Fast computation of sums of Gaussians in high dimensions

VIKAS CHANDRAKANT RAYKAR, CHANGJIANG YANG, RAMANI DURAISWAMI, and NAIL GUMEROV
Perceptual Interfaces and Reality Laboratory
Department of Computer Science and Institute for Advanced Computer Studies
University of Maryland, CollegePark, MD 20783
{vikas,yangcj,ramani,gumerov}@umiacs.umd.edu

Evaluating sums of multivariate Gaussian kernels is a key computational task in many problems in computational statistics and machine learning. The computational cost of the direct evaluation of such sums scales as the product of the number of kernel functions and the evaluation points. The fast Gauss transform proposed by Greengard and Strain (1991) is a ϵ-exact approximation algorithm that reduces the computational complexity of the evaluation of the sum of N Gaussians at M points in d dimensions from $\mathcal{O}(MN)$ to $\mathcal{O}(M+N)$. However, the constant factor in $\mathcal{O}(M+N)$ grows exponentially with increasing dimensionality d, which makes the algorithm impractical for dimensions greater than three. In this paper we present a new algorithm where the constant factor is reduced to asymptotically polynomial order. The reduction is based on a new multivariate Taylor's series expansion (which can act both as a local as well as a far field expansion) scheme combined with the efficient space subdivision using the k-center algorithm. The proposed method differs from the original fast Gauss transform in terms of a different factorization, efficient space subdivision, and the use of point-wise error bounds. Algorithm details, error bounds, procedure to choose the parameters and numerical experiments are presented. As an example we shows how the proposed method can be used for very fast ϵ-exact multivariate kernel density estimation.

[**CS-TR-4767/UMIACS-TR-2005-69**]: April 8, 2006

Contents

1. INTRODUCTION

In most kernel based machine learning algorithms [Shawe-Taylor and Cristianini 2004] and non-parametric statistics [Izenman 1991] the key computational task is to compute a linear combination of local kernel functions centered on the training data, i.e.,

$$f(x) = \sum_{i=1}^{N} q_i k(x, x_i),$$

where $\{x_i \in \mathbf{R}^d, i = 1, \ldots, N\}$ are the N training data points, $\{q_i \in \mathbf{R}, i = 1, \ldots, N\}$ are the weights, $k : \mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$ is the local kernel function, and $x \in \mathbf{R}^d$ is the test point at which $f(.)$ is to be computed. f is the regression/classification function in case of regularized least squares [Poggio and Smale 2003], Gaussian process regression [Williams and Rasmussen 1996], support vector machines [Cristianini and Shawe-Taylor 2000], kernel regression [Wand and Jones 1995], and radial basis function neural networks [Girosi et al. 1995]. For non-parametric density estimation it is the kernel density estimate [Wand and Jones 1995]. Also many kernel methods like kernel principal component analysis [Smola et al. 1996] and spectral clustering [Chung 1997] algorithms involve computing the eigen values of the Gram matrix. Training Gaussian process machines [Seeger 2004] involves the solution of a linear system of equations. Solutions to such problems can be obtained using iterative methods, where the dominant computation is evaluation of $f(x)$. Recently, such problems have been collectively referred to as *N-body problems in learning* by [Gray and Moore 2001], in analogy with the columbic N-body problems occurring in computational physics.

The most commonly used kernel function is the *Gaussian kernel*

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

where h is called the *bandwidth* of the kernel. The Gaussian kernel is a *local kernel* in the sense that $\lim_{\|x-y\| \to \infty} K(x, y) = 0$. The sum of multivariate Gaussian kernels is known as the *discrete Gauss transform*. More formally, for each *target point* $\{y_j \in \mathbf{R}^d\}_{j=1,\ldots,M}$ the *discrete Gauss transform* is defined as,

$$G(y_j) = \sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2/h^2}. \tag{1}$$

where $\{q_i \in \mathbf{R}\}_{i=1,\ldots,N}$ are the *source weights*, $\{x_i \in \mathbf{R}^d\}_{i=1,\ldots,N}$ are the *source points*, i.e., the center of the Gaussians, and $h \in \mathbf{R}^+$ is the *source scale* or *bandwidth*. In other words $G(y_j)$ is the total contribution at y_j of N Gaussians centered at x_i each with bandwidth h. Each Gaussian is weighted by the term q_i.

The computational complexity to evaluate the discrete Gauss transform (Equation 1) at M target points is $\mathcal{O}(MN)$. This makes the computation for large scale problems prohibitively expensive. In many machine learning tasks data-sets containing more than 10^4 points are already common and larger problems are of interest.

The *Fast Gauss Transform* (FGT) is an $\epsilon - exact$ approximation algorithm that reduces the computational complexity to $\mathcal{O}(M + N)$, at the expense of reduced precision, which however can be arbitrary. The constant depends on the desired precision, dimensionality of the problem, and the bandwidth. Given any $\epsilon > 0$, it computes an approximation $\hat{G}(y_j)$ to $G(y_j)$ such that the maximum absolute error relative to the total weight $Q = \sum_{i=1}^{N} |q_i|$ is upper bounded by ϵ, i.e.,

$$\max_{y_j} \left[\frac{|\hat{G}(y_j) - G(y_j)|}{Q} \right] \leq \epsilon. \tag{2}$$

The FGT was first proposed by [Greengard and Strain 1991] and applied successfully to a few lower dimensional applications in mathematics and physics. However the algorithm has not been widely used much in statistics, pattern recognition, and machine learning applications where higher dimensions occur commonly. An important reason for the lack of use of the algorithm in these areas is that the performance of the proposed FGT degrades exponentially with increasing dimensionality, which makes it impractical for the statistics and pattern recognition applications. There are three reasons contributing to the degradation of the FGT in higher dimensions:

(1) The number of the terms in the Hermite expansion used by the FGT grows exponentially with dimensionality d (as p^d, where p is the truncation number), which causes the constant factor associated with the nominal complexity $\mathcal{O}(M+N)$ increases exponentially with dimensionality. So the total computations and the amount of memory required increases dramatically as the dimensionality increases.

(2) The space subdivision scheme used by the fast Gauss transform is a uniform box subdivision scheme which is tolerable in lower dimensions but is extremely inefficient in higher dimensions.

(3) The constant term due to the translation of the far-field Hermite series to the local Taylor series grows exponentially fast with dimension making it impractical for dimensions greater than three[1].

In this paper we present an *improved fast Gauss transform* (IFGT) suitable for higher dimensions. The IFGT differs from the FGT in the following three ways, addressing each of the issues above.

(1) A multivariate Taylor's series expansion is used to reduce the number of the expansion terms to the polynomial order.

(2) The k-center algorithm is applied to subdivide the space which is more efficient in higher dimensions.

(3) Our expansion can act both as a far-field and local expansion. As a result we do not have separate far-field and local expansions which eliminates the cost of translation.

[1]The new version of the FGT proposed in [Greengard and Sun 1998] reduces the cost of translating the Hermite series. The new version is based on replacing the Hermite and Taylor expansions with an expansion in terms of exponentials (plane waves). Because of this the translation operator becomes diagonal. This reduces the cost of translation from $\mathcal{O}(d(2n+1)^d p^{d+1})$ to $\mathcal{O}(3dp^d)$. In any case the cost of translation grows exponentially with dimension. Also the details of the scheme are presented only for $d \leq 3$.

This paper builds upon two papers [Yang et al. 2005; Yang et al. 2003] where the core IFGT algorithm was first presented in brief. It adds details on the automatic choice of the algorithm parameters, presents a tighter error bound and provides a careful comparison with the original FGT algorithm, and clearly explains the differences between the algorithms. The error bound proposed in the original paper was not tight to be useful in practice. Also the paper did not suggest any strategy for choosing the parameters to achieve the desired bound. An attempt for automatically choosing the parameters was suggested by [Lang et al. 2005]. However the strategy was not the optimal one. In the first place the error bound had to be tightened to make the IFGT algorithm useful in practice. In this paper we use pointwise error bounds for each source point. This naturally leads to tighter error bounds and a good strategy for choosing the parameters. Another novel idea in this paper is that a different truncation number is chosen for each data point depending on its distance from the cluster center. A good consequence of this strategy is that only a few points at the boundary of the clusters have high truncation numbers.

The rest of the paper is organized as follows. In Section 2 we introduce the key technical concepts used in the IFGT algorithm. More specifically, we discuss the multivariate Taylor series which we use to factorize the Gaussian, the multi-index notation, and the space subdivision scheme based on the k-center clustering algorithm. In Section 3 we describe our improved fast Gauss transform and present computational and space complexity. In Section 4 we propose a strategy to choose the free parameters based on pointwise error bounds. In Section 5 we describe related work and how the proposed method differs from the originally proposed FGT. In Section 6 we present numerical results of our algorithm and demonstrate the speedup achieved for higher dimensions. As an example, in Section 7, we show how the IFGT can be used to accelerate multivariate kernel density estimation. We conclude the paper in Section 8. The appendices at the end of the paper discuss a few technical issues.

2. PRELIMNARIES

Before we discuss the IFGT we first discuss the multivariate Taylor's series expansion, multi-index notation, and our space subdivision scheme.

2.1 Multidimensional Taylor's Series

The factorization of the multivariate Gaussian and the evaluation of the error bounds are based on the multidimensional Taylor's series and Lagrange's evaluation of the remainder which we state here without the proof.

THEOREM 2.1. *For any point $x_* \in \mathbf{R}^d$, let $I \subset \mathbf{R}^d$ be an open set containing the point x_*. Let $f : I \to \mathbf{R}$ be a real valued function which is n times partially differentiable on I. Then for any $x = (x_1, x_2, \ldots, x_d) \in I$, there is a $\theta \in \mathbf{R}$ with $0 < \theta < 1$ such that*

$$f(x) = \sum_{k=0}^{n-1} \frac{1}{k!} \left[(x - x_*) \cdot \nabla\right]^k f(x_*) + \frac{1}{n!} \left[(x - x_*) \cdot \nabla\right]^n f(x_* + \theta(x - x_*)),$$

where the operator $\nabla = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_d}\right)$.

Based on the above theorem we have the following corollary which gives the multivariate Taylor's series expansion of the exponential function $e^{2(x-x_*).(y-x_*)/h^2}$.

COROLLARY 2.1. *Let $B_{r_x}(x_*)$ be a open ball of radius r_x with center $x_* \in \mathbf{R}^d$, i.e., $B_{r_x}(x_*) = \{x \in \mathbf{R}^d : \|x - x_*\| < r_x\}$. Let $h \in \mathbf{R}^+$ be a positive constant and $y \in \mathbf{R}^d$ be a fixed point such that $\|y - x_*\| < r_y$. For any $x \in B_{r_x}(x_*)$ and any non-negative integer p the function $f(x) = e^{2(x-x_*).(y-x_*)/h^2}$ can be written as*

$$f(x) = e^{2(x-x_*).(y-x_*)/h^2} = \sum_{k=0}^{p-1} \frac{2^k}{k!} \left[\left(\frac{x - x_*}{h} \right) \cdot \left(\frac{y - x_*}{h} \right) \right]^k + R_p(x), \tag{3}$$

and the residual $R_p(x)$ is bounded as follows.

$$R_p(x) \le \frac{2^p}{p!} \left(\frac{\|x - x_*\|}{h} \right)^p \left(\frac{\|y - x_*\|}{h} \right)^p e^{2\|x-x_*\|\|y-x_*\|/h^2}. \tag{4}$$

$$< \frac{2^p}{p!} \left(\frac{r_x r_y}{h^2} \right)^p e^{2r_x r_y/h^2}. \tag{5}$$

PROOF. Let us define a new function $g(x) = e^{2[x.(y-x_*)]/h^2}$. Using the result

$$[(x - x_*) \cdot \nabla]^k g(x_*) = 2^k e^{2[x_*.(y-x_*)]/h^2} \left[\left(\frac{x - x_*}{h} \right) \cdot \left(\frac{y - x_*}{h} \right) \right]^k$$

and Theorem 2.1, we have for any $x \in B_{r_x}(x_*)$ there is a $\theta \in \mathbf{R}$ with $0 < \theta < 1$ such that

$$\begin{aligned} g(x) &= e^{2[x_*.(y-x_*)]/h^2} \left\{ \sum_{k=0}^{p-1} \frac{2^k}{k!} \left[\left(\frac{x - x_*}{h} \right) \cdot \left(\frac{y - x_*}{h} \right) \right]^k \right. \\ &\quad \left. + \frac{2^p}{p!} \left[\left(\frac{x - x_*}{h} \right) \cdot \left(\frac{y - x_*}{h} \right) \right]^p e^{2\theta[(x-x_*).(y-x_*)]/h^2} \right\}. \end{aligned}$$

Hence

$$f(x) = e^{2(x-x_*).(y-x_*)/h^2} = \sum_{k=0}^{p-1} \frac{2^k}{k!} \left[\left(\frac{x - x_*}{h} \right) \cdot \left(\frac{y - x_*}{h} \right) \right]^k + R_p(x),$$

where,

$$R_p(x) = \frac{2^p}{p!} \left[\left(\frac{x - x_*}{h} \right) \cdot \left(\frac{y - x_*}{h} \right) \right]^p e^{2\theta[(x-x_*).(y-x_*)]/h^2}.$$

Using the Cauchy-Schwartz inequality $x \cdot y \le \|x\|\|y\|$ the remainder is bounded as follows.

$$\begin{aligned} R_p(x) &= \frac{2^p}{p!} \left[\left(\frac{x - x_*}{h} \right) \cdot \left(\frac{y - x_*}{h} \right) \right]^p e^{2\theta[(x-x_*).(y-x_*)]/h^2}, \\ &\le \frac{2^p}{p!} \left(\frac{\|x - x_*\|}{h} \right)^p \left(\frac{\|y - x_*\|}{h} \right)^p e^{2\theta\|x-x_*\|\|y-x_*\|/h^2}, \\ &\le \frac{2^p}{p!} \left(\frac{\|x - x_*\|}{h} \right)^p \left(\frac{\|y - x_*\|}{h} \right)^p e^{2\|x-x_*\|\|y-x_*\|/h^2} [\text{Since } \; 0 < \theta < 1], \\ &< \frac{2^p}{p!} \left(\frac{r_x r_y}{h^2} \right)^p e^{2r_x r_y/h^2} \quad [\text{Since } \|x - x_*\| < r_x \text{ and } \|y - x_*\| < r_y]. \end{aligned} \tag{6}$$

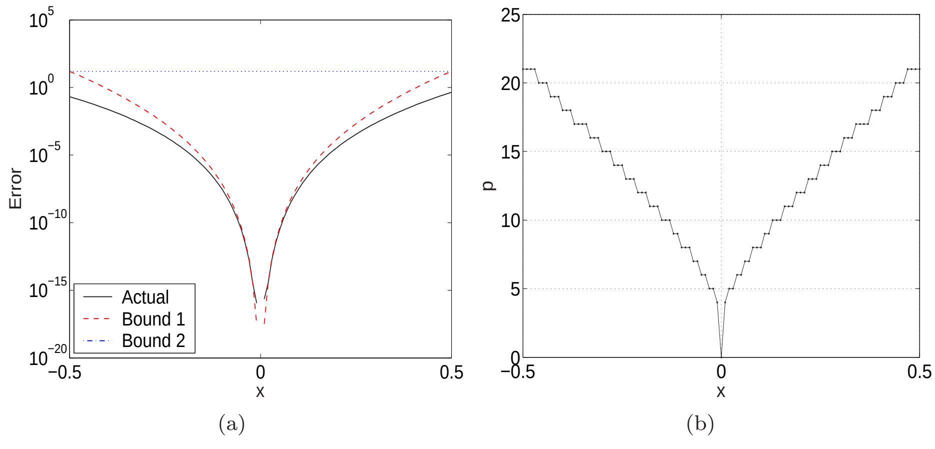


Fig. 1. (a) The actual residual (solid line) and the bound (dashed line) given by Equation 4 as a function of x. [$x_* = 0$, $y = 1.0$, $h = 0.5$, $r_x = 0.5$, $r_y = 1.0$, and $p = 10$]. The residual is minimum at $x = x_*$ and increases as x moves away from x_*. The dotted line shows the very pessimistic bound which is independent of x (Equation 15) used in the original IFGT. (b) The truncation number p required as the function of x so that the error is less than 10^{-6}.

□

Remark: The error bound which we have in Equation 4 is independent of the dimensionality d. Figure 1(a) compares the actual residual and the bound given by Equation 4 as a function of x, for $p = 10$ and $d = 1$. The actual residual $R_p(x)$ is minimum at $x = x_*$ and increases as x moves away from x_*. The dashed line shows the bound given by Equation 4. It can be seen that the bound is pretty tight. The dotted line is the bound which is independent of x (Equation 15). The bound is tight only at $\|x - x_*\| = r_x$. It can be seen that the bound is very pessimistic for $\|x - x_*\| < r_x$. A consequence of this is that a lower truncation number p can achieve a given error, depending on the magnitude of $\|x - x_*\|$. Figure 1(b) shows the truncation number p required as the function of x so that the error is less than 10^{-6}. It can be seen that for points close to x_* we need a very small truncation number compared to points far from the center. The original IFGT and the FGT algorithms used the same truncation number for all the points in the open ball. The truncation number was chosen based on the points at the boundary. However our current approach adaptively chooses p based $\|x - x_*\|$.

2.2 Multi-index Notation

A multi-index $\alpha = (\alpha_1, \alpha_2 \ldots, \alpha_d)$ is a d-tuple of nonnegative integers. The length of the multi-index α is defined as $|\alpha| = \alpha_1 + \alpha_2 + \ldots + \alpha_d$. The factorial of α is defined as $\alpha! = \alpha_1!\alpha_2!\ldots\alpha_d!$. For any multi-index $\alpha \in \mathbf{N}^d$ and $x = (x_1, x_2, \ldots, x_d) \in \mathbf{R}^d$ the d-variate monomial x^α is defined as $x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_d^{\alpha_d}$. x^α is of degree n if $|\alpha| = n$. The total number of d-variate monomials of degree n is $\binom{n+d-1}{d-1}$. The total number of d-variate monomials of degree less than or equal to n is $r_{nd} = \sum_{k=0}^{n} \binom{k+d-1}{d-1} = \binom{n+d}{d}$.

We denote by Π_n^d the space of all real polynomials in d variables of total degree less than or equal to n; its dimensionality is $r_{nd} = \binom{n+d}{d}$. To store, manipulate and evaluate the multivariate polynomials, we consider the monomial representation of polynomials. A polynomial $p \in \Pi_n^d$ can be written as

$$p(x) = \sum_{|\alpha| \leq n} C_\alpha x^\alpha, \qquad C_\alpha \in \mathbf{R}.$$

It is computationally convenient and efficient to stack all the coefficients into a vector. To store all the r_{nd} coefficients C_α in a vector of length r_{nd}, we sort the coefficient terms according to *Graded lexicographic order*. "Graded" refers to the fact that the total degree $|\alpha|$ is the main criterion. Graded lexicographic ordering means that the multi-indices are arranged as

$$\begin{array}{l}(0,0,\ldots,0),\ (1,0,\ldots,0),\ (0,1,\ldots,0),\ \ldots,\ (0,0,\ldots,1),\\ (2,0,\ldots,0),\ (1,1,\ldots,0),\ \ldots,\ (0,0,\ldots,2),\ \ldots\ldots,\ (0,0,\ldots,n).\end{array}$$

Let $x, y \in \mathbf{R}^d$ and $v = x \cdot y = x_1 y_1 + \ldots + x_d y_d$. Then using the multi-index notation v^n can be written as,

$$v^n = \sum_{|\alpha|=n} \frac{n!}{\alpha!} x^\alpha y^\alpha. \tag{7}$$

2.3 Space sub-division

In the IFGT we will appropriately cluster source points and evaluate their contributions using an expression that involves the Taylor's series. Accordingly we need a strategy to choose a set of centers about which to expand the Taylor's series, i.e., we need to subdivide the space. We model the space subdivision task as a k-center problem, which is defined as follows:

k-center problem *Given a set of N points in d dimensions and a predefined number of the clusters k, find a partition of the points into clusters $S_1, \ldots, S_k$, and also the cluster centers $c_1, \ldots, c_k$, so as to minimize the cost function – the maximum radius of clusters,* $\max_i \max_{x \in S_i} \|x - c_i\|$.

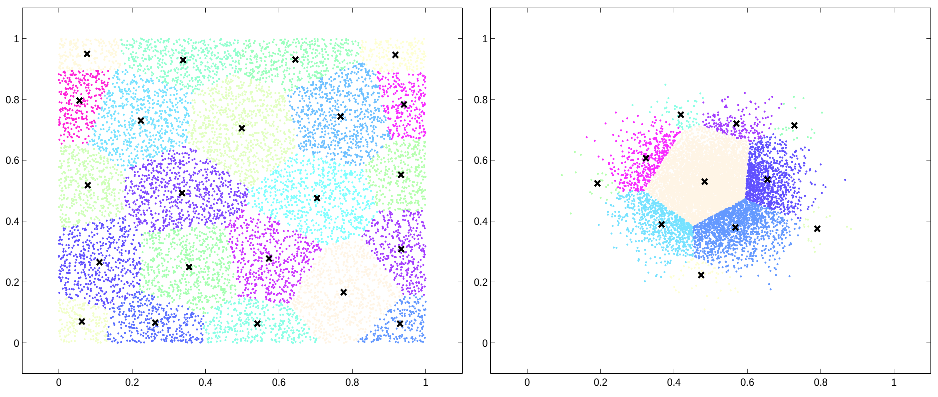
The k-center problem is known to be NP-hard [Bern and Eppstein 1997]. [Gonzalez 1985] proposed a very simple greedy algorithm, called *farthest-point clustering*, and proved that it gives an approximation factor of 2. Initially pick an arbitrary point v_0 as the center of the first cluster and add it to the center set C. Then for $i = 1$ to k do the following: at step i, for every point v, compute its distance to the set C: $d_i(v, C) = \min_{c \in C} \|v - c\|$. Let v_i be the point that is farthest from C, i.e., the point for which $d_i(v_i, C) = \max_v d_i(v, C)$. Add v_i to set C. Report the points $v_0, v_1, \ldots, v_{k-1}$ as the cluster centers. Each point is assigned to its nearest center.

Gonzalez proved the following 2-approximation theorem for the farthest-point clustering algorithm [Gonzalez 1985]:

THEOREM 2.2. *For k-center clustering, the farthest-point clustering computes a partition with maximum radius at most twice the optimum.*

PROOF. For completeness, we provide a simple proof for the above theorem. First note that the radius of the farthest-point clustering solution by definition is

$$d_k(v_k, C) = \max_v \min_{c \in C} \|v - c\|.$$



(a) (b)

Fig. 2. (a) Using the farthest point clustering algorithm 10,000 points uniformly distributed in a unit square are divided into 22 clusters with the maximum radius of the clusters being 0.2. (b) 10,000 points normally distributed in a unit square are divided into 11 clusters with the maximum radius of the clusters being 0.2.

In the optimal k-center case, two of these $k+1$ points, say v_i and v_j, must be in a same cluster centered at c by the pigeon hole principle. Observe that the distance from each point to the set C does not increase as the algorithm progresses. Therefore $d_k(v_k, C) \leq d_i(v_k, C)$ and $d_k(v_k, C) \leq d_j(v_k, C)$. Also by definition, we have $d_i(v_k, C) \leq d_i(v_i, C)$ and $d_j(v_k, C) \leq d_j(v_j, C)$. So we have

$$\|v_i - c\| + \|v_j - c\| \geq \|v_i - v_j\| \geq d_k(v_k, C),$$

by the triangle inequality. Since $\|v_i - c\|$ and $\|v_j - c\|$ are both at most the optimal radius δ, we have the radius of the farthest-point clustering solution $d_k(v_k, C) \leq 2\delta$. □

[Hochbaum and Shmoys 1985] proved that the factor 2 cannot be improved unless $P = NP$.

The direct implementation of farthest-point clustering has running time $\mathcal{O}(Nk)$. [Feder and Greene 1988] give a two-phase algorithm with optimal running time $\mathcal{O}(N \log k)$. The first phase of their algorithm clusters points into rectangular boxes using [Vaidya 1986]'s *box decomposition*– a sort of quadtree in which cubes are shrunk to bounding boxes before splitting. The second phase resembles the farthest-point clustering on a sparse graph that has a vertex for each box. In practice, the initial point has little influence on the final approximation radius, if number of the points is sufficiently large.

Figure 2 displays the results of farthest-point algorithm on a sample two dimensional data-set. After the end of the clustering procedure the center of each cluster is recomputed as the mean of all the points lying in each cluster. The farthest point algorithm is progressive. This means that if we have k centers and we wish to compute the $(k+1)^{th}$ center, the first k centers do not change.

3. IMPROVED FAST GAUSS TRANSFORM

Having established the Taylor's series expansion and the farthest point clustering algorithm for k-center clustering, we are now ready to present the IFGT. The method relies on the expansion of the Gaussian using the truncated Taylor's series expansion. For any point $x_* \in \mathbf{R}^d$ the Gauss Transform at y_j can be written as,

$$\begin{aligned} G(y_j) &= \sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2/h^2}, \\ &= \sum_{i=1}^{N} q_i e^{-\|(y_j - x_*)-(x_i - x_*)\|^2/h^2}, \\ &= \sum_{i=1}^{N} q_i e^{-\|x_i - x_*\|^2/h^2} e^{-\|y_j - x_*\|^2/h^2} e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2}. \end{aligned} \tag{8}$$

In Equation 8 the first exponential inside the summation $e^{-\|x_i - x_*\|^2/h^2}$ depends only on the source coordinates x_i. The second exponential $e^{-\|y_j - x_*\|^2/h^2}$ depends only on the target coordinates y_j. However for the third exponential $e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2}$ the source and target are entangled. The crux of the algorithm is to separate this entanglement via the Taylor's series expansion of the exponentials.

3.1 Factorization

Using Corollary 2.1 the series expansion for $e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2}$ can be written as,

$$e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2} = \sum_{n=0}^{p_i - 1} \frac{2^n}{n!} \left[\left(\frac{y_j - x_*}{h} \right) \cdot \left(\frac{x_i - x_*}{h} \right) \right]^n + error_{p_i}. \tag{9}$$

The truncation number p_i for each source x_i is chosen based on the prescribed error and the distance from the expansion center. A strategy for choosing p_i is discussed in a later section. Using the multi-index notation (Equation 7), this expansion can be written as,

$$e^{2(y_j - x_*)\cdot(x_i - x_*)/h^2} = \sum_{|\alpha| \le p_i - 1} \frac{2^\alpha}{\alpha!} \left(\frac{y_j - x_*}{h} \right)^\alpha \left(\frac{x_i - x_*}{h} \right)^\alpha + error_{p_i}. \tag{10}$$

Ignoring error terms for now $G(y_j)$ can be approximated as,

$$\hat{G}(y_j) = \sum_{i=1}^{N} q_i e^{-\|x_i - x_*\|^2/h^2} e^{-\|y_j - x_*\|^2/h^2} \left[\sum_{|\alpha| \le p_i - 1} \frac{2^\alpha}{\alpha!} \left(\frac{y_j - x_*}{h} \right)^\alpha \left(\frac{x_i - x_*}{h} \right)^\alpha \right]. \tag{11}$$

Let $p_{max} = \max_i p_i$ and $\mathbf{1}_{|\alpha| \le p_i - 1}$ be an indicator function for $|\alpha| \le p_i - 1$, that is,

$$\mathbf{1}_{|\alpha| \le p_i - 1} = \begin{cases} 1 & \text{if } |\alpha| \le p_i - 1 \\ 0 & \text{if } |\alpha| > p_i - 1 \end{cases}.$$

3.2 Regrouping

Rearranging the terms Equation 11 can be written as

$$\begin{aligned}\hat{G}(y_j) &= \sum_{|\alpha|\leq p_{max}-1} \left[\frac{2^\alpha}{\alpha!}\sum_{i=1}^{N} q_i e^{-\|x_i-x_*\|^2/h^2}\left(\frac{x_i-x_*}{h}\right)^\alpha \mathbf{1}_{|\alpha|\leq p_i-1}\right] \\ &\quad e^{-\|y_j-x_*\|^2/h^2}\left(\frac{y_j-x_*}{h}\right)^\alpha, \\ &= \sum_{|\alpha|\leq p_{max}-1} C_\alpha e^{-\|y_j-x_*\|^2/h^2}\left(\frac{y_j-x_*}{h}\right)^\alpha,\end{aligned}$$

where,

$$C_\alpha = \frac{2^\alpha}{\alpha!}\sum_{i=1}^{N} q_i e^{-\|x_i-x_*\|^2/h^2}\left(\frac{x_i-x_*}{h}\right)^\alpha \mathbf{1}_{|\alpha|\leq p_i-1}.$$

The coefficients C_α can be evaluated separately is $\mathcal{O}(N)$. Evaluation of $\widehat{G}_r(y_j)$ at M points is $\mathcal{O}(M)$. Hence the computational complexity has reduced from the quadratic $\mathcal{O}(NM)$ to the linear $\mathcal{O}(N+M)$. Detailed analysis of the computational complexity will be provided later.

3.3 Space subdivision

Thus far, we have used the Taylor's series expansion about a certain point x_*. However if we use the same x_* for all the points we typically would require very high truncation numbers since the Taylor's series is valid only in a small open ball around x_*. We use an data adaptive space partitioning scheme like the farthest point clustering algorithm to divide the N sources into K clusters, S_k for $k = 1, \ldots, K$ with c_k being the center of each cluster. The Gauss transform can be written as,

$$\hat{G}(y_j) = \sum_{k=1}^{K}\sum_{|\alpha|\leq p_{max}-1} C_\alpha^k e^{-\|y_j-c_k\|^2/h^2}\left(\frac{y_j-c_k}{h}\right)^\alpha,$$

where,

$$C_\alpha^k = \frac{2^\alpha}{\alpha!}\sum_{x_i\in S_k} q_i e^{-\|x_i-c_k\|^2/h^2}\left(\frac{x_i-c_k}{h}\right)^\alpha \mathbf{1}_{|\alpha|\leq p_i-1}.$$

3.4 Rapid decay of the Gaussian

Since the Gaussian decays very rapidly a further speedup is achieved if we ignore all the sources belonging to a cluster if the cluster is greater than a certain distance from the target point, $\|y_j - c_k\| > r_y^k$. The cluster cutoff radius depends on the desired precision ϵ. So now the Gauss transform is evaluated as

$$\hat{G}(y_j) = \sum_{\|y_j-c_k\|\leq r_y^k}\sum_{|\alpha|\leq p_{max}-1} C_\alpha^k e^{-\|y_j-c_k\|^2/h^2}\left(\frac{y_j-c_k}{h}\right)^\alpha, \qquad (12)$$

where,

$$C_\alpha^k = \frac{2^\alpha}{\alpha!} \sum_{x_i \in S_k} q_i e^{-\|x_i - c_k\|^2/h^2} \left(\frac{x_i - c_k}{h}\right)^\alpha \mathbf{1}_{|\alpha| \le p_i - 1}. \tag{13}$$

3.5 Comparison with the FGT factorization

The general fast multipole methods (FMM) [Greengard 1988], of which the FGT is a special case use two kind of expansions of the potential function: the far-field expansion and the local expansion. The FGT uses the Hermite expansion of the Gaussian for the far-field and the Taylor expansion of the Gaussian (which is obtained by interchanging y and x_i in the Hermite expansion) as the local expansion. The following are the two expansions used by the FGT.

$$e^{-\|y-x_i\|^2/h^2} = \sum_{\alpha \ge 0} \left[\frac{1}{\alpha!}\left(\frac{x_i - x_*}{h}\right)^\alpha\right] h_\alpha\left(\frac{y - x_*}{h}\right) \text{ [far-field Hermite expansion]},$$

$$e^{-\|y-x_i\|^2/h^2} = \sum_{\beta \ge 0} \left[\frac{1}{\beta!} h_\beta\left(\frac{x_i - x_*}{h}\right)\right] \left(\frac{y - x_*}{h}\right)^\beta \text{ [local Taylor expansion]},$$

where $h_\alpha(y)$ are the multivariate Hermite functions [Greengard and Strain 1991]. The real benefit of FMM is for singular potential functions whose forces are long ranged and locally non-smooth, hence it is necessary to make use of the tree data structures, local expansions, far-field expansions and translation operators between representations. Translation between local and far-filed representations is expansive, but unavoidable in the case of singular potential functions.

The Gaussian is a *regular potential*. For the IFGT we represent the Gaussian into a product of two Gaussians and an exponential (Equation 8), and then use one factorization for the exponential using the Taylor's series. The factorization used by the IFGT can be written as follows.

$$e^{-\|y-x_i\|^2/h^2} = \sum_{|\alpha| \ge 0} \left[\frac{2^\alpha}{\alpha!} e^{-\|x_i - x_*\|^2/h^2} \left(\frac{x_i - x_*}{h}\right)^\alpha\right] e^{-\|y - x_*\|^2/h^2} \left(\frac{y - x_*}{h}\right)^\alpha.$$

This factorization has the property that it is both a good far-field and and local expansion, and in fact does a very good job in the whole domain. Thereby it avoids the need for two different representations and the expensive translation operation.

Figure 3 shows the absolute value of the actual error between the one dimensional gaussian ($e^{-(x_i - y)/h^2}$) and the different series approximations. The Gaussian was centered at $x_i = 0$. All the series were expanded about $x_* = 1.0$. $p = 5$ terms were retained in the series approximation. From the plot it can be seen that the Hermite expansion is essentially a far field expansion which gives better approximation as we move far away from x_*. The Taylor expansion is a local expansion giving good approximation only for a region very close to x_*. *The Taylor expansion used by the IFGT can serve both as the far field as well as the local expansion.*

3.6 Computational Complexity

The farthest point clustering has running time $\mathcal{O}(N \log K)$ [Feder and Greene 1988]. Computing the cluster coefficients C_α^k for all the clusters is of $\mathcal{O}(Nr_{(p_{max}-1)d})$,

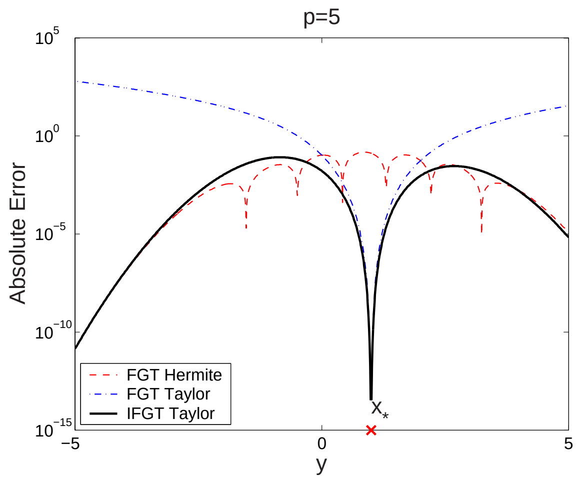


Fig. 3. The absolute value of the actual error between the one dimensional gaussian ($e^{-(x_i-y)/h^2}$) and different series approximations. The Gaussian was centered at $x_i = 0$. All the series were expanded about $x_* = 1.0$. $p = 5$ terms were retained in the series approximation.

where $r_{(p_{max}-1)d} = \binom{p_{max}+d-1}{d}$ is the total number of d-variate monomials of degree less than or equal to $p_{max} - 1$. Computing $\hat{G}(y_j)$ is of $\mathcal{O}(Mnr_{(p_{max}-1)d})$ where n if the maximum number of neighbor clusters (depends on the bandwidth h and the error ϵ) which influence the target. Hence the total computational complexity is

$$\mathcal{O}(N \log K + Nr_{(p_{max}-1)d} + Mnr_{(p_{max}-1)d}).$$

Assuming $M = N$, the complexity is $\mathcal{O}(\left[\log K + (1+n)r_{(p_{max}-1)d}\right] N)$. The constant term depends on the dimensionality, the bandwidth, and the accuracy required. The number of terms $r_{(p_{max}-1)d}$ is asymptotically polynomial in d. For $d \to \infty$ and moderate p, the number of terms is approximately d^p.

A different truncation number is chosen for each data point depending on its distance from the cluster center. A good consequence of this strategy is that only a few points at the boundary of the clusters will have high truncation numbers. Theoretically we expect to get a much better speed up since for many points $p_i < p_{max}$. However some computation resources are used in determining the truncation numbers based on the distribution of the data points. As a result in lower dimensions the proposed method shows only a slight improvement in speedup, compared to using the same truncation number for all points. The speedup is more noticeable especially in higher dimensions where the advantage of a lower order expansion is greater.

3.7 Space Complexity

For each cluster we need to store $r_{(p_{max}-1)d}$ coefficients. So the storage complexity is $\mathcal{O}(Kr_{(p_{max}-1)d} + N + M)$.

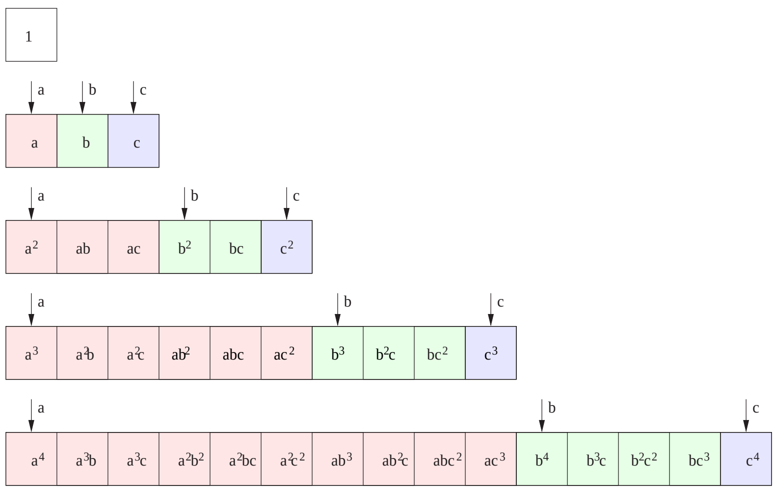


Fig. 4. Efficient expansion of multivariate polynomials.

3.8 Horner's rule

One of the benefits of the graded lexicographic order is that the expansion of multivariate polynomials can be performed efficiently. Evaluating each d-variate monomial of degree n directly requires n multiplications. Hence direct evaluation of all d-variate monomials of degree less than or equal to n requires $\sum_{k=0}^{n} k\binom{k+d-1}{d-1}$ multiplications. The storage requirement is r_{nd}. However, efficient evaluation using the Horner's rule requires $r_{nd}-1$ multiplications. The required storage is r_{nd} (See Table I).

For a d-variate polynomial of order n, we can store all terms in a vector of length r_{nd}. Starting from the order zero term (constant 1), we take the following approach. Assume we have already evaluated terms of order $k-1$. We use an array of size d to record the positions of the d leading terms (the simple terms such as a^{k-1}, b^{k-1}, c^{k-1}, . . . in Figure 4) in the terms of order $k-1$. Then terms of order k can be obtained by multiplying each of the d variables with all the terms between the variables leading term and the end, as shown in the Figure 4 The positions of the d leading terms are updated respectively. The required storage is r_{nd} and the computations of the terms require $r_{nd}-1$ multiplications.

3.9 Partial distance

For each cluster we need to find the clusters which are within a certain radius to it. Computing partial distances helps to reduce the computational burden in nearest-neighbor searches in high dimensional spaces. By *partial distance*, we calculate the distance using some subset r of the full d dimensions. If this partial distance is too great we do not compute distances any further. The partial distance is strictly nondecreasing as we add the contributions from more and more dimensions.

n	2	4	6	8	10	12	15	20
Direct d=2	8	40	112	240	440	728	1360	3080
Efficient d=2	5	14	27	44	65	90	135	230
Direct d=3	15	105	378	990	2145	4095	9180	26565
Efficient d=3	9	34	83	164	285	454	815	1770
Direct d=6	48	720	4752	20592	68640	190944	697680	3946800
Efficient d=6	27	209	923	3002	8007	18563	54263	230229
Direct d=10	120	3640	43680	318240	1679600	7054320	44574000	546273000
Efficient d=10	65	1000	8007	43757	184755	646645	3268759	30045014

Table I. Number of multiplication required for the direct and the efficient method for evaluating all d-variate monomials of degree less than or equal to n.

4. CHOOSING THE PARAMETERS BASED ON POINT WISE ERROR BOUNDS

A criticism [Lang et al. 2005] of the original IFGT [Yang et al. 2005] was that the error bound was too pessimistic, and too many computational resources were wasted as a consequence. Further the choice of the parameters was not automatic. In the following we present an automatic way for choosing the parameters.

Given any $\epsilon > 0$, we want to choose the following parameters, K (the number of clusters), $\{r_y^k\}_{k=1}^K$ (the cut off radius for each cluster), and $\{p_i\}_{i=1}^N$ (the truncation number for each source point x_i) such that for any target point y_j we can guarantee that

$$\frac{|\hat{G}(y_j) - G(y_j)|}{Q} \leq \epsilon,$$

where $Q = \sum_{i=1}^N |q_i|$. Let us define Δ_{ij} to be the point wise error in $\widehat{G}(y_j)$ contributed by the i^{th} source x_i. We now require that

$$|\hat{G}(y_j) - G(y_j)| = \left|\sum_{i=1}^N \Delta_{ij}\right| \leq \sum_{i=1}^N |\Delta_{ij}| \leq Q\epsilon = \sum_{i=1}^N |q_i|\epsilon.$$

One way to achieve this is to let

$$|\Delta_{ij}| \leq |q_i|\epsilon \;\; \forall i = 1, \ldots, N.$$

We choose this strategy because it helps us get tighter bounds. Let c_k be the center of the cluster to which x_i belongs. There are two different ways in which a source can contribute to the error.

The first is due to ignoring the cluster S_k if it is outside a given radius r_y^k from the target point y_j. In this case,

$$\Delta_{ij} = q_i e^{-\|y_j - x_i\|^2/h^2} \quad \text{if } \|y_j - c_k\| > r_y^k. \tag{14}$$

The second source of error is due to truncation of the Taylor's series. For all clusters which are within a distance r_y^k from the target point the error is due to the truncation of the Taylor's series after order p_i. From Equation 8 and 9 we have,

$$\Delta_{ij} = q_i e^{-\|x_i - c_k\|^2/h^2} e^{-\|y_j - c_k\|^2/h^2} error_{p_i} \quad \text{if } \|y_j - c_k\| \leq r_y^k. \tag{15}$$

Our strategy for choosing the parameters is as follows. The cutoff radius r_y^k for each cluster is chosen based on Equation 14 and the radius of each cluster r_x^k. Given r_y^k and $\|x_i - c_k\|$ the truncation number p_i for each source is chosen based on Equation 15. Towards the end we suggest a strategy to choose the number of clusters K and the maximum truncation p_{max} jointly.

4.1 Automatically choosing the cut off radius for each cluster

We ignore all sources belonging to a cluster S_k if $\|y_j - c_k\| > r_y^k$. r_y^k should be chosen such that for all sources in cluster S_k the error $|\Delta_{ij}| = |q_i| e^{-\|y_j - x_i\|^2/h^2} \leq |q_i|\epsilon$. This implies that

$$\|y_j - x_i\| > h\sqrt{\ln(1/\epsilon)}$$

Using the reverse triangle inequality, $\|a - b\| \geq \big|\|a\| - \|b\|\big|$, and the fact that $\|y_j - c_k\| > r_y^k$ and $\|x_i - c_k\| \leq r_x^k$, we have

$$\begin{aligned} \|y_j - x_i\| = \|y_j - c_k + c_k - x_i\| &= \|(y_j - c_k) - (x_i - c_k)\|, \\ &\geq \big|\|(y_j - c_k)\| - \|(x_i - c_k)\|\big|, \\ &> \big|r_y^k - r_x^k\big|. \end{aligned}$$

So in order that the error due to ignoring the faraway clusters is less than $q_i\epsilon$ we have to choose r_y^k and r_x^k such that,

$$\big|r_y^k - r_x^k\big| > h\sqrt{\ln(1/\epsilon)}.$$

If we choose $r_y^k > r_x^k$ then,

$$r_y^k > r_x^k + h\sqrt{\ln(1/\epsilon)}.$$

Let R be the maximum distance between any source and target point. For example if the data were distributed in a d-dimensional hypercube of length a, then $R \leq \sqrt{d}a$, i.e., the length of the maximum diagonal. Hence,

$$r_y^k > r_x^k + \min\left(R, h\sqrt{\ln(1/\epsilon)}\right).$$

4.2 Automatically choosing the truncation number for each source

From Corollary 2.1 we have,

$$error_{p_i} \leq \frac{2^{p_i}}{p_i!}\left(\frac{\|x_i - c_k\|}{h}\right)^{p_i}\left(\frac{\|y_j - c_k\|}{h}\right)^{p_i} e^{2\|x_i - c_k\|\|y_j - c_k\|/h^2}.$$

Hence for all sources for which $\|y_j - c_k\| \leq r_y^k$, substituting in Equation 15 we have

$$\Delta_{ij} \leq q_i \frac{2^{p_i}}{p_i!}\left(\frac{\|x_i - c_k\|}{h}\right)^{p_i}\left(\frac{\|y_j - c_k\|}{h}\right)^{p_i} e^{-(\|x_i - c_k\| - \|y_j - c_k\|)^2/h^2}. \tag{16}$$

For a given source x_i we have to choose p_i such that $|\Delta_{ij}| \leq |q_i|\epsilon$. Δ_{ij} depends both on distance between the source and the cluster center, i.e., $\|x_i - c_k\|$ and the distance between the target and the cluster center, i.e., $\|y_j - c_k\|$. The speedup is achieved because at each cluster S_k we sum up the effect of all the sources. As a result we do not have a knowledge of $\|y_j - c_k\|$ when we are using Equation 13. So

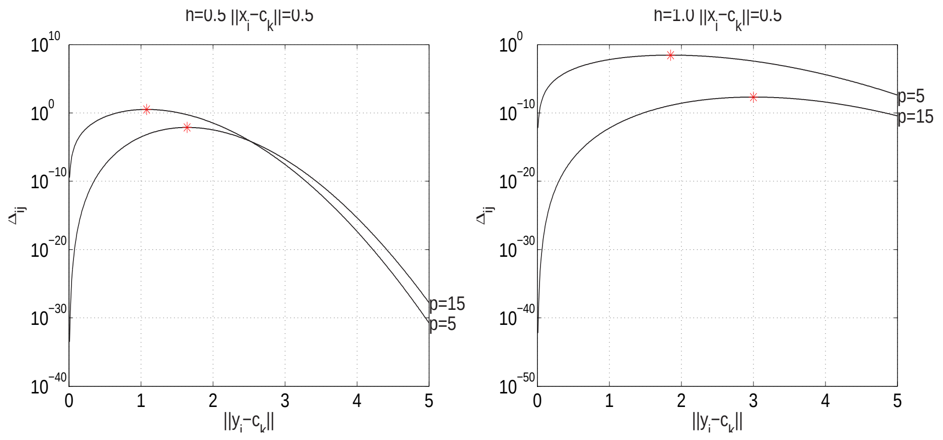


Fig. 5. The error at y_j due to source x_i, i.e., Δ_{ij} [Equation 16] as a function of $\|y_j - c_k\|$ for different values of p and for (a) $h = 0.5$ and (b) $h = 1.0$. The error increases as a function of $\|y_j - c_k\|$, reaches a maximum and then starts decreasing. The maximum is marked as '*'. $q_i = 1$ and $\|x_i - c_k\| = 0.5$.

we will have to bound the right hand side of Equation 16, such that it is independent of $\|y_j - c_k\|$. Figure 5 shows the error at y_j due to source x_i, i.e., Δ_{ij} [Equation 16] as a function of $\|y_j - c_k\|$ for different values of p and for (a) $h = 0.5$ and (b) $h = 1.0$. The error increases as a function of $\|y_j - c_k\|$, reaches a maximum and then starts decreasing. The maximum is attained at

$$\|y_j - c_k\| = \|y_j - c_k\|_* = \frac{\|x_i - c_k\| + \sqrt{\|x_i - c_k\|^2 + 2p_i h^2}}{2}.$$

Hence we choose p_i such that,

$$|\Delta_{ij}| \big|_{\|y_j - c_k\| = \|y_j - c_k\|_*} \leq |q_i|\epsilon.$$

In case $\|y_j - c_k\|_* > r_y^k$ we need to choose p_i based on r_y^k, since Δ_{ij} will be much lower there. Hence out strategy for choosing p_i is,

$$|\Delta_{ij}| \Big|_{\left[\|y_j - c_k\| = \min\left(\|y_j - c_k\|_*, r_y^k\right)\right]} \leq |q_i|\epsilon. \tag{17}$$

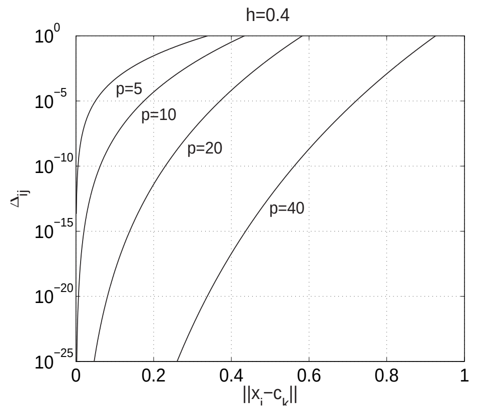
Figure 6(a) shows Δ_{ij} as a function of $\|x_i - c_k\|$ for different values of p, $h = 0.4$ and $\|y_j - c_k\| = \min\left(\|y_j - c_k\|_*, r_y^k\right)$. Figure 6(b) shows the truncation number p_i required to achieve an error of $\epsilon = 10^{-3}$.

4.3 Automatically choosing the number of clusters

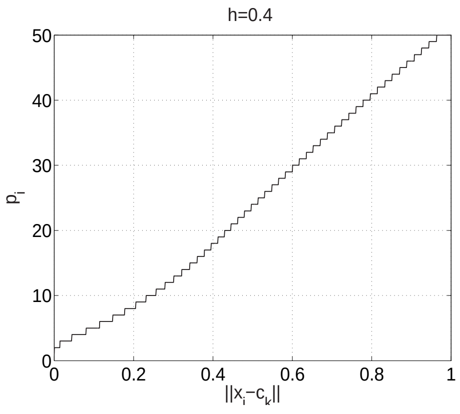
Our strategy for choosing the number of clusters is optimized for uniform distribution of the source points. The total computational complexity assuming $M = N$ is $\mathcal{O}(cN)$. The constant term is given by

$$c = \log K + (1 + n) r_{(p_{max}-1)d}. \tag{18}$$

p_{max} and n are both functions of K. We choose the number of clusters K for which c is minimum. The truncation number p_{max} is a function of the maximum



(a)



(b)

Fig. 6. The error at y_j due to source x_i, i.e., Δ_{ij} [Equation 16] as a function of $\|x_i - c_k\|$ for different values of p, $h = 0.4$ and $\|y_j - c_k\| = \min\ \ \|y_j - c_k\|_*, r_y^k\ \ .$ (b) The truncation number p_i required to achieve an error of $\epsilon = 10^{-3}$.

cluster radius r_x, implicitly via Equation 17. If the source and the target points are uniformly distributed in a unit hypercube then $r_x \sim K^{-1/d}$. [2] The number of influential neighbor clusters is roughly $n \sim (r/r_x)^d$, where $r = h\sqrt{\ln(1/\epsilon)}$ is the cutoff radius.

Figure 7 shows the constant c as function K. Initially the constant c decreases because as K increases the maximum cluster radius r_x decreases, leading to a smaller truncation number p_{max}. However after a certain point the growth in K dominates the decrease in p_{max}. The optimum K can be found by differentiating Equation 18 w.r.t. K and setting it to zero. However since the dependence of p_{max} on K is implicit it is difficult to derive an analytic expression for K. A simple strategy as outlined in Algorithm 1 is to evaluate c for a range of values of K and choose the one for which c is minimum.

We optimized the number of clusters and the maximum truncation number assuming a uniform distribution of source points. However if the data is clustered the truncation number can be smaller. Once we have chosen K we run the farthest point clustering algorithm which will give us the actual maximum cluster radius. Based on this we can determine the actual maximum truncation number required.

The algorithm is summarized in Algorithm 2 and Figure 8(a). Figure 8(b) shows the truncation number p_i required for each data point. Dark shade indicates a higher truncation number. Note that for points close to the cluster center the truncation number required is less than that at the boundary of the cluster.

[2] If the data lies on a lower dimensional manifold, as usually is the case for structured data in high dimensions, we use the relation $r_x \sim K^{-1/d_{eff}}$. d_{eff} is the actual intrinsic dimensionality of the data.

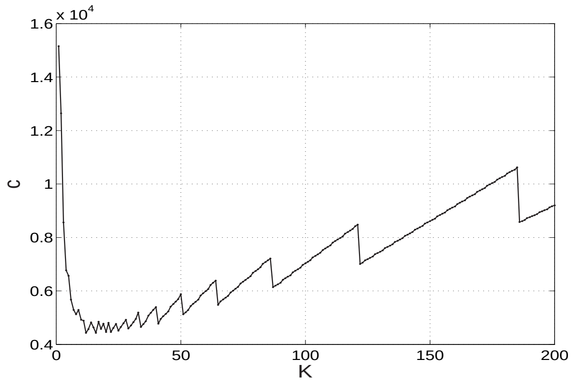


Fig. 7. The constant $c = \log K + (1+n)r_{(p_{max}-1)d}$ as function K. $d = 2$, $h = 0.3$, and $\epsilon = 10^{-6}$.

Algorithm 1: Choosing the parameters for the IFGT.

```
Input  : $d$ (dimension)
         $h$ (bandwidth)
         $\epsilon$ (error)
         $R$ (range of the data)
Output: $K$ (number of clusters)
         $r$ (cutoff radius)
         $p_{max}$ (maximum truncation number)
Define
$\delta(p,a,b) = \frac{1}{p!}\left(\frac{2ab}{h^2}\right)^p e^{-(a-b)^2/h^2}$, $b_*(a,p) = \frac{a+\sqrt{a^2+2ph^2}}{2}$, and $r_{pd} = \binom{p-1+d}{d}$;
Choose the cutoff radius $r \leftarrow \min(R, h\sqrt{\ln(1/\epsilon)})$;
Choose $K_{limit} \leftarrow \lceil 20R/h \rceil$ ;
for $k \leftarrow 1 : K_{limit}$ do
    compute an estimate of the maximum cluster radius as $r_x \leftarrow k^{-1/d}$;
    compute an estimate of the number of neighbors as $n \leftarrow \min\left[(r/r_x)^d, k\right]$;
    choose $p[k]$ such that $\delta(p = p[k], a = r_x, b = \min\left[b_*(r_x, p[k]), r + r_x\right]) \leq \epsilon$;
    compute the constant term $c[k] \leftarrow \log k + (1+n)r_{(p[k]-1)d}$
end
choose $K \leftarrow k_*$ for which $c[k_*]$ is minimum. $p_{max} \leftarrow p[k_*]$.
```

5. RELATED WORK

In this section we briefly discuss the other methods available for efficient computation of the Gauss transform. In particular we elucidate in detail on how our current method differs from the fast Gauss transform, as there appears to be some interest in this issue.

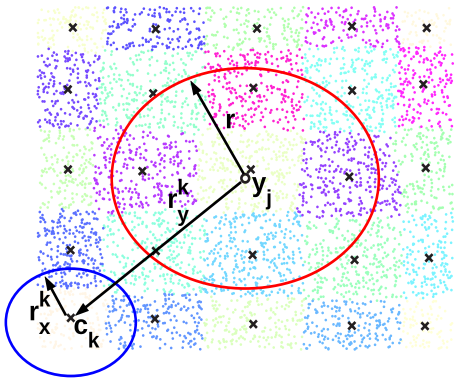
```
Algorithm 2: The improved fast Gauss transform.
  Input :
  $x_i \in \mathbf{R}^d$ $i = 1, \dots, N$ /* $N$ sources in $d$ dimensions.          */
  $q_i \in \mathbf{R}$ $i = 1, \dots, N$ /* source weights.                         */
  $h \in \mathbf{R}^+$ $i = 1, \dots, N$ /* source bandwidth.                       */
  $y_j \in \mathbf{R}^d$ $j = 1, \dots, M$ /* $M$ targets in $d$ dimensions.        */
  $\epsilon > 0$ /* Desired error.                                                  */
  Output: Computes an approximation $\hat{G}(y_j)$ to $G(y_j) = \sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2/h^2}$.
          such that the $|\hat{G}(y_j) - G(y_j)|Q \leq \epsilon$, where $Q = \sum_{i=1}^{N} |q_i|$.
  Step 0 Define $\delta(p, a, b) = \frac{1}{p!}\left(\frac{2ab}{h^2}\right)^p e^{-(a-b)^2/h^2}$ and $b_*(a, p) = \frac{a + \sqrt{a^2 + 2ph^2}}{2}$;
  Step 1 Choose the cutoff radius $r$, the number of clusters $K$, and the
         maximum truncation number $p_{max}$ using Algorithm 1 ;
  Step 2 Divide the $N$ sources into $K$ clusters, $\{S_k\}_{k=1}^{K}$, using the Feder and
         Greene's farthest-point clustering algorithm. Let $c_k$ and $r_x^k$ be the center and
         radius respectively of the $k^{th}$ cluster. Let $r_x = \max_k \left(r_x^k\right)$ be the maximum
         cluster radius ;
  Step 3 Update the maximum truncation number based on the actual $rx$, i.e.,
         choose $p_{max}$ such that $\delta(p = p_{max}, a = rx, \min\left[b_*(rx, p_{max}), r + rx\right]) \leq \epsilon$
  Step 4 For each cluster $S_k$ with center $c_k$ compute the coefficients $C_\alpha^k$.
         $C_\alpha^k = \frac{2^\alpha}{\alpha!} \sum_{x_i \in S_k} q_i e^{-\|x_i - c_k\|^2/h^2} \left(\frac{x_i - c_k}{h}\right)^\alpha \mathbf{1}_{|\alpha| \leq p_i - 1}$ $\forall |\alpha| \leq p_{max} - 1$
         The truncation number $p_i$ for each source is selected such that
         $\delta(p = p_i, a = \|x_i - c_k\|, \min\left[b_*(\|x_i - c_k\|, p_i), r + r_x^k\right]) \leq \epsilon$;
  Step 5 For each target $y_j$ the discrete Gauss transform is evaluated as
         $\hat{G}(y_j) = \sum_{\|y_j - c_k\| \leq r + r_x^k} \sum_{|\alpha| \leq p_{max} - 1} C_\alpha^k e^{-\|y_j - c_k\|^2/h^2} \left(\frac{y_j - c_k}{h}\right)^\alpha$;
```

5.1 Methods based on sparse data-set representation

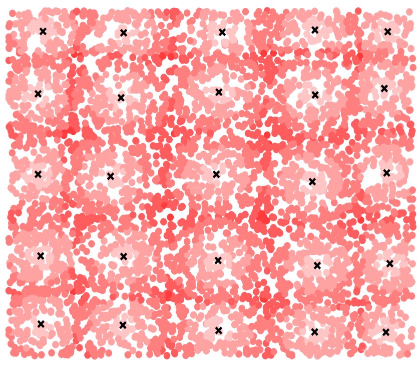
There are many strategies for specific problems which try to reduce this computational complexity by searching for a sparse representation of the data [Williams and Seeger 2001; Smola and Bartlett 2001; Fine and Scheinberg 2001; Lee and Mangasarian 2001]. All these try to find a reduced subset of the original data-set using either random selection or greedy approximation. In these methods there is no guarantee on the approximation of the kernel matrix in a deterministic sense.

5.2 Binned Approximation based on FFT

If the source points are on a evenly spaced grid then we can compute the Gauss transform at an even spaced grid exactly in $\mathcal{O}(N \log N)$ using the fast Fourier transform (FFT). One of the earliest methods, especially proposed for univariate fast



(a)



(b)

Fig. 8. (a) Schematic of the data adaptive improved fast Gauss transform. (b) Illustration of the truncation number p_i required for each data point. Dark red color indicates a higher truncation number. Note that for points close to the cluster center the truncation number required is less than that at the boundary of the cluster.

kernel density estimation was based on this idea [Silverman 1982]. For irregularly spaced data, the space is divided into boxes, and the data is assigned to the closest neighboring grid points to obtain grid counts. The Gauss transform is also evaluated at regular grid points. For target points not lying on the the grid the value is obtained by doing some sort of interpolation based on the values at the neighboring grid points. As a result there is no guaranteed error bound for such kind of methods. Also another problem is that the number of grid points grows exponentially with dimension.

5.3 Dual-tree methods

The dual-tree methods [Gray and Moore 2001; Gray and Moore 2003] are based on space partitioning trees for both the source and target points. This method first builds a spatial tree like *kd*-trees on both the source and target points. Using the tree data structure distance bounds between nodes can be computed.The bounds can be tightened by recursing on both trees. An advantage of the dual-tree methods is that they work for all common kernel choices, not necessarily Gaussian. The series based methods require a different expansion and error bounds for each kernel. Another interesting point is that the dual-tree methods give good speedup for small bandwidths while the series based methods such as IGT and FGT give good speedup for large bandwidths.

5.4 Fast Gauss Transform

The fast Gauss transform (FGT) [Greengard and Strain 1991] is a special case of the more general single level fast multipole method [Greengard and Rokhlin 1987], adapted to the Gaussian potential. The first step of the FGT is the spatial subdivision of the unit hypercube into N_{side}^d boxes of side $\sqrt{2}rh$ were $r < 1/2$. The paper suggests to choose the largest $r \leq 1/2$ such that $N_{side} = 1/\sqrt{2}rh$ is an integer. The sources and targets are assigned to different boxes. Given the sources in one box and the targets in a neighboring box, the computation is performed using one of the following four methods depending on the number of sources and targets in these boxes: Direct evaluation is used if the number of sources and targets are small (In practice a cutoff of the order $\mathcal{O}(p^{d-1})$ is introduced.). If the sources are clustered in a box then they can be transformed into Hermite expansion about the center of the box. This expansion is directly evaluated at each target in the target box if the number of the targets is small. If the targets are clustered then the sources or their expansion are converted to a local Taylor series which is then evaluated at each target in the box. Since the Gaussian decays very rapidly only a few neighboring source boxes will have influence on the target box.

5.4.1 *Series expansions and Translation.* As discussed in Section 3.5 the IFGT algorithm uses the same expansion both for local as well as far-field. The fast multipole methods use two kind of expansions of the potential function. The far-field expansion and the local expansion. For any $x_* \in \mathbf{R}^d$ we call the expansion $\Phi(y, x_i) = \sum_{m=0}^{\infty} b_m(x_i, x_*) S_m(y - x_*)$ *far field* expansion outside a sphere $B^{>}_{R_*}(x_*) = \{y \in \mathbf{R}^d : \|y - x_*\| > R_*\}$, if the series converges for all $y \in B^{>}_{R_*}(x_*)$. For any $x_* \in \mathbf{R}^d$ we call the expansion $\Phi(y, x_i) = \sum_{m=0}^{\infty} a_m(x_i, x_*) R_m(y - x_*)$ *regular (local)* inside a sphere $B^{<}_{r_*}(x_*) = \{y \in \mathbf{R}^d : \|y - x_*\| < r_*\}$, if the series converges for all $y \in B^{<}_{r_*}(x_*)$. If the potential has a singular point x_i, then we use the local expansion for all $\|y - x_*\| < \|x_i - x_*\|$, and far-field expansion for all $\|y - x_*\| > \|x_i - x_*\|$.

The FGT uses the Hermite expansion of the Gaussian for the far-field and the Taylor expansion of the Gaussian (which is obtained by interchanging y and x_i in the Hermite expansion) as the local expansion. The following are the two expansions used by the FGT.

$$e^{-\|y-x_i\|^2/h^2} = \sum_{\alpha \geq 0} \left[\frac{1}{\alpha!} \left(\frac{x_i - x_*}{h} \right)^{\alpha} \right] h_\alpha \left(\frac{y - x_*}{h} \right) \quad \text{[far-field Hermite expansion]},$$

$$e^{-\|y-x_i\|^2/h^2} = \sum_{\beta \geq 0} \left[\frac{1}{\beta!} h_\beta \left(\frac{x_i - x_*}{h} \right) \right] \left(\frac{y - x_*}{h} \right)^{\beta} \quad \text{[local Taylor expansion]},$$

where $h_\alpha(y)$ are the multivariate Hermite functions [Greengard and Strain 1991]. Since the original FGT uses two representations it must convert between them using a process called *translation*. The original FGT in d dimensions represents the solution using p^d coefficients. The cost of translation is

$$\mathcal{O}(dp^{d+1}(2n+1)^d min((\sqrt{2}rh)^{-d/2}, M)).$$

The new version of the FGT proposed in [Greengard and Sun 1998] reduces the cost of translating the Hermite series. The new version is based on replacing the Hermite and Taylor expansions with an expansion in terms of exponentials (plane waves). Because of this the translation operator becomes diagonal. This reduces the cost of translation from $\mathcal{O}(d(2n+1)^d p^{d+1})$ to $\mathcal{O}(3dp^d)$. In any case the cost of translation grows exponentially with dimension [3].

[3]Also the details of the scheme are presented only for $d \leq 3$.

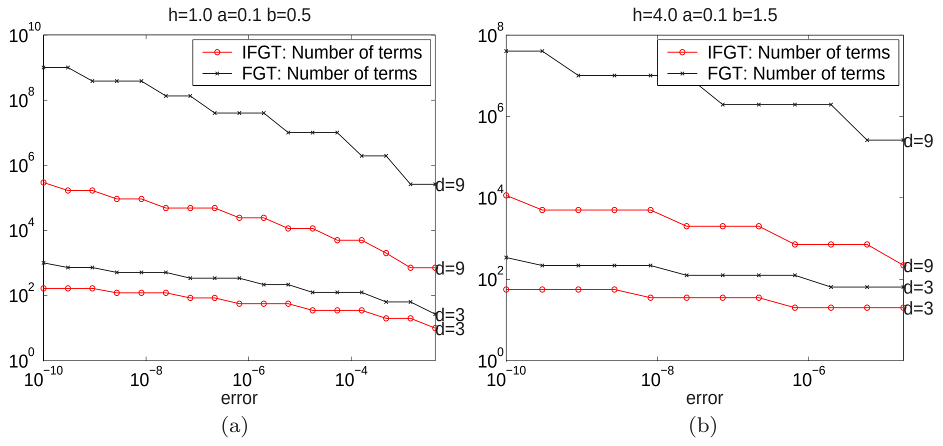


Fig. 9. The total number of terms required by the IFGT and the FGT series expansions to achieve a desired error bound.

In contrast our method uses just one representation with

$$e^{-\|y-x_i\|^2/h^2} = \sum_{|\alpha|\geq 0} \left[\frac{2^\alpha}{\alpha!} e^{-\|x_i-x_*\|^2/h^2} \left(\frac{x_i - x_*}{h} \right)^\alpha \right] e^{-\|y_j-x_*\|^2/h^2} \left(\frac{y_j - x_*}{h} \right)^\alpha .$$

This factorization has the property that it is both a good far-field and and local expansion (See Figure 3). Thereby it avoids the need for two different representations and the expensive translation operation.

5.4.2 *Error bounds.* In this section we compare the number of terms needed to achieve a desired error for the truncated expansions used by the FGT and the IFGT algorithm. We cannot compare both the expressions in terms of p since the truncation method is different for the FGT and the IFGT. We need to see the *total number of terms* that need to be retained to achieve a given target error. For the Hermite expansion all terms with multi indices $\alpha > p$ are ignored (as a result we retain p^d terms) while in the case of IFGT all terms with multi indices of degree $|\alpha| > p$ are ignored (as a result we retain all monomials who's degree is $\leq p-1$ (i.e. a total of $r_{(p-1)d}$ terms)).

Let $|x_i(j) - x_*(j)| = a$ and $|y(j) - x_*(j)| = b$. The error due to truncation in IFGT after ignoring all terms with multi indices of degree $|\alpha| > p$ can be bounded as follows.

$$\begin{aligned} |error_{IFGT}| &< \frac{2^p}{p!} \left(\frac{\|x_i - c_k\|}{h} \right)^p \left(\frac{\|y_j - c_k\|}{h} \right)^p e^{-(\|x_i-c_k\|-\|y_j-c_k\|)^2/h^2}, \\ &= \frac{2^p}{p!} \left(\frac{dab}{h^2} \right)^p e^{-d(a-b)^2/h^2}. \end{aligned}$$

The error due to truncation of either the Hermite series or the Taylor series in FGT after ignoring all terms with multi indices $\alpha > p$ can be bounded as follows (see

Appendix 10 for a detailed derivation).

$$|error_{FGT}| < \frac{e^{-db^2/2h^2}}{(1-r)^d} \sum_{k=0}^{d-1} \binom{d}{k} (1-r^p)^k \left(\frac{r^p}{\sqrt{p!}} \right)^{d-k}.$$

where $r = \sqrt{2}a/h$. Figure 9 compares the total number of terms required by the IFGT and the FGT series expansions to achieve a desired error bound. Our expansion and truncation scheme results in a substantial reduction in the number of terms.

5.4.3 *Spatial data structures.* The original FGT uses boxes to subdivide the space. However such a simple space subdivision scheme is not suitable for high dimensions. If each dimension of a unit hyper cube is divided into N_{side} parts, then the number of boxes grows exponentially with dimension as N_{side}^d, resulting in prohibitive memory requirements. In most statistical and machine learning applications we do not have truly high dimensional data. The data will typically lie on low dimensional manifolds. The consequence of this is that most of the boxes will be empty and we will be spending resources in searching nonempty neighboring boxes. To adaptively fit the density of points, the IFGT uses the farthest-point algorithm to subdivide the space. Table II compares the number of boxes required by the FGT and the number of clusters required by the IFGT as a function of the data dimensionality d. For example in a seven dimensional space while the FGT subdivides the space into 2187 boxes the IFGT just needs 67 clusters.

5.4.4 *Exponential growth of complexity with dimension.* The total computational complexity of the FGT is of the form [Greengard and Strain 1991]

$$O(p^dN) + O(p^dM) + O(dp^{d+1}(2n+1)^d min((\sqrt{2}rh)^{-d/2}, M)).$$

The third term $dp^{d+1}(2n+1)^d min((\sqrt{2}rh)^{-d/2}, M)$ is essentially a constant depending on the number of box-box interactions and the cost of translating a Hermite expansion into a Taylor series. The translation is one of the most expensive step in any FMM algorithm. Even though it does not depend on N the constant term grows exponentially with increasing dimensionality. Table. II shows the constant term as a function of d for $h = 0.5$ and $\epsilon = 10^{-6}$. This suggests that the FGT may not be practical for dimensions > 3. Also the constant term p^d grows exponentially with dimension. Compare this with the computational complexity of the IFGT.

$$O(\log KN) + O(r_{(p-1)d}N) + O(nr_{(p-1)d}M).$$

Assuming $M = N$, the complexity is $\mathcal{O}(\left[\log K + (1+n)r_{(p-1)d}\right] N)$. The constant $r_{(p-1)d}$ is asymptotically polynomial in d. For $d \to \infty$ and moderate p, the number of terms is $\mathcal{O}(d^p)$. Since we cluster only the source points we do not use any expensive translation operation. Table II compares the number of terms required for FGT (p^d) with the number of terms required by IFGT ($r_{(p-1)d}$).

If the bandwidth is large then the FGT will not subdivide the space and work with only one box. Table III shows the same results for $h = 2.0$. Even in this case the number of terms p^d grows much rapidly than the number of terms in the IFGT. Also in order for the FGT to reach the asymptotic performance of $\mathcal{O}(p^dN)$ a large N will be required because of the constant term due to translation.

d	**FGT** # of boxes (N^d_{side})	p	# of terms (p^d)	n	Constant term	**IFGT** # of clusters (K)	p	# of terms ($r_{(p-1)d}$)
1	3	9	9	2	7.014806e+002	5	9	9
2	9	10	100	2	1.500000e+005	7	15	120
3	27	10	1000	2	1.948557e+007	15	16	816
4	81	11	14641	2	3.623648e+009	29	17	4845
5	243	11	161051	2	4.314985e+011	31	20	42504
6	729	12	2985984	2	9.069926e+013	62	20	177100
7	2187	14	105413504	2	3.774303e+016	67	22	1184040

Table II. Comparison of the different parameters chosen by the FGT and the IFGT as a function of the data dimensionality d. $N = 100,000$ points were uniformly distributed in a unit hyper cube. The bandwidth was $h = 0.5$ and the target error was $\epsilon = 10^{-6}$.

d	**FGT** # of boxes (N^d_{side})	p	# of terms (p^d)	n	Constant term	**IFGT** # of clusters (K)	p	# of terms ($r_{(p-1)d}$)
1	1	8	8	0	6.400000e+001	1	10	10
2	1	8	64	0	1.024000e+003	2	8	36
3	1	9	729	0	1.968300e+004	1	10	220
4	1	9	6561	0	2.361960e+005	3	8	330
5	1	9	59049	0	2.657205e+006	2	9	1287
6	1	10	1000000	0	6.000000e+007	2	9	3003
7	1	10	10000000	0	7.000000e+008	2	9	6435

Table III. Comparison of the different parameters chosen by the FGT and the IFGT as a function of the data dimensionality d. $N = 100,000$ points were uniformly distributed in a unit hyper cube.The bandwidth was $h = 2.0$ and the target error was $\epsilon = 10^{-6}$.

6. NUMERICAL EXPERIMENTS

In this section we present some numerical studies of the speedup and error as a function of the number of data points for different dimensions, bandwidths, and data distribution. The algorithms were programmed in C++ and was run on a 1.6 GHz Pentium M processor with 512Mb of RAM. The code for both the FGT and the IFGT implementation are available by contacting the first author for academic use.

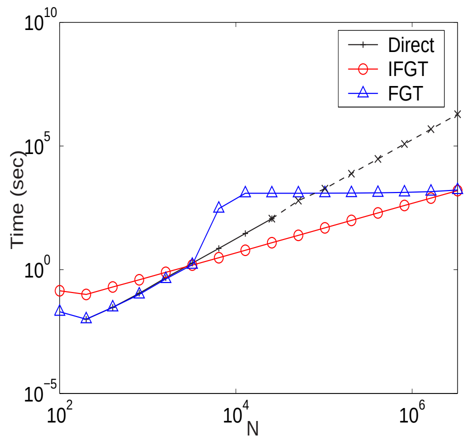
6.1 Speedup as a function of N

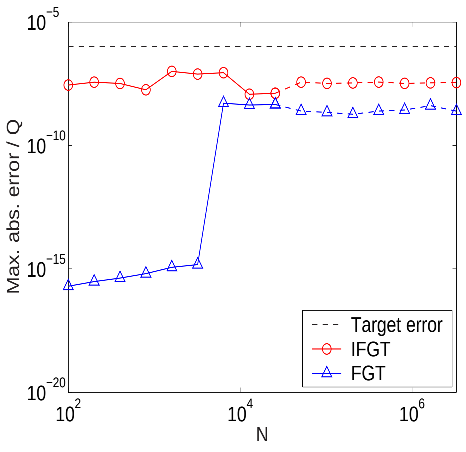
We first study the performance as the function of N for $d = 3$ (where the asymptotic performance of the FGT can be compared with the IFGT for reasonable N). N points were uniformly distributed in a unit cube. The Gauss transform was evaluated at $M = N$ points uniformly distributed in the unit cube. The weights q_i were uniformly distributed between 0 and 1. The parameters were automatically chosen without any user intervention. The target error was set to 10^{-6}.

While the computational complexity of the IFGT grows linearly with N, the linear growth of FGT is a bit intricate because of the various cutoff s involved and the cost of the translation. In order to understand the complexity of the FGT, we refer to Figure 10 where we plot the theoretical complexity($\mathcal{O}(2p^dN + dp^{d+1}(2n+1)^d min((\sqrt{2}rh)^{-d/2}, N)))$ as a function of N for $d = 3$. Initially before the translation has kicked in (i.e. before point A in Figure 10) the growth is linear

Table IV. **[d=3 h=0.4]**The running times in seconds for direct evaluation, FGT, and IFGT. The speedup achieved and the maximum absolute error relative to the total weight Q are also shown. The target error was set to 10^{-6}. The bandwidth was $h = 0.4$. The source and target points were uniformly distributed in a unit cube. The weights q_i were uniformly distributed between 0 and 1. For $N > 25600$ the timing results for the direct evaluation were obtained by evaluating the Gauss transform at $M = 100$ points and then extrapolating the results. The parameters chosen by the FGT were $p = 10$, $n = 3$, and number of boxes $B = 64$. For the IFGT the parameters chosen were $K = 21$ and $p_{max} = 15$.

	Direct	FGT			IFGT		
$N = M$	Time	Time	Speedup	Error	Time	Speedup	Error
100	0.00	0.02	0.00	1.97e-016	0.14	0.00	2.78e-008
200	0.01	0.01	1.00	3.03e-016	0.10	0.10	3.66e-008
400	0.03	0.03	1.00	4.21e-016	0.20	0.15	3.23e-008
800	0.11	0.10	1.11	6.38e-016	0.39	0.28	1.79e-008
1600	0.47	0.42	1.12	1.15e-015	0.78	0.60	9.99e-008
3200	1.83	1.58	1.16	1.48e-015	1.53	1.20	7.72e-008
6400	7.24	292.72	0.02	5.14e-009	3.05	2.37	8.84e-008
12800	29.08	1223.93	0.02	4.33e-009	6.10	4.77	1.18e-008
25600	116.04	1226.07	0.09	4.53e-009	12.33	9.41	1.29e-008
51200	620.54	1229.29	0.50	2.46e-009	24.45	25.37	3.65e-008
102400	1875.97	1235.98	1.52	2.17e-009	49.06	38.24	3.25e-008
204800	7587.84	1250.00	6.07	1.84e-009	97.97	77.45	3.39e-008
409600	29982.72	1277.02	23.48	2.46e-009	196.16	152.85	3.67e-008
819200	120266.75	1330.57	90.39	2.72e-009	392.96	306.05	3.25e-008
1638400	480247.81	1438.42	333.87	4.09e-009	785.11	611.70	3.41e-008
3276800	1930526.72	1661.08	1162.21	2.45e-009	1574.56	1226.07	3.51e-008

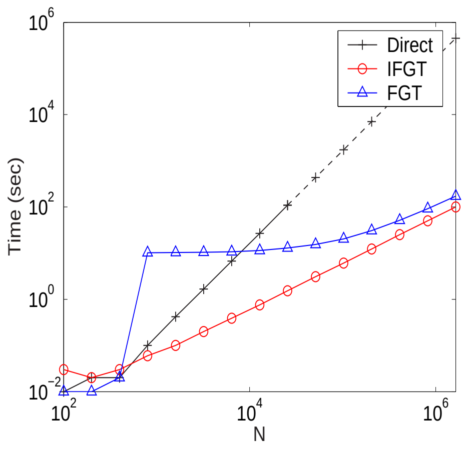


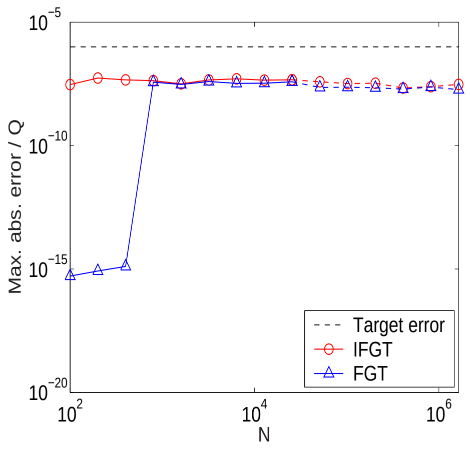


in N. The sudden jump observed is due to the constant associated with the high cost of translation after which the growth is dominated by the constant term. From this point all all box-box interactions are performed only by the translations. However after a large N the asymptotics dominate because the growth in N has dominated the cost of translation (i.e. after point B in Figure 10). In practice especially for high dimensions the constant term due to translation is so large that the N has to be typically very large (e.g. around $N = 10^8$ for $d = 3$) for the asymptotic

Table V. **[d=3 h=1.0]**The running times in seconds for direct evaluation, FGT, and IFGT. The speedup achieved and the maximum absolute error relative to the total weight Q are also shown. The target error was set to 10^{-6}. The bandwidth was $h = 1.0$. The source and target points were uniformly distributed in a unit cube. The weights q_i were uniformly distributed between 0 and 1. For $N > 25600$ the timing results for the direct evaluation were obtained by evaluating the Gauss transform at $M = 100$ points and then extrapolating the results. The parameters chosen by the FGT were $p = 9$, $n = 1$, and number of boxes $B = 8$. For the IFGT the parameters chosen were $K = 2$ and $p_{max} = 14$.

	Direct	FGT			IFGT		
$N = M$	Time	Time	Speedup	Error	Time	Speedup	Error
100	0.01	0.01	1.00	5.18e-016	0.03	0.33	2.94e-008
200	0.02	0.01	2.00	8.31e-016	0.02	1.00	5.46e-008
400	0.02	0.02	1.00	1.27e-015	0.03	0.67	4.59e-008
800	0.10	10.21	0.01	3.73e-008	0.06	1.67	4.25e-008
1600	0.42	10.25	0.04	3.00e-008	0.10	4.21	3.19e-008
3200	1.67	10.43	0.16	3.92e-008	0.20	8.36	4.55e-008
6400	6.74	10.70	0.63	3.31e-008	0.39	17.24	5.05e-008
12800	26.77	11.37	2.36	3.37e-008	0.76	35.18	4.48e-008
25600	109.37	12.85	8.51	3.84e-008	1.53	71.34	4.63e-008
51200	435.71	15.47	28.16	2.27e-008	3.07	141.69	3.83e-008
102400	1733.63	20.26	85.57	2.30e-008	6.07	285.65	3.25e-008
204800	7014.40	30.67	228.68	2.20e-008	12.29	570.83	3.38e-008
409600	28467.20	51.23	555.63	1.95e-008	25.28	1126.25	2.15e-008
819200	114933.76	91.05	1262.30	2.34e-008	49.83	2306.42	2.46e-008
1638400	448593.92	172.31	2603.44	1.85e-008	99.76	4496.55	3.00e-008





performance to kick in.

Table IV and V show the results for $h = 0.40$ and $h = 1.00$. For the IFGT the computational cost grows linearly with N. For the FGT the cost grows linearly only after a large N when the linear term $\mathcal{O}(p^d N)$ dominates the initial cost of Hermite-Taylor translation. The IFGT shows a better speedup than the FGT. However for the case when $h = 0.40$ the FGT finally catches up with IFGT (i.e. the asymptotic performance starts dominating) and shows a better speedup than the IFGT. However this happens typically after a very large N which increases

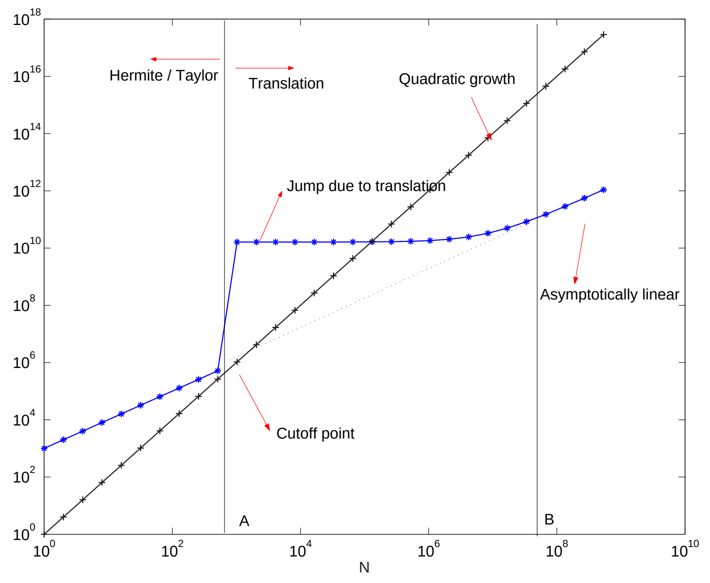


Fig. 10. The theoretical complexity of the FGT showing different regions as a function of $N = M$ for $d = 3$.

with the dimensionality of the problem. With regard to the error the IFGT error is closer to the target than is the FGT. The initial error for the FGT is almost zero due to direct evaluation.

6.2 Speedup as a function of d

The main advantage of the IFGT is in higher dimensions where we can no longer run the FGT algorithm due to its enormous computation and space requirements. Figure 11 shows the performance as a function of N for $d = 3$ and $d = 4$. The bandwidth was set to $h = 1.0$. The FGT becomes impractical after three dimensions. For the IFGT as d increases the crossover point increases. Figure 12 shows the performance for a fixed $N = M = 50,000$ as a function of d for a fixed bandwidth of $h = 2.0$. Since the crossover point increases with d for $N = M = 50,000$ we were able to achieve good speedups unto $d = 10$. Note that after $d = 5$ we could no longer run the FGT. The FGT gave good speedup only for $d \leq 4$.

It should be noted that IFGT and FGT show good speedups especially for large bandwidths. Figure 13 shows the performance for a fixed $N = M = 10,000$ as a function of d. In this case for each dimension we set the bandwidth $h = 0.5d$. With h varying with dimension we were able to run the algorithm for arbitrary high dimensions.

6.3 Speedup as a function of the desired error ϵ

Figure 14 shows the tradeoff between the computational complexity and the desired error ϵ. A decrease in running time is obtained at the expense of reduced precision.

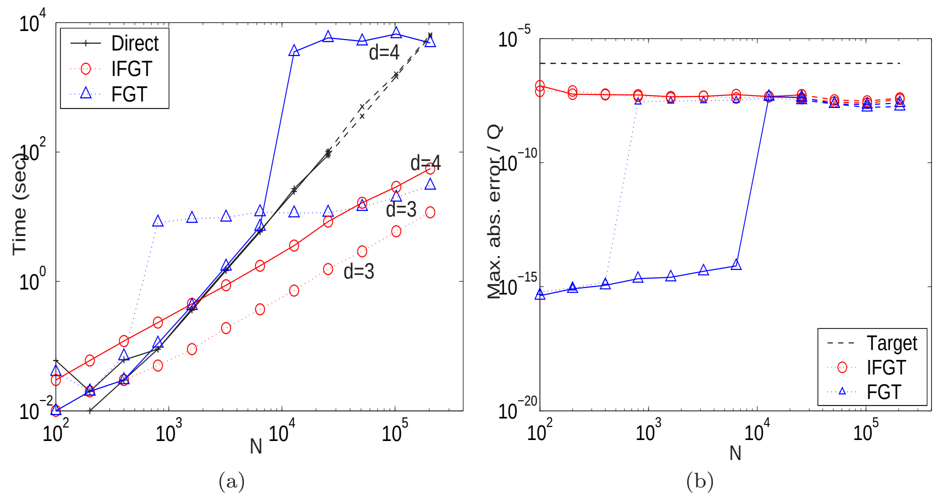


Fig. 11. (a) Comparison of the time taken by the different methods as a function of the number of points N for $d = 3$ and $d = 4$. (b) The corresponding maximum absolute error relative to the total weight Q. The bandwidth was set to $h = 1.0$. The target error was set to 10^{-6}. The source and target points were uniformly distributed in a unit square. The weights q_i were uniformly distributed between 0 and 1. For $N > 256,000$ the timing results for the direct evaluation were obtained by evaluating the Gauss transform at $M = 100$ points and then extrapolating the results.

6.4 Speedup as a function of the bandwidth h

Figure 15 shows the performance as a function of the bandwidth h. Both the FGT and IFGT generally show very good performance when the bandwidth h is large [4]. For smaller bandwidth the performance is often poor. For small bandwidths it may be more efficient to directly evaluate the contribution from its neighbors within a certain radius. Techniques for efficient computation of nearest neighbors [Arya and Mount 1993; Friedman et al. 1977] can be incorporated in the algorithm to automatically do this when the bandwidths are small.

7. APPROXIMATE FAST MULTIVARIATE KERNEL DENSITY ESTIMATION

As a application we show how the IFGT can be used to accelerate multivariate kernel density estimation. We also show the effect of the approximation on the performance of the estimator.

A random variable X on $\mathbf{R}^d$ has a density p if, for all Borel sets A of $\mathbf{R}^d$, $\int_A p(x)dx = \Pr[x \in A]$. The task of density estimation is to estimate p from an

[4]As the dimensionality d increases the volume enclosed by a hypercube increases as R^d. As a result relative to the volume, for a fixed h the Gaussian appears to be at a smaller scale as the dimensionality of the space increases. We need an exponentially large number of samples to cover the increasing volume. So unless the number of samples is very very large using a small h does not make any sense. Especially in tasks like kernel density estimation this leads to a estimate with very high variance. Also in high dimensions the tails of the density contribute significantly to the total probability mass. It is unlikely that we will have a very dense sampling in the tails. Hence we are better off using a large h.

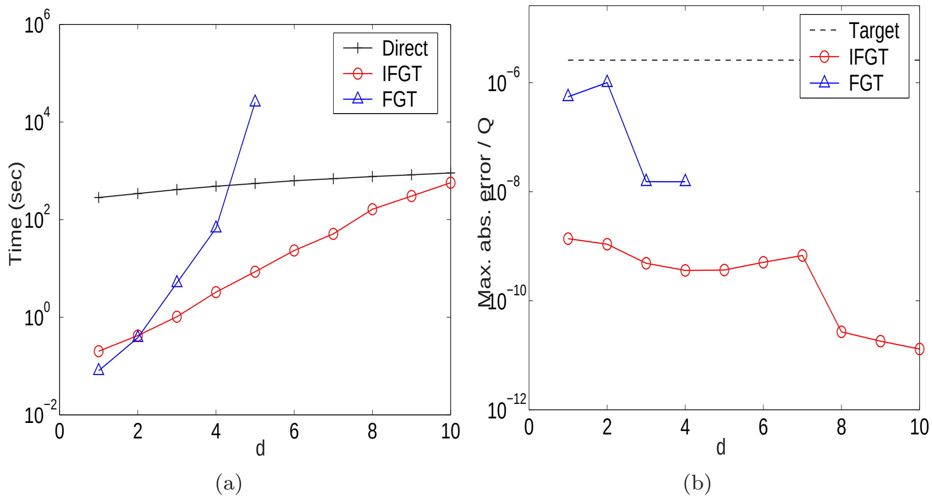


Fig. 12. (a) The running times in seconds and (b) the maximum absolute error relative to the total weight Q for direct evaluation, FGT, and IFGT as a function of the dimension d. The target error was set at $\epsilon = 10^{-6}$. The bandwidth was $h = 2.0$. $N = 50,000$ source and target points were uniformly distributed in a unit hyper cube. The weights q_i were uniformly distributed between 0 and 1.

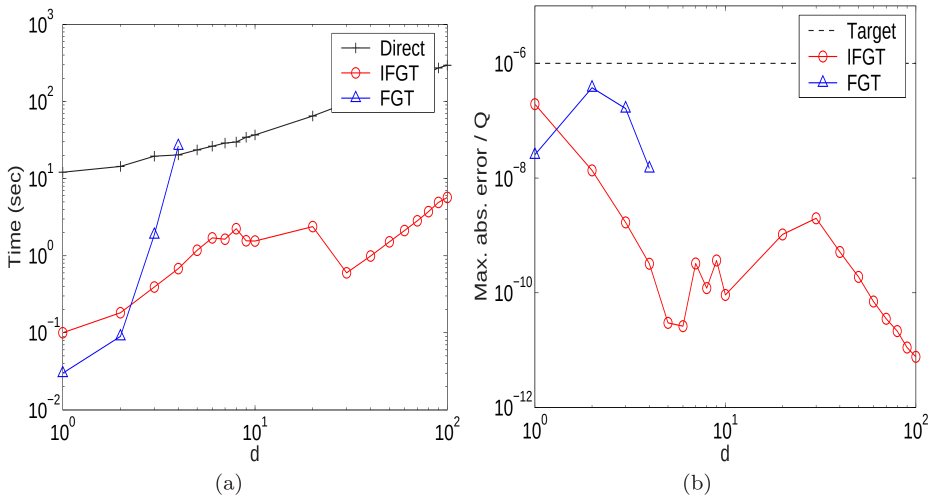


Fig. 13. (a) The running times in seconds and (b) the maximum absolute error relative to the total weight Q for direct evaluation, FGT, and IFGT as a function of the dimension d. The target error was set at $\epsilon = 10^{-6}$. The bandwidth was $h = 0.5d$. $N = 10,000$ source and target points were uniformly distributed in a unit hyper cube. The weights q_i were uniformly distributed between 0 and 1.

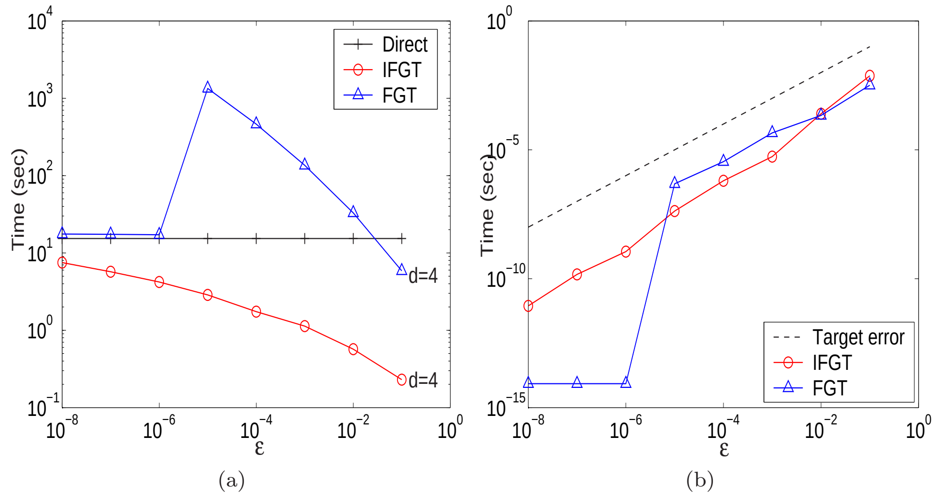


Fig. 14. (a) The running times in seconds and (b) the maximum absolute error relative to the total weight Q for direct evaluation, FGT, and IFGT as a function of the desired error ϵ. The bandwidth was $h = 1.0$. $N = 10,000$ source and target points were uniformly distributed in a unit hyper cube of dimension $d = 4$. The weights q_i were uniformly distributed between 0 and 1.

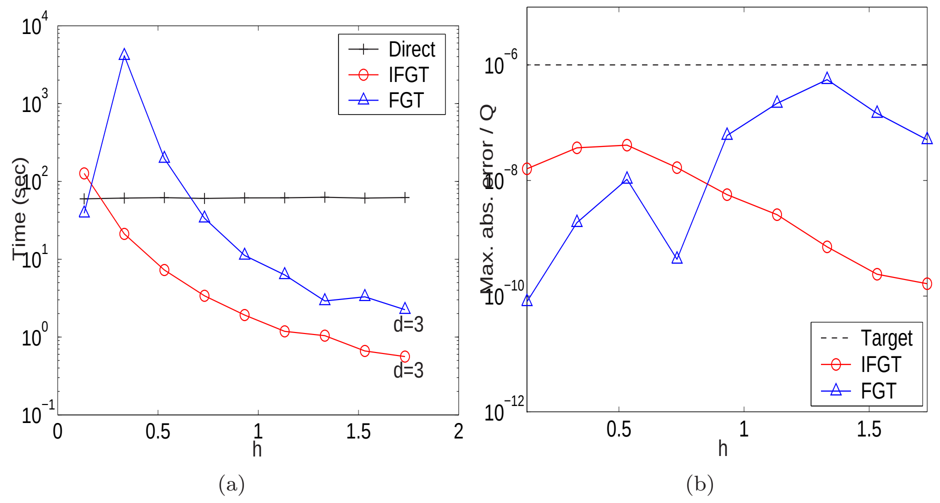


Fig. 15. (a) The running times in seconds and (b) the maximum absolute error relative to the total weight Q for direct evaluation, FGT, and IFGT as a function of the bandwidth h. The target error was set at $\epsilon = 10^{-6}$. $N = 20,000$ source and target points were uniformly distributed in a unit hyper cube of dimension $d = 3$. The weights q_i were uniformly distributed between 0 and 1.

i.i.d. sample $x_1, \ldots, x_N$ drawn from p. The estimate $\widehat{p}_N : \mathbf{R}^d \times \left(\mathbf{R}^d\right)^N \rightarrow \mathbf{R}$ is called the density estimate.

The parametric approach to density estimation assumes a functional form for the density, and then estimates the unknown parameters using techniques like the maximum likelihood estimation. However unless the form of the density is known a priori, assuming a functional form for a density very often leads to erroneous inference. On the other hand nonparametric methods do not make any assumption on the form of the underlying density. This is sometimes referred to as 'letting the data speak for themselves' [Wand and Jones 1995]. The price to be paid is a rate of convergence slower than $1/N$, which is typical of parametric methods. Some of the commonly used non-parametric estimators include histograms, kernel density estimators, and orthogonal series estimators [Izenman 1991]. The histogram is very sensitive to the placement of the bin edges and the asymptotic convergence is much slower than kernel density estimators [5]. The most popular non-parametric method for density estimation is the kernel density estimator (KDE) (also known as the Parzen window estimator [Parzen 1962]) given by

$$\widehat{p}_N(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{V_N} k\left(\frac{x - x_i}{h_N}\right),$$

where $K(u)$ is called *kernel function* and $h = h(N)$ is called the *bandwidth*. The bandwidth h is a scaling factor which goes to zero as $N \rightarrow 0$. In order that $\widehat{p}_N(x)$ is a bona fide density, $k(u)$ is required to satisfy the following two conditions:

$$k(u) \geq 0, \quad \int_{\mathbf{R^d}} k(u) du = 1$$

V_N is the volume occupied by the window $k(u/h_N)$. Hence $V_N = h_N^d$. The most commonly used kernel is the Gaussian of zero mean and unit variance,

$$k(u) = \frac{1}{(2\pi)^{d/2}} e^{-\|u\|^2/2}.$$

In this case the density estimate can be written as,

$$\widehat{p}_N(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{(2\pi h^2)^{d/2}} e^{-\|x - x_i\|^2/2h^2}. \tag{19}$$

The computational cost of evaluating Equation 19 at M points due to N source points is $\mathcal{O}(NM)$, making it prohibitively expensive. The proposed IFGT algorithm can be used to reduce the computational cost to $\mathcal{O}(N + M)$.

In order the study the effect of the $\epsilon - exact$ approximation on the performance of the kernel density estimator we use the notion of L_1 distance. The L_1 distance also known as the integrated absolute error (IAE) between the estimate $\widehat{p}(x)$ and the actual density $p(x)$ is given by,

$$\text{IAE}(\widehat{p}, p) = L_1(\widehat{p}, p) = \int_{\mathbf{R}^d} |\widehat{p}(x) - p(x)| \, dx.$$

[5]The best rate of convergence of the MISE of kernel density estimate is of order $N^{-4/5}$ while that of the histogram is of the order $N^{-2/3}$.

The mean integrated absolute error (MIAE) is given by,

$$\text{MIAE}(\widehat{p}, p) = E[\text{IAE}(\widehat{p}, p)] = E\left[\int_{\mathbf{R}^d} |\widehat{p}(x) - p(x)|dx\right].$$

[Devroye and Lugosi 2000] opine that the total variation criterion is a natural distance measure between two densities. If $\mathbf{B}$ is the class of all Borel sets of $\mathbf{R}^d$, then the total variation (TV) is defined as,

$$\text{TV}(\widehat{p}, p) = \sup_{B \in \mathbf{B}} \left| \int_B \widehat{p}(x)dx - \int_B p(x)dx \right|.$$

By the Scheffe's identity [Theorem 7.1] L_1 distance is twice the total variation criterion.

THEOREM 7.1. *[Scheffe's identity [Devroye and Lugosi 2000]]Let f and g be two functions defined on $\mathbf{R}^d$ satisfying $\int f = \int g = 1$. Let $\mathbf{B}$ denote the class of all Borel sets of $\mathbf{R}^d$. Then*

$$\sup_{B \in \mathbf{B}} \left| \int_B f - \int_B g \right| = \frac{1}{2} \int_{\mathbf{R}^d} |f - g|$$

PROOF. $\sup_{B \in \mathbf{B}} \left|\int_B f - \int_B g\right| = \int_{f>g} (f-g) = \int_{g>f} (g-f) = \frac{1}{2}\int_{\mathbf{R}^d} |f-g|.$ □

Thus if we know that $L_1(f,g) < \epsilon$ then the differences in probabilities are at most $\epsilon/2$. Also the L_1 distance is invariant to monotone continuous change of scale [Devroye and Lugosi 2000]. We have the following theorem which shows the effect of the approximation on the total variation.

THEOREM 7.2. *Let $X \in [0,1]^d \subset \mathbf{R}^d$ (i.e. we assume that the data is scaled to a unit hyper cube) be a random variable with density p, $\widehat{p}$ be the kernel density estimate, and $\widehat{p}_A$ be the approximate kernel density estimate computed using the improved fast Gauss transform such that $|\widehat{p}_A - \widehat{p}| \leq Q\epsilon = (2\pi h^2)^{-d/2}\epsilon$. Then*

$$TV(\widehat{p}_A, p) \leq TV(\widehat{p}, p) + \frac{\epsilon}{2}(2\pi h^2)^{-d/2}. \tag{20}$$

PROOF. By Scheffé's Identity [Theorem 7.1],

$$\begin{aligned} TV(\widehat{p}_A, p) &= \frac{1}{2}\int |\widehat{p}_A - p| \\ &= \frac{1}{2}\int |(\widehat{p} - p) + (\widehat{p}_A - \widehat{p})| \\ &\leq \frac{1}{2}\int |\widehat{p} - p| + |\widehat{p}_A - \widehat{p}| \\ &\leq \frac{1}{2}\int |\widehat{p} - p| + \frac{1}{2}\int (2\pi h^2)^{-d/2}\epsilon \\ &= TV(\widehat{p}, p) + \frac{\epsilon}{2}(2\pi h^2)^{-d/2} \end{aligned}$$

□

8. CONCLUSIONS

We proposed the improved fast Gauss transform which is capable of computing the Gauss transform in $\mathcal{O}(N)$ time in dimensions as high as tens for small bandwidths and as high as hundreds for large bandwidths. The reduction is based on a new multivariate Taylor's series expansion (which can act both as a local as well as a far field expansion) scheme combined with the efficient space subdivision using the k-center algorithm. We derived tight pointwise error bounds and gave a strategy to choose the parameters of the algorithm. Numerical experiments demonstrated the speedup achieved over the original FGT. Finally we showed how the IFGT can be used for fast kernel density estimation.

The following are two simple extensions.

—The bandwidth h is different for each dimension, i.e.,

$$G(y_j) = \sum_{i=1}^{N} q_i e^{-(y_j-x_i)^T \Sigma^{-1} (y_j-x_i)} = \sum_{i=1}^{N} q_i e^{-\sum_{k=1}^{d} (y_j(k)-x_i(k))^2/h_k^2},$$

where Σ is diagonal matrix with the k^{th} element equal to h_k^2. In this case we can divide each co-ordinate of the source and target points with the corresponding bandwidth h_k and then use the IFGT with bandwidth $h = 1$.

—More generally we can have

$$G(y_j) = \sum_{i=1}^{N} q_i e^{-(y_j-x_i)^T H^{-1} (y_j-x_i)},$$

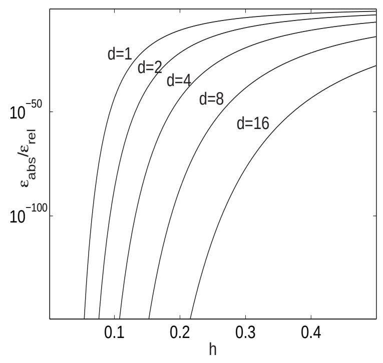
where H is a symmetric positive definite $d \times d$ matrix called the bandwidth matrix. In this case we can factorize the inverse bandwidth matrix as $H^{-1} = U^T \Sigma^{-1} U = (\Sigma^{-1/2} U)^T (\Sigma^{-1/2} U)$. Now we can apply the following linear transformation $x \rightarrow \Sigma^{-1/2} U x$ to each of the source and target points and then use the IFGT with $h = 1$. This is equivalent to rotating and scaling the points before using the IFGT.

We also point out that the algorithm is easily adaptable in an online setting. If a new target point arrives them we just have to sum the contributions from all its influential neighbor clusters. If a new single source point arrives we just add its contribution directly to all the target points. In case a lot of source points arrive in a batch then we update the coefficients of the clusters to which the source points belong to and then reevaluate the contribution at the target points.

9. APPENDIX 1: RELATIVE VS ABSOLUTE ERROR

The astute reader would have observed that we have used the absolute error rather than the relative error. Equation 2 is the error used in the original paper on FGT [Greengard and Strain 1991]. Equation 2 is the error used in the original paper on FGT [Greengard and Strain 1991]. However it would be more reasonable if we had used the maximum absolute relative error,

$$\max_{y_j} \left[\frac{|\hat{G}(y_j) - G(y_j)|}{|G(y_j)|} \right]. \tag{21}$$



(a)

Fig. 16. (a) $\epsilon_{abs}/\epsilon_{rel}$ for different h and d. $R = \sqrt{d}$.

The reason we prefer to use Equation 2 over Equation 21 is that it easy to derive more tighter point-wise error bound than using the relative error. However if the the reader still feels comfortable only with relative bounds, it is possible to upper bound the relative error bound with the absolute error bound as follows. However this is true for only when the weights q_i are positive (for example in kernel density estimation). For any $y_j \in \mathbf{R}^d$

$$\frac{|\hat{G}(y_j) - G(y_j)|}{|G(y_j)|} = \frac{|\hat{G}(y_j) - G(y_j)|}{|\sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2/h^2}|} \leq \frac{|\hat{G}(y_j) - G(y_j)|}{Qe^{-R^2/h^2}},$$

where $R = \max_{i,j} \|y_j - x_i\|$, i.e., the maximum range of the data. For example is all the points are uniformly in a d-dimensional unit hypercube than $R \leq \sqrt{d}$. So if $0 < \epsilon_{rel} < 1$ is the desired relative error then the desired absolute error ϵ_{abs} will be,

$$\epsilon_{abs} \leq e^{-R^2/h^2}\epsilon_{rel}.$$

This will be tight as long as h is large. But for very small e^{-R^2/h^2} will be very small (see Figure 16) and we will be spending more resources than necessary. Our recommendation is to use the absolute error for small h. Also it should the noted that Equation 2 is relative to the the total weight Q, and this in a sense takes care of the relative scale issues.

10. APPENDIX 2: ERROR BOUND FOR HERMITE SERIES TRUNCATION

The error due to truncation of the Hermite series in FGT after ignoring all terms with multi indices $\alpha > p$ can be bounded as follows [Greengard and Strain 1991; Baxter and Roussos 2002].

$$\begin{aligned}
|error_{FGT}| &= \left|\sum_{\alpha\geq p}\left[\frac{1}{\alpha!}\left(\frac{x_i-x_*}{h}\right)^{\alpha}\right]h_\alpha\left(\frac{y-x_*}{h}\right)\right| \\
&\leq \sum_{\alpha\geq p}\frac{1}{\alpha!}\left|\left(\frac{x_i-x_*}{h}\right)^{\alpha}\right|\left|h_\alpha\left(\frac{y-x_*}{h}\right)\right| \\
&\leq \sum_{\alpha\geq p}\prod_{j=1}^{d}\left|\left(\frac{x_i(j)-x_*(j)}{h}\right)^{\alpha_j}\right|\frac{1}{\alpha_j!}\left|h_{\alpha_j}\left(\frac{y(j)-x_*(j)}{h}\right)\right|
\end{aligned}$$

where $x_i(j)$ is the j^{th} coordinate of x_i. Based on the Cramer's inequality we have the following useful bound for Hermite functions. For any $t \in \mathbf{R}$

$$\frac{1}{n!}|h_n(t)| \leq 2^{n/2}\frac{1}{\sqrt{n!}}e^{-t^2/2}.$$

Hence we have,

$$\begin{aligned}
|error_{FGT}| &\leq \sum_{\alpha\geq p}\prod_{j=1}^{d}\frac{1}{\sqrt{\alpha_j!}}\left|\left(\frac{x_i(j)-x_*(j)}{h}\right)^{\alpha_j}\right|2^{\alpha_j/2}e^{-(y(j)-x_*(j))^2/2h^2} \\
&\leq e^{-\|y-x_*\|^2/2h^2}\sum_{\alpha\geq p}\prod_{j=1}^{d}\frac{1}{\sqrt{\alpha_j!}}\left|\frac{\sqrt{2}(x_i(j)-x_*(j))}{h}\right|^{\alpha_j}
\end{aligned}$$

Let $|x_i(j)-x_*(j)| = a$ and $|y(j)-x_*(j)| = b$. Then,

$$|error_{FGT}| \leq e^{-\|y-x_*\|^2/2h^2}\sum_{\alpha\geq p}\prod_{j=1}^{d}\frac{1}{\sqrt{\alpha_j!}}r^{\alpha_j}$$

where $r = \sqrt{2}a/h$. This can be simplified as follows.

$$\begin{aligned}
\sum_{\alpha \geq p} \prod_{j=1}^{d} \frac{1}{\sqrt{\alpha_j!}} r^{\alpha_j} &= \left[\sum_{\alpha \geq 0} \prod_{j=1}^{d} \frac{1}{\sqrt{\alpha_j!}} r^{\alpha_j} - \sum_{\alpha < p} \prod_{j=1}^{d} \frac{1}{\sqrt{\alpha_j!}} r^{\alpha_j} \right] \\
&= \left[\prod_{j=1}^{d} \left\{ \sum_{\alpha_j \geq 0} \frac{1}{\sqrt{\alpha_j!}} r^{\alpha_j} \right\} - \prod_{j=1}^{d} \left\{ \sum_{\alpha_j < p} \frac{1}{\sqrt{\alpha_j!}} r^{\alpha_j} \right\} \right] \\
&= \left[\left\{ \sum_{\alpha_j < p} \frac{1}{\sqrt{\alpha_j!}} r^{\alpha_j} + \sum_{\alpha_j \geq p} \frac{1}{\sqrt{\alpha_j!}} r^{\alpha_j} \right\}^d - \left\{ \sum_{\alpha_j < p} \frac{1}{\sqrt{\alpha_j!}} r^{\alpha_j} \right\}^d \right] \\
&= \sum_{k=0}^{d-1} \binom{d}{k} \left(\sum_{\alpha_j < p} \frac{1}{\sqrt{\alpha_j!}} r^{\alpha_j} \right)^k \left(\sum_{\alpha_j \geq p} \frac{1}{\sqrt{\alpha_j!}} r^{\alpha_j} \right)^{d-k} \quad \text{[Binomial theorem]} \\
&\leq \sum_{k=0}^{d-1} \binom{d}{k} \left(\sum_{\alpha_j < p} r^{\alpha_j} \right)^k \left(\frac{r^p}{\sqrt{p!}} \sum_{\alpha_j \geq 0} r^{\alpha_j} \right)^{d-k} \\
&\leq \sum_{k=0}^{d-1} \binom{d}{k} \left(\frac{1-r^p}{1-r} \right)^k \left(\frac{r^p}{\sqrt{p!}} \frac{1}{1-r} \right)^{d-k} \quad \text{[Geometric series } r < 1.\text{]} \\
&= \frac{1}{(1-r)^d} \sum_{k=0}^{d-1} \binom{d}{k} (1-r^p)^k \left(\frac{r^p}{\sqrt{p!}} \right)^{d-k}
\end{aligned}$$

So we have,

$$|error_{FGT}| \; < \; e^{-db^2/2h^2} \frac{1}{(1-r)^d} \sum_{k=0}^{d-1} \binom{d}{k} (1-r^p)^k \left(\frac{r^p}{\sqrt{p!}} \right)^{d-k}$$

where $r = \sqrt{2}a/h$.

REFERENCES

Arya, S. and Mount, D. M. 1993. Approximate nearest neighbor searching. In *Proc. 4th Ann. ACM-SIAM Symposium on Discrete Algorithms*. 271–280. 29

Baxter, B. J. C. and Roussos, G. 2002. A new error estimate of the fast gauss transform. *SIAM Journal of Scientific and Statistical Computing 24,* 1, 257–259. 35

Bern, M. and Eppstein, D. 1997. *Approximation algorithms for NP-hard problems.* PWS Publishing Co., Boston, Chapter Approximation algorithms for geometric problems, 296–345. 8

Chung, F. 1997. *Spectral Graph Theory.* Amer. Math. Society Press. 3

Cristianini, N. and Shawe-Taylor, J. 2000. *An Introduction to Support Vector Machines (and other kernel-based learning methods).* Cambridge University Press. 3

Devroye, L. and Lugosi, G. 2000. *Combinatorial Methods in Density Estimation.* Springer-Verlag. 33

Feder, T. and Greene, D. 1988. Optimal algorithms for approximate clustering. In *Proc. 20th ACM Symp. Theory of Computing*. 434–444. 9, 12

Fine, S. and Scheinberg, K. 2001. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research 2*, 243264. 20

Friedman, J. H., Bentley, J. L., and Finkel, R. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software 3,* 3, 209–226. 29

Girosi, F., Jones, M., and Poggio, T. 1995. Regularization theory and neural networks architectures. *Neural Computation 7,* 2, 219–269. 3

Gonzalez, T. 1985. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science 38*, 293–306. 8

Gray, A. and Moore, A. 2001. N-body problems in statistical learning. In *Advances in Neural Information Processing Systems*. 521–527. 3, 21

Gray, A. G. and Moore, A. W. 2003. Nonparametric density estimation: Toward computational tractability. In *SIAM International conference on Data Mining*. 21

Greengard, L. and Rokhlin, V. 1987. A fast algorithm for particle simulations. *J. of Comp. Physics 73,* 2, 325–348. 21

Greengard, L. and Strain, J. 1991. The fast Gauss transform. *SIAM Journal of Scientific and Statistical Computing 12,* 1, 79–94. 4, 12, 21, 22, 24, 34, 35

Greengard, L. and Sun, X. 1998. A new version of the fast gauss transform. *Documenta Mathematica Extra Volume ICM,* III, 575 – 584. 4, 22

Greengard, L. F. 1988. *The Rapid Evaluation of Potential Fields in Particle Systems*. The MIT Press. 12

Hochbaum, D. S. and Shmoys, D. B. 1985. A best possible heuristic for the k-center problem. *Mathematics of Operations Research 10*, 180–184. 9

Izenman, A. J. 1991. Recent developments in nonparametric density estimation. *Journal of American Staistical Association 86,* 413, 205–224. 3, 32

Lang, D., Klaas, M., and Freitas, N. 2005. Empirical testing of fast kernel density estimation algorithms. Tech. Rep. UBC TR-2005-03, Dept. of Computer Science, University of British Columbia. 5, 15

Lee, Y.-J. and Mangasarian, O. 2001. Rsvm: Reduced support vector machines. In *First SIAM International Conference on Data Mining, Chicago*. 20

Parzen, E. 1962. On estimation of a probability density function and mode. *Annals of Mathematical Statistics 33,* 3, 1065–1076. 32

Poggio, T. and Smale, S. 2003. The mathematics of learning: Dealing with data. *Notices of the American Mathematical Society 50,* 5, 537–544. 3

Seeger, M. 2004. Gaussian processes for machine learning. *International Journal of Neural Systems 14,* 2, 1–38. 3

Shawe-Taylor, J. and Cristianini, N. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press. 3

Silverman, B. W. 1982. Algorithm AS 176: Kernel density estimation using the fast Fourier transform. *Journal of Royal Statistical society Series C: Applied statistics 31,* 1, 93–99. 21

Smola, A. and Bartlett, B. 2001. Sparse greedy gaussian process regression. In *Advances in Neural Information Processing Systems*. MIT Press, 619625. 20

Smola, A., Scholkopf, B., and Muller, K.-R. 1996. Nonlinear component analysis as a kernel eigenvalue problem. Tech. Rep. 44, Max-Planck-Institut fr biologische Kybernetik, Tubingen. 3

Vaidya, P. M. 1986. An optimal algorithm for the all-nearest-neighbors problem. In *Proc. 27th IEEE FOCS*. 117–122. 9

Wand, M. P. and Jones, M. C. 1995. *Kernel Smoothing*. Chapman and Hall, London. 3, 32

Williams, C. K. I. and Rasmussen, C. E. 1996. Gaussian processes for regression. In *Advances in Neural Information Processing Systems*. Vol. 8. 3

Williams, C. K. I. and Seeger, M. 2001. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*. MIT Press, 682688. 20

Yang, C., Duraiswami, R., and Davis, L. 2005. Efficient kernel machines using the improved fast Gauss transform. In *Advances in Neural Information Processing Systems*. 1561–1568. 5, 15

Yang, C., Duraiswami, R., and Gumerov, N. 2003. Improved fast Gauss transform. Tech. Rep. CS-TR-4495, Dept. of Computer Science, University of Maryland, College Park. 5