Institut für Numerische Simulation
Rheinische Friedrich-Wilhelms-Universität Bonn

Wegelerstraße 6 • 53115 Bonn • Germany
phone +49 228 73-3427 • fax +49 228 73-7527
www.ins.uni-bonn.de

M. Griebel, D. Wissel

Fast Approximation of the Discrete Gauss Transform in Higher Dimensions

INS Preprint No. 1111

October 2011

Fast Approximation of the Discrete Gauss Transform in Higher Dimensions

Michael Griebel · **Daniel Wissel**

October 2011

Abstract We present a novel approach for the fast approximation of the discrete Gauss transform in higher dimensions. The algorithm is based on the dual-tree technique and introduces a new Taylor series expansion. It compares favorably to existing methods especially when it comes to higher dimensions and a broad range of bandwidths. Numerical results with different datasets in up to 62 dimensions demonstrate its performance.

Keywords Gauss transform · fast approximation algorithms · high-dimensional

Mathematics Subject Classification (2000) 65R10 · 65D15 · 35K05 · 41A58

1 Introduction

The Gauss transform is an important numerical tool with many applications in, e.g., image manipulation [1,4] option pricing [5,6] and data mining including classification, regression and density estimation [7,17,22].

The d-dimensional Gaussian kernel $k_h : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is defined as

$$k_h(t,s) = e^{-\|t-s\|^2/h^2}, \tag{1}$$

where the parameter $h \in \mathbb{R}$ denotes the bandwidth and $\|\cdot\|$ is the Euclidean distance. The Gauss integral transform is then given by

$$\mathscr{G}f(t) = \int_\Omega f(s) \cdot k_h(t,s) ds, \tag{2}$$

Michael Griebel
Institute for Numerical Simulation, University of Bonn, Wegelerstr. 6, D–53115 Bonn
Tel.: +49-228-733437
E-mail: griebel@ins.uni-bonn.de

Daniel Wissel
Institute for Numerical Simulation, University of Bonn, Wegelerstr. 6, D–53115 Bonn
Tel.: +49-228-733486
E-mail: wissel@ins.uni-bonn.de

where the function f is given on $\Omega \subseteq \mathbb{R}^d$.

The discrete Gauss transform (DGT) is now defined as

$$G(t_i) = \sum_{j=1}^{N} f_j \cdot k_h(t_i, s_j) \qquad i = 1, \ldots, M, \tag{3}$$

with target points $t_i \in \mathbb{R}^d$, $i = 1, \ldots, M$, source points $s_j \in \mathbb{R}^d$ and weight coefficients $f_j \in \mathbb{R}$, $j = 1, \ldots, N$. It is obtained from (2) via the sampling approximation $f(s) \simeq \sum_{j=1}^{N} f_j \delta(s - s_j)$ and subsequent evaluation at the discrete target points t_i, $i = 1, \ldots, M$. The discrete Gauss transform corresponds to

$$\mathbf{g} = K \cdot \mathbf{f} \tag{4}$$

with the M-vector $\mathbf{g}$, $g_i = G(t_i)$, the $(M \times N)$-matrix K, $K_{ij} = k_h(t_i, s_j)$ and the N-vector $\mathbf{f}$, $f_j = f(s_j)$. This matrix-vector multiplication involves $\mathcal{O}(N \cdot M)$ operations.

In 1991, GREENGARD and STRAIN [10] developed the "Fast Gauss Transform" (FGT), a method for the rapid approximation of the discrete Gauss transform. The algorithm is a variant of the fast multipole method introduced in [9]. Here, truncated Hermite and Taylor expansions are used to break the entanglement of sources and targets in (3). This way, the runtime complexity is reduced from $\mathcal{O}(N \cdot M)$ to $\mathcal{O}(N + M)$. A uniform box structure allows to combine the contributions of nearby points while employing moderate truncation bounds. The FGT (with an error estimate as corrected in [2]) offers good performance in the case of dimensions up to three for uniformly distributed source and target points and moderate values of the bandwidth h. For higher-dimensional problems it however shows severe limitations since the dimension enters exponentially into the order constant.

A novel approach, the "Improved Fast Gauss Transform" (IFGT), was presented by YANG, DURAISWAMI and GUMEROV [18, 23] in 2003. Its essential new ingredients are a different Taylor expansion — with a tighter error bound which is also feasible for slightly higher dimensions — and an adaptive clustering scheme. Hence, the IFGT provides a noticeable speed-up over the FGT in up to 10 dimensions for moderate and high values of the bandwidth, but its performance is mediocre for problems involving lower bandwidths.

A third algorithm for approximating the discrete Gauss transform, the "Dual-Tree Fast Gauss Transform" (DFGT), was proposed by LEE, GRAY and MOORE [14, 15] in 2005. Here, two tree structures define a multi-level clustering of source and target points, respectively. The trees are traversed simultaneously to calculate interactions between node pairs, descending on finer levels to reduce the error. Furthermore, expansions of the FGT with improved truncation rules are used which allow for an efficient evaluation also for low values of h and for datasets in up to 16 dimensions. However, compared to direct computation the existing DFGT implementations suffer from inferior performance for certain bandwidths and higher-dimensional data.

Altogether, there still is a lack of reliable and efficient algorithms when it comes to problems with dimension $d > 3$ and general bandwidth h.

In this article we present a new approach for the fast approximation of the Gauss transform based on the dual-tree method. Our "Optimized Fast Gauss Transform" (OFGT) combines *kd*-trees with a refined error control mechanism and a modified

version of the IFGT series expansion. While the FGT and IFGT come with a time complexity of $\mathcal{O}(f(d)\cdot(N+M))$ that rapidly gets worse than the direct computation complexity $\mathcal{O}(d\cdot N\cdot M)$ for growing d and fixed N and M, our algorithm locally selects the best approximation method in terms of costs ensuring a specified accuracy ε. Computational results show excellent performance over the whole range of tested bandwidth values h for real-world datasets of up to 10 dimensions and still show competitive runtimes (with respect to direct computation) for up to 62 dimensions.

The remainder of this paper is organized as follows. In §2, we present our new algorithm, including a detailed description of the tree structure, the error control mechanism and the new series expansion. Section 3 provides a short overview of the other approximation algorithms and highlights similarities and differences. An in-depth discussion of the corresponding numerical results using both synthetic and real-world data is given in §4. Final remarks in §5 conclude this paper.

2 The Optimized Fast Gauss Transform

First, we present the basic algorithmic structure in pseudocode. Details will be discussed in the following subsections. Table 1 introduces the main notations. In a first step of the main program **OFGT** (compare Figure 1), a source and a target tree are constructed from the source and target points, respectively, providing multi-level clustering structures for the point sets. Here, the binary trees are perfectly balanced, i.e. each inner node S has exactly two child nodes, $S.l$ and $S.r$. After a proper initialization of the error control variables, the recursive procedure **CONTRIB** is invoked with the two tree roots containing all source and target points, respectively. For a given node pair (S,T), this procedure approximates the associated source node contribution $G_S(t_i)=\sum_{s_j\in S} f_j\cdot k_h(t_i,s_j)$ for all targets $t_i\in T$ by a specially chosen local approximation method, or via recursion, deferring the task to the next finer node level. Here, either a mean value approximation, a Taylor series expansion, or direct computation is selected according to an error and cost estimation process which ensures a locally optimal choice. Direct evaluation is only applied on leaf level, when obviously the other approximation methods on coarser tree levels failed to meet the required local cost and error criteria. Note that, for a prescribed precision $\varepsilon>0$, our algorithm computes an approximation $\widetilde{G}(t_i)$ with the relative error bound $\left|\widetilde{G}(t_i)-G(t_i)\right|\le\varepsilon\cdot G(t_i)$ for all t_i.

Table 1 Symbols for the algorithmic descriptions.

N	number of source points	M	number of target points
s_j	source point $\in\mathbb{R}^d$	t_i	target point $\in\mathbb{R}^d$
$f_j\ge 0$	weight coefficient	$h>0$	bandwidth of Gaussian
$S_{(k)}$	source node	$T_{(k)}$	target node
$s^*_{(k)}$	expansion center of $S_{(k)}$	$t^*_{(k)}$	expansion center of $T_{(k)}$
F_S	weight sum of source node S	α	d-dimensional multi-index
$R(S,T)$	local approx. error bound	p	order of series expansion

OFGT$(d, s_{j=1,\ldots,N}, t_{i=1,\ldots,M}, f_{j=1,\ldots,N}, h, \varepsilon)$
Input: dim. d, sources s_j, targets t_i, weights $f_j \geq 0$, bandwidth $h > 0$, prec. $\varepsilon > 0$
Purpose: for all $t_{i=1,\ldots,M}$ calculate $\widetilde{G}(t_i)$ with $\left|\widetilde{G}(t_i) - G(t_i)\right| \leq \varepsilon \cdot G(t_i)$

ConstructSourceTree$(s_{j=1,\ldots,N}, f_{j=1,\ldots,N})$
ConstructTargetTree$(t_{i=1,\ldots,M})$
InitErrorControl$()$
CONTRIB$(SourceTree.Root, TargetTree.Root)$

CONTRIB(S,T)
Input: source node S, target node T
Purpose: for all $t_i \in T$ update $\widetilde{G}(t_i) \leftarrow \widetilde{G}(t_i) + \widetilde{G}_S(t_i)$, where $\widetilde{G}_S(t_i)$ is computed either by one of three approximation methods, or via recursion

if ErrorMean$(S,T) \leq$ errorbound(S,T) **then**
  MEAN(S,T)
else
  if isLeaf(S) **or** isLeaf(T) **then**
    DIRECT(S,T)
  else
    if $\text{cost}_{\text{Direct}}(S,T) \leq \text{cost}_{\text{Taylor}}(S,T)$ **or** $\text{cost}_{\text{Children}}(S,T) \leq \text{cost}_{\text{Taylor}}(S,T)$ **then**
      CONTRIB$(S.l, T.l)$; **CONTRIB**$(S.l, T.r)$
      CONTRIB$(S.r, T.l)$; **CONTRIB**$(S.r, T.r)$
    else
      TAYLOR(S,T)

Fig. 1 Basic structure of the algorithm.

Let us remark that our algorithm is designed to work with arbitrary source and target points, but non-negative weight coefficients $f_j \geq 0$. The reason for this constraint will be given later in the discussion of the error control mechanism.

2.1 Tree structure

Now we present the construction of the source tree and target tree for the s_j $(j = 1,\ldots,N)$ and t_i $(i = 1,\ldots,M)$. Figure 2 shows the algorithmic structure of the source tree construction, the target tree is built analogously. In general, the nodes of our binary trees correspond to subsets of source points. The tree root corresponds to the set of all points. The two child nodes $S.l$ and $S.r$ of an inner node S are constructed by splitting the inner node's point set into two disjoint subsets of (almost) equal size. To this end, given all points of the current node S, the coordinate direction with maximum variance is determined. Then, the median of the components of this coordinate direction is computed and used to split the points into two subsets, thus defining the two child nodes. Nodes are left as leaves if they do not contain more than q points. Here, q is a parameter of our method which we set as $q = 30$ throughout this paper. This way, a perfectly balanced binary tree is built.

The entire construction process works with a single copy of all points; whenever a node is split, its points are rearranged in the point array, and the respective array bounds determine whether a point belongs to the left or the right child.

ConstructSourceTree$(s_{j=1,\ldots,N}, f_{j=1,\ldots,N})$

$Nodes[1] \leftarrow Root \leftarrow \{s_1, \ldots, s_N\}$
$maxLeafSize \leftarrow 30$
$nLevels \leftarrow \lceil \log_2 (N/maxLeafSize) \rceil$
$nNodesPerLevel \leftarrow 1$
for $l = 0, \ldots, nLevels - 1$ **do**
  for $n = 0, \ldots, nNodesPerLevel - 1$ **do**
    DivideNode$(nNodesPerLevel + n)$
  $nNodesPerLevel \leftarrow 2 \cdot nNodesPerLevel$
for all $S \in Nodes$ **do**
  calcBoundingRectangle(S)
  calcBoundingSphere(S)
  $F_S \leftarrow \sum_{s_j \in S} f_j$

DivideNode(n)

$S \leftarrow Nodes[n]$
for $d = 1, \ldots, dim$ **do**
  $variance[d] \leftarrow \mathrm{Var}\{s_j[d] \mid s_j \in S\}$
$d_{cut} \leftarrow \mathrm{argmax}_{d=1,\ldots,dim}\{variance[d]\}$
$med \leftarrow \mathrm{median}\{s_j[d_{cut}] \mid s_j \in S\}$
$Nodes[2n] \leftarrow S.l \leftarrow \{s_j \in S \mid s_j[d_{cut}] \leq med\}$
$Nodes[2n+1] \leftarrow S.r \leftarrow \{s_j \in S \mid s_j[d_{cut}] > med\}$

Fig. 2 Construction of the source tree.

Furthermore, for each tree node, a hyperrectangle and a sphere are stored which contain all points associated to the respective tree node. First, the minimal bounding hyperrectangle is calculated; the center point of this hyperrectangle is then used to calculate the minimal bounding sphere. Also, this point is used later as center for a Taylor series expansion of the corresponding node (see subsection 2.6). Additionally, the weight sums $F_S = \sum_{s_j \in S} f_j$ are precalculated and stored for each node S of the source tree.

2.2 Error control

We now present the error control mechanism of our algorithm in more detail. It guarantees the global relative error bound

$$\left|\widetilde{G}(t_i) - G(t_i)\right| \leq \varepsilon \cdot G(t_i) \tag{5}$$

by limiting the local errors $\left|\widetilde{G}_S(t_i) - G_S(t_i)\right|$ that originate from the local approximation methods. To this end, note that the recursive procedure **CONTRIB** adaptively selects for a given target $t_i \in T$ some set $\mathbb{S}$ of source nodes which represents a disjoint union of all source points. It then calculates the approximated Gaussian as

$$\widetilde{G}(t_i) = \sum_{S \in \mathbb{S}} \widetilde{G}_S(t_i), \tag{6}$$

where each of the approximated source node contributions $\widetilde{G}_S(t_i)$ is determined by one of the three different approximation methods. Now suppose that the particular method for a certain source node S comes with a local error bound $R(S,T)$, i.e.

$$\left|\widetilde{G}_S(t_i) - G_S(t_i)\right| \leq R(S,T) \tag{7}$$

for all $t_i \in T$. We require this error bound to fulfill the condition

$$R(S,T) \leq \varepsilon \cdot G_T^{\min} \cdot F_S/F, \tag{8}$$

where $G_T^{\min} = \min_{t_i \in T} G(t_i)$, $F = \sum_{j=1}^{N} f_j$ and $F_S = \sum_{s_j \in S} f_j$. Then, the relative global error can be bounded from above via

$$\left|\widetilde{G}(t_i) - G(t_i)\right| \leq \sum_{S \in \mathbb{S}} \left|\widetilde{G}_S(t_i) - G_S(t_i)\right| \leq \varepsilon \cdot G_T^{\min} \cdot \sum_{S \in \mathbb{S}} F_S/F \leq \varepsilon \cdot G(t_i) \tag{9}$$

for all $t_i \in T$.

For a viable algorithm, condition (8) needs to be modified in two ways. First note that the $G_T^{\min}$ depend on the exact solution and are therefore unknown. Since all weights f_j are non-negative, the solution vector is monotonically increasing during its successive computation in the run of the algorithm, i.e. while summing up the contributions. Its entries are thus always less than the exact solution. Hence, $G_T^{\min}$ in condition (8) can be replaced by $\widetilde{G}_T^{\min} = \min_{t_i \in T} \widetilde{G}(t_i)$, where $\widetilde{G}$ shall here denote the current solution vector for any actual state of our algorithm. Now, at the beginning we have $\widetilde{G} = 0$, but we need good initial values for the $\widetilde{G}_T^{\min}$. To this end (compare Figure 3), in a first step, the source leaf node S closest to T is determined for each leaf node T of the target tree. Then, for each such pair (S,T), the source node contribution $G_S(t_i)$ is computed directly and these results are used to initialize $\widetilde{G}_T^{\min}$ for all leaf nodes T. Finally, the values are propagated to all inner nodes via $\widetilde{G}_T^{\min} = \min\left\{\widetilde{G}_{T.l}^{\min}, \widetilde{G}_{T.r}^{\min}\right\}$ in a bottom-up traversal of the target tree.

InitErrorControl()

for all target leaf nodes T **do**
  find source leaf node S closest to T
  for all $t_i \in T$ **do**
    $G_S(t_i) \leftarrow \sum_{s_j \in S} f_j \cdot k_h(t_i, s_j)$
  $\widetilde{G}_T^{\min} \leftarrow \min_{t_i \in T} G_S(t_i)$
for all target inner nodes T **do**
  bottom-up propagation of $\widetilde{G}_T^{\min}$ via $\widetilde{G}_T^{\min} \leftarrow \min\left\{\widetilde{G}_{T.l}^{\min}, \widetilde{G}_{T.r}^{\min}\right\}$

Fig. 3 Initialization of the error control variables.

The second modification to condition (8) aims at allowing larger local errors — and thus faster approximations — while still meeting the specified global error bound. Note that condition (8) assigns the fraction F_S/F of the total error to the approximated contribution of source node S. However, if the actual error $R(S,T)$ of the respective

approximation method is smaller than the aimed maximal error $\varepsilon \cdot \widetilde{G}_T^{\min} \cdot F_S/F$, we can allow larger errors in the contributions of other source nodes to the current target node T without violating the global error bound. To this end, the unused error fractions for a target node T are accumulated in the parameter F_T^{save}, and condition (8) is refined to

$$R(S,T) \leq \text{errorbound}(S,T) \stackrel{\text{def}}{=} \varepsilon \cdot \widetilde{G}_T^{\min} \cdot \left(F_S + F_T^{save}\right)/F. \tag{10}$$

At the beginning, F_T^{save} is initialized with zero. In the recursive procedure, whenever some approximation method for (S,T) is applied, condition (10) is fulfilled and F_T^{save} is modified as

$$F_T^{save} \leftarrow F_T^{save} + \left\{ F_S - R(S,T) \cdot F / \left(\varepsilon \cdot \widetilde{G}_T^{\min} \right) \right\}. \tag{11}$$

The effect of this modification is as follows: If condition (10) holds also without the term F_T^{save}, the increment in (11) is non-negative and F_T^{save} accumulates the corresponding unused fraction of the global error; in the other case, the increment is negative and a portion of F_T^{save} is now used to allow a larger local error which results in a cheaper local approximation while still meeting the specified global error bound.

2.3 Recursive main routine

For a particular pair (S,T) of a source node S and target node T, the recursive main procedure **CONTRIB** (see Figure 1) approximates the source node contribution $G_S(t_i) = \sum_{s_j \in S} f_j \cdot k_h(t_i, s_j)$ for all targets $t_i \in T$ either by a specially chosen local approximation method or by deferring the task to the next finer node level via four recursive function calls. For this purpose, first the local error of the mean value method is estimated (see subsection 2.4). The method is applied, if the estimated error undercuts the local error bound (defined above in (10)). Failing this, two basic cases are distinguished. If the recursive routine has arrived at a leaf node, i.e. if at least one of the nodes S or T is a leaf, direct computation is performed, since obviously the fast approximation methods on coarser tree levels have failed to meet the required cost and error criteria. In the second case, that is for two non-leaf nodes S and T, the local costs of direct computation and Taylor series approximation, as well as the costs for approximation on the next finer node level are estimated (details are given in subsection 2.7). If either direct computation or approximation on the finer level is estimated to be cheaper than the Taylor approximation, we let the recursion continue, hoping to find pairs of smaller tree nodes with more favorable approximation costs on finer levels. In the other case, the Taylor series expansion is carried out.

Next, we discuss the details of the three different approximation methods, before we finally present the cost functions and cost complexity estimations.

2.4 Mean value approximation

The mean value method calculates a single average value for all $t_i \in T$ given by

$$\widetilde{G}_S(t_i) = F_S \cdot \frac{1}{2} \left(e^{-\left(\delta_{ST}^{\min}/h\right)^2} + e^{-\left(\delta_{ST}^{\max}/h\right)^2} \right), \tag{12}$$

where $\delta_{ST}^{\min}$ and $\delta_{ST}^{\max}$ are lower and upper bounds on the distances between all sources in S and all targets in T, i.e.

$$\delta_{ST}^{\min} \leq d_{ST}^{\min} \stackrel{\text{def}}{=} \min\left\{\|s_j - t_i\| \mid s_j \in S, t_i \in T\right\} \tag{13}$$

and

$$\delta_{ST}^{\max} \geq d_{ST}^{\max} \stackrel{\text{def}}{=} \max\left\{\|s_j - t_i\| \mid s_j \in S, t_i \in T\right\}. \tag{14}$$

The parameters $\delta_{ST}^{\min}$ and $\delta_{ST}^{\max}$ are determined for a given pair (S,T) by use of the respective bounding hyperrectangles and bounding spheres associated with these nodes. To this end, the maximal as well as the minimal distance between the two bounding rectangles of node S and T are computed via basic geometric considerations. These calculations involve an $\mathcal{O}(d)$ time complexity only. Likewise, the minimal and maximal distance between the two spheres are calculated. Altogether, we thus have two lower and two upper distance bounds, the better of which is then assigned to $\delta_{ST}^{\min}$ and $\delta_{ST}^{\max}$ in each case.

The corresponding local approximation error for the mean value method is obviously given by

$$R(S,T) = F_S \cdot \frac{1}{2}\left(e^{-\left(\delta_{ST}^{\min}/h\right)^2} - e^{-\left(\delta_{ST}^{\max}/h\right)^2}\right). \tag{15}$$

Note here that for each of the three approximation methods, the error control variables $\widetilde{G}_T^{\min}$ and F_T^{save} are updated after the current contribution $\widetilde{G}_S$ has been added to the solution vector $\widetilde{G}$ (compare Figure 4).

Altogether, the cost for the mean value approximation is of the order $\mathcal{O}(d+M_T)$, where M_T is the number of points in the target node T.

ErrorMean(S,T)

$\delta_{ST}^{\min} \leftarrow$ lower bound on $d_{ST}^{\min} = \min\left\{\|s_j - t_i\| \mid s_j \in S, t_i \in T\right\}$
$\delta_{ST}^{\max} \leftarrow$ upper bound on $d_{ST}^{\max} = \max\left\{\|s_j - t_i\| \mid s_j \in S, t_i \in T\right\}$
return $\left\{F_S \cdot \frac{1}{2}\left(e^{-\left(\delta_{ST}^{\min}/h\right)^2} - e^{-\left(\delta_{ST}^{\max}/h\right)^2}\right)\right\}$

MEAN(S,T)

$\widetilde{G}_S = F_S \cdot \frac{1}{2}\left(e^{-\left(\delta_{ST}^{\min}/h\right)^2} + e^{-\left(\delta_{ST}^{\max}/h\right)^2}\right)$
for all $t_i \in T$ **do**
  $\widetilde{G}(t_i) \leftarrow \widetilde{G}(t_i) + \widetilde{G}_S$
$\widetilde{G}_T^{\min} \leftarrow \min_{t_i \in T} \widetilde{G}(t_i)$
$F_T^{save} \leftarrow F_T^{save} + F_S - \textbf{ErrorMean}(S,T) \cdot F / \left(\varepsilon \cdot \widetilde{G}_T^{\min}\right)$

Fig. 4 Mean value approximation error and method.

2.5 Direct computation

If the other two approximation methods are not appropriate due to our error and cost estimation, the source node contribution is directly computed via

$$G_S(t_i) = \sum_{s_j \in S} f_j \cdot e^{-\|t_i - s_j\|^2/h^2} \tag{16}$$

for all $t_i \in T$ which results in a local error of $R(S,T) = 0$. The corresponding procedure is given in Figure 5. As explained above, direct computation is only applied in case that at least one of the nodes S and T is a leaf node of the respective tree. Given a source node S with N_S points and a target node T with M_T points, the corresponding cost complexity is $\mathcal{O}(d \cdot N_S \cdot M_T)$.

DIRECT(S,T)

for all $t_i \in T$ **do**
  for all $s_j \in S$ **do**
    $\widetilde{G}(t_i) \leftarrow \widetilde{G}(t_i) + f_j \cdot k_h(t_i, s_j)$
$\widetilde{G}_T^{\min} \leftarrow \min_{t_i \in T} \widetilde{G}(t_i)$
$F_T^{save} \leftarrow F_T^{save} + F_S$

Fig. 5 Direct computation.

2.6 Taylor series expansion

The key method for the fast approximation is based on a Taylor series expansion. Since the employed technique is similar to the multipole expansion [3], we stick to the notion of moments and we use them to break the entanglement of source and target points in the Gaussian kernel. For a source node S (with expansion center s^*) and a target node T (with expansion center t^*), these moments are defined as

$$M_\alpha(S,T) = \frac{2^{|\alpha|}}{\alpha!} \sum_{s_j \in S} f_j \cdot k_h(s_j, s^*) \cdot e^{2(s_j - s^*)(t^* - s^*)/h^2} \left(\frac{s_j - s^*}{h} \right)^\alpha. \tag{17}$$

The associated approximated local contributions $\widetilde{G}_S(t_i)$ are then given by

$$\widetilde{G}_S(t_i) = \sum_{|\alpha|_1 < p} M_\alpha(S,T) \cdot k_h(t_i, s^*) \left(\frac{t_i - t^*}{h} \right)^\alpha. \tag{18}$$

Here, $\alpha = (\alpha_1, \ldots, \alpha_d)$ is a multi-index and $|\cdot|_1$ denotes the l_1-norm. Due to the condition $|\alpha|_1 < p$, the expansion consists of $\binom{p-1+d}{d}$ terms. The truncation parameter

p is chosen locally for the particular node pair (S,T), i.e. $p = p(S,T)$. Then, a local error bound for the truncated expansion (18) is given by

$$\left\| G_S(t_i) - \widetilde{G}_S(t_i) \right\| \leq R(S,T) \stackrel{\text{def}}{=} F_S \cdot \frac{1}{p!} \left(\frac{2 r_S r_T}{h^2} \right)^p \cdot e^{(r_S + r_T)^2/h^2}, \tag{19}$$

where $r_S = \max_{s_j \in S} \|s_j - s^*\|$ and $r_T = \max_{t_i \in T} \|t_i - t^*\|$ are the radii of the nodes S and T, respectively.

The procedure for the Taylor series approximation is presented in Figure 6. For a source node S with N_S points and a target node T with M_T points, the complexity for calculating the moments is $\mathcal{O}(\binom{p-1+d}{d} \cdot N_S)$ and the complexity for the subsequent evaluation of the $\widetilde{G}_S(t_i)$ is $\mathcal{O}(\binom{p-1+d}{d} \cdot M_T)$.

TAYLOR(S,T)

$s^* \leftarrow$ center point of node S; $\quad r_S \leftarrow$ radius of node S
$t^* \leftarrow$ center point of node T; $\quad r_T \leftarrow$ radius of node T
choose smallest p such that $F_S \cdot \frac{1}{p!} \left(\frac{2 r_S r_T}{h^2} \right)^p \cdot e^{(r_S + r_T)^2/h^2} \leq \varepsilon \cdot \widetilde{G}_T^{\min} \cdot (F_S + F_T^{save}) / F$
for all $\alpha = (\alpha_1, \ldots, \alpha_d)$ with $|\alpha|_1 < p$ **do**
  $M_\alpha(S,T) \leftarrow \frac{2^{|\alpha|}}{\alpha!} \sum_{s_j \in S} f_j \cdot k_h(s_j, s^*) \cdot e^{2(s_j - s^*)(t^* - s^*)/h^2} \left(\frac{s_j - s^*}{h} \right)^\alpha$
for all $t_i \in T$ **do**
  $\widetilde{G}(t_i) \leftarrow \widetilde{G}(t_i) + \sum_{|\alpha|_1 < p} M_\alpha(S,T) \cdot k_h(t_i, s^*) \left(\frac{t_i - t^*}{h} \right)^\alpha$
$\widetilde{G}_T^{\min} \leftarrow \min_{t_i \in T} \widetilde{G}(t_i)$

Fig. 6 Taylor series approximation method.

We now give a short derivation of the presented series expansion, the moment representation, and the local error bound (19). First, the Gaussian kernel $k_h(t_i, s_j) = e^{-\|t_i - s_j\|^2/h^2}$ can be rewritten for any source node S with center s^* and target node T with center t^* by inserting the term $(s^* - s^*)$ and then the term $(t^* - t^*)$ as

$$\begin{aligned} k_h(t_i, s_j) &= k_h(t_i, s^*) \cdot k_h(s_j, s^*) \cdot e^{2(s_j - s^*)(t_i - s^*)/h^2} \\ &= k_h(t_i, s^*) \cdot k_h(s_j, s^*) \cdot e^{2(s_j - s^*)(t^* - s^*)/h^2} \cdot e^{2(s_j - s^*)(t_i - t^*)/h^2}. \end{aligned} \tag{20}$$

Next, the last factor is expanded into a Taylor series yielding

$$e^{2(s_j - s^*)(t_i - t^*)/h^2} = \sum_{\alpha \geq 0} \frac{2^{|\alpha|}}{\alpha!} \left(\frac{s_j - s^*}{h} \right)^\alpha \left(\frac{t_i - t^*}{h} \right)^\alpha. \tag{21}$$

By interchanging the sums and using equations (20) and (21), the contribution $G_S(t_i)$ of the source node S can now be rewritten as

$$G_S(t_i) = \sum_{s_j \in S} f_j \cdot k_h(t_i, s_j) = \sum_{\alpha \geq 0} M_\alpha(S,T) \cdot k_h(t_i, s^*) \left(\frac{t_i - t^*}{h} \right)^\alpha,$$

with the moments $M_\alpha(S,T)$ as defined in (17). Truncating this series expansion with $|\alpha|_1 < p$ results in the approximation given by (18). To derive the associated error bound (19), note that the Lagrange remainder of the Taylor series (21) truncated by $|\alpha|_1 < p$ is given by

$$R_p = \frac{2^p}{p!}\left[\left(\frac{s_j - s^*}{h}\right)\cdot\left(\frac{t_i - t^*}{h}\right)\right]^p e^{2\theta(s_j-s^*)(t_i-t^*)/h^2}$$

with some $\theta \in (0,1)$. Using the Cauchy-Schwarz inequality this remainder can be bounded via

$$\begin{aligned} R_p &\le \frac{2^p}{p!}\left(\frac{\|s_j - s^*\|}{h}\right)^p \left(\frac{\|t_i - t^*\|}{h}\right)^p e^{2\theta\|s_j-s^*\|\|t_i-t^*\|/h^2} \\ &\le \frac{1}{p!}\left(\frac{2r_S r_T}{h^2}\right)^p e^{2\|s_j-s^*\|\|t_i-t^*\|/h^2}, \end{aligned}$$

since $\|s_j - s^*\| \le r_S$ and $\|t_i - t^*\| \le r_T$. The local error can now be estimated by

$$\begin{aligned} \left|\widetilde{G}_S(t_i) - G_S(t_i)\right| \le \sum_{s_j\in S} f_j \cdot \underbrace{k_h(t_i,s^*)}_{(1)} \cdot \underbrace{k_h(s_j,s^*)}_{(2)} \cdot \underbrace{e^{2(s_j-s^*)(t^*-s^*)/h^2}}_{(3)} \\ \cdot \frac{1}{p!}\left(\frac{2r_S r_T}{h^2}\right)^p \underbrace{e^{2\|s_j-s^*\|\|t_i-t^*\|/h^2}}_{(4)}. \end{aligned}$$

Next, we derive a bound for the product $(1)\cdot(2)\cdot(3)\cdot(4)$. To this end, term (1) is rewritten as

$$\begin{aligned} k_h(t_i,s^*) &= e^{-\|t_i-s^*\|^2/h^2} = e^{-\|(t_i-t^*)+(t^*-s^*)\|^2/h^2} \\ &= k_h(t_i,t^*)\cdot k_h(t^*,s^*)\cdot e^{-2(t_i-t^*)(t^*-s^*)/h^2}. \end{aligned}$$

Thus we have $(1)\cdot(2)\cdot(4) =$

$$= \underbrace{e^{-2(t_i-t^*)(t^*-s^*)/h^2}}_{(5)} \cdot \underbrace{k_h(t^*,s^*)}_{(6)} \cdot \underbrace{k_h(t_i,t^*)\cdot k_h(s_j,s^*)\cdot e^{2\|s_j-s^*\|\|t_i-t^*\|/h^2}}_{\le 1}.$$

To bound the remaining product $(3)\cdot(5)\cdot(6)$, we combine terms (3) and (5), and also insert the factor $1 = k_h(s_j-s^*,t_i-t^*)\cdot e^{\|(s_j-s^*)-(t_i-t^*)\|^2/h^2}$. Then, we obtain $(3)\cdot(5)\cdot(6) =$

$$= \underbrace{e^{2((s_j-s^*)-(t_i-t^*))(t^*-s^*)/h^2}\cdot k_h(t^*,s^*)\cdot k_h(s_j-s^*,t_i-t^*)}_{\le 1} \cdot e^{\|(s_j-s^*)-(t_i-t^*)\|^2/h^2}.$$

Using the triangle inequality, we can bound the last factor via

$$e^{\|(s_j-s^*)-(t_i-t^*)\|^2/h^2} \le e^{\left(\|s_j-s^*\|+\|t_i-t^*\|\right)^2/h^2} \le e^{(r_S+r_T)^2/h^2},$$

which yields the final local error bound

$$\left|\widetilde{G}_S(t_i) - G_S(t_i)\right| \le F_S \cdot \frac{1}{p!}\left(\frac{2r_S r_T}{h^2}\right)^p \cdot e^{(r_S+r_T)^2/h^2}$$

as presented in (19).

2.7 Complexity

We now shortly discuss the cost complexity of the different steps involved in our algorithm. The two tree structures are built in a total time complexity of $\mathcal{O}(d \cdot N \cdot \log_2 N)$ and $\mathcal{O}(d \cdot M \cdot \log_2 M)$, respectively. For this purpose, note that the calculation of the variance and the median in the node splitting procedure as well as the construction of the bounding elements can be done in linear time with respect to the number of points and the dimension. Since the trees are perfectly balanced, we get the above time complexity. Furthermore, we achieve a minimal memory complexity of $\mathcal{O}(d \cdot N) + \mathcal{O}(d \cdot M)$, since only a single copy of all points is stored for each tree. The nodes just hold their respective array bounds, the center point, the data of the bounding elements, and the source weight sum.

The initialization of the error control variables $\widetilde{G}_T^{\min}$ (see Figure 3) induces costs of $\mathcal{O}(d \cdot M \cdot q)$, where $q = 30$ is the maximal number of points in each source leaf node.

Next, let us consider the costs for the different approximation methods for a given source node S with N_S points and target node T with M_T points. Regarding the mean value method (Figure 4), the above distance estimates $\delta_{ST}^{\min}$ and $\delta_{ST}^{\max}$ are determined in $\mathcal{O}(d)$ time using the precalculated node bounding elements. The solution variables $\widetilde{G}(t_i)$ for all $t_i \in T$ are then updated in $\mathcal{O}(M_T)$ time. Direct computation obviously results in a cost complexity of $\mathcal{O}(d \cdot N_S \cdot M_T)$.

Furthermore, the Taylor series approximation first requires to fix the smallest $p = p(S,T)$ that satisfies

$$R(S,T) = F_S \cdot \frac{1}{p!} \left(\frac{2 r_S r_T}{h^2} \right)^p \cdot e^{(r_S + r_T)^2 / h^2} \leq \varepsilon \cdot \widetilde{G}_T^{\min} \cdot \left(F_S + F_T^{save} \right) / F, \tag{22}$$

see (10) and (19). Given this p which crucially depends on the respective tree node radii r_S and r_T, the total time complexity for calculating the moments and summing up the contributions is $\mathcal{O}(\binom{p-1+d}{d} \cdot (N_S + M_T))$.

Note that the cost functions in the recursive procedure **CONTRIB** (Figure 1) represent adapted versions of these complexity estimations. In particular, the costs for the children of the pair (S,T) are determined by considering all four child node pairs on the next finer tree level. For each such pair, the minimum of the costs for direct computation and the costs for Taylor approximation is calculated. These minima then add up to the total costs on the finer tree level. The purpose of this entire estimation process is to find that combination of interacting nodes which is locally optimal in terms of cost complexity. Altogether, the preprocessing (including tree construction and error control initialization) requires $\mathcal{O}(d \cdot N \cdot \log_2 N) + \mathcal{O}(d \cdot M \cdot \log_2 M)$ costs. Note that it is independent of the bandwidth h and the accuracy ε. The recursive main part guarantees a local best-case runtime of $\min\{\mathcal{O}(d \cdot N_S \cdot M_T), \mathcal{O}(\binom{p-1+d}{d} \cdot (N_S + M_T))\}$. This results in a global worst-case estimation of $\min\{\mathcal{O}(d \cdot N \cdot M), \mathcal{O}(\binom{p-1+d}{d} \cdot (N + M))\}$ with $p = \max_{(S,T)} \{p(S,T)\}$. Naturally, the runtimes are much more favorable in practice. Moreover they also depend on the local structure of the considered data.

3 Feature comparison with other algorithms

In this section we give a quick overview of the main features of the other three Gauss transform approximation algorithms, i.e. the Fast Gauss Transform, the Improved Fast Gauss Transform, and the Dual-Tree Fast Gauss Transform.

The Fast Gauss Transform [10] employs a uniform subdivision of the unit hypercube into smaller hypercubes called "boxes" of sidelength $\sqrt{2}rh$ with some positive $r \leq \frac{1}{2}$. Source and target points are clustered in the same way. Contributions of far-field source boxes are omitted, near-field contributions are approximated via certain different series expansions based on multi-dimensional Hermite functions. The number of boxes is proportional to $\mathcal{O}(h^{-d})$, and the series expansions require $\mathcal{O}(p^d \cdot (N+M))$ operations. Hence, the FGT is not a good choice for dimensions $d > 3$ nor small bandwidths h.

The Improved Fast Gauss Transform [23] treats the target points individually without arranging them in clusters, while the source points are partitioned by a non-hierarchical scheme called farthest-point clustering. Again, contributions of far-field source clusters are omitted, while the near-field contributions are approximated via a Taylor series expansion similar to the one employed in our algorithm. The corresponding local error bound is given by

$$R(S) = F_S \cdot \frac{1}{p!} \left(\frac{2 r_S r_n}{h^2} \right)^p \tag{23}$$

with the cluster radius $r_S = \max_{s_j \in S} \|s_j - s^*\|$ and the so-called cluster cutoff radius $r_n \geq \|t_i - s^*\|$ for all targets t_i interacting with the source cluster S. In general, the local error bound (19) of our series expansion results more often in smaller truncation bounds p, since the radii r_S and r_T get smaller for tree nodes of deeper levels. The farthest-point clustering requires a reasonable choice of the maximum cluster radius or, alternatively, the number of clusters K. For smaller bandwidth h the clustering costs of $\mathcal{O}(d \cdot N \cdot K)$ increase, since a larger number of clusters is needed to guarantee the error bounds. In general, the IFGT does not offer good performance for small bandwidths.

The Dual-Tree Fast Gauss Transform [14] basically features the same tree structure and error control mechanism as described above in our OFGT algorithm. The series approximations are modified FGT variants based on the Hermite functions. The corresponding local error bound is given by

$$R(S) = F_S \cdot \frac{1}{\sqrt{p!}} \left(\frac{\sqrt{2} r_1}{h} \right)^p, \tag{24}$$

where $r_1 = \max_{s_j \in S} \|s_j - s^*\|_1$ (see [21] for details). In experiments with real-world data of up to 16 dimensions, the DFGT shows relatively good performance for a wide range of bandwidths (see [14]). However, in some cases for dimensions 7 and higher, it is considerably slower than direct computation.

Table 2 summarizes the main features of the three different algorithms. The Fast Gauss Transform with a series expansion truncated via $\|\alpha\|_\infty < p$ involves p^d series terms, while all other three methods with the truncation rule $\|\alpha\|_1 < p$ only require

Table 2 Features of the different algorithms.

	FGT	**Improved FGT**	**Dual-Tree FGT**	**our algorithm**
clustering	uniform grid	farthest-point cl.	dual-tree	dual-tree
series expans.	Hermite function	power function	Hermite function	power function
# series terms	p^d	$\binom{p-1+d}{d}$	$\binom{p-1+d}{d}$	$\binom{p-1+d}{d}$
series error	$\frac{1}{(1-r)^d} \cdot \sum_{k=0}^{d-1} \binom{d}{k} \cdot (1-r^p)^k \left(\frac{r^p}{\sqrt{p!}}\right)^{d-k}$	$\frac{1}{p!}\left(\frac{2r_S r_n}{h^2}\right)^p$	$\frac{1}{\sqrt{p!}}\left(\frac{\sqrt{2}r_1}{h}\right)^p$	$\frac{1}{p!}\left(\frac{2r_S r_T}{h^2}\right)^p \cdot e^{(r_S+r_T)^2/h^2}$
global error	absolute	absolute	relative	relative
prerequisites	$s_j, t_i \in [0,1]^d$	–	$f_j \geq 0$	$f_j \geq 0$

$\binom{p-1+d}{d}$ series terms. Thus, for dimensions $d > 3$, the FGT is clearly inferior in terms of costs. Furthermore, our algorithm combines the best features of both the Improved FGT and the Dual-Tree FGT. It comes with a simple Taylor series expansion which allows for an efficient estimation of the local approximation costs. The corresponding error bound for a series truncated via $\|\alpha\|_1 < p$ is of order $\mathcal{O}\big((p!)^{-1}\big)$ — instead of $\mathcal{O}\big((p!)^{-\frac{1}{2}}\big)$ as for the Dual-Tree FGT — and decreases for smaller node radii r_S and r_T. Like the DFGT, our method features a relative error estimate. Furthermore, the tree structures yield an improved behavior for the whole range of bandwidths h. This will be shown in detail in the following section.

4 Numerical results

We now present empirical results of our new algorithm using synthetic as well as real-world data and compare them to those obtained by direct evaluation. The synthetic datasets comprise uniformly distributed points in the hypercube $[0,1]^d$ as well as clustered data sampled from multiple d-variate normal distributions. The data are generated by the MT19937 "Mersenne Twister" random number generator [16] which produces equidistributed samples in high-dimensional spaces. Furthermore, we choose different real-world datasets with up to 62 dimensions. Their features are given in Table 3. Here, the data are coordinate-wise shifted and scaled to fit into $[0,1]^d$. For our experiments we select the first $50,000$ samples of each dataset. To get lower-dimensional data, we reduce the respective datasets by taking only the first few coordinates. For example, cMo3 denotes the dataset consisting of the first **3** coordinates of the first $50,000$ points of the **co**rel**M**oments dataset.

Note that the exponential function provided by the GNU ISO C++ Library 4.4.3 comes with a runtime anomaly that causes above-average evaluation times for input values in $[-748,-708]$, see Figure 7. For this reason, in our implementations, we employ the modification

$$\text{my\_exp}(x) \stackrel{\text{def}}{=} \{0 \mid x \leq -708, \quad \exp(x) \mid \text{otherwise}\},$$

Table 3 Real-world datasets.

name	dim.	# data	area	repository
acoustic	50	78823	vehicle sensors	LIBSVM Data
bio	62	208506	biol. image features	Signal Compression Lab, UCSB
codrna	8	59535	RNA sequence	LIBSVM Data
corelHisto	32	68040	image features	UCI Machine Learning Rep.
corelMoments	9	68040	image features	UCI Machine Learning Rep.
corelTexture	16	68040	image features	UCI Machine Learning Rep.
covtype	55	581012	forestry	UCI Machine Learning Rep.
galaxy	21	184211	astronomy	Dep. of Astronomy, UMass
shuttle	9	58000	NASA log data	UCI Machine Learning Rep.

which does not affect the actual results of the exp-function for double precision but has a positive effect on its runtime. The code is compiled with the GNU C++ Compiler 4.4.3 using the optimization flag `-O6` and is run on an Intel Xeon W3520 @ 2.67GHz. Runtimes are given in seconds and include the time for the tree construction of the OFGT. For each test run, the source and target points coincide (thus $N = M$), the f_j are uniformly distributed in $[0,1]$, and the kernel bandwidth h is varied within the range $[10^{-4}, 10^2]$. Unless stated otherwise, the relative precision of the OFGT is fixed as $\varepsilon = 10^{-6}$.

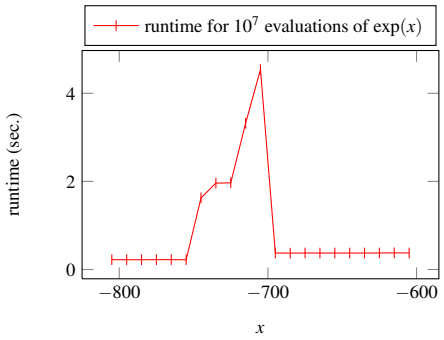

Fig. 7 Runtime anomaly of the exponential function $exp(x)$ provided in `cmath` of the GNU ISO C++ Library 4.4.3 for varying $x \in [-800,-600]$. For each evaluation point x_0, we generate 10^7 random points in the interval $[x_0-5, x_0+5]$. Then, we sum up the exponentials of all these points and measure the time for the whole process. This experiment is repeated three times and the average runtime is presented here.

4.1 Influence of the kernel bandwidth

As a preliminary experiment, we study the dependency of our algorithm on the bandwidth h. So far, in the theoretical complexities given in subsection 2.7, we did not consider the influence of h. However, the practical runtimes heavily depend on it. The reason for this behavior can be seen by analyzing the kernel matrix K, $K_{ij} = e^{-\|s_i - s_j\|^2/h^2}$

for the two degenerate cases $h \to 0$ and $h \to \infty$. In the first case, K tends to the identity matrix while in the second case, K tends to a singular matrix with $K_{ij} = 1$ for all i, j. Hence, for very small bandwidths, the OFGT algorithm approximates the Gauss transform basically via direct evaluations for the nearest-neighbors and some far-field approximations for few lower-level nodes which contain the majority of the sources. For very large bandwidths on the other hand, the contributions of all sources are efficiently combined in a single Taylor series expansion. Finally, for moderate bandwidths, the algorithm locally switches between the different approximation methods and falls back to direct evaluation in the worst case. Table 4 gives empirical evidence for this behavior of our algorithm. It shows the number of applications of the three different evaluation methods for experiments with the 5-dimensional uniform dataset and the 9-dimensional shuttle dataset, both with a relative precision of $\varepsilon = 10^{-6}$. Here, the worst case occurs for the uniform data with $h = 0.5$, where all source–target interactions are computed via direct evaluation in the 2048^2 pairs of tree leaves, each of which contains about 30 points. Moreover, the series expansion is applied in the uniform case only for $h > 0.5$, which results in a dramatic drop of the runtimes. For the real-world data, Taylor expansion is applied already for $h \geq 0.025$, the maximal runtime is considerably smaller, and the decrease in runtimes for increasing bandwidth is much smoother. Altogether, the performance strongly depends on the respective h.

Table 4 OFGT runtimes (of) for $\varepsilon = 10^{-6}$ and number of applications of the three evaluation methods: mean value approximation (#m), Taylor expansion (#t) and direct evaluation (#d).

	uniform, $d = 5$, $N = 50,000$				shuttle, $d = 9$, $N = 50,000$			
h	of	#m	#t	#d	of	#m	#t	#d
0.0001	0.11	11842	0	2448	2.39	3739837	0	33669
0.001	0.17	72120	0	8569	1.48	678050	0	37619
0.01	1.39	276368	0	121937	10.17	291702	0	352639
0.025	5.84	250332	0	224950	32.26	487718	615	1138856
0.05	10.40	365126	0	403598	67.01	590377	12818	2235853
0.1	39.73	677978	0	1568648	63.70	94041	119547	1280971
0.25	103.78	52009	0	4142064	27.33	556	100299	283836
0.5	105.08	0	0	4194304	13.38	0	29785	52356
1	6.79	0	1024	0	5.04	0	6959	5276
2.5	0.41	0	1	0	3.90	0	1	0
5	0.20	0	1	0	0.45	0	1	0

4.2 Uniform data

Now, we consider the case of uniform data in more detail. To this end, let us point out that uniform data is the worst-case scenario in terms of algorithmic performance, since the tree structure can neither detect any far-out clusters that allow to utilize the mean value method nor clusters with particularly small radii that allow a Taylor

approximation with small truncation parameter p. Moreover, uniformly distributed data in higher dimensions $d \gg 10$ are uncommon and rather meaningless in most areas of data mining.

The runtimes of empirical tests with uniformly distributed data can be found in Figure 8 and Table 5. The two different runtime regimes of the direct computation (low bandwidths vs. average and high bandwidths) result from our my_exp-function. However this behavior is common for any efficient implementation of the exponential function. Interestingly, the runtimes for 2, 3, and 4 dimensions hardly differ, which could be explained by dominating costs of the exp-function (as opposed to basic arithmetic operations) or caching effects.

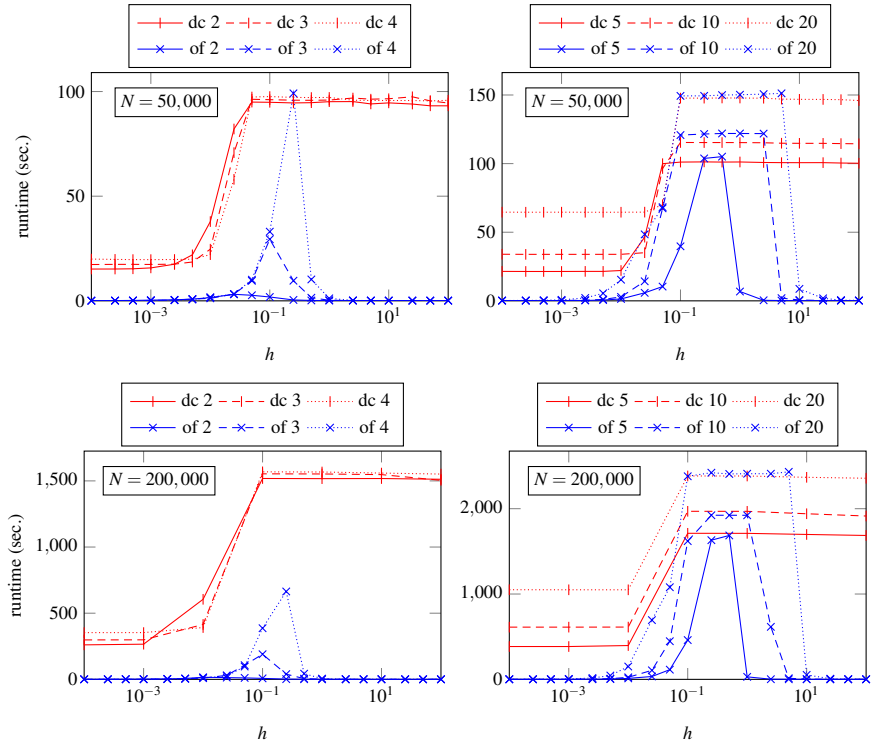

Fig. 8 Runtimes of direct computation (dc) and the OFGT (of) with $\varepsilon = 10^{-6}$ for uniform data in dimensions $d = 2,3,4$ *(left)* and $d = 5,10,20$ *(right)* with 50,000 *(top)* and 200,000 points *(bottom)*.

The runtimes of the OFGT algorithm undercut these of the direct variant by several orders of magnitude in dimensions up to 3 for all tested bandwidths. For higher dimensional uniform data, the Taylor series approximation with $\mathcal{O}(\binom{p-1+d}{d} \cdot (N_S + M_T))$ costs gets less efficient compared to direct evaluation with $\mathcal{O}(d \cdot N_S \cdot M_T)$ costs. Consequently, while the OFGT is still very fast for low and high bandwidths, the runtimes for an increasingly broader range of bandwidths approach those of direct com-

putation, starting at dimension 4 for 50,000 points and at dimension 5 for 200,000 points. If a lower precision (see Table 5 for $\varepsilon = 10^{-2}$) is acceptable, this effect is less severe. Note that in case the OFGT falls back to the direct method for all source–target interactions, the corresponding runtimes are slightly (up to 7%) higher than direct computation due to the computational overhead of the tree structure.

Table 5 Runtimes of direct computation (dc) and the OFGT (of) with $\varepsilon = 10^{-2}, 10^{-6}, 10^{-10}$ for uniform data (50,000 points) of different dimensions from 1 to 20. Wherever the OFGT fails to outperform the direct version, results are in bold type.

setting\h		0.001	0.01	0.025	0.05	0.1	0.25	0.5	1	2.5	5	10	100
of^{-2}	1	0.22	0.16	0.15	0.14	0.13	0.10	0.10	0.09	0.08	0.09	0.09	0.09
of^{-6}	1	0.27	0.21	0.17	0.15	0.14	0.11	0.09	0.10	0.09	0.10	0.10	0.10
of^{-10}	1	0.26	0.22	0.18	0.17	0.14	0.11	0.09	0.09	0.09	0.08	0.09	0.08
dc	1	19.6	63.2	86.3	90.1	90.2	90.3	88.8	89.4	90.4	88.6	89.0	87.7
of^{-2}	2	0.33	1.08	1.67	1.56	1.01	0.35	0.16	0.11	0.10	0.10	0.09	0.09
of^{-6}	2	0.34	1.56	3.12	2.66	1.84	0.43	0.18	0.12	0.10	0.09	0.10	0.10
of^{-10}	2	0.34	1.93	5.37	4.33	2.69	0.52	0.20	0.12	0.09	0.09	0.09	0.08
dc	2	15.7	37.9	82.1	95.0	94.9	94.6	94.7	95.2	95.2	94.1	94.6	93.2
of^{-2}	3	0.16	1.13	2.52	5.64	14.1	5.75	1.02	0.19	0.12	0.11	0.11	0.11
of^{-6}	3	0.17	1.19	3.25	9.58	29.4	9.62	1.48	0.28	0.15	0.12	0.12	0.11
of^{-10}	3	0.16	1.33	4.46	13.3	42.2	16.5	2.21	0.35	0.17	0.12	0.11	0.10
dc	3	17.4	24.7	70.8	96.3	96.1	95.9	95.8	95.9	96.9	96.4	96.6	94.5
of^{-2}	4	0.13	1.59	2.59	7.01	18.3	51.5	3.48	0.58	0.14	0.12	0.12	0.11
of^{-6}	4	0.16	1.72	3.22	10.1	33.1	**99.1**	10.2	1.19	0.20	0.15	0.12	0.11
of^{-10}	4	0.17	1.78	5.16	14.1	46.2	**102**	29.7	2.91	0.32	0.20	0.15	0.12
dc	4	19.7	22.1	58.0	97.4	97.6	**97.2**	97.1	97.2	95.9	95.7	95.7	95.8
of^{-2}	5	0.12	1.18	4.98	8.99	24.5	80.8	39.3	1.84	0.19	0.13	0.12	0.11
of^{-6}	5	0.17	1.39	5.84	10.4	39.7	**104**	**105**	6.79	0.41	0.20	0.15	0.13
of^{-10}	5	0.18	1.57	7.10	17.6	55.4	**107**	**107**	21.7	0.79	0.33	0.2	0.14
dc	5	21.4	22.1	46.4	99.9	101	**101**	**101**	101	101	101	101	100
of^{-2}	10	0.22	2.35	10.5	57.0	113	**122**	**123**	**123**	3.21	0.32	0.23	0.17
of^{-6}	10	0.25	2.86	14.2	67.4	**121**	**122**	**122**	**122**	**122**	1.97	0.63	0.19
of^{-10}	10	0.25	3.84	17.1	76.0	**121**	**122**	**122**	**122**	**121**	15.1	1.32	0.21
dc	10	33.9	33.9	35.1	96.5	**115**	**115**	**115**	**115**	**115**	115	115	114
of^{-2}	20	0.59	10.9	38.9	66.1	**152**	**151**	**151**	**152**	**150**	8.64	0.54	0.27
of^{-6}	20	0.72	15.4	48.4	68.3	**149**	**149**	**150**	**150**	**151**	**151**	8.71	0.31
of^{-10}	20	0.85	19.6	54.6	68.6	**147**	**148**	**147**	**149**	**152**	**148**	**158**	0.57
dc	20	64.6	64.6	64.6	70	**148**	**148**	**148**	**148**	**148**	**147**	**147**	146

4.3 Clustered data

Next, we evaluate runtime results for clustered data. For a given dimension d, we fix 50 random center points, uniformly distributed in $[0,1]^d$. For each center μ, 10^3 data points are sampled from the d-variate normal distribution

$$p(x) = \frac{1}{(2\pi)^{\frac{d}{2}}\sigma} \exp\left(-\frac{\|x-\mu\|^2}{2\sigma^2}\right)$$

with variance $\sigma = 0.01$. The runtime results for different dimensions are shown in Figure 9. Obviously, the performance of the OFGT compared to direct computation is much better than for uniformly distributed data. For up to 4 dimensions, the fast variant yields a speedup factor of at least 10. For $d = 10$, the worst-case bandwidth of $h = 0.5$ results in a speedup of 2, while for $d = 20$, the OFGT still outperforms the direct variant clearly for the whole range of bandwidths. Naturally, the above choice of the variance $\sigma = 0.01$ in the normal distribution is crucial to yield tree nodes with small associated radii. Consider for example the d-dimensional sphere with radius r_d containing 90% of the samples of the above normal distribution. While the respective radii grow moderately with dimension ($r_2 \approx 0.022$, $r_5 \approx 0.031$, $r_{20} \approx 0.054$), the volumes of the associated spheres rapidly decrease to zero ($\mathrm{vol}_2(r_2) \approx 1.52 \times 10^{-3}$, $\mathrm{vol}_5(r_5) \approx 1.51 \times 10^{-7}$, $\mathrm{vol}_{20}(r_{20}) \approx 1.15 \times 10^{-27}$). This effect — called the "empty space phenomenon" — is of course well-known, see e.g. [20].

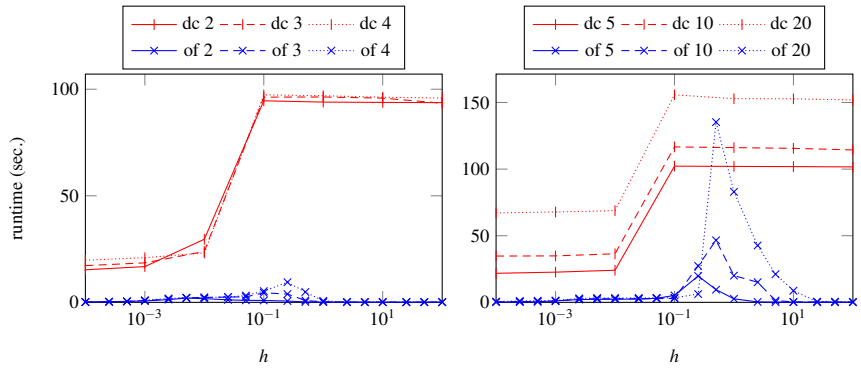

Fig. 9 Runtimes of direct computation (dc) and the OFGT (of) with $\varepsilon = 10^{-6}$ for 50 clusters of 10^3 normally distributed points each (with variance $\sigma = 0.01$) in dimensions $d = 2,3,4$ *(left)* and $d = 5,10,20$ *(right)*.

4.4 Real-world data

The runtime results for real-world datasets (Figure 10 and Table 6) basically have similar characteristics as the results for synthetic data. The overall performance is clearly better than for uniform data and only slightly worse than for the clustered data. For all tested datasets up to 6 dimensions and also the 9-dimensional shuttle data, the OFGT rigorously outperforms the direct variant for the whole range of bandwidths. For the datasets of dimensions of 16 or higher, the OFGT is still fast for low and high bandwidths, and it achieves roughly the same runtime as direct computation for a wider range of average bandwidths.

The reason for the better performance here than for uniform data is of course the inherent structure of most real-world data. While for uniformly distributed data the dual-tree approach has no essential advantage over an adequately sized regular grid

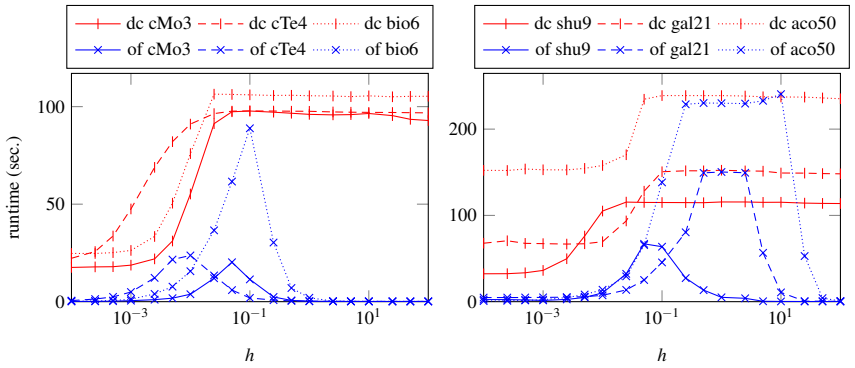

Fig. 10 Runtimes of direct computation and the OFGT for real-world datasets (50,000 points) corelMoments ($d = 3$), corelTexture ($d = 4$), bio ($d = 6$) *(left)* and shuttle ($d = 9$), galaxy ($d = 21$), acoustic ($d = 50$) *(right)*.

as used by the FGT algorithm, the local patterns in real-world data generally result in smaller tree node radii, hence the faster Taylor series approximation can be applied more often.

Finally, in Table 6, we present some more runtime results of the OFGT for different relative precisions $\varepsilon = 10^{-2}, 10^{-6}, 10^{-10}$. These figures show that performance can be improved significantly at the expense of lower precision. On the other hand, if higher precision is required, runtimes increase but never get perceptibly higher than that of direct computation.

5 Conclusions

The newly proposed Optimized Fast Gauss Transform seeks to combine the advantages of all previously existing fast Gauss approximation methods based on Hermite or Taylor expansions. It comes with a refined Taylor series expansion and a better cost estimation. Numerical experiments demonstrate that the runtime performance crucially depends on the bandwidth.

For uniformly distributed data, our algorithm provides speedup factors of up to 100 for three- or lower-dimensional problems; similar performance is achieved in higher-dimensional settings for low and high bandwidths. However, the higher the dimension, the wider the range of average bandwidths gets where our method can no longer outperform direct computation. Note here that this effect is even more severe for the other existing fast Gauss transforms.

For highly clustered data, our algorithm takes advantage of its tree structure, and the fast series approximation can be applied more often. Thus, speedup factors of 10 to 100 are achieved for up to five-dimensional data for all tested bandwidths. Even for $d = 20$, our OFGT is always faster than the direct method.

Real-world datasets generally yield performance results which are between those for uniform data and for clustered data. The tree structures can exploit intrinsic low-

Table 6 Runtimes of direct computation (dc) and OFGT (of) for real-world data of different dimensions.

setting \ h	0.001	0.01	0.025	0.05	0.1	0.25	0.5	1	2.5	5	10	100
of^{-2} cMo3	0.48	2.62	7.12	8.42	4.20	1.13	0.33	0.16	0.11	0.11	0.11	0.10
of^{-6} cMo3	0.51	3.81	12.0	20.2	11.4	2.40	0.65	0.24	0.13	0.12	0.11	0.12
of^{-10} cMo3	0.50	4.88	16.3	33.6	21.0	4.02	1.10	0.36	0.17	0.13	0.11	0.10
dc cMo3	18.7	55.2	91.3	97.4	97.8	97.2	96.6	96.0	95.8	95.9	96.5	92.9
of^{-2} cTe4	3.63	10.6	5.14	2.11	0.69	0.63	0.24	0.17	0.15	0.13	0.13	0.12
of^{-6} cTe4	4.94	23.7	13.4	5.94	1.92	0.79	0.40	0.22	0.17	0.15	0.14	0.12
of^{-10} cTe4	5.90	38.4	25.4	12.3	4.40	1.64	0.77	0.33	0.21	0.18	0.15	0.12
dc cTe4	47.5	91.1	96.4	97.7	97.8	97.8	97.7	97.6	97.3	97.1	97.1	96.8
of^{-2} aco5	0.80	5.17	10.4	16.6	25.4	50.4	24.7	1.39	0.18	0.14	0.13	0.13
of^{-6} aco5	0.89	6.12	12.9	21.5	38.2	88.0	52.5	6.19	0.33	0.17	0.14	0.12
of^{-10} aco5	0.98	7.15	15.2	25.5	49.4	101	80.2	28.5	0.58	0.25	0.20	0.14
dc aco5	22.9	39.0	71.0	103	103	103	103	103	103	102	103	102
of^{-2} bio6	1.16	11.6	25.8	42.9	41.4	9.63	1.88	0.47	0.16	0.14	0.14	0.13
of^{-6} bio6	1.26	15.6	36.6	61.6	88.9	30.3	7.00	2.01	0.30	0.22	0.18	0.14
of^{-10} bio6	1.33	18.8	43.6	71.9	100	76.1	21.6	4.67	0.90	0.29	0.21	0.14
dc bio6	26.3	75.9	106	106	106	106	106	106	105	105	106	105
of^{-2} cod8	0.75	5.38	14.8	30.3	46.1	70.2	65.7	5.93	1.10	0.23	0.20	0.17
of^{-6} cod8	0.77	6.72	20.4	39.5	53.7	**111**	**114**	58.7	3.06	0.62	0.28	0.17
of^{-10} cod8	0.80	8.10	25.2	44.9	59.8	**115**	**115**	**115**	19.6	2.77	0.60	0.19
dc cod8	28.9	44.0	71.2	109	110	**110**	**110**	**110**	109	109	109	109
of^{-2} shu9	1.45	6.50	17.0	24.0	21.8	9.49	3.85	1.55	0.44	0.19	0.17	0.15
of^{-6} shu9	1.48	10.2	32.3	67.0	63.7	27.3	13.4	5.04	3.90	0.45	0.27	0.16
of^{-10} shu9	1.53	13.6	43.9	89.3	108	69.0	35.2	16.9	5.60	1.83	0.92	0.20
dc shu9	36.2	105	115	115	115	115	115	116	116	115	115	114
of^{-2} cov10	0.30	1.81	5.01	11.3	27.7	83.0	**116**	22.5	1.39	0.34	0.23	0.18
of^{-6} cov10	0.32	2.06	6.65	16.6	44.8	**118**	**122**	**124**	12.9	1.36	0.61	0.18
of^{-10} cov10	0.35	2.32	8.25	21.9	60.3	**122**	**122**	**122**	106	9.74	1.34	0.22
dc cov10	34.3	36.8	65.3	115	116	**116**	**116**	**116**	116	115	115	114
of^{-2} cTe16	0.98	10.8	28.1	49.4	74.5	114	61.1	10.5	1.59	0.92	0.46	0.23
of^{-6} cTe16	1.11	14.0	37.3	64.7	97.1	133	**137**	126	17.2	3.71	1.36	0.26
of^{-10} cTe16	1.20	17.0	44.6	75.0	110	136	**138**	138	130	34.0	7.66	0.38
dc cTe16	52.8	100	132	141	142	138	**136**	139	136	134	134	133
of^{-2} gal21	4.92	6.96	11.5	19.3	36.2	61.0	114	151	24.1	2.12	0.58	0.28
of^{-6} gal21	4.94	7.34	13.6	25.1	45.5	80.4	149	150	149	56.6	10.8	0.30
of^{-10} gal21	4.91	7.62	15.7	30.2	52.5	122	151	152	151	151	131	0.56
dc gal21	67.3	69.5	93.6	128	151	152	152	152	152	151	149	148
of^{-2} cHi32	2.09	9.76	28.2	61.7	112	167	180	178	119	31.5	5.78	0.39
of^{-6} cHi32	2.47	11.0	37.8	81.0	138	179	180	180	179	177	92.9	0.81
of^{-10} cHi32	2.79	12.3	46.6	96.3	157	184	183	183	183	183	182	6.60
dc cHi32	103	106	169	192	191	187	186	185	185	185	184	183
of^{-2} aco50	1.99	11.9	24.6	53.0	108	204	231	230	212	157	6.31	0.56
of^{-6} aco50	2.28	13.8	29.3	65.6	138	229	230	230	230	233	**241**	0.75
of^{-10} aco50	2.61	15.5	33.9	78.1	163	234	234	234	234	231	232	32.1
dc aco50	153	158	170	235	239	239	239	239	238	237	**237**	235
of^{-2} bio62	4.71	22.6	57.5	103	152	201	232	259	201	21.7	7.23	0.69
of^{-6} bio62	5.73	25.9	68.9	120	171	224	256	264	264	269	263	1.49
of^{-10} bio62	6.78	29.6	79.6	135	186	241	269	269	269	273	268	16.8
dc bio62	197	195	227	260	277	278	278	278	277	276	276	273

dimensional features and allow for a more frequent application of the fast approximation methods. Thus, problems of dimensions up to 9 can be efficiently treated for arbitrary bandwidths, while for considerably higher dimensions, similar limitations as with uniform data are present.

The current algorithm can directly be applied to bandwidth selection in kernel density estimation, where all weight coefficients are positive constants. This has already been demonstrated for the Improved Fast Gauss Transform in [17]. Note here that the restriction to non-negative weights f_j is no general obstacle, but can be circumvented using various approaches. The most straightforward variant is to split $\mathbf{f}$ into $\mathbf{f}^+$ and $\mathbf{f}^-$, padded with zeros accordingly, to run our algorithm twice, and to combine the two results properly. Another option is to add a sufficiently large constant c to $\mathbf{f}$ to make all entries non-negative, to compute the transform for both, $c \cdot \mathbf{1}$ and $f + c \cdot \mathbf{1}$, and to combine the two results in the end. Here, stability and error control may be an issue, though.

Without the restriction to non-negative weights, our method can also be utilized to solve linear systems $K \cdot \mathbf{x} = \mathbf{b}$ with the Gaussian kernel matrix $K_{ij} = k_h(s_i, s_j)$ within e.g. a preconditioned conjugate gradient method. This will allow us to tackle further applications like for example Gaussian process regression or regularized least-squares classification.

Note that other approaches have recently been proposed for the approximation of the Gauss transform. TAUSCH and WECKIEWICZ [19] use Chebyshev expansions to approximate the solution globally via a single series expansion. The corresponding time complexity is $\mathcal{O}(\binom{p+1+d}{d} \cdot (N+M))$, where p is the associated series truncation parameter. KUNIS et al. [12, 13] introduce another variant based on the nonequispaced Fast Fourier Transform and demonstrate competitive results in 1 and 2 dimensions. An in-depth performance analysis of these algorithms in higher dimensions is however still to be done.

In the end, let us remark that profound concentration effects may occur when dealing with high-dimensional problems, see [11]. Then, the Euclidean distance and the Gaussian kernel might necessarily no longer be an appropriate choice. It is presently unclear if alternatives like the so-called p-Gaussian kernel $k(t,s) = e^{-\|t-s\|^p/h^p}$ as suggested in [8] are really remedying the problem. On the other hand, real-world high-dimensional data almost surely possess intrinsic low-dimensional structures and manifolds for which concentration may pose no longer a problem. Moreover, a virtually uniformly distributed dataset in high dimensions is of no major interest in the praxis of data mining. In the ideal case, an algorithm dealing with high-dimensional data should exploit lower-dimensional patterns even without explicitly recovering these, and the curse of dimensionality should only be present with respect to the intrinsic lower dimension, but not with respect to the nominal dimension. Furthermore, it should fall back to methods with linear complexity with respect to the dimension, if no intrinsic structure is present. The basic framework of our OFGT algorithm partly features this behavior and even allows for further improvements in this direction.

References

1. Ayyagari, V.R., Boughorbel, F., Koschan, A., Abidi, M.A.: A new method for automatic 3D face registration. In: Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., vol. 3, p. 119 (2005)
2. Baxter, B.J.C., Roussos, G.: A new error estimate of the fast Gauss transform. SIAM J. Sci. Comput. **24**(1), 257–259 (2002)
3. Beatson, R., Greengard, L.: A short course on fast multipole methods. In: Wavelets, Multilevel Methods and Elliptic PDEs, pp. 1–37. Oxford University Press (1997)
4. Boughorbel, F., Koschan, A., Abidi, M.: A new multi-sensor registration technique for three-dimensional scene modeling with application to unmanned vehicle mobility enhancement. In: Unmanned Ground Veh. Technol. VII (SPIE Conf. Proc.), vol. 5804, pp. 174–181 (2005)
5. Broadie, M., Yamamoto, Y.: Application of the fast Gauss transform to option pricing. Manag. Sci. **49**(8), 1071–1088 (2003)
6. Broadie, M., Yamamoto, Y.: A double-exponential fast Gauss transform algorithm for pricing discrete path-dependent options. Oper. Res. **53**(5), 764–779 (2005)
7. Elgammal, A., Duraiswami, R., Davis, L.S.: Efficient kernel density estimation using the fast Gauss transform with applications to color modeling and tracking. IEEE Trans. Pattern Anal. Mach. Intell. **25**(11), 1499–1504 (2003)
8. Franois, D., Wertz, V., Verleysen, M.: About the locality of kernels in high-dimensional spaces. In: Proc. Int. Symp. Appl. Stoch. Models Data Anal. (ASMDA), vol. 8, pp. 238–245 (2005)
9. Greengard, L., Rokhlin, V.: A fast algorithm for particle simulations. J. Comp. Phys. **73**(2), 325–348 (1987)
10. Greengard, L., Strain, J.: The fast Gauss transform. SIAM J. Sci. Stat. Comput. **12**(1), 79–94 (1991)
11. Hegland, M., Pestov, V.: Additive models in high dimensions. In: Proc. 12th Comput. Tech. Appl. Conf. (CTAC-2004), *ANZIAM J.*, vol. 46, pp. C1205–C1221 (2005)
12. Keiner, J., Kunis, S., Potts, D.: Fast summation of radial functions on the sphere. Comput. **78**, 1–15 (2006)
13. Kunis, S.: Nonequispaced FFT: Generalisation and Inversion. Shaker (2006)
14. Lee, D., Gray, A.: Faster Gaussian summation: theory and experiment. In: Proc. 22nd Annu. Conf. Uncertain. Artif. Intell. (UAI-06). AUAI Press, Arlington (2006)
15. Lee, D., Gray, A., Moore, A.: Dual-tree fast Gauss transforms. Adv. Neural Inf. Process. Syst. **18**, 747–754 (2006)
16. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**(1), 3–30 (1998)
17. Raykar, V.C., Duraiswami, R.: Very fast optimal bandwidth selection for univariate kernel density estimation. Tech. Rep. CS-TR-4774, Dep. Comput. Sci., Univ. Maryland, Coll. Park (2005)
18. Raykar, V.C., Yang, C., Duraiswami, R., Gumerov, N.A.: Fast computation of sums of Gaussians in high dimensions. Tech. Rep. CS-TR-4767, Dep. Comput. Sci., Univ. Maryland, Coll. Park (2005)
19. Tausch, J., Weckiewicz, A.: Multidimensional fast Gauss transforms by Chebyshev expansions. SIAM J. Sci. Comput. **31**, 3547–3565 (2009)
20. Verleysen, M.: Learning high-dimensional data. In: Limit. Future Trends Neural Comput., *3*, vol. 186, pp. 141–162 (2003)
21. Wissel, D.: The discrete Gauss transform – fast approximation algorithms and applications in high dimensions. Diploma thesis, Inst. Numer. Simul., Univ. Bonn (2008)
22. Yang, C., Duraiswami, R., Davis, L.: Efficient kernel machines using the improved fast Gauss transform. In: Adv. Neural Inf. Process. Syst., vol. 17, pp. 1561–1568. MIT Press (2004)
23. Yang, C., Duraiswami, R., Gumerov, N.A., Davis, L.: Improved fast Gauss transform and efficient kernel density estimation. In: Proc. 9th IEEE Int. Conf. Comput. Vis., vol. 1, pp. 664–671 (2003)