IMPROVED FAST GAUSS TRANSFORM [*]

CHANGJIANG YANG, RAMANI DURAISWAMI, NAIL A. GUMEROV [†]

Abstract. The fast Gauss transform proposed by Greengard and Strain reduces the computational complexity of the evaluation of the sum of N Gaussians at M points in d dimensions from $O(MN)$ to $O(M+N)$. However, the constant factor in $O(M+N)$ grows exponentially with increasing dimensionality d, which makes the algorithm impractical in higher dimensions. In this paper we present an improved fast Gauss transform where the constant factor is reduced to asymptotically polynomial order. The reduction is based on a new multivariate Taylor expansion scheme combined with the space subdivision using the k-center algorithm. The complexity analysis and error bound are presented which helps to determine parameters automatically given a desired precision to which the sum must be evaluated. We present numerical results on the performance of our algorithm and provide numerical verification of the corresponding error estimate.

Key words. Gauss transform, fast algorithms, fast multipole method.

1. Introduction.

Since the original work of Greengard and Strain [16], the fast Gauss transform has proven to be a very efficient algorithm for solving many problems in applied mathematics and physics, and nonparametric statistics [15, 18, 4, 10]. All these problems require the evaluation of the *discrete Gauss transform*

$$G(y_j) = \sum_{i=1}^{N} q_i e^{-\|y_j - x_i\|^2/h^2}, \tag{1.1}$$

where q_i are the weight coefficients, "source" points $\{x_i\}_{i=1,\dots,N}$ are the centers of the Gaussians, h is the bandwidth parameter of the Gaussians. The sum of the Gaussians is evaluated at each of the "target" points $\{y_j\}_{j=1,\dots,M}$. Direct evaluation of the sum at M target points due to N sources requires $O(MN)$ operations, which makes the computation of large scale problems prohibitively expensive.

To break through this computational barrier, Greengard and Strain [16] developed the fast Gauss transform, which requires $O(M+N)$ operations, with a constant factor depending on the dimensionality d and the desired precision. The fast Gauss transform is an "analysis-based" fast algorithm in the sense that it speeds up the computation by approximation of the Gaussian function to achieve a desired precision. The sources and targets can be placed on general positions. In contrast the most popular fast Fourier transform requires the point to be on a regular mesh which is in general not available in the application of statistics and pattern recognition. An implementation in two dimensions demonstrated the efficiency and effectiveness of the fast Gauss transform [16].

Despite its success in lower dimensional applications in mathematics and physics, the algorithm has not been used much in statistics, pattern recognition and machine learning where higher dimensions occur commonly [9]. An important reason for the lack of use of the algorithm in these areas is that the performance of fast Gauss transform degrades exponentially with increasing dimensionality, which makes it impractical for the statistics and pattern recognition applications.

There are two reasons contributing to the degradation of the fast Gauss transform in higher dimensions: firstly, the number of the terms in the Hermite expansion grows exponentially with dimensionality d, which makes the constant factor associated with the nominal

[*]THIS WORK WAS PARTIALLY SUPPORTED BY NSF AWARDS 9987944, 0086075 AND 0219681.

[†]Perceptual Interfaces and Reality Laboratory, University of Maryland, College Park, MD 20742, ({yangcj,ramani,gumerov}@umiacs.umd.edu).

complexity $O(M + N)$ increases exponentially with dimensionality. So the total computations and the amount of memory required increases dramatically as the dimensionality increases. Secondly, 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.

In this paper we present an improved fast Gauss transform which addresses the above issues. In this scheme, a multivariate Taylor expansion is used to reduce the number of the expansion terms to the polynomial order. The k-center algorithm is applied to subdivide the space which is more efficient in higher dimensions.

The organization of the paper is as follows. In section 2 we briefly describe the fast multipole method and the fast Gauss transform. In section 3 we describe our improved fast Gauss transform and present computational complexity analysis and the error bounds. In section 4 we present numerical results of our algorithm and verification of the corresponding error estimate. We conclude the paper in section 5.

2. FMM and FGT.

The fast Gauss transform (FGT) introduced by Greengard and Strain [16, 24] is an important variant of the more general Fast Multipole Method (FMM) [14]. Originally the FMM was developed for the fast summation of potential fields generated by a large number of sources, such as those arising in gravitational or electrostatic potential problems in two or three dimensions. Thereafter, this method was extended to other potential problems, such as those arising in the solution of the Helmholtz [8, 7] and Maxwell equations [6]. The FMM has also found application in many other problems, e.g. in chemistry [3], interpolation of scattered data [5].

In nonparametric statistics, pattern recognition and computer vision [23, 9, 10], the most widely used function is the Gaussian which is an infinitely differentiable and rapidly decaying function, in contrast to the *singular* functions that arose in the sums of Green's functions in the original applications of the FMM. This property makes the FGT distinct from the general FMM in the sense that the Gaussian function decays rapidly in space and has no *singular/multipole* expansion. In the following sections, we briefly describe the FMM and the FGT.

2.1. Fast Multipole Method.

Consider the sum

$$v(\mathbf{y}_j) = \sum_{i=1}^{N} u_i \phi_i(\mathbf{y}_j), \quad j = 1, \dots, M, \tag{2.1}$$

where $\{\phi_i\}$ are a family of functions corresponding to the source function ϕ at different centers $\mathbf{x}_i$, $\mathbf{y}_j$ is a point in d dimensions, and u_i is the weight. Clearly a direct evaluation requires $O(MN)$ operations.

In the FMM, we assume that the functions ϕ_i can be expanded in multipole (singular) series and local (regular) series that are centered at locations $\mathbf{x}_*$ and $\mathbf{y}_*$ as follows:

$$\phi(\mathbf{y}) = \sum_{n=0}^{p-1} b_n(\mathbf{x}_*) S_n\left(\mathbf{y} - \mathbf{x}_*\right) + \epsilon\left(p\right), \tag{2.2}$$

$$\phi(\mathbf{y}) = \sum_{n=0}^{p-1} a_n(\mathbf{y}_*) R_n\left(\mathbf{y} - \mathbf{y}_*\right) + \epsilon\left(p\right), \tag{2.3}$$

where S_n and R_n respectively are multipole (singular) and local (regular) basis functions, $\mathbf{x}_*$ and $\mathbf{y}_*$ are expansion centers, a_n, b_n are the expansion coefficients, and ϵ is the error introduced by truncating a possibly infinite series after p terms. The operation reduction trick of the FMM relies on expressing the sum (2.1) using the series expansions (2.2) and (2.3). Then the reexpansion for (2.3) is:

$$v(\mathbf{y}_j) = \sum_{i=1}^{N} u_i \phi_i(\mathbf{y}_j) = \sum_{i=1}^{N} u_i \sum_{n=0}^{p-1} c_{ni} R_n \left(\mathbf{y}_j - \mathbf{x}_*\right), \qquad j = 1, \ldots, M, \tag{2.4}$$

A similar expression can be obtained for (2.2). Consolidating the N series into one p term series, by rearranging the order of summations, we get

$$v(\mathbf{y}_j) = \sum_{n=0}^{p-1} \left[\sum_{i=1}^{N} u_i c_{ni}\right] R_n \left(\mathbf{y}_j - \mathbf{x}_*\right) = \sum_{n=0}^{p-1} C_n R_n \left(\mathbf{y}_j - \mathbf{x}_*\right). \tag{2.5}$$

The single consolidated p term series (2.5) can be evaluated at all the M evaluation points. The total number of operations required is then $O(Mp + Np) \simeq O(Np)$ for $N \sim M$. The truncation number p depends on the desired accuracy alone, and is independent of M, N.

For the singular functions ϕ, for which the FMM was originally developed, a single series expansion will not be valid over the whole domain. This leads to the idea of the FMM. In the FMM, singular expansions (2.2) are generated around clusters of sources. In a fine-to-coarse pass, the generated coefficients are translated into coarser level singular expansions through a tree data structure by “translation” operators. These translation operations convert the original S expansion to S expansions centered at other points. The R expansions can similarly also be translated to R expansions centered at other points. In a coarse-to-fine pass, the coefficients of the singular expansions at coarser level are converted via a sequence of translations to coefficients of regular expansions at finer levels, then evaluated at each evaluation point. Based on a divide-and-conquer strategy, the tree data structures are used to collect and distribute the influence of the sources hierarchically. The FMM is thus a method of grouping and translation of functions generated by each source to reduce the asymptotic complexity of approximating the sum (2.1).

2.2. Fast Gauss Transform. The original FGT directly applies the FMM idea by using the following expansions for the Gaussian in one dimension:

$$e^{-\|y-x_i\|^2/h^2} = \sum_{n=0}^{p-1} \frac{1}{n!} \left(\frac{x_i - x_*}{h}\right)^n h_n \left(\frac{y - x_*}{h}\right) + \epsilon(p), \tag{2.6}$$

$$e^{-\|y-x_i\|^2/h^2} = \sum_{n=0}^{p-1} \frac{1}{n!} h_n \left(\frac{x_i - y_*}{h}\right) \left(\frac{y - y_*}{h}\right)^n + \epsilon(p), \tag{2.7}$$

where the Hermite functions $h_n(x)$ are defined by

$$h_n(x) = (-1)^n \frac{d^n}{dx^n} \left(e^{-x^2}\right).$$

The two expansions (2.6) and (2.7) are identical, except that the arguments of the Hermite functions and the monomials (Taylor series) are flipped. The first is used as the counterpart of the multipole S expansion (usually a singular series for an FMM, here a series of Hermite functions, which decay), while the second is used as the local expansion. The FGT then uses these expansions and applies the FMM mechanism to achieve its speedup. Conversion of a Hermite series into a Taylor series is achieved via a translation operation. The error bound estimate given by Greengard and Strain [16] was shown to be incorrect, and a new and more complicated error bound estimate was presented in [1].

The extension to higher dimensions was done by treating the multivariate Gaussian as a product of univariate Gaussians, applying the series factorizations (2.6) and (2.7) to each dimension. For convenience's sake, we adopt, the multi-index notation of the original FGT papers [16, 24]. A multi-index $\alpha = (\alpha_1, \ldots, \alpha_d)$ is a d-tuple of nonnegative integers. For any multi-index $\alpha \in \mathbf{N}^d$ and any $x \in \mathbf{R}^d$, we have the monomial

$$x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}.$$

The length of the multi-index α is defined as

$$|\alpha| = \alpha_1 + \alpha_2 + \ldots + \alpha_d,$$

and the factorial of α

$$\alpha! = \alpha_1! \alpha_2! \cdots \alpha_d!.$$

The multidimensional Hermite functions are defined by

$$h_\alpha(x) = h_{\alpha_1}(x_1) h_{\alpha_2}(x_2) \cdots h_{\alpha_d}(x_d).$$

The sum (1.1) is then equal to the Hermite expansion about center x_*:

$$G(y_j) = \sum_{\alpha \geq 0} C_\alpha h_\alpha \left(\frac{y_j - x_*}{h} \right), \tag{2.8}$$

where the coefficients C_α are given by

$$C_\alpha = \frac{1}{\alpha!} \sum_{i=1}^{N} q_i \left(\frac{x_i - x_*}{h} \right)^\alpha. \tag{2.9}$$

The FGT in higher dimensions is then just an accumulation of the product of the Hermite expansions along each dimension. If we truncate each of the Hermite series after p terms (or equivalently order $p-1$), then each of the coefficients C_α is a d-dimensional matrix with p^d terms. The total computational complexity for a single Hermite expansion is $O((M+N)p^d)$. The factor $O(p^d)$ **grows exponentially** as the dimensionality d increases. Despite this defect in higher dimensions, the FGT is quite effective for two and three-dimensional problems, and has already achieved success in some physics, computer vision and pattern recognition problems [15, 18, 10, 19].

Another serious defect of the original FGT is the use of the box data structure. The original FGT subdivides the space into boxes using a uniform mesh. However, such a simple space subdivision scheme is not suitable in higher dimensions, especially in applications where the data might be clustered on low dimensional manifolds. First of all, it may generate too many boxes (largely empty) in higher dimensions to store and manipulate. Suppose the unit box in 10 dimensional space is divided into tenths along each side, there are 10^{10} boxes which may cause trouble in storage and waste time on processing empty boxes. Secondly, and more importantly, having so many boxes makes it more difficult for searching nonempty neighbor boxes. Finally, and most importantly the worst property of this scheme is that the ratio of volume of the hypercube to that of the inscribed sphere grows exponentially with dimension, and thus most points . Thus In other words, the points have a high probability of falling into the area inside the box and outside the sphere. (see Figure xx) The truncation error of the above Hermite expansions (2.6) and (2.7) are much larger near the boundary than near the expansion center. These two factors bring large truncation errors on most of the points.

In brief, the defects of the original FGT that we seek to address in this paper are:

1. The exponential growth of complexity with dimensionality.
2. The use of the box data structure in the FMM is inefficient in higher dimensions.

3. Improved Fast Gauss Transform.

3.1. A different factorization. The defects listed above can be thought as a result of applying the FMM methodology to the FGT blindly. Here we seek to use the complexity improvement of Equation 2.5, using different expansions and data-structures. As we mentioned earlier, the FMM was developed for singular potential functions whose forces are long-ranged and nonsmooth (at least locally), hence it is necessary to make use of the tree data structures, multipole expansions, local expansions and translation operators. In contrast, the Gaussian is far from singular — it is infinitely differentiable! There is no need to perform the multipole expansions which account for the far-field contributions. We present a simple new factorization and space subdivision scheme for the FGT. The new approach is based on the fact that the Gaussian, especially in higher dimensions, decays so rapidly that the contributions outside of a certain radius can be safely ignored.

Assuming we have N sources $\{x_i\}$ centered at x_* and M target points $\{y_j\}$, we can rewrite the exponential term as

$$e^{-\|y_j-x_i\|^2/h^2} = e^{-\|\Delta y_j\|^2/h^2}\, e^{-\|\Delta x_i\|^2/h^2}\, e^{2\Delta y_j\cdot\Delta x_i/h^2}, \tag{3.1}$$

where

$$\Delta y_j = y_j - x_*, \qquad \Delta x_i = x_i - x_*.$$

In expression (3.1) the first two exponential terms can be evaluated individually at either the source points or the target points. The only problem left is to evaluate the last term where sources and target coordinates are entangled. One way of breaking the entanglement is to expand it into the series

$$e^{2\Delta y_j\cdot\Delta x_i/h^2} = \sum_{n=0}^{\infty} \Phi_n(\Delta y_j)\Psi_n(\Delta x_i), \tag{3.2}$$

where Φ_n and Ψ_n are the expansion functions and will be defined in the next section. Denoting $\phi(\Delta y_j) = e^{-\|\Delta y_j\|^2/h^2}$, $\psi(\Delta x_i) = e^{-\|\Delta x_i\|^2/h^2}$, we can rewrite the sum (1.1) as

$$G(y_j) = \sum_{i=1}^{N} q_j \phi(\Delta y_j)\psi(\Delta x_i) \sum_{n=0}^{\infty} \Phi_n(\Delta y_j)\Psi_n(\Delta x_i). \tag{3.3}$$

If the infinite series (3.2) absolutely converges, we can truncate it after p terms so as to obtain a desired precision. Exchanging the summations in (3.3), we obtain

$$G(y_j) = \phi(\Delta y_j)\sum_{n=0}^{p-1} C_n \Phi_n(\Delta y_j) + \epsilon(p), \tag{3.4}$$

$$C_n = \sum_{i=1}^{N} q_i \psi(\Delta x_i)\Psi_n(\Delta x_i). \tag{3.5}$$

The factorization (3.4) is the basis of our algorithm. In the following sections, we will discuss how to implement it in an efficient way.

3.2. Multivariate Taylor Expansions. The key issue to speed up the FGT is to reduce the factor p^d in the computational complexity. The factor p^d arises from the way that the multivariate Gaussian is treated as the product of univariate Gaussian functions and expanded along each dimension. To reduce this factor, we treat the dot product in (3.2) as a *scalar variable* and expand it via the Taylor expansion. The expansion functions Φ and Ψ are expressed as multivariate polynomials [25].

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

$$p(x) = \sum_{|\alpha|\le n} C_\alpha x^\alpha, \qquad C_\alpha \in \mathbf{R}. \tag{3.6}$$

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

$$\begin{aligned}&(0,0,\dots,0),\ (1,0,\dots,0),\ (0,1,\dots,0),\ \dots,\ (0,0,\dots,1),\\ &(2,0,\dots,0),\ (1,1,\dots,0),\ \dots,\ (0,0,\dots,2),\ \dots\dots,\ (0,0,\dots,n).\end{aligned}$$

If we expand the univariate Gaussian function e^v into the Taylor series:

$$e^v = \sum_{n=0}^{\infty} \frac{v^n}{n!}, \tag{3.7}$$

and let $v = 2x\cdot y = 2(x_1y_1 + \cdots + x_dy_d)$, we have

$$v^n = \sum_{|\alpha|=n} 2^n \binom{n}{\alpha} x^\alpha y^\alpha, \tag{3.8}$$

where $\binom{n}{\alpha} = \frac{n!}{\alpha_1!\cdots\alpha_d!}$ are the multinomial coefficients, and the terms of x^α and y^α are sorted in graded lexicographic order. So we have the following multivariate Taylor expansion of the Gaussian functions

$$e^{2x\cdot y} = \sum_{\alpha\ge 0} \frac{2^{|\alpha|}}{\alpha!} x^\alpha y^\alpha. \tag{3.9}$$

From Eqs.(3.1), (3.4) and (3.9), the weighted sum of Gaussians (1.1) can be expressed as a multivariate Taylor expansions about center x_*:

$$G(y_j) = \sum_{\alpha\ge 0} C_\alpha e^{-\|y_j - x_*\|^2/h^2} \left(\frac{y_j - x_*}{h}\right)^\alpha, \tag{3.10}$$

where the coefficients C_α are given by

$$C_\alpha = \frac{2^{|\alpha|}}{\alpha!} \sum_{i=1}^{N} q_i e^{-\|x_i - x_*\|^2/h^2} \left(\frac{x_i - x_*}{h}\right)^\alpha. \tag{3.11}$$

If we truncate the series after total degree $p-1$, the number of the terms $r_{p-1,d} = \binom{p+d-1}{d}$ is much less than p^d in higher dimensions (as shown in Table 3.1). For instance, when $d = 12$

TABLE 3.1
Number of terms in d-variate Taylor expansion truncated after order $p-1$.

$p\backslash d$	1	2	3	4	5	6	7	8	9	10	11	12
4	4	10	20	35	56	84	120	165	220	286	364	455
5	5	15	35	70	126	210	330	495	715	1001	1365	1820
6	6	21	56	126	252	462	792	1287	2002	3003	4368	6188
7	7	28	84	210	462	924	1716	3003	5005	8008	12376	18564
8	8	36	120	330	792	1716	3432	6435	11440	19448	31824	50388
9	9	45	165	495	1287	3003	6435	12870	24310	43758	75582	125970
10	10	55	220	715	2002	5005	11440	24310	48620	92378	167960	293930

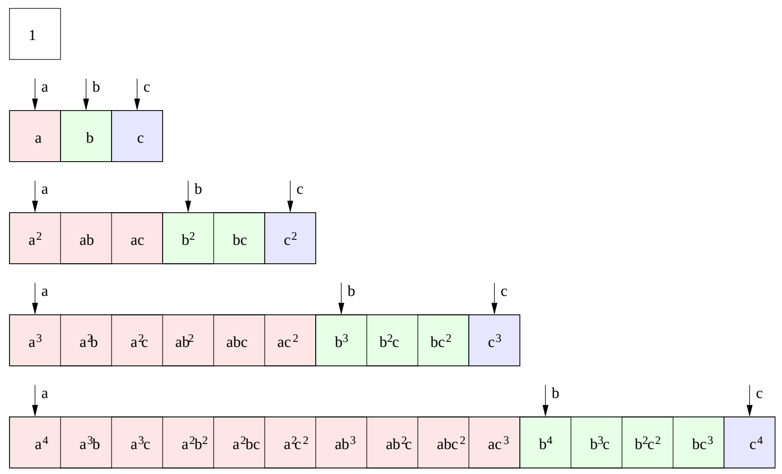

FIG. 3.1. *Efficient expansion of the multivariate polynomials.*

and $p = 10$, the original FGT needs 10^{12} terms, the multivariate Taylor expansion needs only 293930. For $d \longrightarrow \infty$ and moderate p, the number of terms becomes $O(d^p)$, a substantial reduction. The reduction is based on the fact that there are many *like terms* which are combined in the multivariate Taylor expansion (3.9).

One of the benefits of the graded lexicographic order is that the expansion of multivariate polynomials can be performed efficiently. 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 $a^{k-1}, b^{k-1}, c^{k-1}, \ldots$ in Figure 3.1) 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 variable's leading term and the end, as shown in the Figure 3.1. 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.

3.3. Spatial Data Structures. As discussed above, we need to subdivide space into cells and collect the influence of the sources within each cell. The influence on each target can be summarized from its neighboring cells that lie within a certain radius from the target. To efficiently subdivide the space, we must devise a scheme that adaptively subdivides the space according to the distribution of points. It is also desirable to generate cells as compactly as possible (the ideal compact cell is a sphere, though spheres cannot be used to partition space without overlap).

Based on these considerations, we model the space subdivision task as a k-center problem, which is defined as follows: given a set of n points 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

$$\max_i \max_{v \in S_i} \|v - c_i\|.$$

The k-center problem is known to be NP-hard [2]. Gonzalez [12] 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: in iteration i, for every point, compute its distance to the set C: $d_i(v, C) = \min_{c \in C} \|v - c\|$. Let v_i be a point that is farthest away from C, i.e., a 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 [12]:

THEOREM 3.1. *For k-center clustering, the farthest-point clustering computes a partition with maximum radius at most twice the optimum.*

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

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

In the optimal k-centers, 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 $d_k(v_k, C) \le d_i(v_k, C)$ and $d_k(v_k, C) \le d_j(v_k, C)$. Also by definition, we have $d_i(v_k, C) \le d_i(v_i, C)$ and $d_j(v_k, C) \le d_j(v_j, C)$. So we have

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

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 $d_k(v_k, C) \le 2\delta$. □

The proof uses no geometry beyond the triangle inequity, so it holds for *any metric space*. Hochbaum and Shmoys [17] proved that the factor 2 cannot be improved unless $P = NP$. The direct implementation of farthest-point clustering has running time $O(nk)$. Feder and Greene [11] give a two-phase algorithm with optimal running time $O(n \log k)$. The computational complexity of the above implementations is independent of the dimensionality. This simple algorithm and its variants have been successfully embedded into many other algorithms such as clustering and nearest-neighbor search [21, 13, 20].

The predefined number of clusters k can be determined as follows: run the farthest-point algorithm until the maximum radius of clusters decreases to a given distance. The result is bounded by twice the optimum. In practice, the initial point has little influence on the final approximation radius, if number of the points n is sufficiently large. Figure 3.2 displays the results of farthest-point algorithm. In two dimensions, the algorithm leads to a Voronoi tessellation of the space. In three dimensions, the partition boundary resembles the surface of a crystal.

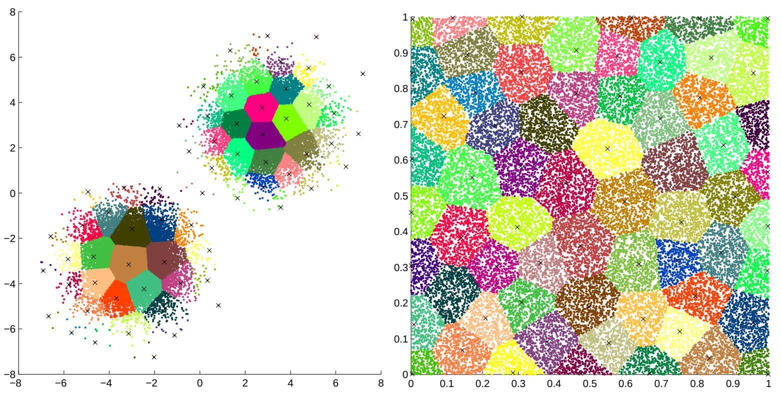

FIG. 3.2. *The farthest-point algorithm divides 40000 points into 64 clusters (with the centers indicated by the crosses) in* 0.48 *seconds on a 900MHZ PIII PC. Left: 2 normal distributions; Right: Uniform distribution.*

3.4. The Algorithm. The improved fast Gauss transform consists of the following steps (as shown in Figure 3.3):

Step 1: Assign the N sources into K clusters using the farthest-point clustering algorithm such that the radius is less than $h\rho_x$.

Step 2: Choose p sufficiently large such that the error estimate (3.14) below is less than the desired precision ϵ.

Step 3: For each cluster S_k with center c_k, compute the coefficients given by the expression (3.11):

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

Step 4: Repeat for each target y_j, find its neighbor clusters whose centers lie within the range $h\rho_y$. Then the sum of Gaussians (1.1) can be evaluated by the expression (3.10):

$$G(y_j) = \sum_{\|y_j - c_k\| \le h\rho_y} \sum_{|\alpha| < p} C_\alpha^k e^{-\|y_j - c_k\|^2/h^2} \left(\frac{y_j - c_k}{h} \right)^\alpha .$$

3.5. Complexity and Error Bounds. The amount of work required in step 1 is $O(NK)$ (for large K, we can use Feder and Greene's $O(N \log K)$ algorithm [11] instead). The amount of work required in step 3 is of $O(N\, r_{pd})$. The work required in step 4 is $O(Mn\, r_{pd})$, where n is the maximum number of the neighbor clusters for each target. For most nonparametric statistics, computer vision and pattern recognition applications, the precision required is moderate, we can get small K and small r_{pd}. Since $n \le K$, the improved fast Gauss transform achieves linear running time. The algorithm needs to store the K coefficients of size r_{pd}, so the storage complexity is reduced to $O(Kr_{pd})$.

The error in the above algorithm arises from two sources. The first is due to truncation of the Taylor series in step 3, and the other due to cutoff of the far field contributions in step 4. The error $E_T(x, p)$ due to truncating the series at source point x after order $p - 1$ satisfies

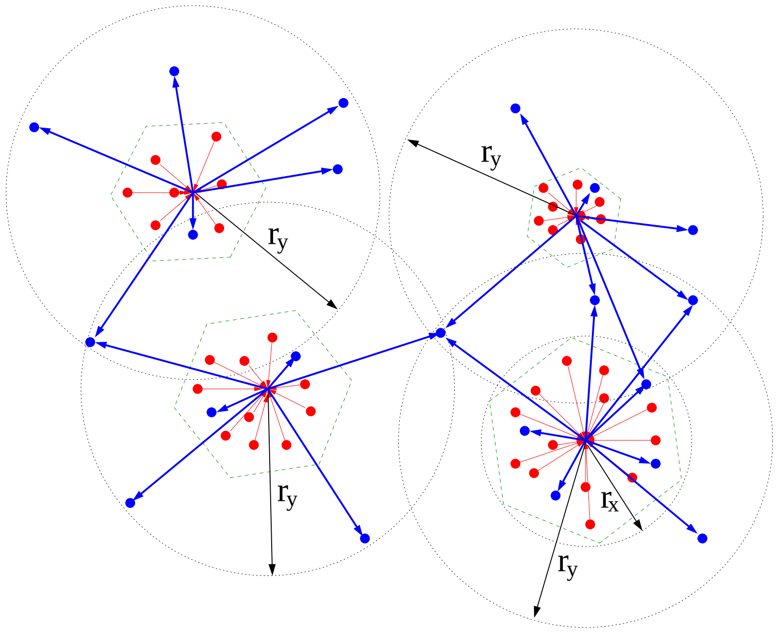

FIG. 3.3. *The improved fast Gauss transform. The sources (red dots) are grouped into k clusters by the farthest-point clustering algorithm. r_x is the radius of the farthest-point clustering algorithm. The contributions on the target (blue dot) outside of the cutoff radius r_y are ignored.*

the bound:

$$\begin{aligned}|E_T(x,p)| &\le e^{-(\|\Delta x\|^2+\|\Delta y\|^2)/h^2}\ \frac{1}{p!}\ e^{2\Delta x\cdot\Delta y/h^2}\ \left(\frac{2\Delta x\cdot\Delta y}{h^2}\right)^p\\ &\le e^{-(\|\Delta x\|^2+\|\Delta y\|^2)/h^2}\ \frac{2^p}{p!}\ e^{2\|\Delta x\|\|\Delta y\|/h^2}\ \left(\frac{\|\Delta x\|\|\Delta y\|}{h^2}\right)^p \\ &\le \frac{2^p}{p!}\ \left(\frac{r_x r_y}{h^2}\right)^p = \frac{2^p}{p!}\ \rho_x^p\rho_y^p.\end{aligned} \tag{3.12}$$

where $\Delta x_i = x_i - x_*$ and $\Delta y_j = y_j - x_*$, r_x is the upper bound of Δx, and r_y is the upper bound of Δy. We also denote the ratios $\rho_x = r_x/h$ and $\rho_y = r_y/h$. The Cauchy inequality

$$\Delta x\cdot\Delta y \le \|\Delta x\|\|\Delta y\|,$$

and the inequality

$$2\|\Delta x\|\|\Delta y\| \le \|\Delta x\|^2 + \|\Delta y\|^2,$$

were used in the above error bound analysis.

The cutoff error $E_C(r_y)$ due to ignoring contributions outside of radius r_y from target point y satisfies the bound:

$$|E_C(r_y)| \le e^{-r_y^2/h^2} = e^{-\rho_y^2}. \tag{3.13}$$

The total error at any target point y satisfies the bound:

$$|E(y)| \le \left|\sum q_j E_T(x,p)\right| + \left|\sum q_j E_C(r_y)\right| \le Q\left(\frac{2^p}{p!}\ \rho_x^p\rho_y^p + e^{-\rho_y^2}\right). \tag{3.14}$$

TABLE 4.1
Running times in milliseconds for direct evaluation, fast Gauss transform and improved fast Gauss transform in three dimensions.

Case	$N = M$	Direct	FGT	IFGT
1	100	2.9	5.5	4.6
2	200	11.4	13.0	12.5
3	400	46.1	37.0	21.1
4	800	184.2	121.8	33.2
5	1600	740.3	446.0	68.1
6	3200	2976.2	1693.8	132.8
7	6400	17421.4	6704.3	263.0
8	12800	68970.2	26138.6	580.2
9	25600	271517.9	103880.8	1422.0

where $Q = \sum |q_j|$.

From above error bound, we can see that p, ρ_x and ρ_y together control the convergence rate. The larger p and ρ_y, the smaller ρ_x, the algorithm converges faster. But the cost of computation and storage increases at the same time. There is always a tradeoff between the speed and the precision. The above error bound is much simpler than the error estimate in [1]. Another interesting fact about the error bound is that it is independent of the dimensionality.

For bandwidth comparable to the range of the data, we can increase the steps of the farthest-point algorithm to decrease the radius r_x. As we seen in Figure 3.4, the radius of farthest-point algorithm always decreases as the algorithm progresses. By this way, we can make $\rho_x\rho_y < 1$, so the error bound (3.14) always converges.

For very small bandwidth (for instance $\rho_x > 10$), the interaction between the sources and targets are highly localized. We can set $p = 0$ which means we directly accumulate the contributions of the neighboring sources and there is no series expansion. All we need is a good nearest neighbor search algorithm [20, 22].

4. Numerical experiments. The first experiment compares the performance of our proposed algorithm with the original fast Gauss transform. Since there is no practical fast Gauss transform in higher dimensions available, we only make comparisons in three dimensions. The sources and targets are uniformly distributed in a unit cube. The weights of the sources are uniformly distributed between 0 and 1. The bandwidth of the Gaussian is $h = 0.2$. We set the relative error bound to 2% which is reasonable for most kernel density estimation in non-parametric statistics where Gauss transform plays an important role, because the estimated density function itself is an approximation. Table 4.1 reports the CPU times using direct evaluation, the original fast Gauss transform (FGT) and the improved fast Gauss transform (IFGT). All the algorithms are programmed in C++ and were run on a 900MHz PIII PC. We can find the running time of the IFGT grows linearly as the number of sources and targets increases, while the direct evaluation and the original FGT grows quadratically, though the original FGT is faster than the direct evaluation. The poor performance of the FGT in 3D is also reported in [10]. This is probably due to the fact that the number of boxes increases significantly by a uniform space subdivision in 3D. The cost to compute the interactions between the boxes grows quadratically. The farthest-point algorithm in the IFGT generates a much better space subdivision and reduces the number of boxes greatly. The multivariate Taylor expansion also reduces the computational cost by a factor 4.3 in 3D (the factor is for order 7, and larger factors in higher dimensions).

The second experiment is to examine the performance of IFGT in higher dimensions. We randomly generate the source and target points in a unit hypercube according to a uniform distribution. The weights of the sources are uniformly distributed between 0 and 1. The

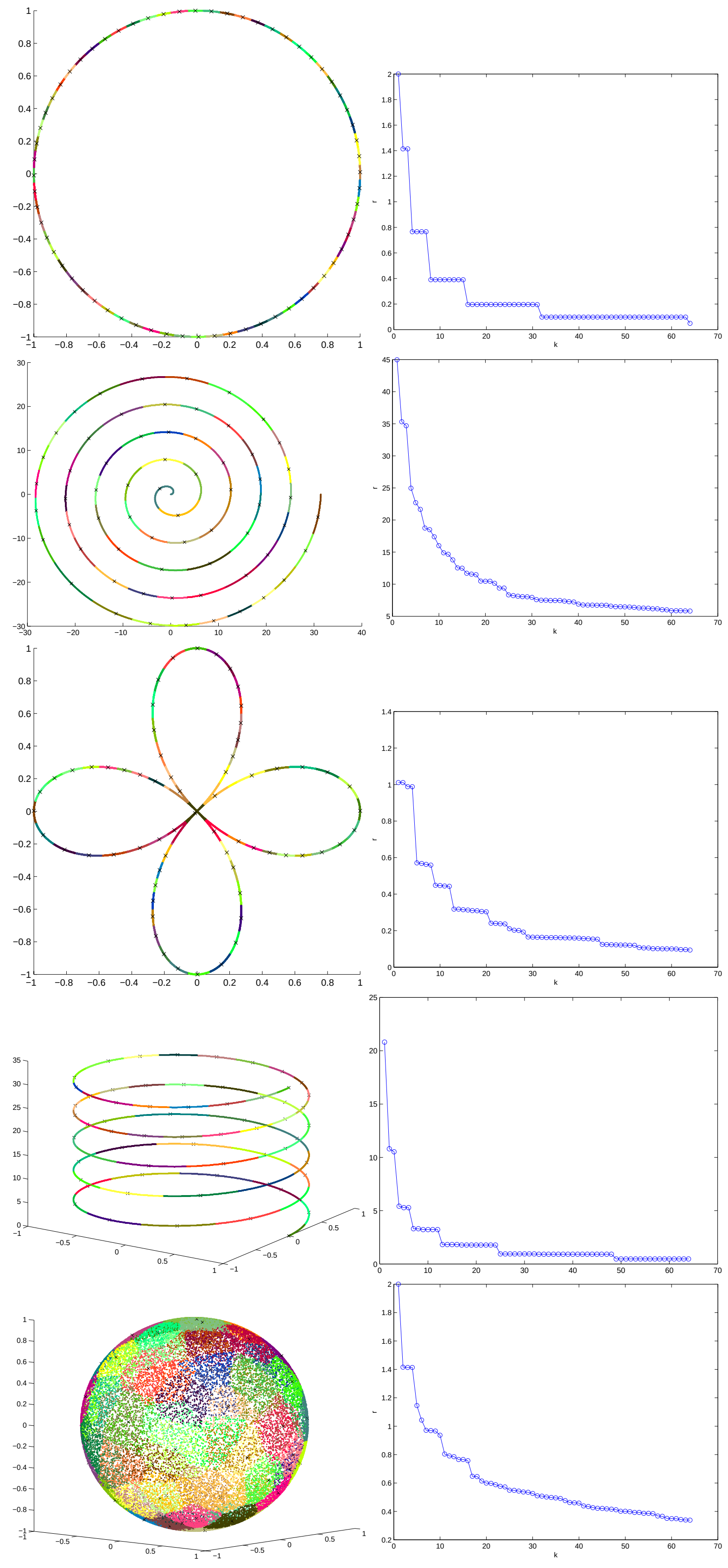

FIG. 3.4. *The radius of the farthest-point algorithm decreases as the algorithm progresses. (Left column) points in 2D or 3D spaces. (Right column) the radius of farthest-point algorithm w.r.t. the steps of the algorithm.*

bandwidth is set to $h = 1$. The results are shown in Fig. 4.1. We compared the running time of the direct evaluation to the IFGT with $h = 1$ and $N = M = 100, \ldots, 10000$. The comparisons are performed in dimensions from 4 to 10 and results in dimensions 4, 6, 8, 10 are reported in Figure 4.1. From the figure we notice that the running time of the direct evaluation grows quadratically with the size of points. The running time of the IFGT grows linearly with the size of the points. In 4, 6, 8, 10 dimensions, the IFGT takes 56ms, 406ms, 619 ms, 1568ms to evaluate the sums on 10000 points, while it takes 35 seconds for a direct evaluation. The maximum relative absolute error as defined in [16] increases with the dimensionality but not with the number of points. The worst relative error occurs in dimension 10, and is below 10^{-3}. We can see that for a 10D problem involving more than 700 Gaussians, the IFGT is faster than direct evaluation, while for a 4D problem the IFGT is faster from almost the outset. We also tested our algorithm on the normal distributions with mean zero, variance one of sources and targets. All data were scaled into unit hypercube. The results are shown in Figure 4.1. We find that the running time is similar to the case of uniform distribution, while the error is much less than the case of uniform distribution.

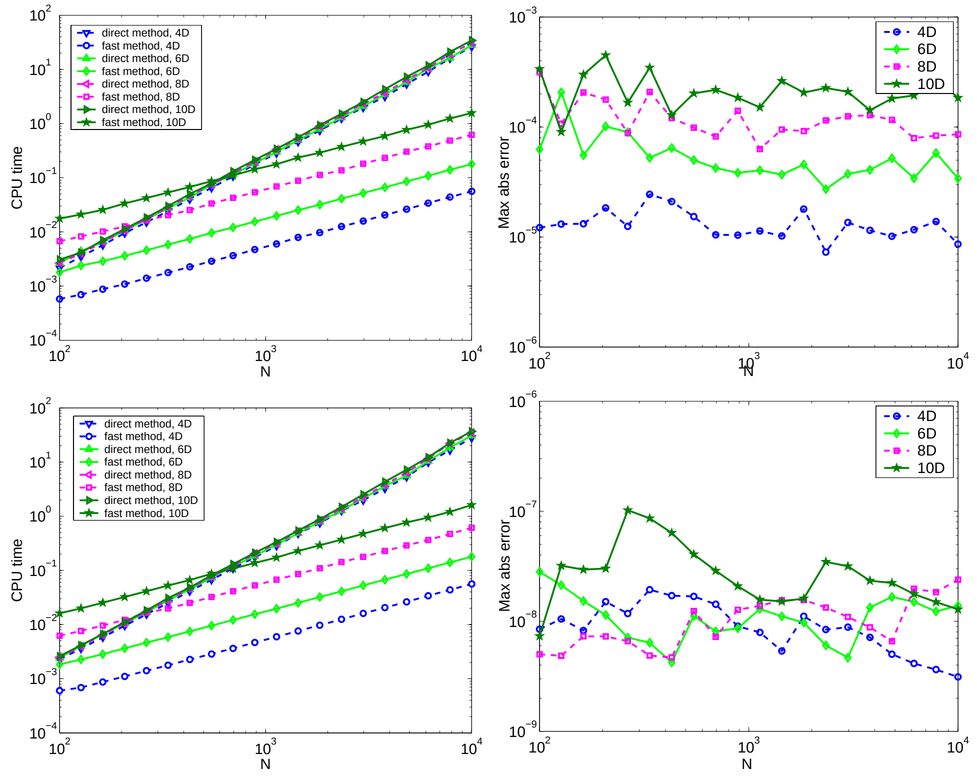

FIG. 4.1. *The running times in seconds (Left column) and maximum relative absolute errors (Right column) of the IFGT ($h = 1$) v.s. direct evaluation in dimensions* $4, 6, 8, 10$ *on uniform distribution (Top row) and normal distribution (Bottom row).*

The third experiment is to examine the error bounds of the IFGT. 1000 source points and 1000 target points in a unit hypercube are randomly generated from a uniform distribution. The weights of the sources are uniformly distributed between 0 and 1. The bandwidth is set to $h = 0.5$. We fix the order of the Taylor series $p = 10$, the radius of the farthest-point clustering algorithm $r_x = 0.5h$, and the cutoff radius $r_y = 6h$, then we vary p, r_x

and r_y, and repeat the experiments in 4 dimensions and 6 dimensions respectively. The comparison between the real maximum absolute errors and the estimated error bounds is shown in Figure 4.2. The estimated error bounds are almost optimal up to a constant factor. The normalized error bounds w.r.t. the order p and the radius r_x with the number of the sources fit the curves of the real errors, which indicates the constant factor for them is roughly the number of the sources. In the case of small cutoff radius, the constant factor is smaller because influence on each target point is highly localized which seems the sources points far away are vanished. The estimated error bounds are useful for choosing the parameters of the IFGT.

5. Conclusions. We have proposed an improved fast Gauss transform that leads to a significant speedup with a major reduction in the amount of the memory required in higher dimensions. A multivariate Taylor expansion is applied to the improved fast Gauss transform which substantially reduces the number of the expansion terms in higher dimensions. The k-center algorithm is utilized to efficiently and adaptively subdivide the higher dimensional space according to the distribution of the points. A simpler and more accurate error estimate is reported, due to the simplification made by the new Taylor expansion and space subdivision schemes. The improved fast Gauss transform is capable of computing the Gauss transform in dimensions as high as tens which commonly occur in nonparametric statistics, pattern recognition. The behaviors of the algorithm in very high dimensional space (such as up to several hundreds) will be studied and reported.

REFERENCES

[1] B. J. C. Baxter and G. Roussos, *A new error estimate of the fast Gauss transform*, SIAM Journal on Scientific Computing, 24 (2002), pp. 257–259.

[2] M. Bern and D. Eppstein, *Approximation algorithms for geometric problems*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing Company, Boston, 1997.

[3] A. H. Boschitsch, M. O. Fenley, and W. K. Olson, *A fast adaptive multipole algorithm for calculating screened coulomb (yukawa) interactions*, Journal of Computational Physics, 151 (1999), pp. 212–241.

[4] M. Broadie and Y. Yamamoto, *Application of the fast Gauss transform to option pricing*, in 5th Columbia JAFEE Conference on Mathematics of Finance, New York, 2002.

[5] J. C. Carr, R. K. Beatson, J. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, *Reconstruction and representation of 3D objects with radial basis functions*, in ACM SIGGRAPH 2001, Los Angeles, CA, 2001, pp. 67–76.

[6] W. C. Chew, J. M. Jin, E. Michielssen, and J. M. Song, eds., *Fast and Efficient Algorithms in Computational Electromagnetics*, Artech House, 2001.

[7] E. Darve, *The fast multipole method (i): Error analysis and asymptotic complexity*, SIAM Numerical Analysis, 38 (2000), pp. 98–128.

[8] ———, *The fast multipole method: Numerical implementation*, Journal of Computational Physics, 160 (2000), pp. 195–240.

[9] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, John Wiley & Sons, New York, 2000.

[10] A. Elgammal, R. Duraiswami, and L. Davis, *Efficient non-parametric adaptive color modeling using fast Gauss transform*, in Proc. IEEE Conf. Computer Vision and Pattern Recognition, Kauai, Hawaii, 2001.

[11] T. Feder and D. Greene, *Optimal algorithms for approximate clustering*, in Proc. 20th ACM Symp. Theory of computing, Chicago, Illinois, 1988, pp. 434–444.

[12] T. Gonzalez, *Clustering to minimize the maximum intercluster distance*, Theoretical Computer Science, 38 (1985), pp. 293–306.

[13] A. Gray and A. Moore, *Rapid evaluation of multiple density models*, in Artificial Iintelligence and Statistics, 2003.

[14] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.

[15] L. Greengard and J. Strain, *A fast algorithm for the evaluation of heat potentials*, Comm. Pure Appl. Math., 43 (1990), pp. 949–963.

[16] ———, *The fast Gauss transform*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 79–94.

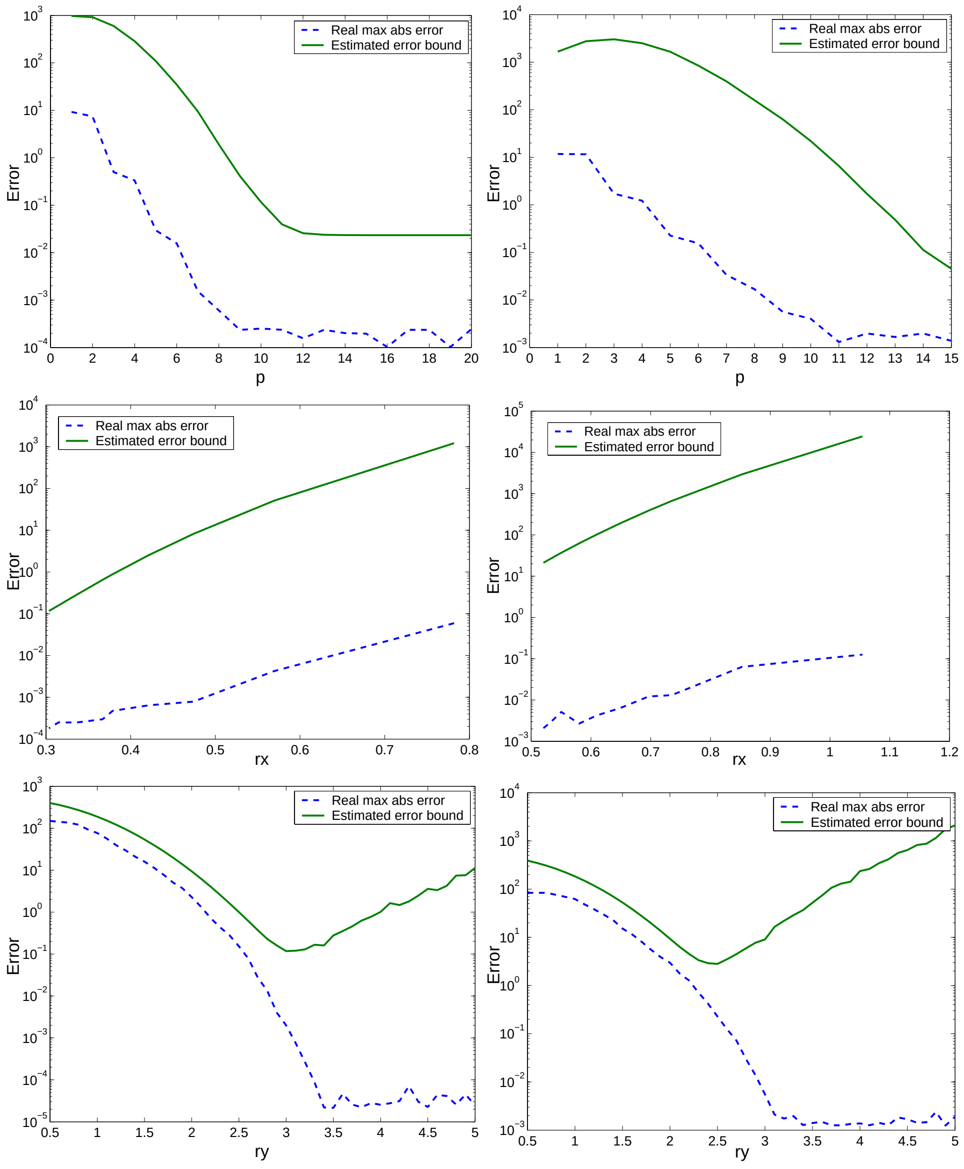

FIG. 4.2. *The comparison between the real maximum absolute errors and the estimated error bounds w.r.t. the order of the Taylor series p (Top row), the radius of the farthest-point clustering algorithm r_x (Middle row), and the cutoff radius r_y (Bottom row). The uniformly distributed sources and target points are in 4 dimensions (Left column) and in 6 dimensions (Right column).*

[17] D. S. HOCHBAUM AND D. B. SHMOYS, *A best possible heuristic for the k-center problem*, Mathematics of Operations Research, 10 (1985), pp. 180–184.

[18] K. N. KUDIN AND G. E. SCUSERIA, *Linear-scaling density-functional theory with Gaussian orbitals and periodic boundary conditions: Efficient evaluation of energy and forces via the fast multipole method*, Phys. Rev. B, 61 (2000), pp. 16440–16453.

[19] C. G. LAMBERT, S. E. HARRINGTON, C. R. HARVEY, AND A. GLODJO, *Efficient on-line nonparametric kernel density estimation*, Algorithmica, 25 (1999), pp. 37–57.

[20] L. MICÓ, J. ONCINA, AND E. VIDAL, *A new version of the nearest-neighbour approximating and eliminating search algorithm (*AESA*) with linear preprocessing-time and memory requirements*, Pattern Recognition

Letters, 15 (1994), pp. 9–17.

[21] A. MOORE, *The anchors hierarchy: Using the triangle inequality to survive high-dimensional data*, in Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence, AAAI Press, 2000, pp. 397–405.

[22] D. M. MOUNT AND S. ARYA, *Ann: Library for approximate nearest neighbor searching*, in Proc. Center for Geometric Computing Second Ann. Fall Workshop Computational Geometry, 1997.

[23] D. W. SCOTT, *Multivariate Density Estimation: Theory, Practical, and Visualization*, Wiley, New York, 1992.

[24] J. STRAIN, *The fast Gauss transform with variable scales*, SIAM Journal on Scientific and Statistical Computing, 12 (1991).

[25] J. VON ZUR GATHEN AND J. GERHARD, *Modern Computer Algebra*, Cambridge University Press, Cambridge, UK, 1999.