Preprint ANL/MCS-P1693-1109

COMPUTING $f(A)b$ VIA LEAST SQUARES POLYNOMIAL APPROXIMATIONS

JIE CHEN*, MIHAI ANITESCU†, AND YOUSEF SAAD*

Abstract. Given a certain function f, various methods have been proposed in the past for addressing the important problem of computing the matrix-vector product $f(A)b$ without explicitly computing the matrix $f(A)$. Such methods were typically developed for a specific function f, a common case being that of the exponential. This paper discusses a procedure based on least squares polynomials that can, in principle, be applied to any (continuous) function f. The idea is to start by approximating the function by a spline of a desired accuracy. Then, a particular definition of the function inner product is invoked that facilitates the computation of the least squares polynomial to this spline function. Since the function is approximated by a polynomial, the matrix A is referenced only through a matrix-vector multiplication. In addition, the choice of the inner product makes it possible to avoid numerical integration. As an important application, we consider the case when $f(t) = \sqrt{t}$ and A is a sparse, symmetric positive-definite matrix, which arises in sampling from a Gaussian process distribution. The covariance matrix of the distribution is defined by using a covariance function that has a compact support, at a very large number of sites that are on a regular or irregular grid. We derive error bounds and show extensive numerical results to illustrate the effectiveness of the proposed technique.

Key words. Matrix function, least squares polynomials, Gaussian process, sampling

AMS subject classifications. 65F35, 65F30, 65F50

1. Introduction.

It is often necessary to compute a matrix-vector product of the form $f(A)b$, where $f(A)$ is a matrix function well defined for the matrix A and b is a column vector. Examples of such functions f include the sign function $\mathrm{sgn}(A)$, the exponential $\exp(A)$, the logarithm $\log(A)$, and the square root $A^{1/2}$. We restrict our study to the case where A is symmetric real (or Hermitian complex), or more generally, when A is diagonalizable and has real eigenvalues. In this paper we address the case when A is a large sparse matrix and f is any (continuous) function. This situation precludes a full-fledged diagonalization of A, which may not be practically feasible and numerically viable [29, 30, 22]. A common approach advocated in the literature is to exploit the fact that there is no need to compute the matrix function $f(A)$ if our goal is to compute $f(A)b$. The situation is similar to that of solving linear systems which corresponds to the case when $f(t) = 1/t$. In such situations there is no need to invert A, and efficient methods (e.g., based on Krylov subspaces) can be designed to solve the problem.

The case $f(A) = \exp(A)$ was extensively studied (see, e.g., [38, 17, 23, 24, 5]). Some approaches have also been considered in the literature for a general f, most notably with the use of Krylov subspaces (see, e.g., [46, 47]). In these methods, an Arnoldi process (a Lanczos process in the symmetric case) with an initial vector $b/\left\|b\right\|_2$ yields in k steps a matrix Q_k with k orthonormal columns and a $k \times k$ Hessenberg

*Department of Computer Science and Engineering, University of Minnesota at Twin Cities, Minneapolis, MN 55455. Email: (jchen,saad)@cs.umn.edu. Work of these authors was supported by NSF grant DMS-0810938 and by DOE grant DE-FG-08ER25841. The first author was supported in part by a University of Minnesota Doctoral Dissertation Fellowship.

†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439. Email: anitescu@mcs.anl.gov. Work of this author was supported by the U.S. Department of Energy, through Contract No. DE-AC02-06CH11357.

matrix H_k (tridiagonal in the symmetric case). Then,

$$f_k = \|b\|_2 \, Q_k f(H_k) e_1$$

is a progressive approximation to $f(A)b$ as k increases. It remains to compute $f(H_k)e_1$. Special techniques can be invoked for a few specific cases for $f(H_k)$ [22]. Otherwise, for any f, when a diagonalization of H_k is feasible, the matrix function can be computed via this diagonalization.

The related problem of computing the quadratic form $b^T f(A)b$ was given special attention; see, e.g., [3, 18, 19]. Here, the Lanczos process is often applied to the symmetric positive A. The product $b^T f(A)b$ can be written in a Riemann-Stieltjes integral form, and because of the orthogonality of the Lanczos vectors (or the associated polynomials), the Lanczos procedure applied to this problem will yield an approximation that can be viewed as an estimate of the integral via a quadrature rule.

It is to be noted that for some particularly challenging problems, an unacceptably large Krylov subspace may be required to obtain a satisfactory approximation. This poses difficulties on issues such as storage, computational time, and reorthogonalization costs.[1] For this reason several alternative approaches have also been proposed. The restarted Krylov subspace method [12] restarts the Arnoldi process periodically, to avoid storing large sets of basis vectors which are no longer orthogonal. The approximation $\|b\|_2 \, Q_k f(H_k) e_1$ is shown to converge [12, 1], and the block bidiagonal structure of H_k can be exploited to efficiently update $f(H_k)e_1$ [2]. This method requires predetermining the restart length, which is crucial for the practical performance. The use of standard Krylov subspaces also gave rise to extensions such as shift-and-invert Krylov subspaces [45, 32] and extended Krylov subspaces [11, 27]. The former builds a subspace for the matrix $(I + \gamma A)^{-1}$, where the convergence is mesh independent for A arising from a discretization of differential operators, but the performance is sensitive to the choice of the scaling factor γ (or equivalently the shift $-1/\gamma$). The latter builds a subspace for both A and A^{-1}. It is shown in [11], that to get an equal approximation quality, one needs to take roughly a square root number of iterations as for the standard Lanczos approximation. Both of these variants of Krylov subspace methods require to solve a linear system ($I + \gamma A$ or A) at each iteration, with a different right-hand vector. This can be a major drawback for situations such as when the systems are indefinite and/or originate from 3D meshes. These two methods are special cases of the broad class of *rational approximation* methods which approximate the function f by a rational function p/q, where p and q are two polynomials. A common treatment [15] is to approximate $f(t)$ by $\sum_i w_i/(t - \sigma_i)$, and therefore

$$f(A)b \approx \sum_i w_i (A - \sigma_i I)^{-1} b. \tag{1.1}$$

Explicit formulas of such approximations in the optimal uniform norm for a few special functions, such as the sign function and the inverse square root, are available from Zolotarjov's work (see e.g., [34]). For a general function f, Padé approximations can be carried out by considering f's formal power series, or in other cases the Remez algorithm can be used at a higher computational cost. This rational approximation framework also requires to solve a number of linear systems (with different shifts

[1] In general, these issues are related to the Arnoldi/Lanczos process, and are irrelevant to the matrix function. Remedies of tackling these difficulties are not discussed here.

σ_i). One possible approach to reduce the cost is to simultaneously solve all the shifted systems by using a single Krylov subspace (one that is constructed for A) [15]. Empirical results show that the convergence may sometimes be very slow (or even fail) for systems with complex shifts. Further, by considering the contour integral

$$f(A) = \frac{1}{2\pi i} \int_\Gamma f(z)(zI - A)^{-1}\, dz,$$

the vector $f(A)b$ can be directly computed by performing a quadrature integration, yielding yet another technique based on rational approximation (1.1) [21]. Conformal mappings of the contour Γ have been investigated in order to obtain good quadrature points and to reduce the number of linear solves with $zI - A$ [21]. For further comparisons of the advantages and disadvantages of the above methods, see [42].

A class of methods that avoid solving linear systems are polynomial approximation approaches [10, 31, 33], i.e., to approximate $f(A)b$ by $p(A)b$, where p denotes a polynomial that approximates f in some optimal sense. A common approach is to expand f in a basis of orthogonal polynomials, such as Chebyshev (see, e.g., [5, 10]). Since these expansions are not explicitly known for an arbitrary function, this approach is limited to very specific functions, for example, the exponential. There have been extensions of this basic idea, specifically for the nonsymmetric case, by exploiting asymptotically optimal approximations, using, for example, Faber polynomials or Fejér polynomials, (see, e.g., [31, 33]). There are advantages and disadvantages to these approaches when compared with Krylov methods. Krylov methods are general purpose and require no estimates of eigenvalues. In contrast, methods based on approximation theory usually require determining a set in the real or complex space that contains the spectrum of A. On the other hand, approximation theory methods tend to be effective for certain functions [5] and they are often easier to analyze theoretically. Despite these conceptual differences, we point out that Krylov methods can indeed also be viewed as a form of polynomial approximation, where the approximant polynomial p interpolates the Ritz values (eigenvalues of H_k in the Hermitian case). This viewpoint bridges the divide between the two methods. In a recent article [39] a conjugate residual-type technique was proposed for approximating $f(A)b$, addressing mostly the case when f is an approximate step function. The method can be viewed as a modification of the traditional conjugate residual method for solving linear systems to compute an optimal residual polynomial. It can be adapted to any function f with a finite value at the origin. For a brief discussion of this approach, see Section 6.4.

In this paper, we consider approximating f by a least squares polynomial, based on an idea originating from an unpublished technical report [13]. The proposed method computes an approximant polynomial ϕ_{k+1} of degree k that is close to f with respect to some weighted L^2 norm. Determining the least squares polynomial approximation to f would normally entail computing expansion coefficients that require numerical integration. To avoid this, we first approximate f by a spline function s, which is approximated, in place of f, by the polynomial ϕ_{k+1}. Then, the expansion coefficients are easy to extract without numerical quadrature, provided appropriate weights of the spline approximation are used in each interval. This paper was initially motivated by a statistical sampling problem (see Section 5.1), that leads to a situation where $f(t) = \sqrt{t}$ and A is symmetric positive (semi)-definite. It turns out that a simple technique, possibly one of the best approaches in this case, is to expand f in Legendre polynomials. The coefficients are then known explicitly; see, for example, [28, p. 59]. Here, we do not consider the Legendre polynomials approach because we

want to emphasize *generality*. The method we propose is applicable to any function, not only those that happen to have a known expansion in some orthogonal basis. Indeed, the only requirement for our technique to work well is that the function f be well approximated by a spline.

Error bounds will be established for the proposed method, which suggest that functions that are not differentiable or have large derivatives in the interval will be difficult to approximate by polynomials. This is the case for $f(t) = \sqrt{t}$, which causes difficulties near the origin. Experimental tests demonstrate the capability of the proposed method for this case for large-scale problems. Numerical results indicate promising performance when this problem goes to extreme scales. We also show experimental results for other functions, such as the logarithm and the exponential, demonstrating the wide applicability of the proposed method.

2. Approximating $f(A)b$. A conceptual framework for computing the matrix-vector product $f(A)b$ for a diagonalizable matrix A with real eigenvalues and an arbitrary well-defined function f, is to approximate f by a polynomial. Assume that $\Lambda(A)$, the spectrum of A, is included in some interval $[l, u]$ and that f is defined and continuous on $[l, u]$. Then, f can be approximated to arbitrary accuracy by a polynomial (of large enough degree). In particular, we can readily compute an approximation that is optimal, in the least squares sense, over any polynomial subspace. If an orthonormal basis $\{P_j(t) \mid j = 1, 2, \ldots, k+1\}$ for the subspace is given, then this optimal approximation is

$$f(t) \approx \sum_{j=1}^{k+1} \gamma_j P_j(t), \quad \text{with} \quad \gamma_j = \langle f(t), P_j(t) \rangle ,$$

where $\langle g, h \rangle$ represents the function inner product between g and h associated with a weight w:

$$\langle g, h \rangle = \int_l^u g(t)h(t)w(t)\, dt. \tag{2.1}$$

We will defer the discussion of the choice of the weight function w to the next section. Here, we simply note that, with a proper choice, numerical integration (e.g., by quadrature) can be avoided. The norm of a function g is correspondingly defined as the induced norm from the inner product:

$$\|g(t)\| := \langle g(t), g(t) \rangle^{1/2} . \tag{2.2}$$

The Stieltjes procedure generates such a required basis. Let $\mathbf{1}$ denote the constant function with value 1. With the initial condition $P_0(t) = 0$ and $P_1(t) = \mathbf{1}/\|\mathbf{1}\|$, the Stieltjes procedure computes a sequence of polynomials $P_j(t)$ with the help of a three-term recurrence of the form

$$\beta_{j+1} P_{j+1}(t) = tP_j(t) - \alpha_j P_j(t) - \beta_j P_{j-1}(t), \qquad j = 1, \ldots, k, \tag{2.3}$$

where α_j is the inner product between $tP_j(t)$ and $P_j(t)$ and where β_{j+1} is a normalization coefficient that ensures that $P_{j+1}(t)$, a polynomial of degree j, has unit norm. The resulting polynomials $\{P_j(t)\}$ form an orthonormal basis of the space $\mathbb{P}_{k+1}$ of all polynomials of degree not exceeding k.

Define v_j to be $P_j(A)b$. We can approximate $f(A)b$ as follows:

$$f(A)b \approx \sum_{j=1}^{k+1} \gamma_j P_j(A)b \;=\; \sum_{j=1}^{k+1} \gamma_j v_j. \tag{2.4}$$

Relation (2.3) induces the following three-term recurrence relation for the v_j's:

$$\beta_{j+1} v_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}. \tag{2.5}$$

Formula (2.5) resembles the three-term recurrence of the Lanczos process for generating an orthonormal basis of a Krylov subspace related to a symmetric matrix A. This resemblance is not surprising because the Lanczos algorithm is nothing but a Stieltjes procedure for computing orthogonal polynomials with respect to a discrete inner product over the space of polynomials [19]. In this paper a continuous inner product is used in the Stieltjes procedure; see the next section.

Algorithm 1 shows a detailed procedure for approximating $f(A)b$ via a sequence of vectors $z_j := \sum_{j'=1}^{j} \gamma_{j'} v_{j'}$ based on relations (2.4) and (2.5). The initial basis polynomial $P_1(t)$ is computed in line 2, and the initial vectors v_0 and v_1 are computed in line 3. Then, the first approximant z_1 is computed in lines 4 and 5. In the loop, lines 7 to 10 compute a new basis polynomial $P_{j+1}(t)$ via the three term recurrence, and line 11 computes the corresponding vector v_{j+1}. Then, lines 12 and 13 update the approximant z_j; and, after k steps, the algorithm returns the vector z_{k+1}.

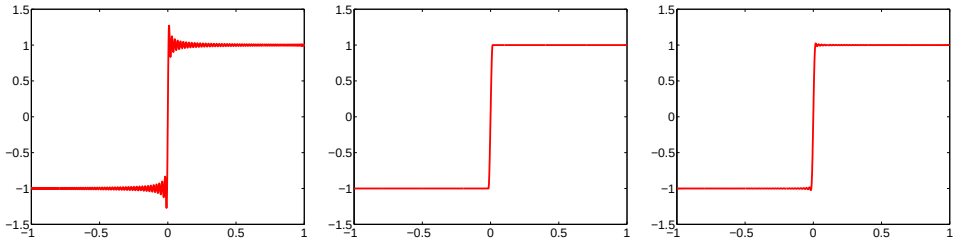
Algorithm 1 Approximating $f(A)b$, initial version

1: $P_0(t) = 0$
2: $S_0(t) = 1$, $\beta_1 = \|S_0(t)\|$, $P_1(t) = S_0(t)/\beta_1$
3: $v_0 = 0$, $v_1 = b/\beta_1$
4: $\gamma_1 = \langle f(t), P_1(t)\rangle$
5: $z_1 = \gamma_1 v_1$
6: **for** $j = 1, \ldots, k$ **do**
7: $\quad \alpha_j = \langle tP_j(t), P_j(t)\rangle$
8: $\quad S_j(t) = tP_j(t) - \alpha_j P_j(t) - \beta_j P_{j-1}(t)$
9: $\quad \beta_{j+1} = \|S_j(t)\|$
10: $\quad P_{j+1}(t) = S_j(t)/\beta_{j+1}$
11: $\quad v_{j+1} = (Av_j - \alpha_j v_j - \beta_j v_{j-1})/\beta_{j+1}$
12: $\quad \gamma_{j+1} = \langle f(t), P_{j+1}(t)\rangle$
13: $\quad z_{j+1} = z_j + \gamma_{j+1} v_{j+1}$
14: **end for**
15: **return** $z_{k+1} \approx f(A)b$

3. Definition and computation of the inner products. Several points remain to be addressed to bring Algorithm 1 into a workable procedure. First, to compute the coefficients α_j, β_{j+1} and γ_{j+1}, we need to define an appropriate weight function w for the inner product (2.1). In addition, it is unlikely that numerical integration can be avoided if f is a truly arbitrary function. Therefore, the idea is to replace f by a spline. The following highlights the strategy proposed in this paper for computing $f(A)b$:

First, approximate the function $f(t)$ by a spline $s(t)$. Then, use $s(t)$ in place of $f(t)$ in Algorithm 1 to compute the vector $z_{k+1} \approx s(A)b$, which in turn approximates $f(A)b$.

Using splines in place of the original f yields many benefits. Since a spline is nothing but a piecewise polynomial, inner products need to be computed on each subinterval only for polynomials. For this, a form of (exact) Gauss-Chebyshev quadrature will allow us to completely bypass numerical integration. In addition, splines can easily be adjusted to handle the problematic situation where the function f has "stiff" regions (see an illustration in Figure 3.1). This section provides the necessary implementation details of an algorithm based on this approach.



(a) (b) (c)

FIG. 3.1. *Polynomial approximation to the sign function* $\mathrm{sgn}(t)$ *and the "spline-smoothed" version of* $\mathrm{sgn}(t)$. (a) *Polynomial approximation to* $\mathrm{sgn}(t)$, *using degree* 300. (b) *Spline approximation to* $\mathrm{sgn}(t)$. *The* -1 *piece and the* 1 *piece are bridged by a cubic polynomial on the interval* $[-0.015, 0.015]$. (c) *Polynomial approximation to the spline in* (b), *using degree* 300. *Note the Gibbs phenomenon exhibited in* (a) *and how it is alleviated in* (c).

In the sequel we consider cubic splines $s(t)$ defined based on the knots $t_0 < t_1 < \cdots < t_{n-1} < t_n$:

$$s(t) := \sum_{i=0}^{n-1} s_i(t), \qquad t \in [l, u],$$

with $l = t_0$ and $t_n = u$, where for each i, the polynomial piece is

$$s_i(t) = \begin{cases} a_i + e_i(t - t_i) + c_i(t - t_i)^2 + d_i(t - t_i)^3 & \text{if } t \in [t_i, t_{i+1}], \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

3.1. Inner product and orthogonal basis. Consider the Chebyshev polynomials of the first kind defined for $|x| \le 1$, by $T_p(x) = \cos(p \cos^{-1} x)$. As is well known, these polynomials satisfy the three-term recurrence $T_{p+1}(x) = 2xT_p(x) - T_{p-1}(x)$, starting with $T_0(x) = 1$ and $T_1(x) = x$. They are also known to constitute a sequence of orthogonal polynomials on the interval $[-1, 1]$ with respect to the weight function $1/\sqrt{1 - x^2}$. If we denote by δ_j the Dirac function $\delta_j = 1$ iff $j = 0$, then we can write[2]

$$\int_{-1}^{1} \frac{T_p(x)T_q(x)}{\sqrt{1 - x^2}}\, dx = \frac{\pi}{2} \left[\delta_{p-q} + \delta_{p+q}\right].$$

[2]The extra term δ_{p+q} takes care of the special situation $p = q = 0$, so the integral is π in this case instead of $\pi/2$. It is not relevant otherwise.

For an interval $[t_i, t_{i+1}]$, we perform the change of variable

$$x^{(i)}(t) = \frac{2}{t_{i+1} - t_i} t - \frac{t_{i+1} + t_i}{t_{i+1} - t_i},$$

and define the polynomials

$$C_p^{(i)}(t) := T_p(x^{(i)}(t)), \qquad t \in [t_i, t_{i+1}], \tag{3.2}$$

and the inner product

$$\langle g(t), h(t) \rangle_{[t_i, t_{i+1}]} := \int_{t_i}^{t_{i+1}} \frac{g(t)h(t)}{\sqrt{(t - t_i)(t_{i+1} - t)}}\, dt. \tag{3.3}$$

This change of variable has a significant implication: The $C_p^{(i)}(t)$'s are orthogonal on the interval $[t_i, t_{i+1}]$ with respect to the inner product defined above, namely,

$$\left\langle C_p^{(i)}(t), C_q^{(i)}(t) \right\rangle_{[t_i, t_{i+1}]} = \frac{\pi}{2} \left[\delta_{p-q} + \delta_{p+q}\right] \ . \tag{3.4}$$

With (3.3), we define the inner product of g and h on the whole interval $[l, u]$ as follows

$$\langle g(t), h(t) \rangle_{[l,u]} := \sum_{i=0}^{n-1} \langle g(t), h(t) \rangle_{[t_i, t_{i+1}]} .$$

The subscript $[l, u]$ will be dropped in the rest of the paper in order to conform with the notation in (2.1); that is, when we use the notation $\langle \cdot, \cdot \rangle$, we always mean the inner product on the whole interval $[l, u]$. The corresponding norm of a function g on the interval $[t_i, t_{i+1}]$ is $\|g(t)\|_{[t_i, t_{i+1}]} := \langle g(t), g(t) \rangle^{1/2}_{[t_i, t_{i+1}]}$, and the overall norm $\|\cdot\|$ on the interval $[l, u]$ as defined in (2.2) satisfies

$$\|g(t)\|^2 = \sum_{i=0}^{n-1} \|g(t)\|^2_{[t_i, t_{i+1}]} .$$

3.2. Computing α_j, β_{j+1}, and γ_{j+1}. With the definition of an inner product on each interval, we can exploit the orthogonality of the basis (cf. Equation (3.4)) to efficiently compute the inner products in Algorithm 1 by expanding the involved functions using the basis *in each interval*. This redundancy allows to bypass numerical integration by using what amounts to a Gauss-Chebyshev quadrature. The three-term recurrence (2.3) along with the standard relations of Chebyshev polynomials allows us to update the required expansions. This approach was described in [37] for the case of two intervals. For completeness, we briefly discuss the derivation of the formulas to compute α_j, β_{j+1}, and γ_{j+1} for the general case.

First we consider the computation of α_j. Let $P_j(t)$ be expressed in $[t_i, t_{i+1}]$ as

$$P_j(t) = \sum_{p=0}^{j-1} \mu_{pj}^{(i)} C_p^{(i)}(t), \qquad j \geq 1.$$

For now we assume that the coefficients $\mu_{pj}^{(i)}$ are known; an update formula will be derived later. We need to compute $\langle tP_j, P_j \rangle$, and we start by noting that $tP_j(t) =$

$\sum_{p=0}^{j-1} \mu_{pj}^{(i)} t C_p^{(i)}(t)$. The usual recurrence relation of the Chebyshev polynomials shows that, in the interval $[t_i, t_{i+1}]$,

$$\begin{aligned} tC_p^{(i)}(t) &= \frac{t_{i+1}-t_i}{4} C_{p+1}^{(i)}(t) + \frac{t_{i+1}+t_i}{2} C_p^{(i)}(t) + \frac{t_{i+1}-t_i}{4} C_{p-1}^{(i)}(t), \qquad p \geq 1, \\ tC_0^{(i)}(t) &= \frac{t_{i+1}-t_i}{2} C_1^{(i)}(t) + \frac{t_{i+1}+t_i}{2} C_0^{(i)}(t). \end{aligned} \tag{3.5}$$

With the conventions that $\mu_{-1,j}^{(i)} = 0$ and $\mu_{pj}^{(i)} = 0$ for all $p \geq j$, this leads to

$$tP_j(t) = \frac{t_{i+1}-t_i}{4} \mu_{0j}^{(i)} C_1^{(i)}(t) + \sum_{p=0}^{j} \left(\frac{t_{i+1}-t_i}{4} \left(\mu_{p-1,j}^{(i)} + \mu_{p+1,j}^{(i)} \right) + \frac{t_{i+1}+t_i}{2} \mu_{pj}^{(i)} \right) C_p^{(i)}(t).$$

We define

$$\sigma_{pj}^{(i)} := \frac{t_{i+1}-t_i}{4} \left(\mu_{p-1,j}^{(i)} + \mu_{p+1,j}^{(i)} \right) + \frac{t_{i+1}+t_i}{2} \mu_{pj}^{(i)}, \qquad p = 0, \ldots, j. \tag{3.6}$$

Then $\alpha_j = \langle tP_j(t), P_j(t) \rangle = \sum_{i=0}^{n-1} \langle tP_j(t), P_j(t) \rangle_{[t_i, t_{i+1}]}$ becomes

$$\begin{aligned} \alpha_j &= \sum_{i=0}^{n-1} \left\langle \frac{t_{i+1}-t_i}{4} \mu_{0j}^{(i)} C_1^{(i)}(t) + \sum_{p=0}^{j} \sigma_{pj}^{(i)} C_p^{(i)}(t), \sum_{p=0}^{j-1} \mu_{pj}^{(i)} C_p^{(i)}(t) \right\rangle_{[t_i, t_{i+1}]} \\ &= \pi \sum_{i=0}^{n-1} \left(\sigma_{0j}^{(i)} \mu_{0j}^{(i)} + \frac{t_{i+1}-t_i}{8} \mu_{0j}^{(i)} \mu_{1j}^{(i)} + \frac{1}{2} \sum_{p=1}^{j} \sigma_{pj}^{(i)} \mu_{pj}^{(i)} \right). \end{aligned} \tag{3.7}$$

We now consider the computation of β_{j+1} starting with the case $j = 0$. Recall that in Algorithm 1 we define $S_j(t) = tP_j(t) - \alpha_j P_j(t) - \beta_j P_{j-1}(t)$. It is easy to see from (2.3) that $\beta_1^2 = \|S_0(t)\|^2 = \sum_{i=0}^{n-1} \|C_0(t)\|_{[t_i, t_{i+1}]}^2 = n\pi$. For $j \geq 1$, we have

$$\begin{aligned} \beta_{j+1}^2 &= \|S_j(t)\|^2 = \|tP_j(t) - \alpha_j P_j(t) - \beta_j P_{j-1}(t)\|^2 \\ &= \sum_{i=0}^{n-1} \left\| \frac{t_{i+1}-t_i}{4} \mu_{0j}^{(i)} C_1^{(i)}(t) + \sum_{p=0}^{j} \sigma_{pj}^{(i)} C_p^{(i)}(t) - \right. \\ &\qquad \left. \alpha_j \sum_{p=0}^{j-1} \mu_{pj}^{(i)} C_p^{(i)}(t) - \beta_j \sum_{p=0}^{j-2} \mu_{p,j-1}^{(i)} C_p^{(i)}(t) \right\|_{[t_i, t_{i+1}]}^2 \\ &= \sum_{i=0}^{n-1} \left\| \frac{t_{i+1}-t_i}{4} \mu_{0j}^{(i)} C_1^{(i)}(t) + \sum_{p=0}^{j} \left(\sigma_{pj}^{(i)} - \alpha_j \mu_{pj}^{(i)} - \beta_j \mu_{p,j-1}^{(i)} \right) C_p^{(i)}(t) \right\|_{[t_i, t_{i+1}]}^2 . \end{aligned}$$

We define

$$\eta_{pj}^{(i)} := \sigma_{pj}^{(i)} - \alpha_j \mu_{pj}^{(i)} - \beta_j \mu_{p,j-1}^{(i)}, \qquad p = 0, \ldots, j. \tag{3.8}$$

Then

$$\beta_{j+1} = \sqrt{\pi \sum_{i=0}^{n-1} \left[{\eta_{0j}^{(i)}}^2 + \frac{1}{2} \left(\eta_{1j}^{(i)} + \frac{t_{i+1}-t_i}{4} \mu_{0j}^{(i)} \right)^2 + \frac{1}{2} \sum_{p=2}^{j} {\eta_{pj}^{(i)}}^2 \right]}. \tag{3.9}$$

Furthermore, since $P_{j+1}(t) = S_j(t)/\beta_{j+1}$, we have the following update formula for $\mu^{(i)}_{p,j+1}$:

$$\mu^{(i)}_{p,j+1} = \begin{cases} \eta^{(i)}_{pj}/\beta_{j+1} & \text{if } p = 0, 2, 3, \ldots, j, \\ \left[\eta^{(i)}_{1j} + \frac{t_{i+1}-t_i}{4}\mu^{(i)}_{0j}\right]/\beta_{j+1} & \text{if } p = 1. \end{cases} \tag{3.10}$$

The initial condition is $\mu^{(i)}_{01} = 1/\beta_1$ for all i, since $P_1(t) = C_0(t)/\beta_1$.

Now consider γ_{j+1}. Since we use $s(t)$ in place of $f(t)$ in Algorithm 1, we have

$$\gamma_{j+1} = \langle s(t), P_{j+1}(t)\rangle = \sum_{i=0}^{n-1} \left\langle s_i(t), \sum_{p=0}^{j} \mu^{(i)}_{p,j+1} C^{(i)}_p(t) \right\rangle_{[t_i,t_{i+1}]}.$$

Note that $s_i(t)$ is a cubic polynomial; therefore, we have the expansion

$$s_i(t) = \xi^{(i)}_0 C^{(i)}_0(t) + \xi^{(i)}_1 C^{(i)}_1(t) + \xi^{(i)}_2 C^{(i)}_2(t) + \xi^{(i)}_3 C^{(i)}_3(t),$$

where $h_i = (t_{i+1} - t_i)/2$, and

$$\begin{aligned} \xi^{(i)}_0 &= \frac{5}{2}d_i h_i^3 + \frac{3}{2}c_i h_i^2 + e_i h_i + a_i, & \xi^{(i)}_2 &= \frac{3}{2}d_i h_i^3 + \frac{1}{2}c_i h_i^2, \\ \xi^{(i)}_1 &= \frac{15}{4}d_i h_i^3 + 2c_i h_i^2 + e_i h_i, & \xi^{(i)}_3 &= \frac{1}{4}d_i h_i^3. \end{aligned} \tag{3.11}$$

Thus,

$$\gamma_{j+1} = \pi \sum_{i=0}^{n-1} \left(\xi^{(i)}_0 \mu^{(i)}_{0,j+1} + \frac{1}{2} \sum_{p=1}^{\min\{j,3\}} \xi^{(i)}_p \mu^{(i)}_{p,j+1} \right). \tag{3.12}$$

3.3. The final algorithm. Algorithm 2 is the final algorithm that incorporates the details just discussed. The orthogonal polynomials $P_j(t)$ do not appear explicitly. They are represented by their expansion coefficients in each interval. The scalars α_j, β_{j+1} and γ_{j+1} are now computed via the expansion coefficients $\sigma^{(i)}_{pj}$, $\eta^{(i)}_{pj}$, $\xi^{(i)}_p$, and $\mu^{(i)}_{pj}$. This approach avoids numerical integration, and the updates of the coefficients are simple. The only operation with the matrix A takes place in line 14 and involves the product of A with the vector v_j. Also, the function f is used only at the beginning of the algorithm, when an interpolating spline $s(t)$ is computed.

Let us analyze the computational costs of Algorithm 2, assuming that A is a sparse matrix of size $m \times m$. The time cost includes the time to perform a spline interpolation, which is linear in the number of knots n. The main body of the algorithm starting from line 2 has a cost of $O(k(kn + m + T_A))$, where T_A is the time for computing a matrix-vector product between A and any right-hand vector. For each iteration, the portion kn comes from computing α_j, β_{j+1}, γ_{j+1}, and other coefficients, and the portion m comes from the length-m vector operations in lines 14 and 16.

Memory costs are likely to be dominated by the storage of the matrix A, although there are applications where the matrix is used only in operator form. For the storage of other variables in the algorithm, note that only three m-dimensional vectors are needed for the v_j's and one vector for the latest z_j, leading to a total of $4m$ storage locations. In addition, we need to use $3kn$ locations to store $\mu^{(i)}_{p,j+1}$, $\mu^{(i)}_{pj}$, and $\mu^{(i)}_{p,j-1}$;

Algorithm 2 Approximating $f(A)b$, final version

Input: $t_0, t_1, \ldots, t_n$, where $t_0 = l$, $t_n = u$ and $t_i < t_{i+1}$ for all i
Input: number of steps k
1: Compute a cubic spline $s(t)$ which interpolates points $(t_i, f(t_i))$ for $i = 0, \ldots, n$, i.e., compute the coefficients a_i, e_i, c_i, d_i in (3.1).
2: $\beta_1 = \sqrt{n\pi}$
3: $v_1 = b/\beta_1$
4: Compute $\xi_p^{(i)}$ for $i = 0, \ldots, n-1$ and $p = 0, \ldots, 3$ using (3.11)
5: $\mu_{01}^{(i)} = 1/\beta_1$ for $i = 0, \ldots, n-1$
6: $\gamma_1 = \pi \sum_{i=0}^{n-1} \xi_0^{(i)} \mu_{01}^{(i)}$
7: $z_1 = \gamma_1 v_1$
8: **for** $j = 1, \ldots, k$ **do**
9: Compute $\sigma_{pj}^{(i)}$ for $i = 0, \ldots, n-1$ and $p = 0, \ldots, j$ using (3.6)
10: Compute α_j using (3.7)
11: Compute $\eta_{pj}^{(i)}$ for $i = 0, \ldots, n-1$ and $p = 0, \ldots, j$ using (3.8)
12: Compute β_{j+1} using (3.9)
13: Compute $\mu_{p,j+1}^{(i)}$ for $i = 0, \ldots, n-1$ and $p = 0, \ldots, j$ using (3.10)
14: $v_{j+1} = (Av_j - \alpha_j v_j - \beta_j v_{j-1})/\beta_{j+1}$
15: Compute γ_{j+1} using (3.12)
16: $z_{j+1} = z_j + \gamma_{j+1} v_{j+1}$
17: **end for**
18: **return** $z_{k+1} \approx f(A)b$

kn locations to store $\sigma_{pj}^{(i)}$; and kn locations to store $\eta_{pj}^{(i)}$. We also need $4n$ locations to store $\xi_p^{(i)}$ and n locations to store h_i. These costs are summarized as $5kn + 5n + 4m$ locations.

So far, we have not discussed how to set t_i (which also determines the size of n). The choice of the knots is experimental and dependent on the function. A general guideline is that when the function derivative is large, we use short subintervals. This is important in order to obtain an accurate spline and a good least squares polynomial approximation to it. In Section 5, we propose a scheme that empirically works well for functions such as the square root and the logarithm. Also in Section 6.5 we briefly mention a scheme for the exponential function.

4. Convergence analysis. In Algorithm 2, we approximate $f(t)$ on the interval $[l, u]$ by a cubic spline $s(t)$, and we project $s(t)$ onto the polynomial space $\mathbb{P}_{k+1}$, which consists of polynomials of degree not exceeding k:

$$s(t) \approx \sum_{j=1}^{k+1} \gamma_j P_j(t) := \phi_{k+1}(t), \tag{4.1}$$

where $\{P_j(t)\}$ is an orthonormal basis of $\mathbb{P}_{k+1}$, with

$$\gamma_j = \langle s(t), P_j(t) \rangle .$$

The approximant is

$$z_{k+1} := \phi_{k+1}(A)b.$$

By way of introduction we present the following easy-to-prove or well-known results.

PROPOSITION 4.1. *For the norm $\|\cdot\|$ defined earlier (cf. Equation (2.2) and Section 3.1),*

$$\phi_{k+1}(t) = \arg\min_{\phi\in\mathbb{P}_{k+1}} \|\phi(t) - s(t)\|. \tag{4.2}$$

In addition, if A is symmetric and its spectrum is included in $[l, u]$, then

$$\|z_{k+1} - f(A)b\|_2 \le \max_{t\in[l,u]} |\phi_{k+1}(t) - f(t)| \, \|b\|_2. \tag{4.3}$$

Proof. The first result is well known for least squares approximations; see e.g., [36]. The second comes from the fact that A can be written as $A = VDV^T$, where V is unitary, and follows trivially by expanding $z_{k+1} - f(A)b = (\phi_{k+1}(A) - f(A))b$ in the eigenbasis. ∎

To bound the difference between $\phi_{k+1}(t)$ and $f(t)$ on the interval $[l, u]$, note that

$$|\phi_{k+1}(t) - f(t)| \le |\phi_{k+1}(t) - s(t)| + |s(t) - f(t)|.$$

Hence, we need to estimate the two terms on the right-hand side of the above inequality separately. For the second term, many known error bounds for splines can be exploited. The following presents a standard result for clamped cubic splines, which indicates a fourth-order accuracy.

THEOREM 4.2 ([41, pp. 57–58]). *If $f(t)$ is fourth-order differentiable on the interval $[l, u]$ and if $s(t)$ is the unique cubic spline that interpolates $f(t)$ on the knots $l = t_0 < t_1 < \cdots < t_n = u$ with the boundary condition*

$$s'(t_0) = f'(t_0) \qquad \text{and} \qquad s'(t_n) = f'(t_n),$$

then

$$\max_{t\in[l,u]} |s(t) - f(t)| \le \frac{5M}{384} \max_{0\le i\le n-1} (t_{i+1} - t_i)^4,$$

where $M = \max_{t\in[l,u]} \left|f^{(4)}(t)\right|$.

To bound the difference $|\phi_{k+1}(t) - s(t)|$, we need the following two lemmas. They are extensions of similar results given in [37], and hence the proofs are omitted.

LEMMA 4.3. *Using the notation of a function norm $\|\cdot\|$ in this paper, we have*

$$\|g(t)\| \le \sqrt{n\pi} \cdot \max_{t\in[l,u]} |g(t)|.$$

LEMMA 4.4. *Let $g_{k+1}(t) \in \mathbb{P}_{k+1}$ be any polynomial of degree not exceeding k. Then, using the notation of a function norm $\|\cdot\|$ in this paper, we have*

$$\max_{t\in[l,u]} |g_{k+1}(t)| \le \sqrt{\frac{2(k+1)}{\pi}} \, \|g_{k+1}(t)\|.$$

By a property of the uniform norm, for any continuous function $g(t)$, there exists a degree-k polynomial $g^*_{k+1}(t)$ such that

$$\max_{t\in[l,u]} \left|g^*_{k+1}(t) - g(t)\right| \le \max_{t\in[l,u]} |\phi(t) - g(t)|, \qquad \forall \phi \in \mathbb{P}_{k+1}.$$

The *modulus of continuity* of a function $g(t)$ on the interval $[l,u]$ is defined for all $\delta > 0$,

$$\omega(g;[l,u];\delta) := \sup_{\substack{t_1,t_2\in[l,u]\\ |t_1-t_2|\le\delta}} |g(t_1)-g(t_2)|\,.$$

We use the shorthand notation $\omega(\delta)$ when the context is clear. We also use ω_r to denote the modulus of continuity of the r-th derivative of g:

$$\omega_r(g;[l,u];\delta) := \omega(g^{(r)};[l,u];\delta).$$

The following is a corollary of Jackson's theorem for bounding the uniform approximation of a function g.

LEMMA 4.5 ([36, Theorem 1.5, p. 23]). *If g has an r-th derivative on $[l,u]$, then for $k > r$,*

$$\max_{t\in[l,u]} \left|g^*_{k+1}(t) - g(t)\right| \le \frac{C_r}{k^r}\omega_r\left(\frac{u-l}{2(k-r)}\right),$$

where $C_r = 6^{r+1}e^r(1+r)^{-1}$.

The above lemmas lead to the following theorem, which gives an upper bound for the convergence rate of $\phi_{k+1}(t)$ to $s(t)$.

THEOREM 4.6. *For $r = 0, 1, \ldots, 3$, the uniform norm of the residual polynomial admits the following bounds:*

$$\max_{t\in[l,u]} |\phi_{k+1}(t) - s(t)| \le \frac{(2\sqrt{2n(k+1)}+1)C_r}{k^r}\omega_r\left(\frac{u-l}{2(k-r)}\right), \tag{4.4}$$

where $C_r = 6^{r+1}e^r(1+r)^{-1}$ and $k > r$.

Proof. We have

$$\max_{t\in[l,u]} |\phi_{k+1}(t) - s(t)| \le \max_{t\in[l,u]} \left|\phi_{k+1}(t) - s^*_{k+1}(t)\right| + \max_{t\in[l,u]} \left|s^*_{k+1}(t) - s(t)\right|. \tag{4.5}$$

From Lemma 4.4,

$$\max_{t\in[l,u]} \left|\phi_{k+1}(t) - s^*_{k+1}(t)\right| \le \sqrt{\frac{2(k+1)}{\pi}}\left\|\phi_{k+1}(t) - s^*_{k+1}(t)\right\|. \tag{4.6}$$

Since

$$\begin{aligned}\left\|\phi_{k+1}(t) - s^*_{k+1}(t)\right\| &\le \|\phi_{k+1}(t) - s(t)\| + \left\|s(t) - s^*_{k+1}(t)\right\| \\ &\le \left\|s^*_{k+1}(t) - s(t)\right\| + \left\|s(t) - s^*_{k+1}(t)\right\| \qquad \text{(by Eqn. (4.2))} \\ &= 2\left\|s^*_{k+1}(t) - s(t)\right\|, \end{aligned} \tag{4.7}$$

the inequality (4.6) becomes

$$\max_{t\in[l,u]} \left|\phi_{k+1}(t) - s^*_{k+1}(t)\right| \le 2\sqrt{\frac{2(k+1)}{\pi}}\left\|s^*_{k+1}(t) - s(t)\right\|.$$

Recall from Lemma 4.3 that $\left\|s^*_{k+1}(t) - s(t)\right\| \le \sqrt{n\pi}\max_{t\in[l,u]}\left|s^*_{k+1}(t) - s(t)\right|$. Therefore,

$$\max_{t\in[l,u]} \left|\phi_{k+1}(t) - s^*_{k+1}(t)\right| \le 2\sqrt{2n(k+1)}\max_{t\in[l,u]}\left|s^*_{k+1}(t) - s(t)\right|.$$

Thus, (4.5) becomes

$$\max_{t\in[l,u]} |\phi_{k+1}(t) - s(t)| \le (2\sqrt{2n(k+1)}+1) \max_{t\in[l,u]} \left|s^*_{k+1}(t) - s(t)\right|.$$

The theorem is established by applying Lemma 4.5. □

A function $g(t)$ defined on $[l,u]$ is ν-Lipschitz with constant K if

$$|g(t_1) - g(t_2)| \le K\,|t_1 - t_2|^{\nu}, \qquad \forall t_1, t_2 \in [l,u].$$

We have the following corollary for Theorem 4.6.

COROLLARY 4.7. *Let the r-th derivative of $s(t)$ be ν-Lipschitz with constant K_r, for $r = 0,\ldots,3$. Then, for $k > r$,*

$$\max_{t\in[l,u]} |\phi_{k+1}(t) - s(t)| \le \frac{(2\sqrt{2n(k+1)}+1)(u-l)^{\nu} C_r K_r}{2^{\nu} k^r (k-r)^{\nu}} = O\left(\frac{1}{k^{r+\nu-1/2}}\right), \quad (4.8)$$

where $C_r = 6^{r+1} e^r (1+r)^{-1}$.

The sub-linear convergence $O(1/k^{r+\nu-1/2})$ given by (4.8) results from Jackson's theorem which shows a C_r/k^r factor on the convergence of polynomial approximation in the uniform norm (Lemma 4.5). When the function is infinitely differentiable on the interval, r can be any positive integer, thus by putting $k = r+1$, we have

$$\frac{C_r}{k^r} = \frac{1}{e}\left(\frac{6e}{r+1}\right)^{r+1}.$$

When r is large enough such that $6e/(r+1)$ is less than some predefined constant c, then $C_r/k^r < e^{-1}c^{r+1}$, which effectively indicates a linear convergence (assuming that K_r is uniformly bounded or increases no faster than exponentially with r). In our situation, the function is a cubic spline, which does not have a 4-th or higher order derivative. This unfortunately restricts the value of r not being larger than 3.

The bound (4.8) suggests that the conditioning of the matrix will affect the approximation for some functions such as the square root. This makes scaling of the matrix not viable—either close to or far away from the origin is the smallest eigenvalue of the matrix, the factor $K_r(u-l)^{\nu}$ will be large if the matrix is ill-conditioned. The asymptotic behavior of the bound also fits most of the observed situations—in the log-log scale, the uniform norm decays like a straight line. In other words, empirically, we can fit some constants $c_1 > 0$ and c_2 such that

$$\log\left(\max|\phi_{k+1}(t) - s(t)|\right) \approx -c_1 \log k + c_2.$$

Experiments in Section 6.2 yield $1 < c_1 < 2$, which corresponds to $r = 1$. Nevertheless, the bound (4.8) may not be tight enough in some cases. In Section 6.6, we show that for a covariance matrix resulting from a statistical application, the uniform norm converges linearly, much faster than the sub-linear rate indicated by the bound. This may be because of the good conditioning of the involved matrix, and this interesting fact will be a topic of future investigation.

The sub-linear convergence $O(k^{-r-\nu+1/2})$ might make the proposed method appear inferior to other methods such as those based on extended Krylov subspaces and contour integrals, which show at least a linear convergence. However, this convergence rate does not take into account the efforts for solving a linear system at each step, which is much harder and much more expensive than performing one matrix-vector

multiplication. As mentioned in the introduction, solving linear systems related to A or shifted systems can be a major hurdle. For example for problems originating from large 3D meshes, direct solvers may not even be feasible due to prohibitive memory requirements. If iterative methods are used then one must remember that the matrices are shifted and can be indefinite, making the systems harder to solve by iterative solvers. In contrast, the method proposed in this paper requires only one matrix-vector multiplication per iteration.

Apart from the above result for the uniform norm of $\phi_{k+1}(t) - s(t)$, we can also give a bound for its norm.

THEOREM 4.8. *For $r = 0, 1, \ldots, 3$, the norm of the residual polynomial admits the following bounds:*

$$\|\phi_{k+1}(t) - s(t)\| \leq \frac{3C_r\sqrt{n\pi}}{k^r}\omega_r\left(\frac{u-l}{2(k-r)}\right),$$

where $C_r = 6^{r+1}e^r(1+r)^{-1}$ and $k > r$.

Proof. This follows from

$$\begin{aligned}
\|\phi_{k+1}(t) - s(t)\| &\leq \left\|\phi_{k+1}(t) - s^*_{k+1}(t)\right\| + \left\|s^*_{k+1}(t) - s(t)\right\| \\
&\leq 3\left\|s^*_{k+1}(t) - s(t)\right\| && \text{(by (4.7))} \\
&\leq 3\sqrt{n\pi}\max_{t\in[l,u]}\left|s^*_{k+1}(t) - s(t)\right| && \text{(by Lemma 4.3)} \\
&\leq \frac{3C_r\sqrt{n\pi}}{k^r}\omega_r\left(\frac{u-l}{2(k-r)}\right). && \text{(by Lemma 4.5)}
\end{aligned}$$

□

From the above result one can trivially obtain a bound analogous to that of Corollary 4.7 for the case when the r-th derivative of $s(t)$ is ν-Lipschitz with constant K_r:

$$\|\phi_{k+1}(t) - s(t)\| \leq \frac{3C_rK_r\sqrt{n\pi}(u-l)^\nu}{2^\nu k^r(k-r)^\nu} = O\left(\frac{1}{k^{r+\nu}}\right). \tag{4.9}$$

5. Application: computing $A^{1/2}b$. We consider a case where $f(t)$ is the square root function and A is symmetric positive definite.[3] We note that the symmetry requirement of A is not necessary; A only needs to be diagonalizable with all its eigenvalues real and positive. Further, the positive definiteness requirement of A can be relaxed to positive semi-definiteness. This will affect only the choice of the interval $[l, u]$ (cf. Section 5.2), which needs to contain only the nonzero eigenvalues of A.

5.1. Background and challenge. Sampling from a multivariate Gaussian distribution with a positive definite covariance matrix $K \in \mathbb{R}^{m\times m}$ is one of the most common endeavors in statistics. The most common approach is to compute the Cholesky factorization $K = LL^T$, where L is a lower triangular matrix. If x is a vector whose entries are independent and are normally distributed with mean 0 and variance 1, that is, $x \sim \mathcal{N}(\mathbf{0}_m, \mathbb{I}_m)$, then $\zeta = \mathbf{m} + Lx$ is a random variable whose distribution is $\mathcal{N}(\mathbf{m}, K)$. Many modern applications often require high-fidelity spatio temporal

[3] Under the same context, the application also requires computing $f(A)b$ for f being the logarithm; see the background description in the next subsection. To illustrate the effectiveness of our general purpose algorithm, in this paper we mainly focus on the square root example. The case of logarithm is similar and is only briefly mentioned.

sampling, which puts m in the range of 10^{12}–10^{15}. This results in the need to identify sampling approaches that have both $O(m)$ complexity and high potential for parallelism.

If the sample sites are on a regular grid and the covariance function is stationary, then several techniques can be used to sample efficiently from the Gaussian distribution. One can use specialized linear algebra to carry out the Cholesky factorization, at least for some one-dimensional problems for a number of sampling sites up to 10^6 [16]. Other possibilities are to use a multigrid-type sampling scheme, Galerkin multigrid Monte Carlo (MGMC) [20], or to embed the covariance matrix in a stationary periodic process [9], followed by a fast Fourier transformation (FFT) technique.

Nevertheless, none of these approaches was demonstrated to work for the case where the data points are not on a regular grid, or for nonstationary covariance functions, or on the scale of problems that we aim to solve. In the case of the structured Cholesky approach of [16], it is unclear whether an indexing can be found that will result in sufficient sparsity of the factors for the non regular grid, nonstationary covariance function, or multiple dimensions. In addition, at extremely large numbers of sites, Cholesky factorization cannot be expected to be as efficient to parallelize as a matrix-free approach. The FFT approach may be difficult to parallelize beyond a thousand processors [6]. More important perhaps, even for small processor counts, FFT cannot be applied when the sampling sites are not on a regular grid or the Gaussian process is not stationary. For MGMC, the compact kernel has much larger bandwidth than do the covariance matrices for which they are traditionally applied, which are of the Laplace-matrix type [20]. This situation may result in rapid densification and increase in storage requirements [14] [44, §7.7.5].

Many examples of interest need to sample the Gaussian process at points that cannot be easily embedded in a regular grid. For example, the positions of windfarms or their wind turbines cannot be easily approximated with a grid, unless the grid cell is exceedingly small [7]. In geostationary applications, the spherical shape of the Earth prevents most spatial grids of interest from being regular when projected on a plane. Moreover, there are countless examples of nonstationary Gaussian processes of interest [43, 35] for which FFT cannot work, and neither Cholesky nor MGMC approaches have been demonstrated to work either.

We therefore turn to an entirely matrix-free approach for computing $K^{1/2}x$. If $x \sim \mathcal{N}(\mathbf{0}_m, \mathbb{I}_m)$ then $\eta = \mathbf{m} + K^{1/2}x$ is a random variable whose distribution is also $\mathcal{N}(\mathbf{m}, K)$. We thus achieve a matrix-free approach of sampling from an arbitrary normal distribution, irrespective of the positions of the sampling sites or the lack of stationarity in the covariance function that generates K.

A related problem under the same context is fitting a Gaussian process. The most common procedure for determining the Gaussian process out of a parametric class that best fits a data set is to use the maximum likelihood approach [43, 35]. The likelihood has two terms that need to be evaluated. One, $y^T K^{-1} y$, where y is the data vector, can be readily computed using an efficient approach such as conjugate gradient. The other term is the log-determinant, $\log(\det(K))$. This is possibly the source of most difficulties in computing exact log-likelihoods for a large number of data points. One approach is to use the identity $\log(\det(K)) = \mathrm{tr}(\log(K))$ [48, 4]. Subsequently, the trace is estimated based on the relationship $\mathrm{tr}(\log(K)) = \mathrm{E}[u^T \log(K) u]$ and the Monte Carlo sample average approximation $\mathrm{E}[u^T \log(K) u] \approx \frac{1}{N} \sum_{i=1}^{N} {u^i}^T \log(K) u^i$ (also known as the Hutchinson esitmator of the trace [25]). Here u is a random vector with independent components, each of which takes the values ± 1 with probability 0.5.

The vectors u^i are independent random vectors drawn from this distribution. The approaches in [48, 4] differ in the approximations used for computing $\log(K)u^i$. In this work, we can use the proposed general approach for computing $f(A)b$ to compute the action of $\log(K)$ on one of the vectors u^i.

Nevertheless, in order for the approach to have an $O(m)$ behavior, the matrix-vector multiplications must take $O(m)$ themselves. Therefore the covariance matrix K must be sparse. We will thus be interested primarily in covariance matrices originating in Gaussian processes with compact kernels. Such processes are widely used in applications and result in sparse covariance matrices [35]. In addition, it is sometimes possible to replace K by the covariance matrix obtained from a compact kernel with little bias or loss of statistical efficiency compared to the original covariance function [26, 16].

5.2. The interval $[l, u]$. We now consider further details to carry out Algorithm 2 specifically for the square root function. One issue is the interval $[l, u]$, which needs to contain the spectrum of A. By Theorem 4.2, the subintervals $[t_i, t_{i+1}]$ should be small enough to yield an accurate spline $s(t)$. However, too many intervals will impose a heavy computational burden, for both the spline approximation and the computation of the coefficients α_j, β_{j+1} and γ_{j+1}. On the other hand, the closer zero is to the interval, the harder it is to interpolate $\sqrt{t}$, since the derivative tends to infinity. A geometric progression of the spacing between the knots works well in practice, so we opt to let

$$t_0 = \lambda_{\min}/(1+a), \qquad t_i = (1+a)^i t_0, \qquad i = 1, 2, \ldots, n,$$

and to let t_n be some value such that $t_n \geq \lambda_{\max}$, where $\lambda_{\min}$ and $\lambda_{\max}$ are the smallest and the largest eigenvalue of A, respectively. From this we have

$$n = \left\lceil \log_{1+a} \frac{\lambda_{\max}}{\lambda_{\min}} \right\rceil + 1 = \left\lceil \log_{1+a} \kappa \right\rceil + 1,$$

where κ is the 2-norm condition number of A. We choose $a = 0.01$. Note that this interval scheme requires an estimate of the two extreme eigenvalues of A. We avoid using $t_0 = \lambda_{\min}$ when the estimate is not accurate enough.

5.3. Convergence test. As shown in the analysis in Section 4, the convergence of z_{k+1} to $f(A)b$ can be split in two parts: the convergence of the spline $s(t)$ to $f(t)$ and the convergence of the least squares approximation $\phi_{k+1}(t)$ to $s(t)$. The spline is usually a sufficiently good approximation to $f(t)$ if one uses the intervals designed in Section 5.2. Therefore, we consider mainly the approximation quality of $\phi_{k+1}(t)$.

As discussed after Corollary 4.7, the bound (4.8) is not useful for determining an appropriate k. Instead, a natural heuristic is to consider the (relative) difference of consecutive iterates

$$\frac{\|z_{k+1} - z_k\|_2}{\|z_{k+1}\|_2} \tag{5.1}$$

and ensure that it falls below a certain tolerance ϵ. Note it is possible that a small difference might identify a stagnation of the approximation rather than actual convergence (see, e.g., [15]).

Another possible criterion is to check the norm of the residual polynomial:

$$\frac{\|s(t) - \phi_{k+1}(t)\|}{\|s(t)\|}. \tag{5.2}$$

Lemmas 4.3 and 4.4 suggest that the norm of $s(t) - \phi_{k+1}(t)$ should decay similarly to that of the uniform norm. Since $\phi_{k+1}(t)$ is a least squares approximation, we have

$$\|s(t) - \phi_{k+1}(t)\|^2 = \|s(t)\|^2 - \|\phi_{k+1}(t)\|^2 .$$

Therefore, the relative difference (5.2) can be easily computed by noting that

$$\|s(t)\|^2 = \sum_{i=0}^{n} \left[\left(\xi_0^{(i)}\right)^2 \pi + \left(\xi_1^{(i)}\right)^2 \frac{\pi}{2} + \left(\xi_2^{(i)}\right)^2 \frac{\pi}{2} + \left(\xi_3^{(i)}\right)^2 \frac{\pi}{2} \right] \quad \text{and} \quad \|\phi_{k+1}(t)\|^2 = \sum_{j=1}^{k+1} \gamma_j^2.$$

However, numerical experiments indicate that (5.1) is more appropriate than (5.2) as a practical criterion for the convergence test; see Section 6.1.

We point out that for the problem of sampling from a Gaussian distribution as discussed in Section 5.1, another possibility emerges for estimating the error. Since $\eta = \mathbf{m} + K^{1/2}x$ and the approximant $\hat{\eta} = \mathbf{m} + \phi_{k+1}(K)x$, we have

$$\eta^\epsilon = \eta - \hat{\eta} = (K^{1/2} - \phi_{k+1}(K))x.$$

If $x \sim \mathcal{N}(\mathbf{0}_m, \mathbb{I}_m)$, then it is easy to see that η^ϵ is normally distributed with mean 0 and variance $\mathbf{Var}(\eta^\epsilon) = K - 2K^{1/2}\phi_{k+1}(K) + \phi_{k+1}^2(K)$. From inequality (4.3) it follows that

$$\sqrt{\|\mathbf{Var}(\eta^\epsilon)\|_2} \le \max_{t\in[l,u]} |\phi_{k+1}(t) - f(t)| . \tag{5.3}$$

The right-hand side of the above bound can then be used for statistical tests characterizing the discrepancy between the approximant $\hat{\eta}$ and the variable η that has the exact sought-after distribution. This uniform norm can be empirically estimated by a large enough sampling $\{\hat{t}_i\}$ of the interval $[l, u]$. (Note the $\hat{t}_i$'s are different from the knots t_i's.) To compute $\phi_{k+1}(t)$, note that the vector $[\phi_{k+1}(\hat{t}_1), \ldots, \phi_{k+1}(\hat{t}_r)]^T$ is indeed the output $\hat{z}_{k+1}$ of Algorithm 2 from a diagonal matrix $\hat{A} = \mathrm{diag}(\hat{t}_1, \ldots, \hat{t}_r)$ with the right-hand vector $\hat{b}$, which is the vector of all 1's. In other words, one needs to replicate lines 14 and 16 with a second set of matrix $\hat{A}$ and vectors $\hat{v}_j$ and $\hat{z}_j$. When $\hat{A}$ is large, this additional computation might be expensive, although its cost is independent of the size of the original matrix A and thus, on the number of sampling sites, which tends to dominate all other sizes involved.

6. Numerical results. In this section, we show several numerical experiments to demonstrate the effectiveness and scalability of Algorithm 2. Note an important relation for a symmetric matrix A:

$$\|\phi_{k+1}(A)b - f(A)b\|_2 = \left\|V\phi_{k+1}(D)V^Tb - Vf(D)V^Tb\right\|_2 = \left\|\phi_{k+1}(D)\tilde{b} - f(D)\tilde{b}\right\|_2 ,$$

where $V^TAV = D$ is a diagonalization of A with a unitary V, and $\tilde{b} = V^Tb$. This means that if the right-hand vector b is drawn from some distribution (such as uniform or $\mathcal{N}(\mathbf{0}_m, \mathbb{I}_m)$), then the approximation error for computing $f(A)b$ is equal to that for computing $f(D)\tilde{b}$, where $\tilde{b}$ can be considered a sample from the same distribution. In other words, in a statistical sense, testing a matrix A is equivalent to testing a diagonal matrix D, which has the same spectrum as A. An advantage of replacing A by D is that the ground truth $f(D)b$ is much easier to compute than $f(A)b$. Therefore, all the experiments here were performed with the diagonal matrix D which is unitarily similar to the original A unless otherwise noted. Also, except for the last two subsections, all the experiments were performed for the square root function $f(t) = t^{1/2}$.

6.1. Tests on matrices from the UF collection. We first tested the proposed algorithm on a set of symmetric positive definite matrices from the University of Florida (UF) sparse matrix collection [8]. We chose these matrices with a moderate size m (in the order of thousands) and from ten different application domains, including statistics, materials, power networks, and structural problems. Table 6.1 lists the matrices and the numerical results. For convenience, we list the matrices in the increasing order of their condition numbers κ. The middle column shows the spline-fitting error, $\max_{t\in\Lambda(A)} |s(t) - f(t)|$, which indicates that the spline is in general a sufficiently good approximation to the original function. In the experiment, we set the maximum number of iterations k to be 200 and the tolerance (cf. Equation (5.1)) to be 10^{-6}. The final residual

$$\frac{\left\|z_{k+1} - A^{1/2}b\right\|}{\left\|A^{1/2}b\right\|}$$

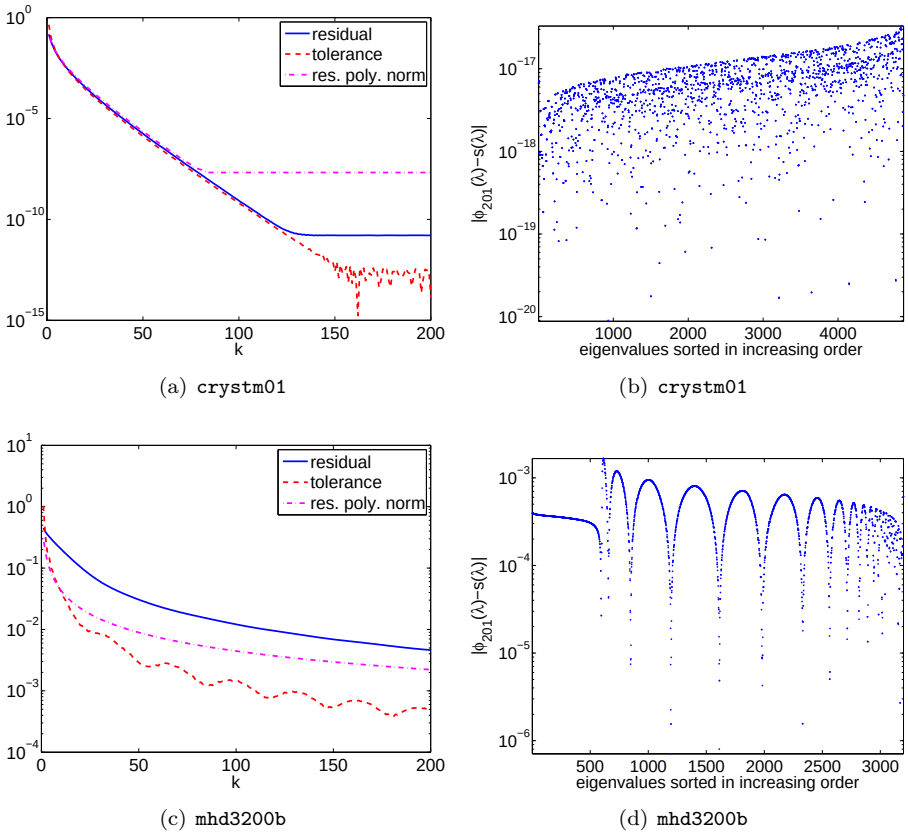
is listed in the last column. We can see that as the condition number of the matrix increases, the approximation in general becomes less accurate. In other words, the polynomial approximation is affected by the conditioning of the matrix. We can also see that the iterate difference is within an order of magnitude difference from the residual, which implies that it can serve as a suitable criterion of the convergence and a good estimate of the actual residual.

TABLE 6.1
Numerical results on the matrices from the UF collection.

Group/Name	κ	Spline Err.	k	Iter. Diff.	Residual
`Boeing/crystm01`	2.28×10^{2}	5.20×10^{-16}	52	9.60×10^{-7}	1.28×10^{-6}
`Bates/Chem97ZtZ`	2.47×10^{2}	8.35×10^{-9}	57	9.37×10^{-7}	4.07×10^{-6}
`JGD/Trefethen_2000`	1.55×10^{4}	3.43×10^{-8}	200	5.80×10^{-6}	4.41×10^{-6}
`TKK/plbuckle`	1.28×10^{6}	3.53×10^{-7}	200	1.08×10^{-4}	2.87×10^{-4}
`Nasa/nasa1824`	1.89×10^{6}	1.12×10^{-6}	200	1.37×10^{-4}	1.26×10^{-3}
`HB/1138_bus`	8.57×10^{6}	3.31×10^{-8}	200	1.94×10^{-4}	1.51×10^{-3}
`Oberwolfach/t2dal_e`	3.76×10^{7}	2.70×10^{-13}	200	1.59×10^{-4}	6.55×10^{-4}
`HB/bcsstk12`	2.21×10^{8}	1.93×10^{-6}	200	2.18×10^{-4}	1.24×10^{-3}
`FIDAP/ex3`	1.68×10^{10}	1.04×10^{-6}	200	1.88×10^{-4}	1.06×10^{-3}
`Bai/mhd3200b`	1.60×10^{13}	3.54×10^{-10}	200	4.73×10^{-4}	4.61×10^{-3}

We performed a further investigation on the matrices of the best (`crystm01`) and the worst (`mhd3200b`) performance (see Figure 6.1). Plot (a) shows three curves as k increases: the residual, the iterate difference (Equation (5.1)), and the norm of the residual polynomial (Equation (5.2)). This plot shows an advantage of using (5.1) rather than (5.2) as the convergence criterion. It suggests that numerically the norm will stop decreasing far before the uniform norm does. Of course, this may affect well-conditioned matrices only, since for ill-conditioned matrices (cf. Plot (b)), within a reasonable number of iterations, say 200, neither norm appears to stop decreasing. To further expose the distribution of the errors, plots (b) and (d) show the value of the residual polynomial $|\phi_{k+1}(t) - s(t)|$ for t equal to the eigenvalues. As expected, the smallest eigenvalues do not seem to contribute in a major way to the residual.

6.2. Scalability. We tested the scalability performance of the algorithm on two types of matrices: "uniform" and "lap2D" . A "uniform" matrix of size $m \times m$ is a



(a) `crystm01` (b) `crystm01`

(c) `mhd3200b` (d) `mhd3200b`

FIG. 6.1. *Residual plots for two matrices:* `crystm01` *and* `mhd3200b`.

diagonal matrix with diagonal entries i/m, $i = 1, 2, \ldots, m$. The condition number is $\kappa = m$. A "lap2D" matrix is the standard Laplacian on a uniform $m_1 \times m_2$ grid. It is of size $m = m_1.m_2$ and is given by

$$\begin{bmatrix} A & -I & & & \\ -I & A & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & A & -I \\ & & & -I & A \end{bmatrix}_{m\times m} \quad \text{with} \quad A = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix}_{m_1\times m_1}.$$

The eigenvalues of lap2D are known:

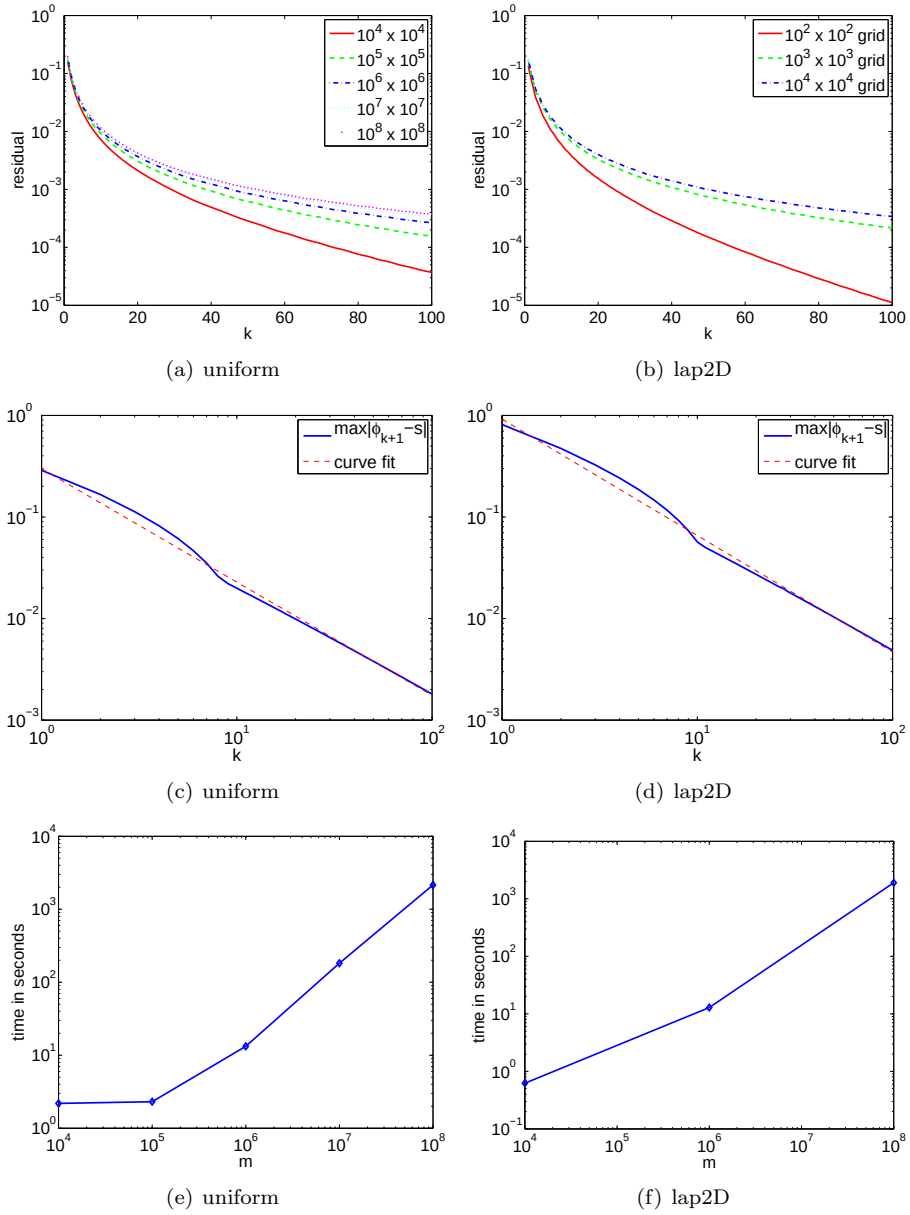
$$4\left[\sin^2\left(\frac{i\pi}{2(m_1+1)}\right) + \sin^2\left(\frac{j\pi}{2(m_2+1)}\right)\right], \quad i = 1, \ldots, m_1, \;\; j = 1, \ldots, m_2.$$

Therefore, its condition number is

$$\kappa = \frac{\sin^2\left(\frac{m_1\pi}{2(m_1+1)}\right) + \sin^2\left(\frac{m_2\pi}{2(m_2+1)}\right)}{\sin^2\left(\frac{\pi}{2(m_1+1)}\right) + \sin^2\left(\frac{\pi}{2(m_2+1)}\right)} \approx \frac{8/\pi^2}{1/m_1^2 + 1/m_2^2}.$$

When $m_1 = m_2 = \sqrt{m}$, it follows that $\kappa = O(m)$. Therefore, both types of matrices have a condition number on the order of their matrix sizes. Note also that in this case the number of knots n is

$$O(\log_{1+a} \kappa) = O(\log_{1+a} m).$$



(a) uniform (b) lap2D

(c) uniform (d) lap2D

(e) uniform (f) lap2D

FIG. 6.2. *Numerical results on "uniform" matrices and "lap2D" matrices.*

Figure 6.2 shows a residual plot, a plot of the uniform norm of the residual polynomial as k increases, and a plot of the run time as m increases, for each type of

matrix. The final residual with $k = 100$ iterations reached 10^{-5} to 10^{-4}. The uniform norm of the residual polynomial was empirically evaluated as

$$\max_{t\in\Lambda(A)} |\phi_{k+1}(t) - s(t)|,$$

for the largest matrix (i.e., $m = 10^8$). A log-log plot of this uniform norm versus the number of iterations shows a straight-line pattern. We fit a line to the plot, and the slope was close to -1. This confirms the rate of convergence given in Corollary 4.7 for the uniform norm of $\phi_{k+1}(t) - s(t)$.

We also plotted the running time of the algorithm (see plots (e) and (f)), in log-log scale. The time included all the computations except the estimation of $\lambda_{\min}$ and $\lambda_{\max}$. Since the algorithm was implemented in Matlab as a serial program, we expect that a more careful implementation in C and/or in a parallel fashion will yield several folds or even magnitudes of time improvement. It can be seen from both plots that the running time is linear as m increases, which is expected from the computational cost analysis.

6.3. Tests on a Gaussian process sampling problem. We consider the covariance matrix K mentioned in Section 5.1, originating in covariance functions with compact kernels described in [40, 35]. Such functions describe processes that can be used in the study of vehicles moving on terrains with random slip coefficients. The covariance function has radial symmetry and rule

$$\ell(r) = \left(1 - \frac{r}{\alpha}\right)_+^{\beta}, \quad \text{where} \quad r = \sqrt{x^2 + y^2}.$$

The covariance matrix K therefore is defined on a 2D grid, with the (i, j) entry $K_{ij} = \ell(d_{ij})$, where d_{ij} is the Euclidean distance between two grid points.

To gauge the effects of the computations at sites that are not necessarily on a regular grid, we also consider the covariance matrix over a deformed space. In this case,

$$r = \sqrt{x^2 + w(x)y^2},$$

where $w(x)$ is a quadratic deforming function that is 2 in the middle of the range of x and 1 at its extremes. With this new definition, $\ell(r)$ can be looked at as either a nonstationary covariance function or a stationary covariance function on the grid shown in Figure 6.3. Neither case can be treated by the FFT approach [9].

We performed tests on the covariance matrices defined on both types of grids with different parameters, as listed in Table 6.2. "Regular" means the grid is uniform, with grid points separated with a spacing 1 unit; "deformed" means the grid is deformed as in Figure 6.3. The middle column shows the condition number κ, which suggests that a smaller α and a larger β will make the matrix better conditioned, and correspondingly the required vector $K^{1/2}b$ will be easier to compute. It also suggests that problems without a regular grid structure can be as easy to solve as the problems on a regular grid by our method. In general, within 100 iterations, the final residual has decreased to 10^{-10}–10^{-11}.

We performed further tests on larger grids, which imposed a difficulty for computing the ground truth $K^{1/2}b$ and the residual. We therefore presented only the iterate difference and the empirical uniform norm of $\phi_{k+1}(t) - f(t)$. From Table 6.2 one sees that the iterate difference is a good indication of the residual, and (5.3) implies

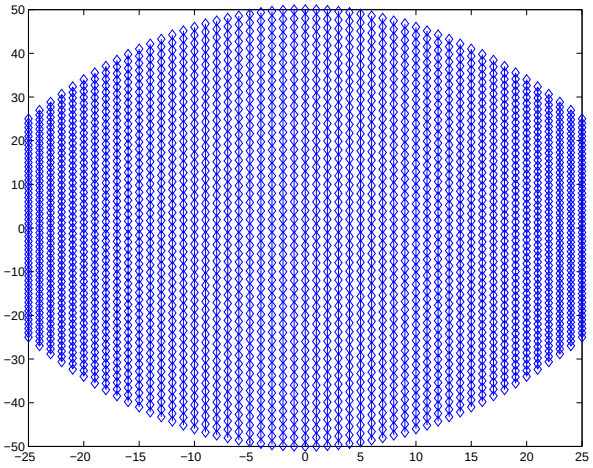


FIG. 6.3. *Position of sampling points in the non uniform grid case.*

TABLE 6.2
Numerical results on the covariance matrices.

Grid	Grid Size	α	β	κ	k	Iter. Diff.	Residual
Regular	100×100	6.5	3	35.10	49	8.0104×10^{-11}	1.2719×10^{-10}
Regular	100×100	12.5	3	243.59	120	9.5001×10^{-11}	4.2465×10^{-10}
Regular	100×100	6.5	5	13.13	31	6.6332×10^{-11}	5.6348×10^{-11}
Regular	100×100	12.5	5	88.01	75	8.8254×10^{-11}	2.3085×10^{-10}
Deformed	100×100	6.5	3	15.95	34	6.3934×10^{-11}	6.9053×10^{-11}
Deformed	100×100	12.5	3	107.10	82	9.0078×10^{-11}	2.6602×10^{-10}
Deformed	100×100	6.5	5	6.20	21	9.2292×10^{-11}	4.9578×10^{-11}
Deformed	100×100	12.5	5	38.65	51	9.2313×10^{-11}	1.5885×10^{-10}

that the uniform norm bounds the variance. To make the computation more feasible, we did not explicitly store the matrix; any matrix-vector multiplication was carried out by considering the special structure of K. Compared with Table 6.2, the results in Table 6.3 suggest that the grid size has little impact on the conditioning of the matrices, and therefore the approximation quality was as good as for smaller grids.

TABLE 6.3
Numerical results on the covariance matrices.

Grid	Grid Size	α	β	κ	k	Iter. Diff.	$\max\lvert\phi_{k+1}-f\rvert$
Regular	$10^3\times10^3$	6.5	3	35.14	48	9.5703×10^{-11}	1.2106×10^{-9}
Regular	$10^3\times10^3$	12.5	3	244.47	122	7.9481×10^{-11}	1.9733×10^{-9}
Regular	$10^3\times10^3$	6.5	5	13.14	32	5.6690×10^{-11}	4.2998×10^{-10}
Regular	$10^3\times10^3$	12.5	5	88.23	79	8.3043×10^{-11}	1.2952×10^{-9}
Deformed	$10^3\times10^3$	6.5	3	15.96	34	4.5546×10^{-11}	4.0252×10^{-10}
Deformed	$10^3\times10^3$	12.5	3	107.36	87	8.7497×10^{-11}	1.2524×10^{-9}
Deformed	$10^3\times10^3$	6.5	5	6.30	22	5.3848×10^{-11}	3.2586×10^{-10}
Deformed	$10^3\times10^3$	12.5	5	38.73	54	6.4973×10^{-11}	9.4807×10^{-10}

Sections 6.1–6.3 present attractive capabilities of the proposed method. We have

applied it on matrices with dimension up to $m = 10^8$ with good results: the residual fell under 10^{-3}. When the technique is used for sampling from a normal distribution of a covariance matrix K, the discrepancy between our simulation and the sought-after distribution is a normal multivariate distribution with at most 10^{-3} relative variance. Moreover, for Gaussian processes with compact kernels, our error estimate indicates that the discrepancy will be much smaller (10^{-9}–10^{-11}). Extrapolation from the 10^4 case and the 10^6 case (Tables 6.2 and 6.3) suggests that an error of 10^{-9} is achievable virtually independent of dimension, by using about $k = 100$ matrix-vector multiplications and thus including the extreme-scale cases that are our ultimate goal.

Of course, for assessing the sought-after 10^{12}–10^{15} range for the number of sites, a parallel program on a high-performance computer will be required. Nevertheless, our approach is factorization-free and thus easily parallelizable. In addition, several efficient ways of estimating the error were given that were demonstrated to be accurate for a large class of matrices. It would be instructive to undertake more extensive computational studies of the relationship between these error estimates and the number k of matrix-vector evaluations.

6.4. Comparison with a related method. In [39], a conjugate residual-type (CR-type) method was proposed which can be adapted for computing $f(A)b = A^{1/2}b$. Instead of computing a polynomial $\phi_{k+1}(t)$ to minimize $\|s(t) - \phi(t)\|$ among all the polynomials $\phi(t)$ of degree not exceeding k, the CR approach computes a different polynomial $\tilde{\phi}_k(t)$ that minimizes $\left\|s(t) - t\tilde{\phi}(t)\right\|$ among all the polynomials $\tilde{\phi}(t)$ of degree not exceeding $k-1$. In other words, to approximate the spline, the proposed algorithm in this paper uses the polynomial $\phi_{k+1}(t) \in \mathbb{P}_{k+1}$, whereas the CR approach uses $\tilde{\phi}_{k+1}(t) = t\tilde{\phi}_k(t)$, where $\tilde{\phi}_k(t) \in \mathbb{P}_k$. The approximation vector from the CR approach algorithm in [39] is thus $A\tilde{\phi}_k(A)b$. From a conceptual point of view the only difference between the two methods is that the CR approach constrains the sought polynomial to have a value of zero at the origin. From a practical point of view the two approaches differ significantly in their implementation. The CR approach draws a parallel with the solution of linear systems and generates a solution polynomial that can be viewed as a residual polynomial of the form $1 - t\tilde{\phi}(t)$ (which therefore approximates the function $1 - s(t)$). Comparisons between the two approaches were made and will not be reproduced here. In short, the two methods deliver similar results, and this is not too surprising.

Note that in the CR approach the polynomial that approximates the spline, $t\tilde{\phi}_k(t)$, has a zero at the origin so it cannot be directly applied as it can when the function/spline $s(t)$ is nonzero at the origin. However, as was seen above (see also [39]), one can consider the "residual polynomial" $1 - t\tilde{\phi}_k(t)$ as the approximating function $\tilde{\phi}_{k+1}$, since this is known to approximate the function $1 - s(t)$, which now has the value 1 (or indeed any other value by using scaling) at the origin. So, this approach forces the polynomial and the function to have the same value at the origin (or some other point).

6.5. Tests with other functions. We further tested the proposed algorithm on two other functions, the natural logarithm and the exponential, using the same set of matrices as in Section 6.1. The logarithm has even larger derivatives than does the square root for t close to the origin; hence it is expected that $\log(A)b$ will be much harder to compute. Since the shapes of the log and the square root functions look similar, we used the same interval scheme as for the square root. On the other hand, a trick for handling the exponential is to scale the matrix such that its spectral

radius is equal to 1, since the derivative of $\exp(t)$ on the interval $[-1, 1]$ is bounded by the small value e. With this preprocessing, the conditioning of the matrix is no longer challenging, and therefore it is not necessary to use the same interval scheme as described in Section 5.2 to perform the spline fitting. We simply used $\log(m)$ knots that were evenly distributed on the spectrum of A.

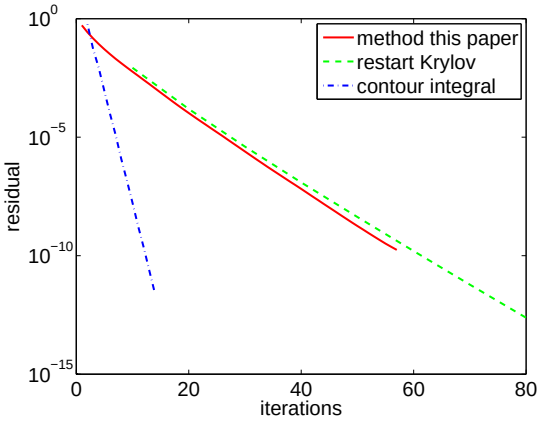
TABLE 6.4
Numerical results for other functions.

Matrix	κ	$\log(A)b$		$\exp(A)b$	
		k	Residual	k	Residual
`Boeing/crystm01`	2.2831×10^{2}	54	1.3587×10^{-6}	9	6.5177×10^{-6}
`Bates/Chem97ZtZ`	2.4721×10^{2}	70	4.0998×10^{-6}	9	9.3029×10^{-6}
`JGD/Trefethen_2000`	1.5518×10^{4}	200	1.8060×10^{-4}	9	9.2387×10^{-6}
`TKK/plbuckle`	1.2833×10^{6}	200	1.0433×10^{-2}	9	1.3273×10^{-5}
`Nasa/nasa1824`	1.8960×10^{6}	200	2.6332×10^{-2}	9	1.2317×10^{-5}
`HB/1138_bus`	8.5726×10^{6}	200	9.0505×10^{-2}	9	7.0982×10^{-6}
`Oberwolfach/t2dal_e`	3.7662×10^{7}	200	3.4874×10^{-2}	9	5.3765×10^{-6}
`HB/bcsstk12`	2.2119×10^{8}	200	5.2497×10^{-2}	9	1.5450×10^{-5}
`FIDAP/ex3`	1.6838×10^{10}	200	1.5316×10^{-1}	9	7.9267×10^{-6}
`Bai/mhd3200b`	1.6035×10^{13}	200	1.5902×10^{-1}	11	2.4630×10^{-6}

The numerical results are listed in Table 6.4. Compared with Table 6.1, it is clear, and expected, that $\log(A)b$ is harder to approximate than $A^{1/2}b$. A fact not shown in the table is that the approximation quality of $\log(K)b$, for the covariance matrix K of the test of Section 6.3, is appealing: We obtained an error estimate no larger than 10^{-10} for $k = 100$ for both the deformed and undeformed mesh cases. This is expected because these matrices are moderately conditioned. Moreover, the results for $\exp(A)b$ indicate a quality of approximation that does not depend on the condition number of the matrix.

6.6. Comparison with other methods on the log function. In this section, we compare the performance of three methods: the method in this paper, the restarted Krylov subspace method [12, 2], and the contour integral method [21], on the covariance matrices K in Section 6.3 and the matrix function log. The purpose of this comparison was to show the general applicability of the proposed method and its advantage over the ones that require solving multiple linear systems. The code used for the restarted Krylov subspace method was obtained from `http://www.mathe.tu-freiberg.de/~guettels/funm_kryl/`, while the one used for the contour integral method was from [21] (`method2.m`). All the tolerances were set to 10^{-10}. In the restarted Krylov subspace method, we used a restart length of 10. In the contour integral method, we used `GMRES(10)` as the inner solver, with tolerance set to 10^{-15}. Results are shown in Figure 6.4.

The residual plots show a linear convergence for all the three methods and this might be a little surprising since the convergence analysis of the proposed method indicates a slower rate theoretically. We conjecture that when the condition number of the matrix is not high, the proposed method can indeed achieve a linear convergence. Further, note that the convergence of the proposed method and the restarted Krylov subspace method is with respect to the number of matrix-vector multiplications, whereas the convergence of the contour integral method is with respect to the number of quadrature points (the number of linear solves). Therefore, this result



100×100 grid

Method	Time (s)
This paper	0.20
Krylov	0.90
Integral	37.56

1000×1000 grid

Method	Time (s)
This paper	88.83
Krylov	112.02
Integral	fail

FIG. 6.4. *Figure: Residual plots for computing* $\log(K)b$ *with the covariance matrix* K *defined on a* 100×100 *grid. The number of iterations for the proposed method and the restart Krylov method is equal to the number of matrix-vector multiplications, whereas "iteration" for the contour integral method means the number of quadrature points. Tables: Computation times for computing* $\log(K)b$ *with the covariance matrix* K *defined on different grids. All the timings exclude the time for estimating the two extreme eigenvalues of* K*.*

shows that the proposed method and the restarted Krylov subspace method are quite close in performance for this application. However, the much faster convergence of the contour integral method may not necessarily mean a better performance. To underline this, we also show the actual run times of the three methods; see the two tables. For the smaller grid, the contour integral method was two orders of magnitude slower than the method presented in this paper, and for the larger grid, `GMRES(10)` failed to solve many shifted linear systems. In the latter case, most often, `GMRES(10)` stagnated at a very large relative residual, and the contour integrals did not converge. This also happened for other parameters and solvers we tried, such as `GMRES(50)` and `BICGSTAB`. Since the systems are based on the same positive definite matrix K (with different shifts), one can also use a Lanczos/CG type method with a single Krylov subspace of K to simultaneously solve the systems. However, as with the case of solving them separately, many systems did not converge. There may exist a suitable method that can successfully solve all the involved linear systems; however, pursuing this further is beyond the scope of this paper. (It is noted that perhaps an exception is the square root function, where all the shifted systems are positive definite, and a CG solver may be used to solve the systems. In such a case, the contour integral method will likely be as efficient as the method proposed in this paper.) The main message to be conveyed here is that solving linear systems is itself a complicated and challenging task, and a method that avoids linear solves and which has a guaranteed convergence may be more desirable.

7. Concluding remarks. We have presented a least squares polynomial approximation method for computing a function of a matrix times a vector $f(A)b$. The method first approximates the function $f(t)$ by a spline $s(t)$ and then projects $s(t)$ onto a polynomial subspace such that $s(A)b$ can be (approximately) evaluated as a polynomial of A times b. This technique avoids explicitly forming $f(A)$; and the matrix A is referenced only through k matrix-vector multiplications, where k is the degree of the polynomial.

The quality of the approximation obtained from the method depends on the nature of the function f. Specific interval selection schemes for using a spline to fit the function must be defined individually for each f. We discussed the case $f(t) = \sqrt{t}$ in detail and briefly mentioned the case $f(t) = \log(t)$ and $f(t) = \exp(t)$. Analysis shows that in order to yield accurate approximations, it is mandatory to place enough knots on the region where $f'(t)$ is large. By following this guideline, effective interval schemes for other functions can also be derived.

Experiments show that the proposed algorithm is efficient for a practical statistical sampling problem, which involves computing $K^{1/2}b$ for a covariance matrix $K \in \mathbb{R}^{m\times m}$ with a stationary/nonstationary covariance function defined on regular/irregular grids. This application is an example where the matrix A need not be explicitly stored. The algorithm was demonstrated on problems with m up to 10^6, and current results point to promising performance for problems at extreme scales, with $m = 10^{12}$ to 10^{15}.

REFERENCES

[1] M. Afanasjew, M. Eiermann, O. G. Ernst, and S. Güttel, *A generalization of the steepest descent method for matrix functions*, Electronic Trans. Numerical Analysis, 28 (2008), pp. 206–222.

[2] ———, *Implementation of a restarted Krylov subspace method for the evaluation of matrix functions*, Linear Algebra Appl., 429 (2008), pp. 2293–2314.

[3] Z. Bai, M. Fahey, and G. Golub, *Some large-scale matrix computation problems*, J. Comput. Appl. Math., 74 (1996), pp. 71–89.

[4] R. Barry and R. Kelley Pace, *Monte Carlo estimates of the log determinant of large sparse matrices*, Linear Algebra and its Applications, 289 (1999), pp. 41–54.

[5] L. Bergamaschi, M. Caliari, and M. Vianello, *Efficient approximation of the exponential operator for discrete 2D advection-diffusion problems*, Numer. Lin. Alg. Appl., 10 (2002), pp. 271–289.

[6] A. Canning, *Scalable parallel 3D FFTs for electronic structure codes*, in High Performance Computing for Computational Science - VECPAR 2008, Springer, Berlin, 2008, pp. 280–286.

[7] E. Constantinescu, V. Zavala, M. Rocklin, S. Lee, and M. Anitescu., *Unit commitment with wind power generation: Integrating wind forecast uncertainty and stochastic programming.*, Tech. Memo. ANL/MCS-TM309., Argonne National Laboratory, Mathematics and Computer Science Division, 9700 S Cass Ave, Argonne IL 60439, September 2009. Available online at http://www.mcs.anl.gov/~anitescu/PUBLICATIONS/TM-309.pdf.

[8] T. A. Davis, *The University of Florida sparse matrix collection*, ACM Trans. Math. Softw., (submitted, 2009).

[9] C. Dietrich and G. Newsam, *Fast and exact simulation of stationary Gaussian processes through circulant embedding of the covariance matrix*, SIAM Journal on Scientific Computing, 18 (1997), pp. 1088–1107.

[10] V. Druskin and L. Knizhnerman, *Two polynomial methods of calculating functions of symmetric matrices*, USSR Computational Mathematics and Mathematical Physics, 29 (1989), pp. 112–121.

[11] ———, *Extended Krylov subspaces: Approximation of the matrix square root and related functions*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 755–771.

[12] M. Eiermann and O. G. Ernst, *A restarted Krylov subspace method for the evaluation of matrix functions*, SIAM J. Numer. Anal., 44 (2006), pp. 2481–2504.

[13] J. Erhel, F. Guyomarc, and Y. Saad, *Least-squares polynomial filters for ill-conditioned linear systems*, Tech. Rep. UMSI 2001/32, University of Minnesota Supercomputing Institute, 2001.

[14] P. Fischer, J. Lottes, D. Pointer, and A. Siegel, *Petascale algorithms for reactor hydrodynamics*, in Journal of Physics: Conference Series, vol. 125(1), Institute of Physics Publishing, 2008, p. 012076.

[15] A. Frommer and V. Simoncini, *Stopping criteria for rational matrix functions of Hermitian and symmetric matrices*, SIAM J. Sci. Comput., 30 (2008), pp. 1387–1412.

[16] R. Furrer, M. Genton, and D. Nychka, *Covariance tapering for interpolation of large spatial*

datasets, Journal of Computational and Graphical Statistics, 15 (2006), pp. 502–523.
[17] E. Gallopoulos and Y. Saad, *Efficient solution of parabolic equations by Krylov approximation methods*, SIAM J. Sci. and Stat. Comput., 13 (1992), pp. 1236–1264.
[18] G. H. Golub and G. Meurant, *Matrices, moments, and quadrature*, in Numerical Analysis 1993, D. F. Griffiths and G. A. Watson, eds., vol. 303, Pitman, Research Notes in Mathematics, 1994, pp. 105–156.
[19] ———, *Matrices, Moments and Quadrature with Applications*, Princeton University Press, 2009.
[20] J. Goodman and A. Sokal, *Multigrid Monte Carlo method: Conceptual foundations.*, Physical Review D: Particles and fields, 40 (1989), p. 2035.
[21] N. Hale, N. J. Higham, and L. N. Trefethen, *Computing $a^\alpha, \log(a)$, and related matrix functions by contour integrals*, SIAM J. Numer. Anal., 46 (2008), pp. 2505–2523.
[22] N. J. Higham, *Functions of Matrices: Theory and Computation*, SIAM, 2008.
[23] M. Hochbruck and C. Lubich, *On Krylov subspace approximations to the matrix exponential operator*, SIAM J. Numer. Anal., 34 (1997), pp. 1911–1925.
[24] M. Hochbruck, C. Lubich, and H. Selhofer, *Exponential integrators for large systems of differential equations*, SIAM J. Sci. Comput., 19 (1998), pp. 1552–1574.
[25] M. Hutchinson, *A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines*, Communications in Statistics-Simulation and Computation, 18 (1989), pp. 1059–1076.
[26] C. Kaufman, M. Schervish, and D. Nychka, *Covariance tapering for likelihood-based estimation in large spatial data sets*, Journal of the American Statistical Association, 103 (2008), pp. 1545–1555.
[27] L. Knizhnerman and V. Simoncini, *A new investigation of the extended Krylov subspace method for matrix function evaluations*, Numer. Lin. Alg. Appl., (to appear).
[28] N. N. Lebedev, *Special Functions and Their Applications*, Dover, 1972.
[29] C. Moler and C. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix*, SIAM Review, 20 (1978), pp. 801–836.
[30] ———, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Review, 41 (2003), pp. 3–49.
[31] I. Moret and P. Novati, *The computation of functions of matrices by truncated Faber series*, Numerical Functional Analysis and Optimization, 22 (2001), pp. 697–719.
[32] ———, *RD-rational approximations of the matrix exponential*, BIT Numerical Mathematics, 44 (2004), pp. 595–615.
[33] P. Novati, *A polynomial method based on Fejér points for the computation of functions of unsymmetric matrices*, Applied Numerical Mathematics, 44 (2003), pp. 201–224.
[34] P. P. Petrushev and V. A. Popov, *Rational Approximation of Real Functions*, Cambridge University Press, 1988.
[35] C. Rasmussen and C. Williams, *Gaussian processes for machine learning*, MIT Press, Cambridge, 2006.
[36] T. J. Rivlin, *An Introduction to the Approximation of Functions*, Dover Publications, 1981.
[37] Y. Saad, *Iterative solution of indefinite symmetric systems by methods using orthogonal polynomials over two disjoint intervals*, SIAM J. Numer. Anal., 20 (1983), pp. 784–811.
[38] ———, *Analysis of some Krylov subspace approximations to the matrix exponential operator*, SIAM J. Numer. Anal., 29 (1992), pp. 209–228.
[39] ———, *Filtered conjugate residual-type algorithms with applications*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 845–870.
[40] K. P. Schmitt, M. Anitescu, and D. Negrut, *Efficient sampling for spatial uncertainty quantification in multibody system dynamics applications*, International Journal for Numerical Methods in Engineering, 80 (2009), pp. 537–564.
[41] M. H. Schultz, *Spline Analysis*, Prentice Hall, 1973.
[42] D. P. Simpson, *Krylov subspace methods for approximating functions of symmetric positive definite matrices with applications to applied statistics and anomalous diffusion*, PhD thesis, Queensland University of Technology, 2008.
[43] M. Stein, *Interpolation of Spatial Data: Some theory for Kriging*, Springer-Verlag, Berlin, 1999.
[44] U. Trottenberg, C. Oosterlee, and A. Schuller, *Multigrid*, Academic Press, 2001.
[45] J. van den Eshof and M. Hochbruck, *Preconditioning Lanczos approximations to the matrix exponential*, SIAM J. Sci. Comput., 27 (2006), pp. 1438–1457.
[46] H. A. Van der Vorst, *An iterative solution method for solving $f(A)x = b$, using Krylov subspace information obtained for the symmetric positive definite matrix A*, J. Comput. Appl. Math., 18 (1987), pp. 249–263.

[47] ———, *Solution of $f(A)x = b$ with projection methods for the matrix A*, in Numerical Challenges in Lattice Quantum Chromodynamics, Springer Verlag, Berlin, 2000, pp. 18–28.

[48] M. Wong, F. Hickernell, and K. Liu, *Computing the trace of a function of a sparse matrix via hadamard-like sampling*, Preprint 377, Department of Mathematics, Hong Kong Baptist University, 2004.