. Cheriyadat and L. M. Bruce. Decision level fusion with best-bases for hyperspectral classification. In *IEEE Workshop on Advances in Techniques for Analysis of Remotely Sensed Data, a workshop honoring Professor David Landgrebe*, pages 399–406, NASA/GSFC, Greenbelt, MD, USA, October 27-28 2003.

[28] G. Christakos. On the problem of permissible covariance and variogram models. *Water Resources Research*, 20(2):251–265, February 1984.

[29] A. A. Cole-Rhodes, K. L. Johnson, J. L. Moigne, and I. Zavorin. Multiresolution registration of remote sensing imagery by optimization of mutual information using a stochastic gradient. *IEEE Transactions on Image Processing*, 12(12):1495–1511, December 2003.

[30] N. A. C. Cressie. *Statistics for Spatial Data*. Wiley Series in probability and mathematical statistics. Applied probability and statistics section. John Wiley & Sons, Inc, New York, U.S.A, 1993.

[31] S. Dasgupta. Learning mixtures of Gaussians. In *Proceeding of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 634–644, New York, NY, USA, October 1999.

[32] S. Dasgupta and P. M. Long. Performance guarantees for hierarchical clustering. *Journal of Computer and System Sciences*, 70(4):555–569, June 2005.

[33] I. Daubechies. *10 Lectures on Wavelets*. CMBS-NSF Series Applications in Mathematics. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1992.

[34] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[35] K. A. Delin and S. P. Jackson. Sensor Web for in situ exploration of gaseous biosignatures. In *Proceedings of the IEEE Aerospace Conference*, volume 7, pages 465–472, 2001.

[36] A. P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm (with discussion). *Journal of the Royal Statistical Society, Series B (Methodological)*, 39:1–38, 1977.

[37] C. V. Deutsch and A. G.Journal. *GSLIB Geostatistical Software Library and User's Guide*, volume second. Oxford University Press, 1998.

[38] I. S. Dhillon, Y. Guan, and J. Kogan. Iterative clustering of high dimensional text data augmented by local search. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 131–138, 2002.

[39] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, January 2001.

[40] T. D. Dixon, E. F. Canga, S. G. Nikolov, T. Troscianko, J. M. Noyes, D. R. Bull, and C. N. Canagarajah. Quality assessment of false-colored fused displays. *Journal of the Society for Information Display*, 14(10):883–894, October 2006.

[41] T. D. Dixon, E. F. Canga, J. M. Noyes, T. Troscianko, S. G. Nikolov, D. R. Bull, and C. N. Canagarajah. Methods for the assessment of fused images. *Transactions on Applied Perception*, 3(3):309–332, July 2006.

[42] P. Drineas, R. Kannan, A. Frieze, S. Vempala, and V. Vinay. Clustering in large graphs and matrices. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 291–299, January 1999.

[43] Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi tesselations: Applications and algorithms. *SIAM Review*, (41):637–676, 1999.

[44] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2nd edition, October 2000.

[45] V. Faber. Clustering and the continuous k-means algorithm. *Los Alamos Science*, (22):138–144, November 1994.

[46] G. Farin. Surfaces over dirichlet tessellations. *Computer Aided Geometric Design*, 7(1–4):281 – 292, June 1990.

[47] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 434–444, 1988.

[48] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, New York, NY, 3rd edition, 1968.

[49] E. Forgey. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. *Biometrics*, 21:768, 1965.

[50] D. A. Forsyth and J. Ponce. *Computer Vision A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey,07458, 2003.

[51] G. Frahling and C. Sohler. A fast k-means implementation using coresets. In *Twenty-second Annual Symposium on Computational Geometry*, pages 135 – 143, Sedona, Arizona, USA, June 2006.

[52] R. Franke and G. Nielson. Smooth interpolation of large sets of scattered data. *International Journal of Numerical Methods in Engineering*, 15(11):1691–1704, 1980.

[53] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.

[54] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Morgan Kaufman, San Diego, CA, 1990.

[55] R. Furrer, M. G. Genton, and D. Nychka. Covariance tapering for interpolation of large spatial datasets. *Journal of Computational and Graphical Statistics*, 15(3):502–523, September 2006.

[56] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[57] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic, Boston, MA, 1992.

[58] P. J. Gibson and C. H. Power. *Introductory Remote Sensing: Digital Image Processing and Applications*. Routledge, NY, 2001.

[59] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[60] P. Goovaerts. *Geostatistics for Natural Resources Evaluation*. Oxford University Press, New York, Oxford, 1997.

[61] L. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. PhD thesis, Yale, NYU, 1987.

[62] L. Greengard and V. Rokhlin. A fast algorithm for particle simulation. *Journal of Computational Physics*, 73(2):325–348, 1987.

[63] L. Greengard and J. Strain. The fast Gauss transform. *SIAM Journal of Scientific and Statistical Computing,*, 12(1):79–94, 1991.

[64] S. Guattery and G. Miller. The quality of spectral separators. *SIAM Journal on Matrix Analysis and Applications*, 19(3):701–719, 1998.

[65] N. A. Gumerov and R. Duraiswami. *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*. The Elsevier Electromagnetism Series. Elsevier Science, March 2005.

[66] D. L. Hall. *Mathematical techniques in multisensor data fusion*. Norwood: Artech House Inc, 1992.

[67] D. L. Hall and J. Llinas. *Handbook of Multisensor Data Fusion*. CRC Press LLC+, 2001.

[68] S. Har-Peled. Clustering motion. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 84, Washington, DC, USA, 2001.

[69] S. Har-Peled. Clustering motion. *Discrete and Computational Geometry*, 31(4):545–565, 2004.

[70] S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. In *Symposium on Computational Geometry (SoCG)*, pages 126–134, 2005.

[71] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *Proc. 36th Annual ACM Symposium on Theory of Computing*, pages 291–300, Chicago, IL, 2004.

[72] S. Har-Peled and B. Sadri. How fast is the k-means method? *Algorithmica*, 41(3):185–202, January 2005.

[73] R. M. Haralick, K. Shanmugan, and I. Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(6):610–621, 1973.

[74] J. K. Hart and K. Martinez. Environmental sensor networks: A revolution in the earth system science? *Earth-Science Reviews*, 78:177–191, 2006.

[75] M. T. Heath. *Scientific Computing: An Introductory Survey.* McGraw-Hill Companies, 1997.

[76] D. Hochbaum and D. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[77] D. Hochbaum and D. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM (JACM)*, 33(3):533–550, July 1986.

[78] D. S. Hochbaum. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company, Boston, MA, 1997.

[79] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge Press, 1985.

[80] E. H. Isaaks and R. M. Srivastava. *An Introduction to Applied Geostatistics.* Oxford University Press, New York, Oxford, 1989.

[81] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[82] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[83] K. Jain and V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1999.

[84] J. R. Jensen. *Introductory Digital Image Processing: A remote sensing perspective.* Prentice Hall, Englewood Cliffs, NJ, 1996.

[85] A. G. Journel and C. J. Huijbregts. *Mining Geostatistics.* Academic Press Inc, New York, 1978.

[86] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad, and spectral. In *Proceedings of the 41st Annual Symposium on the Foundation of Computer Science*, pages 367–380. IEEE Computer Society, 2000.

[87] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892, 2002.

[88] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry: Theory and Applications*, 28:89–112, 2004.

[89] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, New York, NY, 1990.

[90] Roger L. King and Jianwen Wang. A wavelet based algorithm for pan sharpening Landsat 7 imagery. In *International Geoscience and Remote Sensing Symposium (IGARSS)*, volume 2, pages 849–851, 2001.

[91] J. Kittler, M. Hatef, R. Duin, and J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, March 1998.

[92] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, New York, NY, 3rd edition, 1989.

[93] S. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the Euclidean k-median problem. In J. Nesetril, editor, *Proceedings of the Seventh Annual European Symposium on Algorithms*, volume 1643 of *Lecture Notes in Computer Science*, pages 362–371. Springer-Verlag, 1999.

[94] R. Krishnapuram and J. M. Keller. A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, 1(2):98–110, May 1993.

[95] M. Krivelevich and B. Sudakov. The largest eigenvalue of sparse random graphs. *Combinatorics, Probability and Computing*, 12(1):61–72, 2003.

[96] A. Kumar, Y. Sabharwal, and S. Sen. A simple linear time $(1 + \epsilon)$-approximation algorithm for k-means clustering in any dimensions. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 454 – 462, October 2004.

[97] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of Computation*, 87:141–158, 1981.

[98] D. Levin. The approximation power of moving least-squares. *Mathematical Computing*, 67(224):1517–1531, 1998.

[99] Thomas M. Lillesand and Ralph W. Kieffer. *Remote Sensing and Image Interpretation*. John Wiley and Sons, New York, second edition, 1987.

[100] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[101] C. F. Van Loan. *Introduction to Scientific Computing*. Prentice-Hall, Upper Saddle River, NJ 07458, second edition, 2000.

[102] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–296, Berkeley, CA, 1967.

[103] S. Mallat. Theory for multiresolution signal decomposition. *IEEE Pattern Analysis and Machine Intelligence (PAMI)*, 11(7):674–693, 1989.

[104] O. L. Mangasarian. Mathematical programming in data mining. *Data Mining and Knowledge Discovery*, 1:183–201, 1997.

[105] D. Marcotte. Cokriging with Matlab. *Computers and Geosciences*, 17(9):1265–1280, 1991.

[106] J. Matoušek. On approximate geometric k-clustering. *Discrete and Computational Geometry*, 24:61–84, 2000.

[107] N. Memarsadeghi, J. L. Moigne, and D. M. Mount. Image fusion using cokriging. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS'06)*, pages 2518 – 2521, July 2006.

[108] N. Memarsadeghi, J. L. Moigne, D. M. Mount, and J. Morisette. A new approach to image fusion based on cokriging. In *the Eighth International Conference on Information Fusion*, volume 1, pages 622–629, July 2005.

[109] N. Memarsadeghi, D. M. Mount, N. S. Netanyahu, and J. L. Moigne. A fast implementation of the ISOCLUS algorithm. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS'03)*, volume 3, pages 2057–2059, Toulouse, France, July 2003.

[110] N. Memarsadeghi, D. M. Mount, N. S. Netanyahu, and J. L. Moigne. A fast implementation of the ISODATA clustering algorithm. *International Journal of Computational Geometry and Applications (IJCGA)*, 17(1):71–103, 2007.

[111] T. H. Meyer. The discontinuous nature of kriging interpolation for digital terrain modeling. *Cartography and Geographic Information Science,*, 31(4):209–216, 2004.

[112] T. M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, March 1997.

[113] J. Le Moigne, N. Laporte, and N. S. Netanyahu. Enhancement of tropical land cover mapping with wavelet-based fusion and unsupervised clustering of SAR and Landsat image data. In *Proceedings of the Eighth SPIE International Symposium on Remote Sensing*, volume 4541, pages 190–198, Toulouse, France, September 2001.

[114] D. M. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching. `http://www.cs.umd.edu/~mount/ANN/`, May 2005.

[115] D. E. Myers. Pseudo-cross variograms, positive-definiteness, and cokriging. *Mathematical Geology*, 23(6):805–816, 1991.

[116] E. A. Nadaraya. On estimating regression. *Theory of Probability and Its Applications*, 9:141–142, 1964.

[117] E. A. Nadaraya. On non-parametric estimates of density functions and regression curves. *Theory of Probability and Its Applications*, 10:186–190, 1965.

[118] S. G. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill Companies, 1996.

[119] R. T. Ng and J. Han. CLARANS: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[120] D. P. O'Leary, G. W. Stewart, and J. S. Vandergraft. Estimating the largest eigenvalue of a positive definite matrix. *Mathematics of Computation*, 33(148):1289–1292, October 1979.

[121] J. M. Ortega. *Numerical Analysis, A Second Course*. Computer Science and Applied Mathematics. Academic Press, Inc, 111 Fifth Avenue, New York, NY, 10003, 1972.

[122] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, September 1975.

[123] E. Pardo-Igzquiza, M. Chica-Olmo, and P. Atkinson. Downscaling cokriging for image sharpening. *Remote Sensing of Environment*, 102(1–2):86–98, 2006.

[124] PCI Geomatics Corp. ISOCLUS–Isodata clustering program. `http://www.pcigeomatics.com/cgi-bin/pcihlp/ISOCLUS`, January 2005.

[125] V. Petrovic and C. Xydeas. Objective image fusion performance measure. *IEEE Electronics Letters*, 36(4):308–309, February 2000.

[126] G. Piella. New quality measure for image fusion. In *7th International Conference on Information Fusion*, pages 542–546, 2004.

[127] G. Piella and H. Heijmans. A new quality metric for image fusion. In *International Conference on Image Processing (ICIP)*, volume 3, pages 173–176, September 2003.

[128] L. Pitt and R. E. Reinke. Criteria for polynomial-time (conceptual) clustering. *Machine Learning*, 2(4):371–396, April 1988.

[129] C. Pohl and J. L. van Genderen. Multisensor image fusion in remote sensing: concepts, methods, and applications. *International Journal of Remote Sensing*, 19(5):823–854, 1998.

[130] D. Pollard. A centeral limit theorem for k-means clustering. *Annals of Probability*, 10:919–926, 1982.

[131] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C++, The Art of Scientific Computing*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 2nd edition, 2002.

[132] G. H. Qu, D. L. Zhang, and P. F. Yan. Medical image fusion by wavelet transform modulus maxima. *Journal of the Optical Society of America*, 9(4):184–190, 2001.

[133] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[134] V. C. Raykar. *Scalable Machine Learning for Massive Datasets: Fast Summation Algorithms*. PhD thesis, University of Maryland, College Park, MD, 20742, March 2007.

[135] V. C. Raykar and R. Duraiswami. *Large Scale Kernel Machines*, chapter The Improved Fast Gauss Transform with applications to machine learning. MIT Press, 2007.

[136] V. C. Raykar, C. Yang, R. Duraiswami, and N. Gumerov. Fast computation of sums of Gaussians in high dimensions. Technical report, Department of Computer Science, University of Maryland, College Park, MD, 20742, 2005. CS-TR-4767,.

[137] N. Remy. GsTL: The Geostatistical Template Library in C++. Master's thesis, Department of Petroleum Engineering of Stanford University, March 2001.

[138] N. Remy. *The Stanford Geostatistical Modeling Software (S-GeMS).* SCRC Lab, Stanford University, May 2004. `http://sgems.sourceforge.net`.

[139] R. J. Renka. Algorithm 660: QSHEP2D: Quadratic shepard method for bivariate interpolation of scattered data. *ACM Transactions on Mathematical Software (TOMS)*, 14(2):149–150, June 1988.

[140] R. J. Renka. Multivariate interpolation of large sets of scattered data. *ACM Transactions on Mathematical Software (TOMS)*, 14(2):139–148, June 1988.

[141] R. J. Renka. Algorithm 790: CSHEP2D: cubic shepard method for bivariate interpolation of scattered data. *ACM Transactions on Mathematical Software.*, 25(1):70–73, 1999.

[142] J. Richards. Information and understanding: Analysis of remotely sensed data. In *IEEE Workshop on Advances in Techniques for Analysis of Remotely Sensed Data, a workshop honoring Professor David Landgrebe*, NASA/GSFC, Greenbelt, MD, USA, October 27–28 2003.

[143] J. A. Richards and X. Jia. *Remote Sensing Digital Image Analysis.* Springer, Berlin, 1999.

[144] Y. Saad. *Iterative methods for sparse linear systems.* SIAM, 2003.

[145] F. Jr. Sabins. *Remote Sensing Principles and Interpretation.* W.H. Freeman and Company, San Francisco, 1978.

[146] J. Schnase, T. Stohlgren, and J. A. Smith. The National Invasive Species Forecasting System: A strategic NASA/USGS partnership to manage biological invasions. *NASA Earth Science Enterprise Applications Division Special Issue. Earth Observing Magazine*, pages 46–49, August 2002.

[147] S. Z. Selim and M. A. Ismail. K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:81–87, 1984.

[148] M. Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete Computational Geometry*, 18:125–134, 1997.

[149] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *the 1968 23rd ACM national conference*, pages 517–524. ACM Press New York, NY, USA, 1968.

[150] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.

[151] J. A. Shine and Krause. Exploration and estimation of North American climatological data. In 32^{nd} *Symposium on the Interface: Computing Science and Statistics*, April 5-8 2000.

[152] R. Sibson. *Interpreting Multivariate Data*, chapter A brief description of natural neighbour interpolation, pages 21–36. John Wiley & Sons, New York, 1981.

[153] V. Simoncini and D. B. Szyld. Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM Journal of Scientific Computing*, 25(2):454–457, 2004.

[154] D. A. Spielman and S. Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *IEEE Symposium on Foundations of Computer Science*, pages 96–105, 1996.

[155] M. L. Stein. The screening effect in kriging. *Annals of Statistics*, 1(30):298–323, 2002.

[156] J. T. Tou and R. C. Gonzalez. *Pattern Recognition Principles*. Addison-Welsley, London, 1974.

[157] V. Vijayaraj, C. G. O'Hara, and N. H. Younan. Quality analysis of pansharpened images. In *International Geoscience and Remote Sensing Symposium (IGARSS)*, volume 1, pages 85–84. IEEE, September 2004.

[158] Z. Wang and A. C. Bovik. A universal image quality index. *IEEE Signal Processing Letters*, 9(3):81–84, March 2002.

[159] E. W. Weisstein. Interpolation. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/Interpolation.htm`.

[160] R. Welch and W. Ahlers. Merging multiresolution SPOT HRV and Landsat TM data. *Photogrammetric Engineering & Remote Sensing*, 53(3):301–303, 1987.

[161] R. Xu and D. Wunsch II. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.

[162] C. Yang, R. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast Gauss transform. In L. K. Saul, Y. Weiss, and Lon Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17, pages 1561–1568. MIT Press, 2005.

[163] A. Zandifar, S. Lim, R. Duraiswami, N. Gumerov, and L. S. Davis. Multi-level fast multipole method for thin plate spline evaluation. In *IEEE International Conference on Image Processing (ICIP)*, pages 1683–1686, 2004.

[164] Y. Zhang. Understanding image fusion. *Photogrammetric Engineering and Remote Sensing*, pages 657–661, June 2004.

[165] Y. Zhao and G. Karypis. *Functional Genomics: Methods in Molecular Biology*, chapter Clustering in Life Sciences. Humana Press, 2003.

[166] Y. Zhao and G. Karypis. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10(2):141–168, March 2005.

[167] J. Zhou, D. L. Civco, and J. A. Silander. A wavelet transform method to merge Landsat TM and SPOT panchromatic data. *International Journal of Remote Sensing*, 19(4):743–757, 1998.