Preprint ANL/MCS-P4001-1212

# A FAST SUMMATION TREE CODE FOR MATÉRN KERNEL

JIE CHEN*, LEI WANG†, AND MIHAI ANITESCU*

**Abstract.** The Matérn family of functions is a widely used covariance kernel in spatial statistics for Gaussian process modeling, which in many instances requires calculations with a covariance matrix. In this paper, we design a fast summation algorithm for the Matérn kernel in order to efficiently perform matrix-vector multiplications. This algorithm is based on the Barnes–Hut tree code framework and addresses several practical issues: the anisotropy of the kernel, the nonuniform distribution of the point set, and a tight error estimate of the approximation. Even though the algorithmic details differ from the standard tree code in several aspects, empirically the computational cost of our algorithm scales as $O(n \log n)$ for $n$ points. Comprehensive numerical experiments are shown to demonstrate the practicality of the design.

**Key words.** Matérn kernel, Gaussian process, fast summation, tree code

**AMS subject classifications.** 65F30

## 1. Introduction.

The Matérn kernel [31, 33, 28] consists of a family of Matérn functions that are defined based on the modified Bessel functions of the second kind of different orders. The Matérn kernel is positive definite, and it is often used as a covariance function in modeling Gaussian processes for its flexibility in capturing local smoothness of the data. It entails a wide array of applications in spatial statistics, especially geostatistics [15, 33].

The Matérn kernel gives rise to a positive definite covariance matrix $\Phi$, which is fully dense and whose size scales with the square of the number $n$ of observations of the underlying process. The matrix-vector multiplication with respect to $\Phi$ is crucial in many statistical problems, such as sampling, maximum likelihood estimation, and interpolation (also known as kriging) [31, 33]. Some of these problems require the solution of a linear system with respect to $\Phi$, whereby an iterative method with an efficient calculation of the product $\Phi \boldsymbol{q}$ for any vector $\boldsymbol{q}$ is one of the most successful solution techniques [3, 32]. In some other problems, there is no linear system to solve; but the matrix-vector multiplication is an essential tool for a matrix-free style of technique to work for large $n$ [11].

Motivated by the needs of efficiently computing the product $\Phi \boldsymbol{q}$ for the Matérn kernel, we design a fast summation algorithm that runs asymptotically faster than $O(n^2)$, the cost of a straightforward calculation. The goal of the design is an algorithm that handles various practical situations, including arbitrary Matérn orders, multiple vectors for the same matrix, different point set distributions, and the anisotropy when defining distances for high-dimensional points. To this end, we present an algorithm based on the tree code framework pioneered by Barnes and Hut [5]. The tree code was initially designed to efficiently perform force calculations for gravitational $n$-body problems with an $O(n \log n)$ computational complexity. It was later developed for various kernels and different applications (see, e.g., [25, 23, 22]). For the Matérn kernel, preliminary work [3] was conducted for the special order 1.5. The proposed algorithm here applies to an arbitrary order.

*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439. Emails: (jiechen, anitescu)@mcs.anl.gov

†Department of Mathematical Sciences, University of Wisconsin–Milwaukee, Email: wang256@uwm.edu

**1.1. Design features.** The initial step of a tree code is a hierarchical partitioning of the space. In order to cope with the arbitrary shape and distribution of the point set and the anisotropy in the Matérn kernel, we use a point set partitioning scheme that is different from the traditional quadtree (octree) or k-d tree partitioning. Our scheme is based on the principal component analysis that maximizes the separation of the points, so that the resulting clusters are as compact as possible. This scheme encourages a tighter error estimate and well handles kernel anisotropy.

In our tree code, the Taylor approximation is written in the form of a double expansion (that is, the kernel is expanded at both arguments). Double expansions are typical in the FMM-type of algorithms (see, e.g., [29, 18, 19, 14]), but they rarely appear in tree codes (see [8] for an example). The rationale for using double expansions is to reduce the number of source–target pairs that require the computation and storage of the expansion coefficients.

Note that even though abundant results of the expansions of the Bessel functions are known (see, e.g., [16, 1]), it is not straightforward to derive an expansion that is inexpensive to evaluate for the Matérn kernel. The Taylor coefficients here are derived through the use of the properties of the Bessel functions and are computed based on a recurrence relation.

Because of the limited results known for the Taylor approximation of the Matérn kernel, an innovation of this work is that we use a data analysis approach to estimate the truncation error, so as to determine when and how the approximation is performed. The idea of the analysis is to regress the error on several factors, such as centroid distance and cluster radius, based on a set of samples. The form of the regression is motivated by the analytic error bounds seen for other extensively studied kernels, and the data fitting approach conceptually yields more realistic error estimates than does a standard analytic analysis.

We note that although a general tree code framework yields $O(n \log n)$ computational complexity for a set of $n$ points, there is in fact no strict guarantee on achieving such a performance for arbitrary kernels, even if the points are uniformly distributed. The rationale for the $O(n \log n)$ cost is based on the assumption that the Taylor approximation is used whenever the ratio between the cluster radius and the centroid distance falls within a fixed threshold (known as the *multipole acceptance criterion* [5, 4, 30] or *admissibility condition* [9]). This ensures that there is a constant rate of reduction in the summation cost across the tree levels. On the other hand, our algorithm dynamically determines the use of the approximation, and we do not predefine a threshold for this ratio. This distinction makes it hard to analytically conclude a precise complexity of our algorithm. Nevertheless, experimental results for uniformly distributed points agree with the $O(n \log n)$ scaling.

**1.2. Fast summation methods.** Fast summation methods are an extensively studied subject, whereby a series expansion of the kernel is a central analytic tool. The two major variants, tree code and FMM, differ in whether the series expansion is reorganized throughout the hierarchical structure of the points. A contribution of this paper is an easily computable routine for the Taylor expansion of the Matérn kernel.

In the recent years, fast methods that are less dependent on kernel expansions were actively studied. A clear advantage of the kernel-independent methods is that they are designed to be applicable to a wide variety of kernels. In the description of these methods, either the FMM terminologies are maintained (e.g., kernel-independent FMM [35] and black-box FMM [17]), or the matrix terminologies are used to emphasize the alge-

braic angle (e.g., $\mathcal{H}$, $\mathcal{H}^2$ matrices [20, 21, 9] and HSS matrices [10]; in what follows we use the unified term "hierarchical matrices"). Kernel independent methods are based on the observation (which in many cases can be rigorously proven) that the recursive off-diagonal blocks of the matrix are low-rank. With this interpretation, the FMM expansions (multipole and local) share many similarities with the row and column subspace representations of the hierarchical matrices, and the FMM translation operators (M2M, M2L, and L2L) are not very different from the change of basis across levels for hierarchical matrices. One of the natural benefits of using matrix representations is that they in addition admit factorizations and hence preconditioners and direct solvers [9, 34, 10].

A key element of these matrix methods is to find the low-rank representations and their transformations across the tree levels. This corresponds to the design of the expansions and translation operators in the FMM terminology. The natural methods based on truncated SVDs typically incur an expensive quadratic cost, except for $\mathcal{H}$ matrices which can be constructed in linear time, but the saving is paid by later matrix-vector multiplications [21]. In some special scenarios, such as when a separable basis of the kernel is known [21], or when the kernel matrix approximates a boundary integral operator [26], the low-rank approximation can be constructed without the use of an SVD. For these special cases, the FMM analogs are black-box FMM [17] and kernel-independent FMM [35]. The former method approximates the kernel by using interpolating polynomials; the basis is clearly separable and the computation is economic. The latter method defines the various FMM translations via the solutions of small Fredholm integral equations on equivalent surfaces; it is applicable to Green kernels. On the other hand, randomized SVD methods [27, 24] can serve as a general-purpose technique, but they rely on the existence of a matrix-vector multiplication subroutine, which in many scenarios might be lacking and thus forms a causality dilemma.

Overall, a kernel-independent method is appealing just as its name suggests. Ridding the reliance on kernel-specific expansions widens the potential applicability of the method to new kernels and kernel variants. On the other hand, the performance of kernel-independent methods may no longer be as optimal as that of methods tailed for specific kernels. For example, preliminary experiments with the method [17] indicated that strong anisotropy significantly degrades the final accuracy, and the error behavior of the method [35] for the Matérn kernel is irregular. On the other hand, the method presented in this paper, which is based on an analytic expansion and a run-time error analysis, works well in controlling errors and handling anisotropy. Furthermore, the computation of the expansion is interesting in its own right and is probably useful for numerical analysis.

**2. Matérn kernel.** The Matérn function of a one-dimensional variable $r \geq 0$ of order $\nu > 0$ is defined as [28]

$$\phi(r) = \frac{(\sqrt{2\nu}r)^\nu \mathsf{K}_\nu(\sqrt{2\nu}r)}{2^{\nu-1}\Gamma(\nu)}, \tag{2.1}$$

where $\mathsf{K}_\nu$ is the modified Bessel function of the second kind of order $\nu$ and $\Gamma$ is the Gamma function. The denominator $2^{\nu-1}\Gamma(\nu)$ is used for normalization so that $\phi(0) = 1$ for any $\nu$. Figure 2.1 plots the function with several values of $\nu$.

When the function is used as a radial basis kernel, the variable $r$ is the elliptical distance between two $d$-dimensional points $\boldsymbol{x}$ and $\boldsymbol{y}$. Let $\boldsymbol{\ell} = [\ell_1, \ldots, \ell_d]$ be a vector

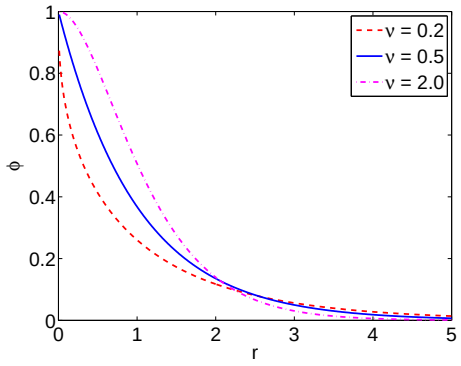



FIG. 2.1. *Matérn function with different values of* $\nu$.

of scaling factors, one for each coordinate. We formally define $r$ as

$$r = \sqrt{\sum_{i=1}^{d} \frac{r_i^2}{\ell_i^2}} \qquad \text{with} \qquad r_i = x_i - y_i. \tag{2.2}$$

For convenience of presentation, we write the Matérn kernel by abuse of notation in different forms: $\phi(r)$, $\phi(\boldsymbol{x} - \boldsymbol{y})$ or $\phi(\boldsymbol{x}, \boldsymbol{y})$. In different contexts these forms will not cause confusion.

The parameter $\nu$ is often called the *smoothness* because the function $\phi(|x|)$, $x \in \mathbb{R}$, has a higher differentiability at the origin when $\nu$ is larger. It also reflects the shape of the samples of the underlying Gaussian process, because when $\nu$ is small, the sample deemed as a function is rough and has strong oscillations. The parameter $\boldsymbol{\ell}$ is called the *scale* because it controls the scaling of the distance between two points.

**3. Fast summation with Matérn kernel.** Given a set of $n$ points $\{\boldsymbol{x}_j \in \mathbb{R}^d\}$ and a set of associated weights $\{q_j\}$, we are interested in computing the summations

$$s_i = \sum_{j=1}^{n} q_j \phi(\boldsymbol{x}_i - \boldsymbol{x}_j), \qquad \text{for} \quad i = 1, \ldots, n. \tag{3.1}$$

In the matrix representation this is equivalent to computing the matrix-vector product $\boldsymbol{s} = \Phi \boldsymbol{q}$ where $\Phi_{ij} = \phi(\boldsymbol{x}_i - \boldsymbol{x}_j)$.

It is sometimes confusing when one distinguishes $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ only by using the index. Therefore, we slightly change the notation from $\boldsymbol{x}_j$ to $\boldsymbol{y}_j$ and rewrite (3.1) as

$$s_i = \sum_{j=1}^{n} q_j \phi(\boldsymbol{x}_i, \boldsymbol{y}_j), \qquad \text{for} \quad i = 1, \ldots, n. \tag{3.2}$$

The points $\boldsymbol{y}_j$'s are the *sources*, and $\boldsymbol{x}_i$'s are the *targets*. At the heart of the fast summation is the Taylor approximation of $\phi$ at the centroid of a cluster of nearby sources and the centroid of nearby targets, so that one can replace the summation of the $n$ terms in (3.2) by the evaluation of a Taylor polynomial. For this, we use $C_s$ to denote a set of sources with a centroid $\boldsymbol{y}_c$, and use $C_t$ to denote a set of targets with a centroid $\boldsymbol{x}_c$. Further, we define the partial sum

$$s_i(C_s) := \sum_{\boldsymbol{y}_j \in C_s} q_j \phi(\boldsymbol{x}_i, \boldsymbol{y}_j).$$

When the whole set of points is partitioned into disjoint subsets $C_s$'s, we have

$$s_i = \sum_{C_s} s_i(C_s).$$

To express the Taylor expansion, we need the following notation for multivariate calculus. For $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and $\boldsymbol{j}, \boldsymbol{k} \in \mathbb{Z}_+^d$, the partial derivative $\partial_{\boldsymbol{x}}^{\boldsymbol{j}} := \partial_{x_1}^{j_1} \partial_{x_2}^{j_2} \cdots \partial_{x_d}^{j_d}$, the integer power $\boldsymbol{x}^{\boldsymbol{j}} := x_1^{j_1} x_2^{j_2} \cdots x_d^{j_d}$, the factorial $\boldsymbol{j}! := j_1! j_2! \cdots j_p!$, the binomial coefficient $\binom{\boldsymbol{k}}{\boldsymbol{j}} := \binom{k_1}{j_1}\binom{k_2}{j_2}\cdots\binom{k_d}{j_d}$, and the norm $\|\boldsymbol{j}\| := j_1 + j_2 + \cdots + j_d$. Note that the last notation means the 1-norm of the nonnegative integer vector $\boldsymbol{j}$; it is not to be confused with the 2-norm of a general vector.

Let the double expansion of $\phi$ at $\boldsymbol{x}_c$ and $\boldsymbol{y}_c$ be

$$\phi(\boldsymbol{x}_c + \Delta\boldsymbol{x}, \boldsymbol{y}_c + \Delta\boldsymbol{y}) = \sum_{\|\boldsymbol{j}\|=0}^{\infty} \sum_{\|\boldsymbol{k}\|=0}^{\infty} \frac{\partial_{\boldsymbol{x}}^{\boldsymbol{j}} \partial_{\boldsymbol{y}}^{\boldsymbol{k}} \phi(\boldsymbol{x}_c, \boldsymbol{y}_c)}{\boldsymbol{j}!\boldsymbol{k}!} (\Delta\boldsymbol{x})^{\boldsymbol{j}} (\Delta\boldsymbol{y})^{\boldsymbol{k}}. \tag{3.3}$$

This can be obtained by first expanding $\phi$ around $\boldsymbol{y}_c$, treating $\boldsymbol{x}_c + \Delta\boldsymbol{x}$ constant, then expanding $\phi$ around $\boldsymbol{x}_c$, treating $\boldsymbol{y}_c$ constant. The expansion is converging since the Matérn kernel is analytic outside the origin. It is used when $\|\Delta\boldsymbol{x}\| + \|\Delta\boldsymbol{y}\| < \|\boldsymbol{x}_c - \boldsymbol{y}_c\|$ to avoid conflict with the origin. Because $\partial_{\boldsymbol{x}}^{\boldsymbol{j}} \phi = (-1)^{\|\boldsymbol{j}\|} \partial_{\boldsymbol{y}}^{\boldsymbol{j}} \phi$, we can write

$$\phi(\boldsymbol{x}_c + \Delta\boldsymbol{x}, \boldsymbol{y}_c + \Delta\boldsymbol{y}) = \sum_{\|\boldsymbol{j}\|=0}^{\infty} \sum_{\|\boldsymbol{k}\|=0}^{\infty} \binom{\boldsymbol{j}+\boldsymbol{k}}{\boldsymbol{j}} \frac{\partial_{\boldsymbol{y}}^{\boldsymbol{j}+\boldsymbol{k}} \phi(\boldsymbol{x}_c, \boldsymbol{y}_c)}{(\boldsymbol{j}+\boldsymbol{k})!} (-\Delta\boldsymbol{x})^{\boldsymbol{j}} (\Delta\boldsymbol{y})^{\boldsymbol{k}}. \tag{3.4}$$

With (3.4), one can approximate the partial sum $s_i(C_s)$ by using an order-$(p_1, p_2)$ Taylor approximation:

$$s_i(C_s) \approx \sum_{\boldsymbol{y}_j \in C_s} q_j \sum_{\|\boldsymbol{j}\|=0}^{p_1} \sum_{\|\boldsymbol{k}\|=0}^{p_2} \binom{\boldsymbol{j}+\boldsymbol{k}}{\boldsymbol{j}} \frac{\partial_{\boldsymbol{y}}^{\boldsymbol{j}+\boldsymbol{k}} \phi(\boldsymbol{x}_c, \boldsymbol{y}_c)}{(\boldsymbol{j}+\boldsymbol{k})!} (\boldsymbol{x}_c - \boldsymbol{x}_i)^{\boldsymbol{j}} (\boldsymbol{y}_j - \boldsymbol{y}_c)^{\boldsymbol{k}}. \tag{3.5}$$

If the approximation error is $\delta$ for any $\boldsymbol{x}_i \in C_t$ and $\boldsymbol{y}_j \in C_s$ and for all $C_t$ and $C_s$, then in the matrix representation the 2-norm error of $\boldsymbol{s}$ is upper bounded by $n\delta\|\boldsymbol{q}\|_2$.

Note that the double expansion (3.3) requires two truncation orders. Although not used in this paper, an expansion that leads to a single truncation order may be of interest:

$$\phi(\boldsymbol{x}_c + \Delta\boldsymbol{x}, \boldsymbol{y}_c + \Delta\boldsymbol{y}) = \sum_{\|\boldsymbol{j}+\boldsymbol{k}\|=0}^{\infty} \frac{\partial_{\boldsymbol{x}}^{\boldsymbol{j}} \partial_{\boldsymbol{y}}^{\boldsymbol{k}} \phi(\boldsymbol{x}_c, \boldsymbol{y}_c)}{(\boldsymbol{j}+\boldsymbol{k})!} (\Delta\boldsymbol{x})^{\boldsymbol{j}} (\Delta\boldsymbol{y})^{\boldsymbol{k}}. \tag{3.6}$$

This expansion simply treats the concatenation of $\boldsymbol{x}_c$ and $\boldsymbol{y}_c$ as a $(2d)$-dimensional argument and expands $\phi$ at this argument. Then, the truncation can occur at, say, $\|\boldsymbol{j}+\boldsymbol{k}\| = p_1 + p_2$. Comparing the truncation of (3.3) with that of (3.6), one sees that the former distinguishes the near and the far fields, whereas the latter does not. It is usually not worth to add a large order for the near field and a small order for the far field, if $\Delta\boldsymbol{x}$ is small and $\Delta\boldsymbol{y}$ is relatively large; thus, we do not consider (3.6) here.

From a computational perspective, we rearrange the terms in (3.5) in order that they are computed at different stages:

$$s_i(C_s) \approx \sum_{\|\boldsymbol{k}\|=0}^{p_2} \sum_{\|\boldsymbol{j}\|=0}^{p_1} \underbrace{\binom{\boldsymbol{j}+\boldsymbol{k}}{\boldsymbol{j}}}_{\text{binom. coef.}} \underbrace{\frac{\partial_{\boldsymbol{y}}^{\boldsymbol{j}+\boldsymbol{k}} \phi(\boldsymbol{x}_c, \boldsymbol{y}_c)}{(\boldsymbol{j}+\boldsymbol{k})!}}_{\text{Taylor coef.}} \underbrace{(\boldsymbol{x}_c - \boldsymbol{x}_i)^{\boldsymbol{j}}}_{\text{target momt.}} \underbrace{\left[ \sum_{\boldsymbol{y}_j \in C_s} q_j (\boldsymbol{y}_j - \boldsymbol{y}_c)^{\boldsymbol{k}} \right]}_{\text{weighted source momt.}}. \tag{3.7}$$

The general idea is to precompute the binomial coefficients and the Taylor coefficients, because they can be shared by different targets. Then, given the weights, the weighted source moments are computed. Thereafter, for each target $\boldsymbol{x}_i$, the target moments are computed and the partial sum $s_i(C_s)$ is accumulated to $s_i$. The details are presented in Section 6. First, however, it is necessary to consider how to obtain the Taylor coefficients.

**4. Taylor coefficients.** We use recursion to compute the Taylor coefficients. Key to the recurrence is the following property of $\mathsf{K}_u(R)$ for any $u > 0$ (see, e.g., [16, p. 294, Table V-1]):

$$\frac{d}{dR} R^u \mathsf{K}_u(R) = -R^u \mathsf{K}_{u-1}(R). \tag{4.1}$$

It relates the derivatives of the Matérn function by successively reducing the order $u$ when is it positive. When $u$ becomes negative, the relation $\mathsf{K}_u = \mathsf{K}_{-u}$ is useful.

**4.1. Recurrence formula.** Define $g_u(R) = R^u \mathsf{K}_u(R)$. We want to obtain a recurrence relation for

$$G_u^{\boldsymbol{k}}(c) := \frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}} g_u(cr)}{2^{u-1}\Gamma(u) \cdot \boldsymbol{k}!}, \quad u > 0 \tag{4.2a}$$

with respect to $\boldsymbol{k}$ and $u$. Then, when $c = \sqrt{2\nu}$, $u = \nu$, and $r$ is the elliptical distance between $\boldsymbol{x}_c$ and $\boldsymbol{y}_c$ as usual, we immediately have the Taylor coefficients $\partial_{\boldsymbol{y}}^{\boldsymbol{k}} \phi(\boldsymbol{x}_c, \boldsymbol{y}_c)/\boldsymbol{k}! = G_\nu^{\boldsymbol{k}}(\sqrt{2\nu})$. Because the derivation of the recurrence is based on (4.1), that is, $u$ is successively reduced, we need to complete the definition of $G_u^{\boldsymbol{k}}(c)$ for $u \le 0$. To this end, we define

$$G_u^{\boldsymbol{k}}(c) := \begin{cases} \dfrac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}} g_u(cr)}{z(cr) \cdot \boldsymbol{k}!}, & u = 0 \\ \dfrac{(cr)^{-2u} \cdot \partial_{\boldsymbol{y}}^{\boldsymbol{k}} g_u(cr)}{2^{-u-1}\Gamma(-u) \cdot \boldsymbol{k}!}, & u < 0 \end{cases} \tag{4.2b}$$

for $c, r > 0$. For now we note that $z$ is some function used for encouraging a stable numerical behavior of the recurrence (because clearly $\Gamma(0)$ is undefined); $z$ will be defined later in (4.11). The case $u < 0$ is so defined because for the initial condition $\boldsymbol{k} = \boldsymbol{0}$, we want that the values for a pair of positive and negative $u$'s are the same, that is, $G_u^{\boldsymbol{0}}(c) = G_{-u}^{\boldsymbol{0}}(c)$.

To proceed, we use a simplified notation and write $\partial_i$ to mean the partial derivative with respect to $y_i$. With (4.1), we have

$$\partial_i g_u(cr) = \frac{dg_u(cr)}{dr} \cdot \frac{\partial r}{\partial r_i} \cdot \frac{dr_i}{dy_i} = \frac{c^2}{\ell_i^2} \cdot r_i \cdot g_{u-1}(cr). \tag{4.3}$$

Then, by applying the Leibniz rule for higher derivatives, we have for $k_i > 1$,

$$\partial_i^{k_i} g_u(cr) = \frac{c^2}{\ell_i^2} \cdot \partial_i^{k_i - 1}(r_i \cdot g_{u-1}(cr)) = \frac{c^2}{\ell_i^2} \cdot \left[ r_i \partial_i^{k_i - 1} g_{u-1}(cr) - (k_i - 1)\partial_i^{k_i - 2} g_{u-1}(cr) \right]. \tag{4.4}$$

If we further differentiate the above formulas with respect to other components of $\boldsymbol{y}$ and then divide the results by $\boldsymbol{k}!$, (4.3) and (4.4) become

$$\frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}} g_u(cr)}{\boldsymbol{k}!} = \frac{c^2}{k_i}\frac{r_i}{\ell_i^2}\cdot\frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}-\boldsymbol{e}_i} g_{u-1}(cr)}{(\boldsymbol{k}-\boldsymbol{e}_i)!}, \qquad (k_i = 1) \qquad (4.5)$$

$$\frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}} g_u(cr)}{\boldsymbol{k}!} = \frac{c^2}{k_i}\frac{r_i}{\ell_i^2}\cdot\frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}-\boldsymbol{e}_i} g_{u-1}(cr)}{(\boldsymbol{k}-\boldsymbol{e}_i)!} - \frac{c^2}{k_i}\frac{1}{\ell_i^2}\cdot\frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}-2\boldsymbol{e}_i} g_{u-1}(cr)}{(\boldsymbol{k}-2\boldsymbol{e}_i)!}, \qquad (k_i > 1) \qquad (4.6)$$

respectively, where $\boldsymbol{e}_i$ means the integer vector whose $i$th entry is 1 and other entries are zero. By adopting the convention that $\partial_{\boldsymbol{y}}^{\boldsymbol{k}} g_u = 0$ if any of the components of $\boldsymbol{k}$ is negative, we can consolidate (4.5) to (4.6). Then, we rewrite (4.6) as

$$G_u^{\boldsymbol{k}}(c) = h(u)\left[\frac{c^2}{k_i}\frac{r_i}{\ell_i^2}G_{u-1}^{\boldsymbol{k}-\boldsymbol{e}_i}(c) - \frac{c^2}{k_i}\frac{1}{\ell_i^2}G_{u-1}^{\boldsymbol{k}-2\boldsymbol{e}_i}(c)\right] \qquad (4.7)$$

for some prefactor $h(u)$. This prefactor can be easily determined from the different cases of the definition of $G_u^{\boldsymbol{k}}(c)$ in (4.2):

$$h(u) = \begin{cases} \dfrac{1}{2(u-1)}, & u > 1 \\ z(cr), & u = 1 \\ \dfrac{(cr)^{2u-2}\Gamma(1-u)}{2^{2u-1}\Gamma(u)}, & 0 < u < 1 \\ \dfrac{1}{(cr)^2 z(cr)}, & u = 0 \\ -\dfrac{2u}{(cr)^2}, & u < 0. \end{cases} \qquad (4.8)$$

Note that the recurrence relation (4.7) is valid for any component $i$. For numerical stability, we perform a weighted averaging. Specifically, we multiply $k_i$ on both sides of (4.7), sum over all $i$, and divide the result by $\|\boldsymbol{k}\|$. Thus,

$$G_u^{\boldsymbol{k}}(c) = \frac{c^2 h(u)}{\|\boldsymbol{k}\|}\left[\sum_{i=1}^{d}\frac{r_i}{\ell_i^2}G_{u-1}^{\boldsymbol{k}-\boldsymbol{e}_i}(c) - \sum_{i=1}^{d}\frac{1}{\ell_i^2}G_{u-1}^{\boldsymbol{k}-2\boldsymbol{e}_i}(c)\right]. \qquad (4.9)$$

Since we adopt the convention that $G_u^{\boldsymbol{k}} = 0$ if any of the components of $\boldsymbol{k}$ is negative, (4.9) is valid for all nonnegative integer vectors $\boldsymbol{k}$ except $\boldsymbol{0}$.

**4.2. Initial condition.** When $\boldsymbol{k} = \boldsymbol{0}$, the definition (4.2) leads to

$$G_u^{\boldsymbol{0}}(c) = \begin{cases} \dfrac{(cr)^u \mathsf{K}_u(cr)}{2^{u-1}\Gamma(u)}, & u > 0 \\ \dfrac{\mathsf{K}_0(cr)}{z(cr)}, & u = 0 \\ \dfrac{(cr)^{-u}\mathsf{K}_{-u}(cr)}{2^{-u-1}\Gamma(-u)}, & u < 0. \end{cases} \qquad (4.10)$$

We examine it case by case. The cases $u > 0$ and $u < 0$ lead to the same value of $G_u^{\boldsymbol{0}}(c)$, for a pair of positive $u$ and negative $u$ that have the same absolute value.

When $c = \sqrt{2\nu}$, $G_u^{\mathbf{0}}(c)$ is equal to an evaluation of the kernel function $\phi(r)$. When $r$ is close to zero, $G_u^{\mathbf{0}}(c)$ is close to 1.

Now consider $u = 0$. When $r \to 0$, $\mathsf{K}_0(cr)$ tends to infinity. The function $z(cr)$ is thus used to scale $\mathsf{K}_0(cr)$ so that $G_u^{\mathbf{0}}(c)$ behaves better when it is close to the origin. Consider an approximation of $\mathsf{K}_0$ around the origin [1, (9.6.12) and (9.6.13)]:

$$\mathsf{K}_0(R) \approx -\gamma - \log\left(\frac{R}{2}\right) \qquad \text{when} \qquad R \approx 0,$$

where

$$\gamma = \int_1^\infty \left(\frac{1}{\lfloor x \rfloor} - \frac{1}{x}\right) dx \approx 0.577216$$

is the Euler–Mascheroni constant. Then, we define

$$z(R) := \begin{cases} -\gamma - \log\left(\frac{R}{2}\right), & 0 < R < R_0 \\ 1, & R \geq R_0. \end{cases} \tag{4.11}$$

Here, the cutoff $R_0 = 2e^{-\gamma-1} \approx 0.413$ is used to make $z$ continuous. With this definition, $G_u^{\mathbf{0}}(c)$ is not far from 1 when $cr < R_0$, and it is equal to $\mathsf{K}_0(cr)$ when $cr \geq R_0$.

**4.3. Summary.** Summarizing the above discussions, we compute the Taylor coefficients

$$\partial_{\boldsymbol{y}}^{\boldsymbol{k}} \phi(\boldsymbol{x}_c, \boldsymbol{y}_c)/\boldsymbol{k}! = G_\nu^{\boldsymbol{k}}(\sqrt{2\nu})$$

using the recurrence formula (4.9) with the initial condition (4.10). The recurrence is based on both $\boldsymbol{k}$ and $u$. We use two $d$-dimensional arrays $A$ and $B$ to store all the intermediate values of $G_u^{\boldsymbol{k}}(\sqrt{2\nu})$. Specifically, given an expansion order $p$, we define (in C/C++ style)

$$A[k_1]\cdots[k_d] = G_u^{\boldsymbol{k}}(\sqrt{2\nu}) \qquad \text{and} \qquad B[k_1]\cdots[k_d] = G_{u-1}^{\boldsymbol{k}}(\sqrt{2\nu}),$$

for $u = \nu - p, \ldots, \nu$ and $0 \leq \|\boldsymbol{k}\| \leq p$. All the values in $A$ are computed by using $B$; and after $A$ is filled, the values of $A$ are copied to $B$. This process is repeated $p+1$ times to increase $u$ until finally $u$ reaches $\nu$. The following subroutine TaylorCoefficients summarizes this procedure.

**subroutine** TaylorCoefficients($p$, $\boldsymbol{x}_c - \boldsymbol{y}_c$)
  // In the following, skip the term(s) whenever array index $< 0$
  Initialize $B$ with zeros
  **for** $j = 0, \ldots, p$ **do**
    $A[0]\cdots[0] = G_{\nu-p+j}^{\mathbf{0}}(\sqrt{2\nu})$ ▷ $\nu - p + j = u$
    **for** $\|\boldsymbol{k}\| = 1, \ldots, j$ **do** ▷ cf. (4.9)
      $A[k_1]\cdots[k_d] = 2\nu h(\nu - p + j)/\|\boldsymbol{k}\| \times$
        $[(r_1 \cdot B[k_1 - 1]\cdots[k_d] - B[k_1 - 2]\cdots[k_d])/\ell_1^2 + \cdots$
        $\cdots + (r_d \cdot B[k_1]\cdots[k_d - 1] - B[k_1]\cdots[k_d - 2])/\ell_d^2]$
    **end for**
    Copy the entries of $A$ to $B$
  **end for**

10: **return** $A$
11: **end subroutine**

The computational cost of TaylorCoefficients is $O(p^d)$ in storage and $O(p^{d+1})$ in time. To reduce memory usage, note that the array $A[k_1]\cdots[k_d]$ (and similarly $B$) uses only the entries with $0 \le k_1 + \cdots + k_d \le p$. The total number of such entries is $\binom{p+d}{d}$. Therefore, instead of using a full $d$-dimensional array of size $(p+1)^d$, one may consider using a 1-dimensional array $A'$ of size $\binom{p+d}{d}$, where the indexing of $A'$ is in accordance with the increasing order of $\|\boldsymbol{k}\|$, that is,

$$\begin{gathered}
A'[0] = A[0]\cdots[0][0], \\
A'[1] = A[0]\cdots[0][1],\ \ldots,\ A'[d] = A[1]\cdots[0][0], \\
A'[d+1] = A[0]\cdots[0][2],\ A'[d+2] = A[0]\cdots[1][1],\ \ldots,\ A'[\tbinom{2+d}{d} - 1] = A[2]\cdots[0][0], \\
\vdots \\
A'[\tbinom{p-1+d}{d}] = A[0]\cdots[0][p],\ \ldots,\ A'[\tbinom{p+d}{d} - 1] = A[p]\cdots[0][0].
\end{gathered}$$

This design comes from a practical consideration, although it does not change the asymptotic storage cost. For some medium-sized $p$ and small $d$ (say, $p = 10$ and $d = 3$), the storage of $A'$ is only about $1/d!$ of that of $A$. When many sets of the Taylor coefficients (for different $\boldsymbol{x}_c - \boldsymbol{y}_c$) are stored, this will be a significant saving.

**4.4. Discussion of numerical behavior.** The initial condition of the recurrence relation is well behaved. When $r$ is close to zero, the value of $G_u^{\mathbf{0}}$ is close to 1 for any $u$; and when $r$ is not abnormally large, $G_u^{\mathbf{0}}$ will not be exceedingly small. In fact, the term "abnormally large" is vague, and we delay its clarification until Section 5 when error control is the concern.

The recurrence depends on the function $h(u)$ defined in (4.8). It has discontinuities at $u = 0$ and $u = 1$, and when approaching these discontinuities the function value grows without bound. Thus, the recurrence may be numerically unstable when the smoothness parameter $\nu$ is close to, but not exactly, an integer. In many applications, it is not necessary to take a smoothness that is too close to an integer (otherwise replacing it by the closest integer may be a practical workaround). Moreover, we demonstrate in Section 7.4 that the recurrence still works in practice for a $\nu$ differing from 1 by only $10^{-5}$.

**5. Error estimate.** It is important to characterize the error, denoted by $\delta$, between the actual value of the kernel and that of its Taylor approximation. Given a pair of expansion orders $(p_1, p_2)$, the factors that affect $\delta$ are the elliptical distance $\tau$ between a pair of centroids and the elliptical radii $\rho$ of the clusters (also called the *expansion radius*). To distinguish sources and targets, we use $\rho_t$ for a target cluster and $\rho_s$ for a source cluster (see Figure 5.1). It is desirable to bound $\delta$ based on $\rho_t$, $\rho_s$, and $\tau$.

Many kernels enjoy such a bound that is analytically derived. When only a single expansion around the source centroid is used, there is only one expansion order $p$. The following lists a few bounds for several kernels (see (2.6), (4.9), and (5.17) of [6]):

$$A_1\rho\left(\frac{1}{2.12...}\right)^p, \quad \frac{A_2}{p+1}\frac{\tau}{\tau-\rho}\left(\frac{\rho}{\tau}\right)^{p+1}, \quad \frac{A_3}{\tau-\rho}\left(\frac{\rho}{\tau}\right)^{p+1},$$

where $A_1$, $A_2$ and $A_3$ are all positive constants. These bounds correspond to a 1-dimensional multiquadric expanded with Laurent series when $\tau \ge 3\rho$, a complex

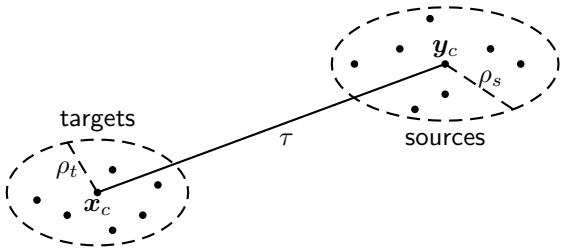



FIG. 5.1. *Expansion radii $\rho_t$, $\rho_s$ and distance $\tau$ (all elliptical).*

logarithm expanded with Laurent series when $\rho < \tau$, and a 3-dimensional reciprocal function expanded with spherical harmonics when $\rho < \tau$, respectively.

An error bound for the Matérn kernel is unknown. We therefore seek a different approach. Motivated by the above examples, we hypothesize that for a single expansion with order-$p$ truncation, the maximum of the errors, denoted by $\delta_p^{\max}$, for all points within an expansion radius $\rho$ can be expressed as

$$\log_{10} \delta_p^{\max}(\rho, \tau) = \alpha_1 + \alpha_2 \log_{10} \tau + \alpha_3 \log_{10}(\rho/\tau), \tag{H1}$$

where $\alpha_1$, $\alpha_2$, and $\alpha_3$ are coefficients to be determined. The quantity $\delta_p^{\max}$ is a function of $\rho$ and $\tau$, given $p$. The term $\rho/\tau$ is the *expansion ratio*, and it is always less than 1. Then, for a double expansion with truncation orders $(p_1, p_2)$, we further hypothesize that the maximum of errors for all pairs $\boldsymbol{x}_i \in C_t$ and $\boldsymbol{y}_j \in C_s$ is computed as

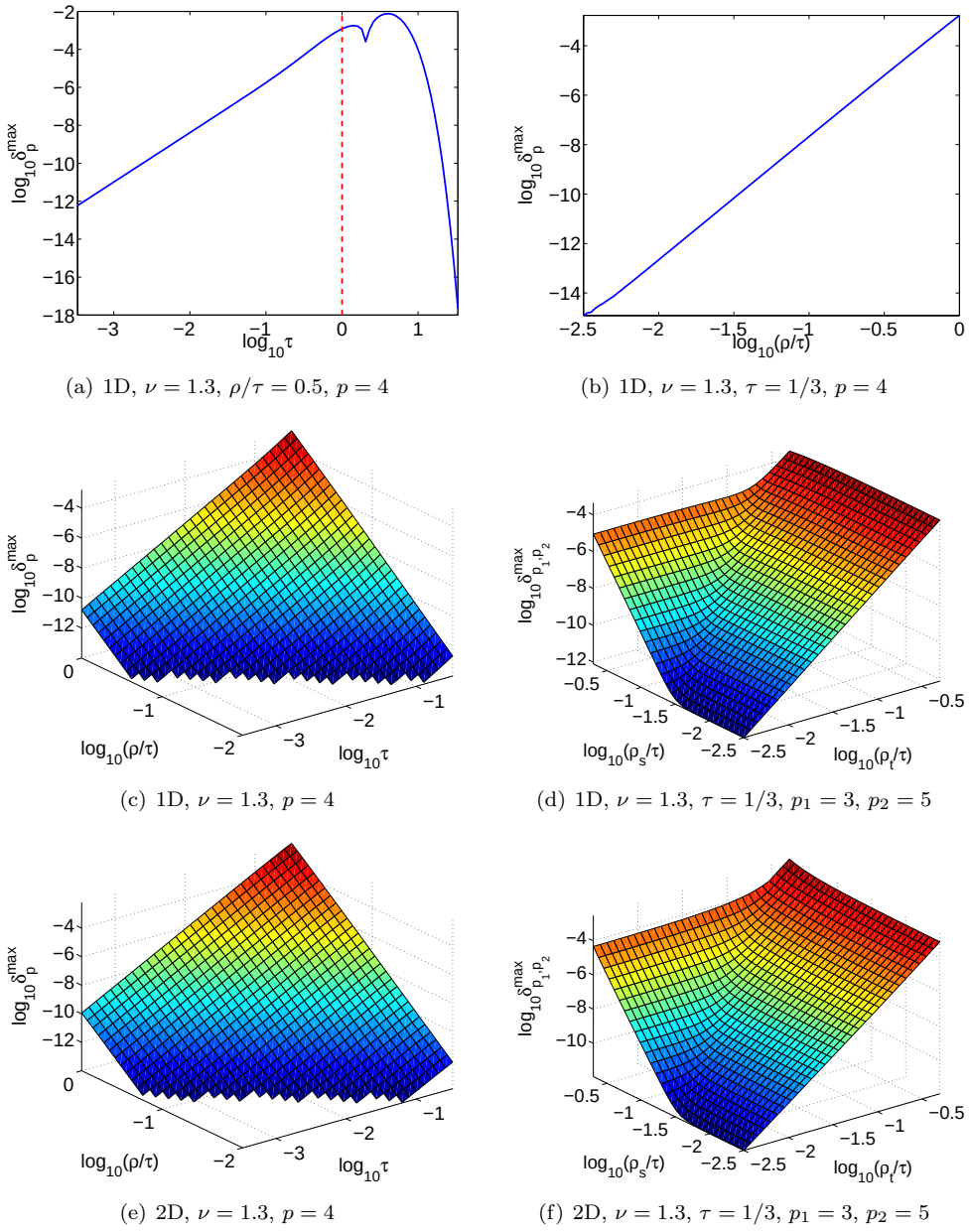
$$\delta_{p_1,p_2}^{\max}(\rho_t, \rho_s, \tau) = \max\{\delta_{p_1}^{\max}(\rho_t, \tau + \rho_s),\ \delta_{p_2}^{\max}(\rho_s, \tau + \rho_t)\}, \tag{H2}$$

where $\delta_{p_1,p_2}^{\max}$ is a function of $\rho_t$, $\rho_s$, and $\tau$, given $(p_1, p_2)$. We do not hypothesize the error as a function of the expansion orders, the reason for which will be clear soon.

**5.1. Rationale.** The rationale of the hypotheses (H1) and (H2) is supported by a series of observations. We begin with the case of one dimension. Figure 5.2(a) plots the variation of $\delta_p^{\max}$ with respect to $\tau$ in the log-log scale, by fixing $\rho/\tau$. One sees that when $\tau \leq 1$, the plot is almost a straight line. Plot (b) also shows a straight-line pattern when we consider the variation of $\delta_p^{\max}$ with respect to $\rho/\tau$, by fixing $\tau$. Varying $\tau$ and $\rho/\tau$ simultaneously results in a plane pattern as shown in plot (c). This indicates that (H1) is highly plausible.

When one considers a double expansion, the hypothesis (H2) states that $\delta_{p_1,p_2}^{\max}$ is an overall effect of two single expansions: $\delta_{p_1}^{\max}(\rho_t, \tau + \rho_s)$, which results from an expansion around the target centroid $\boldsymbol{x}_c$ by assuming that the source $\boldsymbol{y}_c$ is as far as possible (having a distance $\tau + \rho_s$ from $\boldsymbol{x}_c$), and $\delta_{p_2}^{\max}(\rho_s, \tau + \rho_t)$, which results from a similar expansion around the source centroid. We show in plot (d) the change of $\delta_{p_1,p_2}^{\max}$ by fixing $\tau$ but varying $\rho_s$ and $\rho_t$. One sees a surface that looks like the superposition of two planes, which is the reason we hypothesize a maximum of two terms in (H2).

The kernel may behave differently when moving to higher dimensions. To show that (H1) and (H2) are reasonable, we also show the 2D case in plots (e) and (f). They look similar to (c) and (d), respectively. In particular, we see a plane pattern in (e) and a two-plane pattern in (f). The plots for 3D are similar and we do not repeat them here.




(a) 1D, $\nu = 1.3$, $\rho/\tau = 0.5$, $p = 4$

(b) 1D, $\nu = 1.3$, $\tau = 1/3$, $p = 4$

(c) 1D, $\nu = 1.3$, $p = 4$

(d) 1D, $\nu = 1.3$, $\tau = 1/3$, $p_1 = 3$, $p_2 = 5$

(e) 2D, $\nu = 1.3$, $p = 4$

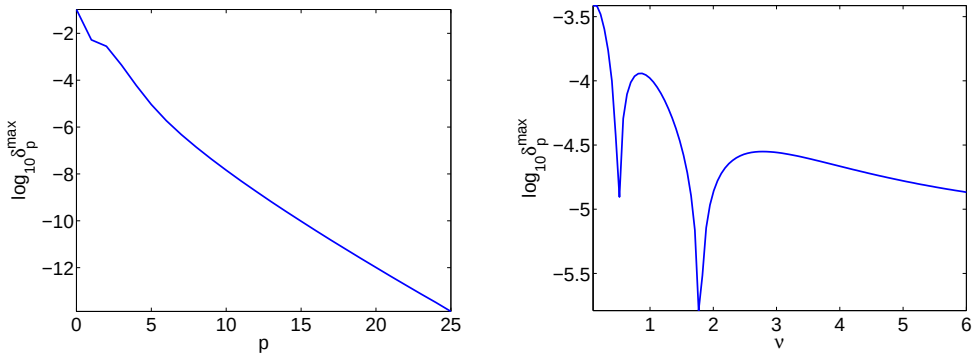
(f) 2D, $\nu = 1.3$, $\tau = 1/3$, $p_1 = 3$, $p_2 = 5$

FIG. 5.2. *Taylor approximation error with respect to $\tau$ and $\rho/\tau$.*

Note that the hypotheses (H1) and (H2) preclude the situation $\tau > 1$. In plot (a), one sees that $\delta_p^{\max}$ behaves completely differently in this situation. We do not know how to characterize this behavior. On the other hand, the kernel used in practice often entails large scaling parameters, making the case $\tau \leq 1$ predominant. Hence, we do not consider the case $\tau > 1$ further in this paper.

One may also be interested in the variation of $\delta_p^{\max}$ with respect to $p$ and $\nu$. Figure 5.3 plots the variations. Plot (a) shows a visually straight-line pattern, which

in fact is not close to straight according to fitting. It is unknown what expression can be used to fit plot (b).




(a) 1D, $\nu = 1.3$, $\tau = 1/3$, $\rho/\tau = 0.5$ (b) 1D, $\tau = 1/3$, $\rho/\tau = 0.5$, $p = 4$

FIG. 5.3. *Taylor approximation error with respect to $p$ and $\nu$.*

**5.2. Subroutine.** For clarity, we provide the subroutine ErrorControlInfo for fitting the coefficients in (H1) and testing the hypotheses (H1) and (H2). First, it chooses a few $\tau$ and $\rho/\tau$, equally spaced in the log-scale from $10^{-2.5}$ to $10^{-0.5}$. For each pair of $\tau$ and $\rho/\tau$, it estimates $\delta_{p_2}^{\max}(\rho, \tau)$, which is clearly defined as

$$\max_{\substack{\|\boldsymbol{x}_c - \boldsymbol{y}_c\|_{\boldsymbol{\ell}} \le \tau \\ \|\Delta \boldsymbol{y}\|_{\boldsymbol{\ell}} \le \rho}} \left| \phi(\boldsymbol{x}_c, \boldsymbol{y}_c + \Delta \boldsymbol{y}) - \sum_{\|\boldsymbol{k}\|=0}^{p_2} \frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}} \phi(\boldsymbol{x}_c, \boldsymbol{y}_c)}{\boldsymbol{k}!} (\Delta \boldsymbol{y})^{\boldsymbol{k}} \right|,$$

where $\|\cdot\|_{\boldsymbol{\ell}}$ denotes the scaled 2-norm with scaling $\boldsymbol{\ell}$ (cf. (2.2)). Of course, we cannot exhaust all such vectors $\boldsymbol{x}_c - \boldsymbol{y}_c$ and $\Delta \boldsymbol{y}$ to determine $\delta_{p_2}^{\max}(\rho, \tau)$; thus, a sampling is used. Note that even though we use the notation $\boldsymbol{x}_c - \boldsymbol{y}_c$ and $\Delta \boldsymbol{y}$, the sampling is independent of the point set; in fact, we only need to sample the $d$-sphere with elliptical radius $\tau$ to obtain several $\boldsymbol{x}_c - \boldsymbol{y}_c$'s and similarly several $\Delta \boldsymbol{y}$'s. With $\delta_{p_2}^{\max}(\rho, \tau)$ estimated for several pairs of $\rho$ and $\tau$, the coefficients $\alpha_1$, $\alpha_2$, and $\alpha_3$ in (H1) are fitted by using the straightforward linear regression. Note that because of machine precision, some computed $\delta_{p_2}^{\max}$'s are not useful for regression if they are less than, say, `1e-14`, to be safe. We say that (H1) is valid if the absolute difference between both sides is not larger than 1, which means that the estimated error departs from the true error by at most one digit.

Similarly, $\delta_{p_1}^{\max}$ is estimated; and another set of coefficients $\alpha_1$, $\alpha_2$, and $\alpha_3$ is produced. With these coefficients, the three terms in (H2), $\delta_{p_1,p_2}^{\max}$, $\delta_{p_1}^{\max}$, and $\delta_{p_2}^{\max}$, are estimated by performing another sampling. This time, $\boldsymbol{x}_c - \boldsymbol{y}_c$, $\Delta \boldsymbol{x}$, and $\Delta \boldsymbol{y}$ are sampled given scaled 2-norms $\tau$, $\rho_t$, and $\rho_s$, respectively. Then, (H2) is tested. Similar to the case of (H1), the validity of (H2) is determined by whether the absolute difference between both sides is no larger than 1.

If either (H1) or (H2) is invalid, the error control scheme developed here is not successful and the straightforward summation has to be used (which never happened in our experience). Otherwise, the two sets of coefficients are returned. Later, (H1) and (H2) with the estimated $\alpha$'s are used to determine whether Taylor expansion can be used in the summation process.

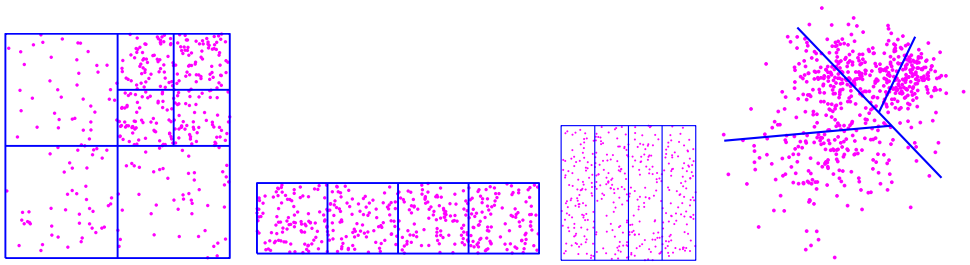
1: **subroutine** ErrorControlInfo
2: Define $S_1$ and $S_2$, each one containing 10 numbers equally spaced in $[-2.5, -0.5]$.
// Test of (H1)
3: **for all** $\tau$ and $\rho$ where $\log_{10}\tau \in S_1$ and $\log_{10}(\rho/\tau) \in S_2$ **do**
4: Estimate $\delta_{p_2}^{\max}(\rho, \tau)$ using a sampling approach.
5: **end for**
6: Use $\rho$, $\tau$, and $\delta_{p_2}^{\max}$ to regress (H1). Obtain $\alpha_1$, $\alpha_2$, and $\alpha_3$.
7: If difference between both sides of (H1) is larger than 1, **return error**
8: Repeat lines 3 to 7 by changing $p_2$ to $p_1$ and obtain another set of $\alpha_1$, $\alpha_2$, $\alpha_3$.
// Test of (H2)
9: **for all** $\tau$, $\rho_t$, $\rho_s$ where $\log_{10}\tau \in S_1$, $\log_{10}(\rho_t/\tau) \in S_2$ and $\log_{10}(\rho_s/\tau) \in S_2$ **do**
10: Estimate $\delta_{p_1,p_2}^{\max}(\rho_t, \rho_s, \tau)$, $\delta_{p_1}^{\max}(\rho_t, \tau + \rho_s)$, and $\delta_{p_2}^{\max}(\rho_s, \tau + \rho_t)$.
11: **end for**
12: Use $\delta_{p_1,p_2}^{\max}$, $\delta_{p_1}^{\max}$, and $\delta_{p_2}^{\max}$ to verify (H2) for all $\rho_t$, $\rho_s$, and $\tau$.
13: If difference between both sides of (H2) is larger than 1, **return error**
14: **return** two sets of $\alpha_1$, $\alpha_2$, $\alpha_3$
15: **end subroutine**

**6. Tree code.** A general tree code algorithm is as follows. First, the whole set of points is recursively partitioned to form a tree structure, where each tree node represents a cluster of points. Each leaf node then contains a set of targets. We consider a target $\boldsymbol{x}_i$ that belongs to some $C_t$, and we initialize $s_i = 0$. A top-down tree-walk in the pre-order style is performed, starting from the root. Any tree node being visited contains a set $C_s$ of sources. Given a tolerance $\epsilon$, if the truncated Taylor expansion at the centroids of $C_s$ and $C_t$ yields an approximation error that is less than $\epsilon$, then the partial sum $s_i(C_s)$ is computed by using (3.7) and is accumulated to $s_i$. Otherwise, the children of the current node are visited. This procedure is performed recursively until a leaf node is reached. At the leaf node, direct summation is performed between $\boldsymbol{x}_i$ and all points in $C_s$. The result is also accumulated to $s_i$. The computation of $s_i$ is complete when all recursive branches of the treewalk terminate.

**6.1. Partitioning of the point set.** Usual partitioning methods result in a quadtree (or an octree in $\mathbb{R}^3$) or a k-d tree structure. The former assumes a square computational domain and recursively partitions subdomains with dense points into equal areas (see Figure 6.1(a)); The latter method (in fact, a common variant) cyclically partitions the point set with respect to each coordinate axis by identifying the mid point; thus, a k-d tree is a binary tree where the two sibling nodes in the same level contain point clusters of the same size.

However, the configuration of the point set is often more complicated than what a quadtree or a k-d tree can efficiently handle. Consider, for example, Figure 6.1(b), where the points are uniform but one side of the rectangular domain is four times as long as the other side. It is more natural to first partition with respect to the longer side into four equal parts, because in this manner the diameter of the resulting clusters is much smaller.

This partitioning is also advantageous even when the domain is square but the distance is defined anisotropically (see Figure 6.1(c)). For example, if the scale parameter for the Matérn kernel is 1 along one dimension and 1/4 along the other, then first partitioning with respect to the latter dimension into four equal parts can yield clusters of smaller elliptical diameters.



(a) Square (b) Rectangle (c) Anisotropy (d) Nonuniform density

FIG. 6.1. *Different point set configurations and partitioning schemes.*

We thus seek a principled method to incorporate the uneven density of the points and the anisotropy. The idea is to use principal component analysis to partition the space so that the elliptical diameters of the resulting clusters are as small as possible; see Figure 6.1(d). The principal direction, along which the variance of the points is maximal, is simply the dominant eigenvector of the covariance matrix of the point set. We thus use a hyperplane with this direction as the normal to separate the point set in two equal-sized parts. Similar ideas have been suggested for the construction of hierarchical matrices (see, e.g., [7]), and our implementation follows that of [12], wherein a cluster is always partitioned in two equal parts. We note that in order to incorporate the anisotropy in the kernel, it is crucial to first prescale each coordinate by the corresponding scale parameter $\ell_i$ and to perform all the principal component calculations on the scaled points.

Principal component partitioning has several advantages. First, the clusters are more compactly grouped by considering the shape and the density of the point set and the scaling parameters. Second, the resulting hierarchy tree is a binary tree, independent of the spatial dimension $d$. Furthermore, the tree is a *full* binary tree, and each leaf node contains almost the same number of points (differing by at most 1). This provides a natural and convenient way to distribute the points in parallel processing. Third, if $n_0$ is the maximum size of a leaf, the computational cost of the overall subdivision is $O(n \log_2(n/n_0))$, which is asymptotically the same as that of the usual quadtree or k-d tree partitioning, even though the computation with eigenvectors may seem slightly complicated.

Of course, the nonregular subdomains (in fact, convex polytopes) resulting from principal component partitioning sacrifice some advantages the usual partitioning methods provide. For example, the directions $\boldsymbol{x}_c - \boldsymbol{y}_c$ are unlikely to be the same for two different pairs of clusters, as oppose to the case of a quadtree where the number of distinct directions are smaller (thus certain accelerations in computations can be achieved). Nevertheless, principal component partitioning is a general scheme. It enables a tighter error estimate because of the more compactly grouped clusters and it implicitly handles anisotropy.

**6.2. Complete algorithm and complexity analysis.** With partitioning and error estimates, the complete procedure is given in Algorithm 1. This algorithm comprises a planning phase and an evaluation phase, whereby all the calculations independent of the weights $\{q_j\}$ are computed only once in the first phase.

Let us consider the computational complexity of the algorithm with respect to $n$ and $n_0$. The partitioning step scales as $O(n \log_2(n/n_0))$ because each tree level

**Algorithm 1** Tree code method for computing (3.1)

**Require:** Parameters: expansion order $(p_1, p_2)$, tolerance $\epsilon$, maximum leaf size $n_0$

// Planning phase

1: Call ErrorControlInfo to fit error formula (H1). If fitting is unsuccessful, exist this algorithm and use straightforward summation to compute (3.1).
2: Recursively partition the point set to form a full binary tree hierarchy.
3: Compute a binomial table to store $\binom{k}{j}$ for all $k \le p_1 + p_2$ and $j \le p_1$.
4: **for** every leaf of the tree **do**
5: Perform a treewalk starting from the root. If a certain source–target node pair admits Taylor expansion, compute and store the Taylor coefficients.
6: **end for**

// Evaluation phase

7: Initialize $s_i = 0$ for all $i$
8: **for** every leaf of the tree **do**
9: Perform a treewalk. If a certain source–target node pair admits Taylor expansion, compute target moment and weighted source moment to obtain $s_i(C_s)$ via Taylor approximation, and accumulate it to $s_i$. Otherwise, if the target node is a leaf, compute $s_i(C_s)$ via direct summation and accumulate it to $s_i$.
10: **end for**
11: **return** $\{s_i\}$

requires an $O(n)$ cost and there are $O(\log_2(n/n_0))$ levels. The construction of the binomial table and the fitting of the error formulas are independent of $n$ and $n_0$, and so are the determination of whether to use Taylor expansions in later treewalks. Let $m_{\text{expand}}$ and $m_{\text{direct}}$ denote the number of node pairs where Taylor expansion occurs and where direct summation occurs, respectively. The cost of the treewalk in the planning phase is $O(m_{\text{expand}} + m_{\text{direct}})$. In the treewalk of the evaluation phase, the total work to compute the partial sums is $O(n_0 \cdot m_{\text{expand}})$, whereas the work to perform the direct summations is $O(n_0^2 \cdot m_{\text{direct}})$. Therefore, the computational complexities of the two phases are

$$\begin{aligned} \text{planning: } & O(n \log_2(n/n_0) + m_{\text{expand}} + m_{\text{direct}}), \\ \text{evaluation: } & O(n_0 \cdot m_{\text{expand}} + n_0^2 \cdot m_{\text{direct}}). \end{aligned}$$

A difficulty in further simplifying the above big-O expressions is that $m_{\text{expand}}$ and $m_{\text{direct}}$ vary with not only $n$ but also the configuration of the point set. In a simplified setting, for each point $\boldsymbol{x}_i$, if one Taylor approximation occurs in each intermediate tree level and one direct summation happens at the leaf that contains $\boldsymbol{x}_i$, then

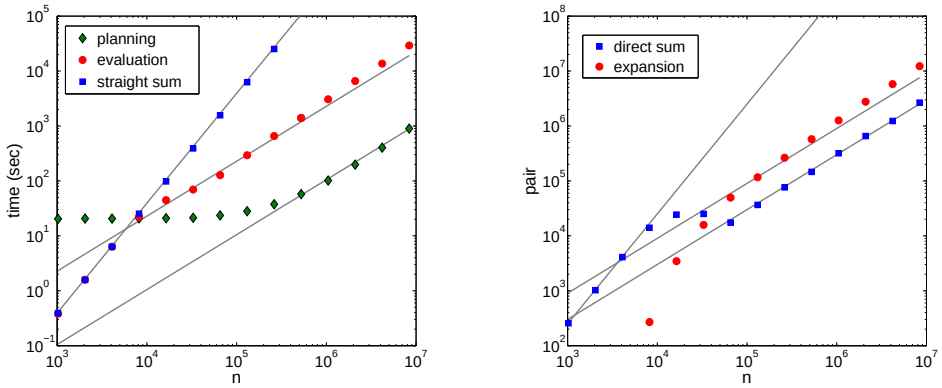
$$m_{\text{expand}} = (n/n_0)(\lceil \log_2(n/n_0) \rceil - 1) \quad \text{and} \quad m_{\text{direct}} = n/n_0. \tag{6.1}$$

In general, estimating $m_{\text{expand}}$ and $m_{\text{direct}}$ is difficult, but empirical results usually approximately agree with (6.1); see Figure 7.1(b). This leads to the overall $O(n \log n)$ scaling, if $n_0$ is ignored.

**7. Numerical results.** In this section we present several experimental results to demonstrate the numerical behavior and the performance of Algorithm 1. For the purpose of practicality and code reuse, we implemented a parallel version of the program in C++; the parallelization is discussed in a separate paper [13]. In this paper, all the experiments were serial, conducted on a Pentium Xeon core of clock rate

2.6 GHz with 36 GB of memory. This setting is particularly helpful for demonstrating the computational complexity of the serial algorithm. The evaluation of $\mathsf{K}_u$ used the `AMOS` package available from `http://www.netlib.org/amos/`; see also [2]. The first few subsections here concern the computational cost and the tree code parameters; thus the kernel was fixed at $\nu = 1.5$ and $\boldsymbol{\ell} = [4, 14, 3]$, and the set of points was uniformly distributed in the unit cube. The final subsection, on the other hand, shows experiments on a variety of kernel parameters and point set distributions. In all experiments, the weights were uniformly random numbers in the interval $[0, 1]$.

**7.1. Computational complexity.** We first show the $O(n \log n)$ scaling of the algorithm, which is one of the crucial factors when considering the usefulness of a tree code. We fix the expansion orders $p_1 = 3$, $p_2 = 5$, the tolerance $\epsilon = 10^{-6}$, and the leaf size $n_0 = 64$, and we vary $n$ from approximately 1,000 to 8 million.




(a) Planning and evaluation time of tree code and running time of straightforward summation.

(b) Expansion node pairs $m_{\text{expand}}$ and direct summation node pairs $m_{\text{direct}}$.

FIG. 7.1. *Computational cost with respect to $n$. The gray lines indicate $O(n)$ and $O(n^2)$.*

Figure 7.1(a) plots the running time of the tree code algorithm (separated in the planning phase and the evaluation phase) and that of the straightforward summation. The gray lines indicate $O(n)$ and $O(n^2)$ scalings. Clearly, when $n$ is sufficiently large, the growths of the planning time and the evaluation time are close to linear, whereas the growth of the straightforward summation time is strictly quadratic.

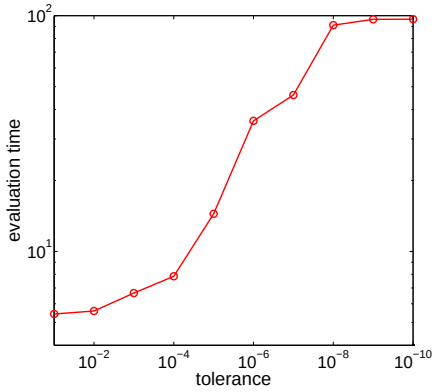
In this particular setting, the tree code outperforms straightforward summation starting at approximately 8,000 points. Before this, the points are too sparse and no Taylor approximation occurs, probably because the choice of the leaf size yields spatially large clusters even for the leaves. Plot (b) confirms this observation. When $n$ is less than 8,000, the number $m_{\text{expand}}$ is zero, and thus all the calculations are direct summations at the leaf level. Reading again plot (a), one finds that the red dotted markers (evaluation time) for $n < 2^{13}$ overlap with the blue squared markers (straightforward summation time). When $n$ is larger than $2^{16}$, both $m_{\text{expand}}$ and $m_{\text{direct}}$ grow linearly.

Note, also, that when $n < 2^{17}$ (approximately 130,000), a nontrivial planning time is spent on the tree code. This cost is attributed to the testing of hypotheses (H1) and (H2) and the fitting of the formulas therein. This "setup" cost is not small, especially for high-dimensional points, because a sufficient sampling is needed to estimate the approximation error well.

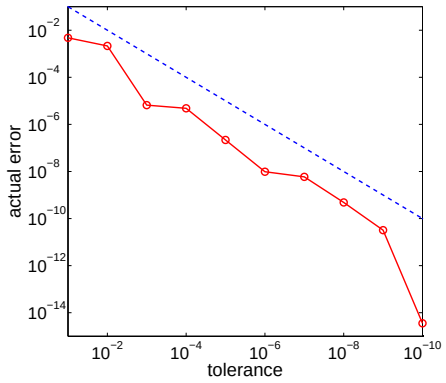
**7.2. Choice of expansion orders.** One characteristic of the proposed algorithm is the need to specify a pair $(p_1, p_2)$ of Taylor orders in order to approximate the summation to a desired accuracy. For $n = 16,384$ and $n_0 = 64$, the table in Figure 7.2(a) shows an optimal choice of $(p_1, p_2)$ for $\epsilon$ ranging from $10^{-1}$ to $10^{-10}$. The criterion of the optimality is the fastest evaluation time, by using an exhaustive search. Since the surface of evaluation time with respect to $p_1$ and $p_2$ has a convex shape, optimality can be located. One sees that as $\epsilon$ becomes smaller, larger orders are preferred, and usually $p_2$ is no smaller than $p_1$. The corresponding evaluation times, plotted in (b), show an increasing trend, although it is unclear how this trend can be described as a simple formula of $\epsilon$.

(a) Optimal expansion orders $p_1, p_2$ (in terms of evaluation time) given $\epsilon$.

| $\log_{10}\epsilon$ | $-1$ | $-2$ | $-3$ | $-4$ | $-5$ | $-6$ | $-7$ | $-8$ | $-9$ | $-10$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $p_1$ | 0 | 1 | 2 | 2 | 3 | 4 | 4 | 4 | 5 | 2 |
| $p_2$ | 0 | 1 | 2 | 3 | 4 | 5 | 8 | 10 | 8 | 0 |




(b) Evaluation time versus $\epsilon$.




(c) Actual error versus $\epsilon$.

FIG. 7.2. *Choice of expansion orders by varying tolerance* $\epsilon$ ($n = 16,384$).

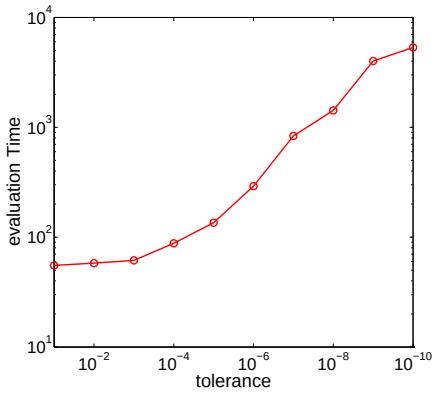
The actual approximation error, defined as the relative 2-norm error of the matrix-vector product, versus the tolerance $\epsilon$ is shown in plot (c). Following the discussions in page 5 we see that this error is bounded by $n\epsilon$, but the bound is too pessimistic. Plot (c) shows that the error is less than $\epsilon$ (the red circular markers are all under the blue dashed reference line). This plot also shows that the error is often one to two orders of magnitude smaller than $\epsilon$, which indicates that $\epsilon$ can be used as a useful reference when one is interested in computing the summation to a particular accuracy.

One may notice that when $\epsilon = 10^{-10}$, not only do the values of $p_1$ and $p_2$ deviate from the general trend, but also is the relative error far away from $\epsilon$. Under this particular choice, no Taylor approximations are used. In fact, examining the experiment logs of all the $p$ pairs, we find that approximations are never used whenever $p_1$ or $p_2$ is less than 6. The choice $p_1 = 2$ and $p_2 = 0$ tops over these cases merely because of noise in the computing environment. When both $p_1$ and $p_2$ are larger than or equal to 6, some approximations are used, but the gain in approximations does not compete with direct summations. This is probably because the number of Taylor coefficients explodes but $m_{\text{expand}}$ is not large enough.

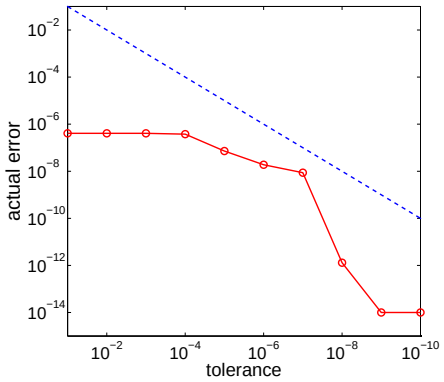
We also perform experiments on a larger $n$ $(131,072)$ and report the results in

(a) Optimal expansion orders $p_1, p_2$ (in terms of evaluation time) given $\epsilon$.

| $\log_{10}\epsilon$ | $-1$ | $-2$ | $-3$ | $-4$ | $-5$ | $-6$ | $-7$ | $-8$ | $-9$ | $-10$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $p_1$ | 0 | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 5 | 5 |
| $p_2$ | 1 | 1 | 2 | 2 | 3 | 5 | 6 | 7 | 7 | 7 |




(b) Evaluation time versus $\epsilon$.




(c) Actual error versus $\epsilon$.

FIG. 7.3. *Choice of expansion orders by varying tolerance* $\epsilon$ ($n = 131,072$).

Figure 7.3. In this case, one sees a nice trend of the optimal expansion orders for all $\epsilon$. Moreover, the orders are generally smaller than those in Figure 7.2. The reason is that the points are more densely populated and hence the spatial size of the leaves is smaller. Thus, in order to achieve the same accuracy, the expansion order for source nodes on the same level of the hierarchy tree need not be as large as that in the smaller $n$ case. One sees from plot (c), however, that the actual error is far away from $\epsilon$, meaning that perhaps using even smaller orders is sufficient to achieve the tolerance, although the computational time increases.

**7.3. Choice of leaf size.** A subtle issue in a general tree code is the choice of the leaf size. A size $n_0 > 1$ is often used in order to ensure that the number of tree levels is not too large and that the source clusters are not too small. Because we use double expansion, an additional reason for setting $n_0 > 1$ is to control the size of the target clusters. We investigate the optimal leaf size (again, in terms of evaluation time) as $n$ varies. The expansion orders are fixed at $p_1 = 3$ and $p_2 = 5$ and the tolerance $\epsilon = 10^{-6}$.

Figure 7.4 plots the evaluation times for several choices of the leaf size $n_0$. The solid blue squared markers indicate the fastest time, one for each $n$. These markers are highlighted within a yellow band. In general, the curves show a decreasing–increasing–stabilized trend, from small $n_0$ to large $n_0$. The decreasing–increasing trend provides the opportunity to use an $n_0$ that is larger than 1 to reduce the computational time. The stabilized part is caused by the fact that when $n_0$ is sufficiently large, no approximation will be accurate enough so that eventually direct summations are used everywhere. Figure 7.4 seems to suggest that as $n$ becomes larger, $n_0$ should also increase accordingly, although its increase is much slower than that of $n$.

**7.4. Tests of different kernel parameters and point distributions.** To demonstrate that the algorithm is able to handle different kernel parameters and

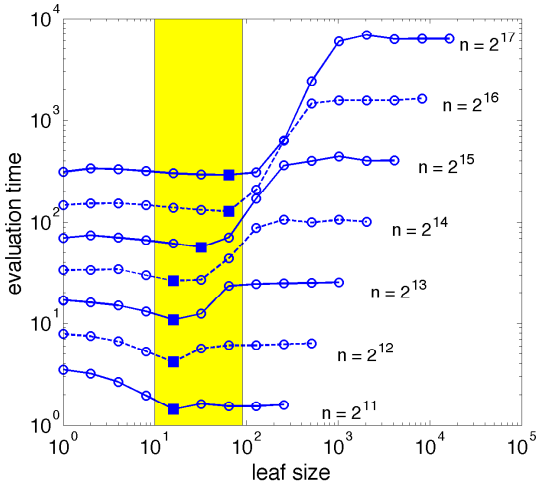



FIG. 7.4. *Evaluation time versus leaf size $n_0$ for different $n$.*

point distributions of practical interests, we show the results of some test cases in Table 7.1. These tests are all performed on $n = 131,072$ points with $p_1 = 3$, $p_2 = 5$, $\epsilon =$ `1e-6` and $n_0 = 64$. In rows 1 to 4 of the table, we arbitrarily vary $\nu$ and $\boldsymbol{\ell}$. In rows 5 to 7, the points are sampled on a sphere of unit radius, with uniformly random azimuthal angle and polar angle. In rows 8 to 10, the points are sampled from the 30°N to 60°N of this sphere. Such a portion of the sphere is called a "spherical segment," and it simulates a band of latitudes of the globe. In rows 11 to 15, we make $\nu$ to be progressively closer to 1, in order to test the numerical viability of the algorithm in handling $\nu$ that is close to an integer.

TABLE 7.1

*Tests by varying kernel parameters and point distributions. "Plan." means planning time, "eval." means evaluation time, "straight." means straightforward summation time. All times are in seconds.*

| | $\nu$ | $\boldsymbol{\ell}$ | Distri. | Plan. | Eval. | Straight. | Err. $\delta$ |
|---|---|---|---|---|---|---|---|
| 1: | 1.5 | $40, 14, 30$ | cube | 25.31 | 205.79 | 6602.0 | `5.40e-09` |
| 2: | 1.75 | $40, 14, 30$ | cube | 32.25 | 305.40 | 18733.0 | `1.19e-09` |
| 3: | 2 | $20, 30, 130$ | cube | 26.74 | 200.76 | 9642.2 | `9.59e-11` |
| 4: | 2.25 | $4, 14, 30$ | cube | 32.02 | 351.72 | 24435.5 | `9.13e-09` |
| 5: | 1.25 | $40, 14, 30$ | sphere | 41.44 | 289.14 | 18019.7 | `3.41e-09` |
| 6: | 1 | $20, 30, 130$ | sphere | 30.48 | 301.79 | 10035.3 | `1.88e-08` |
| 7: | 0.75 | $20, 30, 130$ | sphere | 41.96 | 583.95 | 18001.9 | `1.69e-08` |
| 8: | 1.25 | $40, 14, 30$ | sph. seg. | 37.69 | 176.05 | 17617.6 | `4.77e-09` |
| 9: | 1 | $20, 30, 130$ | sph. seg. | 28.51 | 191.05 | 10420.9 | `2.85e-09` |
| 10: | 0.75 | $20, 30, 130$ | sph. seg. | 34.53 | 287.47 | 18815.5 | `2.41e-09` |
| 11: | 1.1 | $40, 14, 30$ | cube | 32.33 | 320.32 | 17435.0 | `3.86e-09` |
| 12: | 1.01 | $40, 14, 30$ | cube | 32.53 | 338.07 | 17508.7 | `5.13e-09` |
| 13: | 1.001 | $40, 14, 30$ | cube | 32.59 | 337.23 | 16897.4 | `5.36e-09` |
| 14: | 1.0001 | $40, 14, 30$ | cube | 32.49 | 337.25 | 16839.6 | `5.36e-09` |
| 15: | 1.00001 | $40, 14, 30$ | cube | 34.40 | 339.52 | 17899.0 | `5.39e-09` |

One sees that in all cases the tree code is able to finish in a reasonable time, whereas the straightforward summation is far more costly. Note also that although $n$ is the same across different cases, the straightforward summation spends different amounts of time because the evaluation of the Matérn function has a different cost for different $\nu$.

**8. Concluding remarks.** We have developed a fast summation algorithm for the Matérn kernel based on the tree code framework. The algorithm handles arbitrary kernel orders, multiple sets of weights, different point set distributions, and the anisotropy in the definition of distances. With serial experiments of $n$ up to $2^{23}$ (8 million), we have demonstrated that the running time of the algorithm scales as $O(n \log n)$.

A restriction of the algorithm is that the proposed error estimation is not applicable when points are far away (specifically, when the centroid distance $\tau$ of two clusters is larger than 1). Although for many strongly correlated data the scale $\boldsymbol{\ell}$ is sufficiently large so that the algorithm can be used, it will be more favorable to rid the constraint on $\tau$ in order to widen the applicability of the algorithm.

The algorithm aims at performing computations with an arbitrary order $\nu$. Sometimes, an integer order or a half integer order is of particular interest. More efficient methods may exist for handling these special cases. When $\nu$ is an integer, a series expansion (with mostly integer powers) of the Matérn kernel is easy to obtain based on that of $\mathsf{K}_\nu$ (see, e.g., [1, 16]). When $\nu$ is a half integer, the Matérn function reduces to an exponential times a polynomial [28]. One may design different methods for these cases in order to bypass the relatively expensive computation of the Taylor coefficients for a general $\nu$.

**Acknowledgments.** We are thankful to the anonymous referees whose comments have substantially improved the paper. We also gratefully acknowledge the use of the Fusion cluster in the Laboratory Computing Resource Center at Argonne National Laboratory. This work was supported by the U.S. Department of Energy under Contract DE-AC02-06CH11357.

REFERENCES

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*, Dover Publications, 1965.

[2] D. E. Amos, *Algorithm 644: A portable package for Bessel functions of a complex argument and nonnegative order*, ACM Trans. Math. Softw., 12 (1986), pp. 265–273.

[3] M. Anitescu, J. Chen, and L. Wang, *A matrix-free approach for solving the parametric Gaussian process maximum likelihood problem*, SIAM J. Sci. Comput., 34 (2012), pp. A240–A262.

[4] J. E. Barnes, *A modified tree code: don't laugh; it runs*, J. Comput. Phys., 87 (1990), pp. 161–170.

[5] J. E. Barnes and P. Hut, *A hierarchical $O(N \log N)$ force-calculation algorithm*, Nature, 324 (1986), pp. 446–449.

[6] R. Beatson and L. Greengard, *A short course on fast multipole methods.* `http://math.nyu.edu/faculty/greengar/shortcourse_fmm.pdf`.

[7] M. Bebendorf, *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, vol. 63 of Lecture Notes in Computational Science and Engineering, Springer, 2008.

[8] H. A. Boateng, *Cartesian Treecode Algorithms for Electrostatic Interactions in Molecular Dynamics Simulations*, PhD thesis, University of Michigan, 2010.

[9] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Engineering Analysis with Boundary Elements, 27 (2003), pp. 405–422.

[10] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, *A fast solver for HSS representations via sparse matrices*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 67–81.
[11] J. Chen, M. Anitescu, and Y. Saad, *Computing $f(A)b$ via least squares polynomial approximations*, SIAM J. Sci. Comput., 33 (2011), pp. 195–222.
[12] J. Chen, H.-R. Fang, and Y. Saad, *Fast approximate kNN graph construction for high dimensional data via recursive Lanczos bisection*, Journal of Machine Learning Research, 10 (2009), pp. 1989–2012.
[13] J. Chen, L. Wang, and M. Anitescu, *A parallel tree code for computing matrix-vector products with the Matérn kernel*, Tech. Rep. ANL/MCS-P5015-0913, Argonne National Laboratory, 2013.
[14] H. Cheng, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm in three dimensions*, J. Comput. Phys., 155 (1999), pp. 468–498.
[15] J.-P. Chilès and P. Delfiner, *Geostatistics: Modeling Spatial Uncertainty*, Wiley-Interscience, 1999.
[16] O. J. Farrell and B. Ross, *Solved Problems: Gamma and Beta Functions, Legendre Polynomials, Bessel Functions*, Macmillan, 1963.
[17] W. Fong and E. Darve, *The black-box fast multipole method*, J. Comput. Phys., 228 (2009), pp. 8712–8725.
[18] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.
[19] L. F. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*, MIT Press, 1988.
[20] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part I: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.
[21] W. Hackbusch and S. Börm, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.
[22] R. Krasny and L. Wang, *Fast evaluation of multiquatric RBF sums by a Cartesian treecode*, SIAM J. Sci. Comput., 33 (2011), pp. 2341–2355.
[23] P. Li, H. Johnston, and R. Krasny, *A Cartesian treecode for screened Coulomb interactions*, J. Comput. Phys., 228 (2009), pp. 3858–3868.
[24] L. Lin, J. Lu, and L. Ying, *Fast construction of hierarchical matrix representation from matrixvector multiplication*, J. Comput. Phys., 230 (2011), pp. 4071–4087.
[25] K. Lindsay and R. Krasny, *A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow*, J. Comput. Phys., 172 (2001), pp. 879–907.
[26] P. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23.
[27] P. G. Martinsson, *A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1251–1274.
[28] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, MIT Press, 2006.
[29] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), pp. 187–207.
[30] J. K. Salmon and M. S. Warren, *Skeletons from the treecode closet*, J. Comput. Phys., 111 (1994), pp. 136–155.
[31] M. Stein, *Interpolation of Spatial Data: Some theory for Kriging*, Springer-Verlag, 1999.
[32] M. L. Stein, J. Chen, and M. Anitescu, *Stochastic approximation of score functions for Gaussian processes*, Annals of Applied Statistics, 7 (2013), pp. 1162–1191.
[33] H. Wendland, *Scattered Data Approximation*, Cambridge University Press, 2005.
[34] J. Xia and M. Gu, *Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2899–2920.
[35] L. Ying, G. Biros, and D. Zorin, *A kernel-independent adaptive fast multipole algorithm in two and three dimensions*, J. Comput. Phys., 196 (2004), pp. 591–626.

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. DE-AC02-06CH11357 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.