Linear Algebra and its Applications 533 (2017) 95–117

Contents lists available at ScienceDirect

Linear Algebra and its Applications

www.elsevier.com/locate/laa

A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix

Christos Boutsidis, Petros Drineas [a,*], Prabhanjan Kambadur [b], Eugenia-Maria Kontopoulou [a,*], Anastasios Zouzias

[a] *Purdue University, West Lafayette, IN, United States*
[b] *Bloomberg L.P. New York, NY, United States*

ARTICLE INFO

Article history:
Received 15 June 2016
Accepted 5 July 2017
Available online 24 July 2017
Submitted by H. Rauhut

MSC:
15A15
65F99
68W20
68W25

Keywords:
Logdeterminant approximation
SPD matrix
Randomized algorithm
Power method
Trace estimation

ABSTRACT

We introduce a novel algorithm for approximating the logarithm of the determinant of a symmetric positive definite (SPD) matrix. The algorithm is randomized and approximates the traces of a small number of matrix powers of a specially constructed matrix, using the method of Avron and Toledo [1]. From a theoretical perspective, we present additive and relative error bounds for our algorithm. Our additive error bound works for any SPD matrix, whereas our relative error bound works for SPD matrices whose eigenvalues lie in the interval $(\theta_1, 1)$, with $0 < \theta_1 < 1$; the latter setting was proposed in [16]. From an empirical perspective, we demonstrate that a C++ implementation of our algorithm can approximate the logarithm of the determinant of large matrices very accurately in a matter of seconds.

* Corresponding authors.
E-mail addresses: christos.boutsidis@gmail.com (C. Boutsidis), pdrineas@purdue.edu (P. Drineas), pkambadur@bloomberg.net (P. Kambadur), ekontopo@purdue.edu (E.-M. Kontopoulou), zouzias@gmail.com (A. Zouzias).

http://dx.doi.org/10.1016/j.laa.2017.07.004
0024-3795/

1. Introduction

Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, the determinant of $\mathbf{A}$, denoted by $\det(\mathbf{A})$, is one of the most important quantities associated with $\mathbf{A}$. Since its invention by Cardano and Leibniz in the late 16th century, the determinant has been a fundamental mathematical concept with countless applications in numerical linear algebra and scientific computing. The advent of Big Data, which are often represented by matrices, increased the applicability of algorithms that compute, exactly or approximately, matrix determinants; see, for example, [20,32,31,6,17] for machine learning applications (e.g., gaussian process regression) and [19,18,9,22,23] for several data mining applications (e.g., spatial-temporal time series analysis).

Formal definitions of the determinant include the well-known formulas derived by Leibniz and Laplace; however, neither the Laplace nor the Leibniz formula can be used to design an efficient, polynomial-time, algorithm to compute the determinant of $\mathbf{A}$. To achieve this goal, one should rely on other properties of the determinant. For example, a standard approach would be to leverage the so-called LU matrix decomposition or the Cholesky decomposition for symmetric positive definite matrices (SPD) to get an $O(n^3)$ deterministic algorithm to compute the determinant of $\mathbf{A}$. (Recall that an SPD matrix is a symmetric matrix with strictly positive eigenvalues.)

In this paper, we are interested in approximating the logarithm of the determinant of a symmetric positive definite (SPD) matrix $\mathbf{A}$. The logarithm of the determinant, instead of the determinant itself, is important in several settings [20,32,31,6,17,19,18,9,22,23].

Definition 1. [LogDet problem definition] Given an SPD matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, compute, exactly or approximately, $\text{logdet}(\mathbf{A})$.

Note that since all the eigenvalues of $\mathbf{A}$ are strictly positive, the determinant of $\mathbf{A}$ is strictly positive. The best exact algorithm for the above problem simply computes the determinant of $\mathbf{A}$ in cubic time and takes its logarithm. Few approximation algorithms have appeared in the literature, but they either lack a proper theoretical convergence analysis or do not work for all SPD matrices. We will discuss prior work in detail in Section 1.2.

1.1. Our contributions

We present a fast approximation algorithm for the problem of Definition 1. Our main algorithm (Algorithm 3) is randomized and its running time is

$$\mathcal{O}\left(\text{nnz}(\mathbf{A})\left(m\varepsilon^{-2}+\log n\right)\log(1/\delta)\right),$$

where $\text{nnz}(\mathbf{A})$ denotes the number of non-zero elements in $\mathbf{A}$, $0<\delta<1$ denotes the failure probability of our algorithm, and (integer) $m>0$ and (real) $\varepsilon>0$ are user-controlled

accuracy parameters that are specified in the input of the algorithm. The first step of our approximation algorithm uses the power method to compute an approximation to the dominant eigenvalue of $\mathbf{A}$. This value will be used in a normalization (preconditioning) step in order to compute a convergent matrix-Taylor expansion. The second step of our algorithm leverages a truncated matrix-Taylor expansion of a suitably constructed matrix in order to compute an approximation of the log determinant. This second step leverages a randomized trace estimation algorithm from [1].

Let $\widehat{\text{logdet}}\,(\mathbf{A})$ be the value returned by our approximation algorithm (Algorithm 3); let $\text{logdet}\,(\mathbf{A})$ be the true log determinant of $\mathbf{A}$; let $\lambda_i\,(\mathbf{A})$ denote the i-th eigenvalue of $\mathbf{A}$ for all $i = 1, \ldots, n$ with $\lambda_1(\mathbf{A}) \geq \lambda_2(\mathbf{A}) \geq \ldots \geq \lambda_n(\mathbf{A}) > 0$; and let $\kappa\,(\mathbf{A}) = \lambda_1(\mathbf{A})/\lambda_n(\mathbf{A})$ be the condition number of $\mathbf{A}$. Our main result, proven in Lemma 6, is that if

$$m \geq \left\lceil 7\kappa\,(\mathbf{A}) \log\left(\frac{1}{\varepsilon}\right)\right\rceil, \tag{1}$$

then, with probability at least $1 - 2\delta$,

$$\left|\widehat{\text{logdet}}\,(\mathbf{A}) - \text{logdet}\,(\mathbf{A})\right| \leq 2\varepsilon\Gamma, \tag{2}$$

where

$$\Gamma = \sum_{i=1}^{n} \log\left(7 \cdot \frac{\lambda_1(\mathbf{A})}{\lambda_i(\mathbf{A})}\right).$$

We now take a careful look at the above approximation bound. First, given our choice of m in eqn. (1), the running time of the algorithm becomes

$$\mathcal{O}\left(\text{nnz}(\mathbf{A})\left(\kappa\,(\mathbf{A}) \log\left(1/\varepsilon\right)\varepsilon^{-2} + \log n\right)\log(1/\delta)\right). \tag{3}$$

Thus, the running time of our algorithm increases linearly with the condition number of $\mathbf{A}$. The error of our algorithm scales with Γ, a quantity that is not immediately comparable to $\text{logdet}\,(\mathbf{A})$. It is worth noting that the Γ term increases *logarithmically* with respect to the ratios $\lambda_1(\mathbf{A})/\lambda_i(\mathbf{A}) \geq 1$. An obvious, but potentially loose upper bound for the sum of those ratios, is

$$\Gamma = \sum_{i=1}^{n} \log\left(7 \cdot \frac{\lambda_1(\mathbf{A})}{\lambda_i(\mathbf{A})}\right) \leq n \cdot \log\left(7\kappa(\mathbf{A})\right). \tag{4}$$

Our second result handles the family of SPD matrices whose eigenvalues all lie in the interval $(\theta_1, 1)$, with $0 < \theta_1 < 1$; this setting was proposed in [16]. In this case, a simplified version of Algorithm 3 returns a relative error approximation to the log-determinant of the input matrix. Indeed, Lemma 8 proves that, with probability at least $1 - \delta$,

$$\left|\widehat{\text{logdet}}\,(\mathbf{A}) - \text{logdet}\,(\mathbf{A})\right| \leq 2\varepsilon|\text{logdet}\,(\mathbf{A})\,|.$$

The running time of the simplified algorithm is

$$\mathcal{O}\left(\frac{\log(1/\varepsilon)\log(1/\delta)}{\varepsilon^2\theta_1}nnz(\mathbf{A})\right). \tag{5}$$

Finally, we implemented our algorithm in `C++` and tested it on several large dense and sparse matrices. Our dense implementation runs on top of `Elemental` [25], a linear algebra library for distributed matrix computations with dense matrices. Our sparse implementation runs on top of `Eigen`,[1] a software library for sparse matrix computations. Our code is available to download on Github (see Section 5 for details and a link to our code).

1.2. Related work

The most relevant result to ours is the work in [2]. Barry and Pace [2] described a randomized algorithm for approximating the logarithm of the determinant of a matrix with special structure that we will describe below. They show that in order to approximate the logarithm of the determinant of a matrix $\mathbf{A}$, it suffices to approximate the traces of $\mathbf{D}^k$, for $k = 1, 2, 3...$ for a suitably constructed matrix $\mathbf{D}$. Specifically, [2] deals with approximations to SPD matrices $\mathbf{A}$ of the form $\mathbf{A} = \mathbf{I}_n - \alpha\mathbf{D}$, where $0 < \alpha < 1$ and all eigenvalues of $\mathbf{D}$ are in the interval $[-1, 1]$. Given such a matrix $\mathbf{A}$, the authors of [2] seek to derive an estimator $\widehat{\text{logdet}}\,(\mathbf{A})$ that is close to $\text{logdet}\,(\mathbf{A})$. [2] proved (using the so-called Martin expansion [21]) that

$$\log(\det(\mathbf{A})) = -\sum_{k=1}^{m}\frac{\alpha^k}{k}\mathbf{tr}\left(\mathbf{D}^k\right) - \sum_{k=m}^{\infty}\frac{\alpha^k}{k}\mathbf{tr}\left(\mathbf{D}^k\right).$$

They considered the following estimator:

$$\widehat{\text{logdet}}\,(\mathbf{A}) = \frac{1}{p}\sum_{i=1}^{p}\left(\underbrace{-n\sum_{k=1}^{m}\left(\frac{\alpha^k}{k}\frac{\mathbf{z}_i^{\mathrm{T}}\mathbf{D}^k\mathbf{z}_i}{\mathbf{z}_i^{\mathrm{T}}\mathbf{z}_i}\right)}_{V_i}\right).$$

All V_i for $i = 1 \ldots p$ are random variables and the value of p controls the variance of the estimator. The algorithm in [2] constructs vectors $\mathbf{z}_i \in \mathbb{R}^n$ whose entries are independent identically distributed standard Gaussian random variables. The above estimator ignores the trailing terms of the Martin expansion and only tries to approximate the first m terms. [2] presented the following approximation bound:

[1] http://eigen.tuxfamily.org/.

$$\left|\widehat{\text{logdet}}\,(\mathbf{A}) - \text{logdet}\,(\mathbf{A})\right| \leq \frac{n \cdot \alpha^{m-1}}{(m+1)(1-\alpha)} + 1.96 \cdot \sqrt{\frac{\sigma^2}{p}},$$

where σ^2 is the variance of the random variable V_i. The above bound fails with probability at most 0.05.

We now compare the results in [2] with ours. First, the idea of using the Martin expansion [21] to relate the logarithm of the determinant and traces of matrix powers is present in both approaches. Second, the algorithm of [2] is applicable to SPD matrices that have special structure, while our algorithm is applicable to any SPD matrix. Intuitively, we overcome this limitation of [2] by estimating the top eigenvalue of the matrix in the first step of our algorithm. Third, our error bound is much better that the error bound of [2]. To analyze our algorithm, we used the theory of randomized trace estimators of Avron and Toledo [1], which relies on stronger measure-concentration inequalities than [2], which uses the weaker Chebyshev's inequality.

A similar idea using Chebyshev polynomials appeared in the paper [24]; to the best of our understanding, there are no theoretical convergence properties of the proposed algorithm. Applications to Gaussian process regression appeared in [20,32,31]. The work of [26] uses an approximate matrix inverse to compute the n-th root of the determinant of $\mathbf{A}$ for large sparse SPD matrices. The error bounds in this work are a posteriori and thus not directly comparable to our bounds.

[14] provides a strong worst-case theoretical result which is, however, only applicable to Symmetric Diagonally Dominant (SDD) matrices. The algorithm is randomized and guarantees that, with high probability, $\left|\widehat{\text{logdet}}\,(\mathbf{A}) - \text{logdet}\,(\mathbf{A})\right| \leq \varepsilon \cdot n$, for a user specified error parameter $\varepsilon > 0$. This approach also uses the Martin expansion [21] as well as ideas from preconditioning systems of linear equations with Laplacian matrices [29]. The algorithm of [14] runs in time $O\left(\text{nnz}(\mathbf{A})\varepsilon^{-2}\log^3(n)\log^2(n\kappa(\mathbf{A})/\varepsilon)\right)$. To compare to our approach, we need to combine the suboptimal upper bound for Γ from eqn. (4) with the bound of eqn. (2). Then, we can run Algorithm 3 with input

$$\varepsilon' = \frac{\varepsilon}{\log(7\kappa(\mathbf{A}))},$$

instead of ε to guarantee that the final error of our approximation will be bounded by $\varepsilon' n$. Then, we can observe that the running time of [14] depends *logarithmically* on the condition number of the input matrix $\mathbf{A}$, whereas our algorithm has a linear dependency on the condition number. Notice, however, that our method is applicable to any SPD matrix while the method in [14] is applicable only to SDD matrices; given current state-of-the-art on Laplacian preconditioners it looks hard to extend the approach of [14] to general SPD matrices.

Independently and in parallel with our work, [16] presented an algorithm using Stochastic Chebyshev Expansions for the log-determinant problem. The algorithm is very similar in spirit to our approach, using the Chebyshev instead of the Taylor expansion and achieves relative-error guarantees for a special class of SPD matrices, namely

matrices whose eigenvalues all lie in the interval $(\theta_1, 1-\theta_1)$ for some $0 < \theta_1 < 1/2$. As we already discussed, our algorithm also achieves a relative error bound under such an assumption; the only difference is that the running time of [16] is proportional to $\sqrt{\frac{1}{\theta_1}} \log \frac{1}{\theta_1}$, whereas the running time of our approach (see eqn. (5)) is proportional to $\frac{1}{\theta_1}$. This slightly improved running time might be due to the use of the Stochastic Chebyshev Expansions. However, importantly, our algorithm works for *any* SPD matrix, with arbitrary eigenvalues. Not surprisingly, the added generality comes with a loss in accuracy and the relative error bound becomes an additive error bound.

Finally, two very recent papers [15,27][2] presented algorithms to approximate the logdet of a matrix, highlighting the renewed importance of the topic. The work of [27] presents a very novel approach to approximate the logdet of a positive semi-definite matrix, using a randomized subspace iteration approach. To the best of our understanding, the relevant bounds in their work (Theorem 2 in [27]) are not directly comparable to our bounds. The work of [15] follows the lines of [16] and leverages the use of Chebyshev approximations to propose novel estimators for the trace of a matrix function. Among the many exciting applications of the proposed approach is an additive-error approach to approximate the logdet of any square non-singular matrix; the algorithm needs as inputs upper and lower bounds for all the singular values of the input matrix. Similar to the running time of our additive error algorithm in eqn. (3), the time complexity of the proposed algorithm depends on the condition number of the input matrix (see Corollary 7 of [15]).

We conclude by noting that common algorithms for the determinant computation assume floating point arithmetic and do not measure bit operations. If the computational cost is to be measured in bit operations, the situation is much more complicated and an exact computation of the determinant, even for integer matrices, is not trivial. We refer the interested reader to [8] for more details.

2. Preliminaries

2.1. Notation

Let $\mathbf{A}, \mathbf{B}, \ldots$ denote matrices and let $\mathbf{a}, \mathbf{b}, \ldots$ denote column vectors. $\mathbf{I}_n$ is the $n \times n$ identity matrix; $\mathbf{0}_{m\times n}$ is the $m \times n$ matrix of zeros; $\mathbf{tr}(\mathbf{A})$ is the trace of a square matrix $\mathbf{A}$; the Frobenius and the spectral matrix-norms are: $\|\mathbf{A}\|_F^2 = \sum_{i,j} \mathbf{A}_{ij}^2$ and $\|\mathbf{A}\|_2 = \max_{\|\mathbf{x}\|_2=1} \|\mathbf{A}\mathbf{x}\|_2$. We denote the determinant of a matrix $\mathbf{A}$ by $\det(\mathbf{A})$ and the (natural) logarithm of the determinant of $\mathbf{A}$ by $\mathrm{logdet}(\mathbf{A})$. We use $\log x$ to denote the natural logarithm of x. Finally, given an event $\mathcal{E}$, $\mathbb{P}[\mathcal{E}]$ denotes the probability of the event.

[2] Both papers appeared after an earlier version of this paper was posted on ArXiv on March 2015 and cite this earlier version of our work.

For an SPD matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{log}\,[\mathbf{A}]$ is an $n \times n$ matrix defined as: $\mathbf{log}\,[\mathbf{A}] = \mathbf{U}\mathbf{D}\mathbf{U}^{\mathrm{T}}$, where $\mathbf{U} \in \mathbb{R}^{n\times n}$ contains the eigenvectors of $\mathbf{A}$ and $\mathbf{D} \in \mathbb{R}^{n\times n}$ is diagonal with entries being

$$\log(\lambda_1(\mathbf{A})), \log(\lambda_2(\mathbf{A})), \ldots, \log(\lambda_n(\mathbf{A})).$$

Let x be a scalar variable that satisfies $|x| < 1$. Then, using the Taylor expansion,

$$\log(1-x) = -\sum_{k=1}^{\infty} \frac{x^k}{k}.$$

A matrix-valued generalization of this identity is the following statement.

Lemma 1. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a symmetric matrix whose eigenvalues all lie in the interval $(-1, 1)$. Then,*

$$\log(\mathbf{I}_n - \mathbf{A}) = -\sum_{k=1}^{\infty} \frac{\mathbf{A}^k}{k}.$$

2.2. Power method

The first step in our algorithm for approximating the determinant of an SPD matrix is to obtain an estimate for the largest eigenvalue of the matrix. Given an SPD matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ we will use the power-method (Algorithm 1) to obtain an accurate estimate of its largest eigenvalue. This estimated eigenvalue is denoted by $\tilde{\lambda}_1(\mathbf{A})$.

Algorithm 1 Power method, repeated q times.

- Input: SPD matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, integers $q,\ t > 0$
- For $j = 1, \ldots, q$
  1. Pick uniformly at random a vector $\mathbf{x}_0^j \in \{+1, -1\}^n$
  2. For $i = 1, \ldots, t$
     - $\mathbf{x}_i^j = \mathbf{A} \cdot \mathbf{x}_{i-1}^j$
  3. Compute: $\tilde{\lambda}_1^j(\mathbf{A}) = \frac{\mathbf{x}_t^{j\mathrm{T}} \mathbf{A} \mathbf{x}_t^j}{\mathbf{x}_t^{j\mathrm{T}} \mathbf{x}_t^j}$
- Return: $\tilde{\lambda}_1(\mathbf{A}) = \max_{j=1\ldots q} \lambda_1^j$ (and the corresponding vector $\mathbf{x}_t = \mathbf{x}_t^j$)

Algorithm 1 requires $\mathcal{O}(qt(n + nnz(\mathbf{A})))$ arithmetic operations to compute $\tilde{\lambda}_1(\mathbf{A})$. Lemma 2 (see [30] for a proof) argues that any $\tilde{\lambda}_1^j(\mathbf{A})$ is close to $\lambda_1(\mathbf{A})$.

Lemma 2. *For any fixed $j = 1 \ldots q$, and for any $t > 0$, $\varepsilon > 0$, with probability at least 3/16,*

$$\frac{(1-\varepsilon)}{1 + 4n(1-\varepsilon)^{2t}} \lambda_1(\mathbf{A}) \leq \frac{\mathbf{x}_t^{j\mathrm{T}} \mathbf{A} \mathbf{x}_t^j}{\mathbf{x}_t^{j\mathrm{T}} \mathbf{x}_t^j} = \tilde{\lambda}_1^j(\mathbf{A}).$$

Let $e = 2.718\ldots$ and let $\varepsilon = 1 - (1/e)$ and $t = \lceil \log \sqrt{4n} \rceil$; then, with probability at least 3/16, for any fixed $j = 1 \ldots q$,

$$\frac{1}{6}\lambda_1(\mathbf{A}) \leq \frac{1}{2e}\lambda_1(\mathbf{A}) \leq \tilde{\lambda}_1^j(\mathbf{A}).$$

It is now easy to see that the largest value $\tilde{\lambda}_1(\mathbf{A})$ (and the corresponding vector $\mathbf{x}_t$) fails to satisfy the inequality $(1/6)\lambda_1(\mathbf{A}) \leq \tilde{\lambda}_1(\mathbf{A})$ with probability at most

$$\left(1 - \frac{3}{16}\right)^q = \left(\frac{13}{16}\right)^q \leq \delta,$$

where the last inequality follows by setting $q = \lceil 4.82 \log(1/\delta) \rceil \geq \log(1/\delta)/\log(16/13)$. Finally, we note that, from the min-max principle, $\tilde{\lambda}_1(\mathbf{A}) \leq \lambda_1(\mathbf{A})$. We summarize the above discussion in the following lemma.

Lemma 3. *Let $\tilde{\lambda}_1(\mathbf{A})$ be the output of Algorithm 1 with $q = \lceil 4.82 \log(1/\delta) \rceil$ and $t = \lceil \log \sqrt{4n} \rceil$. Then, with probability at least $1 - \delta$,*

$$\frac{1}{6}\lambda_1(\mathbf{A}) \leq \tilde{\lambda}_1(\mathbf{A}) \leq \lambda_1(\mathbf{A}).$$

The running time of Algorithm 1 is $\mathcal{O}\left((n + nnz(\mathbf{A}))\log(n)\log\left(\frac{1}{\delta}\right)\right)$.

2.3. Trace estimation

Even though computing the trace of a square $n \times n$ matrix requires only $O(n)$ arithmetic operations, the situation is more complicated when $\mathbf{A}$ is given through a matrix function, e.g., $\mathbf{A} = \mathbf{X}^2$, for some matrix $\mathbf{X}$ and the user only observes $\mathbf{X}$. For situations such as these, Avron and Toledo [1] analyzed several algorithms to estimate the trace of $\mathbf{A}$. Algorithm 2 and Lemma 4 present the relevant results from their paper.

Algorithm 2 Randomized trace estimation.

- Input: SPD matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, accuracy $0 < \varepsilon < 1$, and failure probability $0 < \delta < 1$.
  1. Let $p = \lceil 20 \log(2/\delta)/\varepsilon^2 \rceil$
  2. Let $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_p$ be a set of independent Gaussian vectors in $\mathbb{R}^n$
  3. Let $\gamma = 0$
  4. For $i = 1, \ldots, p$
     - $\gamma = \gamma + \mathbf{g}_i^\top \mathbf{A}\mathbf{g}_i$
  5. $\gamma = \gamma/p$
- Return: γ

Lemma 4. *Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be an SPD matrix, let $0 < \varepsilon < 1$ be an accuracy parameter, and let $0 < \delta < 1$ be a failure probability. If $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_p \in \mathbb{R}^n$ are independent random*

standard Gaussian vectors, then, for $p = \lceil 20 \log(2/\delta)/\varepsilon^2 \rceil$, *with probability at least* $1-\delta$,

$$\left| \boldsymbol{tr}(\mathbf{A}) - \frac{1}{p} \sum_{i=1}^{p} \mathbf{g}_i^\top \mathbf{A} \mathbf{g}_i \right| \leq \varepsilon \cdot \boldsymbol{tr}(\mathbf{A}).$$

The above lemma is immediate from Theorem 5.2 in [1].

3. Additive error approximation for general SPD matrices

Lemma 5 is the starting point of our main algorithm for approximating the determinant of a symmetric positive definite matrix. The message in the lemma is that computing the log determinant of an SPD matrix $\mathbf{A}$ reduces to the task of computing the largest eigenvalue of $\mathbf{A}$ and the trace of all the powers of a matrix $\mathbf{C}$ related to $\mathbf{A}$.

Lemma 5. *Let* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *be an SPD matrix. For any* α *with* $\lambda_1(\mathbf{A}) < \alpha$, *define* $\mathbf{B} := \mathbf{A}/\alpha$ *and* $\mathbf{C} := \mathbf{I}_n - \mathbf{B}$. *Then,*

$$\operatorname{logdet}(\mathbf{A}) = n \log(\alpha) - \sum_{k=1}^{\infty} \frac{\boldsymbol{tr}\left(\mathbf{C}^k\right)}{k}.$$

Proof. Observe that $\mathbf{B}$ is an SPD matrix with $\|\mathbf{B}\|_2 < 1$. It follows that

$$\begin{aligned} \operatorname{logdet}(\mathbf{A}) &= \log(\alpha^n \det(\mathbf{A}/\alpha)) \\ &= n \log(\alpha) + \log\left(\prod_{i=1}^{n} \lambda_i(\mathbf{B}) \right) \\ &= n \log(\alpha) + \sum_{i=1}^{n} \log(\lambda_i(\mathbf{B})) \\ &= n \log(\alpha) + \mathbf{tr}\left(\log\left[\mathbf{B}\right]\right). \end{aligned}$$

Here, we used standard properties of the determinant, standard properties of the logarithm function, and the fact that (recall that $\mathbf{B}$ is an SPD matrix),

$$\mathbf{tr}\left(\log\left[\mathbf{B}\right]\right) = \sum_{i=1}^{n} \lambda_i(\log\left[\mathbf{B}\right]) = \sum_{i=1}^{n} \log(\lambda_i(\mathbf{B})).$$

Now,

$$\mathbf{tr}\left(\log\left[\mathbf{B}\right]\right) = \mathbf{tr}\left(\log\left[\mathbf{I}_n - (\mathbf{I}_n - \mathbf{B})\right]\right) = \mathbf{tr}\left(-\sum_{k=1}^{\infty} \frac{(\mathbf{I}_n - \mathbf{B})^k}{k} \right) = -\sum_{k=1}^{\infty} \frac{\mathbf{tr}\left(\mathbf{C}^k\right)}{k}. \quad (6)$$

The second equality follows by the Taylor expansion because all the eigenvalues of $\mathbf{C} = \mathbf{I}_n - \mathbf{B}$ are contained[3] in $(0, 1)$ and the last equality follows by the linearity of the trace operator. □

3.1. Algorithm

Lemma 5 indicates the following high-level procedure for computing the logdet of an SPD matrix $\mathbf{A}$:

1. Compute some α with $\lambda_1(\mathbf{A}) < \alpha$.
2. Compute $\mathbf{C} = \mathbf{I}_n - \mathbf{A}/\alpha$.
3. Compute the trace of *all* the powers of $\mathbf{C}$.

To implement the first step in this procedure we use the power iteration from the numerical linear algebra literature (see Section 2.2). The second step is straightforward. To implement the third step, we keep a finite number of summands in the expansion $\sum_{k=1}^{\infty} \mathbf{tr}\left(\mathbf{C}^k\right)$. This step is important since the quality of the approximation, both theoretically and empirically, depends on the number of summands (denoted with m) that will be kept. On the other hand, the running time of the algorithm increases with m. Finally, to estimate the traces of the powers of $\mathbf{C}$, we use the randomized algorithm of Section 2.3. Our approach is described in detail in Algorithm 3; notice that step 7 in Algorithm 3 is an efficient way of computing

$$\widehat{\log\det}\,(\mathbf{A}) := n\log(\alpha) - \sum_{k=1}^{m}\left(\frac{1}{p}\sum_{i=1}^{p}\mathbf{g}_i^\top\mathbf{C}^k\mathbf{g}_i\right)/k.$$

Algorithm 3 Randomized log determinant estimation.

1: **INPUT**: $\mathbf{A} \in \mathbb{R}^{n\times n}$, accuracy parameter $\varepsilon > 0$, and integer $m > 0$.
2: Compute $\tilde{\lambda}_1(\mathbf{A})$ using Algorithm 1 with (integers)$t = \mathcal{O}(\log n)$ and $q = \mathcal{O}(\log(1/\delta))$
3: Pick $\alpha = 7\tilde{\lambda}_1(\mathbf{A})$
4: Set $\mathbf{C} = \mathbf{I}_n - \mathbf{A}/\alpha$
5: Set $p = \lceil 20\log(2/\delta)/\varepsilon^2 \rceil$
6: Let $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_p \in \mathbb{R}^n$ be i.i.d. random Gaussian vectors.
7: For $i = 1, 2 \ldots, p$
- $\mathbf{v}_1^{(i)} = \mathbf{C}\mathbf{g}_i$ and $\gamma_1^{(i)} = \mathbf{g}_i^\top\mathbf{v}_1^{(i)}$
- For $k = 2, \ldots, m$
  1. $\mathbf{v}_k^{(i)} := \mathbf{C}\mathbf{v}_{k-1}^{(i)}$.
  2. $\gamma_k^{(i)} = \mathbf{g}_i^\top\mathbf{v}_k^{(i)}$ (Inductively $\gamma_k^{(i)} = \mathbf{g}_i^\top\mathbf{C}^k\mathbf{g}_i$)
- EndFor

8: EndFor
9: **OUTPUT:** $\widehat{\log\det}\,(\mathbf{A}) = n\log(\alpha) - \sum_{k=1}^{m}\left(\frac{1}{p}\sum_{i=1}^{p}\gamma_k^{(i)}\right)/k$

[3] Indeed, $\lambda_i(\mathbf{C}) = 1 - \lambda_i(\mathbf{B})$ and $0 < \lambda_i(\mathbf{B}) < 1$ for all $i = 1 \ldots n$.

3.2. Error bound

The following lemma proves that Algorithm 3 returns an accurate approximation to the logdet of $\mathbf{A}$.

Lemma 6. *Let* $\widehat{\text{logdet}}\,(\mathbf{A})$ *be the output of Algorithm 3 on inputs* $\mathbf{A}$, m, *and* ε. *Then, with probability at least* $1-2\delta$,

$$\left|\widehat{\text{logdet}}\,(\mathbf{A}) - \text{logdet}\,(\mathbf{A})\right| \leq \left(\epsilon + \left(1 - \frac{1}{7\kappa\,(\mathbf{A})}\right)^m\right) \cdot \Gamma,$$

where $\Gamma = \sum_{i=1}^{n} \log\left(7 \cdot \frac{\lambda_1(\mathbf{A})}{\lambda_i(\mathbf{A})}\right)$. *If* $m \geq \lceil 7\kappa\,(\mathbf{A}) \log\left(\frac{1}{\varepsilon}\right)\rceil$, *then*

$$\left|\widehat{\text{logdet}}\,(\mathbf{A}) - \text{logdet}\,(\mathbf{A})\right| \leq 2\epsilon\Gamma.$$

Proof. First, note that using our choice for α in Step 3 of Algorithm 3 and applying Lemma 3, we get that, with probability at least $1-\delta$,

$$\lambda_1(\mathbf{A}) < \frac{7}{6}\lambda_1(\mathbf{A}) \leq \alpha \leq 7\lambda_1(\mathbf{A}). \tag{7}$$

The strict inequality at the leftmost side of the above equation follows since all eigenvalues of $\mathbf{A}$ are strictly positive. Let's call the event that the above inequality holds $\mathcal{E}_1$; obviously, $\mathbb{P}\left[\mathcal{E}_1\right] \geq 1-\delta$ (and thus $\mathbb{P}\left[\bar{\mathcal{E}}_1\right] \leq \delta$). We condition all further derivations on $\mathcal{E}_1$ holding and we manipulate $\Delta = \left|\widehat{\text{logdet}}\,(\mathbf{A}) - \text{logdet}\,(\mathbf{A})\right|$ as follows:

$$\begin{aligned}
\Delta &= \left|\sum_{k=1}^{m}\left(\frac{1}{p}\sum_{i=1}^{p}\mathbf{g}_i^\top\mathbf{C}^k\mathbf{g}_i\right)/k - \sum_{k=1}^{\infty}\mathbf{tr}\left(\mathbf{C}^k\right)/k\right| \\
&\leq \left|\sum_{k=1}^{m}\left(\frac{1}{p}\sum_{i=1}^{p}\mathbf{g}_i^\top\mathbf{C}^k\mathbf{g}_i\right)/k - \sum_{k=1}^{m}\mathbf{tr}\left(\mathbf{C}^k\right)/k\right| + \left|\sum_{k=m+1}^{\infty}\mathbf{tr}\left(\mathbf{C}^k\right)/k\right| \\
&= \underbrace{\left|\frac{1}{p}\sum_{i=1}^{p}\mathbf{g}_i^\top\left(\sum_{k=1}^{m}\mathbf{C}^k/k\right)\mathbf{g}_i - \mathbf{tr}\left(\sum_{k=1}^{m}\mathbf{C}^k/k\right)\right|}_{\Delta_1} + \underbrace{\left|\sum_{k=m+1}^{\infty}\mathbf{tr}\left(\mathbf{C}^k\right)/k\right|}_{\Delta_2}.
\end{aligned}$$

Below, we bound the two terms Δ_1 and Δ_2 separately. We start with Δ_1: the idea is to apply Lemma 4 on the matrix $\sum_{k=1}^{m}\mathbf{C}^k/k$ with $p = \lceil 20\log(2/\delta)/\varepsilon^2\rceil$. Let $\mathcal{E}_2$ denote the probability that Lemma 4 holds; obviously, $\mathbb{P}\left[\mathcal{E}_2\right] \geq 1-\delta$ (and thus $\mathbb{P}\left[\bar{\mathcal{E}}_2\right] \leq \delta$) given our choice of p. We condition all further derivations on $\mathcal{E}_2$ holding as well to get

$$\Delta_1 \leq \varepsilon \cdot \mathbf{tr}\left(\sum_{k=1}^{m}\mathbf{C}^k/k\right) \leq \varepsilon \cdot \mathbf{tr}\left(\sum_{k=1}^{\infty}\mathbf{C}^k/k\right).$$

In the last inequality we used the fact that $\mathbf{C}$ is a positive matrix, hence for all k, $\mathbf{tr}\left(\mathbf{C}^k\right) > 0$. The second term Δ_2 is bounded as follows:

$$\begin{aligned}
\Delta_2 &= \left| \sum_{k=m+1}^{\infty} \mathbf{tr}\left(\mathbf{C}^k\right)/k \right| \leq \sum_{k=m+1}^{\infty} \mathbf{tr}\left(\mathbf{C}^k\right)/k \\
&= \sum_{k=m+1}^{\infty} \mathbf{tr}\left(\mathbf{C}^m \cdot \mathbf{C}^{k-m}\right)/k \leq \sum_{k=m+1}^{\infty} \|\mathbf{C}^m\|_2 \cdot \mathbf{tr}\left(\mathbf{C}^{k-m}\right)/k \\
&= \|\mathbf{C}^m\|_2 \cdot \sum_{k=m+1}^{\infty} \mathbf{tr}\left(\mathbf{C}^{k-m}\right)/k \leq \|\mathbf{C}^m\|_2 \cdot \sum_{k=1}^{\infty} \mathbf{tr}\left(\mathbf{C}^k\right)/k \\
&\leq \left(1 - \frac{\lambda_n(\mathbf{A})}{\alpha}\right)^m \cdot \sum_{k=1}^{\infty} \mathbf{tr}\left(\mathbf{C}^k\right)/k.
\end{aligned}$$

In the first inequality, we used the triangle inequality and the fact that $\mathbf{C}$ is a positive matrix. In the second inequality, we used the following fact[4]: given two positive semidefinite matrices $\mathbf{A}, \mathbf{B}$ of the same size, $\mathbf{tr}(\mathbf{AB}) \leq \|\mathbf{A}\|_2 \cdot \mathbf{tr}(\mathbf{B})$. In the last inequality, we used the fact that

$$\lambda_1(\mathbf{C}) = 1 - \lambda_n(\mathbf{B}) = 1 - \lambda_n(\mathbf{A})/\alpha.$$

Combining the bounds for Δ_1 and Δ_2 gives

$$\left| \widehat{\text{logdet}}(\mathbf{A}) - \text{logdet}(\mathbf{A}) \right| \leq \left(\epsilon + \left(1 - \frac{\lambda_n(\mathbf{A})}{\alpha}\right)^m \right) \cdot \sum_{k=1}^{\infty} \frac{\mathbf{tr}\left(\mathbf{C}^k\right)}{k}.$$

We have already proven in Lemma 5 that

$$\sum_{k=1}^{\infty} \frac{\mathbf{tr}\left(\mathbf{C}^k\right)}{k} = -\mathbf{tr}\left(\mathbf{log}\left[\mathbf{B}\right]\right) = n\log(a) - \text{logdet}(\mathbf{A}).$$

Notice that the assumption of Lemma 5 (namely, $\lambda_1(\mathbf{A}) < \alpha$) is satisfied from the inequality of eqn. (7). We further manipulate the last term as follows:

$$\begin{aligned}
n\log(a) - \text{logdet}(\mathbf{A}) &= n\log(\alpha) - \log(\prod_{i=1}^{n} \lambda_i(\mathbf{A})) \\
&= n\log(\alpha) - \sum_{i=1}^{n} \log(\lambda_i(\mathbf{A}))
\end{aligned}$$

[4] This follows from Von Neumann's trace inequality.

$$= \sum_{i=1}^{n} \left(\log(\alpha) - \log(\lambda_i(\mathbf{A}))\right)$$

$$= \sum_{i=1}^{n} \log\left(\frac{\alpha}{\lambda_i(\mathbf{A})}\right).$$

Collecting our results together, we get:

$$\left|\widehat{\text{logdet}}(\mathbf{A}) - \text{logdet}(\mathbf{A})\right| \leq \left(\epsilon + \left(1 - \frac{\lambda_n(\mathbf{A})}{\alpha}\right)^m\right) \cdot \sum_{i=1}^{n} \log\left(\frac{\alpha}{\lambda_i(\mathbf{A})}\right).$$

Using the inequality of eqn. (7) (only the upper bound on α is needed here) proves the first inequality of the lemma. To prove the second inequality, we use the well-known fact that $\left(1 - x^{-1}\right)^x \leq e^{-1}$ (where $e = 2.718\ldots$ and $x > 0$) and our choice for m.

Finally, recall that we conditioned all derivations on events $\mathcal{E}_1$ and $\mathcal{E}_2$ both holding, which can be bounded as follows:

$$\mathbb{P}\left[\mathcal{E}_1 \cap \mathcal{E}_2\right] = 1 - \mathbb{P}\left[\bar{\mathcal{E}}_1 \cup \bar{\mathcal{E}}_2\right] \geq 1 - \mathbb{P}\left[\bar{\mathcal{E}}_1\right] - \mathbb{P}\left[\bar{\mathcal{E}}_1\right] \geq 1 - 2\delta.$$

The first inequality in the above derivation follows from the union bound. □

3.3. Running time

Step 2 takes $\mathcal{O}(nnz(\mathbf{A})\log(n)\log(1/\delta))$ time; we assume that $nnz(\mathbf{A}) \geq n$, since otherwise the determinant of $\mathbf{A}$ would be trivially equal to zero. For each $k > 0$, $\mathbf{v}_k = \mathbf{C}^k\mathbf{g}_i$. The algorithm inductively computes $\mathbf{v}_k$ and $\mathbf{g}_i^\top\mathbf{C}^k\mathbf{g}_i = \mathbf{g}_i^\top\mathbf{v}_k$ for all $k = 1, 2, \ldots, m$. Given $\mathbf{v}_{k-1}$, $\mathbf{v}_k$ and $\mathbf{g}_i^\top\mathbf{C}^k\mathbf{g}_i$ can be computed in $nnz(\mathbf{C})$ and $\mathcal{O}(n)$ time, respectively. Notice that $nnz(\mathbf{C}) \leq n + \text{nnz}(\mathbf{A})$. Therefore, step 7 requires $\mathcal{O}(p \cdot m \cdot \text{nnz}(\mathbf{A}))$ time. Since $p = O(\varepsilon^{-2}\log(1/\delta))$, the total cost is

$$\mathcal{O}\left(\text{nnz}(\mathbf{A}) \cdot \left(\frac{m}{\varepsilon^2} + \log n\right) \cdot \log\left(\frac{1}{\delta}\right)\right).$$

4. Relative error approximation for SPD matrices with bounded eigenvalues

In this section, we argue that a simplified version of Algorithm 3 achieves a relative error approximation to the logdet of the SPD matrix $\mathbf{A}$, under the assumption that all the eigenvalues of $\mathbf{A}$ lie in the interval $(\theta_1, 1)$, where $0 < \theta_1 < 1$. This is a mild generalization of the setting introduced in [16].

Given the upper bound on the largest eigenvalue of $\mathbf{A}$, the proof of the following lemma (which is the analog of Lemma 5) is straightforward.

Lemma 7. *Let* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *be an SPD matrix whose eigenvalues lie in the interval* $(\theta_1, 1)$, *for some* $0 < \theta_1 < 1$. *Let* $\mathbf{C} := \mathbf{I}_n - \mathbf{A}$; *then,*

$$\text{logdet}\left(\mathbf{A}\right) = -\sum_{k=1}^{\infty} \frac{\boldsymbol{tr}\left(\mathbf{C}^k\right)}{k}.$$

Proof. Similarly to the proof of Lemma 5,

$$\text{logdet}\left(\mathbf{A}\right) = \log\left(\prod_{i=1}^{n} \lambda_i(\mathbf{A})\right) = \sum_{i=1}^{n} \log(\lambda_i(\mathbf{A})) = \mathbf{tr}\left(\mathbf{log}\left[\mathbf{A}\right]\right).$$

Now,

$$\mathbf{tr}\left(\mathbf{log}\left[\mathbf{A}\right]\right) = \mathbf{tr}\left(\mathbf{log}\left[\mathbf{I}_n - \left(\mathbf{I}_n - \mathbf{A}\right)\right]\right) = \mathbf{tr}\left(-\sum_{k=1}^{\infty} \frac{\left(\mathbf{I}_n - \mathbf{A}\right)^k}{k}\right) = -\sum_{k=1}^{\infty} \frac{\mathbf{tr}\left(\mathbf{C}^k\right)}{k}.$$

The second equality follows by the Taylor expansion since all the eigenvalues of $\mathbf{C} = \mathbf{I}_n - \mathbf{A}$ are contained in the interval $(0, 1)$. □

4.1. The algorithm and the relative error bound

We simplify Algorithm 3 as follows: we skip steps 2 and 3 and in step 4 we set $\mathbf{C} = \mathbf{I}_n - \mathbf{A}$. The following lemma proves that in this special case the modified algorithm returns a relative error approximation to the log determinant of the input matrix $\mathbf{A}$.

Lemma 8. *Let* $\widehat{\text{logdet}}\left(\mathbf{A}\right)$ *be the output of the (modified) Algorithm 3 on inputs* $\mathbf{A}$ *and* ε. *Then, with probability at least* $1 - \delta$,

$$\left|\widehat{\text{logdet}}\left(\mathbf{A}\right) - \text{logdet}\left(\mathbf{A}\right)\right| \leq 2\varepsilon \cdot \left|\text{logdet}\left(\mathbf{A}\right)\right|.$$

Proof. Similarly to the proof of Lemma 6, we manipulate $\Delta = \left|\widehat{\text{logdet}}\left(\mathbf{A}\right) - \text{logdet}\left(\mathbf{A}\right)\right|$ as follows:

$$\begin{aligned}
\Delta &= \left|\sum_{k=1}^{m}\left(\frac{1}{p}\sum_{i=1}^{p}\mathbf{g}_i^\top\mathbf{C}^k\mathbf{g}_i\right)/k - \sum_{k=1}^{\infty}\mathbf{tr}\left(\mathbf{C}^k\right)/k\right| \\
&\leq \left|\sum_{k=1}^{m}\left(\frac{1}{p}\sum_{i=1}^{p}\mathbf{g}_i^\top\mathbf{C}^k\mathbf{g}_i\right)/k - \sum_{k=1}^{m}\mathbf{tr}\left(\mathbf{C}^k\right)/k\right| + \left|\sum_{k=m+1}^{\infty}\mathbf{tr}\left(\mathbf{C}^k\right)/k\right| \\
&= \underbrace{\left|\frac{1}{p}\sum_{i=1}^{p}\mathbf{g}_i^\top\left(\sum_{k=1}^{m}\mathbf{C}^k/k\right)\mathbf{g}_i - \mathbf{tr}\left(\sum_{k=1}^{m}\mathbf{C}^k/k\right)\right|}_{\Delta_1} + \underbrace{\left|\sum_{k=m+1}^{\infty}\mathbf{tr}\left(\mathbf{C}^k\right)/k\right|}_{\Delta_2}.
\end{aligned}$$

We now bound the two terms Δ_1 and Δ_2 separately. We start with Δ_1: the idea is to apply Lemma 4 on the matrix $\sum_{k=1}^{m} \mathbf{C}^k/k$ with $p = \lceil 20 \log(2/\delta)/\varepsilon^2 \rceil$. Hence, with probability at least $1-\delta$ (this is the only probabilistic event in this lemma and hence $1-\delta$ is a lower bound on the success probability of the lemma):

$$\Delta_1 \leq \varepsilon \cdot \mathbf{tr}\left(\sum_{k=1}^{m} \mathbf{C}^k/k\right) \leq \varepsilon \cdot \mathbf{tr}\left(\sum_{k=1}^{\infty} \mathbf{C}^k/k\right).$$

In the last inequality we used the fact that $\mathbf{C}$ is positive definite, hence for all k, $\mathbf{tr}\left(\mathbf{C}^k\right) > 0$. Bounding Δ_2 follows the lines of the proof of Lemma 6:

$$\begin{aligned}
\Delta_2 &= \left|\sum_{k=m+1}^{\infty} \mathbf{tr}\left(\mathbf{C}^k\right)/k\right| = \left|\sum_{k=m+1}^{\infty} \mathbf{tr}\left(\mathbf{C}^m \cdot \mathbf{C}^{k-m}\right)/k\right| \\
&\leq \left|\sum_{k=m+1}^{\infty} \|\mathbf{C}^m\|_2 \cdot \mathbf{tr}\left(\mathbf{C}^{k-m}\right)/k\right| = \|\mathbf{C}^m\|_2 \cdot \left|\sum_{k=m+1}^{\infty} \mathbf{tr}\left(\mathbf{C}^{k-m}\right)/k\right| \\
&\leq \|\mathbf{C}^m\|_2 \cdot \left|\sum_{k=1}^{\infty} \mathbf{tr}\left(\mathbf{C}^k\right)/k\right| \leq (1-\lambda_n(\mathbf{A}))^m \left|\sum_{k=1}^{\infty} \mathbf{tr}\left(\mathbf{C}^k\right)/k\right|.
\end{aligned}$$

In the last inequality, we used the fact that $\lambda_1(\mathbf{C}) = 1 - \lambda_n(\mathbf{A})$. Combining the bounds for Δ_1 and Δ_2 gives

$$\left|\widehat{\mathrm{logdet}}(\mathbf{A}) - \mathrm{logdet}(\mathbf{A})\right| \leq (\epsilon + (1-\lambda_n(\mathbf{A}))^m) \cdot \sum_{k=1}^{\infty} \frac{\mathbf{tr}\left(\mathbf{C}^k\right)}{k}.$$

We have already proven in Lemma 7 that

$$\sum_{k=1}^{\infty} \frac{\mathbf{tr}\left(\mathbf{C}^k\right)}{k} = -\mathbf{tr}\left(\log\left[\mathbf{A}\right]\right) = -\mathrm{logdet}(\mathbf{A}).$$

Collecting our results, we get:

$$\left|\widehat{\mathrm{logdet}}(\mathbf{A}) - \mathrm{logdet}(\mathbf{A})\right| \leq (\varepsilon + (1-\lambda_n(\mathbf{A}))^m) \cdot \left|\mathrm{logdet}(\mathbf{A})\right|.$$

Using $1-\lambda_n(\mathbf{A}) < 1-\theta_1$, we conclude that

$$\left|\widehat{\mathrm{logdet}}(\mathbf{A}) - \mathrm{logdet}(\mathbf{A})\right| \leq (\varepsilon + (1-\theta_1)^m) \cdot \left|\mathrm{logdet}(\mathbf{A})\right|.$$

Setting

$$m = \left\lceil \frac{1}{\theta_1} \cdot \log\left(\frac{1}{\varepsilon}\right) \right\rceil$$

and using $\left(1-x^{-1}\right)^x \leq e^{-1}$ (where $e = 2.718\ldots$ and $x > 0$), guarantees that $(1-\theta_1)^m \leq \varepsilon$ and concludes the proof of the lemma. □

We conclude by discussing the running time of the simplified Algorithm 3, which is equal to $\mathcal{O}(p \cdot m \cdot nnz(\mathbf{A}))$. Since $p = \mathcal{O}\left(\frac{\log(1/\delta)}{\varepsilon^2}\right)$ and $m = \mathcal{O}\left(\frac{\log(1/\varepsilon)}{\theta_1}\right)$, the running time becomes

$$\mathcal{O}\left(\frac{\log(1/\varepsilon)\log(1/\delta)}{\varepsilon^2\theta_1} nnz(\mathbf{A})\right).$$

5. Experiments

The goal of our experimental section is to establish that our approximation to logdet $(\mathbf{A})$ (as computed by Algorithm 3) is both accurate and fast for both dense and sparse matrices. The accuracy of Algorithm 3 is measured by comparing its result against the exact logdet $(\mathbf{A})$ computed via the Cholesky factorization. The rest of this section is organized as follows: in Section 5.1, we describe our software for approximating logdet $(\mathbf{A})$; in Section 5.2 we describe the computational environment that we used; and in Sections 5.3 and 5.4 we discuss experimental results for dense and sparse SPD matrices, respectively.

5.1. Software

We developed high-quality, shared- and distributed-memory parallel C++ code for the algorithms listed in this paper. All of the code that was developed for this paper is hosted at http://web.ics.purdue.edu/~ekontopo/software.html. In it's current state, our software supports: (1) ingesting dense (binary and text format) and sparse (binary, text, and matrix market format) matrices, (2) generating large random SPD matrices, (3) computing both approximate and exact spectral norms of matrices, (4) computing both approximate and exact traces of matrices, and (5) computing both approximate and exact log determinants of matrices. Currently, we support both Eigen [10] and Elemental [25] matrices. The Eigen software package supports both dense and sparse matrices, while the Elemental software package mostly supports dense matrices and only recently added support for sparse matrices (pre-release). As we wanted the random SPD generation to be fast, we have used parallel random number generators from Random123 [28] in conjunction with Boost.Random.

5.2. Environment

All our experiments were run on "Nadal", a 60-core machine, where each core is an Intel® Xeon® E7-4890 machine running at 2.8 Ghz. Nadal has 1 TB of RAM and runs Linux kernel version 2.6-32. For compilation, we used GCC 4.9.2. We used Eigen 3.2.4, OpenMPI 1.8.4, Boost 1.55.7, and the latest version of Elemental at https://github.com/elemental. For experiments with Elemental, we used OpenBlas, which is an extension

of GotoBlas [13], for its parallel prowess; Eigen has built-in the BLAS and LAPACK packages.

5.3. Dense matrices

Data generation. In our experiments, we used two types of synthetic SPD matrices. The first type were diagonally dominant SPD matrices and were generated as follows. First, we created $\mathbf{X} \in \mathbb{R}^{n \times n}$ by drawing n^2 entries from a uniform sphere with center 0.5 and radius 0.25. Then, we generated a symmetric matrix $\mathbf{Y}$ by setting

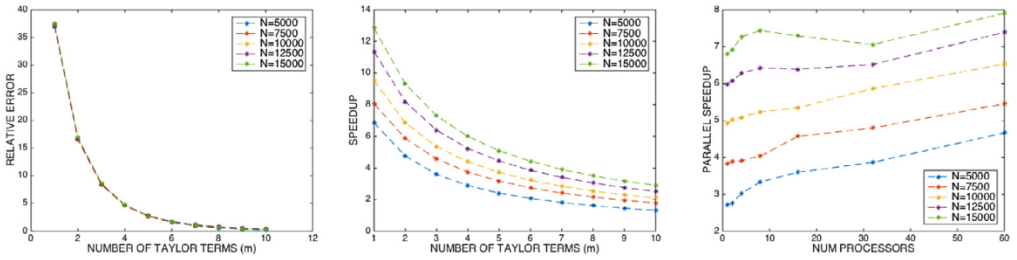
$$\mathbf{Y} = 0.5 * (\mathbf{X} + \mathbf{X}^\top).$$

Finally, we ensured that the desired matrix $\mathbf{A}$ is positive definite by adding the value n to each diagonal entry [3] of $\mathbf{Y}$: $\mathbf{A} = \mathbf{Y} + n\mathbf{I}_n$. We call this method `randSPDDenseDD`.

The second approach generates SPD matrices that are not diagonally dominant. We created $\mathbf{X}, \mathbf{D} \in \mathbb{R}^{n \times n}$ by drawing n^2 and n entries, respectively, from a uniform sphere with center 0.5 and radius 0.25; $\mathbf{D}$ is a diagonal matrix with small entries. Next, we generated an orthogonal random matrix $\mathbf{Q} = \mathrm{qr}(\mathbf{X})$. Thus, $\mathbf{Q}$ is an orthonormal basis for $\mathbf{X}$. Finally, we generated $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T$. We call this method `randSPDDense`. `randSPDDense` is more expensive than `randSPDDenseDD`, as it requires an additional $O(n^3)$ computations for the QR factorization and the matrix-matrix product.

Evaluation. To evaluate the runtime of Algorithm 3 against a baseline, we used the Cholesky decomposition to compute the $\mathrm{logdet}(\mathbf{A})$. More specifically, we computed $\mathbf{A} = \mathbf{L}\mathbf{L}^T$ and returned $\mathrm{logdet}(\mathbf{A}) = 2 \cdot \mathrm{logdet}(\mathbf{L})$. Since Elemental provides distributed and shared memory parallelism, we restricted ourselves to experiments with Elemental matrices throughout this section. Note that we measured the accuracy of the approximate algorithm in terms of the relative error to ensure that we have numbers of the same scale for matrices with vastly different values for $\mathrm{logdet}(\mathbf{A})$; we defined the relative error e as $e = 100(x - \tilde{x})/x$, where x is the true value and $\tilde{x}$ is the approximation. Similarly, we defined the speedup s as $s = t_x/t_{\tilde{x}}$, where t_x is the time needed to compute x and $t_{\tilde{x}}$ is the time needed to compute the approximation $\tilde{x}$.

Results. For dense matrices, we first used synthetic matrices generated using `randSPDDense`; these are relatively ill-conditioned matrices. We experimented with values of n (number of rows and columns of $\mathbf{A}$) in the set $\{5,000,\ 7,500,\ 10,000,\ 12,500,\ 15,000\}$. The three key points pertaining to these matrices are shown in Fig. 1. First, we discuss the effect of m, the number of terms in the Taylor series used to approximate $\mathrm{logdet}(\mathbf{A})$; Fig. 1(a) depicts our results for the sequential case. On the y-axis, we see the relative error, which is measured against the exact $\mathrm{logdet}(\mathbf{A})$ as computed via the Cholesky factorization. We observe that for these ill-conditioned matrices, for small values of m (less than four) the relative error is high. However, for all values of $m \geq 4$, we observe that the error drops significantly and stabilizes. We note that in each iteration, all random processes were re-seeded with new values; we have plotted the



(a) Accuracy vs. m (b) Speedup vs. m (c) Parallel Speedup

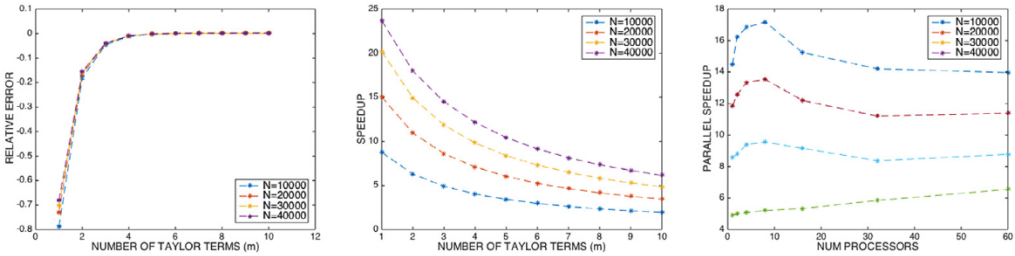
Fig. 1. Panels 1(a) and 1(b) depict the effect of m (see Algorithm 3) on the accuracy of the approximation and the time to completion, respectively, for dense matrices generated by `randSPDDense`. For all the panels, $p = 60$ and $t = 2\log\sqrt{4n}$. The baseline for all experiments was the Cholesky factorization, which was used to compute the exact value of logdet $(\mathbf{A})$. For panels 1(a) and 1(b), the number of cores, np, was set to one. The last panel 1(c) depicts the relative speedup of the approximate algorithm when compared to the baseline solver (at $m = 4$). Elemental was used as the backend for these experiments. For the approximate algorithm, we report the mean and standard deviation of ten iterations.

Table 1
Accuracy and sequential running times (at $p = 60$, $m = 4$ and $t = \log\sqrt{4n}$) for dense random matrices generated using `randSPDDense`. Baselines were computed using the Cholesky factorization; mean and standard deviation are reported over ten iterations.

n	logdet $(\mathbf{A})$			Time (secs)		
	Exact	Mean	std	Exact	Mean	std
5000	−3717.89	−3546.920	8.10	2.56	1.15	0.0005
7500	−5474.49	−5225.152	8.73	7.98	2.53	0.0015
10000	−7347.33	−7003.086	7.79	18.07	4.47	0.0006
12500	−9167.47	−8734.956	17.43	34.39	7.00	0.0030
15000	−11100.9	−10575.16	15.09	58.28	10.39	0.0102

error bars throughout Fig. 1. The standard deviation for both accuracy and time was consistently small; indeed, it is not visible to the naked eye at scale. To see the benefit of approximation, we look at Fig. 1(b) together with Fig. 1(a). For example, at $m = 4$, for all matrices, we get at least a factor of two speedup. As n gets larger, the speedups of the approximation also increase. For example, for $n = 15,000$, the speedup at $m = 4$ is nearly six-fold. In terms of accuracy, Fig. 1(a) shows that at $m = 4$, the relative error is approximately 4%. This speedup is expected as the Cholesky factorization requires $O(n^3)$ operations; Algorithm 3 only relies on matrix-matrix products where one of the matrices has a small number of columns (equal to p), which is independent of n.

Finally, we discuss the parallel speedup in Fig. 1(c), which shows the relative speedup of the approximate algorithm with respect to the baseline Cholesky algorithm. For this evaluation, we set $m = 4$ and varied the number of processes, denoted by np, from 1 to 60. The main take away from Fig. 1(c) is that the approximate algorithm provides nearly the same or increasingly better speedups relative to a parallelized version of the exact (Cholesky) algorithm. For example, for $n = 15,000$, the speedups for using the approximate algorithm are consistently better that $6.5x$. The absolute values for logdet $(\mathbf{A})$ and timing along with the baseline numbers for this experiment are given in Table 1. We report the numbers in Table 1 at $m = 4$ at which point, we have low relative error.



(a) Accuracy vs. m (b) Speedup vs. m (c) Parallel Speedup

Fig. 2. Panels 2(a) and 2(b) depict the effect of m (see Algorithm 3) on the accuracy of the approximation and the time to completion, respectively, for diagonally dominant dense random matrices generated by `randSPDDenseDD`. For all the panels, $p = 60$ and $t = 2\log\sqrt{4n}$. The baseline for all experiments was the Cholesky factorization, which was used to compute the exact value of logdet $(\mathbf{A})$. For panels 2(a) and 2(b), the number of cores, np, was set to one. The last panel 2(c) depicts the relative speedup of the approximate algorithm when compared to the baseline solver (at $m = 2$). Elemental was used as the backend for these experiments. For the approximate algorithm, we report the mean and standard deviation over ten iterations.

Table 2
Accuracy and sequential running times (at $p = 60$, $m = 2$, and $t = 2\log\sqrt{4n}$) for diagonally dominant dense random matrices generated using `randSPDDenseDD`. Baselines were computed using the Cholesky factorization; mean and standard deviation are reported over ten iterations.

n	logdet $(\mathbf{A})$			Time (secs)		
	Exact	Mean	std	Exact	Mean	std
10000	92103.1	92269.5	5.51	18.09	2.87	0.01
20000	198069.0	198397.4	9.60	135.92	12.41	0.02
30000	309268.0	309763.8	20.04	448.02	30.00	0.12
40000	423865.0	424522.4	14.80	1043.74	58.05	0.05

For the second set of dense experiments, we generated diagonally dominant matrices using `randSPDDenseDD`; we were able to quickly generate and run benchmarks on matrices of sizes $n \times n$ with n in the set $\{10,000,\ 20,000,\ 30,000,\ 40,000\}$ due to the relatively simpler procedure involved in matrix generation. In this set of experiments, due to the diagonal dominance, all matrices were well-conditioned. The results of our experiments on these well-conditioned matrices are presented in Fig. 2 and show a marked improved over the results presented in Fig. 1. First, notice that very few terms of the Taylor series (i.e., small m) are sufficient to get high accuracy approximations; this is apparent in Fig. 2(a). In fact, we see that even at $m = 2$, we are near convergence and at $m = 3$, for most of the matrices, we have near-zero relative error. This experimental result, combined with Fig. 2(b) is particularly encouraging; at $m = 2$, we seem to not only have a nearly lossless approximation of logdet $(\mathbf{A})$, but also have at least a five-fold speedup. Similarly to Fig. 1, the speedups are better for larger matrices. For example, for $n = 40,000$, the speedup at $m = 2$ is nearly twenty-fold. We conclude our analysis by presenting Fig. 2(c), which similarly to Fig. 1(c), points out that at any level of parallelism, Algorithm 3 maintains its relative performance over the exact (Cholesky) factorization. The absolute values for logdet $(\mathbf{A})$ and the corresponding running times, along with the baseline for this experiment are presented in Table 2. We report the numbers in Table 2 at $m = 2$, at which point we have a low relative error.

Table 3
Description of the SPD matrices from the University of Florida sparse matrix collection [7] that were used in our experiments. All experiments were run sequentially ($np = 1$) using Eigen. Accuracy results for Algorithm 3 are reported using both the mean and the standard deviation over ten iterations at (with $t = 5$ and $p = 5$); we only report the mean for the running times, since the standard deviation is negligible. The exact logdet $(\mathbf{A})$ was computed using the Cholesky factorization.

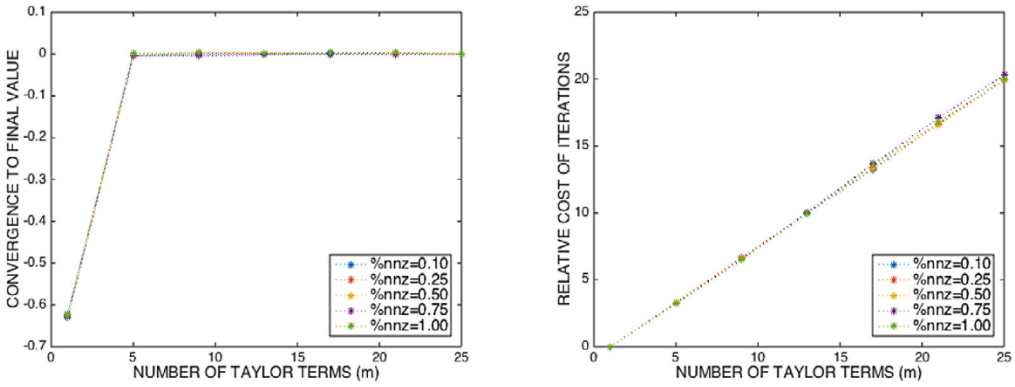
Name	n	nnz	Area of origin	logdet $(\mathbf{A})$			Time (sec)		m
				Exact	Approx		Exact	Approx	
					Mean	std		Mean	
thermal2	1228045	8580313	Thermal	1.3869e6	1.3928e6	964.79	31.28	31.24	149
ecology2	999999	4995991	2D/3D	3.3943e6	3.403e6	1212.8	18.5	10.47	125
ldoor	952203	42493817	Structural	1.4429e7	1.4445e7	1683.5	117.91	17.60	33
thermomech_TC	102158	711558	Thermal	−546787	−546829.4	553.12	57.84	2.58	77
boneS01	127224	5516602	Model reduction	1.1093e6	1.106e6	247.14	130.4	8.48	125

5.4. Sparse matrices

Data synthesis. To generate a sparse, synthetic matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, with nnz non-zeros, we use a Bernoulli distribution to determine the location of the non-zero entries and a uniform distribution to generate the values. First, we completely fill the n principle diagonal entries. Next, we generate $(nnz - n)/2$ index positions in the upper triangle for the non-zero entries by sampling from a Bernoulli distribution with probability $(nnz - n)/(n^2 - n)$. We reflect each entry across the principle diagonal to ensure that $\mathbf{A}$ is symmetric and we add n to each diagonal entry to ensure that $\mathbf{A}$ is SPD (actually, $\mathbf{A}$ is also diagonally dominant).

Real data. To demonstrate the prowess of Algorithm 3 on real-world data, we used SPD matrices from the University of Florida's sparse matrix collection [7]. The complete list of matrices from this collection used in our experiments, as well as a brief description of each matrix, is given in columns 1–4 of Table 3.

Evaluation. It is tricky to pick any single method as the "exact method" to compute the logdet $(\mathbf{A})$ for a sparse SPD matrix $\mathbf{A}$. One approach would be to use direct methods such as Cholesky decomposition of $\mathbf{A}$ [5,12]. For direct methods, it is difficult to derive an analytical solution for the number of operations required for the factorization as a function of the number of non-zero entries of the matrix, as this is highly dependent on the structure of the matrix [11]. In the distributed setting, one also needs to consider the volume of communication involved, which is often the bottleneck. Alternately, we can use iterative methods to compute the eigenvalues of $\mathbf{A}$ [4] and use the eigenvalues to compute logdet $(\mathbf{A})$. It is clear that the worst case performance of both the direct and iterative methods is $O(n^3)$. However, iterative methods are typically used to compute a few eigenvalues and eigenvectors: therefore, we chose to use the Cholesky factorization based on matrix reordering to compute the exact value of logdet $(\mathbf{A})$. It is important to note that both the direct and iterative methods are notoriously hard to implement, which comes to stark contrast with the almost trivial implementation of Algorithm 3, which is also readily parallelizable.



(a) Convergence with m (b) Cost as a function of m

Fig. 3. Panels 3(a) and 3(b) depict the effect of the number of terms in the Taylor expansion, m, (see Algorithm 3) on the convergence to the final solution and the time to completion of the approximation. The matrix size was fixed at $n = 10^6$ and sparsity was varied as $0.1\%, 0.25\%, 0.5\%, 0.75\%$, and 1%. Experiments were run sequentially ($np = 1$) and we set $p = 60$, $t = 2\log\sqrt{4n}$. For panel 3(a), the baseline is the final value of logdet $(\mathbf{A})$ at $m = 25$. For panel 3(b), the baseline is the time to completion of the approximation algorithm with $m = 1$. Eigen was used as the backend for these experiments.

Results. The true power of Algorithm 3 lies in its ability to approximate logdet $(\mathbf{A})$ for sparse $\mathbf{A}$. The Cholesky factorization can introduce $O(n^2)$ non-zeros during factorization due to fill-in; for many problems, there is insufficient memory to factorize a large, sparse matrix. In our first set of experiments, we wanted to show the effect of m on: (1) convergence of logdet $(\mathbf{A})$, and (2) cost of the solution. To this end, we generated sparse, diagonally dominant SPD matrices of size $n = 10^6$ and varied the sparsity from 0.1% to 1% in increments of 0.25%. We did not attempt to compute the exact logdet $(\mathbf{A})$ for these synthetic matrices — our aim was to merely study the speedup with m for different sparsities, while t and p were held constant at $2\log\sqrt{4n}$ and 60 respectively. The results are shown in Fig. 3. Fig. 3(a) depicts the convergence of logdet $(\mathbf{A})$ measured as a relative error of the current estimate over the final estimate. As can be seen — for well conditioned matrices – convergence is quick. Fig. 3(b) shows the relative cost of increasing m; here the baseline is $m = 1$. Therefore, the additional cost incurred by increasing m is linear when all other parameters are held constant.

The results of running Algorithm 3 on the UFL matrices are shown in Table 3. The numbers reported for the approximation are the mean and standard deviation over ten iterations, $t = 5$, and $p = 5$.[5] The value of m was varied between one and 150 in increments of five to select the best average accuracy. The matrices shown in Table 3 have a nice structure, which lends itself to nice reorderings and therefore an efficient computation of the Cholesky factorization. We see that even in such cases, the performance

[5] We experimented with different p, t and settled on the smallest values that did not result in loss in accuracy.

of Algorithm 3 is commendable due to its lower algorithmic complexity. In the case of `thermomech_TC`, we achieve good accuracy while achieving a 22x speedup.

6. Conclusions

Prior work has presented approximation algorithms for the logarithm of the determinant of a symmetric positive definite matrix; those algorithms either do not work for all SPD matrices, or do not admit a worst-case theoretical analysis, or both. In this work, we presented an approximation algorithm to compute the logarithm of the determinant of a SPD matrix that comes with strong theoretical worst-case analysis bounds and can be applied to *any* SPD matrix. A simplification of our algorithm delivers relative-error approximation guarantees for a popular special case of SPD matrices. Using state-of-the-art C++ numerical linear algebra software packages for both dense and sparse matrices, we demonstrated that the proposed approximation algorithm performs remarkably well in practice in serial and parallel environments.

References

[1] H. Avron, S. Toledo, Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix, J. ACM 58 (2) (2011) 8.
[2] Ronald Paul Barry, R. Kelley Pace, Monte Carlo estimates of the log determinant of large sparse matrices, Linear Algebra Appl. 289 (1) (1999) 41–54.
[3] Paul F. Curran, On a variation of the Gershgorin circle theorem with applications to stability theory, in: IET Irish Signals and Systems Conference, ISSC 2009, 2009.
[4] Ernest R. Davidson, The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices, J. Comput. Phys. 17 (1) (1975) 87–94.
[5] Timothy A. Davis, Direct Methods for Sparse Linear Systems, vol. 2, SIAM, 2006.
[6] Alexandre d'Aspremont, Onureena Banerjee, Laurent El Ghaoui, First-order methods for sparse covariance selection, SIAM J. Matrix Anal. Appl. 30 (1) (2008) 56–66.
[7] Timothy A. Davis, Yifan Hu, The university of Florida sparse matrix collection, ACM Trans. Math. Software 38 (1) (2011) 1.
[8] Wayne Eberly, Mark Giesbrecht, Gilles Villard, On computing the determinant and smith form of an integer matrix, in: Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on, IEEE, 2000, pp. 675–685.
[9] Jerome Friedman, Trevor Hastie, Robert Tibshirani, Sparse inverse covariance estimation with the graphical lasso, Biostatistics 9 (3) (2008) 432–441.
[10] Gaël Guennebaud, Benoît Jacob, et al., Eigen v3, http://eigen.tuxfamily.org, 2010.
[11] Anshul Gupta, George Karypis, Vipin Kumar, Highly scalable parallel algorithms for sparse matrix factorization, IEEE Trans. Parallel Distrib. Syst. 8 (5) (1997) 502–520.
[12] Anshul Gupta, Wsmp: Watson sparse matrix package (part-i: direct solution of symmetric sparse systems), Tech. Rep. RC, 21886, IBM TJ Watson Research Center, Yorktown Heights, NY, 2000.
[13] Kazushige Goto, Robert Van De Geijn, High-performance implementation of the level-3 blas, ACM Trans. Math. Software 35 (1) (2008) 4.
[14] Timothy Hunter, Ahmed El Alaoui, Alexandre Bayen, Computing the log-determinant of symmetric, diagonally dominant matrices in near-linear time, arXiv preprint, arXiv:1408.1693, 2014.
[15] Insu Han, Dmitry Malioutov, Haim Avron, Jinwoo Shin, Approximating the spectral sums of large-scale matrices using Chebyshev approximations, arXiv preprint, arXiv:1606.00942, 2016.
[16] Insu Han, Dmitry Malioutov, Jinwoo Shin, Large-scale log-determinant computation through stochastic Chebyshev expansions, in: David Blei, Francis Bach (Eds.), Proceedings of the 32nd International Conference on Machine Learning, ICML-15, JMLR Workshop and Conference Proceedings, 2015, pp. 908–917.

[17] Cho-Jui Hsieh, Mátyás A. Sustik, Inderjit S. Dhillon, Pradeep K. Ravikumar, Russell Poldrack, Big & quic: sparse inverse covariance estimation for a million variables, in: Advances in Neural Information Processing Systems, 2013, pp. 3165–3173.
[18] Prabhanjan Kambadur, Aurelie Lozano, A parallel, block greedy method for sparse inverse covariance estimation for ultra-high dimensions, in: Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, 2013, pp. 351–359.
[19] James P. LeSage, R. Kelley Pace, Spatial dependence in data mining, in: Data Mining for Scientific and Engineering Applications, Springer, 2001, pp. 439–460.
[20] W.E. Leithead, Yunong Zhang, D.J. Leith, Efficient gaussian process based on bfgs updating and logdet approximation, in: The 16th IFAC World Congress, 2005.
[21] R.J. Martin, Approximations to the determinant term in gaussian maximum likelihood estimation of some spatial models, Comm. Statist. Theory Methods 22 (1) (1992) 189–205.
[22] R. Kelley Pace, Ronald Barry, Quick computation of spatial autoregressive estimators, Geogr. Anal. 29 (3) (1997) 232–247.
[23] R. Kelley Pace, Ronald Barry, Otis W. Gilley, C.F. Sirmans, A method for spatial–temporal forecasting with an application to real estate prices, Int. J. Forecast. 16 (2) (2000) 229–246.
[24] R. Kelley Pace, James P. LeSage, Chebyshev approximation of log-determinants of spatial weight matrices, Comput. Statist. Data Anal. 45 (2) (2004) 179–196.
[25] Jack Poulson, Bryan Marker, Robert A. Van de Geijn, Jeff R. Hammond, Nichols A. Romero, Elemental: a new framework for distributed memory dense matrix computations, ACM Trans. Math. Software 39 (2) (2013) 13.
[26] Arnold Reusken, Approximation of the determinant of large sparse symmetric positive definite matrices, SIAM J. Matrix Anal. Appl. 23 (3) (2002) 799–818.
[27] Arvind K. Saibaba, Alen Alexanderian, Ilse C.F. Ipsen, Randomized matrix-free trace and log-determinant estimators, Numer. Math. (2017), http://dx.doi.org/10.1007/s00211-017-0880-z.
[28] John K. Salmon, Mark A. Moraes, Ron O. Dror, David E. Shaw, Parallel random numbers: as easy as 1, 2, 3, in: High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, IEEE, 2011, pp. 1–12.
[29] Daniel A. Spielman, Shang-Hua Teng, Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems, in: Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, ACM, 2004, pp. 81–90.
[30] Luca Trevisan, Graph partitioning and expanders, Handout 7 (2011).
[31] Yunong Zhang, William E. Leithead, Approximate implementation of the logarithm of the matrix determinant in gaussian process regression, J. Stat. Comput. Simul. 77 (4) (2007) 329–348.
[32] Yunong Zhang, W.E. Leithead, D.J. Leith, L. Walshe, Log-det approximation based on uniformly distributed seeds and its application to gaussian process regression, J. Comput. Appl. Math. 220 (1) (2008) 198–214.