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 calculation 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 several important aspects are addressed: the partitioning of the point set, the computation of the Taylor approximation with error estimates, and the handling of multiple sets of weights originating from multiple matrix-vector multiplications with the same matrix. The computational cost of the derived algorithm scales as $O(n \log n)$ for n points. Comprehensive numerical experiments are shown to demonstrate the practicality of the design. The development of a similar algorithm based on the multipole expansion framework is also discussed.

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

AMS subject classifications. 65F30

1. Introduction.

The Matérn kernel [21, 23, 18] 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 [11, 23].

The Matérn kernel gives rise to a positive definite covariance matrix Φ, 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 Φ is crucial in many statistical problems, such as sampling, maximum likelihood estimation, and interpolation (also known as kriging) [21, 23]. Some of these problems require the solution of a linear system with respect to Φ, 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, 22]. 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 [8].

Motivated by the needs of efficiently computing the product $\Phi \boldsymbol{q}$ for the Matérn kernel, we have designed 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 possible 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., [17, 16, 15]). For the Matérn kernel, preliminary work [3] was conducted for the special order 1.5. In this work, we address the desired functionality and propose several nonconventional designs based on the framework.

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

Following a general tree code, the proposed algorithm recursively divides a set of points to form a hierarchy of clusters. The evaluation of the kernel for a pair of points can be approximated by the truncated series expansion of the kernel around the centroids of the clusters that contain the two points, if the clusters are sufficiently far away. Series expansion is a central analytic tool in a fast summation algorithm. For the Matérn kernel, it is not straightforward to derive an expansion that is inexpensive to evaluate, even though abundant results of the expansions of the Bessel functions are known (see, e.g., [12, 1]). Thus, we first derive recurrence formulas to compute the Taylor expansion coefficients through the use of several properties of the Bessel functions.

The expansion of the kernel at two centroids is called a *double expansion*, whereas the usual expansion at only one centroid is called a *single expansion*. Double expansion was considered in [7]. An advantage of double expansion is that the number of centroid pairs that require expansion coefficients is potentially significantly reduced. Since the computation of expansion coefficients is expensive (to be discussed shortly), double expansion is a natural design to reduce the cost.

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. The hypothesized regression form works fairly well for our case.

To achieve wide applicability of the algorithm in real-life scenarios, we propose a point set partitioning scheme that is different from the traditional axes-aligned approaches (for example, a 3D octree). This scheme is based on the principal component analysis that maximizes the separation of two clusters, in order to encourage that the radii of the resulting clusters are as small as possible. It copes with the arbitrary distribution of the point set and the possible anisotropy when defining distances. Furthermore, this scheme always partitions a point set into two balanced subsets; and thus it results in a complete binary tree with equally sized leaves, which suggests a natural distribution of the data for parallel processing. The parallel implementation of our algorithm will be discussed in a forthcoming paper.

Because of the targeted use with multiple sets of weights, the algorithm is naturally separated in two phases: planning and evaluation. All the computations independent of the weights are put in the first phase, so that the second phase can reuse the same information for different sets of weights as much as possible. These computations include the partitioning, the regression of the error formulas, and most notably the generation of the Taylor coefficients. The computational work of computing the Taylor coefficients is not negligible, especially when the points are in high dimensions; thus it is advantageous to precompute and store them in order to avoid repeated calculations.

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 occurs whenever the ratio between the cluster radius and the centroid distance falls within a fixed threshold (known as the *multipole acceptance criterion* [5, 4, 20]). 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 occurrence of approximation, and we do not predefine a threshold for this ratio. This distinction makes it hard to conclude a precise complexity of our algorithm. Nevertheless, experimental results for uniformly distributed points agree with the $O(n \log n)$ scaling.

The rest of the paper is organized as follows. Section 2 formally defines the Matérn kernel, and Section 3 the notation of summation and expansion. The computational routine for computing the Taylor coefficients is developed in Section 4. Section 5 completes several details of the tree code, including the partitioning of the point set in Section 5.1 and the estimation of errors in Section 5.2. Then, Section 6 presents the formal algorithm, together with an estimate of the computational cost. Comprehensive numerical experiments are shown in Section 7. Section 8 presents concluding remarks and discusses the possibility of developing a fast summation algorithm based on the multipole expansion framework.

2. Matérn kernel. The Matérn function of a one-dimensional variable $r \geq 0$ of order $\nu > 0$ is defined as [18]

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

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

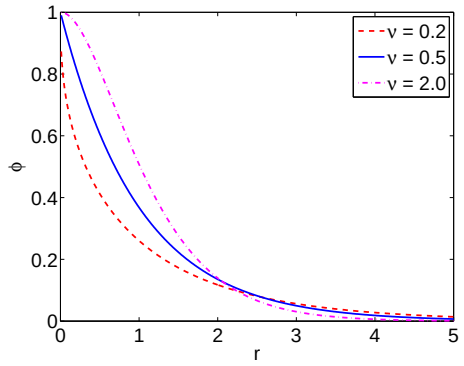


FIG. 2.1. *Matérn function with different values of ν.*

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 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 ν is often called the *smoothness* because the function $\phi(|x|)$, $x \in \mathbb{R}$, has a higher differentiability with respect to x when ν is larger. It also reflects the

shape of the samples of the underlying Gaussian process, because when ν 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 ϕ at the centroid of a cluster of nearby sources and the centroid of a cluster of nearby targets, so that one can replace the summation of the n terms in (3.2) by the evaluation of a Taylor polynomial, if the two clusters are sufficiently far away. 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 ϕ 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 ϕ around $\boldsymbol{y}_c$, treating $\boldsymbol{x}_c + \Delta\boldsymbol{x}$ constant, then expanding ϕ around $\boldsymbol{x}_c$, treating $\boldsymbol{y}_c$ constant. Since $\partial_{\boldsymbol{x}}^{\boldsymbol{j}} \phi = (-1)^{\|\boldsymbol{j}\|} \partial_{\boldsymbol{y}}^{\boldsymbol{j}} \phi$, 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 δ 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 approximation error of $\boldsymbol{s}$ is less than $\delta \|\boldsymbol{q}\|_2$.

Note that the double expansion (3.3) naturally requires two truncation orders, p_1 for the near field (where target points are located) and p_2 for the far field (where source points are located). 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 one $(2d)$-dimensional variable and expands ϕ at this variable. Then, the truncation of this expansion 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 often a good idea to place a small order for the near field and a large order for the far field, if $\Delta\boldsymbol{x}$ is small and $\Delta\boldsymbol{y}$ is relatively large.

From a computational perspective, we rearrange the terms in (3.5) in order that the expression on the right can be efficiently evaluated:

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

The general idea of the fast summation is to precompute the binomial coefficients and the Taylor coefficients, because they can be shared by different targets. Given the weights, the weighted source moments are also computed. Then, for each target $\boldsymbol{x}_i$, the target moments are computed for each cluster C_s of sources, and the computed partial sum $s_i(C_s)$ is accumulated to s_i. The details are presented in Sections 5 and 6. First, however, it is necessary to consider how the Taylor coefficients are computed.

4. Taylor coefficients. We use recursion to compute the Taylor coefficients $\partial_{\boldsymbol{y}}^{\boldsymbol{k}} \phi / \boldsymbol{k}!$ for $\|\boldsymbol{k}\|$ up to some order p. Key to the recurrence is the following property of $\mathsf{K}_\nu(R)$ for any $\nu > 0$ (see, e.g., [12, p. 294, Table V-1]):

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

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

4.1. Recurrence formula. Define $c = \sqrt{2\nu}$, $R = cr$, $g_\nu(R) = R^\nu \mathsf{K}_\nu(R)$. Then

$$\frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}} \phi}{\boldsymbol{k}!} = \frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}} g_\nu(cr)}{2^{\nu-1}\Gamma(\nu) \cdot \boldsymbol{k}!}.$$

For a general $c > 0$ and any $\boldsymbol{k}$ and ν, we derive a recurrence formula for evaluating

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

given $r \neq 0$. Then, for the specific ν in the kernel and $\boldsymbol{k}$, we substitute c by $\sqrt{2\nu}$ and obtain

$$\frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}}\phi}{\boldsymbol{k}!} = G_\nu^{\boldsymbol{k}}(\sqrt{2\nu}).$$

The reason for breaking the definition of $G_\nu^{\boldsymbol{k}}(c)$ into several cases with respect to ν is to encourage a better numerical behavior of the recurrence relation. For now we note that z is some function used for this purpose; it will be defined in the next subsection. The case of $\nu < 0$ for $G_\nu^{\boldsymbol{k}}(c)$ is needed even though a Matérn order is always positive, because a recurrence may eventually decrease ν to negative even though it starts from positive.

To simplify notation, we write ∂_i to mean the partial derivative with respect to y_i. With (4.1), we have

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

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

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

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

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

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

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

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

where h, a function of ν, is determined from the different cases of the definition of

$G_\nu^{\boldsymbol{k}}(c)$ in (4.2):

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

This formula suggests that the $\boldsymbol{k}$th partial derivative at the level ν depends on lower derivatives at level $\nu - 1$.

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

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

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

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

$$G_\nu^{\mathbf{0}}(c) = \begin{cases} \dfrac{(cr)^\nu \mathsf{K}_\nu(cr)}{2^{\nu-1}\Gamma(\nu)}, & \nu > 0 \\ \dfrac{\mathsf{K}_0(cr)}{z(cr)}, & \nu = 0 \\ \dfrac{(cr)^{-\nu}\mathsf{K}_{-\nu}(cr)}{2^{-\nu-1}\Gamma(-\nu)}, & \nu < 0. \end{cases} \tag{4.10}$$

We examine it case by case. The cases $\nu > 0$ and $\nu < 0$ lead to the same value of $G_\nu^{\mathbf{0}}(c)$, for a pair of positive ν and negative ν that have the same absolute value. When $c = \sqrt{2\nu}$, $G_\nu^{\mathbf{0}}(c)$ is equal to an evaluation of the Matérn function $\phi(r)$. When r is close to zero, the function value is close to 1.

Now consider $\nu = 0$. 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 $G_\nu^{\mathbf{0}}(c)$ behaves better when it is close to the origin. Consider an approximation of 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, $R_0 = 2e^{-\gamma-1} \approx 0.413$ is used to make z continuous. With this definition, $G^{\mathbf{0}}_\nu(c)$ 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

$$\partial^{\boldsymbol{k}}_{\boldsymbol{y}}\phi/\boldsymbol{k}! = G^{\boldsymbol{k}}_\nu(\sqrt{2\nu})$$

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

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

for $\nu' = \nu - p, \ldots, \nu$ 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 ν' until finally A (and B) holds the values of $G^{\boldsymbol{k}}_\nu(\sqrt{2\nu})$. The following subroutine TaylorCoefficients summarizes this procedure.

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

$$\sum_{i=0}^{p} \binom{i+d-1}{d-1} = \binom{p+d}{d}.$$

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,

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

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 $\boldsymbol{x} - \boldsymbol{y}$) 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 $G_\nu^{\mathbf{0}}$ is close to 1 for any ν; and when r is not abnormally large, $G_\nu^{\mathbf{0}}$ will not be exceedingly small. In fact, the term "abnormally large" is vague, and we delay its clarification until Section 5.2 when error control is the concern.

The recurrence depends on the function $h(\nu')$ defined in (4.8). It has discontinuities at $\nu' = 0$ and $\nu' = 1$, and when approaching these discontinuities the function value grows without bound. Thus, the recurrence may be numerically unstable when the smoothness parameter ν 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 ν differing from 1 by only 10^{-5}.

5. Tree code framework. A general tree code 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 is performed starting from the root. Any tree node being visited contains a set C_s of sources. Given a tolerance ϵ, if the truncated Taylor expansion at the centroids of C_s and C_t yields an approximation error that is less than ϵ, 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.

In practice, multiple sets of weights $\{q_j\}$ may be of interest since they correspond to matrix-vector multiplications with the same matrix and multiple right-hand sides. Then, it is beneficial to split the recursive treewalk in two phases so that the work of computing the Taylor coefficients is not repeated for different sets of weights. This idea leads to two subroutines, TreeWalk-Planning and TreeWalk-Evaluation. In the planning phase, the Taylor coefficients up to order $p = p_1 + p_2$ for the expansion at $\boldsymbol{x}_c$ and $\boldsymbol{y}_c$, which are the centroids of the leaf node `leaf` and the tree node `node` being visited, respectively, are computed if the expansion criterion is met. The subroutine TaylorCoefficients introduced in Section 4.3 is used for this computation. Then, in the evaluation phase, the partial sums are computed by using either Taylor approximation or direct summation. Both phases require a subroutine CanExpand that checks whether the Taylor approximation yields an error less than ϵ. This subroutine is related to the error control scheme to be discussed in Section 5.2 and is presented there.

```
1: subroutine TreeWalk-Planning(leaf, node, p)
2:     Let C_t and C_s be the point clusters that leaf and node contain, respectively
3:     if CanExpand(C_t, C_s, ε) then
4:         Let x_c and y_c be the centroids of C_t and C_s, respectively
5:         Call TaylorCoefficients(p, x_c − y_c) to compute and store
```

```
            the Taylor coefficients ∂_y^k φ(x_c, y_c)/k! for all ‖k‖ ≤ p
    else if node has no children then
        // Do nothing
    else
        For all child of node, call TreeWalk-Planning(leaf, child, p)
    end if
end subroutine

subroutine TreeWalk-Evaluation(leaf, node)
    Let C_t and C_s be the point clusters that leaf and node contain, respectively
    if CanExpand(C_t, C_s, ε) then
        for all x_i ∈ C_t do
            Compute target moment and weighted source moment
            Compute s_i(C_s) using Taylor approximation (3.7)
            Update s_i ← s_i + s_i(C_s)
        end for
    else if node has no children then
        for all x_i ∈ C_t do
            Update s_i ← s_i + Σ_{y_j ∈ C_s} q_j φ(x_i, y_j) via direct summation
        end for
    else
        For all child of node, call TreeWalk-Evaluation(leaf, child)
    end if
end subroutine
```

Based on the above framework, we address the following issues to complete the algorithm:

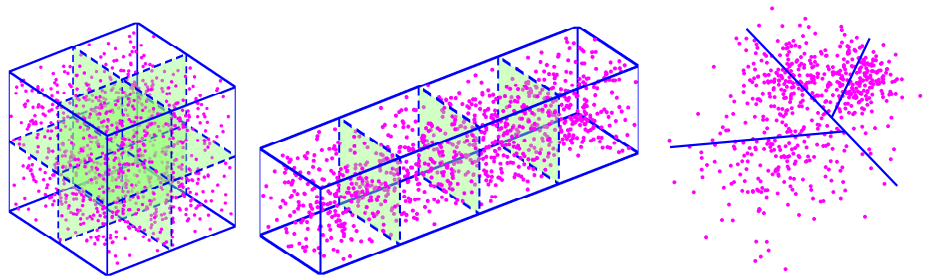
1. How to partition the point set?
2. How to determine whether the Taylor approximation yields an error less than ϵ?

The answers are in the following two subsections.

5.1. Partitioning of the point set. For simplicity, let us consider $\mathbb{R}^3$. When the points are uniformly distributed in a cube, it is natural to rotate the cube and align its faces with the coordinate axes. Usually, the cube is partitioned into 8 equal subcubes and the partitioning is recursive (see Figure 5.1(a)). This results in an octree hierarchy. If the points are not uniform in space, one can selectively partition dense cubes so that leaf nodes of the tree represent clusters with roughly the same number of points, or one can always partition a cube into 8 subcubes that contain equal numbers of points. In the latter case, the resulting hierarchy is a complete tree.

However, the configuration of the point set is often more complicated in practice. Consider, for example, Figure 5.1(b), where the points are uniform. Let us say the x-dimension of the cube is stretched to four times that of the unit cube. Instead of the 8-way partitioning scheme used for (a), it is more natural to first partition the x-dimension into four equal parts, because in this manner the diameter of the resulting clusters is much smaller than that resulting from the usual 8-way partitioning. In other words, the points are more compactly grouped together. A benefit of having a smaller diameter is that the Taylor approximation is more accurate.

Even when the cube is not stretched, but if the scale parameters are, for example, $[1/4, 1, 1]$, it is still beneficial to first partition the x-dimension into four equal parts



(a) Cube (b) Stretched cube (c) Arbitrary shape/density

FIG. 5.1. *Different point set configurations and partitioning schemes.*

in order to obtain clusters of a small elliptical diameter.

Consider a general setting where the point set has an arbitrary shape and uneven density (see Figure 5.1(c)). To systematically partition the point set, we consider the *binary space partitioning*. 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. Without loss of generality, we prescale each coordinate of the points by the corresponding scale parameter ℓ_i. Then, we find the principal direction along which the variance of the scaled points is the largest. This direction is simply the dominant eigenvector of the covariance matrix of the point set. For any hyperplane passing through the centroid, using this vector as the normal direction minimizes the squared sum of the distances between the points and the hyperplane. Therefore, partitioning with this hyperplane encourages clusters that are more compactly grouped. Usually, the hyperplane passing through the centroid does not yield equal-sized clusters; thus the hyperplane has to be shifted along the normal direction. A convenient way to perform the shift is to find the median of the signed distances between the points and the hyperplane that passes through the centroid. By finding this median we obtain the partitioning result.

The subroutine BIPARTITIONING summarizes the computations. The procedure results in a complete binary tree because each time a cluster is bi-partitioned. It requires a parameter n_0 that specifies the maximum size of a leaf. Therefore, the number of tree levels (including the root) is $\lceil \log_2(n/n_0) \rceil + 1$. The subroutine is recursive, and input to the subroutine is the cluster X to be partitioned and the tree `level` where X is located. When `level` has not reached $\lceil \log_2(n/n_0) \rceil + 1$, the subroutine recursively calls itself with the two resulting partitioned clusters. For more information on the procedure, see [9], where the same idea is used to construct a nearest neighbors graph for a set of points in a divide-and-conquer manner.

subroutine BIPARTITIONING(X, n_0, `level`)
  For each $\boldsymbol{x}_i \in X$, scale $\boldsymbol{x}_i$ for each coordinate by $\boldsymbol{\ell}$ to get $\boldsymbol{x}'_i$. Let $X' = \{\boldsymbol{x}'_i\}$
  Compute the centroid $\boldsymbol{x}'_c$ of X'
  Form the covariance matrix C of the point set X'
  Compute the dominant eigenvector $\boldsymbol{u}$ of C
  Compute signed distances $\xi_i = \langle \boldsymbol{x}'_i - \boldsymbol{x}'_c,\ \boldsymbol{u} \rangle$ for all i
  Find ξ_m, the median of $\{\xi_i\}$
  Let two resulting clusters $X_l = \{\boldsymbol{x}_i \mid \xi_i \le \xi_m\}$ and $X_r = \{\boldsymbol{x}_i \mid \xi_i > \xi_m\}$
  if `level` $< \lceil \log_2(n/n_0) \rceil + 1$ **then**

10: BiPartitioning(X_l, n_0, `level+1`)
11: BiPartitioning(X_r, n_0, `level+1`)
12: **end if**
13: **end subroutine**

Binary space 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 complete, 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, the computational cost of lines 2 to 8 (that is, all the work excluding the recursion) is linear in the point set size.[1] This cost is asymptotically the same as that of the usual, say 8-way, partitioning (in the 3D case). Then, including recursion, the overall cost of calling BiPartitioning with the whole set of n points is $O(n \log_2(n/n_0))$.

5.2. Error control. It is important to characterize the error, denoted by δ, 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 the bound of δ are the elliptical distance τ between a pair of centroids and the elliptical radius ρ for a cluster. The quantity ρ is the expansion radius. To distinguish sources and targets, we use ρ_t for a target cluster and ρ_s for a source cluster (see Figure 5.2). It is desirable to bound δ based on ρ_t, ρ_s, and τ.

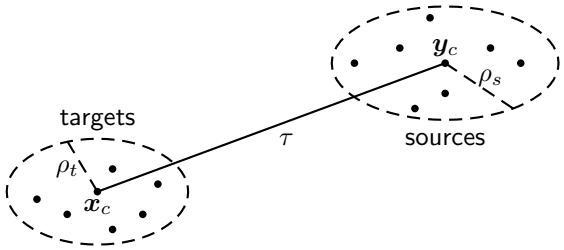


FIG. 5.2. *Expansion radii ρ_t, ρ_s and distance τ (all elliptical).*

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

[1] To be more exact, assuming that the current point set X has n' points, the computational cost of lines 2 to 8 is $O(d^2n' + d^3)$, where the first term comes from computing the covariance matrix and the second term comes from computing eigenvectors. Because d is a small constant, it is ignored.

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 ρ 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 α_1, α_2, and α_3 are coefficients to be determined. The quantity $\delta_p^{\max}$ is a function of ρ and τ, given p. The term ρ/τ 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 ρ_t, ρ_s, and τ, given (p_1, p_2). When there is no confusion, we write $\delta^{\max}$ for simplicity. We do not hypothesize $\delta^{\max}$ as a function of the expansion orders, the reason for which will be clear soon.

5.2.1. 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 5.3(a) plots the variation of $\delta^{\max}$ with respect to τ in the log-log scale, by fixing ρ/τ. 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 $\delta^{\max}$ with respect to ρ/τ, by fixing τ. Varying τ and ρ/τ simultaneously results in a plane pattern as shown in plot (c). This indicates that (H1) is highly plausible.

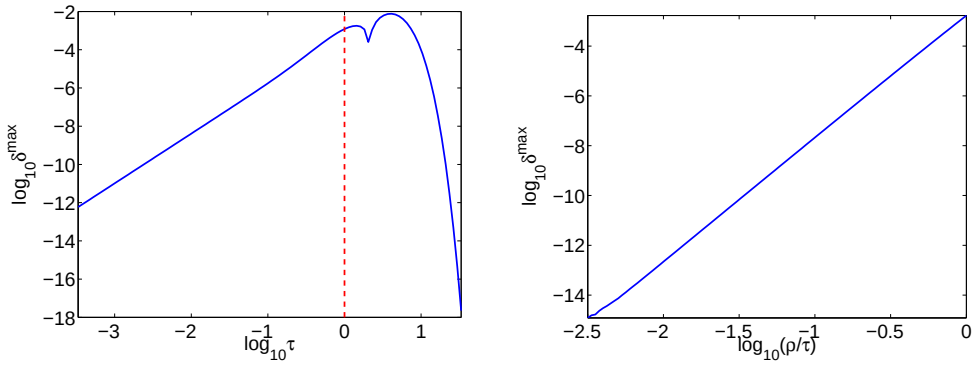
When one considers a double expansion, the hypothesis (H2) states that $\delta_{p_1,p_2}^{\max}$ is an overall effect of two single expansions: $\delta_{p_1}^{\max}(\rho_t, \tau + \rho_s)$, which results from an expansion around the target centroid $\boldsymbol{x}_c$ by assuming that the source $\boldsymbol{y}_c$ 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 τ but varying ρ_s and ρ_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).

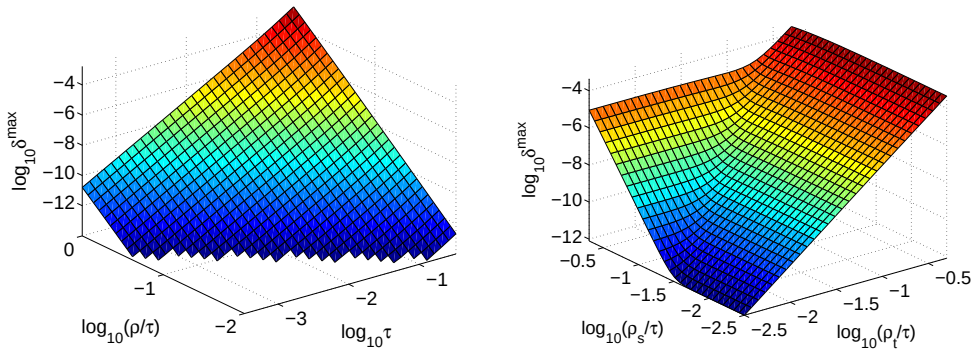
Note that the hypotheses (H1) and (H2) preclude the situation $\tau > 1$. In plot (a), one sees that $\delta^{\max}$ behaves completely differently in this situation. We do not know how to characterize this behavior. On the other hand, the kernel used in practice often entails large scaling parameters, 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 $\delta^{\max}$ with respect to p and ν. Figure 5.4 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).

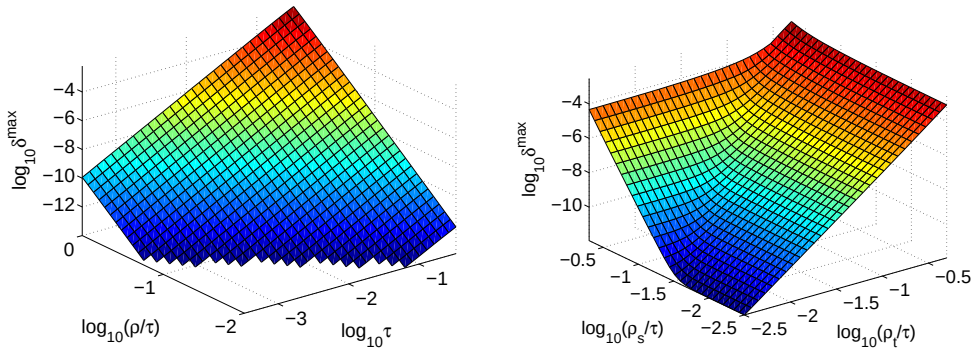
5.2.2. Subroutines. Two subroutines are used to perform error control. The first one, ErrorControlInfo, is used to test the hypotheses (H1) and (H2). It chooses a few τ and ρ/τ, equally spaced in the log-scale from $10^{-2.5}$ to $10^{0.5}$. For each pair of τ and ρ/τ, it estimates $\delta_{p_2}^{\max}(\rho, \tau)$, which is the maximum absolute difference



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



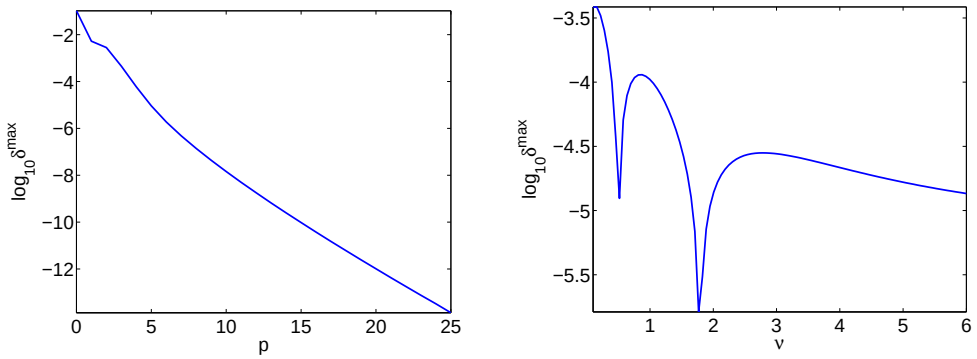
(e) 2D, $\nu = 1.3$, $p = 4$ (f) 2D, $\nu = 1.3$, $\tau = 1/3$, $p_1 = 3$, $p_2 = 5$

FIG. 5.3. *Taylor approximation error with respect to τ and ρ/τ.*

between $\phi(\boldsymbol{x}_c, \boldsymbol{y}_c + \Delta\boldsymbol{y})$ and

$$\sum_{\|\boldsymbol{k}\|=0}^{p_2} \frac{\partial_{\boldsymbol{y}}^{\boldsymbol{k}} \phi(\boldsymbol{x}_c, \boldsymbol{y}_c)}{\boldsymbol{k}!} (\Delta\boldsymbol{y})^{\boldsymbol{k}},$$

for all $\|\boldsymbol{x}_c - \boldsymbol{y}_c\|_2 = \tau$ and $\|\Delta\boldsymbol{y}\|_2 = \rho$ (see (3.4) when $\Delta\boldsymbol{x}$ is zero). Of course, we cannot exhaust all such vectors $\boldsymbol{x}_c - \boldsymbol{y}_c$ and $\Delta\boldsymbol{y}$, and thus a sampling is used. With



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

FIG. 5.4. *Taylor approximation error with respect to p and ν.*

τ, ρ, and $\delta_{p_2}^{\max}$, (H1) is then regressed to obtain the coefficients α_1, α_2, and α_3. 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 α_1, α_2, and α_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/computed by performing another sampling. This time, $\boldsymbol{x}_c - \boldsymbol{y}_c$, $\Delta\boldsymbol{x}$, and $\Delta\boldsymbol{y}$ are sampled with a 2-norm fixed at τ, ρ_t, and ρ_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. A second subroutine CANEXPAND uses these coefficients in the recursive treewalks to determine whether Taylor expansion occurs.

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** τ and ρ 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 ρ, τ, and $\delta_{p_2}^{\max}$ to regress (H1). Obtain α_1, α_2, and α_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 α_1, α_2, α_3.
// Test of (H2)
9: **for all** τ, ρ_t, ρ_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 ρ_t, ρ_s, and τ.
13: If difference between both sides of (H2) is larger than 1, **return error**
14: **return** two sets of α_1, α_2, α_3
15: **end subroutine**

subroutine CanExpand(C_t, C_s, ϵ)
  If $\rho_t + \rho_s \geq \tau$, **return false**
  Compute $\delta^{\max}_{p_1,p_2}(\rho_t, \rho_s, \tau)$ using the fitted values of α_1, α_2 and α_3.
  If $\delta^{\max}_{p_1,p_2}(\rho_t, \rho_s, \tau) < \epsilon$, **return true**; otherwise, **return false**
end subroutine

6. The algorithm. We are ready to present the overall algorithm. As mentioned, the algorithm comprises a planning phase and an evaluation phase. The planning phase is independent of the weights and is computed only once. In this phase, the point set is recursively partitioned, the error formula is fitted, a binomial table is constructed, and a treewalk is performed for each target set C_t (residing in a leaf node) to compute Taylor coefficients. Then comes the evaluation phase given a set of weights. The same recursive treewalk is performed in this phase. In the intermediate tree levels, if Taylor expansion can be used, the target moments and the weighted source moments are computed. On the other hand, in the leaf level, direct summation is performed if Taylor expansion yields an unacceptable error. Algorithm 1 summarizes this procedure, where `leaf` and `root` are the nodes of the hierarchy tree with their literal meanings.

Algorithm 1 Tree code method for computing (3.1)

Require: Parameters: expansion order (p_1, p_2), tolerance ϵ, maximum leaf size n_0
  // Planning phase
Call ErrorControlInfo to fit error formula (H1). If fitting is unsuccessful, exist this algorithm and use straightforward summation to compute (3.1).
Call BiPartitioning($\{\boldsymbol{x}_i\}$, n_0, `1`) to construct a tree hierarchy for $\{\boldsymbol{x}_i\}$
Compute a binomial table to store $\binom{k}{j}$ for all $k \leq p_1 + p_2$ and $j \leq p_1$
for all `leaf` **do**
  Call TreeWalk-Planning(`leaf`, `root`, $p_1 + p_2$) to obtain Taylor coefficients
end for
  // Evaluation phase
Initialize $s_i = 0$ for all i
for all `leaf` **do**
  Call TreeWalk-Evaluation(`leaf`, `root`) to update $\{s_i\}$ for all $\{\boldsymbol{x}_i\} \in$ `leaf`
end for
return $\{s_i\}$

Let us consider the computational complexity of the algorithm with respect to n and n_0. The bi-partitioning step scales as $O(n \log_2(n/n_0))$ as concluded in Section 5.1. The fitting of the error formula and the construction of the binomial table are independent of n and n_0, and so are the CanExpand calls in later treewalks. Let m_{expand} and m_{direct} denote the number of node pairs where Taylor expansion occurs and where direct summation occurs, respectively. The cost of TreeWalk-Planning is $O(m_{\text{expand}} + m_{\text{direct}})$. In TreeWalk-Evaluation, 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_{expand} and

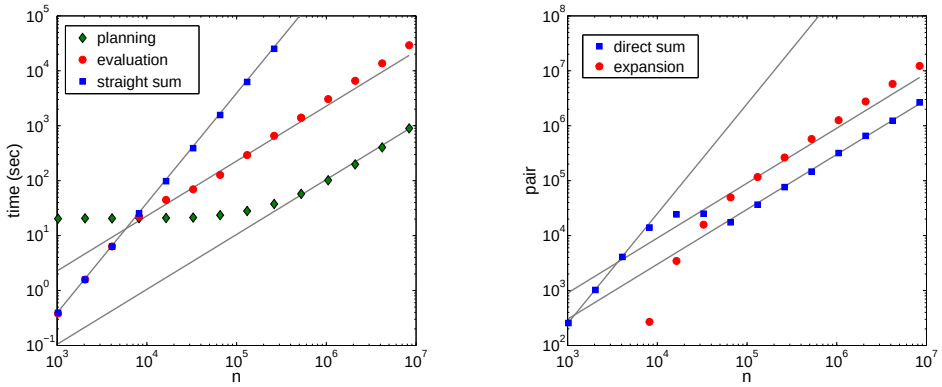
m_{direct} vary with not only n but also the configuration of the point set. In a simplified setting, for each point $\boldsymbol{x}_i$, if one Taylor approximation occurs in each intermediate tree level and one direct summation happens at the leaf that contains $\boldsymbol{x}_i$, then

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

In general, estimating m_{expand} and m_{direct} is difficult, but empirical results usually approximately agree with (6.1); see Figure 7.1(b). This concludes the overall $O(n \log n)$ cost of our algorithm, 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 the proposed algorithm. For the purpose of practicality and code reuse, we implemented a parallel version of the program in C++; but because of the complexity and several nontrivial designs of the parallelism, we will discuss the parallelization in a separate paper. Thus, 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 K_ν used the `AMOS` package available from `http://www.netlib.org/amos/`; see also [2]. The first few subsections concern the computational cost and the input 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 other choices of the 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 the design. We set 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_{expand} and direction summation node pairs m_{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.

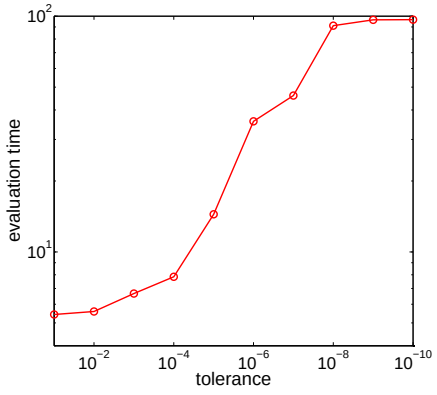
In this particular setting, the tree code outperforms straightforward summation starting at approximately 8,000 points. Before this, the points are too sparse and no Taylor approximation occurs, probably because the choice of the leaf size yields spatially large clusters even for the leaves. Plot (b) confirms this observation. When n is less than 2^{13} (approximately 10,000), the number m_{expand} of nodes pairs where expansion occurs is zero, and thus all the calculations are direction 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_{expand} and m_{direct} grow linearly.

Note, also, that when $n < 2^{17}$ (approximately 100,000), 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.

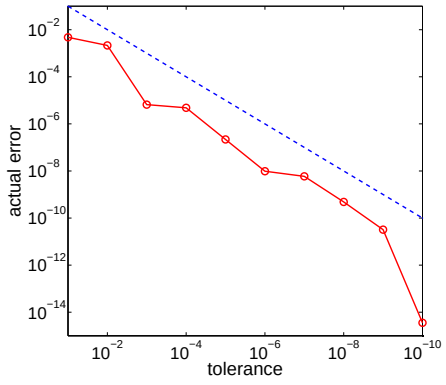
7.2. Choice of expansion orders. One characteristic of the proposed algorithm is the need to specify a pair (p_1, p_2) of Taylor orders in order to approximate the summation to a desired accuracy. For $n = 16,384$ and $n_0 = 64$, the table in Figure 7.2(a) shows an optimal choice of (p_1, p_2) for ϵ 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 ϵ 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 ϵ.

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

$\log_{10}\epsilon$	-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 ϵ.



(c) Actual error δ versus ϵ.

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

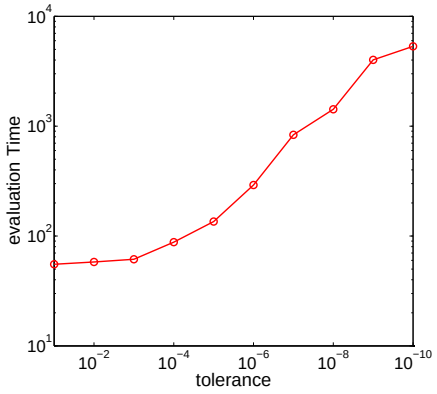
The actual approximation error δ (equivalently defined as the relative error of the

matrix-vector product) versus the tolerance ϵ is shown in plot (c). By the nature of the algorithm, δ is less than ϵ, as is shown in the plot, where the red circular markers are all under the blue dashed reference line. This plot also shows that δ is often one to two orders of magnitude smaller than ϵ, which indicates that ϵ can be used as a useful reference when one is interested in computing the summation to a particular accuracy.

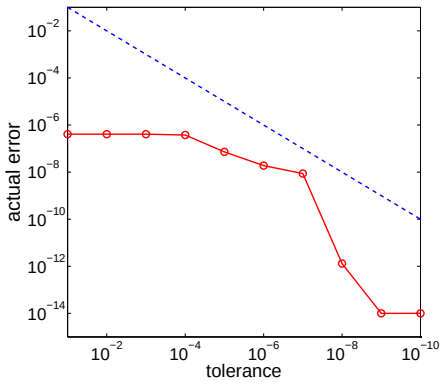
One may notice that when $\epsilon = 10^{-10}$, not only do the values of p_1 and p_2 deviate from the general trend, but also δ is far away from ϵ. A possible explanation is that large expansion orders yield expensive evaluation of the Taylor approximation because the number of coefficients explodes. Then, the gain in aggressive Taylor approximation at the top levels of the hierarchy tree is perhaps less significant compared with performing the approximation at lower levels of the hierarchy, where cluster radii are smaller and expansion orders do not need to be that large to achieve the designated tolerance.

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

$\log_{10}\epsilon$	-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 ϵ.



(c) Actual error δ versus ϵ.

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

We also perform experiments on a larger n $(131,072)$ and report the results in Figure 7.3. In this case, one sees a nice trend of the optimal expansion orders for all ϵ. 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 δ is far away from ϵ, 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 $\epsilon = 10^{-6}$.

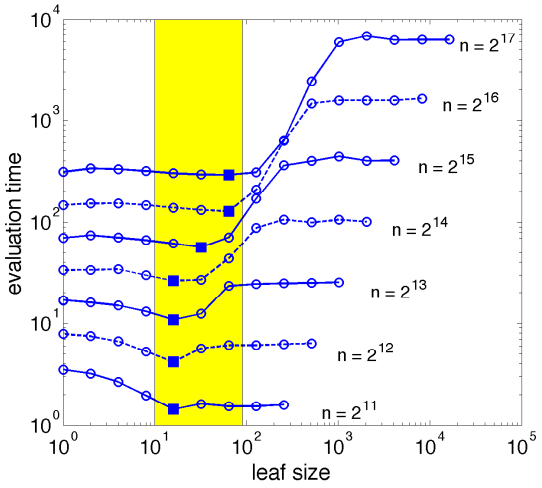


FIG. 7.4. *Evaluation time versus leaf size n_0 for different n.*

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 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 ν and ℓ. In rows 5 to 7, the points are sampled on a sphere of radius 1, 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 ν to be progressively closer to 1, in order to test the numerical viability of the algorithm in handling ν that is close to an integer.

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 (essentially the evaluation of K_ν) has a different cost for different ν.

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.

	ν	$\boldsymbol{\ell}$	Distri.	Plan.	Eval.	Straight.	Err. δ
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

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 control scheme is not applicable when points are far away (specifically, when the centroid distance τ 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 τ in order to widen the applicability of the algorithm.

The algorithm aims at performing computations with an arbitrary order ν. Sometimes, an integer order or a half integer order is of particular interest. More efficient methods may exist for handling these special cases. When ν is an integer, a series expansion (with mostly integer powers) of the Matérn kernel is easy to obtain based on that of K_ν (see, e.g., [1, 12]). When ν is an integer plus $\frac{1}{2}$, the Matérn function reduces to an exponential times a polynomial [18]. One can design different methods for these cases in order to bypass the relatively expensive computation of the Taylor coefficients for a general ν.

A natural question is whether a similar fast summation algorithm can be developed based on the framework of fast multipole method (FMM; see, e.g., [19, 13, 14, 10]) instead. An advantage of FMM is that in general it entails an $O(n)$ computational complexity. To apply this framework, one needs to perform a series expansion of the kernel, where the series bases are inexpensive to evaluate and easy to convert to polynomials. Note that the Taylor expansion in this paper is a local expansion that depends on the centroid distance, whereas a series expansion used for FMM is one that expands at the origin. The idea of a recurrence formulation in Section 4 is perhaps useful for computing the coefficients of a series expansion, although the

formulas therein do not directly apply because r cannot be zero in (4.2). It remains to investigate other analytic techniques to fill the gap of such an FMM-type of method.

Acknowledgments. We 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] H. A. Boateng, *Cartesian Treecode Algorithms for Electrostatic Interactions in Molecular Dynamics Simulations*, PhD thesis, University of Michigan, 2010.
[8] 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.
[9] 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.
[10] H. Cheng, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm in three dimensions*, J. Comput. Phys., 155 (1999), pp. 468–498.
[11] J.-P. Chilès and P. Delfiner, *Geostatistics: Modeling Spatial Uncertainty*, Wiley-Interscience, 1999.
[12] O. J. Farrell and B. Ross, *Solved Problems: Gamma and Beta Functions, Legendre Polynomials, Bessel Functions*, The Macmillan Conmpany, 1963.
[13] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.
[14] L. F. Greengard, *The Rapid Evaluation Of Potential Fields In Particle Systems*, MIT Press, 1988.
[15] R. Krasny and L. Wang, *Fast evaluation of multiquatric RBF sums by a Cartesian treecode*, SIAM J. Sci. Comput., 33 (2011), pp. 2341–2355.
[16] P. Li, H. Johnston, and R. Krasny, *A Cartesian treecode for screened Coulomb interactions*, J. Comput. Phys., 228 (2009), pp. 3858–3868.
[17] 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.
[18] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, MIT Press, 2006.
[19] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), pp. 187–207.
[20] J. K. Salmon and M. S. Warren, *Skeletons from the treecode closet*, J. Comput. Phys., 111 (1994), pp. 136–155.
[21] M. Stein, *Interpolation of Spatial Data: Some theory for Kriging*, Springer-Verlag, 1999.
[22] M. L. Stein, J. Chen, and M. Anitescu, *Stochastic approximation of score functions for Gaussian processes*, Tech. Rep. ANL/MCS-P2091-0512, Argonne National Laboratory, 2012.
[23] H. Wendland, *Scattered Data Approximation*, Cambridge University Press, 2005.