Diplomarbeit

Die Diskrete Gauß-Transformation – schnelle Approximationsverfahren und Anwendungen in hohen Dimensionen

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

angefertigt am
Institut für Numerische Simulation

Daniel Wissel

April 2008

Diploma Thesis

The Discrete Gauss Transform – Fast Approximation Algorithms and Applications in High Dimensions

submitted to the
Faculty of Mathematics and Natural Sciences of the
Rheinische Friedrich-Wilhelms-Universität Bonn, Germany

composed at the
Institute for Numerical Simulation

Daniel Wissel

April 2008

Contents

1 Einleitung

Seit sich der Einsatz moderner Computer in nahezu jeder wissenschaftlichen Fachrichtung und sämtlichen Wirtschaftssparten etabliert hat, liegt deren Hauptaufgabe in vielen Bereichen nicht mehr in der Ausführung komplexer Berechnungen, sondern im Sammeln, Speichern und Verwalten großer Datenmengen. Die wachsende Anzahl und Größe automatisch erzeugter Datensätze hat in den vergangenen Jahrzehnten die Entwicklung neuer Methoden auf dem Gebiet der *statistischen Lerntheorie* [64], [33] vorangetrieben. Die entsprechenden Algorithmen des maschinellen Lernens werden gewöhnlich eingesetzt, um gewisse Beziehungen (zum Beispiel Abstände, Korrelationen, Reihenfolgen, Häufungen oder Unterteilungen) in einer bestimmten Art von Daten (beispielsweise numerische Messwerte, räumliche Objektpositionen, Textdokumente oder Bildmaterial) aufzufinden. Dabei sind konkrete Anwendungen etwa die automatische Erkennung handschriftlicher Symbole beim Sortieren von Versandartikeln, die Einteilung von Kunden eines Unternehmens in bestimmte Kategorien anhand statistisch erhobener Daten zwecks zielgruppenoptimierter Werbung, oder die Klassifikation von Himmelskörpern unter Verwendung von Messungen des elektromagnetischen Spektrums. Zahlreiche weitere Einsatzgebiete finden sich in der Geostatistik [16] sowie Bioinformatik [3], [57].

Kernfunktionen in der statistischen Lerntheorie

Die allgemeine Problemstellung des maschinellen Lernens läßt sich mit Hilfe der folgenden Notation veranschaulichen. Das Tupel

$$D = ((x_1, y_1), \ldots, (x_N, y_N)) \in \left(\mathbb{R}^d \times \mathbb{R}\right)^N \tag{1.1}$$

besteht aus den d-dimensionalen Repräsentationen x_i (auch "Merkmale" genannt) der Datenobjekte sowie den Antwortvariablen y_i, welche den zu untersuchenden Sachverhalt beschreiben. Das Ziel besteht nun darin, unter bestimmten Annahmen über die Daten und Messwerte einen funktionalen Zusammenhang

$$f : \mathbb{R}^d \to \mathbb{R},\ f(x_i) \approx y_i, \tag{1.2}$$

herzustellen, welcher die Vorhersage der Antwortwerte für neue Datenobjekte ermöglichen soll. Die entsprechenden Voraussetzungen werden dabei üblicherweise mittels statistischer Modelle formalisiert. Grundsätzlich lassen sich drei verschiedene Formen von Lernproblemen unterscheiden: erstens Dichteschätzung, wobei eine angenommene zugrundeliegenden Wahrscheinlichkeitsdichte der als Stichprobe aufgefassten x_i geschätzt wird – dabei sind die y_i konstant; zweitens Klassifikation der Datenobjekte, für den Fall dass die y_i nur endlich viele Werte annehmen; und schließlich Regression, das bedeutet in diesem Zusammenhang die Rekonstruktion einer Funktion, falls die y_i aus einem kontinuierlichen Wertebereich entstammen.

Eine bedeutsame Zahl von Algorithmen aus dem Umfeld des maschinellen Lernens macht Gebrauch von sogenannten *Kernmethoden* [54], [56], das sind Techniken im Zusammenhang mit Kernfunktionen. Eine Kernfunktion (oder auch: ein *Kern*) ist definiert als eine Funktion

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \tag{1.3}$$

wobei $\mathcal{X}$ ganz allgemein die Menge der zu untersuchenden Objekte ist. In der Praxis verwendete Kerne haben weitere nützliche Eigenschaften wie z. B. Symmetrie und positive Definitheit. Sie dienen gewöhnlich als Ähnlichkeitsmaß für zwei Datenobjekte, etwa in Form von Kovarianzen. Eine symmetrisch positiv definite Kernfunktion k kann des weiteren zur Konstruktion eines eindeutig bestimmten Hilbertraumes $\mathcal{H}_k$ reeller Funktionen verwendet werden; dieser Raum wird dann als *reproduzierender Kern-Hilbertraum* [2] für den zugehörigen reproduzierenden Kern k bezeichnet. Damit können nun verschiedene Aufgabenstellungen aus der Lerntheorie als Minimierung des sogenannten *Tikhonov-Funktionals* [64] formuliert werden:

$$\min_{f \in \mathcal{H}_k} \frac{1}{N} \sum_{i=1}^{N} V\left(y_i, f(x_i)\right) + \lambda \left\| f \right\|_k^2 . \tag{1.4}$$

Dabei ist $\| \cdot \|_k$ die mit dem Hilbertraum $\mathcal{H}_k$ assoziierte Norm, $V : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ ein Fehlermaß für die Abweichung des vorhergesagten Ausgabewerts $f(x_i)$ von dem Messwert y_i, und schließlich $\lambda > 0$ der Regularisierungsparameter, welcher den Grad der Glattheit der Lösung kontrolliert. Unter gewissen schwachen Voraussetzungen gewährleistet das *Repräsentertheorem* [55] die Existenz einer eindeutigen Lösung f_{min} von (1.4), welche sich darstellen läßt als Linearkombination von reproduzierenden Kernfunktionen:

$$f_{min}\left(x\right) = \sum_{i=1}^{N} \alpha_i k(x, x_i). \tag{1.5}$$

Man kann weiter zeigen, dass der Lösungsvektor $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N) \in \mathbb{R}^N$ dem Gleichungssystem

$$(K + \lambda I) \cdot \boldsymbol{\alpha} = \mathbf{y} \tag{1.6}$$

genügen muss, wobei $K = (K_{ij})_{i,j=1,\ldots,N}$, die sogenannte Kernmatrix, gegeben ist durch $K_{ij} = k(x_i, x_j)$, I die Einheitsmatrix sowie $\mathbf{y} = (y_1, \ldots, y_N)$ ist. Somit wird das Minimierungsproblem im wesentlichen auf das Lösen eines linearen Gleichungssystems mit der Kernmatrix K reduziert.

Die am häufigsten eingesetzte Kernfunktion ist der *Gauß-Kern* [54], [58], [56]

$$k_h(x, y) = e^{-\|x-y\|^2/h^2}, \tag{1.7}$$

mit dem Parameter h, der in bestimmten Zusammenhängen als "Bandbreite" bezeichnet wird. Die Popularität des Gauß-Kerns resultiert vor allem aus seinen gutartigen analytischen Eigenschaften wie beispielsweise positive Definitheit und beliebige Glattheit auf dem gesamten Definitionsbereich, sowie seiner Bedeutung in der Wahrscheinlichkeitstheorie im Zusammenhang mit dem zentralen Grenzwertsatz [54].

Schnelle Näherungsverfahren für die diskrete Gauß-Transformation

Der Einsatz von iterativen Verfahren (wie etwa der Methode der *konjugierten Gradienten* oder der *GMRES-Methode* [61], [26]) zur Lösung des Gleichungssystems (1.6) erfordert bei Wahl der gaußschen Kernfunktion die wiederholte Berechnung der *diskreten Gauß-Transformation*

$$G(x_j) = \sum_{i=1}^{N} w_i \cdot e^{-\|x_j - x_i\|^2/h^2}, \quad (j = 1, \ldots, N)\,. \tag{1.8}$$

Das Hauptaugenmerk unserer Arbeit liegt auf der Untersuchung und dem Vergleich bekannter sowie der Entwicklung neuer Approximationsverfahren für die schnelle Berechnung der (diskreten) Gauß-Transformation. Die entsprechenden Algorithmen basieren allesamt auf dem Konzept der *schnellen Multipolmethode* (FMM, eng. *fast multipole method*) [36], [6]; hierbei handelt es sich um ein Näherungsverfahren zur Beschleunigung der mehrfachen Auswertung einer Funktion $f : \mathbb{R}^d \to \mathbb{R}$ der Form

$$f(t) = \sum_{i=1}^{N} w_i \cdot k(t, s_i) \tag{1.9}$$

an verschiedenen Zielpunkten $t_1, \ldots, t_M$ für vorgegebene Quellpunkte $s_1, \ldots, s_N$. Die Multipolmethode umfaßt drei wesentliche Techniken. Die Entwicklung der Kernfunktion k, üblicherweise eine unendliche Reihenentwicklung, zielt darauf ab, den Einfluss von Quell- und Zielpunkten in der Funktionsauswertung zu separieren. Im allgemeinen ist die Entwicklung nur lokal zur Näherung einsetzbar, daher wird als zweiter Bestandteil ein System zur effizienten Raumunterteilung benötigt, um auf dem gesamten Gebiet von dem Approximationsverfahren zu profitieren. Bei der Mehrebenen-Multipolmethode wird ein hierarchisches Unterteilungsschema angewendet; zusätzlich werden als dritte Komponente die Translationsoperatoren eingeführt, welche Teilergebnisse zwischen verschiedenen Ebenen austauschen können, um die Berechnung weiter zu beschleunigen. Dabei ist das Zusammenspiel der richtig aufeinander abgestimmten Techniken entscheidend für den optimalen Geschwindigkeitszuwachs der FMM.

Der erste Algorithmus, welcher die schnelle Multipolmethode auf die Gauß-Transformation übertrug, wurde von Greengard und Rokhlin [29] unter dem Namen *Fast Gauss Transform (FGT)* vorgestellt. Dieser markierte einen Durchbruch in Bezug auf Performanz in bestimmten Anwendungen aus der Physik und Finanzmathematik, obwohl nur eine schlichte nicht-hierarchische Raumunterteilung in Form eines uniformen Gitters verwendet wird und das Verfahren nur für Dimensionen $d \leq 3$ gut funktioniert. Die *Improved Fast Gauss Transform (IFGT)* [67] von Yang, Duraiswami und Gumerov setzt sowohl eine andere Reihenentwicklung als auch ein verbessertes, adaptives Cluster-Verfahren ("farthest-point clustering") ein, was in den meisten Fällen zu höherer Performanz führt, zum Teil auch in geringfügig höheren Dimensionen. Beide Methoden können jedoch ihre Vorteile nur für bestimmte Wertebereiche der Bandbreite h ausspielen; während der *FGT*-Algorithmus lediglich für mittlere Bandbreiten zu gebrauchen ist, profitiert die *IFGT*-Variante vor allem von relativ hohen Werten von h. Ein weiterer Algorithmus auf diesem Gebiet, die *Dual-Tree Fast Gauss Transform (DFGT)* [42], wurde von Lee, Gray und Moore eingeführt und kombiniert die Entwicklungstechniken der *Fast Gauss Transform* mit einer vollständigen Mehrebenen-Multipolmethode. Hier wer-

den zwei Baumstrukturen verwendet, um Quell- und Zielpunkte hierarchisch zu gliedern. Diese Vorgehensweise ermöglicht eine sehr effiziente lokale Fehlerabschätzung sowie das schnelle Auffinden der nächsten Nachbarn zwecks Zusammenfassen von Teilresultaten, was letztlich in einer schnellen Berechnung für eine große Spanne von verschiedenen Bandbreiten resultiert.

Eigene Beiträge

Die wesentlichen Beiträge dieser Arbeit finden sich in den Kapiteln 4 und 5; dabei handelt es sich um folgende Aspekte.

- Wir untersuchen und vergleichen die unterschiedlichen algorithmischen Ansätze (*FGT*, *IFGT* and *DFGT*) im Detail, insbesondere deren jeweilige Reihenentwicklungen, zugehörige Fehlerschranken, Methode zur Raumunterteilung sowie Laufzeitkomplexität.
- Wir entwickeln ein adaptives Cluster-Verfahren für die *Improved Fast Gauss Transform* mit effektiver Kostenschätzung, welches einen Aufwandsvergleich mit der direkten Auswertung während der Laufzeit ermöglicht.
- Wir präsentieren verbesserte lokale Fehlerschranken für die Reihenentwicklungen der *Dual-Tree Fast Gauss Transform*.
- Erstmals kombinieren wir die *IFGT*-Reihenentwicklung mit einem Dual-Tree-Verfahren, um die Vorteile beider Methoden in bezug auf Geschwindigkeitszuwachs zu vereinen; den resultierenden Algorithmus nennen wir *Dual-Tree Improved Fast Gauss Transform (DIFGT)*.
- Um die Leistung des *DIFGT*-Algorithmus zu optimieren, entwickeln wir eine neue Baumstruktur, indem wir bestimmte Eigenschaften des VAM-Split-Baumes [65] und des SR-Baumes [38] kombinieren.
- Zusätzlich erweitern wir diese neue Baumstruktur und erhalten eine zweite Variante, welche eine lokale Hauptkomponentenanalyse beinhaltet, um die Baumknoten besser an die Verteilung der Daten anzupassen. Dieses neuartige Raumunterteilungsverfahren ermöglicht das Erkennen lokaler niederdimensionaler Strukturen in hochdimensionalen Daten und führt uns zu einem weiteren neuen Algorithmus, der *Single-Tree Improved Fast Gauss Transform (SIFGT)*.
- Schließlich implementieren wir die verschiedenen diskutierten Methoden unter besonderer Berücksichtigung von Geschwindigkeitsoptimierungen aller Art. Wir vergleichen die Laufzeiten und die gemessenen Approximationsfehler anhand synthetisch generierter Daten, wobei wir diverse technische Details der Implementationen darlegen. Darüber hinaus werden die Algorithmen auf Datensätze aus realen Messungen angewendet, um ihre Effizienz im praktischen Einsatz zu demonstrieren.

Es stellt sich letztlich heraus, dass unsere zwei neuartigen Algorithmen erstmalig dazu in der Lage sind, bei der Approximation der diskreten Gauß-Transformation in beliebig hohen Dimensionen mit beliebiger Genauigkeit die direkte Auswertung in jeder von uns getesteten

Parameterkonfiguration geschwindigkeitsmäßig zu übertreffen, wobei sie in vielen Anwendungen deutlich schneller als alle vorherigen Approximationsverfahren agieren.

Aufbau der Arbeit

Kommen wir nun zum Aufbau des Hauptteils der Arbeit.

In *Kapitel 2* führen wir zunächst die grundlegende Theorie von Kernfunktionen und den zugehörigen reproduzierenden Kern-Hilberträumen ein. Anschließend werden einige Grundlagen der Wahrscheinlichkeitstheorie vorgestellt, die benötigt werden, um die Details der verschiedenen Methoden aus der statistischen Lerntheorie zu untersuchen. Wir diskutieren die Konzepte der Kerndichteschätzung [58], der Regression mittels Gaußscher Prozesse [47], [22] und der regularisierten Klassifikation nach dem Kleinste-Quadrate-Ansatz [51], [50], und zeigen die Rolle der diskreten Gauß-Transformation im jeweiligen Zusammenhang auf.

Kapitel 3 beginnt mit den analytischen Voraussetzungen für die Näherungsverfahren. Hier werden zunächst die Taylorentwicklung für Funktionen in mehreren Variablen sowie die Hermitefunktionen vorgestellt; anschließend führen wir die schnelle Multipolmethode ein und diskutieren detailliert die entsprechenden Techniken anhand des Beispiels eines elektrostatischen Potentials verursacht durch mehrere punktförmige Ladungsträger, welches ursprünglich den Anstoß für die Entwicklung der Approximationsmethode gab.

Die konkreten Näherungsverfahren werden in *Kapitel 4* diskutiert. Hier legen wir sämtliche theoretischen und technischen Aspekte dar: Reihenentwicklung, Fehlerabschätzungen, Verfahren zur Raumunterteilung sowie Implementationsdetails und Komplexitätsanalyse. Wir arbeiten Vorzüge und Schwachpunkte der verschiedenen Varianten heraus und studieren die Überlegungen, die zu Neuerungen und Verbesserungen geführt haben. Die letzten zwei Abschnitte behandeln unsere neu entwickelten Algorithmen und beinhalten zusätzlich eine ausführliche Diskussion diverser Baumstrukturen.

In *Kapitel 5* präsentieren wir numerische Resultate anhand von Simulationsläufen mit verschiedenen Datensätzen und Parameterkombinationen, welche unsere vorherigen theoretischen Überlegungen untermauern. Zunächst werden mit Hilfe synthetischer Daten charakteristische Eigenschaften bestimmter Methoden aufgezeigt, während anschließende Testläufe mit Messdaten aus realen Anwendungen den praktischen Nutzen der Verfahren demonstrieren.

Das letzte Kapitel fasst schließlich unsere Resultate zusammen und gibt Anregungen für zukünftige interessante Forschungsmöglichkeiten.

Danksagung

Bedanken möchte ich mich an dieser Stelle bei meinem Betreuer, Prof. Dr. Michael Griebel, für die interessante Fragestellung, seine kontinuierliche Unterstützung meiner Arbeit und viele hilfreiche Anregungen. Besonderer Dank gebührt Dr. Marc Alexander Schweitzer, der mich in zahlreichen Diskussionen über theoretische sowie algorithmische Aspekte mit seinen Fachkenntnissen und Ideen unterstützte, und damit meine Arbeit in vielfacher Hinsicht bereicherte. Darüber hinaus bin ich Prof. Dr. Rolf Krause zu Dank verpflichtet für die Übernahme des Zweitgutachtens. Ich möchte mich auch bei allen meinen Freunden und insbesondere meinen

Kollegen am Institut für Numerische Simulation für die sehr angenehme Arbeitsatmosphäre bedanken.

Schließlich möchte ich meine tiefe Dankbarkeit gegenüber meinen Eltern und meinem Bruder ausdrücken für ihre Unterstützung in allen Lebenslagen.

1 Introduction

Since the use of modern computers has become mainstream in almost every scientific discipline and branch of economy, their main job of performing fast computations has in many areas been substituted by the task of collecting, storing and organizing large amounts of data. In the last decades, the growing size of automatically gathered datasets promoted the development of methods in the wide field of *statistical learning theory* [64], [33]. In general, machine learning algorithms are utilized to discover certain types of relations (for example distances, correlations, sequences, clusters, classifications) in particular types of data (for instance numerical measurements, spatial positions, text documents, images). Common applications include the process of recognizing handwritten numbers in address data when sorting shipping items, categorizing customers into special target groups by means of personal data in order to optimize the distribution of advertisement, or classifying astronomical objects using their measured visible spectrum. Further current application areas can be found in geostatistics [16] and bioinformatics [3], [57].

1.1 The Role of Kernels in Statistical Learning Tasks

The general problem of learning theory can be formalized using the following notation. The tuple

$$D = ((x_1, y_1), \ldots, (x_N, y_N)) \in \left(\mathbb{R}^d \times \mathbb{R}\right)^N \tag{1.1}$$

consists of the d-dimensional representations x_i (sometimes called features) of the data and the observed response values y_i. Given some assumptions about the nature of the data and measured values, the goal is to find a certain functional relation

$$f : \mathbb{R}^d \to \mathbb{R},\ f(x_i) \approx y_i, \tag{1.2}$$

allowing to make predictions about the response value of new data objects. The assumptions are usually expressed in terms of statistical models. Considering the above notation we can principally distinguish three different applications: density estimation, that is, finding an estimate of a supposed underlying probability density function for the distribution of the x_i's, where the y_i's are constant; classification (or categorization), in case the y_i's only take finitely many values; and regression, in case the y_i's originate from a continuous range.

A significant amount of algorithms in the context of learning theory involve *kernel methods* [54], [56]. These techniques make use of kernel functions. A kernel (function) is most generally defined as a function

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \tag{1.3}$$

where $\mathcal{X}$ is the set of objects to be analyzed. Kernels commonly used in practice have further beneficial properties such as symmetry and positive definiteness. They usually serve as measures of similarity between two data objects, e. g. via covariances. A symmetric, positive semidefinite kernel k can also be used to construct a uniquely determined Hilbert space $\mathcal{H}_k$ of real-valued functions, then called *reproducing kernel Hilbert space* [2] for the appropriate *reproducing kernel*. Several tasks in learning theory can be formulated as *Tikhonov minimization* [64] problems of the form

$$\min_{f \in \mathcal{H}_k} \frac{1}{N} \sum_{i=1}^{N} V\left(y_i, f(x_i)\right) + \lambda \left\| f \right\|_k^2, \tag{1.4}$$

where $\| \cdot \|_k$ is the associated norm of $\mathcal{H}_k$, $V : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is some error measure for the distance between the predicted outputs $f(x_i)$ and the measured outputs y_i, and the scalar $\lambda > 0$ is the regularization parameter controlling the tradeoff between the accuracy and the smoothness of the solution. Under some weak assumptions, the *Representer Theorem* [55] then guarantees the unique solution f_{min} of (1.4) to be a finite linear combination of reproducing kernel functions, formally

$$f_{min}\left(x\right) = \sum_{i=1}^{N} \alpha_i k(x, x_i). \tag{1.5}$$

The solution vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N) \in \mathbb{R}^N$ can be shown to satisfy

$$(K + \lambda I) \cdot \boldsymbol{\alpha} = \mathbf{y}, \tag{1.6}$$

where $K = (K_{ij})_{i,j=1,\ldots,N}$ is the kernel matrix defined by $K_{ij} = k(x_i, x_j)$, I is the identity matrix, and $\mathbf{y} = (y_1, \ldots, y_N)$. The minimization task is therefore basically reduced to the problem of solving a linear system with the kernel matrix K.

The most widespread kernel in fact is the well-known *Gaussian kernel* [54], [58], [56]

$$k_h(x, y) = e^{-\|x-y\|^2/h^2}, \tag{1.7}$$

where h is a scaling parameter referred to as "bandwidth". The popularity of the Gaussian kernel results from its good analytic properties such as positive definiteness, infinite smoothness on the whole domain, and its significance in probability theory due to the central limit theorem [54].

1.2 Fast Approximation of the Discrete Gauss Transform

The choice of the Gaussian kernel in combination with the use of iterative methods (such as the *conjugate gradients* or *generalized minimal residual* method [61], [26]) for the solution of equation (1.6) entails the multiple evaluation of the *discrete Gauss Transform*

$$G(x_j) = \sum_{i=1}^{N} w_i \cdot e^{-\|x_j - x_i\|^2/h^2}, \quad (j = 1, \ldots, N). \tag{1.8}$$

The main focus of our work lays on the analysis and comparison of existing as well as the development of new approximation algorithms for the evaluation of the (discrete) Gauss Transform. The considered algorithms are variants of the *fast multipole method (FMM)* [36], [6], an approximation procedure to speed up the multiple evaluation of an arbitrary function $f : \mathbb{R}^d \to \mathbb{R}$ of the form

$$f(t) = \sum_{i=1}^{N} w_i \cdot k(t, s_i) \tag{1.9}$$

at some target points $t_1, \ldots, t_M$ for given source points $s_1, \ldots, s_N$. The FMM comprises three basic techniques. The expansion of the kernel function k, which usually is an infinite series expansion, aims to decouple the effect of source and target points in the evaluation process. In general, this can only be done locally, therefore a space partitioning scheme is required to benefit from the fast approximation on the whole domain. The (multilevel) FMM employs a hierarchical partitioning scheme and introduces additional translation operators that shift results between different levels in order to further speed up calculations. The crux with the FMM is to optimize the interplay of the utilized techniques in order to achieve the fastest evaluation.

The first algorithm adapting the fast multipole method to the Gauss Transform was presented by Greengard and Rokhlin [29], named the *Fast Gauss Transform (FGT)*. It represented a breakthrough in terms of speed for certain problems in physics and financial mathematics, even though it utilizes a plain single-level procedure with a simple uniform grid and only works well for dimensions $d \leq 3$. The *Improved Fast Gauss Transformm (IFGT)* [67] by Yang, Duraiswami and Gumerov employs a different expansion as well as an improved (but still single-level) space partitioning scheme called "farthest-point clustering" resulting in better performance for most scenarios even in slightly higher dimensions. Both methods however only work well for certain values of the bandwidth h; while the *FGT* performs well solely for moderate bandwidths, the *IFGT* benefits from relatively high bandwidths. Another competing algorithm, the *Dual-Tree Fast Gauss Transform (DFGT)* [42] was introduced by Lee, Gray and Moore and combines the expansions of the *Fast Gauss Transform* with the full multilevel FMM technique including translation operators. This variant utilizes two tree structures to get a hierarchical space partitioning scheme for both the source and target points. This technique allows an efficient local error control as well as fast nearest neighbor searching, which permits the possibility of speeding up calculations for a wide range of different bandwidths.

1.3 Contributions

The major contributions of this thesis are presented in Chapters 4 and 5 and comprise the following.

- We analyze and compare the different approaches (*FGT*, *IFGT* and *DFGT*) in detail including discussions of their series expansions, corresponding error bounds, space partitioning schemes and runtime complexity.
- We develop an adaptive clustering scheme for the *Improved Fast Gauss Transform* including effective cost estimation allowing to control the performance trade-off between

the fast approximation algorithm and direct evaluation.

- We present improved local error bounds for the series expansions of the *Dual-Tree Fast Gauss Transform*.
- For the first time, we combine the *IFGT* expansion with a dual-tree algorithm in order to profit from both their performance advantages, calling the resulting algorithm the *Dual-Tree Improved Fast Gauss Transform (DIFGT)*.
- In order to optimize the performance of our *DIFGT* algorithm, we develop a new tree structure combining features of the VAM-Split-tree [65] and the SR-tree [38].
- We further extend the new tree structure to obtain a more complex variant that utilizes a local principal component analysis to adapt the tree nodes to the distribution of the data. This new hierarchical space partitioning scheme permits to discover local low-dimensional features in high-dimensional data and leads to another new algorithm called the *Single-Tree Improved Fast Gauss Transform (SIFGT)*.
- Finally, we implement all different methods discussed above with a special focus on performance optimization. We compare the runtimes and measured approximation errors with synthetic data including a discussion of various technical issues of the implementations. In addition, the algorithms are applied to real-world datasets in order to demonstrate their efficiency for practical purposes.

It eventually turns out that the two novel algorithms, for the first time, are able to approximate the discrete Gauss Transform in any dimension up to an arbitrary precision and perform faster than direct evaluation for any tested dimension and parameter range, while performing considerably faster than all previous approximation algorithms in many real-world scenarios.

1.4 Outline of the Thesis

The remainder of this thesis is structured as follows.

In *Chapter 2*, we first introduce the basic theory of kernels and associated reproducing kernel Hilbert spaces. Next, the fundamentals of probability theory are presented that are necessary to study the details of different methods in statistical learning theory. We review the main concepts of kernel density estimation [58], Gaussian process regression [47], [22] and regularized least-squares classification [51], [50], and point out the role of the discrete Gauss transform in the respective context.

Chapter 3 starts with the analytic preliminaries for the approximation algorithms, including multivariate Taylor expansions and Hermite functions. Then, the fast multipole method is presented and discussed in detail using the example of the electrostatic potential due to a number of point charges, which originally gave rise to the approximation technique.

The specific fast approximation algorithms are discussed in *Chapter 4*. Here, we present all theoretical as well as technical issues concerning series expansions, error bounds, space partitioning methods as well as implementation details and complexity analysis. We highlight advantages and drawbacks of the different versions and present the considerations that led to

new and improved variants. The last two sections address our two newly developed algorithms including a concise discussion of diverse tree structures.

Chapter 5 finally backs our theoretical analysis by providing numerical results of the algorithms using different datasets and parameter combinations. Synthetic data are first used to expose characteristic behavior of the particular methods, while testing with real-world data demonstrates their practical virtues.

In the last chapter, we conclude the thesis by summarizing our results and providing encouragement for future research.

1.5 Acknowledgements

I would like to thank my advisor Prof. Dr. Michael Griebel for sponsoring this thesis and providing continuous support and many helpful ideas. Special thanks go out to Dr. Marc Alexander Schweitzer, who helped me in numerous discussions about theoretical and technical issues and supported my work in many ways. Furthermore, I am very grateful to Prof. Dr. Rolf Krause, the co-referee of my thesis. I also wish to thank my friends and particularly my colleagues at the Institute for Numerical Simulation for the pleasant working atmosphere.

Last but not least, I would like to express my gratitude to my parents and my brother for their support in every respect.

2 The Gaussian Kernel in Statistical Learning Theory

2.1 Kernels and Reproducing Kernel Hilbert Spaces

In the last few decades *kernel functions* or *kernels* [56] have been playing an increasingly important role in areas like *learning theory* [64] and *meshless methods* for the numerical solution of partial differential equations [30]. Special types of kernels have been introduced under varying names in different contexts, for example *radial basis functions*, *generalized finite elements* or *shape functions*. In order to cover all of these areas we introduce kernels in the most general way. Our short survey basically follows the detailed discussion of kernel techniques [54] by Robert Schaback and Holger Wendland. An elaborate reference is provided by Schölkopf and Smola in [56].

2.1.1 Basic Properties of Kernels

Definition 2.1 (Kernel)**.** *A kernel is a function*

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \quad (x, y) \mapsto k(x, y),$$

where $\mathcal{X}$ is a non-empty set.

This rather general definition allows the use of diverse domains for $\mathcal{X}$, since in learning theory $\mathcal{X}$ can be a set of texts to be classified as well as a set of digital images or a record of statistical customer data. Thus, we do not require $\mathcal{X}$ to have any a priori structure at all. The intuitive view in learning theory is that the kernel function is some kind of measure of (dis-)similarity of two arbitrary objects from the domain $\mathcal{X}$.

In settings where no kernel is specified by the particular application, the concept of *feature maps* is a common way to provide the necessary structure. A feature map $\Phi : \mathcal{X} \to \mathcal{F}$ has values in a Hilbert space $\mathcal{F}$ (in this context also called *feature space*) and should map each element $x \in \mathcal{X}$ onto its distinguishing features, that reside in the high-dimensional Hilbert space $\mathcal{F}$. We can now use the associated inner product $\langle \cdot, \cdot \rangle_{\mathcal{F}}$ to define the canonical kernel $k(x, y) = \langle \Phi(x), \Phi(y) \rangle_{\mathcal{F}}$ for all $x, y \in \mathcal{X}$.

Important properties of kernels are symmetry and positive (semi-)definiteness.

Definition 2.2 (Symmetry of kernels)**.** *A kernel $k(x, y)$ is called* symmetric*, if $k(x, y) = k(y, x)$ for all $x, y \in \mathcal{X}$.*

Definition 2.3 (Positive (semi-)definiteness of kernels)**.** *A symmetric kernel $k(x, y)$ is called* positive (semi-)definite*, if for any finite subset $(x_1, \ldots, x_N)$ of distinct points of $\mathcal{X}$ the corresponding kernel matrix given by $K_{ij} = (k(x_i, x_j))_{i,j=1,\ldots,N}$ is positive (semi-)definite.*

The interpretation of the kernel function as a similarity measure already suggests that most kernels in learning theory are symmetric. Nota bene, some authors refer to positive semidefinite kernels simply as "positive kernels", others define kernels to be only positive definite kernels in our notation. We stick to the most general variant and explicitly state whether a kernel is positive definite or semidefinite. A useful property of positive semidefinite kernels is the analogon of the Cauchy-Schwarz inequality for kernels.

Proposition 2.4 (Cauchy-Schwarz inequality for kernels)**.** *For any positive semidefinite kernel* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *and* $x_1, x_2 \in \mathcal{X}$*, it holds that*

$$|k(x_1, x_2)|^2 \leq k(x_1, x_1) \cdot k(x_2, x_2). \tag{2.1}$$

Proof. By the above definition of a positive semidefinite kernel, the (2×2)-matrix given by $K_{ij} = (k(x_i, x_j))_{i,j \in \{1,2\}}$ is positive semidefinite. This implies that both its eigenvalues λ_1, λ_2 are nonnegative, thus $\lambda_1 \cdot \lambda_2 = \det(K)$ is also nonnegative. The proposition now follows from

$$0 \leq \det(K) = K_{11}K_{22} - K_{12}K_{12} = k(x_1, x_1) \cdot k(x_2, x_2) - |k(x_1, x_2)|^2 .$$

□

Next we provide a criterion that allows to instantly classify certain kernels as positive definite. Therefore, consider the following definition.

Definition 2.5 (Completely monotonic function)**.** *A function* $f \in C^\infty(\mathbb{R}^+)$ *is called* completely monotonic*, if* $(-1)^k f^{(k)}(t) \geq 0$ *for all* $t > 0$ *and* $k \in \mathbb{N}_0$*.*

Here, $C^\infty(\mathbb{R}^+)$ is the space of infinitely differentiable functions defined on all positive reals, and $f^{(k)}$ is the kth derivative of f. Examples for completely monotonic functions are $f(t) = e^{-\alpha t}$ for any nonnegative α, and $f(t) = (t + c^2)^{-1/2}$ for any $c \in \mathbb{R}$, respectively.

Proposition 2.6. *Let* $f \in C^\infty(\mathbb{R}^+)$ *be completely monotonic and non-constant. Then for any* $n \in \mathbb{N}$ *the kernel* $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$*,* $k(x, y) = f\left(\|x - y\|^2\right)$ *is positive definite.*

Here, $\|\cdot\|$ is the Euclidean norm of the $\mathbb{R}^n$ given by $\|x\|^2 = x_1^2 + \ldots + x_n^2$ for $x = (x_1, \ldots, x_n)$. Proposition (2.6) (formulated in a slightly different notation) and a corresponding proof can be found in [11]. We finally introduce the *Gaussian kernel*, the most important kernel in our further considerations.

Definition 2.7 (Gaussian kernel)**.** *The* n*-dimensional* Gaussian kernel $k_h : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ *is given by*

$$k_h(x, y) = e^{-\|x-y\|^2/h^2}, \tag{2.2}$$

where $h > 0$ *is a scaling parameter.*

Since $f(t) = e^{-\alpha t}$ is completely monotonic, the proposition provided above guarantees the Gaussian kernel of any dimension to be positive definite. The real scaling parameter h is a feature that many kernels share. It is also referred to as *smoothing parameter* or *bandwidth* in

different contexts and plays a crucial role in the methods of learning theory that are going to be discussed later on. Further examples of diverse kernel functions will be presented in section 2.3 dealing with density estimation.

Remark 2.8. *Many kernels based on a set $\mathcal{X}$ with some underlying geometric structure are often presented using a simplified form (in a minor abuse of notation). For instance a kernel is defined as a function $k : \mathcal{X} \to \mathbb{R}$, where the actual kernel $\overline{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is given by $\overline{k}(x, y) = k(x - y)$.*

Kernel functions are a mathematical tool used in various areas. Kernels can generate convolutions, covariances, and trial functions for certain methods to solve partial differential equations [54]. The next topic we are going to address is a specific kind of Hilbert spaces that is closely connected with positive definite kernels.

2.1.2 Reproducing Kernel Hilbert Spaces

The term "reproducing kernel" was originally brought up about the same time by Moore [45] and Aronszajn [2]. The latter provides a good overview, a different perspective on the topic including some more details can be found in [56]. To begin with let us recall the definition of a Hilbert space.

Definition 2.9 (Hilbert space)**.** *A real or complex vector space $\mathcal{H}$ with an inner product $\langle \cdot, \cdot \rangle$, that is complete under the norm $\|\cdot\|$ defined by $\|x\| = \sqrt{\langle x, x \rangle}$, is called a* Hilbert Space*.*

Definition 2.10 (Reproducing kernel Hilbert space)**.** *Let $\mathcal{X}$ be an arbitrary set and $\mathcal{H}$ a Hilbert space of real-valued functions $f : \mathcal{X} \to \mathbb{R}$. Then $\mathcal{H}$ is called* reproducing kernel Hilbert space*, if for all $x \in \mathcal{X}$ the evaluation functional $\mathscr{F}_x : \mathcal{H} \to \mathbb{R}$, $\mathscr{F}_x(f) = f(x)$ is continuous.*

By the *Riesz representation theorem*, we can reason that for every $x \in \mathcal{X}$ there exists a Hilbert space element $k_x \in \mathcal{H}$ satisfying

$$\mathscr{F}_x(f) = \langle f(\cdot), k_x(\cdot) \rangle \qquad \forall f \in \mathcal{H}. \tag{2.3}$$

The function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ defined by

$$k(x, y) = k_x(y) \tag{2.4}$$

is then called *reproducing kernel* for the Hilbert space $\mathcal{H}$ due to its *reproducing property*

$$f(x) = \mathscr{F}_x(f) = \langle f(\cdot), k(x, \cdot) \rangle. \tag{2.5}$$

If on the other hand, for any Hilbert space $\mathcal{H}$ there exists a kernel k with the above reproducing property, the evaluation functional $\mathscr{F}$ is continuous because of the properties of the inner product. This justifies the term *reproducing kernel Hilbert space* (RKHS) in Definition 2.10. The reproducing kernel is specified more precisely by the following lemma.

Lemma 2.11. *The reproducing kernel k for a RKHS is symmetric and positive semidefinite.*

Proof. The kernel properties derive directly from the reproducing property and the characteristics of an inner product.
Symmetry: For any $x, y \in \mathcal{X}$

$$k(y,x) = k_y(x) = \langle k_y(\cdot), k_x(\cdot)\rangle = \langle k_x(\cdot), k_y(\cdot)\rangle = k_x(y) = k(x,y).$$

Positive semidefiniteness: For any $x_1, \ldots, x_N \in \mathcal{X}$ let $K = (k(x_i, x_j))_{i,j=1,\ldots,N}$. Then for any $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N) \in \mathbb{R}^N$:

$$\begin{aligned}\boldsymbol{\alpha}^T K \boldsymbol{\alpha} &= \sum_{i=1}^N \alpha_i \sum_{j=1}^N \alpha_j k(x_i, x_j) = \sum_{i=1}^N \alpha_i \sum_{j=1}^N \alpha_j \langle k_{x_i}(\cdot), k_{x_j}(\cdot)\rangle = \\ &= \left\langle \sum_{i=1}^N \alpha_i k_{x_i}(\cdot), \sum_{j=1}^N \alpha_j k_{x_j}(\cdot) \right\rangle = \left\| \sum_{i=1}^N \alpha_i k_{x_i}(\cdot) \right\|^2 \geq 0.\end{aligned}$$

□

Furthermore, note that the reproducing kernel k is unique for the Hilbert space $\mathcal{H}$. Assuming there is another reproducing kernel k', the reproducing property and symmetry of both kernels yield

$$k(x,y) = k(y,x) = k_y(x) = \langle k_y(\cdot), k'_x(\cdot)\rangle = \langle k'_x(\cdot), k_y(\cdot)\rangle = k'_x(y) = k'(x,y)$$

for all $x, y \in \mathcal{X}$, thus the two kernels coincide.

Conversely, given a symmetric positive semidefinite kernel k, one can construct a uniquely determined associated reproducing kernel Hilbert space $\mathcal{H} = \mathcal{H}_k$. The exact procedure is presented in the following lemma.

Lemma 2.12. *Let $\mathcal{X}$ be an arbitrary set and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a positive semidefinite kernel. We define the vector space*

$$\mathcal{F} = \left\{ f : \mathcal{X} \to \mathbb{R} \;\middle|\; \exists N \in \mathbb{N},\, \alpha_1, \ldots, \alpha_N \in \mathbb{R},\, x_1, \ldots, x_N \in \mathcal{X} : f(x) = \sum_{i=1}^N \alpha_i k(x, x_i) \right\}.$$

Let further $f, g \in \mathcal{F}$ be two arbitrary functions with $f(x) = \sum_{i=1}^N \alpha_i k(x, x_i)$, $g(x) = \sum_{j=1}^M \beta_j k(x, y_j)$. Then the inner product $\langle \cdot, \cdot \rangle$ given by

$$\langle f, g\rangle = \sum_{i=1}^N \sum_{j=1}^M \alpha_i \beta_j k(x_i, y_j)$$

is well-defined. Furthermore the completion of the pre-Hilbert space $(\mathcal{F}, \langle \cdot, \cdot \rangle)$ with respect to the associated norm $\|\cdot\| = \sqrt{\langle \cdot, \cdot \rangle}$ is a reproducing kernel Hilbert space $\mathcal{H} = \overline{\mathcal{F}}$ for the kernel k.

Proof. The definition of the inner product is unambiguous, since it can be rewritten in the following two variants

$$\langle f, g\rangle = \sum_{i=1}^{N} \alpha_i g(x_i), \qquad \langle f, g\rangle = \sum_{j=1}^{M} \beta_j f(y_j),$$

where the expressions are independent of the choice of the expansion for f and g, respectively. The bilinearity follows directly from the definition, while symmetry results from the symmetry of the kernel. Equally, the property of positive semidefiniteness is transferred from the kernel to the bilinear form by

$$\langle f, f\rangle = \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) \geq 0. \tag{2.6}$$

Notice now, that the function $\langle\cdot,\cdot\rangle : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$ is itself a positive semidefinite kernel. To show the positive semidefiniteness (in the kernel sense), we remark that for any $f_1, \ldots, f_M \in \mathcal{F}$ and $\gamma_1, \ldots, \gamma_M \in \mathbb{R}$ we have

$$\sum_{i=1}^{M}\sum_{j=1}^{M} \gamma_i \gamma_j \langle f_i, f_j\rangle = \left\langle \sum_{i=1}^{M} \gamma_i f_i, \sum_{j=1}^{M} \gamma_j f_j \right\rangle \geq 0. \tag{2.7}$$

Next, from the definition of $\langle\cdot,\cdot\rangle$ we can derive the *reproducing property*

$$\langle f(\cdot), k(x,\cdot)\rangle = \sum_{i=1}^{N} \alpha_i k(x, x_i) = f(x). \tag{2.8}$$

This allows us to deduce the positive definiteness of the bilinear form, for which we have to show $\langle f, f\rangle = 0 \Rightarrow f = 0$. Therefore, we apply the Cauchy-Schwarz inequality (2.1) to the kernel function $\langle\cdot,\cdot\rangle$ to get

$$|f(x)|^2 = |\langle f(\cdot), k(x,\cdot)\rangle|^2 \leq \langle f, f\rangle \cdot \langle k(x,\cdot), k(x,\cdot)\rangle = \langle f, f\rangle \cdot k(x,x). \tag{2.9}$$

Thus we have shown, that $\langle\cdot,\cdot\rangle$ is a well-defined inner product for the vector space $\mathcal{F}$. The completion is therefore a Hilbert space $\mathcal{H}$ and in particular a reproducing kernel Hilbert space for k due to the reproducing property (2.8).

□

In [1] the uniqueness of the RKHS constructed in Lemma 2.12 is shown. We can therefore summarize the results in the following theorem.

Theorem 2.13. *For any RKHS $\mathcal{H}$ there exists a unique symmetric, positive semidefinite kernel k with the reproducing property. Likewise, for any symmetric, positive semidefinite kernel k there exists a unique RKHS with k being its reproducing kernel.*

As an effect of this theorem, given an arbitrary symmetric, positive semidefinite kernel, we can study its features using the associated Hilbert space. However, for a large number of popular kernels the RKHS is known to be infinite-dimensional. In context of regularization theory [64], we are later going to introduce the famous *Representer Theorem* [55] allowing finite-dimensional representations of elements of the RKHS, that arise as solutions for regularized minimization problems.

To introduce the applications in statistical learning theory, the next section provides the corresponding basics of probability theory.

2.2 Fundamentals of Probability Theory

The common method to formalize a stochastic or random experiment (for instance throwing a dice) is the notion of the probability space introduced by KOLMOGOROV.

Definition 2.14 (Probability space)**.** *A* probability space *is a triple* $(\Omega, \mathscr{A}, P)$ *with the following properties:*

- Ω *is a nonempty (possibly infinite) set of outcomes, also known as* sample space*.*
- $\mathscr{A}$ *is a* σ-algebra over Ω*, that means*
  1. $\Omega \in \mathscr{A}$*;*
  2. $A \in \mathscr{A} \Rightarrow \Omega \backslash A \in \mathscr{A}$*;*
  3. $A_1, A_2, \ldots \in \mathscr{A} \Rightarrow \bigcup_{n \in \mathbb{N}} A_n \in \mathscr{A}$*.*

  The elements of $\mathscr{A}$ *are called* events*.*
- P *is a* probability measure*, that means*
  1. $P(A) \geq 0$ *for all* $A \in \mathscr{A}$*;*
  2. $P(\Omega) = 1$*;*
  3. $\forall A_1, A_2, \ldots \in \mathscr{A}$ *with* $A_n \cap A_m = \emptyset$ *for* $n \neq m$*:* $P\left(\bigcup_{n \in \mathbb{N}} A_n\right) = \sum_{n \in \mathbb{N}} P(A_n)$*.*

Intuitively, the sample space Ω contains all possible outcomes of the experiment, the elements ω of Ω are also called elementary events. In the example of throwing a dice, the outcomes could be named “1”, “2”, “3”, “4”, “5”, “6”. The elements of $\mathscr{A}$ are the events, for which the measure P provides probabilities between 0 and 1. Events for the dice throw would be for instance “number greater than 3” or “even number”.

Consider now the experiment of drawing 2 balls from an urn without returning them. Suppose there are 10 balls, 5 white ones and 5 black ones. Let A be the event “the first ball is white” and B the event “the second ball is white”. Without any further information, the probability is calculated as $P(A) = P(B) = \frac{1}{2}$. However if we know that event A has happened (the first ball was white), the probability of the event B must be re-evaluated as $\frac{4}{9}$. The general concept of calculating probabilities of an event B under the condition that another event A has happened is known as the *conditional probability* $P(B|A)$.

Definition 2.15 (Conditional probability)**.** *Let* $(\Omega, \mathscr{A}, P)$ *be a probability space and* $A \in \mathscr{A}$ *with* $P(A) > 0$*. Then for any* $B \in \mathscr{A}$

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

is called conditional probability of A given B*.*

In case $P(A) = 0$, one can set $P(B|A) = 0$. Another notation for the conditional probability is $P_A(B) := P(B|A)$. A simple technique to calculate conditional probabilities is provided by *Bayes' law.*

Theorem 2.16 (Bayes' law)**.** *Let $(\Omega, \mathscr{A}, P)$ be a probability space and $\Omega = \bigcup_{i\in I} B_i$, where I is countable and $B_i \in \mathscr{A}$ are pairwise disjoint. Then for any $A \in \mathscr{A}$ with $P(A) > 0$ and all $k \in I$:*

$$P(B_k|A) = \frac{P(B_k)P(A|B_k)}{P(A)} = \frac{P(B_k)P(A|B_k)}{\sum_{i\in I} P(B_i)P(A|B_i)}.$$

Bayes' law is easily deduced from the definition of the conditional probability and the probability measure.

Another term relating two events is *(stochastic) independence.* Intuitively, two events A and B should be independent, if the fact that any one of the two events has happened does not give reason to re-evaluate the probability of the other event; thus $P(A|B) = P(A)$ and $P(B|A) = P(B)$. The following definition supports this intuition.

Definition 2.17 (Independence of events)**.** *For a given probability space $(\Omega, \mathscr{A}, P)$ two events $A, B \in \mathscr{A}$ are called* independent*, if $P(A \cap B) = P(A) \cdot P(B)$. A collection of events $(A_i)_{i\in I}$ is called* mutually independent*, if for any finite subset $S \subset I$: $P\left(\bigcap_{i\in S} A_i\right) = \prod_{i\in S} P(A_i)$.*

Nota bene the concept of independence and conditional probabilities does not coincide with the intuitional idea of causality. Two events can have a causal connection and still be independent. Some concrete examples can be found in [21]. Also, pairwise independence of a collection of events does not induce independence of the whole collection.

Having formalized the experiment, one is often not interested in the pure outcome, but rather in some functional relation of the result. Imagine we would like to know the probability of having at least one "6" throwing a dice 10 times. The concept of random variables is used for this purpose.

Definition 2.18 (Random variable)**.** *Let $(\Omega, \mathscr{A}, P)$ be a probability space and $(\Omega', \mathscr{A}')$ be a* measurable space *(that is the pair of a nonempty set Ω' and a σ-algebra $\mathscr{A}'$ over Ω'). Let $X : \Omega \to \Omega'$ be a* measurable function with respect to $\mathscr{A}$, $\mathscr{A}'$ *(that means for all $A' \in \mathscr{A}'$: $X^{-1}(A') \in \mathscr{A}$). Then X is called an $(\Omega', \mathscr{A}')$-random variable.*

If the measurable space is $\left(\mathbb{R}^d, \mathscr{B}^d\right)$, we are speaking of real-valued d-dimensional random variables. Here the *Borel algebra* $\mathscr{B}^d$ is the minimal σ-algebra over $\mathbb{R}^d$ containing all open sets. We will limit our further considerations to real-valued random variables, since we focus on applications with real-valued results.

Note that there are important differences between discrete and continuous random variables. A (real-valued) random variable X is called discrete, if its range $X(\Omega)$ is finite or countably infinite, otherwise it is called continuous.

For many real-life applications, we do not have full knowledge of the exact coherences and therefore can not (or do not want to) define the probability space explicitly. In this case, the considered random variable can still be characterized further, if its distribution function is known.

Definition 2.19 (Cumulative distribution function)**.** *Let $(\Omega, \mathscr{A}, P)$ be a probability space and $X : \Omega \to \mathbb{R}$ be a real-valued random variable. The* cumulative distribution function $F : \mathbb{R} \to [0,1]$ *of X is then defined by*

$$F(x) = P(X \leq x) := P\left(\{\omega \in \Omega \mid X(\omega) \in (-\infty, x]\}\right).$$

The cumulative distribution function (cdf) allows to calculate probabilities of the kind $P(X \in I)$ for some interval $I \subset \mathbb{R}$. From its definition it is easily seen that the cdf is monotonically increasing with $\lim_{x\to-\infty} F(x) = 0$ and $\lim_{x\to\infty} F(x) = 1$. Another way to characterize a continuous random variable is the probability density function (pdf), that is closely related to the cdf:

Definition 2.20 (Probability density function)**.** *Let $(\Omega, \mathscr{A}, P)$ be a probability space and $X : \Omega \to \mathbb{R}$ be a real-valued continuous random variable. An integrable function $f : \mathbb{R} \to \mathbb{R}_0^+$ is called* probability density function *of X, if for all $a, b \in \mathbb{R}$ with $a \leq b$*

$$\int_a^b f(t)dt = P\left(a \leq X \leq b\right).$$

For any continuous random variable X with a given probability density function f, the associated cumulative distribution function F can be expressed via

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)dt,$$

while on the other hand, if the cdf F is differentiable, the pdf can be written as the derivative

$$f(x) = \frac{dF(x)}{dx}.$$

The distribution function and its associated density function might seem equipollent in terms of data analysis, since they can be transformed into each other easily. However the pdf has frequently turned out to be much more appropriate to intuitional interpretation by humans than the cdf, especially in plots representing three-dimensional data (see e. g. [58]). We therefore focus on the task of density estimation – as most of the literature does.

Two additional important characteristics of a random variable are its *expected value* and *variance*.

Definition 2.21 (Expected value, Variance)**.** *Let $(\Omega, \mathscr{A}, P)$ be a probability space, $X : \Omega \to \mathbb{R}$ be a real-valued continuous random variable and $F(x)$ the associated cumulative distribution function. The* expected value *or* mean *of X is then defined by*

$$E(X) = \int_\Omega X dF(x),$$

in case the Lebesgue integral exists and is finite. The variance *of X is defined by*

$$\mathrm{Var}(X) = E\left[(X - E(X))^2\right].$$

The square root of the variance is also called standard deviation $\sigma_X = \sqrt{\mathrm{Var}(X)}$.

Example 2.22. *One of the most popular probability distributions is the* normal distribution. *The associated probability density function – often called* bell curve *due to its shape – is given by*

$$\phi(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2},$$

where μ is the mean *and σ^2 the variance. The distribution with $\mu = 0$ and $\sigma = 1$ is referred to as* standard normal distribution. *If a real-valued random variable X has a normal distribution with mean μ and variance σ^2, we write*

$$X \sim \mathcal{N}\left(\mu, \sigma^2\right).$$

Dealing with multivariate data, one often has to presume that different components are derived from different random variables. To formalize the simultaneous consideration of multiple random variables we need joint distribution and density functions.

Definition 2.23 (Joint distribution function)**.** *Let $(\Omega, \mathscr{A}, P)$ be a probability space and $X_1, \ldots, X_d : \Omega \to \mathbb{R}$ be real-valued continuous random variables. A function $F : \mathscr{B}^d \to [0,1]$ is called* joint distribution function *of $X_1, \ldots, X_d$, if for all $B \in \mathscr{B}^d$*

$$F(B) = P\left((X_1, \ldots, X_d) \in B\right) := P\left(\bigcap_{i=1}^{d} \{\omega \in \Omega \mid X_i(\omega) \in B\}\right).$$

Definition 2.24 (Joint density function)**.** *Let $(\Omega, \mathscr{A}, P)$ be a probability space and $X_1, \ldots, X_d : \Omega \to \mathbb{R}$ be real-valued continuous random variables. A function $f : \mathbb{R}^d \to \mathbb{R}$ is called* joint density function *of $X_1, \ldots, X_d$, if for all $B \in \mathscr{B}^d$*

$$P\left((X_1, \ldots, X_d) \in B\right) = \int_B f\left(t_1, \ldots, t_d\right) d\left(t_1, \ldots, t_d\right).$$

Example 2.25. *The generalization of the normal distribution to multiple dimensions is the* multivariate normal distribution. *A vector $\mathbf{X} = (X_1, \ldots, X_d)$ of random variables follows a multivariate normal distribution, if the corresponding joint density function $\phi_d : \mathbb{R}^d \to \mathbb{R}$ is given by*

$$\phi_d(x) = \frac{1}{\sqrt{(2\pi)^d \left|\Sigma\right|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)},$$

where $\mu \in \mathbb{R}^d$ is the mean vector, $\Sigma \in \mathbb{R}^{d \times d}$ is the covariance matrix of X and $|\Sigma|$ its determinant. The covariance matrix is defined by $\Sigma_{ij} = \mathrm{cov}\left[X_i, X_j\right] := E\left[\left(X_i - \mu_i\right)\left(X_j - \mu_j\right)\right]$, it is symmetric positive semidefinite, and its diagonal elements are the variances of the X_i's.

To indicate that the random vector $\mathbf{X}$ follows a multivariate normal distribution with mean vector μ and covariance matrix Σ, we write

$$\mathbf{X} \sim \mathcal{N}_d\left(\mu, \Sigma\right).$$

Finally, we introduce the concept of stochastic independence. A finite number of events $A_1, \ldots, A_d \in \mathscr{A}$ is called independent, if $P(A_1 \cap \ldots \cap A_d) = P(A_1) \cdot \ldots \cdot P(A_d)$. The definition for a real-valued random variable is quite similar.

Definition 2.26 (Independence of random variables)**.** *Let $(\Omega, \mathscr{A}, P)$ be a probability space and $X_1, \ldots, X_d : \Omega \to \mathbb{R}$ be real-valued continuous random variables. Then $X_1, \ldots, X_d$ are called independent, if for all $(x_1, \ldots, x_d) \in \mathbb{R}^d$*

$$P\left(\bigcap_{i=1}^{d} \{\omega \in \Omega \mid X_i(\omega) \leq x_i\}\right) = \prod_{i=1}^{d} P\left(X_i \leq x_i\right).$$

2.3 Density Estimation with the Gaussian Kernel

We now present a short introduction to density estimation with kernels, see [58] for details. The basic scheme of density estimation can be summarized in the following way: Given a d-dimensional dataset

$$D = \{x_1, \ldots, x_N\} \subset \mathbb{R}^d$$

we assume that the points x_i have been generated by independent and identically-distributed random variables X_i. Identically distributed means, that they have the same underlying distribution, thus the same probability density function f. The goal is to find a "good" estimate for f.

The reconstruction of the unknown continuous pdf from the discrete data is an ill-posed problem due to various aspects. First, if no further constraints are made, there exist infinitely many solutions with different degrees of smoothness. This issue is treated by *regularization*, that is reformulating the problem using certain smoothness assumptions. In the context of statistical learning one should also keep in mind that the resulting density is generally intended to be used for predictions for new data. A perfect fit for the given data is not necessarily the best solution. Eventually the data is likely to be tainted with errors, since it consists of (measured) results from stochastic experiments. A detailed discussion of various methods of regularization theory can be found in chapter 4 of [56].

In the context of density estimation, the terms parametric and nonparametric estimation have been emerged. Parametric estimation is applied if the density f is known to be of a certain structure with some unknown parameters, for instance if the underlying random variable is normally distributed, $X \sim \mathcal{N}\left(\mu, \sigma^2\right)$, then the mean μ and the standard deviation σ are the parameters to be estimated.

If on the other hand nothing is known about the structure of the density function a priori, there are diverse methods for estimating the entire density function. These methods are subsumed under the term "nonparametric".

For our further discussion we stick to the one-dimensional case, since the basic principles can thus be grasped much easier.

2.3.1 Histograms

The simplest and presumably oldest nonparametric density estimator is the histogram. We give a short description of density estimation with histograms, since it shares an important

aspect with kernel density estimation. Assume the given one-dimensional dataset $D \subset \mathbb{R}$ to be arranged such that

$$D = \{x_1 \leq x_2 \leq \ldots \leq x_N\}\,.$$

The interval $[x_1, x_N]$ is then partitioned into M smaller intervals $I_1, \ldots, I_M$ – called bins – of equal bin width $h = \frac{x_N - x_1}{M}$. The height of each bin is proportional to the number of data points in its interval. The approximated density function $\widetilde{f}_{hist}$ thus is a piecewise constant function that is normed such that it integrates up to 1:

$$\widetilde{f}_{hist}(x) = \frac{1}{N \cdot h} \sum_{j=1}^{M} \sum_{i=1}^{N} \mathcal{X}_{I_j}(x_i) \cdot \mathcal{X}_{I_j}(x), \tag{2.10}$$

where $\mathcal{X}_{I_j}$ is the indicator function of I_j, which is 1 for all $x \in I_j$ and 0 elsewhere.

Histograms are widely-used mainly since they are easy to calculate and interpret. However a density function approximation obtained by histograms is neither differentiable nor continuous. Furthermore, the bin width has crucial influence on the quality of the approximation as can be clearly seen in figure 2.1. An extensive discussion of histograms, especially of methods to determine the optimal bin width, is presented in the third chapter of [58]; generalizations of histograms can be found in the subsequent two chapters.

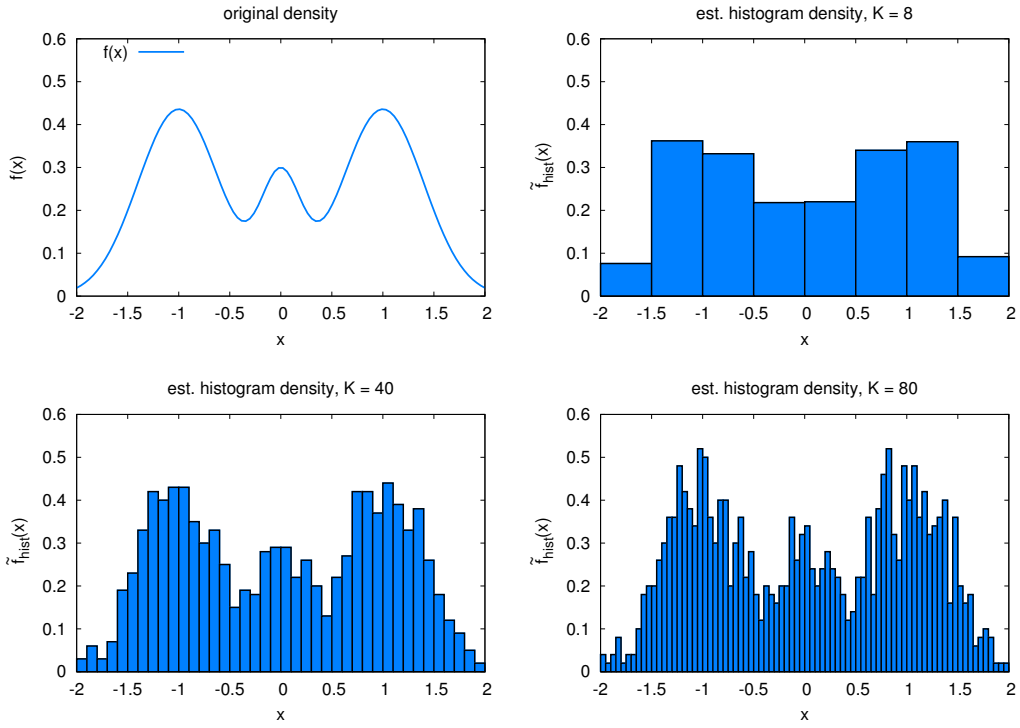


FIGURE 2.1: A given density function *(top left)* has been sampled with 1000 points. The sample is visualized by histograms with different bin width.

2.3.2 Kernel Density Estimation

Kernel density estimation (KDE) is nowadays the most popular form of nonparametric density estimation. The basic idea is quite similar to the histogram approach. In case of the histogram, a bin segment of fixed height is assigned to each point, then all these bin segments are added up to get the final histogram. In KDE, to each point of the sample a weighted kernel function centered at the given point is assigned, then all these functions are added up. Thus, the general version of a kernel density estimator with kernel function $k(x)$ is given by

$$\widetilde{f}_h(x) = \frac{1}{N \cdot h} \sum_{i=1}^{N} w_i \cdot k\left(\frac{x - x_i}{h}\right) = \frac{1}{N} \sum_{i=1}^{N} w_i \cdot k_h\left(x - x_i\right), \tag{2.11}$$

where $k_h(x) = k(x/h)/h$. Since $\widetilde{f}_h$ is the approximated probability density function, the kernel function must satisfy $\int_{-\infty}^{\infty} k(x)dx = 1$. The weights are often constant, some authors assume $w_i = 1$ a priori. Usual kernel functions are bell-shaped or peak-shaped. Some prominent examples are presented below:

Obviously, the class of kernel functions can be partitioned in those with bounded and unbounded domain. When it comes to evaluating the kernel numerically, the functions with bounded domain are usually privileged, since they completely ignore points outside their domain and thus limit the computational effort. However, they are not differentiable at the boundaries, while the Gaussian kernel is infinitely differentiable on the whole domain and moreover also positive definite, as shown above. Ultimately, the choice of an appropriate kernel mainly depends on the current problem and no particular kernel can be recommended for all purposes. Yet, the Gaussian kernel, which has also been examined theoretically quite in-depth, is one of the most widely-used kernels in KDE. The kernel density estimator with Gaussian kernel is obviously given by

$$\widetilde{g}_h(x) = \frac{1}{\sqrt{2\pi} \cdot N \cdot h} \sum_{i=1}^{N} w_i \cdot e^{-(x-x_i)^2/h^2}. \tag{2.12}$$

This is the 1-dimensional version of the Gauss Transform.

In cases where no particular kernel is provided by the problem statement, various investigations have discovered that the quality of the density estimate mainly depends on the choice of the smoothing parameter h (also called bandwidth) rather than on the choice of the kernel function [58], [54]. This is the above-mentioned analogon to the histogram approach, where the choice of the bin width is a crucial issue. The effect of varying the smoothing parameter can be seen in figure 2.2. Choosing the bandwidth too large will result in oversmoothing, the characteristics of the original density function are smoothed out. A low bandwidth on the other hand overemphasizes the influence of single points resulting in a spiky function with many peaks. The choice of the optimal bandwidth is a field of various techniques (e. g. cross-validation, plug-in methods) with their particular advantages and drawbacks. A quick review can be found in [37], an extensive discussion is presented for instance in [63].

Speaking of an *optimal* bandwidth, we have to be able to rate the quality of the estimated density. For this purpose, the most popular error measures will be introduced.

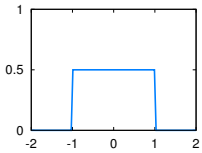
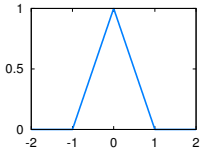
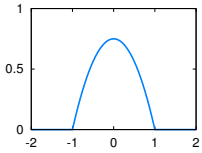
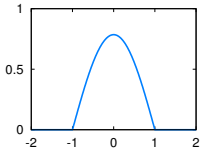
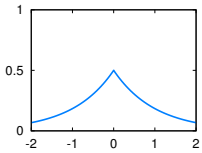
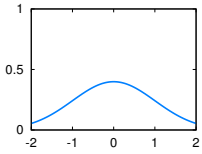
Kernel	Equation	Plot
Uniform	$k(x) = \frac{1}{2}\mathcal{X}_{[-1,1]}(x)$	
Triangle	$k(x) = (1 - \lvert x\rvert)_+$	
Epanechnikov	$k(x) = \frac{3}{4}(1 - x^2)_+$	
Cosine	$k(x) = \frac{\pi}{4}\cos\left(\frac{\pi}{2}x\right)\mathcal{X}_{[-1,1]}(x)$	
Double exponential	$k(x) = \frac{1}{2}e^{-\lvert x\rvert}$	
Gaussian	$k(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$	

TABLE 2.1: Some common kernel functions used in density estimation.

2.3.3 Error Measures

At first, we suppose that the original density function f is given explicitly. To estimate the distance between an approximated density function $\widetilde{f}_h$ and f, a common way is to use an L_p-norm, especially the standard L_1, L_2 and L_∞ versions. In KDE, the L_2-norm has turned out to be the most popular choice due to its easy handling attributes and is referred to as *integrated squared error* or *ISE*:

$$ISE(h) = \int_{\mathbb{R}} \left(\widetilde{f}_h(x) - f(x)\right)^2 dx.$$

Nota bene in this setting, the estimator $\widetilde{f}_h(x)$ is interpreted as a random variable derived from the random variables X_i that generate the sample points x_i. Thus, the *ISE* itself is a

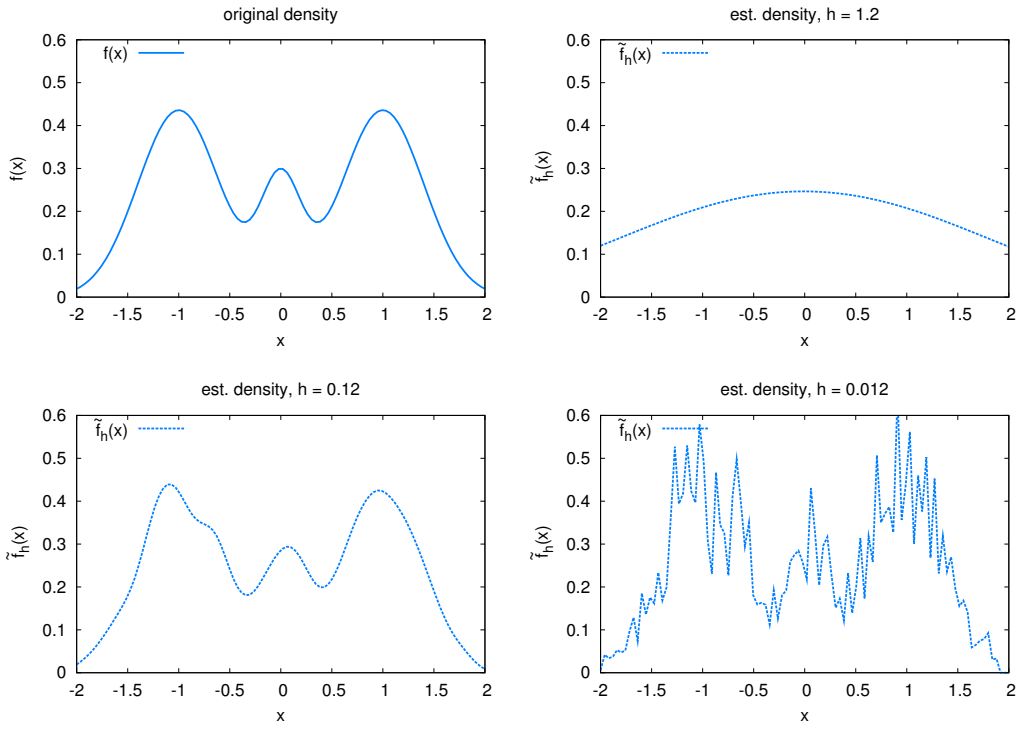


FIGURE 2.2: A given density function *(top left)* has been sampled with 1000 points. KDE on the sample using the Gaussian kernel results in different estimated density functions for bandwidths $h = 1.2$ *(top right)*, $h = 0.12$ *(bottom left)* and $h = 0.012$ *(bottom right)*.

(rather complicated) random variable. In order to get a real-valued quantity to describe the quality of the estimator, that is independent from the number and distribution of sample points, the expected value is applied to get the *mean integrated squared error* or *MISE*:

$$MISE(h) = E\left[ISE(h)\right] = E\left[\int_{\mathbb{R}} \left(\widetilde{f}_h(x) - f(x)\right)^2 dx\right].$$

Notice further, the *mean squared error* or *MSE* can be rewritten in the following way:

$$\begin{aligned} MSE(h) &= E\left[\left(\widetilde{f}_h - f\right)^2\right] \\ &= E\left[\left(\widetilde{f}_h - E\left[\widetilde{f}_h\right] + E\left[\widetilde{f}_h\right] - f\right)^2\right] \\ &= E\left[\left(\widetilde{f}_h - E\left[\widetilde{f}_h\right]\right)^2\right] + E\left[\left(E\left[\widetilde{f}_h\right] - f\right)^2\right] \\ &\quad +2 \cdot \underbrace{E\left[\widetilde{f}_h \cdot E\left[\widetilde{f}_h\right] - \widetilde{f}_h \cdot f - E\left[\widetilde{f}_h\right] \cdot E\left[\widetilde{f}_h\right] + E\left[\widetilde{f}_h\right] \cdot f\right]}_{=0} \end{aligned}$$

$$= \mathrm{Var}\left[\widetilde{f}_h\right] + \mathrm{Bias}^2\left[\widetilde{f}_h\right],$$

where $\mathrm{Bias}\left[\widetilde{f}_h\right] = E\left[\widetilde{f}_h\right] - f$. Applying Fubini's Theorem (see e. g. [4]), the *MISE* can now be rewritten as

$$\begin{aligned} MISE(h) &= E\left[\int_{\mathbb{R}} \left(\widetilde{f}_h(x) - f(x)\right)^2 dx\right] \\ &= \int_{\mathbb{R}} E\left[\left(\widetilde{f}_h(x) - f(x)\right)^2\right] dx \\ &= \int_{\mathbb{R}} \mathrm{Var}\left[\widetilde{f}_h\right] dx + \int_{\mathbb{R}} \mathrm{Bias}^2\left[\widetilde{f}_h\right] dx. \end{aligned}$$

Many bandwidth selection methods seek to find the bandwidth $\hat{h}$ that minimizes the *MISE* of the corresponding estimator. The aspect of oversmoothing and undersmoothing is often referred to as *bias-variance tradeoff* in this context. Minimizing the bias, that describes the mean distance of the estimator to the original density, results in overfitting of the data and thus in a high variance depending on the chosen sample. Minimizing the variance on the other hand has the effect of oversmoothing and therefore entails a high bias value.

The *MISE* itself can not be used to determine the optimal bandwidth $\hat{h}$ in a straightforward way, since the dependence on h is rather complex. Instead, an *asymptotic MISE* or *AMISE* is derived via Taylor series expansion and the bandwidth h^* that minimizes the *AMISE* is determined.

For a nonnegative kernel density estimator with a kernel function k satisfying $\int_{\mathbb{R}} k(x)dx = 1$, $\int_{\mathbb{R}} x \cdot k(x)dx = 0$ and $\int_{\mathbb{R}} x^2 k(x)dx = \sigma^2 > 0$ the *AMISE* is defined by[1]

$$AMISE(h) = \frac{R(k)}{N \cdot h} + \frac{1}{4}\sigma^4 h^4 R\left(f''\right), \tag{2.13}$$

where N is the number of samples, $R(g) = \int_{\mathbb{R}} g^2(x)dx$ and f'' is the second derivative of f.

The first term represents the asymptotic integrated variance and the second term the asymptotic integrated squared bias. The bandwidth h^* that minimizes equation (2.13) is given by

$$h^* = \left[\frac{R(k)}{\sigma^4 R\left(f''\right) N}\right]^{1/5}. \tag{2.14}$$

Both equations still depend on the second derivative of the unknown original density f. The estimation of the term $R\left(f''\right)$ is performed in various different ways by methods such as *biased cross-validation* or *solve-the-equation plug-in*. For a deeper insights, the reader is again referred to the excellent survey in [63].

[1] The density function f is assumed to be differentiable 3 times, where f'' is absolutely continuous and $f''' \in L_2$.

2.4 Gaussian Process Regression

The next topic in statistical learning theory we are going to discuss is Bayesian regression via Gaussian processes, also known as Gaussian process regression. Extensive studies on Gaussian process regression can be found in [47], further good resources are [22] and [56]. Regression is closely related to density estimation. The basic task is the reconstruction of a function from sampled data. Suppose we are given a dataset $D = (X, Y)$ with

$$X = (x_1, \ldots, x_N) \in \left(\mathbb{R}^d\right)^N,$$
$$Y = (y_1, \ldots, y_N) \in \mathbb{R}^N,$$

where x_i are the sampling points and y_i the observed scalar output values. The output Y is supposed to be the result of a function evaluation plus some noise

$$y_i = f(x_i) + \epsilon_i.$$

The task is to find a good estimate for the underlying function f. Gaussian process regression makes some further assumptions about the structure of the function as well as the noise. We first introduce some statistical concepts of Bayesian regression such as *likelihood*, *prior* and *posterior distributions* in an intuitional way.

2.4.1 Bayesian Estimation

The Bayesian approach uses probability distributions (and their associated density functions, respectively) as a key concept to specify model attributes. Given the data and a particular model, the techniques of probability theory are then applied to draw conclusions for instance about the suitability of the model. We first give a quick summary of the whole process before exploring further details.

Suppose we are given some sampling points X and associated output values Y that shall be analyzed using a model M with certain hypotheses about its structure. First, we specify our basic a-priori-assumptions about the general properties of the model and encode this information in a probability density function $p(M)$. The reconstruction process usually depends on some functional relation or a set of parameters W and we can thus introduce another prior density function[2] $p(W|M)$ expressing our prior beliefs about the value of the parameters. Once the data is known, the model should allow us to make predictions about the plausibility of the output data given the sampling points, the particular parameters and the model hypotheses. The corresponding density function $p(Y|X, W, M)$ is typically called *likelihood*. In the next step, Bayes' law (that in fact generalizes from probabilities to probability distributions and densities) is applied in order to get the following *posterior density function*:

[2] Nota bene we use the slightly suboptimal, but standard notation of Bayesian estimation, where $p(M)$ is a different function than $p(W|M)$, the actual function thus depending on the names of the variables rather than the function name itself.

$$\underbrace{p(W|Y,X,M)}_{posterior} = \frac{\overbrace{p(Y|X,W,M)}^{likelihood} \cdot \overbrace{p(W|M)}^{prior}}{\underbrace{p(Y|X,M)}_{evidence}}. \tag{2.15}$$

The *posterior density function* provides an estimate for the value of the parameters W given the current data and model. The evidence $p(Y|X,M)$ is independent of the weights and considered a normalizing constant, therefore equation (2.15) is often written as

$$p(W|Y,X,M) \propto p(Y|X,W,M) \cdot p(W|M), \tag{2.16}$$

where $a \propto b$ reads as "a is proportional to b".

In case the evidence $p(Y|X,M)$ can be evaluated explicitly, a second step can optionally be performed. Applying Bayes' law another time yields

$$p(M|X,Y) = \frac{p(Y|X,M) \cdot p(M)}{p(Y)}. \tag{2.17}$$

The posterior density function $p(M|X,Y)$ allows to draw conclusions about different models M given our current data X, Y.

Whenever a specific model is considered without questioning the model characteristics, equation (2.15) reduces to

$$p(W|Y,X) = \frac{p(Y|X,W) \cdot p(W)}{p(Y|X)}. \tag{2.18}$$

2.4.2 Gaussian Processes

Gaussians processes are the main ingredient to specify the prior assumptions. The basic motive is the assumption that observations close to each other are correlated in some way. The coupling is assumed to be due to the covariance matrix of a normal distribution. The formal definition is provided below.

Definition 2.27 (Gaussian process)**.** *Let X be an arbitrary index set and $g(x)$, $x \in X$ be a* stochastic process*, that is a collection of random variables $\{g(x)|x \in X\}$. Then $g(x)$ is called* Gaussian process*, if for any $n \in \mathbb{N}$ and $x_1, \ldots, x_n \in X$ the random variables $g(x_1), \ldots, g(x_n)$ are normally distributed.*

In the context of regression, we naturally assume a finite $X = \{x_1, \ldots, x_N\}$ and thus a finite vector of random variables $\mathbf{g} = (g(x_1), \ldots, g(x_N))$. We further denote by $\mu \in \mathbb{R}^N$ the mean of the distribution and by K the covariance matrix, thus $\mathbf{g} \sim \mathcal{N}_N(\mu, K)$. In the standard setting, μ is chosen as zero and a positive semidefinite kernel $k(s,t)$ is used as generating function for the entries of the covariance matrix $K_{ij} = \operatorname{cov}[g(x_i), g(x_j)] = k(x_i, x_j)$.

A common choice for the kernel (see e. g. [47], [22]) is the Gaussian covariance kernel

$$k(x,y) = e^{-\|x-y\|^2/h^2}. \tag{2.19}$$

The *prior* specifying our assumptions about the functional relation between sampling points and output data can now be formalized in terms of the density function

$$p(\mathbf{g}) = \frac{1}{\sqrt{(2\pi)^N |K|}} e^{-\frac{1}{2}\mathbf{g}^T K^{-1}\mathbf{g}}. \tag{2.20}$$

Considering our standard regression model

$$y_i = g(x_i) + \epsilon_i,$$

we still have to make another a-priori assumption about the noise ϵ_i, that will allow us to calculate the likelihood $p(Y|X,\mathbf{g})$. The most common procedure (see [47]) is to assume that ϵ_i are independent, identically distributed random variables having Gaussian distribution with zero mean and variance σ^2, formally

$$\epsilon_i \sim \mathcal{N}\left(0, \sigma^2\right).$$

Apart from being a standard assumption about the noise whenever no particular information is available, the choice of normally distributed noise is also theoretically beneficial, since all distributions are thus normal allowing us to derive exact solutions. The *likelihood* encoding our assumptions about the noise $\epsilon_i = y_i - g(x_i)$ is consequently given by the density function

$$\begin{aligned} p\left(Y|X,\mathbf{g}\right) &= \prod_{i=1}^{N} p\left(y_i|x_i,\mathbf{g}\right) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-(y_i - g(x_i))^2/2\sigma^2} \\ &= \frac{1}{\left(2\pi\sigma^2\right)^{N/2}} e^{-\frac{1}{2}(\mathbf{y}-\mathbf{g})^T\left(\sigma^2 I\right)^{-1}(\mathbf{y}-\mathbf{g})}, \end{aligned} \tag{2.21}$$

where $\mathbf{y} = (y_1, \ldots, y_N)$.

We can finally calculate the *posterior* density function as

$$\begin{aligned} p\left(\mathbf{g}|Y,X\right) &\propto p\left(Y|X,\mathbf{g}\right) \cdot p(\mathbf{g}) \\ &\propto e^{-\frac{1}{2}(\mathbf{y}-\mathbf{g})^T\left(\sigma^2 I\right)^{-1}(\mathbf{y}-\mathbf{g})} \cdot e^{-\frac{1}{2}\mathbf{g}^T K^{-1}\mathbf{g}}. \end{aligned} \tag{2.22}$$

2.4.3 Maximum-a-posteriori Approximation

Now that we have a posterior distribution, the question arises how to derive inferences from it. One method to accommodate this is the "maximum-a-posteriori approximation". The root idea is to favor the particular choice of prior parameters (in our case the random vector $\mathbf{g}$) which maximizes the posteriori distribution, formally

$$\mathbf{g}_{\mathrm{MAP}} = \operatorname*{argmax}_{\mathbf{g}} p(\mathbf{g}|Y,X), \tag{2.23}$$

where $\mathbf{g}_{\mathrm{MAP}}$ is called *maximum-a-posterior estimate*. Since the denominator in the posterior $p(\mathbf{g}|Y,X) = p\left(Y|X,\mathbf{g}\right) \cdot p(\mathbf{g}) / p\left(Y|X\right)$ does not depend on $\mathbf{g}$, it is justified to omit the evidence

$p(Y|X)$ for the purpose of maximization. We therefore have to maximize the right-hand side of (2.22), which is accomplished by minimizing the negative logarithm of the posterior:

$$\mathbf{g}_{\text{MAP}} = \underset{\mathbf{g}}{\operatorname{argmin}} \left(-\ln p(\mathbf{g}|Y,X)\right) = \underset{\mathbf{g}}{\operatorname{argmin}} \left(\frac{1}{2\sigma^2} \|\mathbf{y}-\mathbf{g}\|^2 + \frac{1}{2}\mathbf{g}^T K^{-1}\mathbf{g} \right). \tag{2.24}$$

The existence and representation of a solution of the minimization problem (2.24) is provided by the *Representer Theorem*. The classical Representer Theorem was first published by Kimeldorf and Wahba in [40]. We cite a generalized version by Schölkopf, Herbrich and Smola [55]:

Theorem 2.28 (Nonparametric Representer Theorem)**.** *Let $\mathcal{X}$ be a nonempty set, $k : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ a positive definite kernel and $\{x_1,\dots,x_N\} \subset \mathcal{X}$ sampling points with associated output values $\{y_1,\dots,y_N\} \subset \mathbb{R}$. Let further $g : [0,\infty) \to \mathbb{R}$ be a strictly monotonically increasing function, $c : \left(\mathcal{X}\times\mathbb{R}^2\right)^N \to \mathbb{R}\cup\{\infty\}$ an arbitrary loss function, and $\mathcal{H}_k$ the reproducing kernel Hilbert space for the kernel k with its associated norm $\|\cdot\|_k$, that, for a function $f(\cdot) = \sum_{i=1}^{\infty}\beta_i k(\cdot,z_i)$, is given by*

$$\|f\|_k^2 = \sum_{i=1}^{\infty}\sum_{j=1}^{\infty}\beta_i\beta_j k(z_i,z_j).$$

Then any $f \in \mathcal{H}_k$ minimizing the regularized risk functional

$$J(f) = c\left[(x_1,y_1,f(x_1)),\dots,(x_N,y_N,f(x_N))\right] + g(\|f\|_k)$$

has a finite representation of the form

$$f(\cdot) = \sum_{i=1}^{N}\alpha_i k(\cdot,x_i).$$

The loss function c basically measures the quality of the prediction function f on the sample points x_i given the observed output y_i; it therefore satisfies $c\left[(x_1,y_1,y_1),\dots,(x_N,y_N,y_N)\right] = 0$ for any $x_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$ and usually takes only nonnegative values.

The Representer Theorem allows to express the solution in terms of a finite sum of expansions about the sampling points x_i. The proof and a semiparametric variant can be found in [55].

For our minimization problem (2.24), the solution vector $\boldsymbol{\alpha}_{\text{MAP}} = (\alpha_1,\dots,\alpha_N)$ can actually be written (see e. g. [47]) as

$$\boldsymbol{\alpha}_{\text{MAP}} = \left(K+\sigma^2 I\right)^{-1}\mathbf{y}. \tag{2.25}$$

2.4.4 Iterative Solution of the Linear System

For large N, equation (2.25) is not solved by direct matrix inversion requiring $\mathcal{O}(N^3)$ operations, but rather by iterative methods such as *conjugate gradients* (CG) or *generalized minimal residual* (GMRES) [61], [26].

The CG and GMRES both belong to the class of Krylov subspace methods to solve general N-dimensional linear equations $Av = b$ for given $A \in \mathbb{R}^{N\times N}$, $b \in \mathbb{R}^N$ and unknown

$v \in \mathbb{R}^N$. Starting with an initial vector v_0, an iterative process generates new vectors $v_k \in \text{span}\left\{b, Ab, \ldots, A^{k-1}b\right\}$ converging towards the exact solution and theoretically reaching it after at most N steps (in practice, roundoff errors might prevent this). While the CG method only works with symmetric positive definite matrices, the GMRES method can handle arbitrary non-singular matrices. The crucial fact for our purpose is that the most time-consuming step in the iteration process requires a matrix-vector multiplication.

In our case, the multiplication to be carried out, is given by

$$\widetilde{v}_{k+1} = \left(K + \sigma^2 I\right)\widetilde{v}_k = K\widetilde{v}_k + \sigma^2\widetilde{v}_k. \tag{2.26}$$

Remember K is the covariance matrix, and in case the Gaussian kernel $k(x,y) = e^{-\|x-y\|^2/h^2}$ is chosen as generating function, the matrix-vector multiplication $K\widetilde{v}_k$ turns out to be the discrete Gauss Transform:

$$\widetilde{v}_{k+1}(j) = \sum_{i=1}^{N} \widetilde{v}_k(i) \cdot e^{-\|x_i - x_j\|^2/h^2} + \sigma^2\widetilde{v}_k(j), \qquad j = 1, \ldots, N. \tag{2.27}$$

Notice in this context, the Gauss transform is carried out in each step of the iteration with the same Gaussian matrix K, but different weight vector $\widetilde{v}_k$. Furthermore, since the Gaussian kernel is positive definite, we can apply the CG method that is simpler and faster than the GMRES method. In the whole regression process the solution of the linear system (2.25) is the most time-consuming step, therefore algorithms to speed up the Gauss transform are of great practical importance.

2.5 Regularized Least-Squares Classification

The final application in statistical learning theory we are going to present is a common type of binary classification called *regularized least-squares classification*. The task of classification is in general closely related to regression. As a matter of fact the methods for Gaussian process regression illustrated above can be transformed to methods for Gaussian process classification using techniques such as *logistic regression* or *probit regression* [47]. We focus on regularized least-squares classification (RLSC), as it directly leads to a scenario where the Gauss transform can be applied. Deeper insights on the topic of RLSC can be found in [51] and [50].

In binary classification, we are given a dataset $D = (X, Y)$ with

$$X = (x_1, \ldots, x_N) \in \left(\mathbb{R}^d\right)^N,$$
$$Y = (y_1, \ldots, y_N) \in \{0,1\}^N,$$

where x_i are the sampling points and y_i the observed *binary* output values. As in the case of regression, the output Y is supposed to be the result of a function evaluation plus some noise

$$y_i = f(x_i) + \epsilon_i, \tag{2.28}$$

where we wish to find a good estimate for f.

We now assume explicitly that the target function f in (2.28) is an element of the RKHS $\mathcal{H}$ with the associated reproducing kernel k. In this context, the Hilbert space should be thought of as a space of functions with a certain degree of smoothness, for instance square integrable functions with square integrable derivatives up to a certain order. The associated norm $\|\cdot\|_k$ of the Hilbert space would then be a linear combination of the L_2-norm of the function and its derivatives.

The naive problem we would like to solve can now be written as a minimization problem

$$\min_{f\in\mathcal{H}} \frac{1}{N}\sum_{i=1}^{N} V\left(y_i, f(x_i)\right), \tag{2.29}$$

where $V : \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ is some error measure for the distance between the predicted outputs $f(x_i)$ and the measured outputs y_i. As we already discussed in context of density estimation, this precise minimization problem is ill-defined and it does not take into account the measurement errors ϵ_i. A popular approach to tackle this problem is referred to as *Tikhonov regularization*, which can be formalized in the following modified minimization task:

$$\min_{f\in\mathcal{H}} \frac{1}{N}\sum_{i=1}^{N} V\left(y_i, f(x_i)\right) + \lambda \|f\|_k^2 \tag{2.30}$$

The regularization term $\lambda \|f\|_k^2$ provides a measure for the smoothness of the function f, the regularization parameter λ controls the tradeoff between the accuracy and the smoothness of the solution. The choice of the scalar $\lambda > 0$ is a matter of intense discussions and various approaches, a good overview can be found in [39] for instance.

The choice of the error cost function V leads to different learning schemes. The *hinge loss function* $V\left(y, f(x)\right) = \left(1 - y\cdot f(x)\right)_+$ leads to the widespread theory of *support vector machines* [15], while the *square loss function* $V\left(y, f(x)\right) = \left(y - f(x)\right)^2$ gives rise to least-squares methods, in this case the regularized least-squares classification.

While the support vector machine approach requires the solution of a convex optimization problem, the regularized least-squares classification can be reduced to the solution of a linear equation system by means of the Representer Theorem (2.28). When choosing the square loss function, the Representer Theorem guarantees the existence of a unique solution f_{min} of the minimization problem (2.30) that can be expressed as

$$f_{min}(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i). \tag{2.31}$$

We can therefore rewrite (2.30) as

$$\min J(\boldsymbol{\alpha}) = \min_{\boldsymbol{\alpha}\in\mathbb{R}^N} \frac{1}{N}(\mathbf{y} - K\boldsymbol{\alpha})^T(\mathbf{y} - K\boldsymbol{\alpha}) + \lambda\boldsymbol{\alpha}^T K\boldsymbol{\alpha}, \tag{2.32}$$

where $\mathbf{y} = (y_1, \ldots, y_N)$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)$ and $K_{ij} = \left(k(x_i, x_j)\right)_{i,j=1,\ldots,N}$. Since the above expression is a convex and differentiable function, we can calculate its minimum by finding the root of the derivative with respect to α:

$$\nabla J(\boldsymbol{\alpha}) = \frac{2}{N}(\mathbf{y} - K\boldsymbol{\alpha})^T(-K) + 2\lambda K\boldsymbol{\alpha} = 2\left(-\frac{1}{N}K\mathbf{y} + \frac{1}{N}K^2\boldsymbol{\alpha} + \lambda K\boldsymbol{\alpha}\right).$$

Choosing the Gaussian kernel, the kernel matrix K is positive definite and in particular nonsingular, thus the minimum can be found be solving the following linear system:

$$(K + \lambda N \cdot I)\,\boldsymbol{\alpha} = \mathbf{y}. \tag{2.33}$$

We finally have a linear system with the Gaussian kernel matrix, that can be solved in the same way as (2.25) in the case of Gaussian Process regression, for instance using the method of conjugate gradients.

We conclude our discussion on the applications that benefit from fast algorithms for the Gauss transform. In the next chapter, we present the basic theoretical principles that are necessary to understand the approximation algorithms that will be introduced in chapter 4.

3 Theoretical Preliminaries and Algorithmic Approaches

The purpose of this chapter is to introduce the theoretical and methodological basics required for the approximation algorithms presented in the subsequent chapter, which lead to the speed-ups of the Gauss Transform. In order to find different representations of the discrete Gauss Transform, that are crucial for fast approximate evaluations, we require the Taylor expansion for multivariate functions as well as multi-dimensional Hermite polynomials. These are presented in the next two sections. In the first section, we will basically follow chapter 1.6 of [34].

3.1 Taylor Expansion for Functions of Several Variables

In order to present an economic formulation of the multivariate Taylor expansion we make use of the multi-index notation. A multi-index is – loosely spoken – the canonical generalization of an index to multiple dimensions, thus, a tuple of nonnegative integers. In order to deal with more complex terms involving multi-indices, we give a precise definition.

Definition 3.1 (Multi-index)**.** *Let $d \geq 1$ be an arbitrary, but fixed integer. Let $\alpha_1, \ldots, \alpha_d$ be nonnegative integers. Then $\alpha = (\alpha_1, \ldots, \alpha_d)$ is called a multi-index. Let further $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ and ∂_i be the partial derivative with respect to the ith coordinate in $\mathbb{R}^d$. For any multi-index $\alpha \in \mathbb{N}^d$ we define:*

$$
\begin{aligned}
|\alpha| &:= \alpha_1 + \ldots + \alpha_d \\
\alpha! &:= \alpha_1! \cdot \ldots \cdot \alpha_d! \\
x^\alpha &:= x_1^{\alpha_1} \cdot \ldots \cdot x_d^{\alpha_d} \\
\nabla^\alpha &:= \partial_1^{\alpha_1} \cdot \ldots \cdot \partial_d^{\alpha_d}
\end{aligned}
$$

We write $\alpha > p$ for an integer p, if $\alpha_i > p$ for all $i = 1, \ldots, d$.

Now let Ω be an open subset of $\mathbb{R}^d$ and $C^p(\Omega, \mathbb{R})$ the class of p-times differentiable functions from Ω to $\mathbb{R}$. Then Taylor's theorem can be put as:

Theorem 3.2 (Multivariate Taylor expansion)**.** *Let $f \in C^{p+1}(\Omega, \mathbb{R})$ and $x, x^* \in \Omega$ be two points, such that $\delta x + (1-\delta)x^* \in \Omega$ for any $\delta \in [0,1]$. Then the p^{th}-order Taylor expansion around point x^* is given by*

$$
f(x) = \sum_{|\alpha| \leq p} \frac{1}{\alpha!} (x - x^*)^\alpha \cdot \nabla^\alpha f(x^*) + R_{p+1}(x, x^*),
$$

where the Lagrange remainder $R_{p+1}(x, x^*)$ *can be written as*

$$R_{p+1}(x, x^*) = \sum_{|\alpha|=p+1} \frac{1}{\alpha!} (x - x^*)^\alpha \cdot \nabla^\alpha f(x^* + \theta (x - x^*))$$

with some $\theta \in (0, 1)$.

The following Lemma is useful for switching between multi-index and vectorial notation.

Lemma 3.3. *Let* $u = (u_1, \ldots, u_d), v = (v_1, \ldots, v_d) \in \mathbb{R}^d$ *and* $u \cdot v$ *be the scalar product of* u *and* v. *Then* $(u \cdot v)^n$ *can be rewritten as*

$$(u \cdot v)^n = \sum_{|\alpha|=n} \frac{n!}{\alpha!} u^\alpha v^\alpha. \tag{3.1}$$

Proof. Let $f(x) = \left(\sum_{i=1}^d x_i\right)^n$ where $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$. For any $k_1, k_2 \in \{1, 2, \ldots, d\}$ $\partial_{k_1} f(x) = n \left(\sum_{i=1}^d x_i\right)^{n-1}$, $\partial_{k_1}\partial_{k_2} f(x) = n(n-1) \left(\sum_{i=1}^d x_i\right)^{n-2}$, $\ldots$ and therefore,

$$\begin{aligned} \nabla^\alpha f(0) &= 0 \quad \text{for all } |\alpha| < n, \\ \nabla^\alpha f(0) &= n! \quad \text{for all } |\alpha| = n, \\ \nabla^\alpha f(x) &= 0 \quad \text{for all } |\alpha| > n. \end{aligned}$$

Hence, n^{th}-order Taylor expansion of $f(x)$ around 0 yields

$$f(x) = \underbrace{\sum_{|\alpha|<n} \frac{1}{\alpha!} x^\alpha \cdot \nabla^\alpha f(0)}_{=0} + \sum_{|\alpha|=n} \frac{1}{\alpha!} x^\alpha \cdot n! + \underbrace{R_{n+1}(x, 0)}_{=0}.$$

Letting $x = (u_1 v_1, \ldots, u_d v_d)$ implies the desired result.

□

Using Lemma (3.3) with $u = x - x^*$ and $v = \nabla_y$ we get a slightly different formulation of the Taylor expansion in a vectorial form.

Corollary 3.4 (Multivariate Taylor expansion, vectorial form)**.** *Let* $f \in C^{p+1}(\Omega, \mathbb{R})$ *and* $x, x^* \in \Omega$ *be two points, such that* $\delta x + (1 - \delta)x^* \in \Omega$ *for any* $\delta \in [0, 1]$. *Then the* p^{th}*-order Taylor expansion around point* x^* *can be formulated as*

$$f(x) = \sum_{n=0}^{p} \left[\frac{1}{n!} \left((x - x^*) \cdot \nabla_y \right)^n f(y) \right]_{y=x^*} + R_{p+1}(x, x^*),$$

where the Lagrange remainder $R_{p+1}(x, x^*)$ *can be written as*

$$R_{p+1}(x, x^*) = \left[\frac{1}{(p+1)!} \left((x - x^*) \cdot \nabla_y \right)^{p+1} f(y) \right]_{y=x^* + \theta(x - x^*)}$$

with some $\theta \in (0,1)$.

3.2 Hermite Polynomials and Hermite Functions

Hermite functions are – besides Taylor expansions – another key ingredient of fast approximation algorithms for the Gauss Transform. Since the Hermite functions derive from Hermite polynomials, we first start our considerations with a short outline on orthogonal polynomials. Definitions and Theorems are presented as in [14]. A good reference for numerical and computational issues concerning orthogonal polynomials can be found in [20].

Definition 3.5 (Orthogonal polynomial sequence)**.** *Let* w *be a function that is non-negative and Lebesgue-integrable on an interval* (a,b)*, where* $a \in [-\infty, \infty)$*,* $b \in (-\infty, \infty]$*. Let further be*

$$\int_a^b w(x)dx > 0$$

and in case that (a,b) *is unbounded, let all* moments

$$\mu_n = \int_a^b x^n w(x)dx, \qquad n = 0,1,2,\ldots$$

be finite. A sequence of polynomials $\{P_n(x)\}_{n=0}^{\infty}$*,* $P_n(x)$ *of degree* n*, that satisfies*

$$\int_a^b P_m(x)P_n(x)w(x)dx = 0 \qquad \text{for all } m \neq n$$

is then called an orthogonal polynomial sequence *with respect to the weight function* w *on the interval* (a,b).

We will also refer to an orthogonal polynomial sequence simply as "orthogonal polynomials". One of many interesting properties is the existence of a recurrence relation, that is particularly relevant for our purpose, since it allows fast function evaluations.

Theorem 3.6 (Recurrence Formula)**.** *For any monic orthogonal polynomial sequence* $\{P_n(x)\}$ *there exist (complex) constants* c_n *and* $\lambda_n \neq 0$ *such that*

$$P_{n+1}(x) = (x - c_n)P_n(x) - \lambda_n P_{n-1}(x), \qquad n = 0,1,2,\ldots,$$

where we define $P_{-1}(x) = 0$.

Let us now introduce the 1-dimensional Hermite polynomials. A concise representation can be given by the following formula, that we take as the definition.

Definition 3.7 (1-dimensional Hermite polynomials H_n)**.**

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}, \qquad x \in \mathbb{R}. \tag{3.2}$$

The expression is also called a *Rodrigues' type* formula, since a similar, but more general formula was published by Rodrigues in [53]. The first six Hermite polynomials are presented in Figure 3.1.

$$
\begin{aligned}
H_0(x) &= 1 \\
H_1(x) &= 2x \\
H_2(x) &= 4x^2 - 2 \\
H_3(x) &= 8x^3 - 12x \\
H_4(x) &= 16x^4 - 48x^2 + 12 \\
H_5(x) &= 32x^5 - 160x^3 + 120x
\end{aligned}
$$

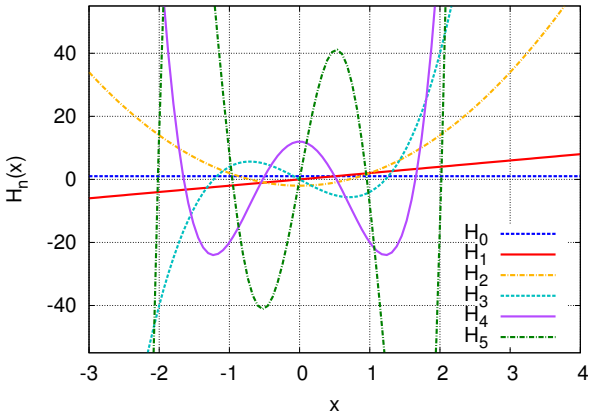


Figure 3.1: The first six Hermite polynomials.

Next we discuss some global properties of Hermite polynomials.

Theorem 3.8 (Properties of Hermite polynomials)**.** *For the* 1*-dimensional Hermite polynomials the following equations hold:*

(i) ***Generating function:***

$$e^{2yx-y^2} = \sum_{n=0}^{\infty} H_n(x)\frac{y^n}{n!}$$

(ii) ***Recurrence formula:***

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)$$

(iii) ***Orthogonality:***

$$\int_{-\infty}^{\infty} H_m(x)H_n(x)e^{-x^2}dx = 2^n n!\sqrt{\pi}\cdot\delta_{nm}$$

Proof. ***Ad (i)***: First rewrite the left side as

$$e^{2yx-y^2} = e^{x^2-x^2+2yx-y^2} = e^{x^2}e^{-(x-y)^2}.$$

A Taylor expansion of $f(x) = e^{-(x-y)^2}$ around $x^* = x + y$ then yields

$$e^{2yx-y^2} = e^{x^2}\sum_{n=0}^{\infty}\frac{(-y)^n}{n!}\frac{d^n}{dx^n}e^{-x^2} = \sum_{n=0}^{\infty}H_n(x)\frac{y^n}{n!}.$$

Ad (ii): We prove the assumption via simultaneous induction for two equations:

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x) \tag{3.3}$$

$$\frac{d}{dx}H_{n+1}(x) \quad = \quad 2(n+1)H_n(x) \tag{3.4}$$

for all $n = 0, 1, 2, \ldots$, where we define $H_{-1}(x) = 0$.

For $n = 0$ and $n = 1$ the equations follow immediately since $H_2 = 4x^2 - 2$, $H_1 = 2x$ and $H_0 = 1$. Now let $N \geq 2$ and let equations (3.3) and (3.4) hold for any $n < N$. We then have

$$\begin{aligned}
H_{N+1}(x) &= (-1)^{N+1} e^{x^2} \frac{d^{N+1}}{dx^{N+1}} e^{-x^2} = \\
&= (-1) \cdot e^{x^2} \frac{d}{dx}\Big(e^{-x^2} \cdot \underbrace{(-1)^N e^{x^2} \frac{d^N}{dx^N} e^{-x^2}}_{=H_N(x)} \Big) = \\
&= (-1) \cdot e^{x^2} \left(-2x \cdot e^{-x^2} \cdot H_N(x) + e^{-x^2} \cdot \frac{d}{dx} H_N(x) \right) = \\
&= 2x \cdot H_N(x) - \frac{d}{dx} H_N(x) = \\
&= 2x \cdot H_N(x) - 2N \cdot H_{N-1}(x),
\end{aligned}$$

where the last step results from the induction hypothesis for $n = N - 1$. We now take the derivative to get

$$\begin{aligned}
\frac{d}{dx}H_{N+1}(x) &= \frac{d}{dx}\Big[2x \cdot H_N(x) - 2N \cdot H_{N-1}(x) \Big] = \\
&= 2 \cdot H_N(x) + 2x \cdot 2N \cdot H_{N-1}(x) - 2N \cdot 2(N-1) H_{N-2}(x) = \\
&= 2 \cdot H_N(x) + 2N \cdot \underbrace{\big[2x \cdot H_{N-1}(x) - 2(N-1) H_{N-2}(x) \big]}_{=H_N(x)} = \\
&= 2(N+1) H_N(x).
\end{aligned}$$

Ad (iii): Let for any m, n with $0 \leq m \leq n$

$$I_{mn} = \int_{-\infty}^{\infty} x^m H_n(x) e^{-x^2} dx = (-1)^n \int_{-\infty}^{\infty} x^m \frac{d^n}{dx^n} e^{-x^2} dx.$$

By means of partial integration we can rewrite

$$(-1)^n I_{mn} = \underbrace{\left[x^m \frac{d^{n-1}}{dx^{n-1}} e^{-x^2} \right]_{-\infty}^{\infty}}_{=0} - m \int_{-\infty}^{\infty} x^{m-1} \frac{d^{n-1}}{dx^{n-1}} e^{-x^2} dx,$$

where the first term vanishes since $\lim_{x \to \pm\infty} \left(p(x) \cdot e^{-x^2} \right) = 0$ for any polynomial $p(x)$. Repeated partial integration yields after m steps

$$(-1)^n I_{mn} = (-1)^m m! \int_{-\infty}^{\infty} \frac{d^{n-m}}{dx^{n-m}} e^{-x^2} dx.$$

Now consider the case where $m < n$. Another integration results in

$$(-1)^n I_{mn} = (-1)^m m! \left[\frac{d^{n-m-1}}{dx^{n-m-1}} e^{-x^2} \right]_{-\infty}^{\infty} = 0 \qquad (m < n). \tag{3.5}$$

However in case that $m = n$ we have

$$I_{nn} = n! \int_{-\infty}^{\infty} e^{-x^2} dx = n! \sqrt{\pi}. \tag{3.6}$$

Equation (3.5) and the linearity of Lebesgue integration result in

$$\int_{-\infty}^{\infty} H_m(x) H_n(x) e^{-x^2} dx = 0 \qquad (m < n).$$

Finally, due to equations (3.5) and (3.6) we have

$$\int_{-\infty}^{\infty} H_n(x) H_n(x) e^{-x^2} dx = c_n \cdot n! \sqrt{\pi},$$

where c_n is the leading coefficient of $H_n(x)$. However from the recurrence formula, it is easily seen that $c_n = 2^n$.

□

Having introduced the most important properties of Hermite polynomials, we can now turn to the closely related Hermite functions $h_n(x)$.

Definition 3.9 (1-dimensional Hermite functions h_n)**.**

$$h_n(x) = e^{-x^2} H_n(x) = (-1)^n \frac{d^n}{dx^n} e^{-x^2}, \qquad x \in \mathbb{R}. \tag{3.7}$$

Theorem 3.10 (Properties of Hermite functions)**.** *For the* 1*-dimensional Hermite functions the following relations hold:*

(i) ***Generating function:***

$$e^{-(x-y)^2} = \sum_{n=0}^{\infty} h_n(x) \frac{y^n}{n!}$$

(ii) ***Recurrence formula:***

$$h_{n+1}(x) = 2x h_n(x) - 2n h_{n-1}(x)$$

(iii) ***Inequality by*** Szász***:***

$$|h_n(x)| \leq \sqrt{2^n n!} \cdot e^{-x^2/2}$$

The first two equations emanate directly from the analogous results for the Hermite polynomials, while the inequality is proven in [62]. We finally present the generalization of the main results to d dimensions.

Definition 3.11 (d-dimensional Hermite polynomials H_α and Hermite functions h_α)**.** *Let $x \in \mathbb{R}^d$ and $\alpha = (\alpha_1, \ldots, \alpha_d)$. Then define*

$$H_\alpha(x) = H_{\alpha_1}(x_1) \cdot \ldots \cdot H_{\alpha_d}(x_d) = (-1)^{|\alpha|} \cdot e^{\|x\|^2} \cdot \nabla_x^\alpha e^{-\|x\|^2}; \tag{3.8}$$

$$h_\alpha(x) = h_{\alpha_1}(x_1) \cdot \ldots \cdot h_{\alpha_d}(x_d) = (-1)^{|\alpha|} \cdot \nabla_x^\alpha e^{-\|x\|^2}. \tag{3.9}$$

Theorem 3.12 (Properties of d-dimensional Hermite functions)**.** *Let $x, y \in \mathbb{R}^d$ and $\alpha = (\alpha_1, \ldots, \alpha_d)$. Then the following relations for d-dimensional Hermite functions hold:*
(i) ***Generating function:***

$$e^{-\|x-y\|^2} = \sum_{\alpha \geq 0} \frac{y^\alpha}{\alpha!} h_\alpha(x) \tag{3.10}$$

(ii) ***Inequality by*** Szász***:***

$$|h_\alpha(x)| \leq \sqrt{2^{|\alpha|}\alpha!} \cdot e^{-\|x\|^2/2} \tag{3.11}$$

The first equation follows from a single application of the multivariate Taylor expansion (3.2) on the function $f(z) = e^{-\|z\|^2}$, while the generalized inequality directly results from the definition of d-dimensional Hermite functions.

3.3 The Fast Multipole Method

The *fast multipole method (FMM)* is an approximation technique to speed up the multiple evaluation of a function $f : \mathbb{R}^d \to \mathbb{R}$ of the form

$$f(t) = \sum_{i=1}^{N} w_i \cdot k(t, s_i) \tag{3.12}$$

at several points $t = t_j$, $j = 1, \ldots, M$. We call

- $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ the *kernel function*,
- $t_j \in \mathbb{R}^d$, $j = 1, \ldots, M$ the *targets*,
- $s_i \in \mathbb{R}^d$, $i = 1, \ldots, N$ the *sources*,
- $w_i \in \mathbb{R}$, $i = 1, \ldots, N$ the *(source) weights*.

The fast multipole method was developed by Greengard and Rokhlin [27] in the context of particle simulations and has since then spread in many areas, e. g. the evaluation of gravitational or electrostatic potentials and fields in physics or density estimation in probability theory. In physical simulations the targets as well as the sources often represent certain particles and hence are the same objects. In this case it is also common to speak of *N-body problems*. In data mining the sources could be existing objects of a database with known qualifiers, while the targets are new input to be qualified by comparison with the values of the sources.

In most applications, the number of targets M approximately equals the number of sources N. For runtime analysis we therefore assume $M = \mathcal{O}(N)$ in our further discussion. The

FMM allows to reduce the complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. Sometimes a simpler variant with $\mathcal{O}(N \log N)$ costs is preferable. In fact, the fast multipole method does not resemble a formula that can be applied to different problems in a straightforward way, but rather comprises a collection of useful techniques that must be adapted to the particular framework. For further reference, the reader should consult [31], [6] and [36], which however address the topic in rather discriminative approaches. Our discussion essentially follows [36].

We first give a sketch of the fundamental structure of the most general variant, that is also referred to as *multilevel* FMM (as opposed to *singlelevel* FMM). Three basic techniques are utilized:

- Expansion of the kernel function.
- Hierarchical space partitioning.
- Translation operators.

At first, we seek a new representation of the kernel function, for instance a Taylor expansion that allows to approximate the function locally. The purpose of this expansion is to break the entanglement of sources and targets that is responsible for the complexity $\mathcal{O}(N^2)$, and find a way to calculate the influence of sources and targets separately. This step usually involves infinite series approximations, hence truncation and local error bounds.

Next, a (hierarchical) space partitioning scheme for the data (the source and target points) is established. This can either be a simple uniform grid that is hierarchically refined or a more complex tree structure specifically adapted to the inherent distribution of the data. The choice of the partitioning scheme crucially depends on the error bounds of the series approximations, and therefore on the decay properties of the kernel. If sources and targets reside in strictly separated regions, two different data structures can be applied. The truncation parameters and error bounds are adjusted to the current space partitioning scheme and, starting at the top level of the hierarchical structure, we descend to find the optimal level to evaluate the particular expansion.

The use of translation operators finally allows to shift local results obtained by evaluations at a certain level to upper and lower levels, respectively. Later on we will see that this technique allows to reduce the complexity from $\mathcal{O}(N \log N)$ to $\mathcal{O}(N)$.

The singlelevel FMM does not utilize a hierarchical, but a "flat" structure. Thus all expansions are carried out at a single level, and no translation operators are required.

A key aspect in the acceleration process is the careful selection and harmonization of the three main concepts; an elaborate series expansion might only result in great performance in combination with an appropriate partitioning scheme, while the translation operators can increase the efficiency crucially in particular scenarios.

We are now going to discuss the FMM techniques in detail considering the physical example of the electrostatic potential due to a number of point charges. Our excursus will basically follow [36], while the mathematical details can be found in [28].

The electrostatic potential at a point $t \in \mathbb{R}^3$ generated by N charges w_i located at points $s_i \in \mathbb{R}^3$ $(i = 1, \ldots, N)$ is given by

$$f(t) = \sum_{i=1}^{N} w_i \frac{1}{\|t - s_i\|}. \tag{3.13}$$

The first task is to find a suitable expansion of the corresponding kernel function $k_{pot}(t, s) = \frac{1}{\|t-s\|}$.

3.3.1 Expansion of the kernel function

Let us first examine the trivial case of an arbitrary *degenerate* or *separable* kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, that means

$$k(t, s) = \sum_{n=1}^{p} \Phi_n(t) \cdot \Psi_n(s) \tag{3.14}$$

for some integer $p \ll N$ and appropriate functions $\Phi_n, \Psi_n : \mathbb{R}^d \to \mathbb{R}$. Then the fast multipole equation (3.12) can be rewritten as

$$f(t) = \sum_{i=1}^{N} w_i \cdot \sum_{n=1}^{p} \Phi_n(t) \cdot \Psi_n(s_i) = \sum_{n=1}^{p} \left(\sum_{i=1}^{N} w_i \cdot \Psi_n(s_i) \right) \Phi_n(t), \tag{3.15}$$

thus the sources and targets are decoupled and the total workload directly reduces to $\mathcal{O}(pN)$.

In practice of course, most kernel functions will not be exactly or even approximately degenerate for all values of s and t. We therefore usually seek to find approximations that hold for certain regions. In the literature, the terms *near-field* and *far-field* expansions have emerged. While *near-field* (also *local*, *inner* or *regular*) expansions are valid for sources and targets close to each other, *far-field* (also *multipole*, *outer* or *singular*) expansions are valid only for sources and targets far away from each other. The expression "multipole", that gives rise to the naming of the FMM, shall be explained soon in this context.

In case of our electrostatic potential, the kernel function is naturally separable when transformed to spherical coordinates. In fact, we can express the kernel as an infinite series,

$$k_{pot}(t, s) = \frac{1}{\|t\|} \sum_{n=0}^{\infty} P_n(\cos\theta) \left(\frac{\|s\|}{\|t\|} \right)^n, \tag{3.16}$$

where θ is the angle between the two points s and t and $P_n(\cdot)$ are the *Legendre polynomials* of degree n, that are orthogonal polynomials on the interval $[-1, 1]$ and can be defined via Rodrigues' formula

$$P_n(u) = \frac{1}{2^n n!} \frac{\partial^n}{\partial u^n} \left[\left(u^2 - 1 \right)^n \right] \tag{3.17}$$

as well as explicitly:

$$P_n(u) = \frac{1}{2^n} \sum_{r=0}^{\lfloor n/2 \rfloor} (-1)^r \binom{n}{r} \binom{2n - 2r}{n} u^{n-2r}. \tag{3.18}$$

The series expansion (3.16) is only convergent for $\frac{\|s\|}{\|t\|} < 1$ and is referred to as a multipole expansion. The reason of this nomenclature lies in the physical background of this particular example. For a given set of charges w_i residing in a small region around the origin, the electrostatic potential at a point t far away from this region can be approximated by only taking into account the first term of (3.16) yielding

$$f(t) \approx \frac{1}{\|t\|} \sum_{i=1}^{N} w_i, \tag{3.19}$$

that is referred to as the field due to the *monopole moment* of the charges w_i. If the sum of the charges cancels out, only the second term of the series is considered, which results in

$$f(t) \approx \frac{1}{\|t\|^2} \sum_{i=1}^{N} w_i \cos\theta_i \, \|s_i\| \,, \tag{3.20}$$

known as the field due to the *dipole moment* of the w_i. Consequently, the complete series expansion is referred to as multipole expansion.

Eventually, equation (3.16) does not quite solve our problem since t and s are still entangled in $\cos\theta$. However a result from the theory of *spherical harmonics* gives us the desired representation.

Theorem 3.13 (Addition Theorem for Legendre Polynomials)**.** *Let $s, t \in \mathbb{R}^3$ and θ be the angle between the vectors s and t. Then*

$$P_n(\cos\theta) = \sum_{m=-n}^{n} Y_n^{-m}(s) \cdot Y_n^m(t). \tag{3.21}$$

Here, Y_n^m are the spherical harmonics (with omitted normalization) defined via the relation

$$Y_n^m(z) = (-1)^m \sqrt{\frac{(n-|m|)!}{(n+|m|)!}} P_n^{|m|}(\cos\theta_z) \cdot e^{im\phi_z}, \tag{3.22}$$

where θ_z and ϕ_z denote the polar and azimuthal angle of z, respectively, and i denotes the imaginary unit. Furthermore, $P_n^m(u)$ are the associated Legendre polynomials *defined for $n \geq 0$, $m \geq 0$ by*

$$P_n^m(u) = (-1)^m \left(1-u^2\right)^{m/2} \frac{\partial^m}{\partial u^m} P_n(u). \tag{3.23}$$

Usually, the spherical harmonics are defined as functions of the two angles (of spherical coordinates), since they do not depend on the distance from the origin. We use the above notation for convenience only. We now have all necessary results to present the final multipole expansion separating the source and target points.

Theorem 3.14 (Multipole Expansion for Potential Problem)**.** *Let N charges w_i be located at source points $s_i \in \mathbb{R}^3$ $(i = 1, \ldots, N)$ situated in a sphere with radius $\|s\|$. Then for any target*

point t outside the sphere ($\|t\| > \|s\|$) the electrostatic potential $f(t)$ is given by

$$f(t) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} M_n^m \cdot \frac{Y_n^m(t)}{\|t\|^{n+1}}, \tag{3.24}$$

where the moments M_n^m are given by

$$M_n^m = \sum_{i=1}^{N} w_i \cdot \|s_i\|^n \cdot Y_n^{-m}(s_i). \tag{3.25}$$

Approximating the infinite series in (3.24) with the first p terms

$$\widetilde{f}_p(t) = \sum_{n=0}^{p-1} \sum_{m=-n}^{n} M_n^m \cdot \frac{Y_n^m(t)}{\|t\|^{n+1}} \tag{3.26}$$

yields the following error bound:

$$\left| f(t) - \widetilde{f}_p(t) \right| \leq \left(\sum_{i=1}^{N} |w_i| \right) \cdot \frac{1}{\|t\| - \|s\|} \left(\frac{\|s\|}{\|t\|} \right)^p. \tag{3.27}$$

To understand the basic approach of the evaluation, let us suppose we are given the same situation as in theorem (3.14), that is a set of N sources enclosed in a sphere and an arbitrary target point far outside. This allows us to choose a reasonably small cutoff parameter p according to (3.27) to meet the desired local error bound. Now, the moments M_n^m can be pre-calculated using some efficient recurrence scheme for the involved spherical harmonics in $\mathcal{O}(pN)$ time. Afterwards, the approximation (3.26) can be calculated for the given target point in $\mathcal{O}(p)$ time.

In order to accomplish this evaluation method for different combinations of source and target points, a hierarchical space partitioning is applied with the effect of grouping (or clustering) sources as well as targets. The according data structures and the exact procedure are described next.

3.3.2 Hierarchical space partitioning and an $\mathcal{O}(N \log N)$ algorithm

In order to present the use of space partitioning in context of fast multipole methods, we first restrict our consideration to a simple uniform partitioning method. We therefore assume all sources and targets to be situated in a hypercube, that is then successively bisected along each dimension. The resulting smaller hypercubes are also referred to as boxes. The induced tree structure is a binary tree in 1D, a quadtree in 2 dimensions (Fig. 3.2), and generally a 2^d-tree in d dimensions.

Notice that the uniform space partitioning has some advantages as opposed to more complex structures. It benefits from a simple implementation and allows fast distance calculations between boxes as well as easy integration of new points. Therefore, it is a good choice in low dimensions and for uniformly distributed data.

In practice however, more advanced approaches are used. A simple improvement can be

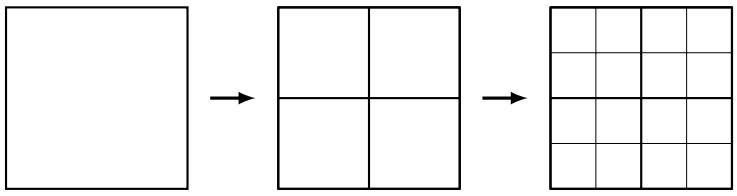


FIGURE 3.2: Hierarchical uniform partitioning in the 2-dimensional case.

made by adaptively refining the tree; naturally, a box containing only a single point (or no points at all) does not require to be divided any further. Besides, the boxes can be shrunk to the minimum size that is necessary to include all points. The purpose of this idea is to optimize the bounds for distance calculations. Finally, for high-dimensional data, a tree with 2^d children per node is not feasible at all. Therefore, usually binary trees are used with diverse and sophisticated rules to split its nodes. A detailed discussion of these tree structures will be presented in the next chapter in context of the particular approximation algorithms.

For now we return to the uniform partitioning scheme that is most suitable to explain the basic procedure of the hierarchical FMM. The fundamental idea is to successively visit each level of the hierarchy, and check for pairs of boxes that are sufficiently far to perform the multipole expansion between the sources in the one and targets in the other box. In this context, we refer to boxes as "source boxes" and "target boxes", respectively, depending on what role they are playing currently. For boxes that are to close to each other, we proceed onto the next level and restart the process with the smaller boxes. This operation continues until the lowest level has been reached, where the interaction of sources and targets, that has been ignored so far, is finally calculated directly. An extract of this procedure is shown in Fig. 3.3.

Let us now return to the previous potential problem and describe the details of the evaluation of the multipole expansion in context of our uniform space partitioning scheme. Remember, for some sources enclosed in a sphere centered at the origin with radius $\|s\|$ and a target t outside the sphere, we have to determine a cutoff parameter p to undercut the given local error bound ϵ with the estimation

$$\left| f(t) - \widetilde{f}_p(t) \right| \leq \frac{W}{\|t\| - \|s\|} \left(\frac{\|s\|}{\|t\|} \right)^p < \epsilon,$$

where $W = \sum_{i=1}^{N} |w_i|$ is the sum of the given charges.

Considering a specific source box with side length l, we let the geometric center of the box be the origin of our new coordinate system. Consequently, all sources lie in a sphere with a radius equaling the space diagonal of the box, thus $\|s\| = \frac{l}{2}\sqrt{d}$, where d is the dimension. We further introduce an integer parameter z referring to the distance of target boxes we take into account for a given source box. While for $z = 0$, all boxes except the source box itself are considered, for $z = 1$ all boxes except adjacent boxes are considered (as done in Fig. 3.3), and so on. For a fixed integer z, the distance of a corresponding target t from the center of the source box can thus be bounded by $\|t\| \geq \frac{2z+1}{2} l$. We can finally give the following estimate for the approximation error

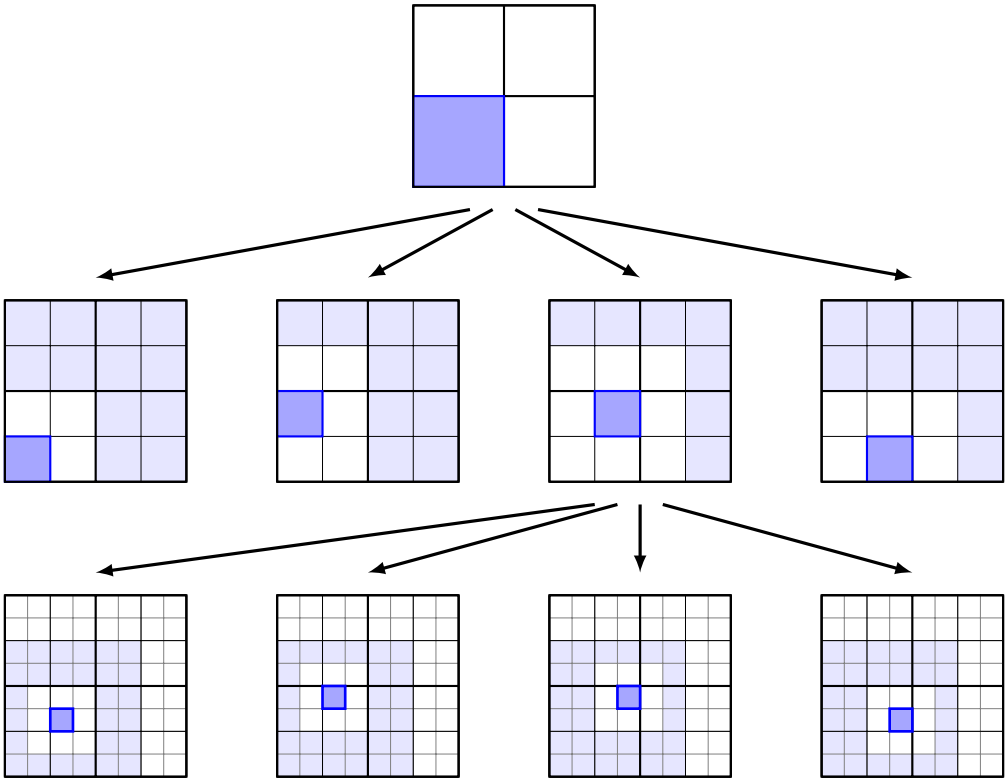


FIGURE 3.3: Descending from level 1 to level 3, each diagram shows the currently considered source box *(blue)* and its interacting target boxes *(light blue)*. Neighboring target boxes as well as target boxes already considered are ignored *(white)*.

$$\left|f(t) - \widetilde{f}_p(t)\right| \leq \frac{2W}{\left(2z+1-\sqrt{d}\right)\cdot l}\left(\frac{\sqrt{d}}{2z+1}\right)^p. \tag{3.28}$$

Requiring this estimate to be smaller than ϵ and solving for p yields the bound

$$p > \left(\log\frac{2W}{\left(2z+1-\sqrt{d}\right)\cdot l} - \log\epsilon\right) \Big/ \log\frac{2z+1}{\sqrt{d}}. \tag{3.29}$$

For a given level of the data structure hierarchy with a fixed box side length l, we can now adapt the parameters z and p using the estimation (3.29) to match the desired local error bound.

Even though we left out some details about combining local error bounds to global ones, the basic concept of the hierarchical FMM procedure should be quite clear by now. However, notice that the current form of the algorithm only allows a runtime bound of $\mathcal{O}(N \log N)$, roughly. This results from the following consideration. The workload for performing the multipole expansion for a single source box b is $\mathcal{O}(p^2 N_b)$, where N_b is the number of sources in b. The

factor of p^2 is due to the double sum in $\widetilde{f}_p(t) = \sum_{n=0}^{p-1} \sum_{m=-n}^{n} M_n^m \cdot \frac{Y_n^m(t)}{\|t\|^{n+1}}$. Adding up all box calculations for a single level takes up costs of $\mathcal{O}(p^2 N)$. The number of levels approximately equals $\log_{2^d} N$, therefore the total costs add up to $\mathcal{O}(p^2 N \log N)$.

A supplemental technique is required to reduce the costs to $\mathcal{O}(N)$. Notice that in the approach described as yet, every single point must be visited on each level in order to calculate the moments for the different source boxes and to evaluate the series expansions at the different target points. So-called "translation operators" will allow us to shift partial results between levels with the effect that only each box must be processed on each level. This finally results in net costs of $\mathcal{O}(1 + 2^d + 4^d + \ldots + N) = \mathcal{O}(N)$.

3.3.3 Translation operators and an $\mathcal{O}(N)$ algorithm

Let us first analyze in detail which work steps must be accelerated specifically. First, the calculation of the moments for each box utilizing all inner points on each level can be avoided by translating moments from lower levels (finer scales) to higher levels (coarser scales). More precisely, we develop a translation operator that – for a certain parent node – combines the moments of all children to the moments of that node. Next, we seek to speed up the evaluation of the multipole expansions at the target points. For this purpose, instead of evaluating expansions of different source boxes at the same target box right away, we combine the contributions so that only one evaluation per target is required. Two new translation operators will serve us to this end. The first one shifts several expansions to a new common center, more precisely the target box center. The second operator transfers expansions centered at a given target box to its children, thus allowing to shift evaluations between tree levels.

Summarizing these observations, we require the following operators:

- *Multipole-translation operator*: An operator allowing to shift moment calculations from children nodes to their parent node.
- *Multipole-to-local operator*: An operator allowing to transfer expansions of multiple source boxes to a new center.
- *Local-translation operator*: An operator allowing to shift expansions from a given target box to its children boxes.

Once we have constructed these operators, the fully featured fast multipole method can be carried out in four basic steps:

1. Calculate the moments for all boxes on the lowest tree level.
2. In a bottom-up pass, use the multipole-translation operator to calculate moments on the next upper level iteratively until the top level (tree root) is reached.
3. The top-down pass beginning at the root consists of two parts per level: first, for each target box use the multipole-to-local operator to transfer expansions from interacting (sufficiently far) source boxes to the particular target box center. Thereafter, use the local-translation operator to shift the expansions from each target box to its children on the next lower level.

4. When the bottom level is reached, evaluate the multipole expansions at each single target point and add the contributions of remaining near neighbor source points directly.

The first and the last step obviously require $\mathcal{O}(N)$ time each. If we are able to perform the evaluation of the translation operators in constant time, the bottom-up and top-down pass each can be carried out in $\mathcal{O}(1 + 2^d + 4^d + \ldots + N) = \mathcal{O}(N)$, resulting in the desired $\mathcal{O}(N)$ algorithm.

We again return to the potential example and show how to define the respective operators.

Multipole-translation operator

The multipole-translation operator is supposed to allow the calculation of moments of a source box at a coarse scale by means of the moments of its children at the finer scale. We start with the multipole expansion of a child box centered at the origin

$$f(t) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} M_n^m \cdot \frac{Y_n^m(t)}{\|t\|^{n+1}}. \tag{3.30}$$

This can now be transferred to the center of the parent box z resulting in the expansion

$$\hat{f}(t) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \hat{M}_n^m \cdot \frac{Y_n^m(t-z)}{\|t-z\|^{n+1}}, \tag{3.31}$$

where the new moments can be expressed as

$$\hat{M}_n^m = \sum_{j=0}^{n} \sum_{k=-j}^{j} M_{n-j}^{m-k} \frac{i^{|m|}}{i^{|k|} i^{|m-k|}} \frac{A_j^k A_{n-j}^{m-k}}{A_n^m} \|z\|^j Y_j^k(z), \tag{3.32}$$

and A_n^m is given by

$$A_n^m = \frac{(-1)^n}{\sqrt{(n-m)! \cdot (n+m)!}}. \tag{3.33}$$

The associated error bound for the approximation with p terms is given by

$$\left| f(t) - \hat{f}_p(t) \right| \leq \frac{W}{\|t-z\| - \|s\| - \|z\|} \left(\frac{\|s\| + \|z\|}{\|t-z\|} \right)^p, \tag{3.34}$$

which is less sharp than the original bound. The workload for calculating a new set of moments adds up to $\mathcal{O}(p^4)$, since each of the p^2 moments is evaluating using a sum with another p^2 terms.

Multipole-to-local operator

We now require an operator for shifting an multipole expansion for an arbitrary target t and an interacting source box to the target box center z. To this end, we assume the radius $\|s\|$ of the source box to be sensibly smaller than $\|z\|$, more precisely we postulate the existence of some constant $c > 1$ with $(c+1) \cdot \|s\| < \|z\|$, and further assume $\|z - t\| < \|s\|$. The expansion at the target box center z then has the form

$$\bar{f}(t) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} L_n^m \cdot \|z-t\|^n Y_n^m(z-t), \tag{3.35}$$

where L_n^m is given by

$$L_n^m = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} \frac{M_j^k}{(-1)^j} \frac{i^{|m-k|}}{i^{|k|} i^{|m|}} \frac{A_j^k A_n^m}{A_{j-n}^{k-m}} \frac{Y_{j+n}^{k-m}(z)}{\|z\|^{j+n+1}}. \tag{3.36}$$

Naturally, we can not apply this result directly seeing that L_n^m consists of an infinite sum. Thus, a new error is introduced by approximating the L_n^m with a finite sum, additionally to the error caused by truncating the sum in (3.35). According to [36], the derivation of a precise error bound is rather laborious. Assuming exact computation of the L_n^m, an error bound for the approximation of (3.35) with p terms can be given by

$$\left| f(t) - \bar{f}_p(t) \right| \leq \frac{W}{(c-1)\,\|s\|} \left(\frac{1}{c} \right)^p. \tag{3.37}$$

The evaluation of the multipole-to-local translation again requires a workload of $\mathcal{O}(p^4)$.

Local-translation operator

The local translation is supposed to shift a multipole expansion from a target box center to any of the given centers of its children. Since this operator is applied just after the multipole-to-local operator, we start with a finite expansion (centered in the target box center that is assumed to be the origin) of the form

$$\bar{f}_p(t) = \sum_{n=0}^{p-1} \sum_{m=-n}^{n} L_n^m \cdot \|t\|^n Y_n^m(t). \tag{3.38}$$

We can rewrite this as an expansion centered at the new point z

$$\check{f}_p(t) = \sum_{n=0}^{p-1} \sum_{m=-n}^{n} \check{L}_n^m \cdot \|t-z\|^n Y_n^m(t-z), \tag{3.39}$$

where

$$\check{L}_n^m = \sum_{j=n}^{p-1} \sum_{k=-j}^{j} \frac{L_j^k}{(-1)^{j+n}} \frac{i^{|k|}}{i^{|k-m|} i^{|m|}} \frac{A_{j-n}^{k-m} A_n^m}{A_j^k} Y_{j-n}^{k-m}(-z)\, \|z\|^{j-n}. \tag{3.40}$$

This translation does not require another approximation and also involves $\mathcal{O}(p^4)$ costs. We finally have collected all necessary techniques for the full multilevel fast multipole algorithm. The rough complexity estimation yields $\mathcal{O}(p^4 N)$, this can however be improved to $\mathcal{O}(p^2 N)$ by choosing the number of points in the boxes of the finest scale directly proportional to p^2.

When considering a fast multipole algorithm customized for a particular problem, the decision between the $\mathcal{O}(N \log N)$ and the $\mathcal{O}(N)$ variant using translation operators must be made with

special care. Since the translation entails additional approximation errors and produces larger constants in the complexity estimation, the simple $\mathcal{O}(N \log N)$ algorithm will usually be faster for relatively small N until a break-even point is reached. Moreover, the constants are in fact exponential with respect to the dimension due to the uniform space partitioning scheme presented above. In [13], Cheng, Greengard and Rokhlin therefore present an improved version for the three-dimensional case using additional rotation operators. We hereby conclude this general survey on the fast multipole method and proceed to our main topic, the analysis of various fast approximation algorithms for the Gauss transform.

4 Approximation Algorithms for the Discrete Gauss Transform

In this chapter we will introduce and compare several different algorithms for the fast approximation of the *Discrete Gauss Transform*. For this purpose we use the following notation, that will stay consistent throughout our discussion.

Definition 4.1 (Discrete Gauss Transform)**.** *Let*

- $t_j \in \mathbb{R}^d$ *for* $j = 1, \ldots, M$ *be the* ***targets****,* M *the number of targets,*
- $s_i \in \mathbb{R}^d$ *for* $i = 1, \ldots, N$ *be the* ***sources****,* N *the number of sources,*
- $w_i \in \mathbb{R}$ *for* $i = 1, \ldots, N$ *be the* ***source weights*** *and*
- $h \in \mathbb{R}^+$ *be the constant* ***bandwidth****.*

We then call

$$\left(G(t_j) = \sum_{i=1}^{N} w_i \cdot e^{-\|t_j - s_i\|^2/h^2} \right)_{j=1,\ldots,M} \tag{4.1}$$

the ***Discrete Gauss Transform*** *due to the sources* $s_1, \ldots, s_N$ *and targets* $t_1, \ldots, t_M$*.*

We will also be speaking of $G(t_j)$ as a "Gaussian" and refer to the *Discrete Gauss Transform* simply as *Gauss Transform*. In the beginning of each new section we give a short summary of the main features of the particular algorithm:

1. **Data structure:** The data structure used to access and manipulate the source and target points.
2. **Expansions:** The series expansion(s) used to combine the interaction of several sources and targets.
3. **Guaranteed global error:** The error bound guaranteed for a single Gaussian at any given target point.
4. **Runtime complexity:** The runtime complexity of the algorithm with respect to the number of sources N, number of targets M, dimension d and some other specific parameters where applicable.

4.1 Fast Gauss Transform

Data structure: Uniform grid (for both sources and targets)

Expansions: *Hermite*, *Taylor* and *Taylor-Hermite Expansion* (l_∞-version)

Guaranteed global error: $\left|\widetilde{G}_{FGT}(t_j) - G(t_j)\right| < \epsilon$ for a fixed bound $\epsilon > 0$

Runtime complexity: $\mathcal{O}((N+M) \cdot p^d \cdot d)$ where p is a truncation parameter for series expansion

The *Fast Gauss Transform (FGT)* was first presented by Leslie Greengard and John Strain [29] in 1991 and belongs to the group of *single-level* fast multipole methods. All sources and targets are assumed to be located in the d-dimensional unit hyper cube $[0,1]^d$. For arbitrary source and target points, all points as well as the bandwidth h are rescaled in order to fulfill the premise. The unit box is then uniformly divided into smaller hyper cubes ("boxes") of side length $l = \sqrt{2}rh$, where r is some positive constant $\leq \frac{1}{2}$ and h is the bandwidth[1].

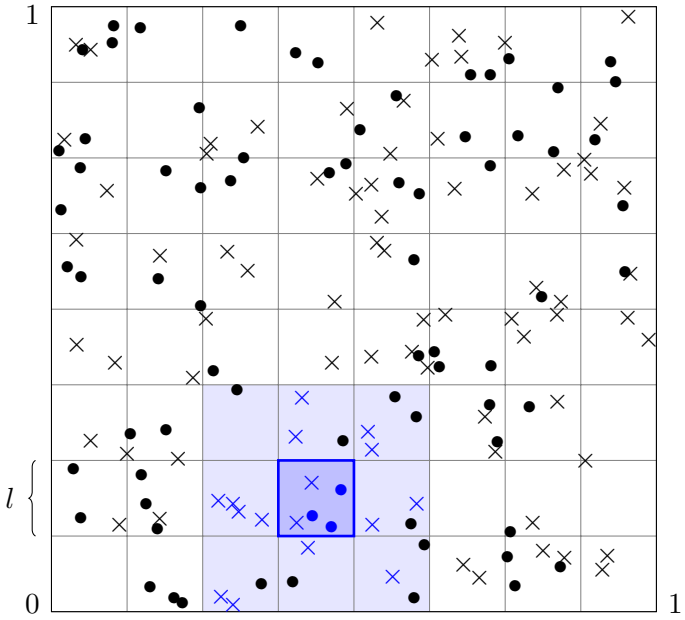


Figure 4.1: Targets *(dots)* in the highlighted box interact with sources *(crosses)* in the same box and neighboring boxes.

A box is called either "source box" or "target box" depending on whether we are interested in the source or target points laying in it. The algorithm considers interactions between source and target boxes rather than individual points. Depending on the number of points laying in

[1] NB: The parameter h corresponds to $\sqrt{\delta}$ in the original paper [29].

a given source and target box, it will choose either one of three different expansions or direct evaluation to calculate the interaction between source and target points.

4.1.1 Hermite, Taylor and Taylor-Hermite Expansions

In order to introduce the necessary expansions, we need to make use of d-dimensional Hermite functions that were presented above. We recall the Hermite functions to be defined by

$$h_\alpha(x) = h_{\alpha_1}(x_1) \cdot \ldots \cdot h_{\alpha_d}(x_d) = (-1)^{|\alpha|} \cdot \nabla_x^\alpha e^{-\|x\|^2},$$

while their generating function is given by

$$e^{-\|x-y\|^2} = \sum_{\alpha \geq 0} \frac{y^\alpha}{\alpha!} h_\alpha(x). \tag{4.2}$$

For a given source box B_{s^*} with center s^* and target box B_{t^*} with center t^* we now consider a reference case for each of the three different expansions.

1. Hermite expansion: Consider the case of a source box B_{s^*} containing many sources and a target box B_{t^*} containing only few targets.

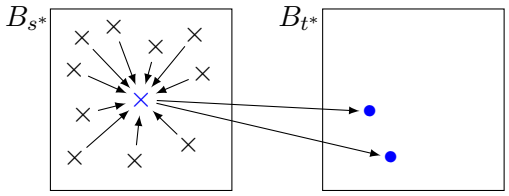


FIGURE 4.2: Hermite expansion. Contribution of many sources is accumulated in box center s^*, which interacts with the targets.

If in (4.2) we let $x = (t_j - s^*)/h$ and $y = (s_i - s^*)/h$, we get the Hermite expansion centered at s^*:

$$e^{-\|t_j - s_i\|^2/h^2} = \sum_{\alpha \geq 0} \frac{1}{\alpha!} \cdot \left(\frac{s_i - s^*}{h} \right)^\alpha \cdot h_\alpha \left(\frac{t_j - s^*}{h} \right). \tag{4.3}$$

2. Taylor expansion: In the second case B_{t^*} contains many targets, while there are only few sources in B_{s^*}.

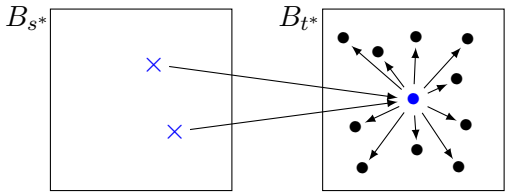


FIGURE 4.3: Taylor expansion. Sources interact with target box center t^* only, the contribution is then spread out to all targets in B_{t^*}.

In (4.2) we let $x = (s_i - t^*)/h$ and $y = (t_j - t^*)/h$ to get the Taylor expansion centered at t^*:

$$e^{-\|t_j - s_i\|^2/h^2} = \sum_{\alpha \geq 0} \frac{1}{\alpha!} \cdot h_\alpha \left(\frac{s_i - t^*}{h} \right) \cdot \left(\frac{t_j - t^*}{h} \right)^\alpha. \tag{4.4}$$

3. Taylor-Hermite expansion: Finally, consider the case where both the source box B_{s^*} and target box B_{t^*} contain many sources and targets, respectively.

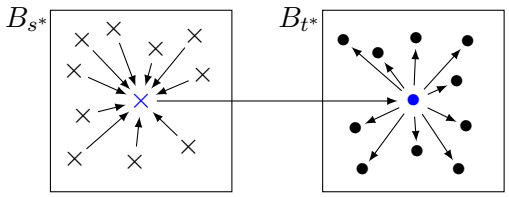


FIGURE 4.4: Taylor-Hermite expansion. Contribution of many sources is accumulated in s^*, which interacts with t^*, the contribution is then spread out to all targets in B_{t^*}.

To combine both the Hermite and Taylor expansion, we note that the multi-dimensional Taylor series (3.2) of the Hermite function $h_\alpha(z)$ can be written as

$$h_\alpha(z) = \sum_{\beta \geq 0} \frac{(-1)^{|\beta|}}{\beta!} \cdot (z - z^*)^\beta \cdot h_{\alpha+\beta}(z^*).$$

Now let $z = (t_j - s^*)/h$ and $z^* = (t^* - s^*)/h$ to replace the last factor in equation (4.3) by the Taylor series. We thus get the Taylor-Hermite expansion centered at s^* and t^*:

$$e^{-\|t_j - s_i\|^2/h^2} = \sum_{\alpha \geq 0} \frac{1}{\alpha!} \cdot \left(\frac{s_i - s^*}{h} \right)^\alpha \cdot \sum_{\beta \geq 0} \frac{(-1)^{|\beta|}}{\beta!} \cdot \left(\frac{t_j - t^*}{h} \right)^\beta \cdot h_{\alpha+\beta} \left(\frac{t^* - s^*}{h} \right). \tag{4.5}$$

The *Fast Gauss Transform* chooses one of the three expansions (4.3), (4.4) and (4.5) depending on the number of sources in B_{s^*} and the number of targets in B_{t^*}. Direct evaluation is used if both numbers are small.

To actually apply each of the given expansions in a calculation with multiple steps as shown in the figures above, note that the Gauss Transform at a target t_j of target box B_{t^*} due to all sources $s_i \in B_{s^*}$ is given by

$$G_{B_{s^*}}(t_j) = \sum_{s_i \in B_{s^*}} w_i \cdot e^{-\|t_j - s_i\|^2/h^2}.$$

Using equations (4.3), (4.4) and (4.5) and interchanging the summations yields:

$$G_{B_{s^*}}(t_j) = \sum_{\alpha \geq 0} \underbrace{\frac{1}{\alpha!} \cdot \sum_{s_i \in B_{s^*}} w_i \cdot \left(\frac{s_i - s^*}{h} \right)^\alpha}_{=: C_\alpha^H(B_{s^*})} \cdot h_\alpha \left(\frac{t_j - s^*}{h} \right); \tag{4.6}$$

$$G_{B_{s^*}}(t_j) = \sum_{\alpha \geq 0} \underbrace{\frac{1}{\alpha!} \cdot \sum_{s_i \in B_{s^*}} w_i \cdot h_\alpha \left(\frac{s_i - t^*}{h} \right)}_{=:C_\alpha^T(B_{s^*}, t^*)} \cdot \left(\frac{t_j - t^*}{h} \right)^\alpha; \tag{4.7}$$

$$G_{B_{s^*}}(t_j) = \sum_{\beta \geq 0} \underbrace{\frac{(-1)^{|\beta|}}{\beta!} \cdot \sum_{\alpha \geq 0} C_\alpha^H(B_{s^*}) \cdot h_{\alpha+\beta} \left(\frac{t^* - s^*}{h} \right)}_{=:C_\beta^{TH_\infty}(B_{s^*}, t^*)} \cdot \left(\frac{t_j - t^*}{h} \right)^\beta. \tag{4.8}$$

Now the evaluation can be done in the following way:

- For the *Hermite Expansion* first calculate the Hermite coefficients $C_\alpha^H(B_{s^*})$ depending only on the sources in B_{s^*}; then using the coefficients, calculate the Gauss Transform for every target t_j by means of (4.6).

- For the *Taylor Expansion* first calculate the Taylor coefficients $C_\alpha^T(B_{s^*}, t^*)$ depending only on the sources in B_{s^*} and the target box center t^*; then using the coefficients, calculate the Gauss Transform for every target t_j by means of (4.7).

- For the *Taylor-Hermite Expansion* first calculate the Hermite coefficients $C_\alpha^H(B_{s^*})$ depending only on the sources in B_{s^*}; in a second step, calculate the Taylor-Hermite coefficients $C_\beta^{TH_\infty}(B_{s^*}, t^*)$ depending only on the Hermite coefficients and the two box centers s^* and t^*; finally, using the Taylor-Hermite coefficients, calculate the Gauss Transform for every target t_j by means of (4.8).

Of course, the number of coefficients used in the calculation has to be restricted to some finite number. The *Fast Gauss Transform* applies a simple rule by choosing a fixed number p depending on the bandwidth h and error bound ϵ and taking into account all terms with multiindices $\|\alpha\|_\infty < p$ resulting in a total number of p^d terms for each expansion. We will now define the main expansions of the *FGT*.

Definition 4.2 (FGT Expansions)**.** *Let the same assumptions apply as in Definition 4.1. Further let B_{s^*} be a source box and B_{t^*} a target box with center points s^* and t^*, respectively. Let $t_j \in B_{t^*}$ and $p \geq 1$ be an arbitrary, but fixed integer. We then define the*

- ***Hermite Coefficient*** *(due to source box B_{s^*})*

$$C_\alpha^H(B_{s^*}) = \frac{1}{\alpha!} \cdot \sum_{s_i \in B_{s^*}} w_i \cdot \left(\frac{s_i - s^*}{h} \right)^\alpha; \tag{4.9}$$

- ***Taylor Coefficient*** *(due to center t^* and source box B_{s^*})*

$$C_\alpha^T(B_{s^*}, t^*) = \frac{1}{\alpha!} \cdot \sum_{s_i \in B_{s^*}} w_i \cdot h_\alpha \left(\frac{s_i - t^*}{h} \right); \tag{4.10}$$

- ***Taylor-Hermite Coefficient*** *(due to center t^* and source box B_{s^*})*

$$C_\beta^{TH_p^\infty}(B_{s^*}, t^*) = \frac{(-1)^{|\beta|}}{\beta!} \cdot \sum_{\|\alpha\|_\infty < p} C_\alpha^H(B_{s^*}) \cdot h_{\alpha+\beta}\left(\frac{t^* - s^*}{h}\right). \tag{4.11}$$

Further, we define the

- ***Hermite Expansion (l_∞-version)*** *(due to target t_j and source box B_{s^*})*

$$\widetilde{G}_{B_{s^*}}^{H_p^\infty}(t_j) = \sum_{\|\alpha\|_\infty < p} C_\alpha^H(B_{s^*}) \cdot\ h_\alpha\left(\frac{t_j - s^*}{h}\right); \tag{4.12}$$

- ***Taylor Expansion (l_∞-version)*** *(due to target t_j and source box B_{s^*})*

$$\widetilde{G}_{B_{s^*}}^{T_p^\infty}(t_j) = \sum_{\|\alpha\|_\infty < p} C_\alpha^T(B_{s^*}, t^*) \cdot \left(\frac{t_j - t^*}{h}\right)^\alpha; \tag{4.13}$$

- ***Taylor-Hermite Expansion (l_∞-version)*** *(due to target t_j and source box B_{s^*})*

$$\widetilde{G}_{B_{s^*}}^{TH_p^\infty}(t_j) = \sum_{\|\beta\|_\infty < p} C_\beta^{TH_p^\infty}(B_{s^*}, t^*) \cdot \left(\frac{t_j - t^*}{h}\right)^\beta. \tag{4.14}$$

The truncation of the infinite series introduces an approximation error. The error bounds not only depend on p, but also on the side length l of the boxes. A smaller side length will yield a tighter error bound, since the Taylor series is a local expansion. We present the error bounds[2] for all three expansions, the proof follows [48].

Lemma 4.3 (Error bound for *Hermite Expansion*)**.** *Let the unit box be uniformly divided into boxes with side length $l = \sqrt{2}rh$, where $0 < r \leq \frac{1}{2}$. Further let t_j be a target point in the target box B_{t^*} and consider all sources s_i in the source box B_{s^*} with center s^*. Then the absolute error introduced by approximating the discrete Gauss Transform $G_{B_{s^*}}(t_j) = \sum_{s_i \in B_{s^*}} w_i \cdot e^{-\|t_j - s_i\|^2/h^2}$ with the Hermite Expansion $\widetilde{G}_{B_{s^*}}^{H_p^\infty}(t_j)$ can be bounded by*

$$\frac{W_{s^*}}{(1-r)^d} \sum_{k=0}^{d-1} \binom{d}{k} (1 - r^p)^k \left(\frac{r^p}{\sqrt{p!}}\right)^{d-k}, \tag{4.15}$$

where $W_{s^*} = \sum_{s_i \in B_{s^*}} |w_i|$.

Proof. Using equation (4.6) and (4.12), the error term $E^{H_p}(t_j) = \left|G_{B_{s^*}}(t_j) - \widetilde{G}_{B_{s^*}}^{H_p^\infty}(t_j)\right|$ can

[2] The bound given in the original paper [29] is incorrect, a corrected version has been presented in [5].

be rewritten as

$$E^{H_p}(t_j) = \left| \sum_{s_i \in B_{s^*}} w_i \cdot \sum_{\|\alpha\|_\infty \geq p} \frac{1}{\alpha!} \cdot \left(\frac{s_i - s^*}{h} \right)^\alpha \cdot h_\alpha \left(\frac{t_j - s^*}{h} \right) \right|.$$

The inequality by Szász (3.11) providing an upper bound for the Hermite function allows the following estimation:

$$\begin{aligned} E^{H_p}(t_j) &\leq \sum_{s_i \in B_{s^*}} |w_i| \cdot \sum_{\|\alpha\|_\infty \geq p} \frac{1}{\alpha!} \cdot \left| \left(\frac{s_i - s^*}{h} \right)^\alpha \right| \cdot \left| h_\alpha \left(\frac{t_j - s^*}{h} \right) \right| \\ &\leq W_{s^*} \cdot \sum_{\|\alpha\|_\infty \geq p} \frac{1}{\alpha!} \cdot \left(\frac{r}{\sqrt{2}} \right)^{|\alpha|} \cdot \sqrt{2^{|\alpha|}\alpha!} \cdot e^{-\|t_j - s^*\|^2/2h^2} \\ &\leq W_{s^*} \cdot \sum_{\|\alpha\|_\infty \geq p} \frac{r^{|\alpha|}}{\sqrt{\alpha!}}. \end{aligned}$$

The sum can be rewritten as

$$\begin{aligned} \sum_{\|\alpha\|_\infty \geq p} \frac{r^{|\alpha|}}{\sqrt{\alpha!}} &= \sum_{\|\alpha\|_\infty \geq 0} \frac{r^{|\alpha|}}{\sqrt{\alpha!}} - \sum_{\|\alpha\|_\infty < p} \frac{r^{|\alpha|}}{\sqrt{\alpha!}} \\ &= \sum_{\|\alpha\|_\infty \geq 0} \prod_{n=1}^{d} \frac{r^{\alpha_n}}{\sqrt{\alpha_n!}} - \sum_{\|\alpha\|_\infty < p} \prod_{n=1}^{d} \frac{r^{\alpha_n}}{\sqrt{\alpha_n!}} \\ &= \prod_{n=1}^{d} \left(\sum_{\alpha_n \geq 0} \frac{r^{\alpha_n}}{\sqrt{\alpha_n!}} \right) - \prod_{n=1}^{d} \left(\sum_{\alpha_n < p} \frac{r^{\alpha_n}}{\sqrt{\alpha_n!}} \right) \\ &= \left(\sum_{a<p} \frac{r^a}{\sqrt{a!}} + \sum_{a \geq p} \frac{r^a}{\sqrt{a!}} \right)^d - \left(\sum_{a<p} \frac{r^a}{\sqrt{a!}} \right)^d. \end{aligned}$$

Applying the binomial theorem to the first term yields

$$\begin{aligned} \sum_{\|\alpha\|_\infty \geq p} \frac{r^{|\alpha|}}{\sqrt{\alpha!}} &= \sum_{k=0}^{d-1} \binom{d}{k} \left(\sum_{a<p} \frac{r^a}{\sqrt{a!}} \right)^k \left(\sum_{a \geq p} \frac{r^a}{\sqrt{a!}} \right)^{d-k} \\ &\leq \sum_{k=0}^{d-1} \binom{d}{k} \left(\sum_{a<p} r^a \right)^k \left(\frac{r^p}{\sqrt{p!}} \sum_{a \geq 0} r^a \right)^{d-k} \\ &= \sum_{k=0}^{d-1} \binom{d}{k} \left(\frac{1 - r^p}{1 - r} \right)^k \left(\frac{r^p}{\sqrt{p!}} \cdot \frac{1}{1 - r} \right)^{d-k} \end{aligned}$$

$$= \frac{1}{(1-r)^d} \cdot \sum_{k=0}^{d-1} \binom{d}{k} (1-r^p)^k \left(\frac{r^p}{\sqrt{p!}} \right)^{d-k}.$$

□

Lemma 4.4 (Error bound for *Taylor Expansion*)**.** *Let the same assumptions apply as in Lemma 4.3. Then the absolute error introduced by approximating the discrete Gauss Transform* $G_{B_{s^*}}(t_j) = \sum_{s_i \in B_{s^*}} w_i \cdot e^{-\|t_j - s_i\|^2/h^2}$ *with the Taylor Expansion* $\widetilde{G}^{T_p^\infty}_{B_{s^*}}(t_j)$ *can be bounded by*

$$\frac{W_{s^*}}{(1-r)^d} \sum_{k=0}^{d-1} \binom{d}{k} (1-r^p)^k \left(\frac{r^p}{\sqrt{p!}} \right)^{d-k}, \tag{4.16}$$

where $W_{s^*} = \sum_{s_i \in B_{s^*}} |w_i|$.

Proof. Analog to (4.3), see [48].

Lemma 4.5 (Error bound for *Taylor-Hermite Expansion*)**.** *Let the same assumptions apply as in Lemma 4.3. Then the error introduced by approximating the discrete Gauss Transform* $G_{B_{s^*}}(t_j) = \sum_{s_i \in B_{s^*}} w_i \cdot e^{-\|t_j - s_i\|^2/h^2}$ *with the Taylor-Hermite Expansion* $\widetilde{G}^{TH_p^\infty}_{B_{s^*}}(t_j)$ *can be bounded by*

$$\begin{aligned} &\frac{W_{s^*}}{(1-r)^d} \sum_{k=0}^{d-1} \binom{d}{k} (1-r^p)^k \left(\frac{r^p}{\sqrt{p!}} \right)^{d-k} + \\ &\frac{W_{s^*}}{(1-\sqrt{2}r)^{2d}} \left[\sum_{k=0}^{d-1} \binom{d}{k} (1-(\sqrt{2}r)^p)^k \left(\frac{(\sqrt{2}r)^p}{\sqrt{p!}} \right)^{d-k} \right]^2, \end{aligned} \tag{4.17}$$

where $W_{s^*} = \sum_{s_i \in B_{s^*}} |w_i|$.

Proof. See [48].

4.1.2 Implementation details of the FGT

Next, we explain the details of the implementation before actually giving a formal description of the algorithm in pseudo-code. All sources and targets are supposed to be situated in the unit box $[0,1]^d$. The unit box is subdivided uniformly and parallel to the axes into smaller boxes with side length $l = \sqrt{2}rh$, where $r \leq \frac{1}{2}$ is chosen maximal such that $\frac{1}{l}$ is an integer. Each source and target is then assigned to the particular box in which lies. Now for a given target box B_{t^*}, we need to calculate the Gaussians for each target in B_{t^*} due to all source boxes. However since the Gaussian kernel decays rapidly, not all but only the nearest source boxes are included in the calculation, introducing another approximation error. The following Lemma provides a rigorous error bound for including the nearest $(2k+1)^d$ boxes.

Lemma 4.6 (Error bound for restriction to nearest source boxes)**.** *Let the unit box be uniformly divided into boxes with side length* $l = \sqrt{2}rh$*, where* $0 < r \leq \frac{1}{2}$*. Further let* t_j *be a target point in the target box* B_{t^*} *and, for a fixed number* k*, let* A *be the union of the* $(2k+1)^d$ *boxes*

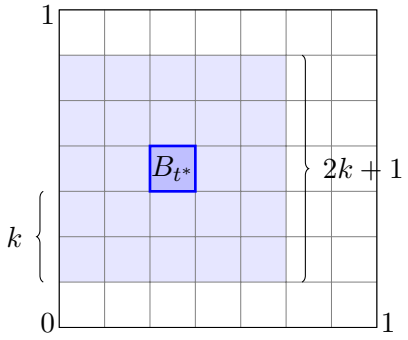


FIGURE 4.5: For a target box B_{t^*} only the contribution of the $(2k+1)^d$ nearest source boxes is calculated.

neighboring B_{t^*} (see Fig. 4.5). Then the error introduced by approximating the discrete Gauss Transform $G(t_j) = \sum_{i=1}^{N} w_i \cdot e^{-\|t_j - s_i\|^2/h^2}$ due to all sources with the discrete Gauss Transform $G_A(t_j) = \sum_{s_i \in A} w_i \cdot e^{-\|t_j - s_i\|^2/h^2}$ due to all sources in A can be bounded by $We^{-2r^2k^2}$, where $W = \sum_{i=1}^{N} |w_i|$.

Proof. Note that any source $s_i \notin A$ has a minimum distance of $k \cdot l$ from any target in B_{t^*}. Therefore, we can estimate the error term as

$$E_A(t_j) = \left| \sum_{s_i \notin A} w_i \cdot e^{-\|t_j - s_i\|^2/h^2} \right| \leq \sum_{s_i \notin A} |w_i| \cdot e^{-k^2 l^2/h^2} \leq W \cdot e^{-2r^2k^2}.$$

□

The parameter k is determined using the condition $We^{-2r^2k^2} \leq \epsilon$, where ϵ is the user-defined absolute error bound. In the same manner, the bound p for the expansions is determined by using the error estimations of Lemma 4.3 and 4.5. Furthermore, the algorithm uses two cutoff parameters C_s and C_t in order to decide for each pair of source and target box which of the four methods (*Hermite*, *Taylor*, *Taylor-Hermite Expansion* or direct evaluation) is used. Let N_s be the number of sources in the source box and N_t be the number of targets in the target box. Then if $N_t < C_t$ the algorithm uses direct evaluation in case $N_s < C_s$ and *Hermite Expansion* in case $N_s \geq C_s$; if on the other hand $N_t \geq C_t$ the algorithm uses *Taylor Expansion* in case $N_s < C_s$ and *Taylor-Hermite Expansion* in case $N_s \geq C_s$.

Notice that there is a subtle difference in the way the *Hermite Expansions* (4.12) are calculated from the way both the *Taylor Expansions* (4.13) and the *Taylor-Hermite Expansions* (4.14) are calculated. The Hermite Coefficients $C_\alpha^H(B_{s^*})$ only depend on the current source box B_{s^*} and can therefore be reused for different target boxes or even pre-calculated, if an application must be run several times for different targets, but the same source points and weights. The *Hermite Expansion* can then be calculated instantly when processing a certain target and source box. On the other hand, the *Taylor* and *Taylor-Hermite Coefficients* $C_\alpha^T(B_{s^*}, t^*)$ and $C_\beta^{TH_p^\infty}(B_{s^*}, t^*)$, respectively, depend on both the current source box B_{s^*} and target box center t^* and are therefore calculated on the fly only for interacting boxes. In order to save workload

due to multiplications, all *Taylor* and *Taylor-Hermite Coefficients* for a fixed target box are added up, while in a post-processing step, the Taylor series are evaluated looping through all relevant target boxes. We now present the technical description of the algorithm.

Algorithm: *Fast Gauss Transform* [29]
Input: N sources s_i, $(i = 1, \ldots, N)$ in $[0,1]^d$
M targets t_j, $(j = 1, \ldots, M)$ in $[0,1]^d$
N weights w_i, $(i = 1, \ldots, N)$ in $\mathbb{R}$
bandwidth $h > 0$, absolute error bound $\epsilon > 0$
Output: $\widetilde{G}_{FGT}(t_j)$, $(j = 1, \ldots, M)$, with $\left|\widetilde{G}_{FGT}(t_j) - G(t_j)\right| < \epsilon$

```
Choose the largest r ≤ 1/2 such that 1/√2 hr is an integer n_side.
Subdivide the unit box into n_side^d boxes.
Determine the number of neighbor boxes k ≥ 0 to go out in each
  direction based on r and ε using Lemma 4.6.
Choose the series cutoff p based on r and ε using Lemma 4.3 and 4.5.
Choose the cutoff parameters C_s and C_t (details see below).

FOR i = 1..n_side^d
  N_s = number of sources in ith box B_{s*}.
  Build the interaction list of not more than (2k+1)^d neighboring
  target boxes within range of B_{s*}.
  IF N_s < C_s
    FOR j = 1..(2k+1)^d
      N_t = number of targets in jth box B_{t*} in interaction list.
      IF N_t < C_t
        Use direct evaluation of Gaussians due to B_{s*} and B_{t*}.
      ELSE
        Use Taylor Expansion (4.13) for each of the N_s sources s_i
        and the center t* of box B_{t*} to compute
        Taylor Coefficients C_α^T(B_{s*}, t*).
  ELSE (N_s ≥ C_s)
    FOR j = 1..(2k+1)^d
      N_t = number of targets in jth box B_{t*} in interaction list.
      IF N_t < C_t
        Compute Hermite Expansion (4.12) for the center s* of box B_{s*}
        and for each target t_j of box B_{t*}.
      ELSE
        Use Taylor-Hermite Expansion (4.14) for the center s* of box B_{s*}
        and the center t* of box B_{t*} to compute
        Taylor-Hermite Coefficients C_β^{TH_p^∞}(B_{s*}, t*).

FOR j = 1..n_side^d
  N_t = number of targets in jth box B_{t*}.
  IF N_t < C_t
    Evaluate the Taylor series for box center t*
    at each target t_j of box B_{t*}.
```

In the original paper [29] that introduces the *Fast Gauss Transform* there are no binding instructions for the exact choice of the cutoff parameters C_s and C_t. The authors only suggest to choose $C_s = \mathcal{O}(p^d)$ and $C_t = \mathcal{O}(p^d)$ in order to get a better workload balance.

4.1.3 Run time analysis

We now give a run time estimation of the *FGT* taking into account the dependence on the number of sources N, number of targets M, the dimension d and p. Calculating interactions between target box B_t with N_t targets and source box B_s with N_s sources using direct evaluation can easily be seen to have a runtime of $\mathcal{O}(N_s \cdot N_t \cdot d)$, while the three different expansions all result in a runtime of $\mathcal{O}(N_s \cdot p^d \cdot d + N_t \cdot p^d \cdot d)$. Since direct evaluation is used only in case of $N_s < C_s = \mathcal{O}(p^d)$ and $N_t < C_t = \mathcal{O}(p^d)$ and the sum of the numbers N_s of all source boxes adds up to the total number of sources N (same for the targets), we get a total runtime of

Runtime Complexity 4.7. *The* FGT *has a runtime of* $\mathcal{O}((N + M) \cdot p^d \cdot d)$.

Here p as well as the constant factor primarily depend on the error bound ϵ.

The *Fast Gauss Transform* – though performing well for many problems in 1, 2 and 3 dimensions – has a few conceptual drawbacks. The main issues are:

- The static box subdivision scheme ignores the inherent structure of the data and only works well for uniformly distributed points.
- The total number of boxes n_{side}^d as well as the number of series terms p^d grow exponentially with the dimension d.
- Important control parameters (k, p, C_s, C_t) are chosen globally instead of locally.
- The *FGT* only guarantees an absolute error bound.

All of these issues are addressed in different ways by the *Improved Fast Gauss Transform*, the algorithm presented in the following section.

4.2 Improved Fast Gauss Transform

Data structure: Farthest-point clustering (only for sources)

Expansions: *IFGT-Taylor Expansion* (l_1-version)

Guaranteed global error: $\left|\widetilde{G}_{IFGT}(t_j) - G(t_j)\right| < \epsilon \cdot \sum_{i=1}^{N} |w_i|$ for a fixed bound $\epsilon > 0$

Runtime complexity: $\mathcal{O}\left((N + M) \cdot \binom{p_{max}-1+d}{d}\right) + \mathcal{O}\left(N \cdot K \cdot d\right)$ where p_{max} is a truncation parameter for series expansion and K is the number of clusters.

The *Improved Fast Gauss Transform (IFGT)* was first published by Changjiang Yang, Ramani Duraiswami and Nail A. Gumerov [67] in 2003. The naming of the algorithm might

be a bit misleading, since it is not quite an improvement of the *Fast Gauss Transform*, but rather a new approximation approach for the discrete Gauss Transform avoiding most issues the *FGT* suffers from (see 4.1.3). The *IFGT* uses a different expansion, different space subdivision scheme as well as local error control mechanisms. We consider the revised version of [49] (2005).

4.2.1 The IFGT Expansion

Instead of choosing between three slightly different expansions, the *IFGT* only uses one expansion based on a Taylor series. First, notice that for any $s^* \in \mathbb{R}^d$ the Gauss Transform can be rewritten as

$$\begin{aligned} G(t_j) &= \sum_{i=1}^{N} w_i \cdot e^{-\|t_j - s_i\|^2/h^2} \\ &= \sum_{i=1}^{N} w_i \cdot e^{-\|(t_j - s^*) - (s_i - s^*)\|^2/h^2} \\ &= \sum_{i=1}^{N} w_i \cdot e^{-\|t_j - s^*\|^2/h^2} \cdot e^{-\|s_i - s^*\|^2/h^2} \cdot e^{2(t_j - s^*)\cdot(s_i - s^*)/h^2}. \end{aligned} \tag{4.18}$$

For an arbitrary but fixed s^* the evaluation of the first factor $e^{-\|t_j - s^*\|^2/h^2}$ for all targets t_j $(j = 1, \ldots, M)$ takes $\mathcal{O}(M \cdot d)$ time, likewise the evaluation of the second factor $e^{-\|s_i - s^*\|^2/h^2}$ for all sources s_i $(i = 1, \ldots, N)$ takes $\mathcal{O}(N \cdot d)$ time. In the third factor the entanglement of targets and sources will be broken via the following Taylor expansion:

Lemma 4.8. *Let $t, s^* \in \mathbb{R}^d$ and $h > 0$. Then the Taylor expansion of the function $f(s) = e^{2(t-s^*)\cdot(s-s^*)/h^2}$ can be written as*

$$f(s) = e^{2(t-s^*)\cdot(s-s^*)/h^2} = \sum_{|\alpha|<p} \frac{2^{|\alpha|}}{\alpha!} \left(\frac{t - s^*}{h}\right)^{\alpha} \left(\frac{s - s^*}{h}\right)^{\alpha} + R_p(s),$$

where the remainder $R_p(s)$ is bounded by

$$R_p(s) \leq \frac{2^p}{p!} \left(\frac{\|t - s^*\|}{h}\right)^p \left(\frac{\|s - s^*\|}{h}\right)^p e^{2\|t-s^*\|\|s-s^*\|/h^2}.$$

Proof. We want to apply Theorem 3.4 to the function $f(s)$. To this end, note that

$$((s - s^*) \cdot \nabla_y) f(y) = 2 \cdot \left(\frac{t - s^*}{h}\right) \cdot \left(\frac{s - s^*}{h}\right) \cdot f(y).$$

A simple induction yields

$$((s - s^*) \cdot \nabla_y)^n f(y) = 2^n \cdot \left[\left(\frac{t - s^*}{h}\right) \cdot \left(\frac{s - s^*}{h}\right)\right]^n \cdot f(y),$$

and plugging this into Theorem 3.4 leads to

$$\begin{aligned}
f(s) &= \sum_{n=0}^{p-1} \left[\frac{2^n}{n!} \left[\left(\frac{t-s^*}{h} \right) \cdot \left(\frac{s-s^*}{h} \right) \right]^n \cdot f(y) \right]_{y=s^*} \\
&\quad + \left[\frac{2^p}{p!} \left[\left(\frac{t-s^*}{h} \right) \cdot \left(\frac{s-s^*}{h} \right) \right]^p \cdot f(y) \right]_{y=s^*+\theta(s-s^*)} \\
&= \sum_{n=0}^{p-1} \frac{2^n}{n!} \left[\left(\frac{t-s^*}{h} \right) \cdot \left(\frac{s-s^*}{h} \right) \right]^n + \underbrace{\frac{2^p}{p!} \left[\left(\frac{t-s^*}{h} \right) \cdot \left(\frac{s-s^*}{h} \right) \right]^p \cdot f\left(s^* + \theta\left(s-s^*\right)\right)}_{=:R_p(s)} \\
&= \sum_{n=0}^{p-1} \sum_{|\alpha|=n} \frac{2^{|\alpha|}}{\alpha!} \left(\frac{t-s^*}{h} \right)^\alpha \left(\frac{s-s^*}{h} \right)^\alpha + R_p(s) \qquad \text{(by (3.1))} \\
&= \sum_{|\alpha|<p} \frac{2^{|\alpha|}}{\alpha!} \left(\frac{t-s^*}{h} \right)^\alpha \left(\frac{s-s^*}{h} \right)^\alpha + R_p(s).
\end{aligned}$$

Finally the Cauchy-Schwarz inequality yields the abovementioned bound for $R_p(s)$:

$$\begin{aligned}
R_p(s) &= \frac{2^p}{p!} \cdot \left[\left(\frac{t-s^*}{h} \right) \cdot \left(\frac{s-s^*}{h} \right) \right]^p \cdot e^{2(t-s^*)\cdot\theta(s-s^*)/h^2} \\
&\leq \frac{2^p}{p!} \left(\frac{\|t-s^*\|}{h} \right)^p \left(\frac{\|s-s^*\|}{h} \right)^p e^{2\|t-s^*\|\|s-s^*\|/h^2},
\end{aligned}$$

where θ can be omitted since $0 < \theta < 1$.

□

Now let Z_n be a cluster of sources with some cluster center s_n^*. Since in Lemma 4.8 an infinite Taylor series expansion can also be applied, the discrete Gauss Transform at any target point t_j due to all sources $s_i \in Z_n$ can be rewritten using equation (4.18) as

$$\begin{aligned}
G_{Z_n}(t_j) &= \sum_{s_i \in Z_n} w_i \cdot e^{-\|t_j - s_n^*\|^2/h^2} \cdot e^{-\|s_i - s_n^*\|^2/h^2} \cdot \sum_{\alpha \geq 0} \frac{2^{|\alpha|}}{\alpha!} \left(\frac{t_j - s_n^*}{h} \right)^\alpha \left(\frac{s_i - s_n^*}{h} \right)^\alpha \\
&= \sum_{\alpha \geq 0} \underbrace{\frac{2^{|\alpha|}}{\alpha!} \sum_{s_i \in Z_n} w_i \cdot e^{-\|s_i - s_n^*\|^2/h^2} \left(\frac{s_i - s_n^*}{h} \right)^\alpha}_{=:C_\alpha(Z_n)} \cdot e^{-\|t_j - s_n^*\|^2/h^2} \left(\frac{t_j - s_n^*}{h} \right)^\alpha. \qquad (4.19)
\end{aligned}$$

Again we have to restrict the infinite series to a finite number of terms as already shown in Lemma 4.8. We thus define the expansion of the *Improved Fast Gauss Transform*.

Definition 4.9 (IFGT Expansion)**.** *Let the same assumptions apply as in Definition 4.1. Further let Z_n be a source cluster with center s_n^* and t_j be any target point. Let $p \geq 1$ be an arbitrary, but fixed integer. We then define the*

- ***IFGT-Taylor Coefficient*** *(due to source cluster Z_n)*

$$C_\alpha(Z_n) = \frac{2^{|\alpha|}}{\alpha!} \sum_{s_i \in Z_n} w_i \cdot e^{-\|s_i - s_n^*\|^2/h^2} \left(\frac{s_i - s_n^*}{h}\right)^\alpha; \tag{4.20}$$

- ***IFGT-Taylor Expansion (l_1-version)*** *(due to target t_j and source cluster Z_n)*

$$\widetilde{G}_{Z_n}^{IT_p}(t_j) = \sum_{|\alpha|<p} C_\alpha(Z_n) \cdot e^{-\|t_j - s_n^*\|^2/h^2} \left(\frac{t_j - s_n^*}{h}\right)^\alpha. \tag{4.21}$$

Note that while all three expansion of the *FGT* use the l_∞-norm to cut the series ($\|\alpha\|_\infty < p$), the *IFGT* uses the l_1-norm instead ($|\alpha| < p$). As a result, the series expansion only consists of $\binom{p-1+d}{d}$ terms compared to the p^d terms of the l_∞-expansion. We provide a short proof for the claimed number of terms before presenting the local error bound for the *IFGT Expansion*.

Lemma 4.10. *The number $N(d,n)$ of monomials in d variables having a total degree less than or equal to n is $N(d,n) = \binom{n+d}{d}$.*

Proof. We have to count all monomials $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}$ with $\sum_{i=1}^d \alpha_i \leq n$. Consider an arbitrary arrangement of $n+d$ characters, out of which n are 1's and d are separators. There are obviously $\frac{(n+d)!}{n!\cdot d!} = \binom{n+d}{d}$ such arrangements. Further, the following one-to-one mapping from all different arrangements to all considered monomials can be established: the sum of all 1's on the left of the 1st (left-most) separator corresponds to α_1, the sum of all 1's between the kth and $(k+1)$th separator ($1 \leq k < d$) corresponds to α_{k+1}, while all 1's on the right of the dth separator add up to $n - \sum_{i=1}^d \alpha_i$. For the sake of clarity consider the following example ($d = 6$, $n = 8$):

$$|\ \overbrace{1\ \ 1\ \ 1}^{\alpha_2=3}\ |\ |\ |\overbrace{1}^{\alpha_5=1}|\ \overbrace{1\ \ 1}^{\alpha_6=2}\ |\ 1\ \ 1 \qquad (\alpha_1 = \alpha_3 = \alpha_4 = 0)$$

The bijectivity of this mapping proves the Lemma.

□

Lemma 4.11 (Error bound for *IFGT-Taylor Expansion*)**.** *Let Z_n be a source cluster with center s_n^* such that the maximum distance between s_n^* and any $s_i \in Z_n$ is bounded by r. Let further t_j be a target point such that $\|t_j - s_n^*\| \leq r^{int}$. Then the absolute error introduced by approximating the discrete Gauss Transform $G_{Z_n}(t_j) = \sum_{s_i \in Z_n} w_i \cdot e^{-\|t_j - s_i\|^2/h^2}$ with the IFGT-Taylor Expansion $\widetilde{G}_{Z_n}^{IT_p}(t_j)$ can be bounded by $W_{Z_n} \frac{1}{p!} \left(\frac{2r^{int}r}{h^2}\right)^p$, where $W_{Z_n} = \sum_{s_i \in Z_n} |w_i|$.*

Proof. First, by equation (4.18) and Lemma 4.8, the Gauss Transform $G_{Z_n}(t_j)$ can be written as

$$G_{Z_n}(t_j) = \sum_{s_i \in Z_n} w_i \cdot e^{-\|t_j - s_n^*\|^2/h^2} \cdot e^{-\|s_i - s_n^*\|^2/h^2} \cdot \left(\sum_{|\alpha|<p} \frac{2^{|\alpha|}}{\alpha!} \left(\frac{t_j - s_n^*}{h}\right)^\alpha \left(\frac{s_i - s_n^*}{h}\right)^\alpha + R_p(s) \right).$$

Using the bound for $R_p(s)$, we can estimate the error term $E^{IT_p}(t_j) = \left|G_{Z_n}(t_j) - \widetilde{G}_{Z_n}^{IT_p}(t_j)\right|$ as

$$\begin{aligned}
E^{IT_p}(t_j) &= \left| \sum_{s_i \in Z_n} w_i \cdot e^{-\|t_j - s_n^*\|^2/h^2} \cdot e^{-\|s_i - s_n^*\|^2/h^2} \cdot R_p(s) \right| \\
&\leq \sum_{s_i \in Z_n} |w_i| \cdot e^{-\|t_j - s_n^*\|^2/h^2} \cdot e^{-\|s_i - s_n^*\|^2/h^2} \\
&\qquad \cdot \frac{2^p}{p!} \left(\frac{\|t_j - s_n^*\|}{h} \right)^p \left(\frac{\|s_i - s_n^*\|}{h} \right)^p e^{2\|t_j - s_n^*\|\|s_i - s_n^*\|/h^2} \\
&\leq W_{Z_n} \cdot e^{-\|t_j - s_i\|^2/h^2} \cdot \frac{2^p}{p!} \left(\frac{r^{int}}{h} \right)^p \left(\frac{r}{h} \right)^p \\
&\leq W_{Z_n} \cdot \frac{1}{p!} \left(\frac{2r^{int} r}{h^2} \right)^p .
\end{aligned}$$

□

4.2.2 Space subdivision – farthest-point clustering

Unlike the *FGT*, the *IFGT* only combines the sources in clusters, while treating the targets individually. A data adaptive space partitioning scheme called *farthest-point clustering algorithm* is employed. Since the Taylor expansion is only effective in a small radius around the expansion center, the goal is to find a certain number K of clusters, such that their maximum radius is as small as possible. This problem is also known as the *K-center problem*:

Problem 4.12. ***(K-center problem)*** *Given N points $s_i \in \mathbb{R}^d$, $i = 1, \ldots, N$ and an integer $K \in \mathbb{N}$, assign the points to K different clusters Z_n with cluster centers s_n^*, $n = 1, \ldots, K$, such that the maximum radius of the clusters $\max_n \max_{s_i \in Z_n} \|s_i - s_n^*\|$ is minimized.*

An exact solution of this problem has been proven to be *NP*-hard [8]. However Gonzalez [25] introduced an approximation algorithm called *farthest-point clustering*, that is fast, easy to implement and gives an approximate solution within two times the optimal solution value. It is a greedy algorithm with respect to the cluster centers. The cluster centers are selected from the given points and each point always belongs to the cluster with the nearest center.

Algorithm: *Farthest-point clustering* [25]
Input: A set of N points $S = \{s_i \mid i = 1, \ldots, N\} \subset \mathbb{R}^d$
Desired number of clusters $K \leq N$
Output: K pairwise disjoint clusters $Z_{n=1,\ldots,K}$ with $\bigcup_{n=1,\ldots,K} Z_n = S$, and
K cluster centers $s_n^* \in Z_n$, such that $\max_n \max_{s_i \in Z_n} \|s_i - s_n^*\|$ is minimal

```
Choose an arbitrary point as the first cluster center s_1^*.
For every point calculate its distance to s_1^*, store the distance
  and the cluster that the point belongs to (= Z_1).
```

```
  FOR n = 2..K
    Find the point with the maximum distance to its cluster center
      to get the next cluster center s*_n.
    For every point compare its distance to the new cluster center s*_n
      with its current distance; choose the smaller distance and
      the corresponding cluster as new values for the point.
```

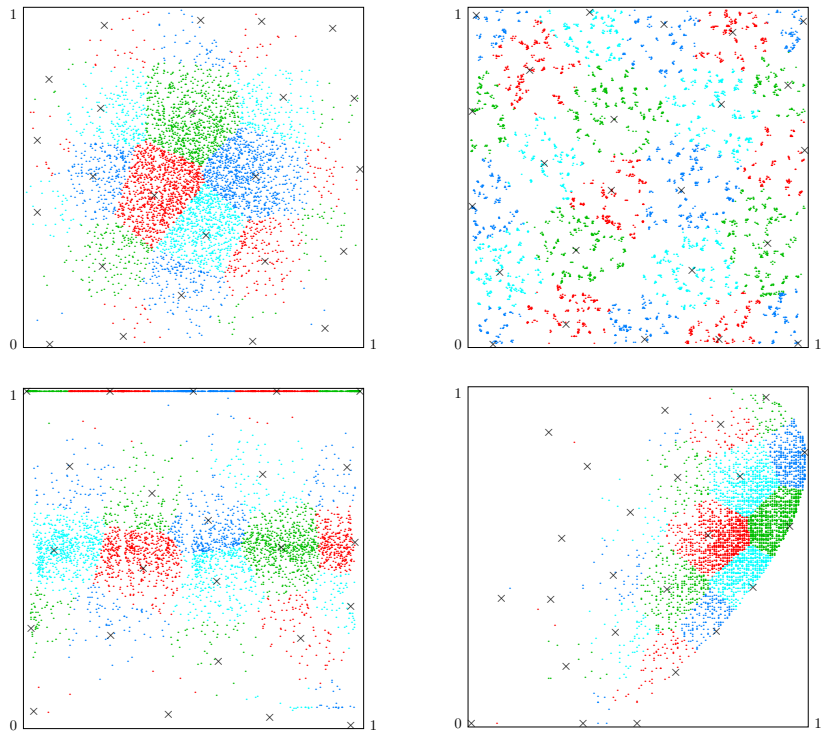


FIGURE 4.6: *Farthest-point clustering* applied to different 2-dimensional data sets with 5000 points and 24 clusters. Center points are indicated with *crosses*.

We will now present the proof for the claimed approximation factor of 2 compared to the optimal solution.

Theorem 4.13. *The* farthest-point clustering *algorithm calculates a partition with a maximum cluster radius not exceeding twice the optimum of the* K-center problem.

Proof. Let s_n^*, $n = 1, \ldots, K$ be the cluster centers computed by the algorithm. The maximum cluster radius r_{fpc} is now given by the distance of the point s_{K+1}^* (that would have been taken as $(K+1)$th cluster center) to its current cluster center:

$$r_{fpc} = \max_{n=1,\ldots,K} \max_{s_i \in Z_n} \|s_i - s_n^*\| = \min_{n=1,\ldots,K} \|s_{K+1}^* - s_n^*\|.$$

Consider now the optimal *K-center* solution with maximum cluster radius r_{opt}. Obviously, two of the $K+1$ points s_n^* $(n = 1, \ldots, K+1)$, say s_i^* and s_j^* $(1 \leq i < j \leq K+1)$, belong to the same cluster center c. This implies $\|s_i^* - c\| \leq r_{opt}$ and $\|s_j^* - c\| \leq r_{opt}$. Further note that

$$r_{fpc} = \min_{n=1,\ldots,K} \|s_{K+1}^* - s_n^*\| \leq \min_{n=1,\ldots,j-1} \|s_{K+1}^* - s_n^*\| \leq \min_{n=1,\ldots,j-1} \|s_j^* - s_n^*\| \leq \|s_j^* - s_i^*\|,$$

where the second $\leq$ estimation holds, since s_j^* was chosen by the *farthest-point clustering* as the point with maximum distance to its cluster center. Using the triangle inequality we get

$$r_{fpc} \leq \|s_j^* - s_i^*\| \leq \|s_j^* - c\| + \|s_i^* - c\| \leq 2 \cdot r_{opt}.$$

□

The factor of 2 cannot be improved unless $P = NP$, see [35] for further details. The straightforward implementation of the *farthest-point clustering* given above has a runtime of $\mathcal{O}(N \cdot K \cdot d)$, since for every new cluster center it calculates the distances from all sources to the center. A (potentially) faster version has been presented by Feder and Greene [18], the runtime $\mathcal{O}(N \cdot \log_2 K \cdot F(d))$ is optimal with respect to N and K. However the dependence on the dimension $F(d)$ is not plain linear as for the simple greedy algorithm. [18] only provides a worst case bound of d^d. Thus, this algorithm should be used with extreme caution in high dimensions.

4.2.3 Implementation details of the IFGT

The main steps of the *Improved Fast Gauss Transform* are the following ones: First, the number of clusters K depending on the bandwidth h and the specified precision ϵ are determined. The sources are then divided into K clusters by the *farthest-point clustering* algorithm. Next, for each cluster Z_n the interaction radius r_n^{int} and the series cutoff parameter p_n are determined (see Lemma 4.14 below) and the *IFGT-Taylor Coefficients* $C_\alpha(Z_n)$ for all $|\alpha| \leq p_n$ are computed. Finally the algorithm passes through all targets t_j calculating interactions with each cluster Z_n for which t_j is within the interaction radius r_n^{int}. We can therefore write the *IFGT* formally as:

$$\widetilde{G}_{IFGT}(t_j) = \sum_{\|t_j - s_n^*\| \leq r_n^{int}} \sum_{|\alpha| < p_n} C_\alpha(Z_n) \cdot e^{-\|t_j - s_n^*\|^2/h^2} \left(\frac{t_j - s_n^*}{h}\right)^\alpha. \tag{4.22}$$

The *IFGT* thus introduces a second kind of error arising from ignoring sources farther than the interaction radius. The following Lemma describes how to choose the parameters in order to yield the desired error bound.

Lemma 4.14 (*IFGT* error control parameters)**.** *Let t_j be a target point; for every source cluster Z_n let s_n^* and r_n be its center and radius, respectively, and let the interaction radius $r_n^{int} = \min\{R, r_n + h \cdot \sqrt{\ln(\epsilon^{-1})}\}$, where $R = \max_{i,j} \|t_j - s_i\|$ is the maximum distance between any two data points. If p_n is chosen, such that $\frac{1}{p_n!}\left(\frac{2 \cdot r_n^{int} \cdot r_n}{h^2}\right)^{p_n} \leq \epsilon$, the error introduced by approximating the discrete Gauss Transform $G(t_j) = \sum_{i=1}^N w_i \cdot e^{-\|t_j - s_i\|^2/h^2}$ with the* IFGT *expansion $\widetilde{G}_{IFGT}(t_j)$ is bounded by $\left|G(t_j) - \widetilde{G}_{IFGT}(t_j)\right| \leq W \cdot \epsilon$, where $W = \sum_{i=1}^N |w_i|$.*

Proof. First, we can bound the total error $E(t_j)$ by the sum of $E_1(t_j)$, the error induced by omitting the remote clusters, and $E_2(t_j)$, the error induced by truncating the Taylor series:

$$E(t_j) = \left| G(t_j) - \widetilde{G}_{IFGT}(t_j) \right| \leq E_1(t_j) + E_2(t_j),$$

where

$$E_1(t_j) = \left| \sum_{\|t_j - s_n^*\| > r_n^{int}} w_i \cdot e^{-\|t_j - s_i\|^2/h^2} \right|,$$

$$E_2(t_j) = \left| \sum_{\|t_j - s_n^*\| \leq r_n^{int}} \left[w_i \cdot e^{-\|t_j - s_i\|^2/h^2} - \sum_{|\alpha|<p} C_\alpha(s_n^*) \cdot e^{-\|t_j - s_n^*\|^2/h^2} \left(\frac{t_j - s_n^*}{h} \right)^\alpha \right] \right|.$$

To derive an estimation for $E_1(t_j)$, we provide a lower bound for $\|t_j - s_i\|$:

$$\|t_j - s_i\| = \|t_j - s_n^* + s_n^* - s_i\| \geq \left| \|t_j - s_n^*\| - \|s_n^* - s_i\| \right| \geq \left| r_n^{int} - r_n \right| \geq h \cdot \sqrt{\ln\left(\epsilon^{-1}\right)},$$

where the last estimation follows from the assumption on r_n^{int}. We can now bound $E_1(t_j)$ by

$$E_1(t_j) \leq \sum_{\|t_j - s_n^*\| > r_n^{int}} |w_i| \cdot e^{-h^2 \cdot \ln\left(\epsilon^{-1}\right)/h^2} = \epsilon \cdot \sum_{\|t_j - s_n^*\| > r_n^{int}} |w_i| \,. \tag{4.23}$$

On the other hand, using Lemma 4.11 the second error term $E_2(t_j)$ can be bounded by

$$E_2(t_j) \leq \sum_{\|t_j - s_n^*\| \leq r_n^{int}} |w_i| \cdot \frac{1}{p_n!} \left(\frac{2 \cdot r_n^{int} \cdot r_n}{h^2} \right)^{p_n} \leq \epsilon \cdot \sum_{\|t_j - s_n^*\| \leq r_n^{int}} |w_i| \,.$$

This proves the Lemma, since

$$E(t_j) \leq E_1(t_j) + E_2(t_j) \leq \epsilon \cdot \sum_{i=1}^{N} |w_i| = \epsilon \cdot W.$$

□

The guaranteed error bound of $W \cdot \epsilon$ for the *Improved Fast Gauss Transform* is simply an adapted absolute error bound. In [49] the authors note that in case all weights are positive the relative error can be bounded by

$$\epsilon_{rel} = \frac{\left| \widetilde{G}_{IFGT}(t_j) - G(t_j) \right|}{|G(t_j)|} = \frac{\left| \widetilde{G}_{IFGT}(t_j) - G(t_j) \right|}{\left| \sum_{i=1}^{N} w_i \cdot e^{-\|t_j - s_i\|^2/h^2} \right|} \leq \frac{\left| \widetilde{G}_{IFGT}(t_j) - G(t_j) \right|}{W \cdot e^{-R^2/h^2}} \leq \epsilon \cdot e^{R^2/h^2},$$

where $R = \max_{i,j} \|t_j - s_i\|$ is the maximum distance between any source and target point. However this bound is only useful for large h, that is $h \geq R$. If on the other hand $h \ll R$, the estimation $\epsilon_{rel} \leq \epsilon \cdot e^{R^2/h^2}$ will be much too pessimistic.

Before presenting the formal description of the *IFGT* in pseudo-code, we will highlight two

more issues for an efficient implementation. As shown in Lemma 4.14, the truncation numbers p_n for the Taylor expansion are chosen individually for each cluster Z_n instead of globally as done in the *FGT*. This requires some extra work and will only improve performance, if the p_n are different from each other, that is if the clusters have different radii r_n. Even though the farthest-point clustering is designed to minimize (and therefore equalize) the radii, experiments with varying datasets have shown the superiority of variable p_n's in terms of speed. In [49] it is further suggested that the truncation number is not only adapted to each cluster, but even to each source point belonging to the cluster, such that in the calculation of the coefficients

$$C_\alpha(Z_n) = \frac{2^{|\alpha|}}{\alpha!} \sum_{s_i \in Z_n} w_i \cdot e^{-\|s_i - s_n^*\|^2/h^2} \left(\frac{s_i - s_n^*}{h} \right)^\alpha$$

sources close to the cluster center s^* are not taken into account for high values of $|\alpha|$. However the extra effort created by managing different truncation numbers for each source has not turned out to be compensated by resulting time savings in practical scenarios.

The second issue concerns the way the coefficients $C_\alpha(Z_n)$ are stored and calculated. The $\binom{p_n-1+d}{d}$ coefficients are stored in graded lexicographic order in a simple vector. They can hence be computed in a hierarchical way. This means that the products for $|\alpha| = n$ are calculated multiplying the values for $|\alpha| = n-1$ with some value for $|\alpha| = 1$. Figure 4.7 illustrates the procedure for a given point $v \in \mathbb{R}^3$ with $v = (x, y, z)$ and the calculation of v^α for all $|\alpha| \leq 4$.

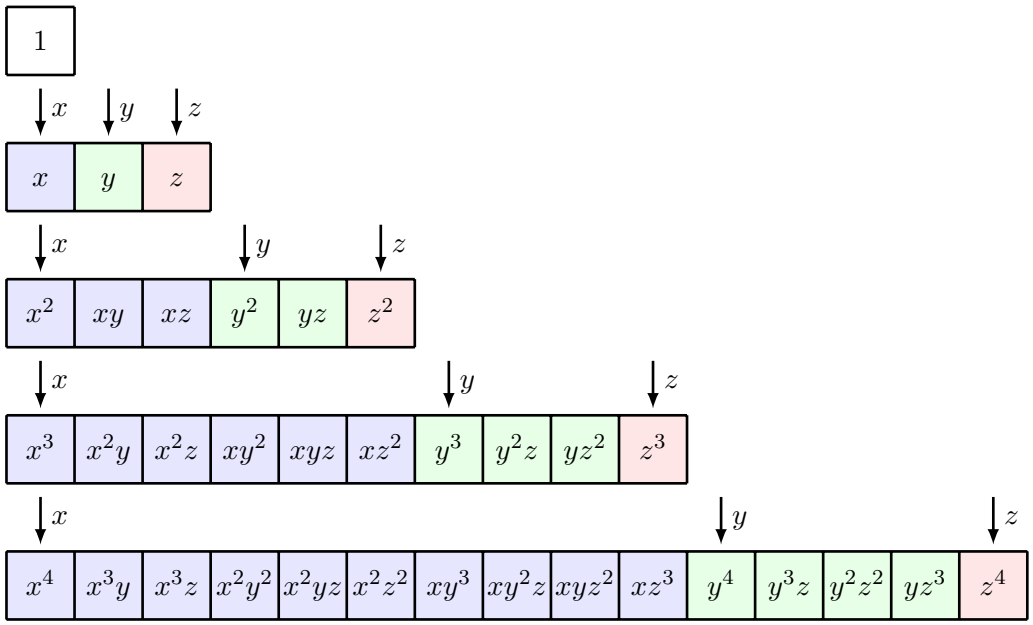


FIGURE 4.7: Efficient calculation of coefficients. The products for $|\alpha| = n$ are calculated by multiplying the values for $|\alpha| = n-1$ with x, y and z.

Thus each of the $\binom{p_n-1+d}{d}$ products requires only a single multiplication. This results in a total runtime of $\mathcal{O}\left(N \cdot \binom{p_{max}-1+d}{d}\right)$ for the calculation of all coefficients, where $p_{max} = max_{(n)}\{p_n\}$.

We now introduce the pseudo-code of the *IFGT*. To keep the code clear and easily readable, we only provide the main steps of the implementation. Particularly the calculation of the number of clusters K will be discussed straight after the formal description of the algorithm.

Algorithm: *Improved Fast Gauss Transform* [67], [49]
Input: N sources s_i, $(i = 1, \ldots, N)$ in $\mathbb{R}^d$
M targets t_j, $(j = 1, \ldots, M)$ in $\mathbb{R}^d$
N weights w_i, $(i = 1, \ldots, N)$ in $\mathbb{R}$
bandwidth $h > 0$, error bound $\epsilon > 0$
Output: $\widetilde{G}_{IFGT}(t_j)$, $(j = 1, \ldots, M)$, with $\left|\widetilde{G}_{IFGT}(t_j) - G(t_j)\right| < \epsilon \cdot \sum_{i=1}^{N} |w_i|$

```
Choose the number of clusters K.
Divide the N source points into K clusters Z_n, n = 1,...,K,
  with centers s*_n and radii r_n.
For each cluster Z_n determine the interaction radius r^int_n
  and the truncation number p_n using Lemma 4.14.

For each cluster Z_n compute coefficients C_α(Z_n) for all |α| < p_n:
  C_α(Z_n) = (2^|α| / α!) Σ_{s_i ∈ Z_n} w_i · e^{-||s_i - s*_n||^2/h^2} ((s_i - s*_n)/h)^α.
For each target t_j find clusters Z_n with ||t_j - s*_n|| ≤ r^int_n and compute
  G̃_IFGT(t_j) = Σ_{||t_j - s*_n|| ≤ r^int_n} Σ_{|α|<p} C_α(Z_n) · e^{-||t_j - s*_n||^2/h^2} ((t_j - s*_n)/h)^α.
```

4.2.4 A new approach for the adapted choice of the number of clusters

In the revised version of the *Improved Fast Gauss Transform* [49] the authors suggest a method to choose the number of clusters K, which is optimized for uniform distribution of the source points. However in practice, most problems deal with rather non-uniformly distributed datasets. On the other hand, for uniformly distributed data a simple box structure – like the one used in the *FGT* – would be as good as the clustering, easier to implement and faster in terms of data access and distance calculations. Therefore, we suggest a different method described below.

For a better understanding of a good choice of K consider the two extreme cases:

- $\mathbf{K = 1}$: The whole evaluation is reduced to a single Taylor expansion centered at the point s_1^*. Further we have $r_1^{int} \approx r_1 \approx R = \max_{i,j} \|t_j - s_i\|$. So a moderate p_1 satisfying $\frac{1}{p_1!}\left(\frac{2 \cdot r_1^{int} \cdot r_1}{h^2}\right)^{p_1} \approx \frac{1}{p_1!}\left(\frac{2 \cdot R^2}{h^2}\right)^{p_1} < \epsilon$ can only be found if $h \gg R$, that is for high bandwidth.
- $\mathbf{K = N}$: The algorithm corresponds to the direct evaluation except for the influence of the interaction radii. Every cluster Z_n contains a single point, its center s_n^*. If we assume $s_n^* = s_n$, we get $C_0(Z_n) = w_n$ and $C_\alpha(Z_n) = 0$ for $|\alpha| > 0$. Therefore the evaluation reduces to $\widetilde{G}_{IFGT}(t_j) = \sum_{\|t_j - s_n\| \le r_n^{int}} w_n \cdot e^{-\|t_j - s_n\|^2/h^2}$.

While the case $K = 1$ is quite relevant for problems with high bandwidth and often results in very fast computation, the case $K = N$ should be avoided, since the clustering itself already requires $\mathcal{O}(N^2 \cdot d)$, or $\mathcal{O}(N \cdot \log_2 N \cdot F(d))$ if using FEDER and GREENE's version. Adding this to the runtime $\mathcal{O}(N \cdot M \cdot d)$ of the direct evaluation results in a more than suboptimal runtime. In order to minimize the total cost of the computation, we suggest an adaptive version of the clustering process.

- Using GONZALEZ' greedy algorithm, we start with a single cluster, determine the parameters r_1, r_1^{int} and p_1 and estimate the costs for this configuration.
- Iteratively, we add a new cluster to get a cost estimation for the new configuration.
- Once the costs have been increasing for too many steps or have been decreasing too slowly, we stop the process and choose the number of clusters with minimal costs so far.

Superfluous clusters (if any) can be deleted easily and quickly because of the greedy nature of the algorithm. Finally, the minimal costs can be compared with the costs for direct evaluation in order to choose the cheapest method. In order to present a pseudo-code solution for the adaptive clustering we first have to provide runtime estimations for the two main steps of the *IFGT*:

Using the hierarchical calculation presented above, the coefficients for a cluster Z_n containing l_n sources can be carried out in $\mathcal{O}\left(l_n \cdot \binom{p_{max}-1+d}{d}\right)$ time. To estimate the costs for the evaluation of the Taylor series we need to go through all M targets and find interacting source clusters. Let n_c be the maximum number of clusters interacting with an arbitrary target. Since the series can be evaluated using the efficient coefficient scheme, the runtime adds up to $\mathcal{O}\left(M \cdot n_c \cdot \binom{p_{max}-1+d}{d}\right)$. For the adaptive clustering, we have to find a good estimate for n_c to be able to calculate the approximate costs. As it would be too expensive to go through all targets and find the interacting clusters in each step, we rather first combine the targets in $K_T \ll M$ (target) clusters, only work with their centers and assume every target point has the same interacting (source) clusters as its (target cluster) center. We now present the formal description of the adaptive clustering process. Speaking of "clusters" we always refer to source clusters, otherwise we say "target clusters".

Algorithm: *IFGT with adaptive clustering*
Input: N sources s_i, $(i = 1, \ldots, N)$ in $\mathbb{R}^d$
M targets t_j, $(j = 1, \ldots, M)$ in $\mathbb{R}^d$
N weights w_i, $(i = 1, \ldots, N)$ in $\mathbb{R}$
bandwidth $h > 0$, error bound $\epsilon > 0$
Output: $\widetilde{G}_{IFGT}(t_j)$, $(j = 1, \ldots, M)$, with $\left|\widetilde{G}_{IFGT}(t_j) - G(t_j)\right| < \epsilon \cdot \sum_{i=1}^{N} |w_i|$

```
Divide the M target points into K_T target clusters
  and store the target cluster centers t*_m and populations l^t_m.
Estimate costs for direct evaluation cost_direct.
```

```
Initialize the current number of clusters c = 0.
REPEAT
  Add 1 new cluster (as in farthest-point clustering); c = c + 1.
  Determine cluster population l_n, interaction radius r_n^int
    and Taylor cutoff p_n for each cluster Z_n.
  Estimate costs for calculating the coefficients:
    cost^c_coeff = sum_{n=1}^{c} O( l_n · C(p_n - 1 + d, d) ).
  Estimate costs for evaluating Taylor series using the targets t*_m:
    cost^c_eval = sum_{||t*_m - s*_n|| <= r_n^int} O( l^t_m · C(p_n - 1 + d, d) ).
  Total costs cost^c_total = cost^c_coeff + cost^c_eval.
  Estimate costs for the clustering cost^c_cluster done so far.
UNTIL (exit condition)

Choose final number of clusters K with cost^K_total = min_{i=1,...,c} { cost^i_total }.
IF (cost_direct < cost^K_total)
  Direct evaluation.
ELSE
  Delete superfluous clusters Z_{K+1}, ..., Z_c if necessary.
  IFGT.
```

Some remarks concerning the code: We are using the Landau notation for the computation of $cost^c_{coeff}$ and $cost^c_{eval}$. In practice, the cost estimation naturally has to take into account the particular implementation and use appropriate constants to achieve better results. The *exit condition* in the `REPEAT-UNTIL` expression has to be chosen carefully to get a reasonable ratio between runtime spent for the clustering and runtime of the remaining calculations. We experimented with various constraints and worked out multiple exit conditions that can roughly be summarized as follows:

1) if $cost^c_{cluster} > cost_{naive}$;

2) if $c > 10$ and $cost^c_{total} \gg cost_{naive}$;

3) if $cost^c_{total}$ has only been increasing for the last 10 steps;

4) if $cost^c_{total}$ has been decreasing too slowly for the last 20 steps;

5) if $c > N/10$.

Note that the first constraint assures that the *IFGT with adaptive clustering* never exceeds twice the runtime of the direct evaluation, which was not the case for the original *IFGT*. The last constraint guarantees that the final number of clusters K does not come to close to the useless case $K = N$. Of course, the given parameters should still be optimized experimentally, as they will strongly depend on the implementation as well as the range of the main parameters N, M and d.

4.2.5 Runtime analysis and comparison with the FGT

The *Improved Fast Gauss Transform* is obviously more complex than the *Fast Gauss Transform*, but does it manage to avoid all the defects the *FGT* suffers from? First, the clustering scheme is able to deal with datasets in much higher dimensions than the simple box structure used in the *FGT*. It does not scale exponentially with the dimension and works well with non-uniformly distributed data. As analyzed above, the runtime complexity can be summarized as follows.

Runtime Complexity 4.15. *The* IFGT *has a runtime of*

$$\mathcal{O}\left(N \cdot K \cdot d + (N + M \cdot n_c) \cdot \binom{p_{max} - 1 + d}{d}\right),$$

where K is the number of clusters, n_c is the maximum number of interacting clusters and p_{max} is the maximum Taylor cutoff.

Compared to the *FGT*'s runtime $\mathcal{O}((N + M) \cdot p^d \cdot d)$ (where $p = p_{max} - 1$ can be assumed) the main improvement is the factor $\binom{p_{max}-1+d}{d}$ which increases much slower for $d \to \infty$ than p^d. To compare the two terms, we make use of *Stirling's Formula*.

Lemma 4.16. ***Stirling's Formula*** *The faculty of any positive integer n can be approximated by*

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n,$$

more precisely the error goes to 0 with $n \to \infty$:

$$\lim_{n\to\infty} \frac{n!}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n} = 1.$$

Now let $p = p_{max} - 1$ to rewrite

$$\binom{p+d}{d} = \frac{(p+d)!}{p! \cdot d!} \approx \frac{\sqrt{2\pi}\sqrt{p+d}\left(\frac{p+d}{e}\right)^{p+d}}{\sqrt{2\pi}\sqrt{p}\left(\frac{p}{e}\right)^p \cdot \sqrt{2\pi}\sqrt{d}\left(\frac{d}{e}\right)^d} = \frac{(p+d)^{p+d+\frac{1}{2}}}{\sqrt{2\pi}p^{p+\frac{1}{2}} \cdot d^{d+\frac{1}{2}}} \approx \frac{(p+d)^{p+d+\frac{1}{2}}}{p^{p+\frac{1}{2}} \cdot d^{d+\frac{1}{2}}}.$$

Take the logarithm to get

$$\begin{aligned}\log\binom{p+d}{d} &\approx \left(p+d+\frac{1}{2}\right)\cdot\log(p+d) - \left(p+\frac{1}{2}\right)\cdot\log p - \left(d+\frac{1}{2}\right)\cdot\log d \\ &= \underbrace{p\cdot\log(p+d)}_{\text{logarithmic growth in } d} + \left(d+\frac{1}{2}\right)\cdot[\log(p+d) - \log d] - \underbrace{\left(p+\frac{1}{2}\right)\cdot\log p}_{\text{constant in } d}.\end{aligned}$$

Thus, we basically have to compare the term

$$\left(d+\frac{1}{2}\right)\cdot[\log(p+d) - \log d] = \left(d+\frac{1}{2}\right)\cdot\log\left(1+\frac{p}{d}\right)$$

with the logarithm of the FGT-term

$$\log p^d = d \cdot \log p.$$

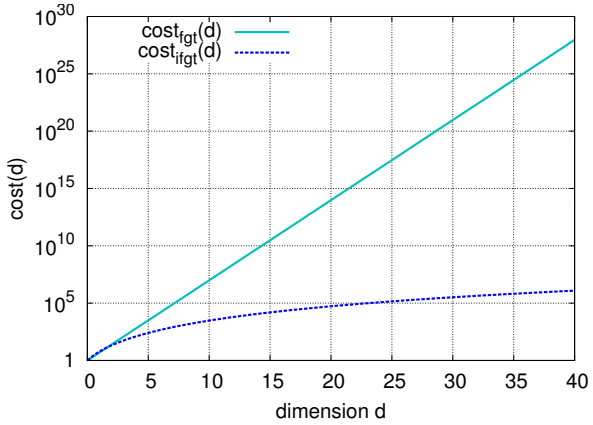


FIGURE 4.8: $cost_{\text{fgt}}(d) = p^d$ and $cost_{\text{ifgt}}(d) = \binom{p+d}{d}$ for $p = 5$ in a logarithmic plot.

The last term has a plain linear growth in d, while for the first term, the linear growth of $\left(d + \frac{1}{2}\right)$ is substantially reduced by the factor $\log\left(1 + \frac{p}{d}\right)$ going to 0 for $d \to \infty$. The effect can clearly be seen in Figure 4.8.

Unlike the FGT, the *Improved Fast Gauss Transform* uses local error control parameters p_n and r_n^{int} to guarantee an error close to the requested bound ϵ with minimal costs. However, the $IFGT$ also suffers from some drawbacks.

4.2.6 Discussion of bandwidth domains

The main issues of the *Improved Fast Gauss Transform* are:

- The $IFGT$ only provides a scaled absolute error bound, not a relative one.
- The choice of the parameters for the clustering process – even using the *adaptive clustering* – is non-trivial and requires more experimental studies.
- The $IFGT$ does not perform well dealing with problems involving low bandwidths.

In order to get a better understanding of the influence of the bandwidth h on the algorithmic performance, we consider the example of four Gaussians $G_s(t) = e^{-\|t-s\|^2/h^2}$ centered at the source points $s_1 = -2$, $s_2 = 0$, $s_3 = 0.3$ and $s_4 = 1$, respectively, with different bandwidths $h = 0.1$, $h = 1$ and $h = 10$ given in Figure 4.9. Our task is to calculate the sum of all Gaussians at the target point $t = 0$.

The left plot shows the situation of relatively small bandwidth $h = 0.1$. Obviously, the only Gaussian with a significant contribution to the target is the one centered at the very closest source (in fact, the closest source is equal to the target in this case). The plot in the middle provides an example of average bandwidth ($h = 1$). Here the influence of the different

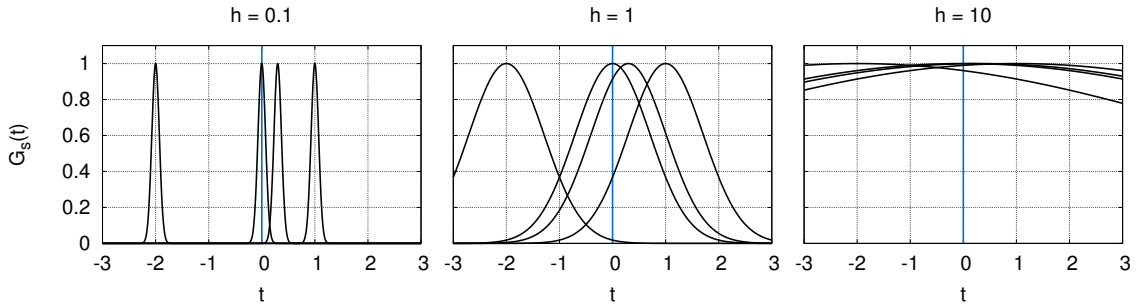


FIGURE 4.9: Four Gaussians $G_s(t) = e^{-\|t-s\|^2/h^2}$, $s \in \{-2,\ 0,\ 0.3,\ 1\}$ with bandwidth $h = 0.1$ *(left)*, $h = 1$ *(center)* and $h = 10$ *(right)*.

Gaussians on t heavily depends on the distance $\|s_i - t\|$, only sources far away from the target can be ignored. On the right hand side we finally have an example of high bandwidth $h = 10$. The contribution of the Gaussians centered at s_2, s_3 and s_4 is nearly equal.

This brief debate leads us to the following requirements for an algorithm that has to deal with varying bandwidths:

- **Low bandwidth:** For each target, the algorithm has to find the very nearest sources and ignore contributions from the other sources. A fast *nearest neighbor search* is mandatory.
- **High bandwidth:** Sources must be combined in clusters such that their influence on a target can be accumulated in an efficient expansion (as done in the *FGT* and *IFGT*).
- **Moderate bandwidth:** Flexible switching and good decision strategies are necessary to find the most cost-efficient methods.

While the Taylor series work pretty well for the case of high bandwidth, both the *FGT* and the *IFGT* lack of an appropriate mechanism to handle the nearest neighbor problem. The *FGT* already suffers from dividing up the unit hyper cube $[0,1]^d$ in n_{side}^d boxes with $n_{side} = 1/\sqrt{2}rh$ ($r < \frac{1}{2}$), which – for small h – results in a huge number of boxes slowing down the computation excessively. The *IFGT* on the other hand searches for the nearest source clusters for each target individually, which is not a good method if there are many clusters.

Consequently a much more flexible data structure is necessary in order to fulfill the demands from the different bandwidth domains. The algorithm to be discussed next comes up with appropriate features.

4.3 Dual-Tree Fast Gauss Transform

Data structure: Two tree structures (SR-Tree) for sources and targets

Expansions: *Hermite*, *Taylor* and *Taylor-Hermite Expansion* (l_1-version)

Guaranteed global error: $\left|\widetilde{G}_{DFGT}(t_j) - G(t_j)\right| < \epsilon \cdot \left|G(t_j)\right|$ for a fixed bound $\epsilon > 0$

Runtime complexity: $\mathcal{O}\left(\min\left(dNM, (N+M)\cdot\binom{p_{max}-1+d}{d}\right)\right) +$
$\mathcal{O}\left(d\left(N\log_2 N + M\log_2 M\right)\right)$, where p_{max} is a truncation parameter for series expansion.

Another variant for the approximation of the *Discrete Gauss Transform* was presented by Dongryeol Lee, Alexander Gray and Andrew Moore [42] in 2005, the so-called *Dual-Tree Fast Gauss Transform (DFGT)*. The Dual-Tree algorithm – as opposed to the *FGT* and *IFGT* algorithms – can be classified as a genuine *multilevel* fast multipole method. Instead of using a static clustering of source (target) points, two trees are constructed, a source tree and a target tree, whose nodes represent clusters, that are hierarchically contained in each other.

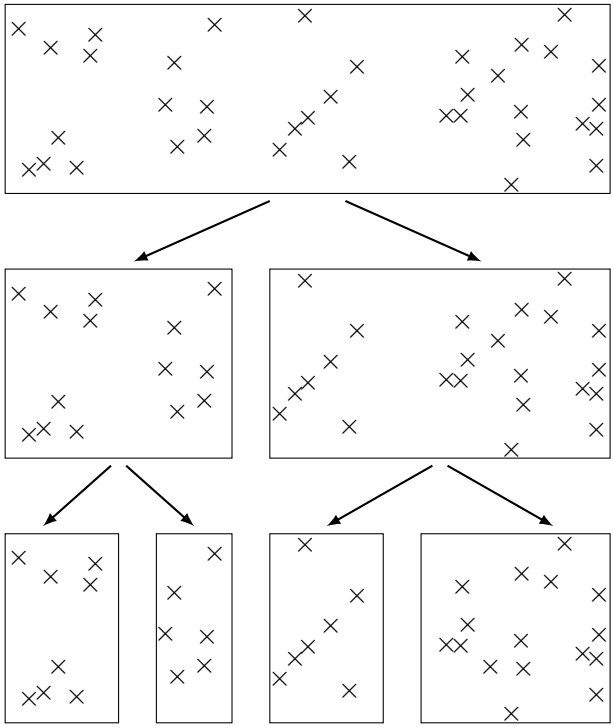


Figure 4.10: A binary tree structure for two-dimensional data.

An example for a binary tree is shown in Figure 4.10, yet the *DFGT* can be based on more complex variants as well. The root of the source (target) tree represents the cluster of all sources (targets), while the child nodes of some inner node represent a disjoint partition of this node. Each node of the tree has its own expansion center. The leaves (nodes with no children) are assumed to contain either only 1 point or a number of points bounded by some constant. After building the two trees the algorithm traverses them simultaneously starting in the roots, calculating interactions between nodes instead of single points. If a node cluster is too big to provide a satisfactory approximation, the children are used instead. The key advantage of this technique is illustrated in Figure 4.11.

For target point t_1 far away from the source cluster C_1 it is advantageous to consider the influence of the whole cluster onto t_1, while for the target t_2 within cluster C_1 it might be necessary to make use of the children clusters $C_{11}, \ldots, C_{14}$. Furthermore, since the targets are clustered as well, decisions can be made for target clusters instead of single targets.

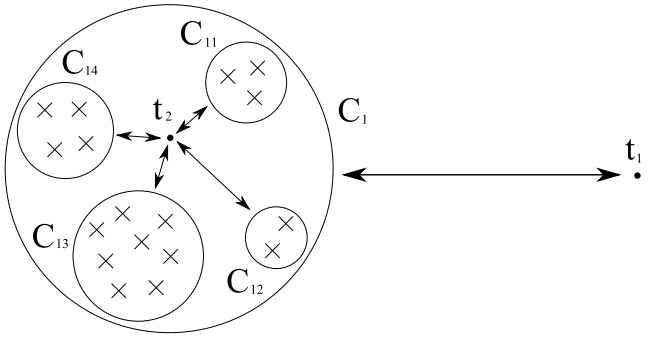


FIGURE 4.11: Source cluster C_1 with four children $C_{11}, C_{12}, C_{13}, C_{14}$. *Crosses* denote source points. Targets t_1, t_2 *(dots)* interact with either C_1 or each of $C_{11}, \ldots, C_{14}$.

4.3.1 Modified FGT-expansions and translation operators

In the original paper [42], LEE, GRAY and MOORE took the same expansions that were used in the *Fast Gauss Transform* (Def. 4.2) including the same error bounds. In 2006 the authors came up with a new version [41] using l_1-expansions with $\binom{p+d}{d}$ terms instead of l_∞-expansions with p^d terms. We therefore present the new definitions of the three expansions.

Definition 4.17 (DFGT Expansions)**.** *Let the same assumptions apply as in Definition 4.1. Further let B_{s^*} be a source cluster and B_{t^*} a target cluster with expansion centers s^* and t^*, respectively. Let $t_j \in B_{t^*}$ and $p \geq 1$ be an arbitrary, but fixed integer. We then define the*

- ***Hermite Coefficient*** *(due to source cluster B_{s^*})*

$$C_\alpha^H(B_{s^*}) = \frac{1}{\alpha!} \cdot \sum_{s_i \in B_{s^*}} w_i \cdot \left(\frac{s_i - s^*}{h} \right)^\alpha; \tag{4.24}$$

- ***Taylor Coefficient*** *(due to center t^* and source cluster B_{s^*})*

$$C_\alpha^T(B_{s^*}, t^*) = \frac{1}{\alpha!} \cdot \sum_{s_i \in B_{s^*}} w_i \cdot h_\alpha \left(\frac{s_i - t^*}{h} \right); \tag{4.25}$$

- ***Taylor-Hermite Coefficient*** *(due to center t^* and source cluster B_{s^*})*

$$C_\beta^{TH_p}(B_{s^*}, t^*) = \frac{(-1)^{|\beta|}}{\beta!} \cdot \sum_{|\alpha| < p} C_\alpha^H(B_{s^*}) \cdot h_{\alpha+\beta} \left(\frac{t^* - s^*}{h} \right). \tag{4.26}$$

Further, we define the

- ***Hermite Expansion*** ***(l_1-version)*** *(due to target t_j and source cluster B_{s^*})*

$$\widetilde{G}^{H_p}_{B_{s^*}}(t_j) = \sum_{|\alpha|<p} C^H_\alpha(B_{s^*}) \cdot \; h_\alpha\left(\frac{t_j - s^*}{h}\right); \tag{4.27}$$

- ***Taylor Expansion*** ***(l_1-version)*** *(due to target t_j and source cluster B_{s^*})*

$$\widetilde{G}^{T_p}_{B_{s^*}}(t_j) = \sum_{|\alpha|<p} C^T_\alpha(B_{s^*}, t^*) \cdot \left(\frac{t_j - t^*}{h}\right)^\alpha; \tag{4.28}$$

- ***Taylor-Hermite Expansion*** ***(l_1-version)*** *(due to target t_j and source cluster B_{s^*})*

$$\widetilde{G}^{TH_p}_{B_{s^*}}(t_j) = \sum_{|\beta|<p} C^{TH_p}_\beta(B_{s^*}, t^*) \cdot \left(\frac{t_j - t^*}{h}\right)^\beta. \tag{4.29}$$

In order to benefit from the hierarchical tree structure, the *DFGT* makes use of two new translation operators to shift results from a node to its children and vice versa. The *Hermite-to-Hermite Operator* provides a fast method to calculate the Hermite coefficients $C^H_\alpha(B_s)$ for a source cluster B_s, if the coefficients $C^H_\alpha(B_{s'})$ of all child nodes $B_{s'}$ are known. The following easy calculation, that involves a multi-dimensional Taylor expansion of the Hermite function, shifts the expansion from the child node to the parent node:

$$\begin{aligned}
G^H_{B_{s'}}(t_j) &= \sum_{\alpha\geq 0} C^H_\alpha(B_{s'}) \cdot h_\alpha\left(\frac{t_j - s'}{h}\right) \\
&= \sum_{\alpha\geq 0} C^H_\alpha(B_{s'}) \cdot \sum_{\gamma\geq 0} \frac{1}{\gamma!}\left(\frac{s' - s}{h}\right)^\gamma \cdot h_{\alpha+\gamma}\left(\frac{t_j - s}{h}\right) \\
&= \sum_{\alpha\geq 0}\sum_{\gamma\geq 0} \frac{1}{\gamma!}\, C^H_\alpha(B_{s'}) \cdot \left(\frac{s' - s}{h}\right)^\gamma \cdot h_{\alpha+\gamma}\left(\frac{t_j - s}{h}\right) \\
&= \sum_{\beta\geq 0}\left[\sum_{0\leq\alpha\leq\beta} \frac{1}{(\beta-\alpha)!}\, C^H_\alpha(B_{s'}) \cdot \left(\frac{s' - s}{h}\right)^{\beta-\alpha}\right] \cdot h_\beta\left(\frac{t_j - s}{h}\right),
\end{aligned}$$

where $\beta = \alpha + \gamma$. We thus have the following result.

Lemma 4.18 (Hermite-to-Hermite Operator)**.** *Let the same assumptions apply as in Definition 4.1. Let B_s be a source cluster (with expansion center s) in the source tree and $B_{s'_1}, \ldots, B_{s'_n}$ be its children (with expansion centers $s'_1, \ldots, s'_n$), such that $B_s = \bigcup_{j=1}^n B_{s'_j}$. Then the Hermite Coefficients due to source cluster B_s can be calculated using the Hermite Coefficients due to*

the children source clusters $B_{s'_1}, \ldots, B_{s'_n}$ *by*

$$C_\beta^H(B_s) = \sum_{j=1}^{n} \left[\sum_{0 \le \alpha \le \beta} \frac{1}{(\beta-\alpha)!} C_\alpha^H\left(B_{s'_j}\right) \cdot \left(\frac{s'_j - s}{h}\right)^{\beta-\alpha} \right]. \tag{4.30}$$

The second operator, called *Local-to-Local Operator*, shifts a *Taylor Expansion* centered at the expansion center of a given node to the corresponding expansion center of a child node. More precisely, given a target node B_t with expansion center t and the corresponding *Taylor Coefficients* $C_\alpha^T(B_s, t)$, the operator allows the efficient computation of the *Taylor Coefficients* $C_\beta^T(B_s, t')$ of a child node $B_{t'}$ with expansion center t'. The following calculation involves a multi-dimensional Taylor expansion of $f(t_j) = \left(\frac{t_j - t}{h}\right)^\alpha$ around the expansion center t':

$$\begin{aligned}
G_{B_s}^T(t_j) &= \sum_{\alpha \ge 0} C_\alpha^T(B_s, t) \cdot \left(\frac{t_j - t}{h}\right)^\alpha \\
&= \sum_{\alpha \ge 0} C_\alpha^T(B_s, t) \sum_{\beta \ge 0} \frac{1}{\beta!} \left(D^\beta \left[\left(\frac{t_j - t}{h}\right)^\alpha \right](t') \right) (t_j - t')^\beta \\
&= \sum_{\alpha \ge 0} C_\alpha^T(B_s, t) \sum_{\beta \le \alpha} \frac{1}{\beta!} \left(\frac{1}{h}\right)^\beta \left(\prod_{k=1}^{d} \alpha_k (\alpha_k - 1) \cdots (\alpha_k - \beta_k + 1) \right) \\
&\qquad \cdot \left(\frac{t' - t}{h}\right)^{\alpha-\beta} (t_j - t')^\beta \\
&= \sum_{\alpha \ge 0} \sum_{\beta \le \alpha} C_\alpha^T(B_s, t) \cdot \frac{\alpha!}{\beta!(\alpha-\beta)!} \left(\frac{t' - t}{h}\right)^{\alpha-\beta} \left(\frac{t_j - t'}{h}\right)^\beta \\
&= \sum_{\beta \ge 0} \left[\sum_{\alpha \ge \beta} \frac{\alpha!}{\beta!(\alpha-\beta)!} \cdot C_\alpha^T(B_s, t) \left(\frac{t' - t}{h}\right)^{\alpha-\beta} \right] \cdot \left(\frac{t_j - t'}{h}\right)^\beta.
\end{aligned}$$

We thus get the following Lemma for the *Local-to-Local Operator*.

Lemma 4.19 (Local-to-Local Operator)**.** *Let the same assumptions apply as in Definition 4.1. Let* B_s *be a source cluster,* B_t *be a target cluster (with expansion center* t*) and* $B_{t'}$ *(with expansion center* t'*) be a child of* B_t*. Further let* $t_j \in B_{t'}$*. Then the Taylor Coefficients due to center* t' *and source cluster* B_s *can be calculated using the Taylor Coefficients due to center* t *and source cluster* B_s *by*

$$C_\beta^T(B_s, t') = \sum_{\alpha \ge \beta} \frac{\alpha!}{\beta!(\alpha-\beta)!} \cdot C_\alpha^T(B_s, t) \left(\frac{t' - t}{h}\right)^{\alpha-\beta}. \tag{4.31}$$

4.3.2 Improved error bounds for l_1-expansions

The truncation of the infinite series in the *DFGT Expansions* 4.17 introduces an approximation error. Lee and Gray [41] suggest the following local error bounds:

Lemma 4.20 (Error bounds for the *DFGT Expansions*)**.** *Let t_j be a target point in the target cluster B_{t^*} with expansion center t^* and $r_{t^*}^{\infty} = \max_{t \in B_{t^*}} \|t - t^*\|_\infty / h$. Consider all sources s_i in the source cluster B_{s^*} with expansion center s^* and $r_{s^*}^{\infty} = \max_{s \in B_{s^*}} \|s - s^*\|_\infty / h$. Then the absolute errors introduced by approximating the discrete Gauss Transform $G_{B_{s^*}}(t_j) = \sum_{s_i \in B_{s^*}} w_i \cdot e^{-\|t_j - s_i\|^2/h^2}$ with the Hermite Expansion $\widetilde{G}^{H_p}_{B_{s^*}}(t_j)$, the Taylor Expansion $\widetilde{G}^{T_p}_{B_{s^*}}(t_j)$ and the Taylor-Hermite Expansion $\widetilde{G}^{TH_p}_{B_{s^*}}(t_j)$, respectively, have the following absolute truncation errors:*

$$E^{H_p}_{B_{s^*}}(t_j) = \left| G_{B_{s^*}}(t_j) - \widetilde{G}^{H_p}_{B_{s^*}}(t_j) \right| \leq W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{\binom{d+p-1}{d-1}}{\sqrt{\left(\lfloor \frac{p}{d} \rfloor !\right)^{d-p'} \left(\lceil \frac{p}{d} \rceil !\right)^{p'}}} \cdot \left(\sqrt{2} r_{s^*}^{\infty}\right)^p ;$$

$$E^{T_p}_{B_{s^*}}(t_j) = \left| G_{B_{s^*}}(t_j) - \widetilde{G}^{T_p}_{B_{s^*}}(t_j) \right| \leq W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{\binom{d+p-1}{d-1}}{\sqrt{\left(\lfloor \frac{p}{d} \rfloor !\right)^{d-p'} \left(\lceil \frac{p}{d} \rceil !\right)^{p'}}} \cdot \left(\sqrt{2} r_{t^*}^{\infty}\right)^p ;$$

$$\begin{aligned} E^{TH_p}_{B_{s^*}}(t_j) &= \left| G_{B_{s^*}}(t_j) - \widetilde{G}^{TH_p}_{B_{s^*}}(t_j) \right| \\ &\leq W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{\binom{d+p-1}{d-1}}{\sqrt{\left(\lfloor \frac{p}{d} \rfloor !\right)^{d-p'} \left(\lceil \frac{p}{d} \rceil !\right)^{p'}}} \\ &\quad \cdot \left[\left(\sqrt{2} r_{t^*}^{\infty}\right)^p + (2 r_{s^*}^{\infty})^p \cdot \binom{d+p-1}{d} \cdot \max\left\{ (2 r_{t^*}^{\infty})^{p-1}, 1 \right\} \right], \end{aligned}$$

where $W_{s^*} = \sum_{s_i \in B_{s^*}} |w_i|$, $\delta^{min}_{s^*t^*} = \min_{s \in B_{s^*}, t \in B_{t^*}} \|s - t\|_2 / h$ and $p' = p \mod d$.

The proof for these bounds can be found in [41]. Note, that they have an explicit dependence on d and involve the l_∞-norm in $r_{t^*}^{\infty}$ and $r_{s^*}^{\infty}$, which is counterintuitive, since l_1-approximations have been used. We propose improved error bounds, which will then be compared with the original ones.

Lemma 4.21 (Improved error bounds for the *DFGT Expansions*)**.** *Let t_j be a target point in the target cluster B_{t^*} with expansion center t^* and $r_{t^*} = \max_{t \in B_{t^*}} \|t - t^*\|_1 / h$. Consider all sources s_i in the source cluster B_{s^*} with expansion center s^* and $r_{s^*} = \max_{s \in B_{s^*}} \|s - s^*\|_1 / h$. Then the absolute errors introduced by approximating the discrete Gauss Transform $G_{B_{s^*}}(t_j) = \sum_{s_i \in B_{s^*}} w_i \cdot e^{-\|t_j - s_i\|^2/h^2}$ with the Hermite Expansion $\widetilde{G}^{H_p}_{B_{s^*}}(t_j)$, the Taylor Expansion $\widetilde{G}^{T_p}_{B_{s^*}}(t_j)$ and the Taylor-Hermite Expansion $\widetilde{G}^{TH_p}_{B_{s^*}}(t_j)$, respectively, have the following absolute truncation errors:*

$$E^{H_p}_{B_{s^*}}(t_j) \leq W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{1}{\sqrt{p!}} \cdot \left(\sqrt{2} r_{s^*}\right)^p ;$$

$$E^{T_p}_{B_{s^*}}(t_j) \leq W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{1}{\sqrt{p!}} \cdot \left(\sqrt{2} r_{t^*}\right)^p ;$$

$$E^{TH_p}_{B_{s^*}}(t_j) \leq W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{1}{\sqrt{p!}} \cdot \left[\left(\sqrt{2} r_{t^*}\right)^p + (2r_{s^*})^p \cdot \min \left\{ \frac{(2r_{t^*})^p - 1}{2r_{t^*} - 1}, \sqrt{(p-1)!} \cdot e^{2r_{t^*}} \right\} \right],$$

where $W_{s^*} = \sum_{s_i \in B_{s^*}} |w_i|$ *and* $\delta^{min}_{s^*t^*} = \min_{s \in B_{s^*}, t \in B_{t^*}} \|s - t\|_2 / h$.

Proof. Let $x_{(k)}$ be the k^{th} coordinate of a point $x = \left(x_{(1)}, \ldots, x_{(d)}\right)$ and let $\mathbb{1} = (1, \ldots, 1)$ be the unit vector in $\mathbb{R}^d$. For a given source point s let $\widetilde{s} = \left(\left|s_{(1)} - s^*_{(1)}\right|, \ldots, \left|s_{(d)} - s^*_{(d)}\right|\right)$, equivalently for a target point t. First consider the error term $E^{H_p}_{B_{s^*}}(t_j)$ of the *Hermite Expansion*:

$$\begin{aligned}
E^{H_p}_{B_{s^*}}(t_j) &= \left| \sum_{s_i \in B_{s^*}} w_i \cdot e^{-\|t_j - s_i\|^2/h^2} - \sum_{|\alpha| < p} \frac{1}{\alpha!} \cdot \sum_{s_i \in B_{s^*}} w_i \cdot \left(\frac{s_i - s^*}{h}\right)^\alpha \cdot h_\alpha \left(\frac{t_j - s^*}{h}\right) \right| \\
&\leq \sum_{s_i \in B_{s^*}} |w_i| \left| e^{-\|t_j - s_i\|^2/h^2} - \sum_{|\alpha| < p} \frac{1}{\alpha!} \cdot \left(\frac{s_i - s^*}{h}\right)^\alpha \cdot h_\alpha \left(\frac{t_j - s^*}{h}\right) \right| \\
&\leq W_{s^*} \cdot \sum_{|\alpha| = p} \frac{1}{\alpha!} \cdot \max_{s \in B_{s^*}, t \in B_{t^*}} \left| h_\alpha \left(\frac{t - s}{h}\right) \right| \cdot \left| \left(\frac{s - s^*}{h}\right)^\alpha \right|,
\end{aligned}$$

where the last estimation uses the Lagrange remainder from Theorem (3.2). Applying the inequality by Szász (3.11) further yields

$$\begin{aligned}
E^{H_p}_{B_{s^*}}(t_j) &\leq W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \max_{s \in B_{s^*}} \sum_{|\alpha| = p} \frac{1}{\alpha!} \sqrt{2^{|\alpha|} \alpha!} \cdot \prod_{k=1}^{d} \left(\frac{\left|s_{(k)} - s^*_{(k)}\right|}{h} \right)^{\alpha_k} \\
&= W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \max_{s \in B_{s^*}} \sum_{|\alpha| = p} \frac{\sqrt{\alpha!}}{\alpha!} \left(\frac{\sqrt{2}\widetilde{s}}{h}\right)^\alpha \\
&\leq W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{1}{\sqrt{p!}} \cdot \max_{s \in B_{s^*}} \sum_{|\alpha| = p} \frac{p!}{\alpha!} \left(\frac{\sqrt{2}\widetilde{s}}{h}\right)^\alpha \cdot \mathbb{1}^\alpha \\
&= W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{1}{\sqrt{p!}} \cdot \max_{s \in B_{s^*}} \left(\left(\frac{\sqrt{2}\widetilde{s}}{h}\right) \cdot \mathbb{1} \right)^p \qquad \text{(by (3.3))} \\
&= W_{s^*} \cdot e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{1}{\sqrt{p!}} \cdot \left(\sqrt{2} r_{s^*}\right)^p.
\end{aligned}$$

The approximation error of the *Taylor Expansion* (4.28) can be derived in an analogical way. We finally present the derivation of the error bound for the *Taylor-Hermite Expansion* (4.29). First we can split the error by defining

$$E_1 := \sum_{|\beta| \geq p} \frac{(-1)^{|\beta|}}{\beta!} \cdot h_\beta \left(\frac{t^* - s_i}{h}\right) \cdot \left(\frac{t_j - t^*}{h}\right)^\beta,$$

$$E_2 := \sum_{|\beta|<p} \frac{(-1)^{|\beta|}}{\beta!} \cdot \sum_{|\alpha|\geq p} \frac{1}{\alpha!} \cdot \left(\frac{s_i - s^*}{h}\right)^{\alpha} \cdot h_{\alpha+\beta}\left(\frac{t^* - s^*}{h}\right) \cdot \left(\frac{t_j - t^*}{h}\right)^{\beta},$$

which leads to

$$\begin{aligned} E^{TH_p}_{B_{s^*}}(t_j) &= \left| G_{B_{s^*}}(t_j) - \sum_{|\beta|<p} \frac{(-1)^{|\beta|}}{\beta!} \cdot \sum_{|\alpha|<p} C^H_\alpha(B_{s^*}) \cdot h_{\alpha+\beta}\left(\frac{t^* - s^*}{h}\right) \cdot \left(\frac{t_j - t^*}{h}\right)^{\beta} \right| \\ &\leq W_{s^*} \cdot (|E_1| + |E_2|)\,. \end{aligned}$$

The first term can obviously be bounded by $|E_1| \leq e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{1}{\sqrt{p!}} \cdot \left(\sqrt{2} r_{t^*}\right)^p$ as shown above, while for the second term we get

$$\begin{aligned} |E_2| &\leq \sum_{|\beta|<p} \frac{1}{\beta!} \cdot \sum_{|\alpha|=p} \frac{1}{\alpha!} \cdot \max_{s\in B_{s^*}, t\in B_{t^*}} \left| h_{\alpha+\beta}\left(\frac{t-s}{h}\right)\right| \cdot \left|\left(\frac{s-s^*}{h}\right)^{\alpha}\right| \cdot \left|\left(\frac{t-t^*}{h}\right)^{\beta}\right| \\ &\leq e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \max_{s\in B_{s^*}, t\in B_{t^*}} \sum_{|\beta|<p} \frac{1}{\beta!} \cdot \sum_{|\alpha|=p} \frac{1}{\alpha!} \sqrt{2^{|\alpha+\beta|}(\alpha+\beta)!} \cdot \left|\left(\frac{s-s^*}{h}\right)^{\alpha}\right| \cdot \left|\left(\frac{t-t^*}{h}\right)^{\beta}\right| \\ &= e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \max_{s\in B_{s^*}, t\in B_{t^*}} \sum_{|\beta|<p} \frac{1}{\sqrt{\beta!}} \cdot \sum_{|\alpha|=p} \frac{1}{\sqrt{\alpha!}} \sqrt{\frac{(\alpha+\beta)!}{\alpha!\beta!}} \sqrt{2}^{|\alpha+\beta|} \cdot \left(\frac{\widetilde{s}}{h}\right)^{\alpha} \cdot \left(\frac{\widetilde{t}}{h}\right)^{\beta} \\ &\leq e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \max_{s\in B_{s^*}, t\in B_{t^*}} \sum_{|\beta|<p} \frac{1}{\sqrt{\beta!}} \cdot \sum_{|\alpha|=p} \frac{1}{\sqrt{\alpha!}} \sqrt{2}^{|\alpha+\beta|} \sqrt{2}^{|\alpha+\beta|} \cdot \left(\frac{\widetilde{s}}{h}\right)^{\alpha} \cdot \left(\frac{\widetilde{t}}{h}\right)^{\beta} \\ &= e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \max_{t\in B_{t^*}} \sum_{|\beta|<p} \frac{1}{\sqrt{\beta!}} \cdot \max_{s\in B_{s^*}} \sum_{|\alpha|=p} \frac{1}{\sqrt{\alpha!}} \cdot \left(\frac{2\widetilde{s}}{h}\right)^{\alpha} \cdot \left(\frac{2\widetilde{t}}{h}\right)^{\beta} \\ &\leq e^{-{\delta^{min}_{s^*t^*}}^2/2} \cdot \frac{1}{\sqrt{p!}} \cdot (2r_{s^*})^p \cdot \underbrace{\max_{t\in B_{t^*}} \sum_{|\beta|<p} \frac{1}{\sqrt{\beta!}} \cdot \left(\frac{2\widetilde{t}}{h}\right)^{\beta}}_{=:E_{22}} \cdot \end{aligned}$$

The factor E_{22} can be bounded by

$$E_{22} = \sum_{k=0}^{p-1} \max_{t\in B_{t^*}} \sum_{|\beta|=k} \frac{1}{\sqrt{\beta!}} \cdot \left(\frac{2\widetilde{t}}{h}\right)^{\beta} \leq \sum_{k=0}^{p-1} \frac{1}{\sqrt{k!}} \cdot (2r_{t^*})^k$$

and further either by a geometric series

$$E_{22} \leq \sum_{k=0}^{p-1} (2r_{t^*})^k = \frac{(2r_{t^*})^p - 1}{2r_{t^*} - 1}$$

or alternatively by the exponential function

$$E_{22} \leq \sum_{k=0}^{p-1} \frac{\sqrt{k!}}{k!} \cdot (2r_{t^*})^k \leq \sqrt{(p-1)!} \cdot \sum_{k=0}^{p-1} \frac{1}{k!} \cdot (2r_{t^*})^k \leq \sqrt{(p-1)!} \cdot e^{2r_{t^*}}.$$

Thus, we get the final bound

$$|E_2| \leq e^{-\delta_{s^*t^*}^{min\,2}/2} \cdot \frac{1}{\sqrt{p!}} \cdot (2r_{s^*})^p \cdot \min\left\{ \frac{(2r_{t^*})^p - 1}{2r_{t^*} - 1}, \sqrt{(p-1)!} \cdot e^{2r_{t^*}} \right\},$$

which concludes the proof.

□

We choose the bound for the *Hermite Expansion* to be compared with the original one. The first two factors of the bounds coincide, while in the last we have to estimate $r_{s^*}^{\infty} = \max_{s \in B_{s^*}} \|s - s^*\|_\infty / h$ against $r_{s^*} = \max_{s \in B_{s^*}} \|s - s^*\|_1 / h$. Since the norms satisfy $\|x\|_\infty \leq \|x\|_1 \leq d \cdot \|x\|_\infty$ for every $x \in \mathbb{R}^d$, for a fixed $r_{s^*}^{\infty}$ we can provide a lower and an upper bound for r_{s^*}. However typically (depending on the way the clusters are built), we will have a Euclidean distance bound for all $s \in B_{s^*}$, and therefore the relations $\|x\|_\infty \leq \|x\|_2$ and $\|x\|_1 \leq \sqrt{d} \cdot \|x\|_2$ provide a good estimate for the average case. For a fixed $r = r_{s^*}^{\infty}$ and fixed dimension d we thus compare the following functions:

$$\begin{aligned}
e(p) &= \frac{\binom{d+p-1}{d-1}}{\sqrt{\left(\lfloor \frac{p}{d} \rfloor !\right)^{d-p'} \left(\lceil \frac{p}{d} \rceil !\right)^{p'}}} \cdot \left(\sqrt{2}r\right)^p; \\
e_{min}(p) &= \frac{1}{\sqrt{p!}} \cdot \left(\sqrt{2}r\right)^p; \\
e_{med}(p) &= \frac{1}{\sqrt{p!}} \cdot \left(\sqrt{2}r \cdot \sqrt{d}\right)^p; \\
e_{max}(p) &= \frac{1}{\sqrt{p!}} \cdot \left(\sqrt{2}r \cdot d\right)^p;
\end{aligned}$$

where $p' = p \mod d$. Nota bene the bounds coincide for $d = 1$.

As can be seen from the graphs in Figure 4.12, the new bound is tighter than the original one in the average case, where $r_{s^*}^{\infty} = \sqrt{d} \cdot r_{s^*}$. If only an l_∞-estimation is provided and no l_1-estimation except for $r_{s^*} \leq d \cdot r_{s^*}^{\infty}$ is known, the original bound yields slightly better results.

4.3.3 Relative error control and implementation details of the DFGT

The error bounds presented above are absolute error bounds, while in most scenarios the user of a numerical algorithm will be interested in relative errors. The *Dual-Tree Fast Gauss Transform* offers a method to guarantee a pointwise bound for the relative error, under the assumption that all weights w_i $(i = 1, \ldots, N)$ are non-negative. This is the case in many applications, for example *Kernel Density Estimation* and numerical integration.

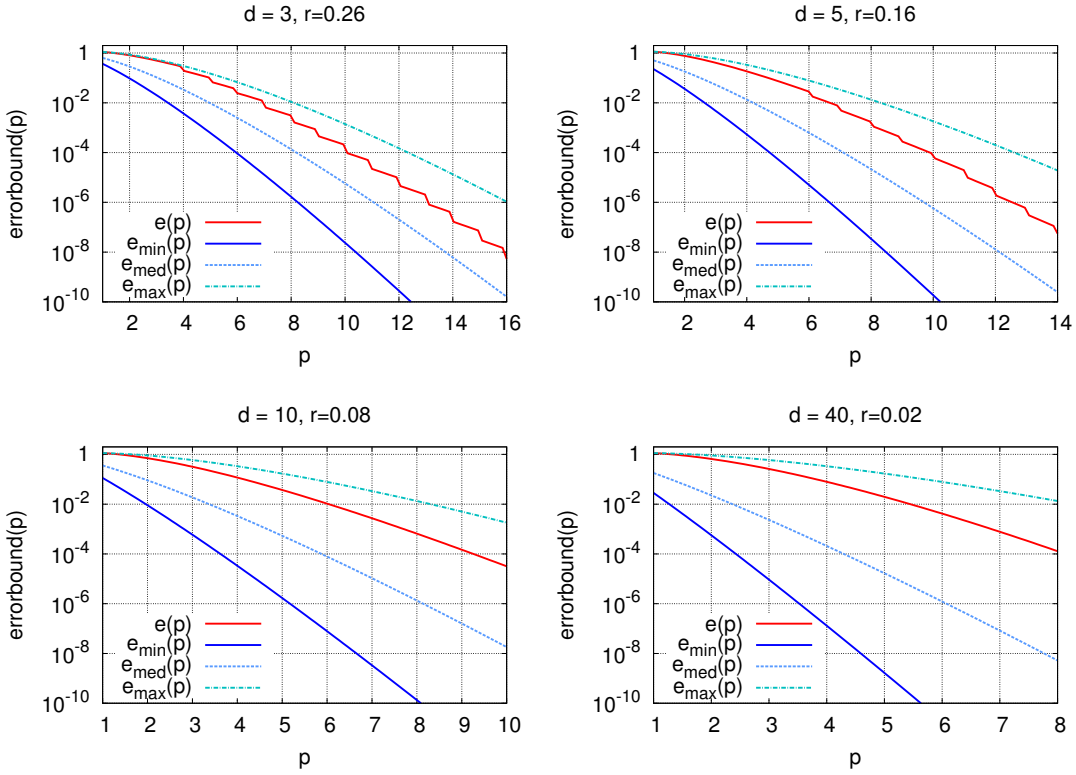


FIGURE 4.12: Comparison of the *DFGT* Hermite error bounds. For fixed dimension d the parameter r has been chosen such that $r \cdot d < 1$ in order to have convergence from the beginning and achieve usable values for small p.

Theorem 4.22. *Consider a set of methods $\mathbb{A}$ for approximating the contribution of a source cluster B_{s^*} to a single target $t_j \in B_{t^*}$, where $E^A_{B_{s^*}}(t_j)$ shall denote the maximum absolute error of method $A \in \mathbb{A}$. Then the pointwise relative error ϵ can be guaranteed, if $E^A_{B_{s^*}}(t_j) \leq \epsilon \cdot \frac{W_{s^*}}{W} G^{min}_{t^*}$ for all $A \in \mathbb{A}$, where $W_{s^*} = \sum_{s_i \in B_{s^*}} |w_i|$, $W = \sum_{i=1}^{N} |w_i|$ and $G^{min}_{t^*} \leq \min_{t \in B_{t^*}} G(t)$.*

Proof. For a given target t_j let $B_{s_i^*}$ $(i = 1, \ldots, k)$ be the source clusters contributing to the computed result $\widetilde{G}(t_j)$ and $A_i \in \mathbb{A}$ be the method used for cluster $B_{s_i^*}$. We can then estimate

$$\begin{aligned}\left|G(t_j) - \widetilde{G}(t_j)\right| &= \left|\sum_{i=1}^{k} G_{B_{s_i^*}}(t_j) - \widetilde{G}_{B_{s_i^*}}(t_j)\right| \leq \sum_{i=1}^{k} \left|G_{B_{s_i^*}}(t_j) - \widetilde{G}_{B_{s_i^*}}(t_j)\right| \\ &\leq \sum_{i=1}^{k} E^{A_i}_{B_{s_i^*}}(t_j) \leq \sum_{i=1}^{k} \epsilon \cdot \frac{W_{s_i^*}}{W} G^{min}_{t^*} \leq \epsilon \cdot G^{min}_{t^*} \leq \epsilon \cdot G(t_j).\end{aligned}$$

□

The approximation rule induced by this Theorem grants each source cluster a maximum relative error proportional to the sum of the weights of source points it contains, which seems

to be a well working strategy. The global error control is further optimized by two additional features.

First, in case *direct evaluation* is chosen as method for a certain cluster B_{s^*}, there is no absolute error and the sum of the corresponding weights W_{s^*} can be stored and used to allow greater errors in the contributions of other clusters. Second, a strategy the authors of [41] call finite-difference pruning is applied. For each target cluster B_{t^*}, the algorithm holds and continuously updates the two values $G_{t^*}^{min}$ and $G_{t^*}^{max}$, that fulfill the relation $G_{t^*}^{min} \leq G(t_j) \leq G_{t^*}^{max}$ for all $t_j \in B_{t^*}$. $G_{t^*}^{min}$ is initialized with 0, which can be done since all weights are non-negative, $G_{t^*}^{max}$ is initialized with $\sum_{i=1}^{N} w_i$, the maximum possible value for any target t_j. As mentioned above another value denoted by $W_{t^*}^{save}$ is stored collecting the source weights w_i that have not yet contributed to the error $E(t_j)$ ($t_j \in B_{t^*}$) and can therefore still be used to save computational time. Now finite-difference pruning first tries to approximate the contribution of source cluster B_{s^*} to any target t_j in target cluster B_{t^*} without even looking at the individual sources and targets by taking the rough approximation

$$G_{B_{s^*}}(t_j) \approx \frac{G_{s^*t^*}^{max} + G_{s^*t^*}^{min}}{2}, \tag{4.32}$$

where $G_{s^*t^*}^{max} = W_{s^*} \cdot e^{-\delta_{s^*t^*}^{min\,2}}$, $G_{s^*t^*}^{min} = W_{s^*} \cdot e^{-\delta_{s^*t^*}^{max\,2}}$ and $\delta_{s^*t^*}^{min} = \min_{s \in B_{s^*}, t \in B_{t^*}} \|s-t\|_2/h$, $\delta_{s^*t^*}^{max} = \max_{s \in B_{s^*}, t \in B_{t^*}} \|s-t\|_2/h$.

Be aware that $\delta_{s^*t^*}^{min}$ and $\delta_{s^*t^*}^{max}$ are *not* determined exactly by calculating all distances between source and target points, but rather approximately by using properties of the corresponding clusters. Thus, the calculation costs are independent of the number of points contained in the clusters.

The maximum absolute error of the contribution of source cluster B_{s^*} to any target t_j can now trivially be bounded by $\frac{|G_{s^*t^*}^{max} - G_{s^*t^*}^{min}|}{2}$, since $G_{s^*t^*}^{min} \leq G(t_j) \leq G_{s^*t^*}^{max}$ for all $t_j \in B_{t^*}$. Thus Theorem 4.22 yields the condition

$$\frac{\left|G_{s^*t^*}^{max} - G_{s^*t^*}^{min}\right|}{2} \leq \epsilon \cdot \frac{W_{s^*}}{W} G_{t^*}^{min}, \tag{4.33}$$

to use approximation (4.32). If condition (4.33) is not fulfilled, the term $W_{t^*}^{save}$ is used to modify the approximation condition to

$$\frac{\left|G_{s^*t^*}^{max} - G_{s^*t^*}^{min}\right|}{2} \leq \epsilon \cdot \frac{W_{s^*} + W_{t^*}^{save}}{W} G_{t^*}^{min}. \tag{4.34}$$

Solving for $W_{t^*}^{save}$ yields

$$W_{t^*}^{save} \geq \frac{W \cdot \left|G_{s^*t^*}^{max} - G_{s^*t^*}^{min}\right|}{2\epsilon \cdot G_{t^*}^{min}} - W_{s^*}. \tag{4.35}$$

When using this pruning condition, we have to decrement the term $W_{t^*}^{save}$ by the value of the right hand side of (4.34) afterwards. In order to minimize computational effort, we do not instantly add the estimated value to all Gaussians, but rather store it in another variable $G_{t^*}^{est}$ and propagate it downwards to the leaves and further to the individual targets in a post-processing step. During this pass, Taylor Coefficients are shifted using the *Local-To-Local*

Operator (4.31) from parent to their child nodes until arrived at the leaves, where the evaluation of the Taylor series at every target takes place.

In [41] the authors also suggest to pre-compute the *Hermite Coefficients* up to a certain order using the Hermite-to-Hermite Operator (4.30). However the suggestions must be considered with care, since useful bounds for the order can only be guessed and for certain data sets and parameters, neither the *Hermite Expansion* nor the *Taylor-Hermite Expansion* might be used at all making the pre-computation superfluous. The Hermite-to-Hermite Operator can still be used to shift Hermite Coefficients from child nodes to their parent nodes on the fly while traversing the tree.

The main part of the *DFGT*-algorithm consists of the recursive procedure `cc(S,T)` (cc being an abbreviation for "collect contributions"), which operates on a source node `S` and a target node `T` and computes the approximate contribution of the current source cluster to all points in the current target cluster. It basically performs the following steps:

1. If source node `S` is far enough away from `T`, finite difference pruning is applied.
2. Otherwise, if the costs for any of the three approximation methods *Hermite* (4.27), *Taylor* (4.28), *Taylor-Hermite Expansion* (4.29) is lower than costs for direct evaluation, choose the cheapest one for evaluation.
3. If direct evaluation is cheapest, do not perform any calculation right away, but descend in the tree to calculate interactions for children instead.

Once the recursion arrives at a leaf (without having found any cheap approximation method), it drops back to direct evaluation. Thus, in the worst case, the algorithm performs direct evaluation for all data points in addition to traversing the whole tree.

We have not been talking about the details of a suitable tree structure. In [41], Lee and Gray use a sphere-rectangle tree (SR-Tree) [38] without considering any alternatives. We will therefore leave this discussion to the next section, assuming for now we are dealing with any kind of binary tree.

Algorithm: *Dual-Tree Fast Gauss Transform* [41]
Input: N sources s_i, $(i = 1, \ldots, N)$ in $\mathbb{R}^d$
M targets t_j, $(j = 1, \ldots, M)$ in $\mathbb{R}^d$
N weights w_i, $(i = 1, \ldots, N)$ in $[0, \infty)$
bandwidth $h > 0$, error bound $\epsilon > 0$
Output: $\widetilde{G}_{DFGT}(t_j)$, $(j = 1, \ldots, M)$, with $\left|\widetilde{G}_{DFGT}(t_j) - G(t_j)\right| < \epsilon \cdot \left|G(t_j)\right|$

```
Build a (binary) source tree R_S using all N sources;
build a (binary) target tree R_T using all M targets.

For each node B_{t*} of the target tree R_T initialize
  G^min_{t*} = 0;  G^max_{t*} = sum_{i=1}^{N} w_i;  W^save_{t*} = 0.
```

```
For each target t_j initialize
  G(t_j) = 0; G^min_{t_j} = 0; G^max_{t_j} = Σ_{i=1}^N w_i.

cc(root(R_S), root(R_T))

Traverse the target tree R_T via a breadth-first search and
  shift Taylor Coefficients from parent to their child nodes
  using the Local-To-Local Operator (4.31).
  When arrived at a leaf, evaluate Taylor series at each target t_j.
```

```
PROCEDURE cc(source node S, target node T)

  B_{s*} = source cluster of node S with expansion center s*
  B_{t*} = target cluster of node T with expansion center t*
  δ^min_{s*t*} = minimum distance between B_{s*} and B_{t*}
  δ^max_{s*t*} = maximum distance between B_{s*} and B_{t*}

  G^max_{s*t*} = W_{s*} · e^{-δ^min_{s*t*}^2}
  G^min_{s*t*} = W_{s*} · e^{-δ^max_{s*t*}^2}
  W^test = W · |G^max_{s*t*} − G^min_{s*t*}| / (2ε · G^min_{t*}) − W_{s*}

  // Finite-Difference Pruning
  IF W^save_{t*} ≥ W^test
    W^save_{t*} -= W^test
    G^min_{t*} += G^min_{s*t*}
    G^max_{t*} += G^max_{s*t*}
    G^est_{t*} += 0.5 · (G^min_{s*t*} + G^max_{s*t*})
  ELSE // Fast Gauss methods
    Choose cheapest method A from Hermite Expansion, Taylor Expansion,
      Taylor-Hermite Expansion (Def. 4.17) and Direct Evaluation
      using Lemma 4.21 and Theorem 4.22.
    IF A == Hermite Expansion
      For all t_j ∈ B_{t*}
        Compute Hermite Expansion (4.27) due to source cluster B_{s*}.
    ELSE IF A == Taylor Expansion
      Compute Taylor Coefficients C^T_α(B_{s*}, t*) via (4.28).
    ELSE IF A == Taylor-Hermite Expansion
      Compute Taylor-Hermite Coefficients C^{TH_p}_β(B_{s*}, t*) via (4.29).
    IF A ≠ Direct Evaluation
      W^save_{t*} += W_{s*} − W · E^A / (ε · G^min_{t*})
      G^min_{t*} += G^min_{s*t*}
      G^max_{t*} += G^max_{s*t*}
    ELSE
      IF leaf(S) OR leaf(T)
        For all s_i ∈ B_{s*}, t_j ∈ B_{t*}
```

```
            c = w_i · e^{-‖t_j − s_i‖²/h²}
            G(t_j) += c
            G_{t_j}^{min} += c
            G_{t_j}^{max} += c - w_i
        W_{t*}^{save} += W_{s*}
        G_{t*}^{min} = min_{t_j ∈ B_{t*}} G_{t_j}^{min}
        G_{t*}^{max} = max_{t_j ∈ B_{t*}} G_{t_j}^{max}
      ELSE
        cc(leftChild(S), leftChild(T))
        cc(leftChild(S), rightChild(T))
        cc(rightChild(S), leftChild(T))
        cc(rightChild(S), rightChild(T))
```

4.3.4 Runtime analysis

In order to estimate the runtime of the *Dual-Tree Fast Gauss Transform*, note that for a balanced binary tree, the construction time complexity is of $\mathcal{O}(N \log_2 N)$ for N data points. Since in our case, all data – that is sources and targets – are given a priori, we can assume to deal with a well-balanced tree. Including distance calculations, we get a runtime bound of $\mathcal{O}\left(d \cdot (N \log_2 N + M \log_2 M)\right)$ for tree construction.

As mentioned above, the worst case for the recursive procedure occurs, if the whole tree is traversed without any fast method being applied, such that direct evaluation is carried out for all points at leaf level. In this case, the runtime adds up to $\mathcal{O}\left(d \cdot (N \log_2 N + M \log_2 M + N \cdot M)\right)$. However, since the algorithm compares the costs for every combination of source and target nodes it arrives at, and l_1-expansions with $\binom{p_{max}-1+d}{d}$ terms are used if advantageous, we obtain the following runtime bound:

Runtime Complexity 4.23. *The* DFGT *has a runtime of*

$$\mathcal{O}\left(d \cdot (N \log_2 N + M \log_2 M) + \min\left(d \cdot N \cdot M, (N+M) \cdot \binom{p_{max}-1+d}{d}\right)\right).$$

The *Dual-Tree Fast Gauss Transform* is the first algorithm, that efficiently combines fast expansion techniques for the Gaussian Kernel with a hierarchical data structure. It does not suffer from the intrinsic performance problem of the *FGT* and *IFGT* dealing with low bandwidth applications. However it still leaves two important open questions:

- How does the expansion of the *Improved Fast Gauss Transform* perform compared to those of the *Fast Gauss Transform* in combination with dual trees?
- What is a good or the best possible choice for the tree structure?

In order to answer these questions and discuss some further optimization techniques, we introduce two new algorithms called the *Dual-Tree* and *Single-Tree Improved Fast Gauss Transform*.

4.4 Dual-Tree Improved Fast Gauss Transform

Data structure: Two tree structures (VAM-Split SR-Tree) for sources and targets

Expansions: *IFGT-Taylor Expansion* (l_1-version)

Guaranteed global error: $\left|\widetilde{G}_{DFGT}(t_j) - G(t_j)\right| < \epsilon \cdot \left|G(t_j)\right|$ for a fixed bound $\epsilon > 0$

Runtime complexity: $\mathcal{O}\left(\min\left(dNM, (N+M)\cdot\binom{p_{max}-1+d}{d}\right)\right) +$
$\mathcal{O}\left(d\left(N\log_2 N + M\log_2 M\right)\right)$, where p_{max} is a truncation parameter for series expansion.

The *Dual-Tree Improved Fast Gauss Transform (DIFGT)* is mainly based upon the *Dual-Tree Fast Gauss Transform* presented above. We will therefore not repeat the principal structure of the algorithm but rather focus on the differences between the *DIFGT* and *DFGT* beginning with a short discussion of the tree structure.

4.4.1 The Tree structure - combining the SR-tree and VAM-Split tree

In the previous section we have not been talking about the precise implementation of the tree structure for the source and target tree. Lee and Gray [41] use Sphere-Rectangle (SR-)Trees [38] that have been proven to perform much better than other dynamic index structures such as K-D-B-Trees [52], R*-trees [7] and SS-trees [66], especially in high dimensions. We now throw a short glance at the different trees. For a detailed description, the reader is referred to the mentioned articles.

Trees are often used to process a huge amount of data changing over time. Therefore most tree structures provide routines for efficient insertion and removal of data points. These routines have to take care of changes in the affected nodes and rebuild parts of the tree with moderate computational effort. We call this kind of tree a *dynamic* index structure – in contrast to a *static* index structure, where all data points are given a priori and do not change while the tree is used.

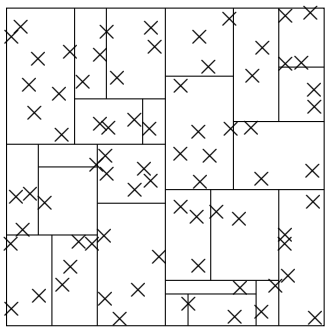


Figure 4.13: K-D-B-tree: space partitioning parallel to the coordinate axes.

The K-D-B-tree is a dynamic index structure using coordinate planes to divide the given search space into disjoint subareas. The main advantage of the disjointness among nodes of the same level is the unique path from the tree root to a leaf for a given data point. However the propagation of plane splits from a node to its children can induce the creation of empty or hardly populated nodes, a fact the performance of K-D-B-trees suffers from.

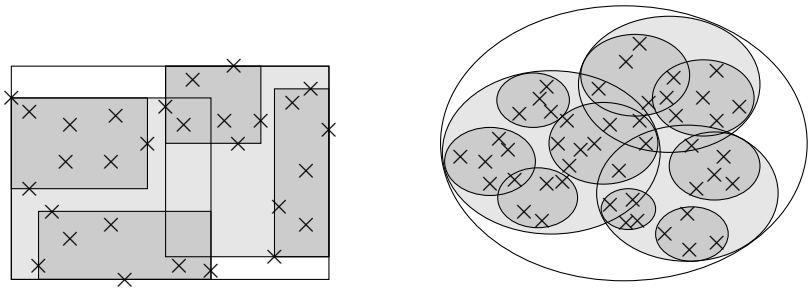


FIGURE 4.14: Overlapping nodes of an R*-tree *(left)* and SS-tree *(right)*. Darker hue symbolizes a deeper level.

R*-trees are an optimized variant of R-trees [32], whose nodes correspond to rectangles. The boundaries of a node are determined by the smallest rectangle comprising all child rectangles. In order to allow easier insertions afterwards an overlap of nodes on the same level is permitted. The SS-tree has been developed to further improve the performance of the R*-tree and utilizes bounding spheres (that can actually have an elliptic shape) instead of rectangles.

Finally the key idea of the SR-tree is to employ the intersection of a bounding sphere and a bounding rectangle as search region. This concept derives from practical experiences in tree construction made with high-dimensional data sets [38] that have shown that while the average volume of bounding rectangles turns out to be considerably smaller than that of bounding spheres, the latter come up with shorter average diameters than the former. SR-trees pledge to combine the advantages of both shapes, additionally improving the disjointness among regions. In [38] KATAYAMA and SATOH show that the SR-tree – while suffering from higher creation costs and storage requirements due to the combination of two shapes – indeed outperforms all previous variants regarding nearest neighbor queries.

All indexing structures considered so far are dynamic structures in the sense explained above. However, in our problem statement there is no necessity to deal with insertions or deletions of source or target points at a later time, the complete data set is known from the beginning. On the other hand, a static structure is easier to implement and can take advantage of the full knowledge of all data in the construction process. We therefore favor the utilization of a static structure.

The VAM-Split R-tree was introduced by WHITE and JAIN in [65]. The tree is constructed using the following simple rules:

1. The root of the tree contains all data.
2. A node is split recursively into two children by the following process:
   a. For all points in the node find the dimension with maximum variance in the coordinates.

b. Choose a value that is approximately a median of these coordinates to divide the points in two sets.

A real number m is called median of a set of real numbers A, if for $B = \{a \in A \mid a < m\}$ we have $\left|\overline{B} - \overline{A \setminus B}\right| \leq 1$. Simply spoken, a median splits a set into two equally sized parts (if the set size is even, otherwise the parts' cardinality only differs by 1).

The reason the VAM-Split (variance – approximately – median) R-tree does not use an exact median split, but rather an approximate, is given by the fact that the authors adjust the node size to disk blocks in order to minimize disk access costs. We do not follow this approach for two reasons. First, the disk block size may be unknown and differ on different platforms. Second, performing an exact median split results in a well-balanced binary tree, which provides fast and easy access routines.

In [38], the comparison of several dynamic index structures with the VAM-Split R-tree shows a superior performance of both the SR-tree and the VAM-Split R-tree, while in most cases the SR-tree is performing slightly better. In order to benefit from both their advantages, we combine the two trees calling the result an VAM-Split SR-tree. For that purpose, in a first top-down step, a VAM-Split R-tree is constructed (choosing the exact median) as described above. In a second bottom-up step, we calculate bounding rectangles and bounding spheres for all nodes. These bounding objects are later used for distance calculations.

In order to be able to handle huge data sets, we do not copy and store the data points within the tree nodes. The points are rather re-sorted in a single array during the splitting process and every node only keeps the index of its first point in the array and its size. A node of the VAM-Split SR-tree consists of the following entries:

- `index`: index of first point in data array;
- `size`: number of points;
- `boundL(1..d), boundR(1..d)`: bounds of bounding rectangle;
- `center(1..d)`: center point of bounding sphere;
- `radius`: radius of bounding sphere.

The creation costs of the tree are summarized in the following scheme:

1. The variance calculation can be carried out in linear time and must be done on each level for all points and all dimensions resulting in an approximate runtime of $\mathcal{O}(d \cdot N \log_2(N))$.
2. The calculation of the median can also be performed in linear time (a good approach can be found in [46]) and must be done on each level for all points resulting in an approximate runtime of $\mathcal{O}(N \log_2(N))$.
3. The calculations of all bounding object variables have again linear time complexity on each level resulting in a runtime of $\mathcal{O}(d \cdot N \log_2(N))$.

Finally we remark that the creation of new children is not repeated recursively until only a single point is left in each leaf. In fact it terminates as soon as a certain threshold for the minimum number of points is hit, in order to achieve the best possible performance. In our computational setup we found values between 10 and 20 to be a good choice.

4.4.2 Expansion and implementation details

As the naming of the algorithm already suggests, it uses the series expansion introduced by the *Improved Fast Gauss Transform* rather than those of the *Fast Gauss Transform*. For the reader's convenience we repeat the according expansion:

Definition 4.24 (IFGT Expansion)**.** *Let the same assumptions apply as in Definition 4.1. Further let Z_n be a source cluster with center s_n^* and t_j be any target point. Let $p \geq 1$ be an arbitrary, but fixed integer. We then define the*

- ***IFGT-Taylor Coefficient*** *(due to source cluster Z_n)*

$$C_\alpha(Z_n) = \frac{2^{|\alpha|}}{\alpha!} \sum_{s_i \in Z_n} w_i \cdot e^{-\|s_i - s_n^*\|^2/h^2} \left(\frac{s_i - s_n^*}{h}\right)^\alpha; \tag{4.36}$$

- ***IFGT-Taylor Expansion (l_1-version)*** *(due to target t_j and source cluster Z_n)*

$$\widetilde{G}_{Z_n}^{IT_p}(t_j) = \sum_{|\alpha|<p} C_\alpha(Z_n) \cdot e^{-\|t_j - s_n^*\|^2/h^2} \left(\frac{t_j - s_n^*}{h}\right)^\alpha. \tag{4.37}$$

There are several reasons why we prefer the *IFGT* expansion over the three *FGT* expansions. First, in practice the *IFGT* has proven to perform very well in high bandwidth setups. Since the Dual-Tree algorithm – providing a good solution to the nearest neighbor problem as explained above – already offers good performance for low bandwidth, we expect to get an algorithm with good performance on a wide bandwidth range. Second, the fact of using a single expansion rather than three allows us to re-use the calculated Taylor coefficients whenever the same source cluster is involved in a series expansion twice. Finally, cost estimations and the decision, if further descent in the tree is helpful or not, are accelerated and simplified. This is actually a crucial aspect in the algorithm, where most of the optimizations can be carried out.

Remember the *DFGT* chooses a fast method as soon as its costs are lower than those of the direct evaluation. This can however be a suboptimal strategy, since on a deeper tree level, costs can possibly be saved by ignoring far clusters and using Taylor expansions in smaller regions with smaller Taylor bounds. Thus, whenever a Taylor expansion is faster than direct evaluation, we first check the costs and Taylor bounds of the children and – for promising values – descent deeper in the tree.

As explained in the previous section, the Dual-Tree algorithm keeps and updates lower bounds $G_{t^*}^{min}$ for all Gaussians of a target cluster B_{t^*} with cluster center t^*. After the construction of the tree, these bounds are initialized in a bottom-up way by calculating the direct evaluation for every target due to all sources in the nearest leaf of the source cluster tree. Since the number of sources in a leaf is bounded by a small constant, this step only requires $\mathcal{O}(d \cdot M)$ time. This pre-processing step speeds up the calculation especially in case of low bandwidth, where only the very nearest sources are required to meet the error bound.

For a formal description of the algorithm, the reader is referred to the previous section, since the basic structure of the algorithms is quite similar.

4.4.3 Runtime analysis

As shown above, the construction of a VAM-Split SR-tree with N points can be carried out in $\mathcal{O}(d \cdot N \log_2(N))$ time. Since both a tree for sources and targets are utilized, the total runtime adds up to $\mathcal{O}(d \cdot (N \log_2(N) + M \log_2(M))$.

The runtime bound for the recursive main part can be derived in a equivalent way as done for the *Dual-Tree FGT*. Thus we get the same total time complexity.

Runtime Complexity 4.25. *The* DIFGT *has a runtime of*

$$\mathcal{O}\left(d \cdot (N \log_2 N + M \log_2 M) + \min\left(d \cdot N \cdot M, (N+M) \cdot \binom{p_{max} - 1 + d}{d}\right)\right).$$

4.5 Single-Tree Improved Fast Gauss Transform

Data structure: A single tree (VAM-Split SR-Tree with PCA) for source points, which coincide with target points (thus $N = M$)

Expansions: *IFGT-Taylor Expansion* (l_1-version)

Guaranteed global error: $\left|\widetilde{G}_{DFGT}(t_j) - G(t_j)\right| < \epsilon \cdot \left|G(t_j)\right|$ for a fixed bound $\epsilon > 0$

Runtime complexity: $\mathcal{O}\left(\min\left(dN^2, 2N \cdot \binom{p_{max}-1+d}{d}\right)\right) + \mathcal{O}\left(N \cdot \left(d \cdot log_2 N + d^3\right)\right)$,

where p_{max} is a truncation parameter for series expansion.

With the *Single-Tree Improved Fast Gauss Transform (SIFGT)* we extend the idea of the *Dual-Tree Improved Fast Gauss Transform* using a different tree structure to yield better performance for certain high-dimensional datasets. For reasons that will become clear later, the improvements can only be applied, if sources and targets coincide and thus can be stored in a single tree. In order to introduce the changes in the tree structure, we first give a short overview on the dimension reduction problem and the principal component analysis.

4.5.1 Dimension reduction and the principal component analysis

Many applications dealing with high-dimensional datasets suffer from a high workload due to an exponential dependence on the dimension. This fact is also known as the curse of dimensionality. However various investigations on real-world data have shown that many datasets situated in high-dimensional space actually have low-dimensional features – that means, some parts of the data are situated on low-dimensional submanifolds. The term "effective dimension" of a dataset has been introduced and defined in different ways. An obvious case of a dataset in a d-dimensional space with effective dimension $k < d$ are points taken from a k-dimensional submanifold. However, since data is often tainted with errors, a dataset is also said to be of effective dimension k, if the other $d - k$ directions have a variance in coordinates smaller than some given bound. Several methods for dimension reduction and the determination of

the effective dimension have been investigated. Introductions to this topic can be found in [10], [19] and [12]. A well-known technique is the principal component analysis (PCA), also called Karhunen-Loève transform or Hotelling transform in particular contexts. The PCA tries to reduce the dimension by finding the direction with maximum variance in data and a corresponding orthogonal system. Directions with variance lower than some bound can then be ignored.

For a given set of points $x_k = \left(x_k^1, \ldots, x_k^d\right) \in \mathbb{R}^d$, $k = 1, \ldots, N$, the PCA carries out the following steps:

1. Calculate the centroid $m = \frac{1}{N} \sum_{k=1}^{N} x_k$ of all points and shift them to get new points $\widetilde{x}_k = x_k - m$ with centroid 0.

2. Set up the $(d \times d)$-covariance matrix $C = (c_{ij})$, $c_{ij} = \sum_{k=1}^{N} \widetilde{x}_k^i \cdot \widetilde{x}_k^j$ for the shifted points.

3. Find the eigenvalues and corresponding eigenvectors of the symmetric matrix C.

The i^{th} eigenvalue λ_i corresponds to the variance of the data in direction of the i^{th} eigenvector e_i. Assuming the eigenvalues are ordered such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$, the direction with greatest variance is given by e_1 and called the principal component.

The dimension can now be reduced by projecting the data on the given eigenvectors and ignoring coordinates with low variance.

However, the PCA can of course only discover global low-dimensional features, since it adds up the coordinates of all the data without considering local attributes. The effect can be seen in Figure 4.15. While the 1-dimensional nature of the diagonal can still be discovered in the left example, even though the data is heavily tainted with errors, the PCA of the circle does not distinguish a particular direction and could therefore also be derived from any uniformly distributed 2-dimensional dataset.

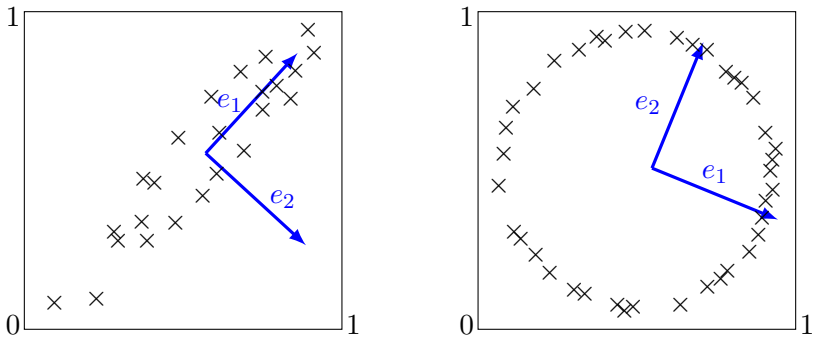


FIGURE 4.15: *Left*: The 1-dimensional structure in the data can be recognized by the principal component e_1 having an eigenvalue $\lambda_1 = 2.547$ much greater than $\lambda_2 = 0.09052$. *Right*: For a circle, the eigenvalues nearly coincide, the local 1-dimensional features cannot be discovered by the PCA ($\lambda_1 = 3.556$, $\lambda_2 = 3.733$).

4.5.2 The tree structure

Since our approach of combining the PCA with a space partitioning tree structure analyzes local subsets of the dataset, we hope to also take advantage of local low-dimensional features. A simple variant of this technique has already been applied by Sproull in [59] in context of nearest neighbor searching. In order to separate the two children of a node in a binary tree Sproull suggests the use of an arbitrary partition hyperplane instead of a plane parallel to a coordinate axis. The hyperplane in d-dimensional space is chosen to be the $(d-1)$-dimensional subspace orthogonal to the principal component of the data in the current node. Thus, the split is performed orthogonal to the direction of maximum variance in the data.

Our variant chooses the same hyperplane to perform the split and additionally calculates a minimum bounding hyperrectangle in the new coordinate system determined by the PCA. A first advantage can be clearly seen in Figure 4.16. Given two point sets S and T with low-dimensional features that are not parallel to the coordinate axes, the distance of the corresponding PCA rectangles can be much closer to the real distance $min_{s\in S} min_{t\in T} \|s-t\|$ than the distance of rectangles parallel to the coordinate axes. In Figure 4.16, the distance between rectangles B_1 and B_3 is much greater (and thus gives a better estimate) than the distance between rectangles A_1 and A_3. Second, following the effective dimension paradigm, rectangle sides with a length smaller than some constant do not have to be stored, thus reducing the dimension of the hyperrectangle from d to the effective dimension d_{eff} of the node data.

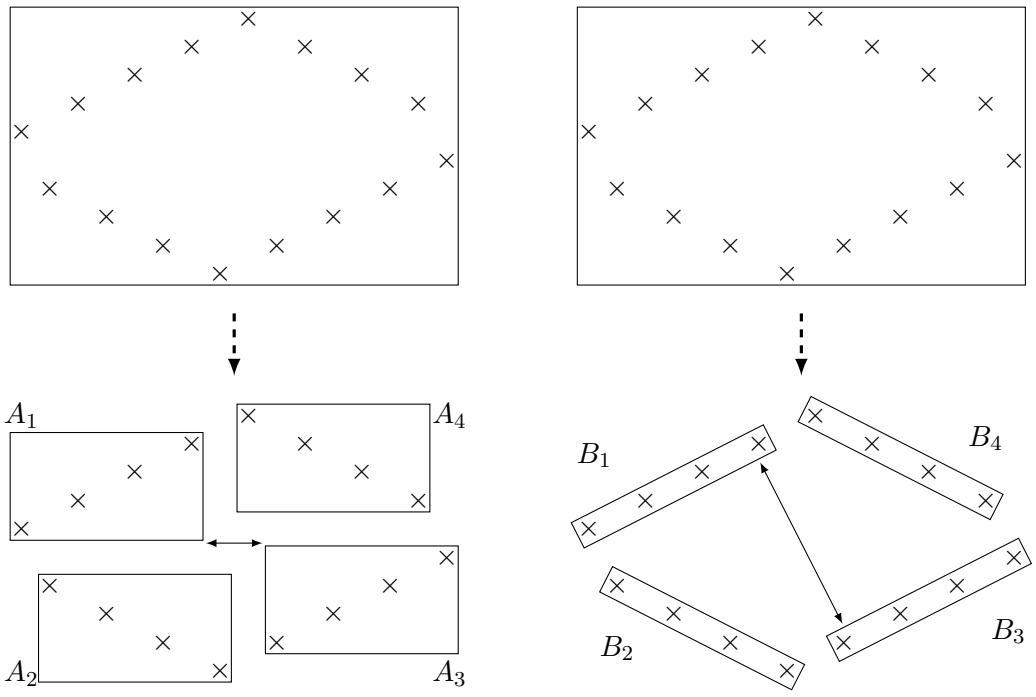


Figure 4.16: Rectangles parallel to the coordinate system *(left)* vs. rectangles determined by the PCA *(right)*.

However, the direct approach of calculating distances between arbitrarily rotated hyperrect-

angles is quite expensive in high dimensions. The two most well-known and intensely examined algorithms for the computation of distances between convex polytopes, the GILBERT-JOHNSON-KEERTHI (GJK) [23] and the LIN-CANNY (LC) algorithm [43], both originate from problems in robotics and are therefore designed to deal with 3-dimensional problems. Their time complexity is essentially linear to the total number of vertices of both polytopes. Since the number of vertices of a d-dimensional hyperrectangle is 2^d, a moderate runtime cannot be guaranteed for high dimension. We experimented with a different algorithm for distance calculation between hyperrectangles, that only requires one rotation ($\mathcal{O}(d^2)$) and another d minimization steps to find the distance. However even the modest costs of $\mathcal{O}(d^2)$ turned out to be too much to provide better performance due to more accurate distance estimations.

We therefore apply another strategy. Remember each node has its own "eigendirections" determined by the PCA. For a given node X, we calculate several bounding hyperrectangles using the eigendirections of its ancestors (father, grandfather, etc.). Whenever a distance between X and another node Y is required, we find their youngest common ancestor Z and use the appropriate hyperrectangles to compute the distance. Thus, no rotation is necessary and the distance calculation can be carried out in time $\mathcal{O}(d_{\text{eff}})$, where d_{eff} is the effective dimension of node Z.

Since the improved distance estimations induced by the PCA hyperrectangles have the biggest effect for nodes close to each other and most distance calculations are carried out between nodes close to each other anyway, it is sufficient to store bounding PCA hyperrectangles for only two or three generations of ancestors; in case no adequate hyperrectangle exists, the trivial bounding boxes parallel to the coordinate axes are used.

The fact that we are using a common ancestor to compute the distance between two nodes also supplies the reason, why this algorithm only makes sense in case the targets coincide with the sources. If we had two trees, a rotation of the PCA hyperrectangles would be necessary, which is not efficient. We experimented with a single tree that stores both sources and targets, but found this variant to be clearly inferior to the usual dual-tree approach.

4.5.3 Runtime analysis

When using the *Single Tree* algorithm, one has to keep in mind, that only small performance improvements due to faster and more precise distance calculations can be expected for data with low-dimensional features. On the other hand, the construction of the tree is much more expensive than the construction of the VAM-Split SR-Tree, since the PCA requires a singular value decomposition. This is realized by transforming the given symmetric $(d \times d)$-matrix to a tridiagonal matrix using the Householder reduction before applying the QL-algorithm with implicit shifts to determine the eigenvalues and corresponding eigenvectors. Both algorithms (see [60] for the theoretical background and [46] for efficient implementations) have a workload of $\mathcal{O}(d^3)$ and are applied to every node in the tree. Since we are dealing with a balanced binary tree and the leaves contain more than 1 point each, we can estimate the tree height by $h < \log_2 N$ and therefore the total number of nodes $k = 2^{h+1} - 1$ is bounded by N. The total runtime for tree construction thus adds up to $\mathcal{O}\left(N \cdot \left(d \cdot log_2 N + d^3\right)\right)$. This leads to the following result:

Runtime Complexity 4.26. *The* SIFGT *has a runtime of*

$$\mathcal{O}\left(N \cdot \left(d \cdot log_2 N + d^3\right)\right) + \min\left(dN^2, 2N \cdot \binom{p_{max} - 1 + d}{d}\right)\right).$$

5 Numerical Results

5.1 The Implementations

Before presenting numerical results of the different algorithms, we give a concise description of our validation environment as well as an overview on the main features of the implementations.

All simulations are performed on a computer with an *Intel Pentium D 3.20GHz* Dual-Core CPU (1024KB L2-cache per core) and 1024MB RAM under `Ubuntu 6.06` using the `GNU g++ 4.2.0` compiler with the optimization parameters `-O6 -march=nocona`.

We implemented several algorithms, most of them as presented in the previous chapter:

- DIRECT: The direct Gauss Transform that evaluates the equation

$$G(t_j) = \sum_{i=1}^{N} w_i \cdot e^{-\|t_j - s_i\|^2/h^2}$$

  for all targets t_j $(j = 1, \ldots, M)$ in the trivial way and therefore requires $\mathcal{O}(N \cdot M \cdot d)$ operations. Source and target points are stored in a simple two-dimensional, weights in a one-dimensional array. No additional optimizations concerning the data organization are carried out.

- DT DIRECT: The direct Gauss Transform in combination with the dual VAM-Split SR-Tree structure described in section 4.4. Sources and targets are stored in the source and target tree, respectively. Then the recursive procedure of the dual tree algorithm is called and descends in the tree – without performing any expansions or approximations – right till leaf level, where the full direct Gauss Transform is evaluated. The algorithm supplies an exact solution to the problem and demonstrates the superiority of efficient pre-processing and data access methods.

- FGT: The *Fast Gauss Transform* as described in section 4.1. According to the authors' recommendation to chose the cutoff parameters C_s, C_t as $\mathcal{O}(p^d)$ and after performing multiple trial runs with different values, we let $C_s = C_t = 10 \cdot p^d$. Furthermore, the calculation is aborted whenever the number of boxes n_{side}^d exceeds the constant $c = 100 \cdot \max(N, M)$, since in this case, the uniform space partitioning is obviously inefficient and the runtime will be much higher than that of the direct version.

- IFGT: The *Improved Fast Gauss Transform* as described in section 4.2. In order to choose a good value for the number of clusters K, we follow the approach described in subsection 4.2.4. If the costs for direct evaluation are lower than the costs with the optimal number of clusters, the calculation is aborted.

- DFGT: The *Dual-Tree Fast Gauss Transform* as described in section 4.3 using the VAM-Split SR-Tree structure and all optimizations described in section 4.4. Thus, the difference between using the *FGT* expansions (DFGT) and the *IFGT* expansion (DIFGT) can be worked out clearly.
- DIFGT: The *Dual-Tree Improved Fast Gauss Transform* developed in section 4.4.
- SIFGT: The *Single-Tree Improved Fast Gauss Transform* developed in section 4.5.

We perform tests with different setups and varying parameters. The main attributes are the dimension d, the number of source points N and the bandwidth h. If nothing else is specified, the target points coincide with the source points, which implies $M = N$, and the weights w_i are all set to 1.

For the approximation algorithms, the following error bounds are applied:

- FGT, IFGT: $\left|\widetilde{G}(t_j) - G(t_j)\right| < \epsilon \cdot W$, where $W := \sum_{i=1}^{N} |w_i|$.
- DFGT, DIFGT, SIFGT: $\left|\widetilde{G}(t_j) - G(t_j)\right| < \epsilon \cdot |G(t_j)|$.

In the standard setting, we let $\epsilon = 10^{-6}$. Nota bene the error bound of the dual tree methods is always tighter, since $|G(t_j)| \leq W$ for all t_j.

5.2 General Performance Issues

Before validating the algorithms with synthetic as well as real world data, we are going to address some particularities we have to deal with when comparing different performance results.

5.2.1 Influence of different bandwidths on runtime performance

First we point out a crucial issue that should be considered in all following runtime charts. During our test runs, we noticed a reproducible difference in runtimes for the direct Gauss Transform when using different bandwidths. While in theory the runtime does not depend on the bandwidth, in practice, the implementation of the exponential function seems to be responsible for the different performance results, since the costs for a single evaluation of the `exp`-function heavily depend on the argument. There is a whole theory and a lot of literature about efficient implementations of the exponential function both in software libraries and in hardware instruction sets, which is why we will not go into detail and analyze the particular implementation. Instead, we evaluate the DIRECT algorithm for a whole range of different bandwidths and whenever finding a deviant performance, we start more runs with nearby values. The results of this procedure for two uniform datasets can be found in Figure 5.1.

We can basically separate three different domains. A low bandwidth domain having the shortest runtimes, a high bandwidth domain with runtimes about 20 – 30% above, and a peak bandwidth domain in-between with runtimes exceeding the low bandwidth runtimes by more than 100%. In order to avoid misleading implications drawn from the speed comparison of the fast approximation algorithms due to these variations in performance, we either state explicitly which bandwidth domain we are currently dealing with or provide the runtimes of the direct Gauss Transform to get a fair rating.

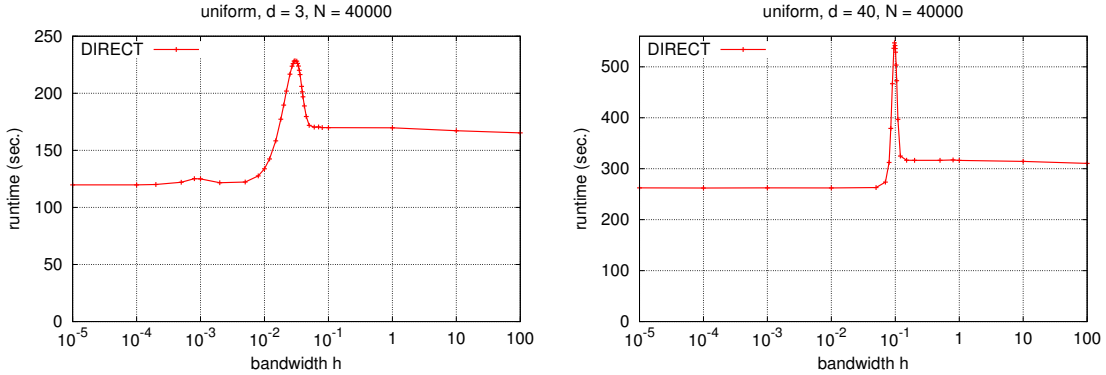


FIGURE 5.1: Runtimes of the Direct Gauss Transform for uniform data with varying bandwidth.

Uniform data	$d = 3$, $N = 40000$			$d = 40$, $N = 40000$		
bandwidth h	10^{-2}	$3 \cdot 10^{-2}$	10^{-1}	10^{-2}	$9.7 \cdot 10^{-2}$	1
runtime (sec.)	133.9	228.5	169.9	262.2	546.9	316.2

TABLE 5.1: Runtimes of the Direct Gauss Transform for uniform data for selected bandwidths.

5.2.2 Influence of the dual tree implementation on runtime performance

Another fact we have to point out before actually comparing the fast algorithms is the influence of our dual tree implementation on the runtime performance. When comparing the direct Gauss Transform using the dual tree structure (DT DIRECT) with the standard variant DIRECT, there are two antipodal effects: on the one hand, the dual tree version has to traverse the whole tree until arriving at the leaves in order to perform the direct evaluation, which slows it down; on the other hand, the data has been rearranged by the tree construction and access to points stored in a tree can be more cache efficient than access to points stored in a simple array.

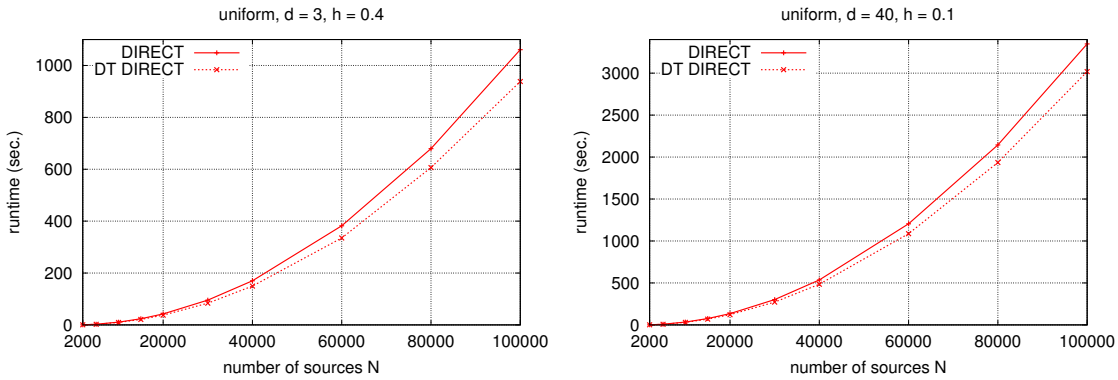


FIGURE 5.2: The Direct Gauss Transform vs. Direct Gauss Transform with dual tree structure.

The results show, that the dual tree variant is faster for every tested setting, on average the

speedup is about 10 %. Nota bene the tree construction time is included in the above runtimes. In Figure 5.2, the quadratic workload dependency on N can also be clearly recognized.

5.2.3 Tree construction times

Next, we compare the runtimes of the construction of the two tree structures presented in sections 4.4 and 4.5. We have shown that the creation of two VAM-Split SR-Trees (used by the DFGT and DIFGT algorithms) for N sources and M targets, respectively, has a time complexity of $\mathcal{O}\left(d\left(N\log_2 N + M\log_2 M\right)\right)$, while the VAM-Split SR-Tree with PCA (used by the SIFGT algorithm) has creation costs of $\mathcal{O}\left(N\cdot\left(d\cdot\log_2 N + d^3\right)\right)$. Thus for large d, the creation costs of the PCA variant will be much higher. Our test runs show, that for $d = 3$ the PCA variant takes about 2 times longer, whereas for $d = 40$ it takes approximately 20 times longer. However these costs are still much smaller than the actual evaluation costs.

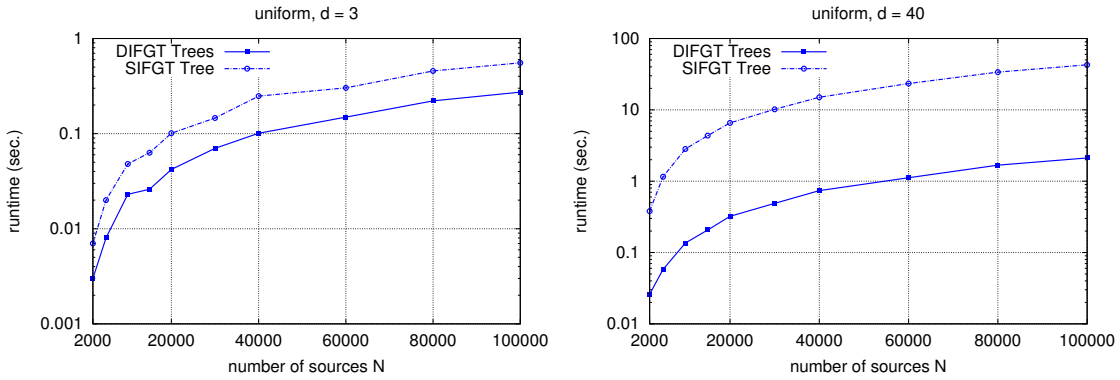


FIGURE 5.3: Runtimes for construction of trees in DIFGT and SIFGT algorithms.

Uniform data	$d = 3$, tree construction (sec.)		$d = 40$, tree construction (sec.)	
# sources N	DIFGT	SIFGT	DIFGT	SIFGT
2000	0.003	0.007	0.026	0.380
5000	0.008	0.020	0.058	1.154
10000	0.023	0.048	0.136	2.826
15000	0.026	0.063	0.207	4.351
20000	0.042	0.101	0.322	6.570
30000	0.070	0.146	0.489	10.18
40000	0.101	0.248	0.738	15.01
60000	0.149	0.303	1.119	23.37
80000	0.221	0.456	1.678	33.79
100000	0.273	0.555	2.116	42.81

TABLE 5.2: Runtimes for construction of trees in DIFGT and SIFGT algorithms.

5.3 Validation with Synthetic Data

The first tests are performed using equidistributed uniform data in the d-dimensional unit box $[0,1]^d$. The points have been generated by a pseudo-random number generator known as "Mersenne Twister" [44] developed by Makoto Matsumoto and Takuji Nishimura in 1997. The pseudo-random numbers computed by this algorithm are equidistributed in 623 dimensions and have a period of $2^{19937} - 1 \approx 4 \cdot 10^{6001}$. We utilize the variant named "MT 19937" implemented in the GNU Scientific Library [24].

5.3.1 Varying number of sources

We performed test runs with all algorithms DIRECT, FGT, IFGT, DFGT, DIFGT and SIFGT for equidistributed uniform data in fixed dimensions with varying number of sources N. In case the FGT or IFGT algorithms abort calculation due to the cost estimations presented above, their runtimes are ignored. Since the bandwidth h has such a strong influence on the performance, we provide results for multiple bandwidths to highlight particularities of the algorithms in specific bandwidth regimes. Our test runs for uniform data showed that these bandwidth regimes essentially do not depend on the number of sources, however on the dimension of the data. We therefore present results for 3-dimensional as well as 40-dimensional data at that time.

From the analysis of the algorithms in the previous section, we expect the Dual Tree algorithms DFGT, DIFGT and SIFGT to perform well for low bandwidths as well as very high bandwidths. For moderate bandwidths, they might not be much faster than the direct evaluation. Furthermore, the algorithms based on the *IFGT* expansion should perform better for high bandwidths.

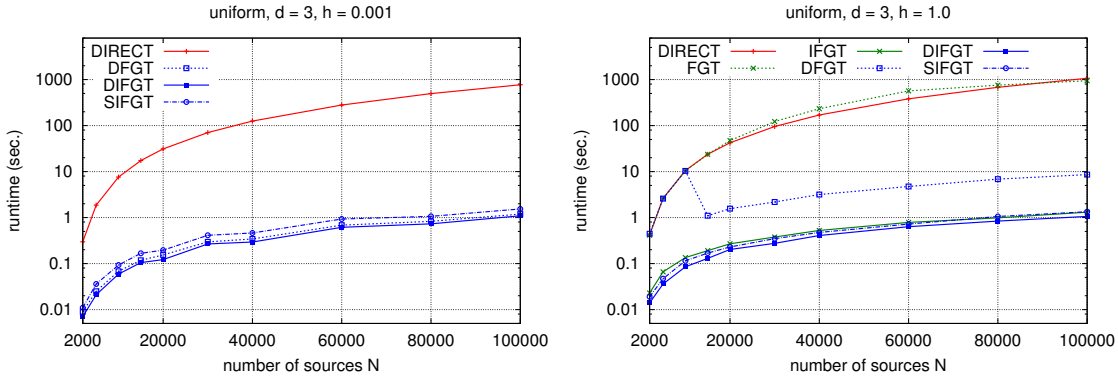


Figure 5.4: Uniform data in 3 dimensions, bandwidth $h = 0.001$ *(left)* and $h = 1$ *(right)*.

We choose 4 different bandwidths representing the most important characteristic cases. The results for low-dimensional data ($d = 3$) can be summarized as follows:

- $h = 0.001$ (Figure 5.4): For low bandwidth, all dual tree algorithms perform equally well and up to a factor of nearly 10^3 faster than DIRECT, while the FGT and IFGT algorithms abort evaluation, since their data structures cannot handle this case.

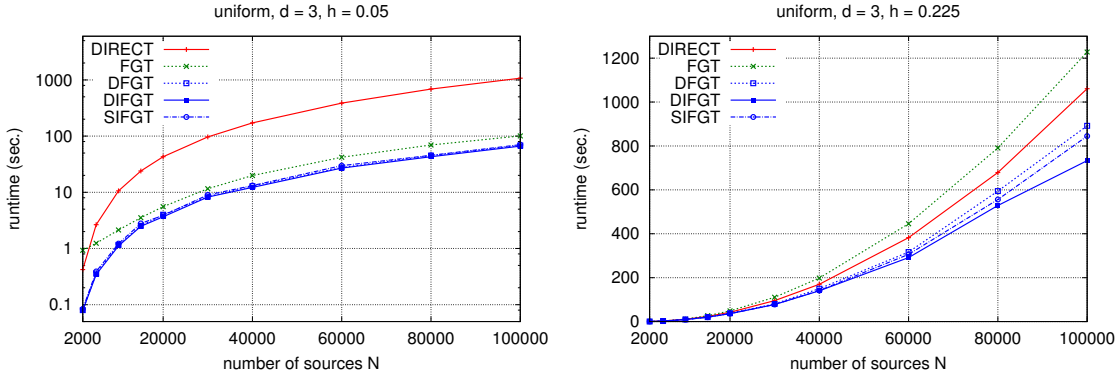


FIGURE 5.5: Uniform data in 3 dimensions, bandwidth $h = 0.05$ *(left)* and $h = 0.225$ *(right)*.

- $h = 1.0$ (Figure 5.4): For high bandwidth, all algorithms using the *IFGT* expansion (IFGT, DIFGT and SIFGT) outperform the direct variant by a factor of up to 10^3. While the FGT is slightly slower than DIRECT for $N \leq 80000$, the corresponding dual tree method DFGT is still up to 10^2 times faster.
- $h = 0.05$ (Figure 5.5): This bandwidth is taken out of a range where the FGT performs especially well. While it clearly outperforms direct evaluation, the dual tree algorithms still perform a little better, particularly for small number of sources.
- $h = 0.225$ (Figure 5.5): This bandwidth is chosen as the worst case bandwidth of the DIFGT algorithm, that is the bandwidth with its highest runtime. It is obviously still faster than all the other algorithms, even though all approximation algorithms have the same asymptotic performance than direct evaluation. Mainly due to the efficient tree implementation, the tree methods are still about 10 – 30% faster, while the FGT is roughly 15% slower than DIRECT.

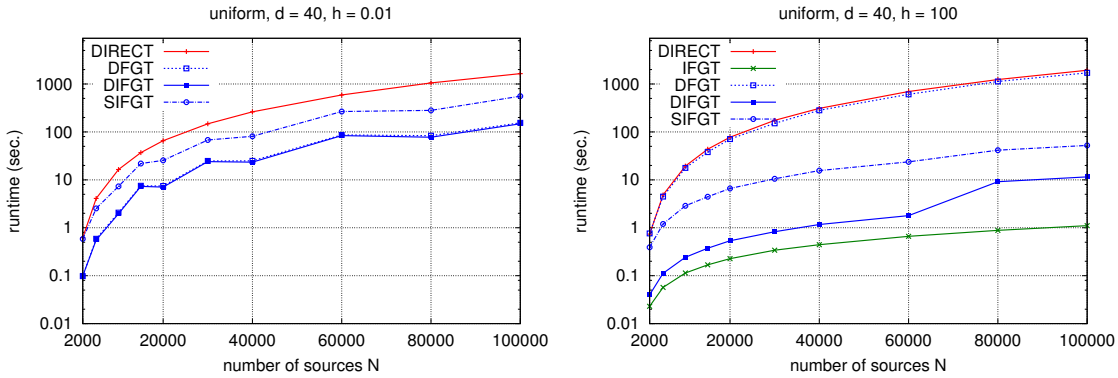


FIGURE 5.6: Uniform data in 40 dimensions, bandwidth $h = 0.01$ *(left)* and $h = 100$ *(right)*.

The results for high-dimensional data ($d = 40$) differ in various subtle ways:

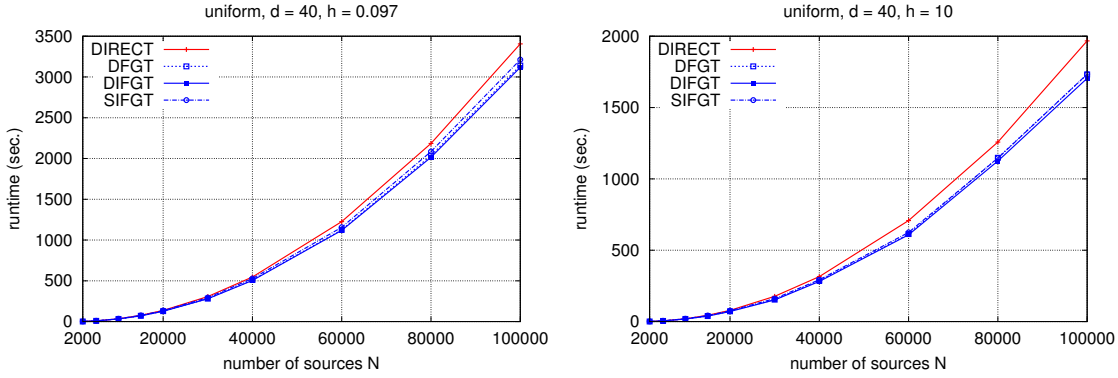


FIGURE 5.7: Uniform data in 40 dimensions, bandwidth $h = 0.097$ *(left)* and $h = 10$ *(right)*.

- $h = 0.01$ (Figure 5.6): For low bandwidth, the dual tree algorithms DFGT and DIFGT perform about 10 times faster than the direct version, while the SIFGT algorithm is obviously not only suffering from higher tree construction times (see Table 5.2), but also from the different arrangement of data points in the tree structure.
- $h = 100$ (Figure 5.6): For high bandwidth, the algorithms using the *IFGT* expansion again outperform the direct variant by far; this time, the SIFGT's inferior performance is solely due to tree construction costs. The DFGT algorithm on the other hand cannot outrun the DIRECT method perceptibly.
- $h \in [0.09, 10]$ (Figure 5.7): For the peak bandwidth, all dual tree algorithms have again the same asymptotic performance than direct evaluation. However the peak bandwidth regime is much broader, in this case, all values of h in the interval $[0.09, 10]$ yield the same asymptotic performance.

We can summarize the results as follows: For small dimensions, the dual tree methods perform very well for a wide range of bandwidths, while the FGT and IFGT only show their strengths for medium and high bandwidth, respectively. The latter two fail when it comes to higher dimensions, where the dual tree methods still clearly outperform the direct evaluation for very low and very high values of h. The best overall performance is delivered by the proposed DIFGT.

5.3.2 Varying dimension

Next, we consider a series of test runs for a fixed number of sources $N = 50000$ and vary the dimension d. As noted above, the runtime performance highly depends on the particular bandwidth, which itself should be chosen according to the dimension. Therefore, the following results must be interpreted with extreme care.

We omit the results of the FGT and IFGT since they do not perform any better (or even worse) than DIRECT for most bandwidths in dimensions greater than 3. The dual-tree variants on the other hand obviously do not suffer from increasing dimension. For low bandwidth (Figure 5.8), they perceptibly outperform direct evaluation with the SIFGT falling behind the performance

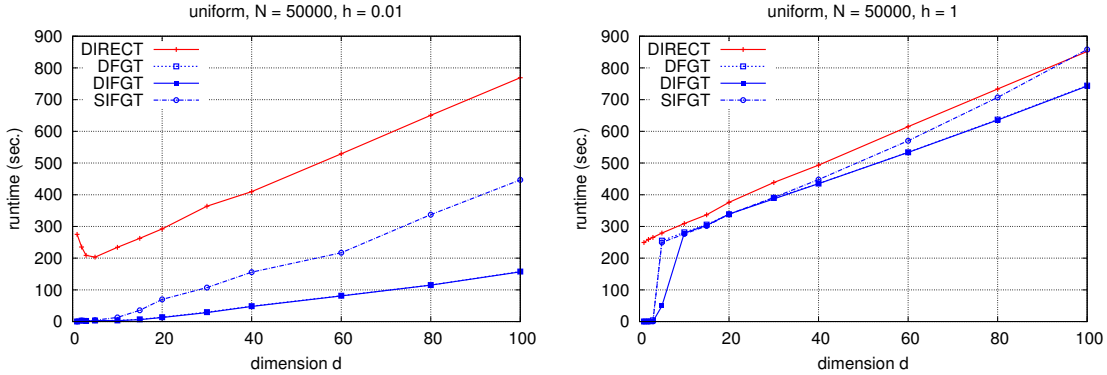


FIGURE 5.8: Uniform data ($N = 50000$ points) in varying dimensions, bandwidth $h = 0.01$ *(left)* and $h = 1$ *(right)*. Values of the DFGT in the plot are covered by the DIFGT values, that nearly coincide.

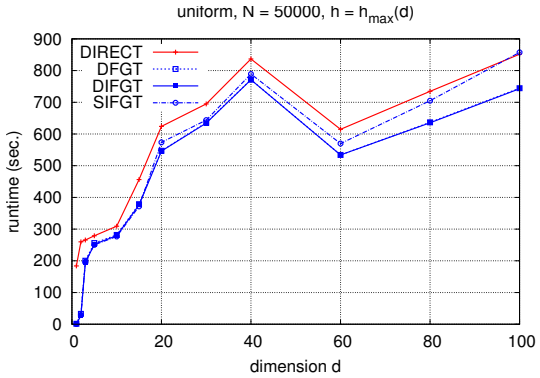


FIGURE 5.9: Uniform data ($N = 50000$ points) in varying dimensions, peak bandwidth $h_{max}(d)$ is chosen as bandwidth of the maximum DIFGT runtime for each dimension.

of the DFGT and DIFGT algorithms. For high bandwidth (Figure 5.8), the dual-tree methods can only benefit in low dimensions. The same pattern can be found when considering the worst case bandwidth $h_{max}(d)$ for each dimension (Figure 5.9), that is the bandwidth with the highest runtime of the DIFGT algorithm. To this end, we tested various different bandwidths in the range of $\left[10^{-5}, 1000\right]$ for all dimensions. The worst case values can be found in Table 5.3.

Still, we note that the DFGT and DIFGT never fall behind the direct evaluation in terms of performance, although the uniformly distributed dataset surely is the scenario, where the dual tree methods benefit least from their data structure.

dimension	1	2	3	5	10	15	20	30	40	60	80	100
bandwidth	0.0005	0.05	0.2	0.5	0.5	0.07	0.07	0.09	0.1	0.2	0.2	0.2

TABLE 5.3: Worst case bandwidths for different dimensions.

5.3.3 Varying error bound

We next run several tests with varying error bounds. Again, we first consider uniform data in 3 dimensions, then in 40 dimensions. Since the procedure of the dual tree algorithms strongly depends on the bandwidth h, all tests are performed for a wide range of different values of h. Remember, the error bounds guaranteed by the algorithms slightly differ:

- FGT, IFGT: $\left|\widetilde{G}(t_j) - G(t_j)\right| < \epsilon \cdot \sum_{i=1}^{N} |w_i|$.
- DFGT, DIFGT, SIFGT: $\left|\widetilde{G}(t_j) - G(t_j)\right| < \epsilon \cdot |G(t_j)|$.

For our test runs, we choose $\epsilon = 10^{-2}$, 10^{-6} and 10^{-10}. Our output has 14 significant digits. We calculate the maximum relative error for all target points:

$$err_{max} = \max_{j=1,\ldots,M} \frac{\left|\widetilde{G}(t_j) - G(t_j)\right|}{|G(t_j)|}. \tag{5.1}$$

If $err_{max} = 0$, we indicate this in the plot with a value of 10^{-15} in order to fit the result in the logarithmic scale.

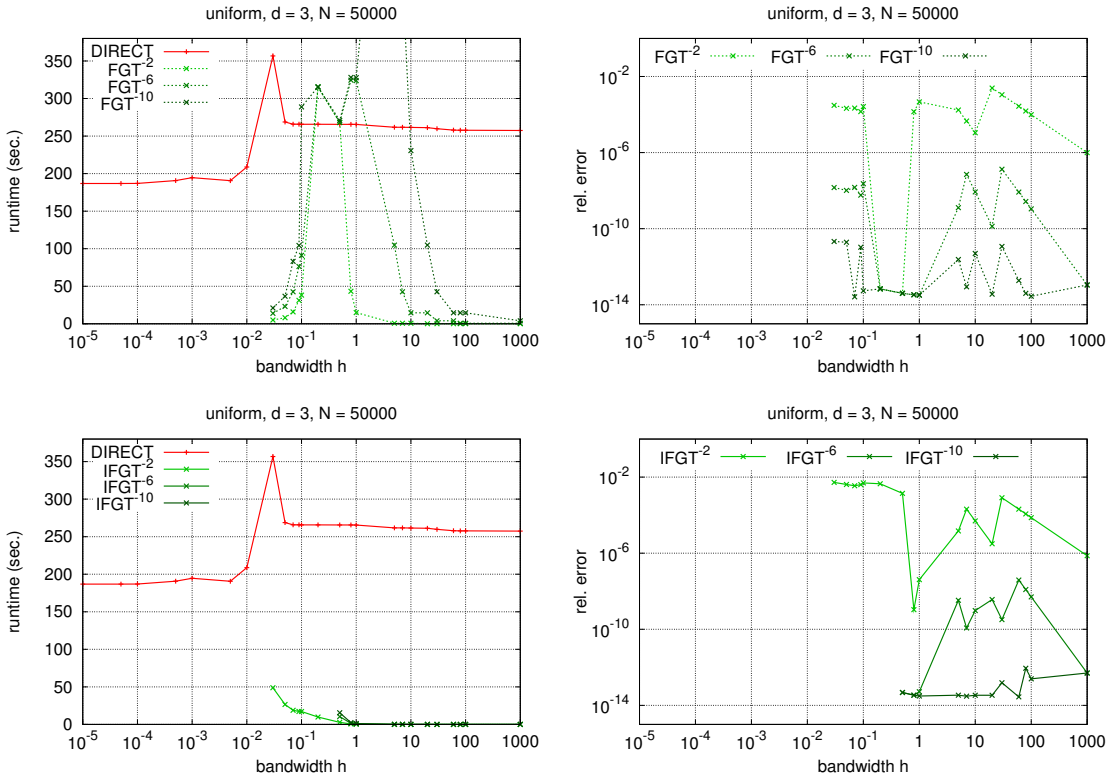


FIGURE 5.10: Performance of the FGT and IFGT on uniform 3D data for different relative error bounds 10^{-2}, 10^{-6} and 10^{-10}. *Left:* Runtimes. *Right:* Measured maximum error.

The FGT and IFGT algorithms (Figure 5.10) only apply for bandwidths of 0.03 and greater. Interestingly, the measured error err_{max} in fact undercuts the more rigorous bound of the dual tree methods. The FGT runtimes and errors vary considerably for different bandwidths and error bounds. For $\epsilon = 10^{-10}$, the runtimes for moderately high bandwidths surpass those of direct evaluation. The IFGT algorithm on the other hand yields excellent performance using $\epsilon = 10^{-2}$ for all bandwidths of 0.03 and greater, while letting $\epsilon = 10^{-6}$ or 10^{-10} requires bandwidths of 0.5 or greater.

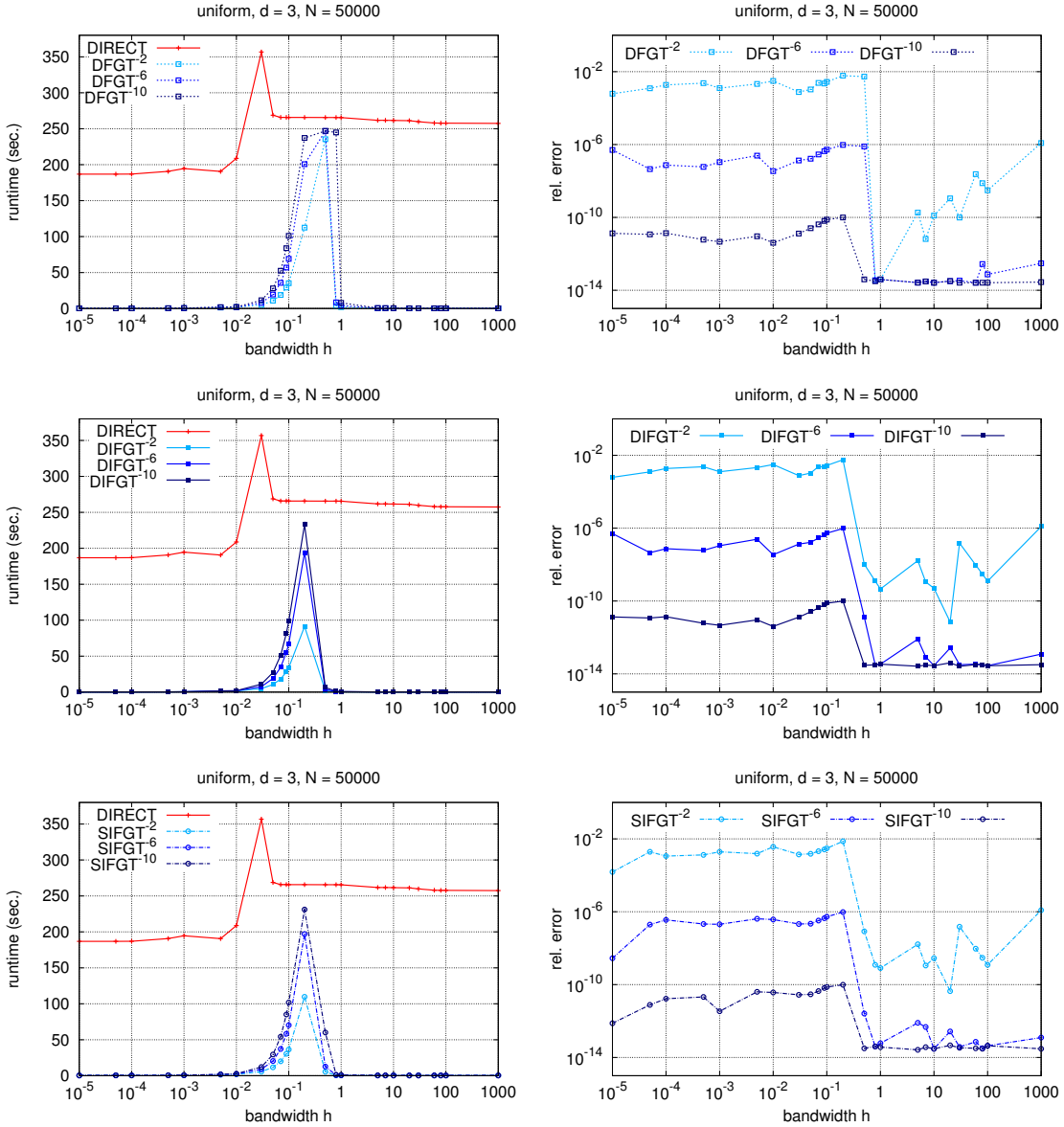


FIGURE 5.11: Performance of the DFGT, DIFGT and SIFGT on uniform 3D data for different relative error bounds 10^{-2}, 10^{-6} and 10^{-10}. *Left:* Runtimes. *Right:* Measured maximum error.

The dual tree algorithms (Figure 5.11) – especially the DIFGT and SIFGT variants – feature an excellent performance for the whole range of bandwidths between 10^{-5} and 1000. The runtime charts show that the algorithms have the highest workload in the bandwidth range of $[0.01, 0.5]$; the maximum relative error in this region is extremely close to the respective value of ϵ attesting sharp, reliable error bounds of the implemented dual tree methods. The fact that for $\epsilon = 10^{-2}$ and bandwidths around 10^{-1}, the IFGT is noticeably faster than the DIFGT and SIFGT can be reduced to the uniform distributed data which promotes simple partitioning schemes as the IFGT's farthest-point clustering as opposed to hierarchical tree structures.

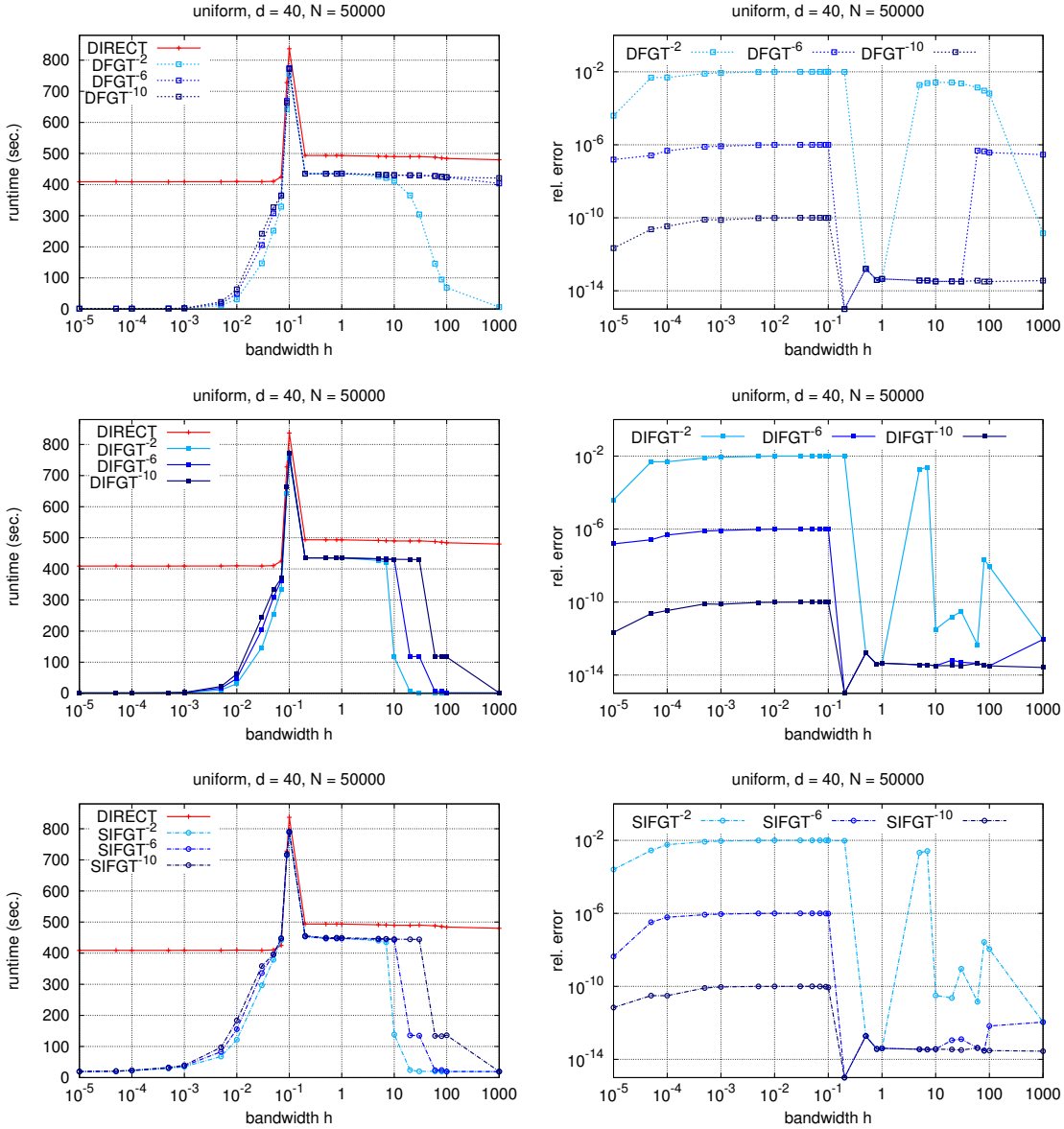


FIGURE 5.12: Performance of the DFGT, DIFGT and SIFGT on uniform 40D data for different relative error bounds 10^{-2}, 10^{-6} and 10^{-10}. *Left:* Runtimes. *Right:* Measured maximum error.

We finally consider the same test case for 40-dimensional uniformly distributed data (Figure 5.12). The results of the FGT and IFGT are omitted since the FGT does not perform faster than direct evaluation and the IFGT only performs faster for extremely high bandwidths. The dual-tree methods again perform well for low and high bandwidths, they are however not substantially faster than direct evaluation in the bandwidth range of $[0.1, 10]$. An interesting fact is the drop of the measured error for all values of ϵ for bandwidths around 1. In this case, the algorithm must carry out direct evaluation for nearly all involved sources and targets, the "worst case bandwidth" does not allow to leave out or approximate any calculations.

5.4 Application to Real World Data

In this section we perform test runs with various datasets taken from real world measurements. All records can be found on the Internet (see below for URLs) and have been used in various other testbeds in the context of performance analysis. Some data is taken from photographic images, other derives from classification or control problems. We now provide some details about the utilized datasets.

- shuttle ($d = 10,\ N = 58000$), URL (1), (2)
  This data has been used in the European "StatLog" project, that focused on machine learning methods in context of classification, prediction and control problems. The vectors consist of 9 (integer) attributes and a class, the attributes are shuttle control parameters supplied by the NASA.

- coocTexture ($d = 16,\ N = 68040$) and colorHistogram ($d = 32,\ N = 68040$), URL (1)
  The "Corel Image Features Data Set" has been generated from a collection of 68040 photo images from various categories. Each vector corresponds to a single photo and represents different color and texture values, that are used to measure (dis-)similarity between two images.

- acoustic ($d = 50,\ N = 78823$) and seismic ($d = 50,\ N = 78823$), URL (2), (3)
  The data was collected as part of a project named "SensIT" with the objective of vehicle classification in a wireless distributed sensor network. The two datasets are acoustic and seismic signals recorded by microphones and geophones. More information can be found on the project homepage (5) of the Sensor Networks Research Group and in [17].

- coverType ($d = 55,\ N = 581012$), URL (1)
  The data consists of cartographic and geological variables (no remotely sensed data) and shall be used in order to predict the corresponding forest cover type in a given area. The values represent divers attributes such as elevation in meters, slope in degrees, hill shade at different times of day and soil type. All values are integer or binary. More information can be found in [9].

- aerial ($d = 60,\ N = 275465$), URL (4)
  Each 60-dimensional vector represents texture information extracted from blocks of large aerial satellite photographs.

- bioRetina ($d = 62$, $N = 208506$), URL(5), (6)
  The data was generated from images of feline retinas as a part of a project at the Center for BioImage Informatics, University of California (Santa Barbara).

(1) http://archive.ics.uci.edu/ml/

(2) http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

(3) http://www.ece.wisc.edu/~sensit/

(4) http://vision.ece.ucsb.edu/download.html

(5) http://www.scl.ece.ucsb.edu/datasets/

(6) http://www.bioimage.ucsb.edu/

All datasets were scaled by components to fit in the unit box $[0, 1]^d$. In order to get manageable runtimes, we only use the first 50000 vectors of each dataset. In our test runs sources and targets coincide, all weights are set to 1 and various bandwidths between 10^{-5} and 100 are chosen. Nota bene the runtimes shown in all subsequent figures do not include tree creation times provided in Table 5.4. This is done since the tree construction is independent of most parameters such as weights, bandwidth and error bound and thus, for all applications that perform the Gauss Transform several times with different parameters – such as kernel density estimation, Gaussian process regression and regularized least-squares classification – the tree construction has to be done only once and can therefore be considered as pre-processing.

dataset ($N = 50000$)	dim	DIFGT trees (sec.)	SIFGT tree (sec.)
shuttle	10	0.260	0.851
coocTexture	16	0.412	2.611
colorHistogram	32	0.770	11.64
acoustic	50	1.107	27.80
seismic	50	1.116	27.87
coverType	55	1.346	19.76
aerial	60	1.522	42.34
bioRetina	62	1.659	49.20

TABLE 5.4: Runtimes for construction of trees in DIFGT and SIFGT algorithms.

Now let us consider the runtime performance results presented in Figures 5.13 and 5.14. The left column shows the runtime for $\epsilon = 10^{-6}$, the right column the runtime for $\epsilon = 10^{-2}$. For moderate dimensions ($d = 10$, $d = 16$), our new algorithms (DIFGT and SIFGT) perform noticeably faster than DIRECT for a wide bandwidth range. The SIFGT yields particularly good results for the shuttle dataset, which might be due to the fact that we are dealing with integer data that is more likely to have local low-dimensional features than continuous data. The DFGT on the other hand drops behind the other tree methods for high bandwidths, as already seen for uniformly distributed data in high dimensions.

All plots in the runtime charts basically have the same appearance.

- For small bandwidth, the runtimes range between several hundred milliseconds and ten seconds roughly, this scenario is handled by the quick nearest-neighbor search abilities of the dual tree methods.
- For increasing bandwidth, the runtimes also increase until reaching a maximum close to the direct evaluation runtime. For this bandwidth range, the dual tree method cannot achieve a great performance enhancement since all interactions of sources and targets are relevant, but the bandwidth is still not high enough for the *IFGT* expansion scheme to work well.
- Finally for even higher bandwidths, the runtimes begin to decrease again until reaching very neat performance again, when the series expansion is applied at very low tree levels already.

We notice that especially in the bandwidth range of increasing and decreasing performance, the SIFGT is often faster than its competitors due to the improved data partitioning and distance calculation scheme.

Let us finally consider the measured maximum relative errors for different error bounds ($\epsilon = 10^{-2}, 10^{-6}, 10^{-10}$) presented in Figures 5.15 and 5.16. For the bandwidth range of increasing and maximum runtime, the measured errors are almost consistently very close to the demanded error bound, while never surpassing it. This can be attributed to the excellent error control mechanisms of our dual tree implementation, that seem to work especially well for real world datasets. For higher bandwidths, the maximum error often decreases about several orders of magnitude. This is a consequence of the application of the *IFGT* expansion, that often yields higher precision than guaranteed by the appropriate local error bound.

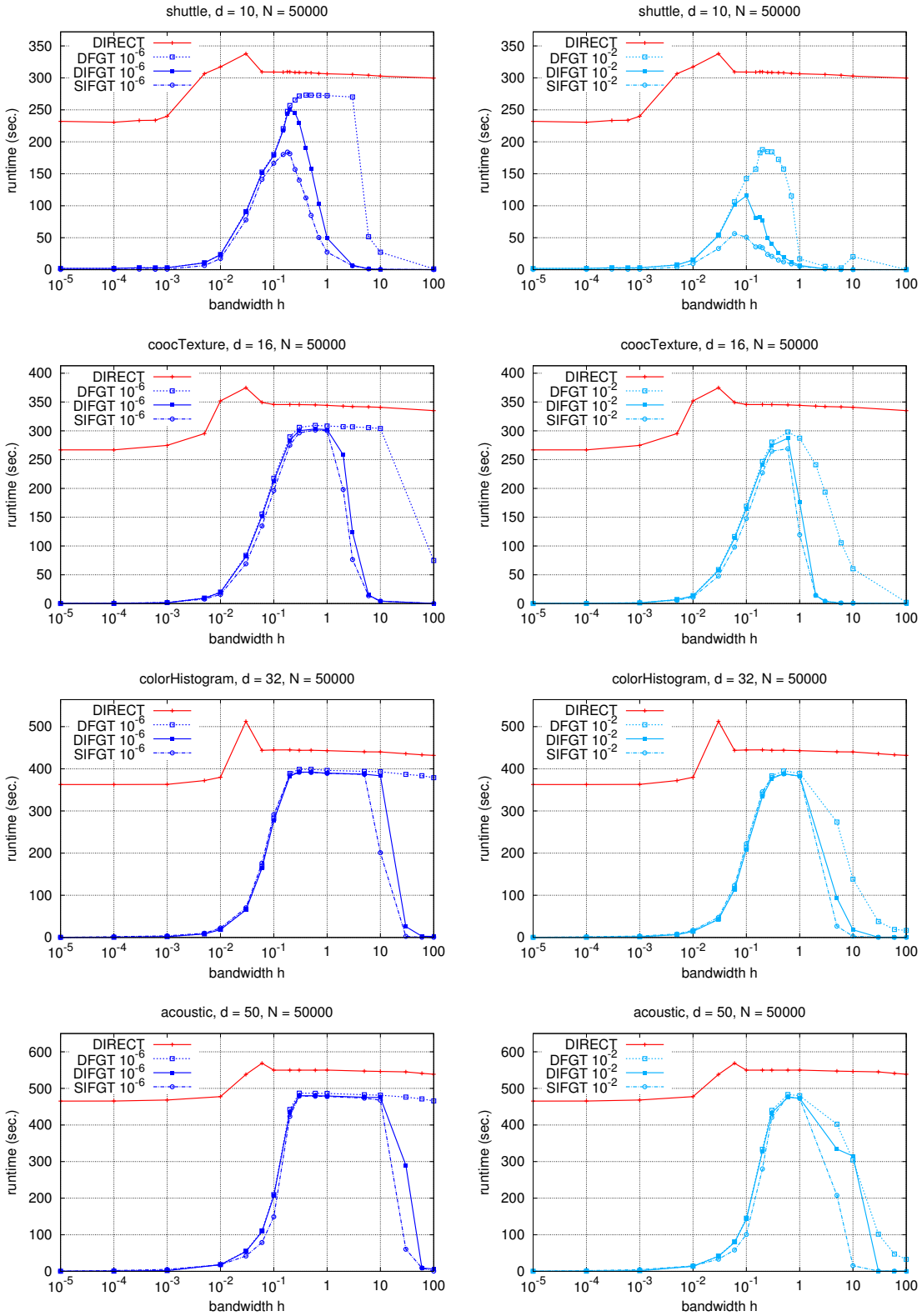


FIGURE 5.13: Performance of the different tree algorithms for high-dimensional real world datasets. *Left:* Error bound $\epsilon = 10^{-6}$. *Right:* $\epsilon = 10^{-2}$.

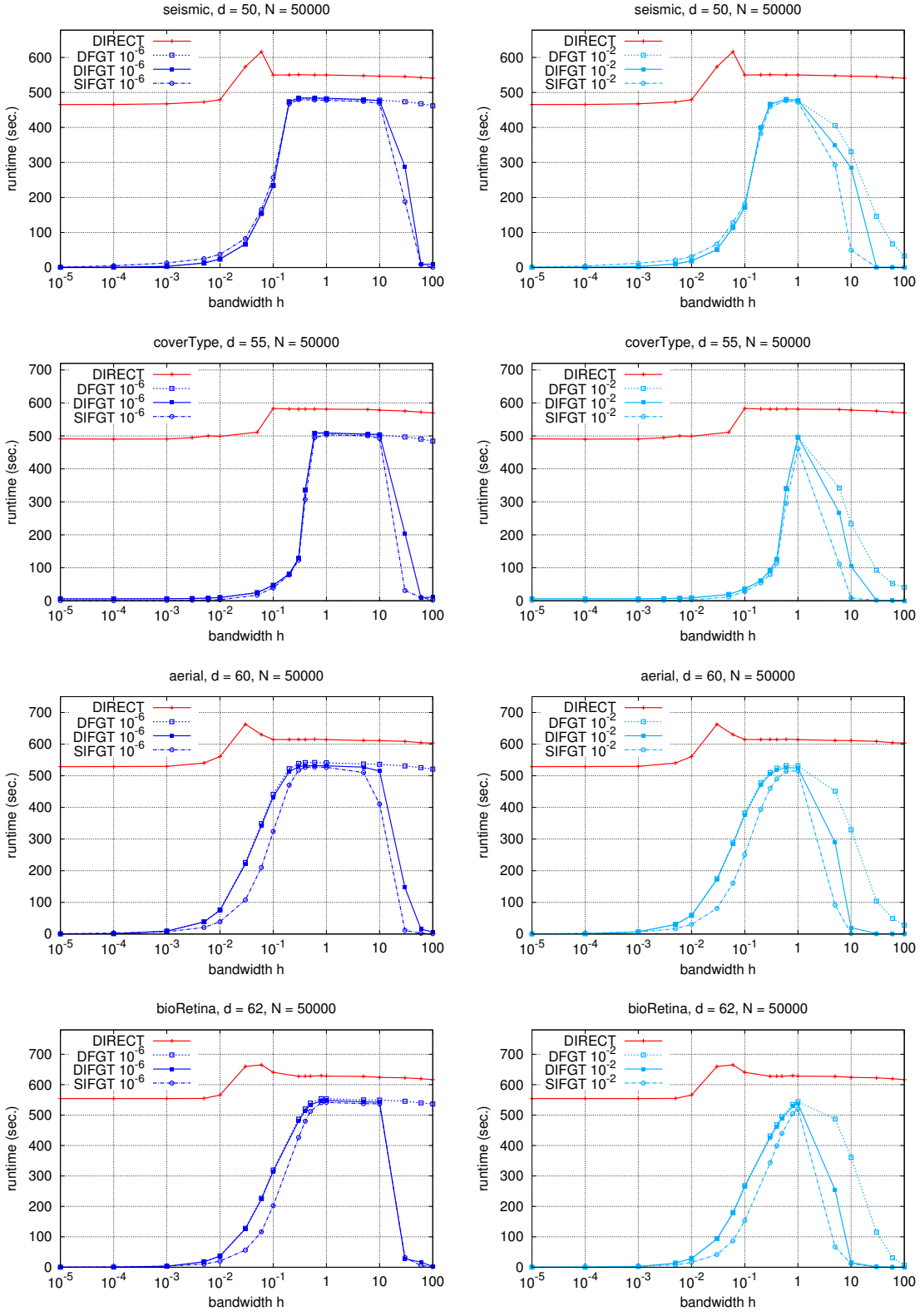


FIGURE 5.14: Performance of the different tree algorithms for high-dimensional real world datasets. *Left:* Error bound $\epsilon = 10^{-6}$. *Right:* $\epsilon = 10^{-2}$.

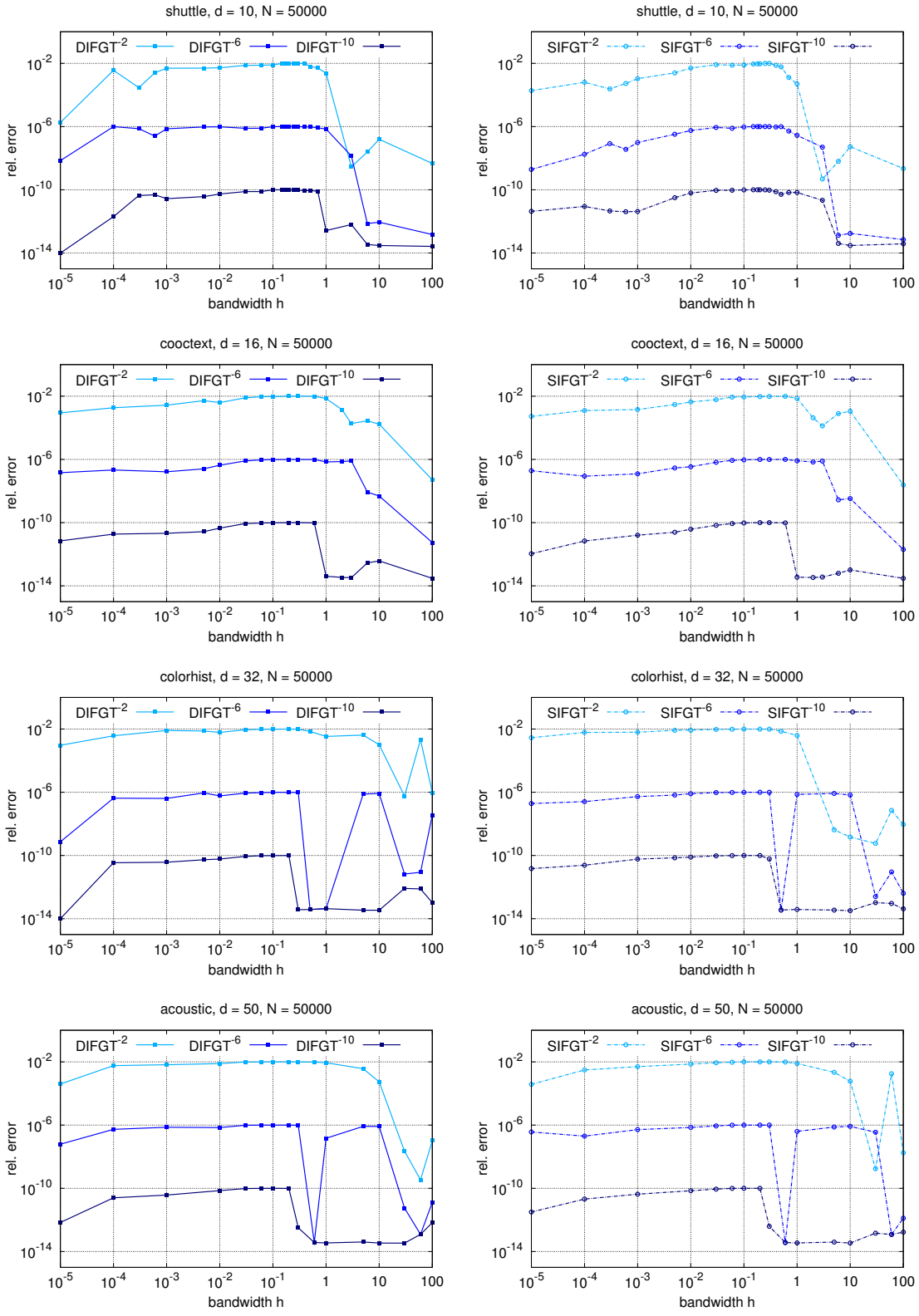


FIGURE 5.15: Measured maximum relative errors of tree algorithms for specified relative error bounds of 10^{-2}, 10^{-6} and 10^{-10}. *Left:* DIFGT algorithm. *Right:* SIFGT algorithm.

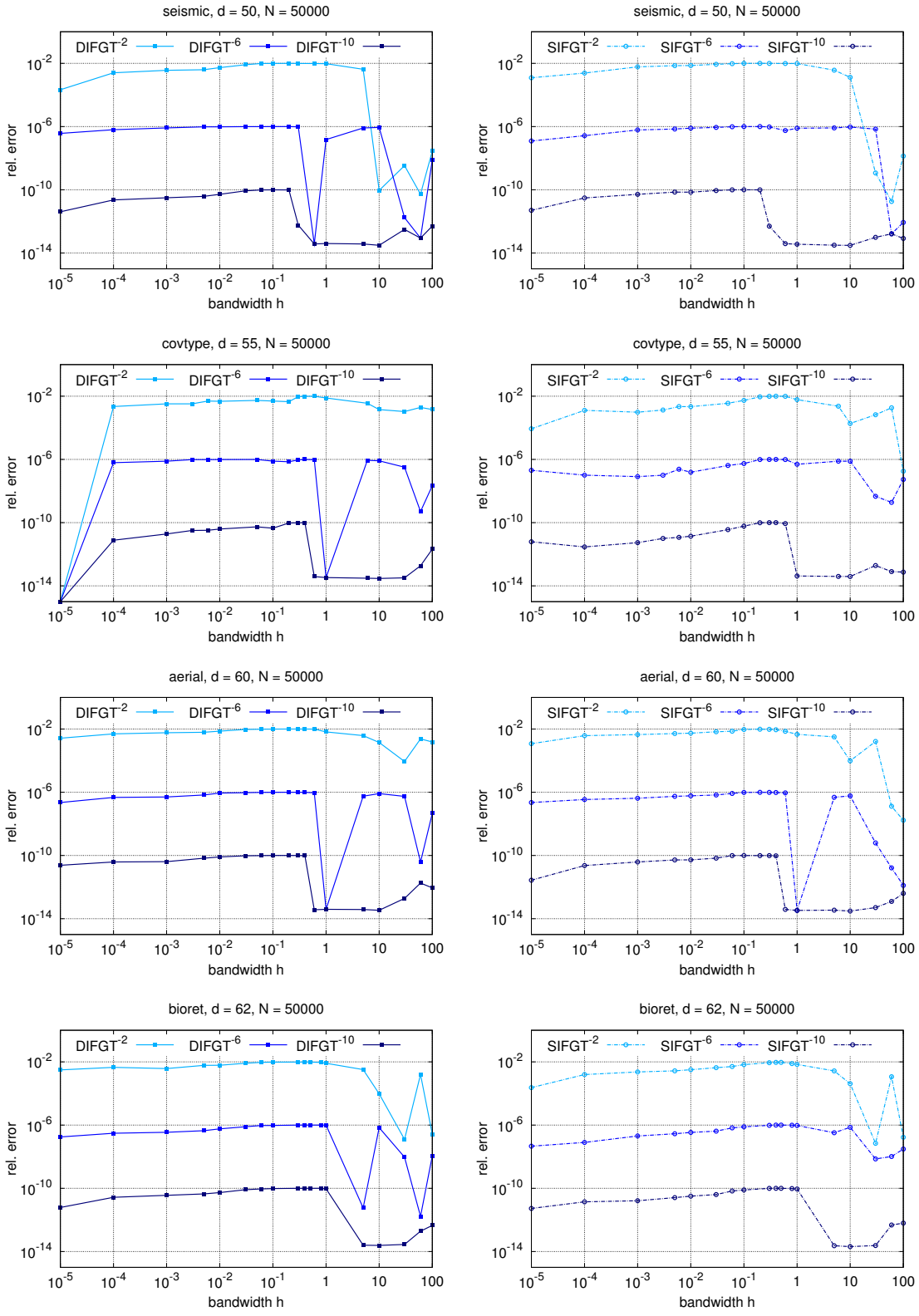


FIGURE 5.16: Measured maximum relative errors of tree algorithms for specified relative error bounds of 10^{-2}, 10^{-6} and 10^{-10}. *Left:* DIFGT algorithm. *Right:* SIFGT algorithm.

6 Concluding Remarks

6.1 Summary

In this work, we presented an extensive disquisition on diverse algorithms for the fast approximation of the discrete Gauss transform. We first introduced several applications in statistical learning theory, in fact kernel density estimation, Gaussian process regression and regularized least-squares classification. The main workload of these important learning tasks can be traced back to the solution of a linear system with the Gaussian kernel matrix inducing the multiple evaluation of the Gauss transform. We then provided an overview on multivariate Taylor expansions, Hermite functions and the fast multipole methods, the theoretical basics required for the detailed understanding of the approximation techniques. Reaching our main focus, we discussed and analyzed the established *Fast Gauss Transform*, the *Improved Fast Gauss Transform* as well as the rather recent *Dual-Tree Fast Gauss Transform*, highlighting their specific advantages as well as their drawbacks. We then combined the dual-tree idea and the expansion of the *IFGT* with a new data structure well-adapted to our particular approximation scheme to get the *Dual-Tree Improved Fast Gauss Transform (DIFGT)*; next, we developed a refined partitioning scheme discovering local low dimensional features in the data, that led to our *Single-Tree (SIFGT)* variant. The implementations were first tested with uniformly distributed data; the dual-tree methods yield remarkable performance improvements especially in high dimension, where both the *FGT* and *IFGT* algorithms suffer from intrinsic issues. The test cases with real world data finally showed the effectiveness of the developed data structures and the good interplay of the selected speed-up techniques in the *DIFGT* and *SIFGT*, resulting in crucial cost reductions for diverse datasets of dimensions up to 60.

6.2 Outlook

We thus have introduced two new algorithms that surpass the existing ones in terms of performance and can be used in diverse applications. A next step should be the empirical testing of these algorithms in the discussed context of statistical learning problems. Since the approximation methods work well for a wide range of bandwidths, we expect a considerable speed-up of the selection process of the optimal bandwidth. Further tests must show for which kinds of data the fast algorithms still achieve sensible performance improvements for the chosen optimal value. Finally, we would like to encourage the use of dual-tree techniques in the wide field of fast multipole methods, since the basic concept is independent of the kernel function. An appropriate expansion and a well-fitted tree structure are the central prerequisites that can permit fast evaluations for a variety of different kernel methods.

Bibliography

[1] N. Aronszajn. La théorie générale des noyaux reproduisants et ses applications, Première Partie. In *Proceedings of the Cambridge Philosophical Society*, volume 39, page 133, 1944.

[2] N. Aronszajn. Theory of Reproducing Kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, May 1950.

[3] Pierre Baldi and Søren Brunak. *Bioinformatics: The Machine Learning Approach*. The MIT Press, second edition, 2001.

[4] Heinz Bauer. *Wahrscheinlichkeitstheorie, 5. Auflage*. de Gruyter, 2002.

[5] B. J. C. Baxter and George Roussos. A new Error Estimate of the Fast Gauss Transform. *SIAM Journal on Scientific Computing*, 24(1):257–259, January 2002.

[6] Rick Beatson and Leslie Greengard. A short course on fast multipole methods. In *Wavelets, Multilevel Methods and Elliptic PDEs (Numerical Mathematics and Scientific Computation)*, pages 1–37. Oxford University Press, 1997.

[7] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, 1990.

[8] Marshall Bern and David Eppstein. Approximation Algorithms for Geometric Problems. In *Approximation Algorithms for NP-hard Problems*, pages 325–329. PWS Publishing Co., Boston, 1996.

[9] Jock A. Blackard and Denis J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151, December 1999.

[10] D. S. Broomhead and Michael Kirby. A New Approach for Dimensionality Reduction: Theory and Algorithms. *SIAM Journal of Applied Mathematics*, 60:2114–2142, 2000.

[11] Martin D. Buhmann. *Radial Basis Functions: Theory and Implementations*. Cambridge University Press, 2003.

[12] Francesco Camastra. Data Dimensionality Estimation Methods: A survey. *Pattern Recognition*, 36(12):2945–2954, December 2003.

[13] Hongwei Cheng, Leslie Greengard, and Vladimir Rokhlin. A Fast Adaptive Multipole Algorithm in Three Dimensions. *Journal of Computational Physics*, 155(2):468–498, November 1999.

[14] T. S. Chihara. *An Introduction to Orthogonal Polynomials*. Gordon and Breach, Science Publishers, Inc., 1978.

[15] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[16] Roussos Dimitrakopoulos, editor. *Geostatistics for the Next Century*. Kluwer Academic Publishers, 1994.

[17] Marco Duarte and Yu-Hen Hu. Vehicle Classification in Distributed Sensor Networks. *Journal of Parallel and Distributed Computing*, 64(7):826–838, July 2004.

[18] Tomás Feder and Daniel H. Greene. Optimal Algorithms for Approximate Clustering. In *Proceedings of the 20th annual ACM symposium on Theory of computing*, pages 434–444, 1988.

[19] Imola K. Fodor. A survey of dimension reduction techniques. Technical report, Lawrence Livermore National Laboratory, June 2002. No. UCRL-ID-148494.

[20] Walter Gautschi. *Orthogonal Polynomials: Computation and Approximation*. Oxford University Press, 2004.

[21] Hans-Otto Georgii. *Stochastik - Einführung in die Wahrscheinlichkeitstheorie und Statistik, 2. Auflage*. de Gruyter, 2004.

[22] Marc N. Gibbs. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, Department of Physics, University of Cambridge, 1997.

[23] Elmer G. Gilbert, Daniel W. Johnson, and S. Sathiya Keerthi. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, April 1988.

[24] GNU. GNU Scientific Library. `http://www.gnu.org/software/gsl/`, January 2008.

[25] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[26] Anne Greenbaum. *Iterative Methods for Solving Linear Systems*. Society for Industrial Mathematics, Philadelphia, USA, 1997.

[27] Leslie Greengard and Vladimir Rokhlin. A Fast Algorithm for Particle Simulations. *Journal of Computational Physics*, 73(2):325–348, December 1987.

[28] Leslie Greengard and Vladimir Rokhlin. A new version of the Fast Multipole Method for the Laplace equation in three dimensions. In *Acta Numerica*, volume 6, pages 229–269. Cambridge University Press, 1997.

[29] Leslie Greengard and John Strain. The Fast Gauss Transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, January 1991.

[30] Michael Griebel and Marc Alexander Schweitzer, editors. *Meshfree Methods for Partial Differential Equations (Lecture Notes in Computational Science and Engineering)*, volume 26. Springer Verlag, 1st edition, 2002.

[31] Nail A. Gumerov and Ramani Duraiswami. *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*. Elsevier Series in Electromagnetism, 2004.

[32] Antonin Guttman. R-trees: a Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, 1984.

[33] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer series in statistics, corr. 3rd printing edition, 2003.

[34] Stefan Hildebrandt. *Analysis 2*. Springer-Verlag Berlin Heidelberg, 2003.

[35] Dorit S. Hochbaum and David B. Shmoys. A Best Possible Heuristic for the K-Center Problem. *Mathematics of Operations Research*, 10(2):180–184, May 1985.

[36] Alexander T. Ihler. An Overview of Fast Multipole Methods. Technical report, MIT Area Exam, 2004. http://www.ics.uci.edu/~ihler/papers/ihler_area.pdf.

[37] M. C. Jones, J. S. Marron, and S. J. Sheather. A Brief Survey of Bandwidth Selection for Density Estimation. *Journal of the American Statistical Association*, 91(433):401–407, March 1996.

[38] Norio Katayama and Shin'ichi Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 369–380, Tucson, 1997.

[39] Misha E. Kilmer and Dianne P. O'Leary. Choosing Regularization Parameters in Iterative Methods for Ill-posed Problems. *SIAM Journal on Matrix Analysis and Applications*, 22(4):1204–1221, 2001.

[40] George Kimeldorf and Grace Wahba. Some Results on Tchebycheffian Spline Functions. *Journal of Mathematical Analysis and Applications*, 33:82–95, 1971.

[41] Dongryeol Lee and Alexander Gray. Faster Gaussian Summation: Theory and Experiment. In *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*. AUAI Press, Arlington, 2006.

[42] Dongryeol Lee, Alexander Gray, and Andrew Moore. Dual-Tree Fast Gauss Transforms. *Advances in Neural Information Processing Systems*, 18:747–754, 2006.

[43] Ming C. Lin and John F. Canny. A Fast Algorithm for Incremental Distance Calculation. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1008–1014, April 1991.

[44] Makoto Matsumoto and Takuji Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[45] E. H. Moore. General analysis, Part I / Part II. *Memoirs of the American Philosophical Society*, 1935 / 1939.

[46] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[47] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[48] Vikas C. Raykar. The fast Gauss transform with all the proofs. University of Maryland, http://www.umiacs.umd.edu/~vikas/publications/FGT.pdf, January 2008.

[49] Vikas C. Raykar, Changjiang Yang, Ramani Duraiswami, and Nail. A. Gumerov. Fast Computation of Sums of Gaussians in High Dimensions. Technical report, Dept. of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, 2005. No. CS-TR-4767.

[50] Ryan Rifkin. *Everything Old is New Again: A Fresh Look at Historical Approaches in Machine Learning*. PhD thesis, Massachusetts Institute of Technology, 2002.

[51] Ryan Rifkin, Gene Yeo, and Tomaso Poggio. Regularized Least-Squares Classification. In *Advances in Learning Theory: Methods, Models and Applications (NATO Science Series III: Computer and Systems Sciences)*, volume 190. VIOS Press, Amsterdam, 2003.

[52] John T. Robinson. The K-D-B-tree: a Search Structure for Large Multidimensional Dynamic Indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 10–18, Ann Arbor, 1981.

[53] O. Rodrigues. Mémoire sur l'attraction des sphéroides. *Correspondence sur l'Ecole Royale Polytechnique*, 3:361–385, 1816.

[54] Robert Schaback and Holger Wendland. Kernel Techniques: From Machine Learning to Meshless Methods. In *Acta Numerica*, volume 15, pages 543–639. Cambridge University Press, 2006.

[55] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A Generalized Representer Theorem. In *Proceedings of the Annual Conference on Computational Learning Theory*, pages 416–426, 2001.

[56] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels - Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2002.

[57] Bernhard Schölkopf, Koji Tsuda, and Jean-Philippe Vert, editors. *Kernel Methods in Computational Biology*. The MIT Press, 2004.

[58] David W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley Series in probability and mathematical statistics, 1992.

[59] Robert F. Sproull. Refinements to Nearest-Neighbor Searching in k-Dimensional Trees. *Algorithmica*, 6(1):579–589, 1991.

[60] Josef Stoer. *Numerische Mathematik 1, 9. Auflage*. Springer-Verlag, 2005.

[61] Josef Stoer. *Numerische Mathematik 2, 5. Auflage*. Springer-Verlag, 2005.

[62] Otto Szász. On the relative extrema of the Hermite orthogonal functions. *Journal of the Indian Mathematical Society*, 15:129–134, 1951.

[63] Berwin A. Turlach. Bandwidth Selection in Kernel Density Estimation: A Review. Technical report, Université Catholique de Louvain, C.O.R.E. and Institut de Statistique, 1993. No. DP9317.

[64] Vladimir N. Vapnik. *Statistical learning theory*. Wiley-Interscience, 1998.

[65] David A. White and Ramesh Jain. Similarity Indexing: Algorithms and Performance. In *Proceedings of SPIE: Storage and Retrieval for Still Image and Video Databases IV*, volume 2670, pages 62–75, 1996.

[66] David A. White and Ramesh Jain. Similarity Indexing with the SS-tree. In *Proceedings of the 12th International Conference on Data Engineering*, pages 516–523, New Orleans, 1996.

[67] Changjiang Yang, Ramani Duraiswami, and Nail A. Gumerov. Improved Fast Gauss Transform. Technical report, Dept. of Computer Science and Institute for Advanced Computer Studies, University of Maryland, 2003. No. CS-TR-4495.