The Fast Gauss Transform
with Variable Scales *

John Strain †
Courant Institute of Mathematical Sciences
251 Mercer Street
New York, NY 10012

20 April 1990


**Keywords:** Gauss Transform, Fast Algorithms, Hermite Polynomials.

**AMS (MOS) Subject Classifications:** 65R10, 33A65, 35K05, 62G20, 65P05.

**Electronic Mail:** strain@math.princeton.edu

**Running Head:** The Fast Gauss Transform

---

*Supported by DARPA/AFOSR Contract No. F-49620-87-C-0065 and a NSF Mathematical Sciences Postdoctoral Research Fellowship.

†Current address: Department of Mathematics, Princeton University, Princeton, NJ 08544
Abstract

This paper presents a fast algorithm which evaluates a d-dimensional Gaussian convolution sum with scales which vary from point to point. This algorithm evaluates the sum of $N$ Gaussians at $M$ arbitrarily distributed points in $O(N + M)$ work, where $C$ depends only on the precision required and the essential minimum of the scales. It achieves a speedup of 1776 with $d = 2, N = M = 100,000$ random uniformly distributed points in $[0, 1]^2$, and scales bounded below by $1/100$, and breaks even at less than 50 points.

1 Introduction

Many numerical calculations require the evaluation of $d$-dimensional Gaussian convolution sums having one of the two forms

\[(1) \quad S f(t_i) = \sum_{j=1}^{N} f_j e^{-|t_i-s_j|^2/\delta_j} \]

or

\[(2) \quad T f(t_i) = \sum_{j=1}^{N} f_j e^{-|s_j|^2/\delta_i} \]

at a collection of $M$ points $t = t_i \in \mathbb{R}^d$. Here $s_j$ are given points in $\mathbb{R}^d$, $|x|^2 = x_1^2 + \cdots + x_d^2$, and $\delta_i$ are given positive real numbers with a lower bound $\delta_m$. The difference between (1) and (2) is that in (1) the “scale” $\delta$ depends on the source index $j$, while in (2) $\delta$ depends on the target index $i$.

Since this calculation requires the application of an $M \times N$ matrix to an $N$-vector, it would seem to require $O(MN)$ computational work, making it CPU-intensive when $M$ and $N$ are large. Since we begin with $O(N)$ data and get $O(M)$ output values, it is natural to seek an algorithm which requires only $O(N + M)$ work. Fortunately, the matrix has enough structure to make this possible.

This paper presents an algorithm for evaluating (1) and (2) in $O(M + N)$ time, with the constant factor in $O(N + M)$ depending only on the precision desired and the lower bound $\delta_m$ of the scales $\delta_i$ or $\delta_j$. Our algorithm is an extension of the scheme presented in [5], which treated the special case
when $\delta_j = \delta$ are all the same. It combines Taylor or Hermite expansion with scaling and direct evaluation, after organizing either sources or targets into a convenient data structure. The algorithms for source-dependent (1) and target-dependent (2) scales are similar in organization.

§2 of the paper reviews the necessary facts about Hermite expansions, §3 presents the source-dependent scale algorithm and §4 treats the target-dependent case. §5 presents numerical examples which show that our algorithm is much faster than direct evaluation for large-scale problems, achieving a speedup of 2400 when $N = M = 100,000$ and the scales are not too small. §6 presents our conclusions.

## 2 Hermite expansions

This section describes the properties of Hermite expansions we will need. Good references for this material are [1, 2, 4] and particularly Hille’s classical paper [6].

The Hermite polynomials $H_n(t)$ are defined by the Rodrigues formula

$$H_n(t) = (-1)^n e^{t^2} D^n e^{-t^2} \quad t \in \mathbb{R}$$

where $D = d/dt$. This implies the generating function

$$e^{2s - s^2} = \sum_{n=0}^{\infty} \frac{s^n}{n!} H_n(t).$$

Multiplying through by $e^{-t^2}$ gives

$$e^{-(t-s)^2} = \sum_{n=0}^{\infty} \frac{s^n}{n!} h_n(t),$$

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

$$(3) \quad h_n(t) = e^{-t^2} H_n(t).$$

(Note that these are not the usual orthonormal Hermite functions; these Hermite functions form a biorthogonal set with Hermite polynomials, rather than themselves forming an orthogonal set.) In practice, we will use a shifted

$$3$$
and scaled version of this formula: for $s_0 \in \mathbb{R}$ and $\delta > 0$, we have

$$e^{-(t-s)^2/\delta} = e^{-(t-s_0-(s-s_0))^2/\delta}$$

$$= \sum_{n=0}^{\infty} \frac{1}{n!} \left( \frac{s-s_0}{\sqrt{\delta}} \right)^n h_n \left( \frac{t-s_0}{\sqrt{\delta}} \right)$$

$$= e^{-(t-s_0)^2/\delta} \sum_{n=0}^{\infty} \frac{1}{n!} \left( \frac{s-s_0}{\sqrt{\delta}} \right)^n H_n \left( \frac{t-s_0}{\sqrt{\delta}} \right).$$

This formula tells us how to evaluate the Gaussian field $e^{-(t-s)^2/\delta}$ at the target $t$ due to the source at $s$, as an Hermite expansion centered at $s_0$. Thus we can shift a Gaussian centered at $s$ to a sum of Hermite polynomials times a Gaussian, all centered at $s_0$.

We can also interchange $t$ and $s$ to write

$$e^{-(t-s)^2/\delta} = \sum_{n=0}^{\infty} \frac{1}{n!} h_n \left( \frac{s-t_0}{\sqrt{\delta}} \right) \left( \frac{t-t_0}{\sqrt{\delta}} \right)^n. \quad (4)$$

Looked at this way, the expansion expresses a Gaussian with target $t$ as a Taylor series about a nearby target $t_0$; the coefficients of the Taylor series are the Hermite functions evaluated at $t_0$. Thus the same expansion serves as both a near-field (Taylor) and a far-field (Hermite) expansion.

The final one-dimensional results we will need are the recurrence relation

$$h_{n+1}(t) = 2t h_n(t) - 2n h_{n-1}(t) \quad t \in \mathbb{R},$$

for Hermite functions and Cramer’s inequality for Hermite polynomials:

$$|H_n(t)| \leq K 2^{n/2} \sqrt{n!} e^{t^2/2}$$

where $K$ is a numerical constant less than 1.09 in value.

In summarizing the extensions of these facts to $d > 1$ dimensions, we will find it convenient to adopt multiindex notation. A $d$-dimensional multiindex $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_d)$ is a $d$-tuple of nonnegative integers, playing the role of a multidimensional index. For any multiindex $\alpha$ and any $t \in \mathbb{R}^d$, we define

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

$$\alpha! = \alpha_1! \alpha_2! \ldots \alpha_d!$$

$$t^\alpha = t_1^{\alpha_1} t_2^{\alpha_2} \ldots t_d^{\alpha_d}$$

4
\[ D^\alpha = \partial_1^{\alpha_1} \partial_2^{\alpha_2} \ldots \partial_d^{\alpha_d} \]

where \( \partial_i \) is differentiation with respect to the \( i \)th coordinate in \( \mathbb{R}^d \). If \( p \) is an integer, we say \( \alpha \geq p \) if \( \alpha_i \geq p \) for \( 1 \leq i \leq d \).

The multidimensional Hermite polynomials and Hermite functions are then defined by

\[ h_\alpha(t) = e^{-|t|^2} H_\alpha(t) = h_{\alpha_1}(t_1) \ldots h_{\alpha_d}(t_d) \]

where \( |t|^2 = t_1^2 + \ldots + t_d^2 \).

The Hermite expansion of a Gaussian in \( \mathbb{R}^d \) is then simply

\[ e^{-|t-s|^2} = \sum_{\alpha \geq 0} \frac{(t-s_\alpha)^\alpha}{\alpha!} h_\alpha(s-s_0). \]

Cramer’s inequality implies

\[ \frac{1}{\alpha!} |h_\alpha(t)| \leq K e^{-|t|^2/2} 2^{l/2} \frac{1}{\sqrt{\alpha!}}, \]

where \( K \) is less than \( (1.09)^d \).

We can now present the two key expansions on which our algorithms rely.

First, scale the Hermite expansion (6) by \( \delta_j \) to get

\[ e^{-|t-s|^2/\delta_j} = \sum_{\beta \geq 0} \frac{1}{\beta!} h_\beta \left( \frac{s_j - t_C}{\sqrt{\delta_j}} \right) \left( \frac{t - t_C}{\sqrt{\delta_j}} \right)^\beta \]

\[ = \sum_{\beta \geq 0} \frac{1}{\beta!} \left( \frac{t - t_C}{\sqrt{\delta_j}} \right)^\beta \left( \frac{s_j - t_C}{\sqrt{\delta_j}} \right) \left( \frac{t - t_C}{\sqrt{\delta_j}} \right)^\beta. \]

This expresses a Gaussian centered at \( s_j \) as a Taylor series about an arbitrary target \( t_C \), with a scaling factor \( \delta \) independent of \( j \). By Cramer’s inequality, the error in truncating the Taylor series (8) after the first \( p^d \) terms is bounded by

\[ |E_p| \leq K \sum_{\beta \geq p} \frac{1}{\beta!} 2^{l/2} \left( \frac{|t - t_C|}{\sqrt{\delta_j}} \right)^{|\beta|}. \]

Here \( |t|_\infty = \max_{1 \leq i \leq d} |t_i| \). Suppose \( t \) lies in a box \( C \) with center \( t_C \) and sides of length \( r \sqrt{2\delta} \leq r \sqrt{2\delta_j} \) parallel to the axes. Then Stirling’s formula implies that

\[ |E_p| \leq \frac{r^p}{1 - r_p} \]

5
whenever \( r^2_p = e r^2 \delta / \delta_j (p + 1) < 1 \). This error bound decreases very rapidly as \( p \) increases. If \( r = 1 \) and \( \delta_j = \delta \), then \( |E_p| \leq 10^{-4} \) when \( p = 12 \) and \( |E_p| \leq 10^{-6} \) when \( p = 17 \). Even if \( r = 2 \), accuracy \( 10^{-4} \) is attained with \( p = 24 \) and accuracy \( 10^{-6} \) with \( p = 31 \). In practice, this error bound overestimates the error by two orders of magnitude or more.

This error bound is more useful than the (erroneous) bound used in [5], because that bound required \( r < 1 \). The current bound allows us to use \( r \geq 1 \) if we use more terms in the Hermite series. This is practically useful because we can cluster the points into larger groups and use longer vectors, which makes the code run much faster on vector processors.

Our second formula is the dual of the first, obtained by scaling with a target-dependent scale \( \delta_i \) rather than \( \delta_j \) and shifting to a source center \( s_B \) rather than a target center \( t_C \). We find

\[
e^{-|t_i - s_j| \delta_i / \delta_j} = \sum_{\beta \geq 0} \frac{1}{\beta!} \left( \frac{\delta_i}{\delta_j} \right)^{\beta/2} h_{\beta} \left( \frac{t_i - s_B}{\sqrt{\delta_i}} \right) \left( \frac{s_j - s_B}{\sqrt{\delta_j}} \right)^{\beta}.
\]

This expresses a Gaussian centered at \( s_j \) as a scaled Hermite series centered at a nearby source box center \( s_B \), with coefficients independent of \( i \). If the source \( s_j \) lies inside a box of side length \( r \sqrt{2 \delta_i} \) centered at \( s_B \), then the error in truncating the Hermite series after the first \( p^d \) terms is bounded by (9).

These two formulas provide the basis for the two fast algorithms developed in this paper.

### 3 Source-dependent Scales

In this section, we present our algorithm for evaluating

\[
Sf(t_i) = \sum_{j=1}^{N} f_j e^{-|t_i - s_j| \delta_j / \delta_j}
\]

at \( M \) targets \( t_i \). We assume (by scaling the \( \delta_j \)'s if necessary) that all the targets lie in the unit box \( B_0 = [0, 1]^d \), for convenience of exposition. The sources \( s_j \) may lie anywhere in \( \mathbb{R}^d \).

Our algorithm requires \( O(M + N) \) work; the constant factor in \( O(M + N) \) depends only on the precision \( \epsilon \) and the minimum \( \delta_m \) of the scales \( \delta_j \).
Choose cutoffs $a$ and $b$ and split the scales into three groups: $0 \leq \delta \leq a$, $a < \delta < b$, and $b \leq \delta$. We treat each group by an appropriate method, and choose $a$ and $b$ to minimize the work required.

First consider $0 \leq \delta \leq a$. Gaussians with small scales decay rapidly away from the source, so each of these sources can affect only targets which are quite nearby. Hence we treat these sources by direct evaluation, with a data structure constructed as in [11].

To do this, let $R$ be the range of the longest-range Gaussian in this group; thus $e^{-R^2/a} \leq \epsilon$ and $R = \sqrt{-a \log \epsilon}$. Divide $[0,1]^d$ into $n^d$ boxes of side length $\geq R$, so $n = [1/R]$. Sort the targets $t_i$ into boxes by location. Now run through the sources $s_j$. Put $s_j$ into a box, $B$ say, by location, and let $k_j = \lfloor \sqrt{-\delta_j/a} \rfloor$. Then $s_j$ affects boxes with indices within $k_j$ of $B$, to accuracy $\epsilon$. Run through these boxes, incrementing $Sf(t_i)$ at each target $t_i$ in each box. This concludes the treatment of sources with $0 \leq \delta_j \leq a$.

Now consider the large scales, with $b \leq \delta_j < \infty$. These Gaussians have very long ranges, so direct evaluation is very expensive. For the same reason, however, they are very smooth. If $b$ is chosen large enough, all these sources can be incorporated into a single $p^d$-term truncated Taylor expansion about the box center $t_0 = (1/2, \ldots, 1/2)$ of $[0,1]^d$, with error less than $Fe$ where $F = \sum |f_j|$. Indeed, (8) implies that

\[
\sum_{\delta_j \geq b} f_j e^{-|t-t_j|/\delta_j} = \sum_{\beta \leq p} C_\beta \left( \frac{t-t_0}{\sqrt{\delta}} \right)^\beta + E_p
\]

where

\[
C_\beta = \frac{1}{\beta!} \sum_{\delta_j \geq b} \left( \frac{\delta_j}{\delta} \right)^{|\beta|/2} h_\beta \left( \frac{s_j-t_0}{\sqrt{\delta_j}} \right)
\]

and

\[
E_p \leq F \frac{r_p}{1-r_p} \quad r_p = \frac{e}{p+1} r < 1.
\]

Here $r = 1/\sqrt{2\delta} \leq 1/\sqrt{2b}$ and $F = \sum |f_j|$.

Thus the large variances are very easy to deal with; we simply form $p^d$ coefficients $C_\beta$ at a cost of $O(p^dN)$ work and evaluate the Taylor series at $M$ targets, with a total cost of $O(p^d(M+N))$. The coefficient $p^d$ depends only on the desired precision $\epsilon$.

Finally, we must deal with the intermediate sources with $a < \delta_j < b$. We treat these by a combination of the previous two methods. The targets are
sorted into boxes and each source contributes to a Taylor expansion about the center of each box within range. When each Taylor expansion is complete, it is evaluated at each target in the box.

We begin by sorting the targets into boxes $C$ say, with centers $t_C$ and sides of lengths $r \sqrt{2\delta}$ parallel to the axes. Here $\delta = \min_{a \leq \delta_j \leq b} \delta_j \geq a$. Now each source is converted into a Taylor expansion (8) about the center $t_C$ of each target box $C$ within range $R_j = \sqrt{-\delta_j \log \epsilon}$ of $s_j$. Thus we have, for $t \in C$,

$$\sum_{|s_j - t_C| \leq R_j} F_j e^{-r_j |s_j|^2} = \sum_{\beta \leq p} C_{\beta} \left( \frac{t - t_C}{\sqrt{\delta}} \right)^\beta + E_p$$

where

$$C_{\beta} = \frac{1}{\beta!} \sum_{\delta_j \geq b} \left( \frac{\delta}{\delta_j} \right)^{\beta/2} h_{\beta} \left( \frac{s_j - t_C}{\sqrt{\delta_j}} \right)$$

and $E_p$ satisfies the error bound

$$E_p \leq F \frac{r_p^p}{1 - r_p}$$

with $r_p = \sqrt{\frac{p}{p+1}} < 1$. We choose $r$ and $p$ to make $|E_p| \leq F \epsilon$. After each source has contributed to all Taylor expansions in range, each Taylor expansion is evaluated at the targets $t_i$ in its box.

Two slight refinements of this procedure reduce the overhead considerably in practice. First, it is inefficient to form a Taylor series and evaluate it for a box containing only a few targets. Thus a breakpoint $M_T$ is introduced, boxes with more than $M_T$ targets receive Taylor expansions, while direct evaluation is more efficient for boxes with fewer than $M_T$ targets. Typically $M_T = 8$ or $10$ is a reasonable value when single precision accuracy is desired. The right value of $M_T$ can be determined by the program when it is run, in a reasonably straightforward way.

The second refinement is introduced in order to avoid a needlessly accurate computation. If a scaling factor $\delta = \min \delta_j$ is used to determine the box size, then many sources will have $\delta_j$ much larger than $\delta$. For these sources, $p^d$ terms is far too many to provide the desired accuracy, and $p$ should be reduced. Hence we can compute how many terms are necessary for each source independently, and increment only those coefficients. Thus we compute $q^d$ terms where $q$ is chosen to satisfy the error estimate

$$\frac{r_{qj}^d}{1 - r_{qj}} \leq \epsilon$$
with \( r_{q_i} = \sqrt{e^\delta / \delta_j(q + 1)} \). The small overhead required to compute \( q \) for each source is compensated for by the reduction in effort required to evaluate Hermite coefficients. In practice, this refinement often produces a factor of two speedup in the calculation.

A third refinement is possible; one can subdivide the center interval \((a, b)\) further, in order to avoid treating longer-range smoother Gaussians on wastefully small scales. A geometric subdivision as in [12] is possible, or some other problem-dependent subdivision may be more appropriate.

The algorithm is completed by the choice of the cutoffs \( a \) and \( b \). If the scales \( \delta_j \) are distributed in a fairly uniform way over the interval \([\delta_m, \delta_M]\), then one can take \( a = \delta_m + O(1/N) \) and \( b \) fixed, say \( b = 1/10 \). The choice of \( a \) is made because we can afford to have only a fixed number of sources below \( a \). The choice of \( b = 1/10 \) is about the smallest for which we can make the Hermite series accurate with a practical number of terms, like \( 20^d \) or \( 30^d \). The choice of \( a \) is more important when there are a few outlying scales \( \delta_j \), with all the rest lying considerably higher. Then we should choose \( a \) to be the essential minimum of the \( \delta_j \)'s, in other words the smallest lower bound for all but a fixed number of the scales as \( N \to \infty \). Thus only the outliers will be less than \( a \). This can substantially improve the performance of the medium scales, because we can take larger boxes, and the cost of evaluating a few sources directly is then worth it.

## 4 Target-dependent Scales

In this section, we describe how to evaluate

\[
T_f(t_i) = \sum_{j=1}^{N} f_j e^{-|t_i - s_j|^2 / \delta_i}
\]

at \( M \) points \( t_i \in \mathbb{R}^d \), under the assumption (for convenience) that all the sources \( s_j \) lie in \([0, 1]^d\). The algorithm is structurally dual to the algorithm of §3, so we concentrate on the differences.

We divide the \( \delta_i \)'s into three sets in the same way as before, and treat each set of targets as a group.

For targets with \( 0 \leq \delta_i \leq a \), we sort the sources into boxes rather than the targets, and apply direct evaluation.
For the middle interval $a < \delta_i < b$, we use the Hermite expansion (10) with $\delta = \min_{a \leq \delta_i \leq b} \delta_i \geq a$. We impose a box structure on the sources and collect all the sources in each box $B$ (with center $s_B$ and sides of length $r \sqrt{2\delta}$ parallel to the axes) into a scaled Hermite expansion about the box center, valid everywhere in $[0, 1]^d$:

$$T_B f(t_i) = \sum_{s_j \in B} f_j e^{-\|t_i - s_j\|^2/\delta_i} = \sum_{\beta \leq p} B_\beta \left( \frac{\delta}{\delta_i} \right)^{\beta/2} h_\beta \left( \frac{t_i - s_B}{\sqrt{\delta_i}} \right)^\beta + E_p$$

where

$$B_\beta = \frac{1}{\beta!} \sum_{s_j \in B} f_j \left( \frac{s_j - s_B}{\sqrt{\delta}} \right)^\beta$$

and we make $E_p \leq F\epsilon$ by choice of $r$ and $p$. Then we evaluate a fixed number of Hermite expansions at each target $t_i$:

$$T f(t_i) = \sum_{|t_i - t_j| \leq R_i} T_B f(t_i),$$

with $R_i = \sqrt{-\delta_i \log \epsilon}$. Some overhead can be avoided if we form Hermite series only for boxes with more than say $N_S$ sources; boxes with fewer than $N_S$ sources are handled by direct evaluation. Also, targets with $\delta_i$ much larger than $\delta$ need only evaluate a smaller Hermite series, since the error decays faster for them.

Finally, the sources with $\delta_i \geq b$ are handled by forming all of them into a single Hermite expansion about the center of $[0, 1]^d$ and evaluating it at each target.

The parameters $a$, $b$ and $N_S$ are chosen in a way very like the source-dependent algorithm.

5 Numerical Results

In this section, we present some numerical results showing the speedups achievable by our algorithms. We tested the source-dependent algorithm on sets of $N = M$ sources and targets in $d = 2$ dimensions; the target-dependent algorithm will exhibit highly similar statistics. We used random uniformly distributed strengths on $[0, 1]$, with source and target points uniformly distributed either on $[0, 1]^d$ or on a circle in $[0, 1]^d$. The scales were uniformly distributed on $[\delta_m, \delta_M]$ for various choices of $\delta_m$ and $\delta_M$. 

10
Tables 1-4 present results for the source-dependent scale fast Gauss transform for four examples. The time taken by the fast algorithm is recorded in the column headed $T_f$, the time required to evaluate the Gaussian sum directly under $T_d$, and the error produced by the fast algorithm — relative to $F = \sum |f_j|$ — under the heading $E_p/F$. All computations were carried out in double precision arithmetic with the error tolerance $\epsilon$ set to $10^{-4}$. (This was enough to achieve an actual error well below $10^{-6}$ in every case.) This required $p = 10$ with $r = 1$ for sources with $a \leq \delta_j \leq b$ and $p = 24$ for $\delta_j > b$. In Table 1, we used source and target points uniformly distributed on $[0,1]^d$ and scales $\delta_j$ uniformly distributed between $\delta_m = 0.1$ and $\delta_M = 1.0$. We took $a = 0$, $b = 0.125$, and $M_T = 8$. The results show a beautifully linear growth of computation time with $N$ for the fast algorithm, against a quadratic time for direct evaluation. A speedup of 2275 is obtained with $N = 102400$, indicating a projected breakeven point of 45.

<table>
<thead>
<tr>
<th>Case</th>
<th>$N = M$</th>
<th>$T_f$</th>
<th>$T_d$</th>
<th>$E_p/F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>0.05</td>
<td>0.10</td>
<td>0.222e-06</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>0.09</td>
<td>0.40</td>
<td>0.163e-06</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>0.18</td>
<td>1.60</td>
<td>0.551e-07</td>
</tr>
<tr>
<td>4</td>
<td>800</td>
<td>0.36</td>
<td>6.80</td>
<td>0.110e-07</td>
</tr>
<tr>
<td>5</td>
<td>1600</td>
<td>0.76</td>
<td>25.60</td>
<td>0.449e-07</td>
</tr>
<tr>
<td>6</td>
<td>3200</td>
<td>1.45</td>
<td>99.20</td>
<td>0.128e-07</td>
</tr>
<tr>
<td>7</td>
<td>6400</td>
<td>2.88</td>
<td>406.40</td>
<td>0.186e-07</td>
</tr>
<tr>
<td>8</td>
<td>12800</td>
<td>5.75</td>
<td>1638.40</td>
<td>0.957e-08</td>
</tr>
<tr>
<td>9</td>
<td>25600</td>
<td>11.45</td>
<td>6553.60</td>
<td>0.554e-08</td>
</tr>
<tr>
<td>10</td>
<td>51200</td>
<td>22.96</td>
<td>26060.80</td>
<td>0.114e-07</td>
</tr>
<tr>
<td>11</td>
<td>102400</td>
<td>45.94</td>
<td>104499.20</td>
<td>0.836e-08</td>
</tr>
</tbody>
</table>

Table 1: Times and errors for $0.1 \leq \delta_j \leq 1.0$, with targets and sources distributed uniformly in the unit box.

In Table 2, we used points randomly chosen on a circle with radius 0.29 and center (0.6,0.7), with the same parameters as in Table 1. The non-uniformity of the sources and targets did not slow down the algorithm.

Table 3 presents times and errors for the same calculation as in Example 2, but with the minimum scale reduced to $\delta_m = 0.01$. This slows down the algorithm only slightly; the speedup goes down to 1776 with 102,400 points.
<table>
<thead>
<tr>
<th>Case</th>
<th>$N = M$</th>
<th>$T_f$</th>
<th>$T_d$</th>
<th>$E_p/F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>0.06</td>
<td>0.10</td>
<td>0.115e-06</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>0.10</td>
<td>0.40</td>
<td>0.553e-06</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>0.16</td>
<td>1.60</td>
<td>0.439e-06</td>
</tr>
<tr>
<td>4</td>
<td>800</td>
<td>0.37</td>
<td>6.40</td>
<td>0.403e-06</td>
</tr>
<tr>
<td>5</td>
<td>1600</td>
<td>0.71</td>
<td>24.00</td>
<td>0.352e-06</td>
</tr>
<tr>
<td>6</td>
<td>3200</td>
<td>1.43</td>
<td>102.40</td>
<td>0.335e-06</td>
</tr>
<tr>
<td>7</td>
<td>6400</td>
<td>2.83</td>
<td>409.60</td>
<td>0.181e-06</td>
</tr>
<tr>
<td>8</td>
<td>12800</td>
<td>5.69</td>
<td>1632.00</td>
<td>0.255e-06</td>
</tr>
<tr>
<td>9</td>
<td>25600</td>
<td>11.38</td>
<td>6528.00</td>
<td>0.226e-06</td>
</tr>
<tr>
<td>10</td>
<td>51200</td>
<td>22.62</td>
<td>26214.40</td>
<td>0.155e-06</td>
</tr>
<tr>
<td>11</td>
<td>102400</td>
<td>45.43</td>
<td>104192.00</td>
<td>0.254e-06</td>
</tr>
</tbody>
</table>

Table 2: Times and errors for $0.1 \leq \delta_j \leq 1.0$, with targets and sources distributed uniformly on a circle.

<table>
<thead>
<tr>
<th>Case</th>
<th>$N = M$</th>
<th>$T_f$</th>
<th>$T_d$</th>
<th>$E_p/F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>0.06</td>
<td>0.10</td>
<td>0.145e-06</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>0.12</td>
<td>0.40</td>
<td>0.564e-06</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>0.24</td>
<td>1.60</td>
<td>0.420e-06</td>
</tr>
<tr>
<td>4</td>
<td>800</td>
<td>0.45</td>
<td>6.40</td>
<td>0.252e-06</td>
</tr>
<tr>
<td>5</td>
<td>1600</td>
<td>0.90</td>
<td>25.60</td>
<td>0.311e-06</td>
</tr>
<tr>
<td>6</td>
<td>3200</td>
<td>1.81</td>
<td>102.40</td>
<td>0.275e-06</td>
</tr>
<tr>
<td>7</td>
<td>6400</td>
<td>3.60</td>
<td>400.00</td>
<td>0.170e-06</td>
</tr>
<tr>
<td>8</td>
<td>12800</td>
<td>7.28</td>
<td>1657.60</td>
<td>0.238e-06</td>
</tr>
<tr>
<td>9</td>
<td>25600</td>
<td>14.54</td>
<td>6502.40</td>
<td>0.201e-06</td>
</tr>
<tr>
<td>10</td>
<td>51200</td>
<td>29.03</td>
<td>26291.20</td>
<td>0.144e-06</td>
</tr>
<tr>
<td>11</td>
<td>102400</td>
<td>58.86</td>
<td>104499.20</td>
<td>0.236e-06</td>
</tr>
</tbody>
</table>

Table 3: Times and errors for $0.01 \leq \delta_j \leq 1.0$, with targets and sources distributed uniformly on a circle.
Table 4 is similar to Table 3, but with $\delta_j$'s lying between 0.002 and 0.2. Even with these fairly small scales, we still achieve a respectable speedup of 500 with $N = 102,400$.

<table>
<thead>
<tr>
<th>Case</th>
<th>$N = M$</th>
<th>$T_f$</th>
<th>$T_d$</th>
<th>$E_p/F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>0.14</td>
<td>0.10</td>
<td>0.112e-07</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>0.36</td>
<td>0.30</td>
<td>0.180e-06</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>0.73</td>
<td>1.40</td>
<td>0.123e-06</td>
</tr>
<tr>
<td>4</td>
<td>800</td>
<td>1.55</td>
<td>6.00</td>
<td>0.110e-06</td>
</tr>
<tr>
<td>5</td>
<td>1600</td>
<td>3.18</td>
<td>25.60</td>
<td>0.623e-07</td>
</tr>
<tr>
<td>6</td>
<td>3200</td>
<td>6.59</td>
<td>102.40</td>
<td>0.683e-07</td>
</tr>
<tr>
<td>7</td>
<td>6400</td>
<td>13.22</td>
<td>409.60</td>
<td>0.460e-07</td>
</tr>
<tr>
<td>8</td>
<td>12800</td>
<td>26.88</td>
<td>1644.80</td>
<td>0.936e-07</td>
</tr>
<tr>
<td>9</td>
<td>25600</td>
<td>54.15</td>
<td>6720.00</td>
<td>0.634e-07</td>
</tr>
<tr>
<td>10</td>
<td>51200</td>
<td>107.85</td>
<td>26470.40</td>
<td>0.266e-07</td>
</tr>
<tr>
<td>11</td>
<td>102400</td>
<td>216.64</td>
<td>108083.20</td>
<td>0.720e-07</td>
</tr>
</tbody>
</table>

Table 4: Times and errors for $0.002 \leq \delta_j \leq 0.2$, with targets and sources distributed uniformly on a circle.

6 Conclusions

We have presented fast algorithms which evaluate $d$-dimensional Gaussian convolution sums (1) and (2) with scales dependent either on the source or the target. Such sums are frequently evaluated in statistical computations[3, 10] and also occur in numerical solution of the heat equation [9, 8] and other applications[7]. Our algorithms evaluate these sums in $O(M + N)$ work, with a constant factor depending on the precision required and the minimum scale which occurs.

Numerical results indicate that our method can speed up large calculations by several orders of magnitude; in one example, the computation time was reduced to one minute instead of the 1.2 days required by direct evaluation. Thus this algorithm can be extremely useful in reducing the time required to carry out large-scale computations. We are currently applying a
variant of this algorithm to problems in crystal growth, where it is used to evaluate free-space heat potentials.
References


