EVERYTHING IS VECCHIA: UNIFYING LOW-RANK AND SPARSE INVERSE CHOLESKY APPROXIMATIONS

EAGAN KAMINETZ* AND ROBERT J. WEBBER†

Abstract. The partial pivoted Cholesky approximation accurately represents matrices that are close to being low-rank. Meanwhile, the Vecchia approximation accurately represents matrices with inverse Cholesky factors that are close to being sparse. What happens if a partial Cholesky approximation is combined with a Vecchia approximation of the residual? This paper shows how the sum can be rewritten as a Vecchia approximation of the original matrix with an augmented sparsity pattern. Thus, the Vecchia approximation is a superset of other factored matrix approximations and it has broad applicability. The paper analyzes the optimality properties of the Vecchia approximation, and it tests this approximation on kernel matrices for high-dimensional machine learning applications.

Key words. Vecchia approximation, partial pivoted Cholesky approximation, kernel matrix, factorized sparse approximate inverse

AMS subject classifications. 65F55, 65C99, 15A23

1. Motivation.

The goal of this paper is approximating a large, dense positive-semidefinite matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ by looking up and processing individual entries $\boldsymbol{A}(i,j)$. For example, $\boldsymbol{A}$ might be the kernel matrix for a high-dimensional machine learning data set [24]. In this context, looking up each entry $\boldsymbol{A}(i,j)$ requires a constant, potentially large number of arithmetic operations. Since a kernel matrix can be very large (e.g., $n \geq 10^5$), kernel computations require an approximation $\hat{\boldsymbol{A}} \approx \boldsymbol{A}$ that is generated in *linear* or *sublinear* time. A linear-time algorithm runs in $\mathcal{O}(n^2)$ arithmetic operations, comparable to the cost of looking at each entry of $\boldsymbol{A}$ once. A sublinear-time algorithm runs in $o(n^2)$ operations while looking at just a fraction of $\boldsymbol{A}$'s entries.

Sparse Cholesky approximation is a flexible framework for sublinear- and linear-time matrix approximation. This framework includes the partial pivoted Cholesky approximation [6] and the Vecchia approximation [29], which are traditionally regarded as accurate approximations for different types of matrices. Partial pivoted Cholesky provides an accurate approximation when the target matrix is close to being low-rank [6, Thm. 2.3]. Meanwhile, Vecchia provides an accurate approximation when the inverse Cholesky factor is close to being sparse; see [23, Sec. B.2] and [20]. This paper unifies the two approaches — it shows how the combination of a partial Cholesky approximation and a Vecchia approximation of the residual can be rewritten as a Vecchia approximation with an augmented sparsity pattern.

After positioning the Vecchia approximation as a unifying framework for efficient positive-semidefinite approximations, this paper asks and partially answers, "In what way is the Vecchia approximation optimal?" Optimality theory for the Vecchia approximation was developed in past work [29, 19, 3, 30, 23], but this paper presents a new extension to positive-semidefinite matrices and new error bounds for linear solves and determinant calculations; see section 3 for details. Last, this paper evaluates the Vecchia approximation on kernel matrices for high-dimensional machine learning applications.

*University of California San Diego, La Jolla, CA (ekaminetz@ucsd.edu, rwebber@ucsd.edu).

1.1. Notation. Scalars are written in lower case italics, e.g., m, n, r. Vectors are in lower case boldface, e.g., $\boldsymbol{u}, \boldsymbol{v}$. Matrices are in boldface capital letters, e.g., $\boldsymbol{A}, \boldsymbol{B}$. Index sets are in sans serif font, e.g., R, S. Following standard conventions, $\mathbf{0}$ denotes a vector or matrix of all zeros, $\mathbf{I}$ is an identity matrix, and $\boldsymbol{e}_i$ denotes a standard basis vector, which is all zeros except for 1 in the ith entry.

We use $\boldsymbol{u}(i)$ to refer to to the ith entry of a vector u, and we use $\boldsymbol{A}(i,j)$ to refer to the (i,j) entry of a matrix $\boldsymbol{A}$. Given index sets R, S, we use $\boldsymbol{u}(\mathsf{R})$ to refer to the subvector $(\boldsymbol{u}(i))_{i \in S}$, and we use $\boldsymbol{A}(\mathsf{R}, \mathsf{S})$ to refer to the submatrix $(\boldsymbol{A}(i,j))_{i \in \mathsf{R},\, j \in \mathsf{S}}$. Additionally, $\boldsymbol{A}(i, \cdot)$ and $\boldsymbol{A}(\cdot, i)$ indicate the ith row and column of $\boldsymbol{A}$, and we write $\ell : m$ to refer to $\{\ell, \dots, m\}$ when indexing vectors or matrices.

The complex conjugate of a scalar u is $\overline{u}$. The conjugate transpose of a vector $\boldsymbol{v}$ or matrix $\boldsymbol{A}$ is $\boldsymbol{v}^*$ or $\boldsymbol{A}^*$. The inverse of $\boldsymbol{A}$ is $\boldsymbol{A}^{-1}$, the conjugate transpose inverse is $\boldsymbol{A}^{-*}$, and the Moore-Penrose pseudoinverse is $\boldsymbol{A}^{+}$. Given a positive-semidefinite matrix $\boldsymbol{A} \in \mathbb{C}^{n \times n}$, the $\boldsymbol{A}$-weighted inner product, norm, and distance function are defined as

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_{\boldsymbol{A}} = \boldsymbol{u}^* \boldsymbol{A} \boldsymbol{v}, \quad \|\boldsymbol{v}\|_{\boldsymbol{A}} = \langle \boldsymbol{v}, \boldsymbol{v} \rangle_{\boldsymbol{A}}^{1/2}, \quad \text{and} \quad d_{\boldsymbol{A}}(\boldsymbol{u}, \boldsymbol{v}) = \|\boldsymbol{u} - \boldsymbol{v}\|_{\boldsymbol{A}}.$$

In the rank-deficient case, $\|\cdot\|_{\boldsymbol{A}}$ is not technically a norm. It is a seminorm that defines an equivalence relation $\boldsymbol{u} \sim \boldsymbol{v}$ when $\boldsymbol{u} - \boldsymbol{v} \in \operatorname{null}(\boldsymbol{A})$. Last, in this paper, the volume of a matrix $\boldsymbol{A}$ is the product of its nonzero singular values,

$$\operatorname{vol}(\boldsymbol{A}) = \prod_{\sigma_i(\boldsymbol{A}) > 0} \sigma_i(\boldsymbol{A}),$$

which aligns with the determinant if $\boldsymbol{A}$ is strictly positive definite.

1.2. Organization of paper. The rest of the paper is organized as follows. Section 2 provides background and establishes the main theoretical result. Section 3 derives optimality theory for the Vecchia approximation. Section 4 introduces algorithmic optimization strategies for the Vecchia approximation, and section 5 presents numerical experiments.

2. Background and contributions. This section introduces the framework of sparse Cholesky approximation (subsection 2.1). Then it proves the paper's main theoretical result (subsection 2.2) and discusses broader implications (subsection 2.3).

2.1. Factored approximations based on the Cholesky decomposition. Any positive-semidefinite matrix can be exactly represented through a Cholesky or inverse Cholesky decomposition.

DEFINITION 2.1 (Cholesky and inverse Cholesky decompositions). *The pivoted Cholesky and pivoted inverse Cholesky decompositions for a positive-semidefinite matrix $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ are defined as*

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^*\boldsymbol{P}^* \quad \textit{and} \quad \boldsymbol{A} = \boldsymbol{P}\boldsymbol{C}^{-1}\boldsymbol{D}\boldsymbol{C}^{-*}\boldsymbol{P}^*. \tag{2.1}$$

Here, $\boldsymbol{P} \in \{0,1\}^{n \times n}$ is a permutation matrix, $\boldsymbol{D} \in \mathbb{R}_+^{n \times n}$ is a nonnegative-valued diagonal matrix, and $\boldsymbol{L}, \boldsymbol{C} \in \mathbb{C}^{n \times n}$ are lower triangular matrices with ones on the diagonal.

Cholesky and inverse Cholesky decompositions exist for any positive-semidefinite matrix $\boldsymbol{A}$ and any permutation matrix $\boldsymbol{P}$. See Figure 1 for illustrations. The Cholesky

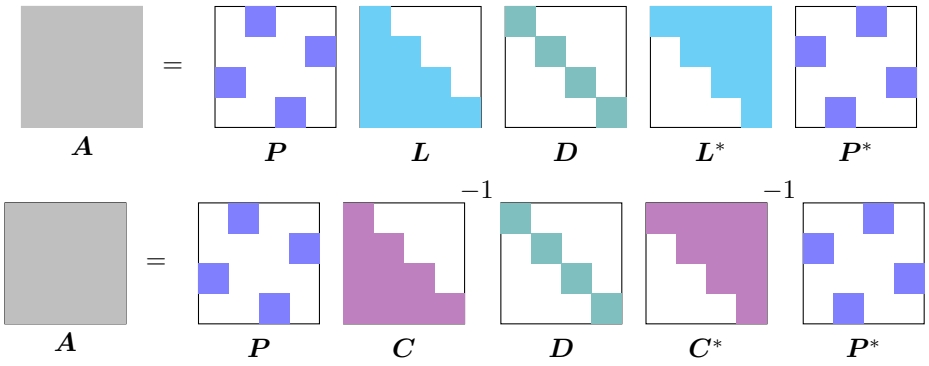

FIG. 1. *Cholesky and inverse Cholesky decompositions of a dense matrix* $\boldsymbol{A}$. *Factors* $\boldsymbol{P}, \boldsymbol{L}, \boldsymbol{D}$ *or* $\boldsymbol{P}, \boldsymbol{C}, \boldsymbol{D}$ *are stored, and inverses* $\boldsymbol{C}^{-1}, \boldsymbol{C}^{-*}$ *are accessed implicitly. Filled boxes show entries that are allowed to be nonzero.*

decomposition can be manipulated into inverse Cholesky form and vice versa through a matrix inversion $\boldsymbol{C} = \boldsymbol{L}^{-1}$.

The Cholesky and inverse Cholesky decompositions facilitate fast matrix computations. Once the factors $\boldsymbol{P}, \boldsymbol{D}, \boldsymbol{L}$ or $\boldsymbol{P}, \boldsymbol{D}, \boldsymbol{C}$ have been generated and stored, the matrix $\boldsymbol{A}$ does not need to be accessed again. Rather, each matrix–vector product $\boldsymbol{v} \mapsto \boldsymbol{A}\boldsymbol{v}$ can be computed by sequentially multiplying the vector with each matrix in the factorization (2.1). The multiplications with $\boldsymbol{C}^{-1}$ or $\boldsymbol{C}^{-*}$ can be carried out via efficient triangular solves, without computing the inverse explicitly. Any consistent linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ can be solved by multiplying the output vector $\boldsymbol{b}$ by a sequence of matrices,

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{L}^{-*}\boldsymbol{D}^{+}\boldsymbol{L}^{-1}\boldsymbol{P}^{*}\boldsymbol{b} \quad \text{or} \quad \boldsymbol{x} = \boldsymbol{P}\boldsymbol{C}^{*}\boldsymbol{D}^{+}\boldsymbol{C}\boldsymbol{P}^{*}\boldsymbol{b}.$$

Thus, the cost of a linear solve is $\mathcal{O}(n^2)$ arithmetic operations.

Motivated by the Cholesky and inverse Cholesky decompositions, we can generate a sparse Cholesky or sparse inverse Cholesky approximation

$$\hat{\boldsymbol{A}} = \boldsymbol{P}\hat{\boldsymbol{L}}\hat{\boldsymbol{D}}\hat{\boldsymbol{L}}^{*}\boldsymbol{P}^{*} \quad \text{or} \quad \hat{\boldsymbol{A}} = \boldsymbol{P}\hat{\boldsymbol{C}}^{-1}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-*}\boldsymbol{P}^{*}. \tag{2.2}$$

Here $\boldsymbol{P}$ is a permutation matrix, $\hat{\boldsymbol{D}}$ is a nonnegative-valued diagonal matrix, and $\hat{\boldsymbol{L}}$ or $\hat{\boldsymbol{C}}$ is a sparse lower triangular matrix with ones on the diagonal. The *sparsity pattern* $\{\mathsf{S}_i\}_{i=1}^{n}$ is a collection of index sets $\mathsf{S}_i \subseteq \{1, \dots, i-1\}$ describing which off-diagonal entries of $\hat{\boldsymbol{L}}$ or $\hat{\boldsymbol{C}}$ are allowed to be nonzero.

The imposition of sparsity leads to three benefits. First, matrix–vector products and linear solves can be computed in $\mathcal{O}(ns)$ arithmetic operations, where s is an upper bound on the cardinality of the sparsity pattern: $|\mathsf{S}_i| \leq s$ for each $i = 1, \dots, n$. Second, the sparse approximation factors can be stored in $\mathcal{O}(sn)$ memory. Third, in many cases generating the approximation $\hat{\boldsymbol{A}}$ is relatively cheap, as it only requires examining $\mathcal{O}(sn)$ or $\mathcal{O}(s^2 n)$ entries of $\boldsymbol{A}$.

Subsections 2.1.1 and 2.1.2 describe two specific examples of sparse Cholesky approximations that can be generated in sublinear or linear time.

2.1.1. Partial pivoted Cholesky. The partial pivoted Cholesky approximation is a common rank-revealing factorization for positive-semidefinite matrices [13]. This approximation is generated from r selected columns of the matrix $\boldsymbol{A}$, where r is a

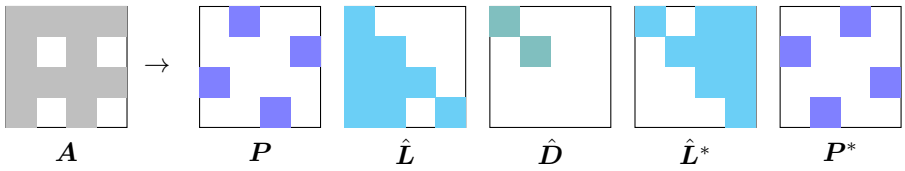

FIG. 2. *Partial pivoted Cholesky accesses the gray-colored entries of* $\boldsymbol{A}$. *The approximation rank is* $r = 2$ *and the columns* $u_1 = 3$ *and* $u_2 = 1$ *are perfectly replicated.*

Algorithm 2.1 Partial pivoted Cholesky approximation

Require: Positive-semidefinite matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ with entry-wise access; permutation matrix $\boldsymbol{P} \in \{0,1\}^{n\times n}$; approximation rank r

Ensure: Sparse Cholesky approximation $\hat{\boldsymbol{A}} = \boldsymbol{P}\hat{\boldsymbol{L}}\hat{\boldsymbol{D}}\hat{\boldsymbol{L}}^*\boldsymbol{P}^*$ in factored form

```
Initialize sparse matrices F̂ = P and D̂ = 0 ∈ ℂ^{n×n}        ▷ F̂ = PL̂
for i = 1, ..., r do
    Identify pivot u_i ∈ {1, ..., n} with P(u_i, i) = 1
    v ← A(·, u_i) − F̂D̂F̂(u_i, ·)*
    if v(u_i) > 0 then
        F̂(·, i) ← v/v(u_i)
        D̂(i, i) ← v(u_i)
    end if
end for
L̂ = P*F̂
```

rank parameter chosen by the user. The approximation exactly replicates the selected columns. See the following definition and see Figure 2 for an illustration.

DEFINITION 2.2 (Partial pivoted Cholesky). *Given a positive-semidefinite matrix* $\boldsymbol{A} \in \mathbb{C}^{n\times n}$, *the partial pivoted Cholesky approximation with permutation* $\boldsymbol{P}$ *and approximation rank* r *is a sparse Cholesky approximation* $\hat{\boldsymbol{A}} = \boldsymbol{P}\hat{\boldsymbol{L}}\hat{\boldsymbol{D}}\hat{\boldsymbol{L}}^*\boldsymbol{P}^*$ *where each row* $\hat{\boldsymbol{L}}(i,\cdot)$ *has sparsity pattern* $\mathsf{S}_i = \{1,\dots,r\} \cap \{1,\dots i-1\}$ *and where*

$$\begin{cases} (\hat{\boldsymbol{L}}\hat{\boldsymbol{D}}\hat{\boldsymbol{L}}^*)(i,j) = (\boldsymbol{P}^*\boldsymbol{A}\boldsymbol{P})(i,j), & \min\{i,j\} \le r, \\ \hat{\boldsymbol{D}}(i,i) = 0, & r+1 \le i \le n. \end{cases}$$

In this definition, the first r rows and columns of $\hat{\boldsymbol{L}}\hat{\boldsymbol{D}}\hat{\boldsymbol{L}}^*$ match the first r rows and columns of the permuted matrix $\boldsymbol{P}^*\boldsymbol{A}\boldsymbol{P}$. The last $n-r$ columns of $\hat{\boldsymbol{L}}$ and $\hat{\boldsymbol{D}}$ can be trivial, containing just zeros and ones.

Algorithm 2.1 generates a partial Cholesky approximation in $\mathcal{O}(r^2 n)$ arithmetic operations. The algorithm processes the columns of $\boldsymbol{A}$ identified by the first r "pivots" $u_1,\dots,u_r$, which are defined by entries $\boldsymbol{P}(u_i,i) = 1$ in the permutation matrix. The $\mathcal{O}(r^2 n)$ cost arises because the ith step forms linear combinations of the first i selected columns.

A long line of research has investigated optimal pivot selection for the partial pivoted Cholesky approximation. "Randomly pivoted Cholesky" is a randomized selection rule that guarantees near-optimal approximation error in the expected trace norm [6], and other common pivot selection rules are based on greedy selection [13] or farthest point sampling [15]. See subsection 4.1 for more analysis and discussion.

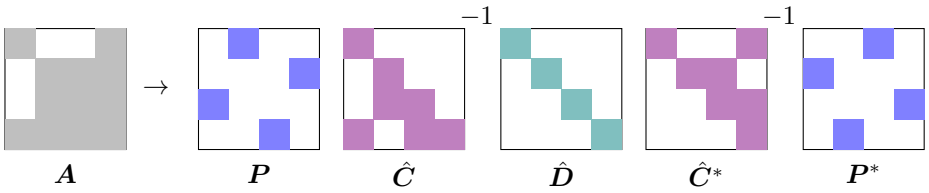

FIG. 3. *Vecchia approximation accesses the gray entries in* $\boldsymbol{A}$. *The pivots are* $u_1 = 3$, $u_2 = 1$, $u_3 = 4$, $u_4 = 2$, *and the sparsity pattern is* $\mathsf{S}_1 = \emptyset$, $\mathsf{S}_2 = \emptyset$, $\mathsf{S}_3 = \{2\}$, $\mathsf{S}_4 = \{1, 3\}$.

Algorithm 2.2 Conventional Vecchia algorithm

Require: Positive-semidefinite matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ with entry-wise access; permutation matrix $\boldsymbol{P} \in \{0,1\}^{n\times n}$; sparsity pattern $\{\mathsf{S}_i\}_{i=1}^n$

Ensure: Vecchia approximation $\hat{\boldsymbol{A}} = \boldsymbol{P}\hat{\boldsymbol{C}}^{-1}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-*}\boldsymbol{P}^*$ in factored form

Initialize sparse matrices $\hat{\boldsymbol{C}} = \mathbf{I} \in \mathbb{C}^{n\times n}$ and $\hat{\boldsymbol{D}} = \mathbf{0} \in \mathbb{C}^{n\times n}$

for $i = 1, \dots, n$ **do** ▷ Can be executed in parallel

$$\begin{bmatrix} \boldsymbol{M} & \boldsymbol{v} \\ \boldsymbol{v}^* & \alpha \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{P}(\cdot, \mathsf{S}_i) & \boldsymbol{P}(\cdot, i) \end{bmatrix}^* \boldsymbol{A} \begin{bmatrix} \boldsymbol{P}(\cdot, \mathsf{S}_i) & \boldsymbol{P}(\cdot, i) \end{bmatrix}$$

Solve positive-semidefinite system $\boldsymbol{M}\boldsymbol{x} = -\boldsymbol{v}$

$\hat{\boldsymbol{C}}(i, \mathsf{S}_i) \leftarrow \boldsymbol{x}^*$

$\hat{\boldsymbol{D}}(i, i) \leftarrow \alpha + \boldsymbol{x}^*\boldsymbol{v}$

end for

2.1.2. Vecchia approximation. The Vecchia approximation [29] is a common approximation of covariance matrices of Gaussian processes. See below for a mathematical definition, and see Figure 3 for an illustration.

DEFINITION 2.3 (Vecchia approximation). *Given a positive-semidefinite matrix* $\boldsymbol{A} \in \mathbb{C}^{n\times n}$, *the Vecchia approximation with permutation* $\boldsymbol{P}$ *and sparsity pattern* $\{\mathsf{S}_i\}_{i=1}^n$ *is a sparse inverse Cholesky approximation* $\hat{\boldsymbol{A}} = \boldsymbol{P}\hat{\boldsymbol{C}}^{-1}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-*}\boldsymbol{P}^*$ *where each row* $\hat{\boldsymbol{C}}(i, :)$ *has sparsity pattern* S_i *and it satisfies*

$$\begin{cases} (\hat{\boldsymbol{C}}\tilde{\boldsymbol{A}})(i, \mathsf{S}_i) = 0, \\ (\hat{\boldsymbol{C}}\tilde{\boldsymbol{A}})(i, i) = \hat{\boldsymbol{D}}(i, i), \end{cases} \quad \text{for } \tilde{\boldsymbol{A}} = \boldsymbol{P}^*\boldsymbol{A}\boldsymbol{P}. \tag{2.3}$$

In this definition, the Vecchia approximation is based on a sequence of n linear solves that determine each row vector $\hat{\boldsymbol{C}}(i, \mathsf{S}_i)$ and associated scalar $\hat{\boldsymbol{D}}(i, i)$. The Vecchia approximation does not necessarily reproduce the the entries of $\boldsymbol{A}$ that were examined during the algorithm, but it guarantees a different type of approximation quality framed in terms of the Kaporin condition number; see section 3.

Algorithm 2.2 presents a conventional Vecchia implementation that evaluates each linear system separately, creating opportunities for parallelism. The total cost of the Vecchia construction is $\mathcal{O}(s^3 n)$ arithmetic operations, where s is an upper bound on the sparsity: $|\mathsf{S}_i| \le s$ for $i = 1, \dots, n$. Yet speedups may be possible by solving the linear systems in a different way — this paper investigates a Vecchia construction that has a smaller cost of $\mathcal{O}(s^2 n)$ arithmetic operations when the sparsity pattern takes a particular structured form; see subsection 2.2.

The design choices in the Vecchia approximation are the permutation matrix $\boldsymbol{P}$ and the sparsity pattern $\{\mathsf{S}_i\}_{i=1}^n$. Historically, researchers chose the sparsity pattern using nearest neighbors in a problem-specific metric [25], and they generated the

permutation matrix by recursively choosing the farthest-away index in the same metric [16]. More recently, Huan et al. [18] proposed optimizing each index set S_i using a greedy heuristic that has been studied in the sparse approximation literature [27].

2.2. Main theoretical result. This paper analyzes a hybrid approximation that combines the partial pivoted Cholesky and Vecchia approximations [31, 5]. We establish the following main result.

THEOREM 2.4 (Partial Cholesky + Vecchia = Vecchia). *Given a target positive-semidefinite matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$, consider the following two-part approximation.*

1. *Generate a partial Cholesky approximation of $\boldsymbol{A}$ with permutation $\boldsymbol{P}$ and approximation rank r. Call it $\hat{\boldsymbol{A}}_{\text{part}}$.*
2. *Generate a Vecchia approximation of the residual $\boldsymbol{R} = \boldsymbol{A} - \hat{\boldsymbol{A}}_{\text{part}}$ with permutation $\boldsymbol{P}$ and sparsity pattern $\{\mathsf{Q}_i\}_{i=1}^n$. Call it $\hat{\boldsymbol{A}}_{\text{res}}$.*

Then $\hat{\boldsymbol{A}}_{\text{part}} + \hat{\boldsymbol{A}}_{\text{res}}$ can be rewritten as a Vecchia approximation of $\boldsymbol{A}$ with permutation $\boldsymbol{P}$ and an augmented sparsity pattern $\mathsf{S}_i = \big(\{1,\dots,r\} \cup \mathsf{Q}_i\big) \cap \{1,\dots,i-1\}$.

Proof. Introduce the permuted matrices $\tilde{\boldsymbol{A}} = \boldsymbol{P}^*\boldsymbol{A}\boldsymbol{P}$, $\tilde{\boldsymbol{R}} = \boldsymbol{P}^*\boldsymbol{R}\boldsymbol{P}$,

$$\tilde{\boldsymbol{A}}_{\text{part}} = \boldsymbol{P}^*\hat{\boldsymbol{A}}_{\text{part}}\boldsymbol{P} = \begin{bmatrix} \hat{\boldsymbol{L}}_{11} & \\ \hat{\boldsymbol{L}}_{21} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{D}}_{11} & \\ & \mathbf{0} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{L}}_{11} & \\ \hat{\boldsymbol{L}}_{21} & \mathbf{I} \end{bmatrix}^*, \text{ and}$$

$$\tilde{\boldsymbol{A}}_{\text{res}} = \boldsymbol{P}^*\hat{\boldsymbol{A}}_{\text{res}}\boldsymbol{P} = \begin{bmatrix} \mathbf{I} & \\ \mathbf{0} & \hat{\boldsymbol{C}}_{22} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0} & \\ & \hat{\boldsymbol{D}}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \\ \mathbf{0} & \hat{\boldsymbol{C}}_{22} \end{bmatrix}^{-*}$$

where $\tilde{\boldsymbol{A}}_{\text{part}}$ and $\tilde{\boldsymbol{A}}_{\text{res}}$ are partitioned into the first r and last $n-r$ entries. We can add the approximations to produce a sparse inverse Cholesky approximation

$$\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}_{\text{part}} + \hat{\boldsymbol{A}}_{\text{res}} = \boldsymbol{P}\hat{\boldsymbol{C}}^{-1}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-*}\boldsymbol{P}^*,$$

where

$$\hat{\boldsymbol{C}} = \begin{bmatrix} \hat{\boldsymbol{L}}_{11}^{-1} & \\ -\hat{\boldsymbol{C}}_{22}\hat{\boldsymbol{L}}_{21}\hat{\boldsymbol{L}}_{11}^{-1} & \hat{\boldsymbol{C}}_{22} \end{bmatrix} \quad \text{and} \quad \hat{\boldsymbol{D}} = \begin{bmatrix} \hat{\boldsymbol{D}}_{11} & \\ & \hat{\boldsymbol{D}}_{22} \end{bmatrix}.$$

By inspection, $\hat{\boldsymbol{A}}$ is a sparse Cholesky approximation with permutation $\boldsymbol{P}$ and sparsity pattern $\{\mathsf{S}_i\}_{i=1}^n$. See Figure 4 for an illustration. To show it is a Vecchia approximation, we need to verify the following equalities for $i = 1,\dots,n$:

$$(\hat{\boldsymbol{C}}\tilde{\boldsymbol{A}})(i,\mathsf{Q}_i) = 0 \quad \text{and} \quad (\hat{\boldsymbol{C}}\tilde{\boldsymbol{A}})(i,i) = \hat{\boldsymbol{D}}(i,i). \tag{2.4}$$

We will do this separately for $i \le r$ and $i \ge r+1$.

First consider $i \le r$. Since $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{A}}_{\text{part}}$ share the same first r rows, we can write

$$\begin{aligned} (\hat{\boldsymbol{C}}\tilde{\boldsymbol{A}})(i,\cdot) &= (\hat{\boldsymbol{C}}\tilde{\boldsymbol{A}}_{\text{part}})(i,\cdot) \\ &= (\hat{\boldsymbol{L}}_{11}^{-1}\hat{\boldsymbol{L}}_{11}\hat{\boldsymbol{D}}_{11})(i,\cdot)\begin{bmatrix}\hat{\boldsymbol{L}}_{11}^* & \hat{\boldsymbol{L}}_{21}^*\end{bmatrix} \\ &= \hat{\boldsymbol{D}}_{11}(i,i)\begin{bmatrix}\hat{\boldsymbol{L}}_{11}(\cdot,i)^* & \hat{\boldsymbol{L}}_{21}(\cdot,i)^*\end{bmatrix}. \end{aligned}$$

Since $\hat{\boldsymbol{L}}_{11}$ is lower triangular with ones on the diagonal, conclude that $(\hat{\boldsymbol{C}}\tilde{\boldsymbol{A}})(i,\cdot)$ is a row vector with the first $i-1$ entries equal to zero and the ith entry equal to $\hat{\boldsymbol{D}}(i,i) = \hat{\boldsymbol{D}}_{11}(i,i)$, confirming (2.4).

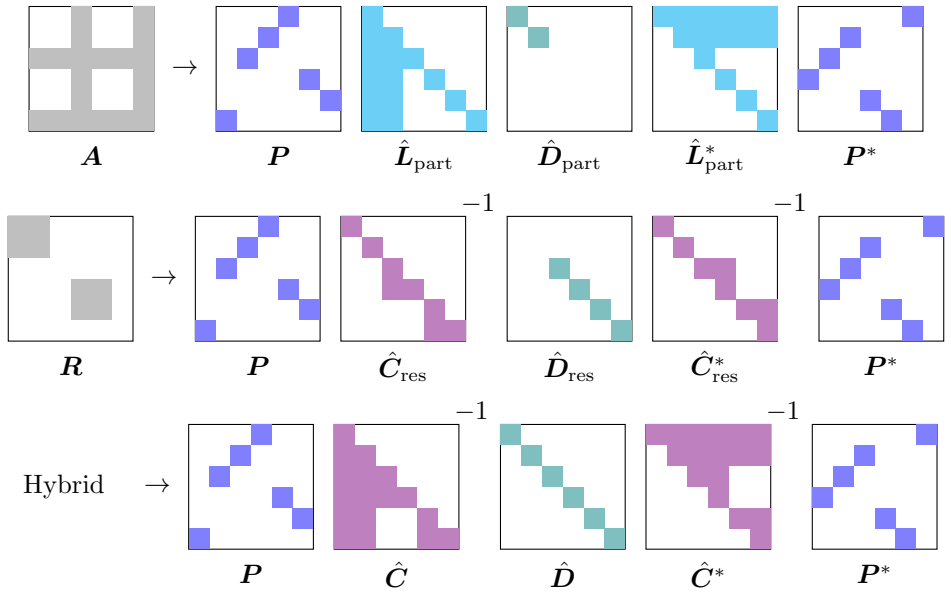

FIG. 4. *First row: partial Cholesky accesses the gray entries of* $\boldsymbol{A}$ *to generate an approximation* $\hat{\boldsymbol{A}}_{\text{part}} = \boldsymbol{P}\hat{\boldsymbol{L}}_{\text{part}}\hat{\boldsymbol{D}}_{\text{part}}\hat{\boldsymbol{L}}^*_{\text{part}}\boldsymbol{P}^*$. *Second row: Vecchia accesses the gray entries of* $\boldsymbol{R} = \boldsymbol{A} - \hat{\boldsymbol{A}}_{\text{part}}$ *to generate an approximation* $\hat{\boldsymbol{A}}_{\text{res}} = \boldsymbol{P}\hat{\boldsymbol{C}}_{\text{res}}^{-1}\hat{\boldsymbol{D}}_{\text{res}}\hat{\boldsymbol{C}}_{\text{res}}^{-*}\boldsymbol{P}^*$. *Third row: partial Cholesky + Vecchia yields an improved approximation* $\hat{\boldsymbol{A}} = \boldsymbol{P}\hat{\boldsymbol{C}}^{-1}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-*}\boldsymbol{P}^*$.

Next consider $i \geq r+1$. We use the fact that $\tilde{\boldsymbol{R}}$ is all zeros except for the bottom right $(n-r)\times(n-r)$ submatrix, which we call $\tilde{\boldsymbol{R}}_{22}$. Therefore,

$$\begin{aligned}
(\hat{\boldsymbol{C}}\tilde{\boldsymbol{A}})(i,\cdot) &= \hat{\boldsymbol{C}}(i,\cdot)(\tilde{\boldsymbol{A}}_{\text{part}} + \tilde{\boldsymbol{R}}) \\
&= \hat{\boldsymbol{C}}_{22}(i-r,\cdot)\begin{bmatrix}-\hat{\boldsymbol{L}}_{21}\hat{\boldsymbol{L}}_{11}^{-1} & \mathbf{I}\end{bmatrix}\left(\begin{bmatrix}\hat{\boldsymbol{L}}_{11}\\ \hat{\boldsymbol{L}}_{21}\end{bmatrix}\hat{\boldsymbol{D}}_{11}\begin{bmatrix}\hat{\boldsymbol{L}}^*_{11} & \hat{\boldsymbol{L}}^*_{21}\end{bmatrix} + \begin{bmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \tilde{\boldsymbol{R}}_{22}\end{bmatrix}\right) \\
&= \hat{\boldsymbol{C}}_{22}(i-r,\cdot)\begin{bmatrix}\mathbf{0} & \tilde{\boldsymbol{R}}_{22}\end{bmatrix} \\
&= \begin{bmatrix}\mathbf{0} & \hat{\boldsymbol{C}}_{22}\end{bmatrix}(i,\cdot)\tilde{\boldsymbol{R}}.
\end{aligned}$$

Since $\begin{bmatrix}\mathbf{0} & \hat{\boldsymbol{C}}_{22}\end{bmatrix}(i,\cdot)$ is a row vector generated by a Vecchia approximation of $\boldsymbol{R}$, we confirm the stated equalities in (2.4) and complete the proof. ∎

2.3. Implications. Theorem 2.4 demonstrates that the partial Cholesky + Vecchia approach which appeared in the previous papers [31, 5] is secretly constructing a Vecchia approximation with the first r indices included in the sparsity pattern. We list two significant implications.

First, the partial Cholesky + Vecchia approach is *computationally efficient*. When $q \geq 1$ is an upper bound on the sparsity level in the residual component: $|\mathsf{Q}_i| \leq q$ for $i = 1,\ldots,n$, the construction cost is $\mathcal{O}((r^2 + rq^2 + q^3)n)$ arithmetic operations, If $q = \mathcal{O}(r^{1/2})$, the construction cost is $\mathcal{O}((r+q)^2 n)$ operations, which is smaller than the conventional Vecchia construction cost of $\mathcal{O}((r+q)^3 n)$ operations using Algorithm 2.2.

Second, the partial Cholesky + Vecchia approach can be very *accurate*. In the simplest case, we set the residual component to have sparsity $\mathsf{Q}_i = \emptyset$ for $i = 1,\ldots,n$,

leading to a "partial Cholesky + diagonal" approximation:

$$\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}_{\text{part}} + \hat{\boldsymbol{A}}_{\text{res}} = \hat{\boldsymbol{A}}_{\text{part}} + \text{diag}(\boldsymbol{A} - \hat{\boldsymbol{A}}_{\text{part}}).$$

In contrast, a previous algorithm for refining a partial Cholesky approximation added a multiple of the identity to the nullspace of $\hat{\boldsymbol{A}}_{\text{part}}$ [11]. The numerical experiments in section 5 show that partial Cholesky + diagonal leads to 5× faster conjugate gradient convergence for a poorly conditioned target matrix.

Going beyond the minimal sparsity pattern, we also consider a small, carefully chosen sparsity pattern $\{\mathsf{Q}_i\}_{i=1}^n$ for the residual component. Specifically, we investigate a two-part algorithm called "randomly pivoted Cholesky + Vecchia" (RPC+V).

1. Apply randomly pivoted Cholesky [6] to generate a rank-r approximation of a $n \times n$ positive-semidefinite matrix.
2. Generate a Vecchia approximation of the residual matrix, where each component of the sparsity pattern comes from a greedy optimization [18] with c candidate points and s selected nonzero entries.

To ensure a linear construction cost, we use the parameter settings $r = \mathcal{O}(n^{1/2})$, $c = \mathcal{O}(n^{1/4})$, and $s = \mathcal{O}(n^{1/4})$. Numerical experiments in section 5 show that RPC+V leads to 1.5× faster conjugate gradient convergence and 10× more accurate log determinant estimation that randomly pivoted Cholesky + diagonal.

In summary, the paper's main contribution is a recognition and theoretical justification that the Vecchia approximation is a superset of other factored matrix approximations. This perspective leads to higher accuracy Vecchia approximations that are also generated more quickly.

3. Kaporin optimality theory and applications. In this section, we ask and partially answer, "In what way is the Vecchia approximation optimal?"

Vecchia optimality theory was previously developed in the papers [29, 19, 3, 30, 23]. Here, we review this optimality theory and push it in new directions. Our analysis is based on the following Kaporin condition number [19, 3], which measures the accuracy of a matrix approximation.

DEFINITION 3.1 (Kaporin condition number). *For any positive-semidefinite matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ and positive-semidefinite approximation $\hat{\boldsymbol{A}} \in \mathbb{C}^{n\times n}$, the Kaporin condition number is*

$$\kappa_{\text{Kap}} = \frac{\left(\frac{1}{r}\operatorname{tr}(\boldsymbol{A}\hat{\boldsymbol{A}}^+)\right)^r}{\operatorname{vol}(\boldsymbol{A}\hat{\boldsymbol{A}}^+)}, \qquad \textit{where } r = \operatorname{rank}(\boldsymbol{A}),$$

if $\boldsymbol{A}$ and $\hat{\boldsymbol{A}}$ share the same range. The Kaporin condition number is $\kappa_{\text{Kap}} = \infty$ if $\boldsymbol{A}$ and $\hat{\boldsymbol{A}}$ have different ranges.

The Kaporin condition number is the average positive eigenvalue of $\boldsymbol{A}\hat{\boldsymbol{A}}^+$ raised to the $\operatorname{rank}(\boldsymbol{A})$ power, divided by the product of the positive eigenvalues of $\boldsymbol{A}\hat{\boldsymbol{A}}^+$. By the arithmetic-geometric mean inequality, the Kaporin condition number is bounded from below by one. Equality holds if and only if all the positive eigenvalues are equal.

Kaporin [19, App. A.3] proved that the Vecchia approximation optimizes κ_{Kap} for strictly positive-definite $\boldsymbol{A}$ subject to any given sparsity pattern. Here we extend this result to any positive-semidefinite $\boldsymbol{A}$. The proof is presented in Appendix A.

THEOREM 3.2 (Optimality of Vecchia). *For any positive-semidefinite matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$, the Vecchia approximation $\hat{\boldsymbol{A}} = \boldsymbol{P}\hat{\boldsymbol{C}}^{-1}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-*}\boldsymbol{P}^*$ is the inverse Cholesky*

approximation with permutation $\boldsymbol{P}$ *and sparsity pattern* $\{\mathsf{S}_i\}_{i=1}^n$ *that achieves the smallest possible Kaporin condition number. When* $\hat{\boldsymbol{A}}$ *and* $\boldsymbol{A}$ *have the same range, the Kaporin condition number is*

$$\kappa_{\text{Kap}} = \prod_{d_{\tilde{\boldsymbol{A}}}(\boldsymbol{e}_i, \text{span}\{\boldsymbol{e}_j\}_{j<i})>0} \frac{d_{\tilde{\boldsymbol{A}}}\big(\boldsymbol{e}_i, \text{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{S}_i}\big)^2}{d_{\tilde{\boldsymbol{A}}}\big(\boldsymbol{e}_i, \text{span}\{\boldsymbol{e}_j\}_{j<i}\big)^2}, \tag{3.1}$$

where $\tilde{\boldsymbol{A}} = \boldsymbol{P}^*\boldsymbol{A}\boldsymbol{P}$ *is the permuted* $\boldsymbol{A}$ *matrix.*

Several points are worth mentioning. First, the range of the Vecchia approximation always contains the range of the target matrix, $\text{range}(\boldsymbol{A}) \subseteq \text{range}(\hat{\boldsymbol{A}})$. The ranges are the same if and only if the sparsity set is sufficiently large. Specifically, when there are "bad" indices i for which

$$d_{\tilde{\boldsymbol{A}}}(\boldsymbol{e}_i, \text{span}\{\boldsymbol{e}_j\}_{j<i}) = 0,$$

the sparsity component S_i must be large enough that

$$d_{\tilde{\boldsymbol{A}}}(\boldsymbol{e}_i, \text{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{S}_i}) = 0.$$

Otherwise, the Kaporin condition number is infinite and there is no sparse Cholesky approximation with the same range as $\boldsymbol{A}$.

Second, there is a way to rewrite the Kaporin condition number (3.1) using an inverse Cholesky decomposition $\boldsymbol{A} = \boldsymbol{P}\boldsymbol{C}^{-1}\boldsymbol{D}\boldsymbol{C}^{-*}\boldsymbol{P}^*$.

$$\kappa_{\text{Kap}} = \frac{\text{vol}(\hat{\boldsymbol{A}})}{\text{vol}(\boldsymbol{A})} = \frac{\text{vol}(\hat{\boldsymbol{D}})}{\text{vol}(\boldsymbol{D})} = \prod_{\boldsymbol{D}(i,i)>0} \frac{\hat{\boldsymbol{D}}(i,i)}{\boldsymbol{D}(i,i)}.$$

The entries $\hat{\boldsymbol{D}}(i,i)$ and $\boldsymbol{D}(i,i)$ are encoding the $\tilde{\boldsymbol{A}}$-weighted distances.

$$\begin{cases} \hat{\boldsymbol{D}}(i,i) = d_{\tilde{\boldsymbol{A}}}(\boldsymbol{e}_i, \text{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{S}_i}), \\ \boldsymbol{D}(i,i) = d_{\tilde{\boldsymbol{A}}}(\boldsymbol{e}_i, \text{span}\{\boldsymbol{e}_j\}_{j<i}). \end{cases}$$

However, the expressions using $\tilde{\boldsymbol{A}}$-weighted distances are more intuitive, and they motivate the Vecchia optimization strategies that will be pursued in section 4.

Third, the Kaporin condition number optimality is useful and important, because it controls the error of several linear algebra calculations, especially those involving determinants. For a list of Kaporin condition number bounds, see Table 1. The rest of the section will introduce these bounds in more detail and explain how to use a factored matrix approximation like a Vecchia approximation to speed up linear algebra.

3.1. Direct linear system solves. In order to solve a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ or a more general linear least-squares problem $\min_{\boldsymbol{x}}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2$, we can make an initial guess $\boldsymbol{x}_0$ and then refine our guess by calculating

$$\hat{\boldsymbol{x}} = \boldsymbol{x}_0 + \hat{\boldsymbol{A}}^+[\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0].$$

This calculation is very fast. We do not even need to examine the entries of $\boldsymbol{A}$ once; we only need to perform a linear solve with a factored approximation $\hat{\boldsymbol{A}}$. The following proposition bounds the error of the resulting direct solve. To our knowledge, the proof is new, and it appears in Appendix B.1.

Method	Error bound	Reference
Linear system, direct solver	$\dfrac{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_\star\|_{\boldsymbol{A}}^2}{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_{\boldsymbol{A}}^2} \le 2\operatorname{rank}(\boldsymbol{A})\log(\kappa_{\mathrm{Kap}})$	Proposition 3.3
Linear system, iterative solver	$\dfrac{\|\boldsymbol{x}_t - \boldsymbol{x}_\star\|_{\boldsymbol{A}}^2}{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_{\boldsymbol{A}}^2} \le \left[\dfrac{3\log(\kappa_{\mathrm{Kap}})}{t}\right]^t$	[3], Proposition 3.4
Determinant, direct solver	$\log\left(\dfrac{\det \hat{\boldsymbol{A}}}{\det \boldsymbol{A}}\right) = \log(\kappa_{\mathrm{Kap}})$	Proposition 3.5
Determinant, iterative solver	$\mathbb{E}\left\|\log\left(\dfrac{\mathrm{e}^{s_t}\det \hat{\boldsymbol{A}}}{\det \boldsymbol{A}}\right)\right\|^2 \le \dfrac{4\log(\kappa_{\mathrm{Kap}})}{t}$	Proposition 3.6

TABLE 1

Summary of error bounds for direct and iterative solvers in terms of the Kaporin condition number κ_{Kap}. The direct solver bounds require the normalization $\operatorname{tr}\big(\boldsymbol{A}\hat{\boldsymbol{A}}^+\big) = \operatorname{rank}(\boldsymbol{A})$ which is automatically satisfied for the Vecchia approximation when $\kappa_{\mathrm{Kap}} < \infty$. The determinant bounds require that $\boldsymbol{A}$ and $\hat{\boldsymbol{A}}$ are strictly positive definite.

PROPOSITION 3.3 (Approximate direct solver for linear systems). *If $\hat{\boldsymbol{A}}$ is normalized so that $\operatorname{tr}\big(\boldsymbol{A}\hat{\boldsymbol{A}}^+\big) = \operatorname{rank}(\boldsymbol{A})$, the approximate direct solver satisfies*

$$\frac{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_\star\|_{\boldsymbol{A}}^2}{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_{\boldsymbol{A}}^2} \le 2\operatorname{rank}(\boldsymbol{A})\log(\kappa_{\mathrm{Kap}}).$$

Here, $\boldsymbol{x}_\star = \boldsymbol{A}^+\boldsymbol{b}$ is the minimum–norm solution to $\min_{\boldsymbol{x}}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2$.

The Vecchia approximation uses the normalization $\operatorname{tr}\big(\boldsymbol{A}\hat{\boldsymbol{A}}^+\big) = \operatorname{rank}(\boldsymbol{A})$, so the guarantees in Proposition 3.3 are valid. Yet the guarantees in Proposition 3.3 are not very strong unless the Kaporin condition number is extremely small, $\log(\kappa_{\mathrm{Kap}}) < 1/\operatorname{rank}(\boldsymbol{A})$.

3.2. Iterative linear system solves. If the approximate direct solver fails to deliver an accurate solution, we can instead use the more expensive preconditioned conjugate gradient algorithm [14, Sec. 11.5] Starting with an initial guess $\boldsymbol{x}_0$, PCG calculates the residual $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$ and the search direction $\boldsymbol{d}_1 = \hat{\boldsymbol{A}}^+\boldsymbol{r}_0$. At each step $t = 1, 2, \ldots$, PCG updates the iterate and residual according to

$$\begin{cases} \boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \alpha_t \boldsymbol{d}_t, \\ \boldsymbol{r}_t = \boldsymbol{r}_{t-1} - \alpha_t \boldsymbol{A}\boldsymbol{d}_t. \end{cases}$$

The step size and search direction are updated according to

$$\alpha_t = \frac{\boldsymbol{r}_{t-1}^*\hat{\boldsymbol{A}}^+\boldsymbol{r}_{t-1}}{\boldsymbol{d}_i^*\boldsymbol{A}\boldsymbol{d}_i} \quad \text{and} \quad \boldsymbol{d}_{t+1} = \hat{\boldsymbol{A}}^+\boldsymbol{r}_t + \frac{\boldsymbol{r}_t^*\hat{\boldsymbol{A}}^+\boldsymbol{r}_t}{\boldsymbol{r}_{t-1}^*\hat{\boldsymbol{A}}^+\boldsymbol{r}_{t-1}}\boldsymbol{d}_t.$$

We can run PCG for any number of iterations, producing better and better estimates of $\boldsymbol{x}_\star = \boldsymbol{A}^+\boldsymbol{b}$ with each iteration. PCG requires just one multiplication with $\boldsymbol{A}$ and one linear solve with $\hat{\boldsymbol{A}}$ per iteration.

Axelsson and Kaporin [3, Thm. 4.3] bounded the error of PCG in terms of κ_{Kap} assuming an even number of iterations $t \ge 0$, and Appendix B.2 extends their proof to handle any even or odd number of iterations $t \ge 0$.

PROPOSITION 3.4 (Convergence of PCG [3]). *At each iteration $t \geq 0$, the iterates produced by preconditioned conjugate gradient satisfy*

$$\frac{\|\boldsymbol{x}_t - \boldsymbol{x}_\star\|_{\boldsymbol{A}}^2}{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_{\boldsymbol{A}}^2} \leq \left[\frac{3\log(\kappa_{\mathrm{Kap}})}{t}\right]^t. \tag{3.2}$$

Here, $\boldsymbol{x}_\star = \boldsymbol{A}^+\boldsymbol{b}$ is the minimum–norm solution to $\min_{\boldsymbol{x}}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2$.

Proposition 3.4 guarantees *superlinear* convergence, which is even stronger than the standard conjugate gradient linear convergence bounds (e.g., [14, Eq. 11.3.27]). Yet in large-scale applications, the superlinear convergence is mainly observed in the latter iterations when PCG has already achieved high accuracy [3]. As such, Proposition 3.4 provides only partial evidence that optimizing the Kaporin condition number is a good idea for solving linear systems.

3.3. Direct determinant calculation. When $\boldsymbol{A}$ and $\hat{\boldsymbol{A}}$ are strictly positive definite, we can use $\det(\hat{\boldsymbol{A}})$ as an estimator for $\det(\boldsymbol{A})$. This computation is very fast for the Vecchia approximation $\hat{\boldsymbol{A}} = \boldsymbol{P}\hat{\boldsymbol{C}}^{-1}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-*}\boldsymbol{P}^*$, because

$$\det(\hat{\boldsymbol{A}}) = \det(\hat{\boldsymbol{D}}) = \prod_{i=1}^{n} \hat{\boldsymbol{D}}(i,i).$$

The next proposition exactly describes the error in the determinant estimation.

PROPOSITION 3.5 (Approximate direct solver for determinants). *If $\boldsymbol{A}$ and $\hat{\boldsymbol{A}}$ are strictly positive definite and $\hat{\boldsymbol{A}}$ is normalized so that $\operatorname{tr}\big(\boldsymbol{A}\hat{\boldsymbol{A}}^{-1}\big) = n$, then*

$$\log\left(\frac{\det \hat{\boldsymbol{A}}}{\det \boldsymbol{A}}\right) = \log(\kappa_{\mathrm{Kap}}).$$

Proof. We rewrite the Kaporin condition number as

$$\kappa_{\mathrm{Kap}} = \frac{\big(\frac{1}{n}\operatorname{tr}(\boldsymbol{A}\hat{\boldsymbol{A}}^{-1})\big)^n}{\det(\boldsymbol{A}\hat{\boldsymbol{A}}^{-1})} = \frac{1}{\det(\boldsymbol{A}\hat{\boldsymbol{A}}^{-1})} = \frac{\det(\hat{\boldsymbol{A}})}{\det(\boldsymbol{A})},$$

using the trace normalization. ∎

The Vecchia approximation uses the normalization $\operatorname{tr}\big(\boldsymbol{A}\hat{\boldsymbol{A}}^{+}\big) = n$, so Proposition 3.5 shows that the Vecchia determinant upper bounds the true determinant of $\boldsymbol{A}$. Note, however, the Vecchia approximation does not always provide the sharpest upper bound on the determinant of $\boldsymbol{A}$ given the revealed entries: this is given by the solution to the *maximum entropy* problem; see [7, pg. 161].

3.4. Iterative determinant calculation. When the approximation $\det \boldsymbol{A} \approx \det \hat{\boldsymbol{A}}$ is not sufficiently accurate, we can refine the approximation with a multiplicative correction term. To that end, we rewrite

$$\log\left(\frac{\det \boldsymbol{A}}{\det \hat{\boldsymbol{A}}}\right) = \log\big(\det\big(\hat{\boldsymbol{A}}^{-1/2}\boldsymbol{A}\hat{\boldsymbol{A}}^{-1/2}\big)\big) = \operatorname{tr}\big(\log\big(\hat{\boldsymbol{A}}^{-1/2}\boldsymbol{A}\hat{\boldsymbol{A}}^{-1/2}\big)\big). \tag{3.3}$$

Here, the matrix logarithm is defined by taking the logarithm of a matrix's eigenvalues. To approximate the error term (3.3), we sample independent length-$\sqrt{n}$ vectors with uniformly random directions, $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m$, and form a *stochastic trace estimator* [28]:

$$s_t = \frac{1}{t}\sum_{i=1}^{t} \boldsymbol{z}_i^* \log\big(\hat{\boldsymbol{A}}^{-1/2}\boldsymbol{A}\hat{\boldsymbol{A}}^{-1/2}\big)\boldsymbol{z}_i.$$

This estimator is unbiased and converges as $t \to \infty$.

The next proposition bounds the mean square error of the stochastic determinant estimate. The proof is in Appendix B.3 and is apparently new.

PROPOSITION 3.6 (Stochastic determinant estimation). *If $\boldsymbol{A}$ and $\hat{\boldsymbol{A}}$ are strictly positive definite $n \times n$ matrices with $\log(\kappa_{\text{Kap}}) \le n$ and the matrix logarithm is computed exactly, the iterates produced by stochastic determinant estimation satisfy*

$$\mathbb{E}\left|\log\left(\frac{\mathrm{e}^{s_t} \det \hat{\boldsymbol{A}}}{\det \boldsymbol{A}}\right)\right|^2 \le \frac{4\log(\kappa_{\text{Kap}})}{t}. \tag{3.4}$$

Proposition 3.6 shows that the stochastic determinant estimator achieves a better mean square error than the direct estimator as soon as the number of iterations is $t \ge 4/\log(\kappa_{\text{Kap}})$. As the one limitation, we cannot really compute the matrix logarithm exactly. In practice, we approximate each term $\log\big(\hat{\boldsymbol{A}}^{-1/2}\boldsymbol{A}\hat{\boldsymbol{A}}^{-1/2}\big)\boldsymbol{z}_i$ by building a Krylov subspace

$$K_q(\boldsymbol{M}, \boldsymbol{z}_i) = \operatorname{span}\{\boldsymbol{z}_i, \boldsymbol{M}\boldsymbol{z}_i, \dots, \boldsymbol{M}^q\boldsymbol{z}_i\} \quad \text{for} \quad \boldsymbol{M} = \hat{\boldsymbol{A}}^{-1/2}\boldsymbol{A}\hat{\boldsymbol{A}}^{-1/2}$$

and applying an explicit polynomial approximation [26] or an implicit Krylov–Ritz approximation [12] of the matrix logarithm. Each Krylov subspace calculation requires q matrix multiplications, and we use $q = 100$ in the section 5 experiments.

4. Optimization strategies. This section explores the following open question.

Question 4.1 (Optimal Vecchia approximation). What permutation $\boldsymbol{P}$ and sparsity pattern $\{\mathsf{S}_i\}_{i=1}^n$ with bounded cardinality $|\mathsf{S}_i| \le s$ lead to a Vecchia approximation with the lowest possible Kaporin condition number (3.1)?

This question is NP-hard to resolve exactly. Nonetheless, we can suggest several approximation schemes.

4.1. Partial Cholesky + diagonal. First we consider the simpler problem of generating an optimal partial Cholesky + diagonal approximation (subsection 2.3). When the pivot set is $\mathsf{R} = \{u_1, \dots, u_r\}$, the condition number can be rewritten as

$$\kappa_{\text{Kap}} = \prod_{d_{\boldsymbol{A}}(\boldsymbol{e}_i, \operatorname{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{R}})>0} \frac{d_{\boldsymbol{A}}\big(\boldsymbol{e}_i, \operatorname{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big)^2}{d_{\boldsymbol{A}}\big(\boldsymbol{e}_i, \operatorname{span}\big(\{\boldsymbol{e}_j\}_{j<i} \cup \{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big)^2}. \tag{4.1}$$

The Kaporin condition number (4.1) only depends on the pivot set R, so we write $\kappa_{\text{Kap}} = \kappa_{\text{Kap}}(\mathsf{R})$. Below we introduce several strategies to find the best pivot set. Only one of these strategies, called "farthest point sampling", has been considered in the recent literature on Vecchia approximation [23, 31, 5].

4.1.1. Adaptive search. The *adaptive search* algorithm starts with an empty set $\mathsf{R} = \emptyset$ and then adds one pivot at a time by a rule that changes ("adapts") based on the previous selections. At each stage, this algorithm tries all the pivots and chooses the pivot causing the greatest decrease in the condition number.

$$\mathsf{R} \leftarrow \mathsf{R} \cup \{i\}, \quad \text{where } i \in \operatorname*{argmin}_{1\le j\le n} \kappa_{\text{Kap}}(\mathsf{R} \cup \{j\}) \qquad \text{[adaptive search]}.$$

The adaptive search is expensive, since each stage requires processing *all* the entries in the matrix. Thus, the cost of producing a cardinality-r pivot set is $\mathcal{O}(n^2 r)$ arithmetic operations.

4.1.2. Adaptive sampling. *Adaptive sampling* is a class of cheaper algorithms that sample new pivots from a distribution that prioritizes distance to the already chosen pivots. Here we describe four variations of adaptive sampling.

- In "randomly pivoted Cholesky" (RPC, [6]), each pivot is sampled as follows.

$$\mathsf{R} \leftarrow \mathsf{R} \cup \{i\}, \quad \text{with prob}(i) = \frac{d_{\boldsymbol{A}}\big(\boldsymbol{e}_i, \operatorname{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big)^2}{\sum_{j=1}^n d_{\boldsymbol{A}}\big(\boldsymbol{e}_j, \operatorname{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big)^2}. \tag{RPC}$$

- In "square distance sampling" (SDS, [1]), we select a slightly different set of sampling probabilities.

$$\mathsf{R} \leftarrow \mathsf{R} \cup \{i\}, \quad \text{with prob}(i) = \frac{d_{\boldsymbol{A}}\big(\boldsymbol{e}_i, \{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big)^2}{\sum_{j=1}^n d_{\boldsymbol{A}}\big(\boldsymbol{e}_j, \{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big)^2}. \tag{SDS}$$

- In "column pivoted Cholesky" (CPC, [17]), we use a simpler rule for inducting new pivots.

$$\mathsf{R} \leftarrow \mathsf{R} \cup \{i\}, \quad \text{where } i \in \operatorname*{argmin}_{1\le j\le n} d_{\boldsymbol{A}}\big(\boldsymbol{e}_j, \operatorname{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big). \tag{CPC}$$

- Last, in "farthest point sampling" (FPS, [15]), we apply the rule

$$\mathsf{R} \leftarrow \mathsf{R} \cup \{i\}, \quad \text{where } i \in \operatorname*{argmin}_{1\le j\le n} d_{\boldsymbol{A}}\big(\boldsymbol{e}_j, \{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big). \tag{FPS}$$

If there are ties in CPC or FPS, they can be broken arbitrarily. Each of the adaptive sampling algorithms is cheap compared to the overall cost of matrix approximation. In the context of partial Cholesky + diagonal, the cost of the partial Cholesky step is already $\mathcal{O}(nr^2)$ arithmetic operations, and adding an adaptive sampling rule requires an extra $\mathcal{O}(nr)$ operations.

Adaptive sampling theory shows how each algorithm approximately minimizes one of the following distance functionals.

$$\begin{aligned}
\eta_{\text{RPC}}(\mathsf{R}) &= \sum_{i=1}^n d_{\boldsymbol{A}}\big(\boldsymbol{e}_i, \operatorname{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big)^2, \\
\eta_{\text{SDS}}(\mathsf{R}) &= \sum_{i=1}^n d_{\boldsymbol{A}}\big(\boldsymbol{e}_i, \{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big)^2, \\
\eta_{\text{CPC}}(\mathsf{R}) &= \max_{1\le i\le n} d_{\boldsymbol{A}}\big(\boldsymbol{e}_i, \operatorname{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big), \text{ or} \\
\eta_{\text{FPS}}(\mathsf{R}) &= \max_{1\le i\le n} d_{\boldsymbol{A}}\big(\boldsymbol{e}_i, \{\boldsymbol{e}_j\}_{j\in\mathsf{R}}\big).
\end{aligned} \tag{4.2}$$

Each algorithm uses a pivot selection rule naturally related to (4.2), and the approximation accuracy is controlled by the following theorem.

THEOREM 4.2 (Adaptive sampling guarantees). *For any positive-semidefinite matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ and any cardinality $r \le \operatorname{rank}(\boldsymbol{A})$, the adaptive sampling algorithms generate a random or deterministic pivot set $\mathsf{R} \subseteq \{1, \dots, n\}$ with cardinality $|\mathsf{R}| = r$*

that satisfies

$$\frac{\mathbb{E}\big[\eta_{\mathrm{RPC}}(\mathsf{R})\big]}{\min_{|\mathsf{S}|\le r}\eta_{\mathrm{RPC}}(\mathsf{S})} \le 2^{r-1}, \qquad \text{[RPC]}$$

$$\frac{\eta_{\mathrm{CPC}}(\mathsf{R})}{\min_{|\mathsf{S}|\le r}\eta_{\mathrm{CPC}}(\mathsf{S})} \le r!, \qquad \text{[CPC]}$$

$$\frac{\mathbb{E}\big[\eta_{\mathrm{SDS}}(\mathsf{R})\big]}{\min_{|\mathsf{S}|\le r}\eta_{\mathrm{SDS}}(\mathsf{S})} \le 5(\log r + 2), \qquad \text{[SDS]}$$

$$\frac{\eta_{\mathrm{FPS}}(\mathsf{R})}{\min_{|\mathsf{S}|\le r}\eta_{\mathrm{FPS}}(\mathsf{S})} \le 2. \qquad \text{[FPS]}$$

The adaptive sampling algorithms approximately minimize the objective functions η_{RPC}, η_{CPC}, η_{SDS}, *and* η_{FPS} *that are defined in* (4.2).

Proof. The RPC error bound is [6, Lem. 5.5]. The CPC error bound is implied by [8, Thm. 1] in the limit $p \to \infty$; see also [10, Thm. 2]. The SDS error bound is [1, Thm. 3.1], with an improved approximation factor due to [21, Lem. 4.1]. The FPS error bound is [15, Thm. 2.2]. ∎

Theorem 4.2 provides approximation guarantees that are independent of the dimensionality n and only depend on the cardinality r of the index set. As a limitation, the approximation factors 2^{r-1} and $r!$ in Theorem 4.2 increase rapidly with r. Yet there is some additional "oversampling" theory that addresses this issue. For example, randomly pivoted Cholesky with slightly more than r pivots controls the error functional nearly as well as the best approximation with r pivots [6, Thm. 2.3].

A second, more major limitation is that the adaptive sampling algorithms optimize the distance functionals η_{RPC}, η_{CPS}, η_{SDS}, and η_{FPS}, which are not the same as the Kaporin condition number κ_{Kap} (4.1). As such, it is no surprise that these algorithms lead to slightly worse approximation quality than adaptive search. However, they run much more quickly; see the experiments in section 5.

4.2. Adding to the sparsity pattern. Next we consider building a partial Cholesky + Vecchia approximation that optimizes the Kaporin condition number

$$\kappa_{\mathrm{Kap}} = \prod_{d_{\tilde{\boldsymbol{A}}}(\boldsymbol{e}_i,\,\mathrm{span}\{\boldsymbol{e}_j\}_{j<i})>0} \frac{d_{\tilde{\boldsymbol{A}}}\big(\boldsymbol{e}_i, \mathrm{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{S}_i}\big)^2}{d_{\tilde{\boldsymbol{A}}}\big(\boldsymbol{e}_i, \mathrm{span}\{\boldsymbol{e}_j\}_{j<i}\big)^2}, \qquad \text{where } \tilde{\boldsymbol{A}} = \boldsymbol{P}^*\boldsymbol{A}\boldsymbol{P}. \tag{4.3}$$

We observe that each numerator term is the square of a distance functional

$$\delta_i(\mathsf{S}_i) = d_{\tilde{\boldsymbol{A}}}\big(\boldsymbol{e}_i, \mathrm{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{S}_i}\big).$$

Therefore, we focus on adding q new indices to the index set S_i that approximately minimize δ_i. Unfortunately, finding the best q indices is NP-hard, by a reduction to the sparse recovery problem studied in [22, Thm. 1]. However, the Vecchia literature suggests a couple heuristic optimization methods [16, 18].

4.2.1. Nearest neighbor search. *Nearest neighbor (NN) search* is the most common method for adding q new indices to a sparsity component [16]. In this method, we recursively find a new index $j \notin \mathsf{S}_i$ that achieves the smallest possible distance $d_{\tilde{\boldsymbol{A}}}(\boldsymbol{e}_i, \boldsymbol{e}_j)$ and we add it.

$$\mathsf{S}_i \leftarrow \mathsf{S}_i \cup \{j\}, \quad \text{where } j \in \operatorname*{argmin}_{k\notin\mathsf{S}_i} d_{\tilde{\boldsymbol{A}}}(\boldsymbol{e}_i, \boldsymbol{e}_k) \qquad \text{[NN search]}.$$

Finding q new indices in this way requires looking up $\mathcal{O}(n)$ entries of the matrix and performing $\mathcal{O}(n)$ arithmetic operations [4]. In special cases we can more efficiently calculate approximate nearest neighbors for all n components, by using advanced data structures [2]. However, even though the NN search can be made relatively efficient, there is no guarantee that it does a good job at optimizing the distance functional δ_i.

4.2.2. Orthogonal matching pursuit. *Orthogonal matching pursuit* (OMP) is an alternative approach for adding q new indices based on the following rule.

$$\mathsf{S}_i \leftarrow \mathsf{S}_i \cup \{j\}, \quad \text{where } j \in \operatorname*{argmin}_{k \notin \mathsf{S}_i} \nu_i\big(\mathsf{S}_i \cup \{k\}\big) \qquad \text{[OMP]}.$$

OMP requires looking up $\mathcal{O}(qn)$ entries of the matrix and performing $\mathcal{O}(q^2 n)$ additional arithmetic operations [18, App. C.1]. Since each optimization is expensive, one option is to restrict the search to c candidate points that come from a NN search or uniformly random sampling [18]. Then, the algorithm only requires $\mathcal{O}(qc)$ entry look-ups and $\mathcal{O}(q^2 c)$ arithmetic operations where potentially $c \ll n$.

OMP is attractive because it directly targets the distance functional δ_i. However, the only available theoretical guarantees [27, Thms. B and C] require fast off-diagonal decay in the entries of $\boldsymbol{A}$, which seldom occurs in practice. Hence, in the worst case, OMP might result in a value δ_i that is incomparably worse than the best choice of q indices.

5. Experiments and analysis. In this section, we study the empirical performance of partial Cholesky + Vecchia approximations as applied to preconditioned conjugate gradient and log determinant estimation on Gaussian kernel matrices in real world kernel ridge regression problems.

5.1. Setup. Specifically, we work with 27 real-world datasets from libsvm and openML. Each dataset consists of a set of training vectors $\{\boldsymbol{x}_i\}_{i=1}^n \subseteq \mathbb{R}^d$, where we have scaled each coordinate so that the sample variance in each coordinate is 1, as well as additional test vectors whose coordinates we scaled in the same way. We use only the first $n = 20{,}000$ training vectors on any dataset with more than this due to VRAM constraints. The datasets are downloadable using scripts originally developed for [9], available at this link.

On each dataset, we take $\mathfrak{K}(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{(\boldsymbol{x}-\boldsymbol{x}')^*(\boldsymbol{x}-\boldsymbol{x}')}{2d})$ our Gaussian kernel function, and let $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ be such that $\boldsymbol{K}(i,j) = \mathfrak{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Finally, we take a nugget parameter and set $\boldsymbol{\Theta} = \boldsymbol{K} + \mu \boldsymbol{I}$. For each dataset, $\boldsymbol{\Theta}$ is the matrix we attempt to approximate. We test the quality of the approximation on the methods discussed in section 3 for both approximate application of the inverse and log determinant estimation.

5.1.1. Preconditioned Conjugate Gradient. For preconditioned conjugate gradient, our goal is to approximate $\boldsymbol{\Theta}^{-1}\boldsymbol{z}_{\text{test}}$ for $\boldsymbol{x}_{\text{test}}$ a random test vector withheld from the training vectors (but still normalized according to the sample variance of the training vectors in each coordinate) and $\boldsymbol{z}_{\text{test}}(i) = \mathfrak{K}(\boldsymbol{x}_i, \boldsymbol{x}_{\text{test}})$ using the method described in subsection 3.2. For our experiments, we choose five random $\boldsymbol{x}_{\text{test}}$ from each dataset and run PCG against $\boldsymbol{z}_{\text{test}}$, starting from the zero vector. For each dataset and preconditioner, we record the number of iterations required so that $\|\boldsymbol{z}_{\text{test}} - \boldsymbol{\Theta}\hat{\boldsymbol{w}}^{(k)}\|_2 \leq 10^{-4}\|\boldsymbol{z}_{\text{test}}\|_2$, where $\hat{\boldsymbol{w}}^{(k)}$ is PCG's approximation of $\boldsymbol{\Theta}^{-1}\boldsymbol{z}_{\text{test}}$ at the kth iteration, at which point we consider the problem "solved". We repeat this for nugget values $10^{-10}, 10^{-6}$, and 10^{-3}. We then plot, as a function of the number of iterations

k, the percentage of kernel ridge regression problems "solved" by iteration k by each preconditioner.

Note that this is in contrast to the usual procedure for kernel ridge regression, where one forms a vector $\boldsymbol{y}$ whose ith coordinate $\boldsymbol{y}(i)$ is the label associated to data point $\boldsymbol{x}_i$ for all $i \in 1 \dots n$ and computes $\boldsymbol{y}^*\boldsymbol{\Theta}^{-1}$. Both aim to eventually compute $\boldsymbol{y}^*\boldsymbol{\Theta}^{-1}\boldsymbol{z}_{\text{test}}$, but computing $\boldsymbol{y}^*\boldsymbol{\Theta}^{-1}$ first requires only one application of $\boldsymbol{\Theta}^{-1}$ even if there are multiple points $\boldsymbol{z}_{\text{test}}^{(1)}, \boldsymbol{z}_{\text{test}}^{(2)}, \dots$ whose labels we want to predict. However, we find empirically that PCG with Vecchia preconditioners requires orders of magnitude fewer iterations when computing $\boldsymbol{\Theta}^{-1}\boldsymbol{z}_{\text{test}}$ as opposed to $\boldsymbol{y}^*\boldsymbol{\Theta}^{-1}$ to attain the same 10^{-4} relative error tolerance, especially for small nugget; see Figure 8. While we also observe improvement using partial Cholesky + Vecchia or partial Cholesky + diagonal over existing methods when approximating $\boldsymbol{y}^*\boldsymbol{\Theta}^{-1}$, we still recommend computing $\boldsymbol{\Theta}^{-1}\boldsymbol{z}_{\text{test}}$ for each test point as opposed to $\boldsymbol{y}^*\boldsymbol{\Theta}^{-1}$ once, unless one must estimate labels at several hundred or more test points.

5.1.2. Log-determinant estimation. For log-determinant estimation, we implement both direct and indirect estimators as described in subsection 3.3 and subsection 3.4 respectively. For the indirect estimator, we use a Krylov subspace depth of 100 and estimate the log determinant as the mean of 10 samples. For our figures, we only plot data from the large nugget ($\mu = 10^{-3}$ case, as it was the only case where any tractable preconditioner we tested attained reasonably accurate estimates (though we present data for other nuggets in tables in Table 2, Table 3, and Table 4. After obtaining the direct estimators and the indirect estimates, we obtained the absolute error of the estimates compared to the ground truth log determinant on each dataset (computed with a GPU), then presented the median of this error over the 27 datasets. Note that we are plotting error in the log determinant, and the y-axis is also in log scale, so the y axes are akin to log-of-log scale of absolute error in the determinant.

Importantly, while the indirect estimator is unbiased if the matrix logarithm is computed exactly, this is not a reasonable assumption in practice for all but the best preconditioners, at least with a feasible Krylov depth (in our case, 100). So while Proposition 3.6 focuses mainly on bounding the variance, the error shown in the figures mostly reflects bias.

5.2. Comparison of Pivot Choosers. In this section, we compare the performance of the four different adaptive sampling pivot choosers from subsection 4.1.2 when used for partial Cholesky + diagonal and partial Cholesky + Vecchia preconditioners, both in preconditioned conjugate gradient and log determinant estimation. In figures, we refer to them by their respective three-letter abbreviations from subsection 4.1.2. We also compare them to the adaptive search preconditioner in subsection 4.1.1, which we made tractable for 20,000 × 20,000 matrices despite its $O(n^2 r)$ runtime by writing a custom GPU kernel for the algorithm using the package `CUDA.jl` and running it on an RTX 3090. We refer to adaptive search as "AS" in figures.

For the partial Cholesky portion, we always take $r = \lfloor \sqrt{n} \rfloor$ as the rank of the partial Cholesky decomposition. For the Vecchia portion, we take $s = \lfloor n^{1/4} \rfloor$ as the maximum number of off-diagonal nonzeros per column. We always use greedy conditional selection to choose the sparsity pattern for partial Cholesky + Vecchia, and always with $c = 10s$ candidates.

For PCG, we plot the results using the procedure described in subsection 5.1.1 in Figure 5. We observe that among adaptive sampling methods, RPC and SDS, which perform similarly, are significantly better than CPC and FPS. Unsurprisingly,

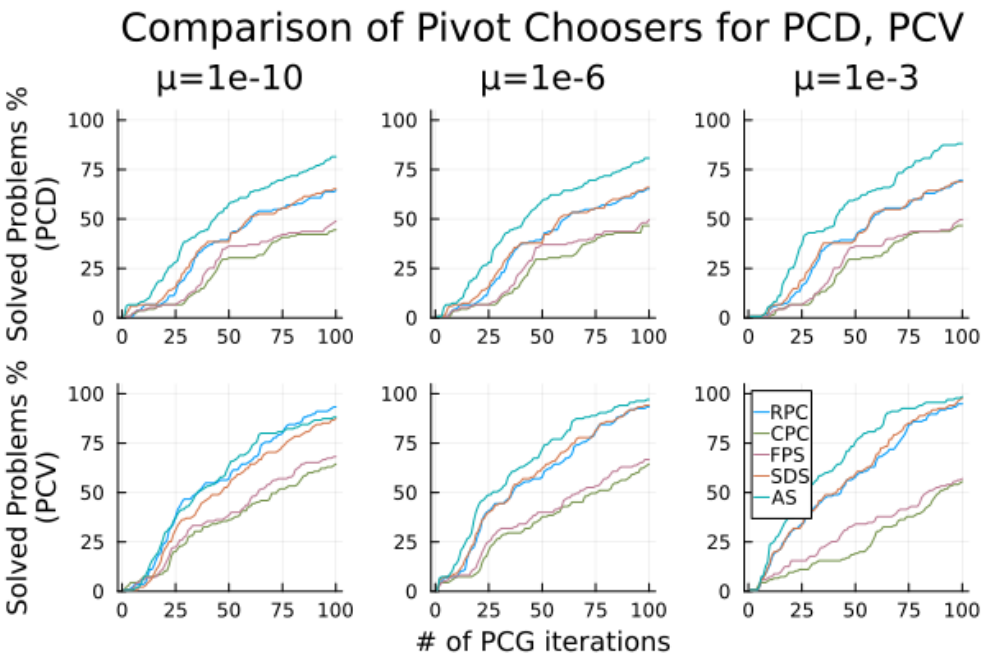

FIG. 5. *Comparison of pivot choosers used in partial Cholesky + diagonal and partial Cholesky + Vecchia preconditioners for conjugate gradient*

adaptive search is generally better than adaptive sampling methods, and the gap is significant for partial Cholesky + diagonal, suggesting the Kaporin condition number could reasonably be used as a heuristic to improve existing pivot choosing methods. However, the gap is nearly insignificant when applied to partial Cholesky + Vecchia, providing evidence that Kaporin-inspired pivot choosers couldn't hope to improve partial Cholesky + Vecchia approximations much over existing pivot choosers. We hypothesize that this is because Kaporin-oriented greedy conditional selection "picks up the slack" left by adaptive sampling methods when choosing the sparsity pattern. Finally, we observe that relative performance is not significantly affected by varying the nugget parameter.

For log determinant estimation, we give a bar chart using the procedure described in subsection 5.1.2 (in particular, this data is only for $\mu = 10^{-3}$) in Figure 6. We again observe that AS outperforms RPC and SDS, which outperform CPC and FPS. The gap between AS and RPC or SDS, especially for partial Cholesky + Vecchia, is more noticeable than with PCG, suggesting the Kaporin condition number is perhaps a better pivot choosing heuristic when applied to log determinant estimation than PCG..

5.3. How much does Vecchia help?. To study the improvement of partial Cholesky + diagonal and/or partial Cholesky + Vecchia over previous methods, we implement [11]'s preconditioner ("Frangella" in figures), [9]'s preconditioner ("Diaz" in figures), and compare them to Randomly Pivoted Cholesky + diagonal, Randomly Pivoted Cholesky + Vecchia with $s = \lfloor n^{1/4} \rfloor$ nonzero entries per column, and Randomly Pivoted Cholesky + Vecchia with $s = \lfloor n^{1/3} \rfloor$ nonzero entries per column

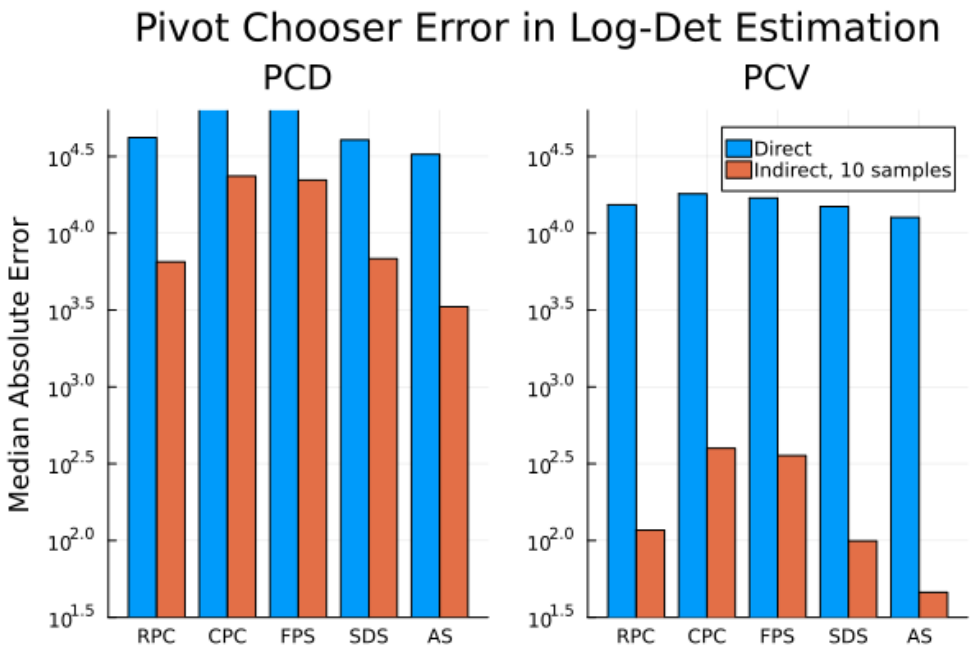

FIG. 6. *Comparison of pivot choosers used in partial Cholesky + diagonal and partial Cholesky + Vecchia estimators or preconditioners for log determinant estimation.*

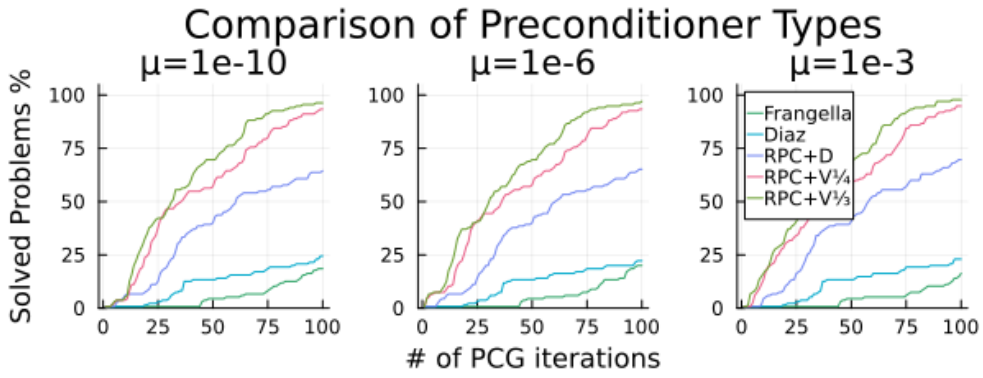

FIG. 7. *Comparison of [11]'s and [9]'s RPC-based preconditioners (which use a Kaporin-suboptimal diagonal), RPC plus diagonal, RPC plus Vecchia with $s = n^{1/4}$, and RPC plus Vecchia with $s = n^{1/3}$, as applied to PCG on 5 kernelized test vectors per dataset as described in subsection 5.1.1.*

("RPC+D", "RPC+V$\frac{1}{4}$", and "RPC+V$\frac{1}{3}$" respectively), each with $c = 10s$ GCS canditates.

For PCG applied to the kernelized test vectors, we observe in Figure 7 that simply moving from [11]'s or [9]'s diagonal heuristics to the Kaporin-optimal choice of diagonal provided by partial Cholesky + Vecchia approximations makes a dramatic difference, more than doubling the number of problems solved within 100 PCG it-

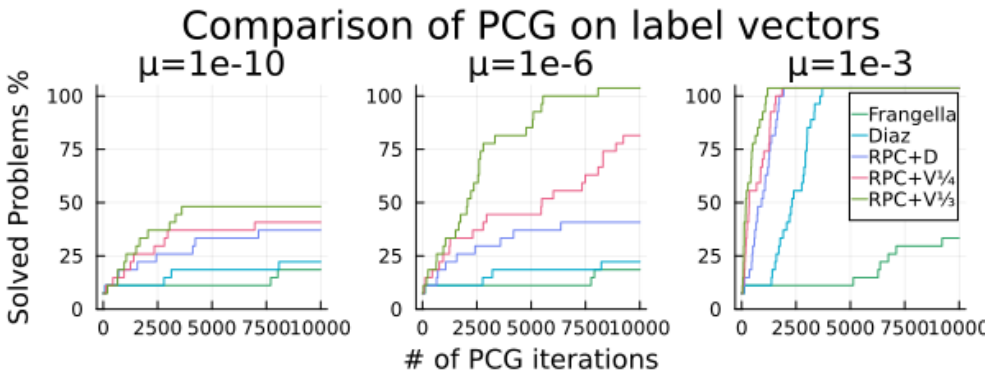

FIG. 8. *Comparison of [11]'s and [9]'s RPC-based preconditioners (which use a Kaporin-suboptimal diagonal), RPC plus diagonal, RPC plus Vecchia with $s = n^{1/4}$, and RPC plus Vecchia with $s = n^{1/3}$, as applied to PCG on the vector of labels for each dataset.*

erations. As both strategies require the same computational complexity, this result strongly favors partial Cholesky + diagonal approximations. Beyond this, adding $s = n^{1/4}$ entries to the sparsity pattern results in a considerable benefit, solving all but two problems within 100 iterations, but there is only modest benefit from increasing s to $n^{1/3}$. We also observe that changing the nugget parameter does not substantially affect the performance of any of these methods, which means they can be used in a variety of settings.

We also examine PCG applied to the vector of labels in Figure 8, where the results are considerably different. Although the relative order of the methods remains unchanged, performance is worse across the board compared to PCG on the kernelized test vectors. Moreover, the results are highly nugget-dependent. In the extremely small nugget regime, even partial Cholesky + Vecchia with $n = s^{1/3}$ fails to solve half the problems within 10000 iterations. As the nugget increases, all methods perform better, and the significance of choosing a larger sparsity set first grows as more problems are able to be solved, then shrinks as simpler Vecchia preconditioners also become capable of solving the problems.

For log determinant estimation (Figure 9), the relative order remains, except that [9]'s estimator slightly outperforms RPC+D. We suspect this is because [9]'s separation of its approximation of $\boldsymbol{K}$ from its addition of the nugget is beneficial in large nugget regimes like $\mu = 10^{-3}$. However, noting the logarithmic scale of the figure, the only methods that could reasonably be usable for most tasks depending on log determinant estimates (e.g. approximating likelihood ratios) are partial Cholesky + Vecchia. We also observe that the indirect estimator is essentially necessary to obtain a reasonably small error, even for partial Cholesky + Vecchia.

Acknowledgments. The authors would like to thank Christopher J. Geoga, Chris Camaño, Ethan N. Epperly, and Florian Schäfer for helpful discussions.

REFERENCES

[1] D. ARTHUR AND S. VASSILVITSKII, *k-means++: The advantages of careful seeding*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, https://dl.acm.org/doi/10.5555/1283383.1283494.

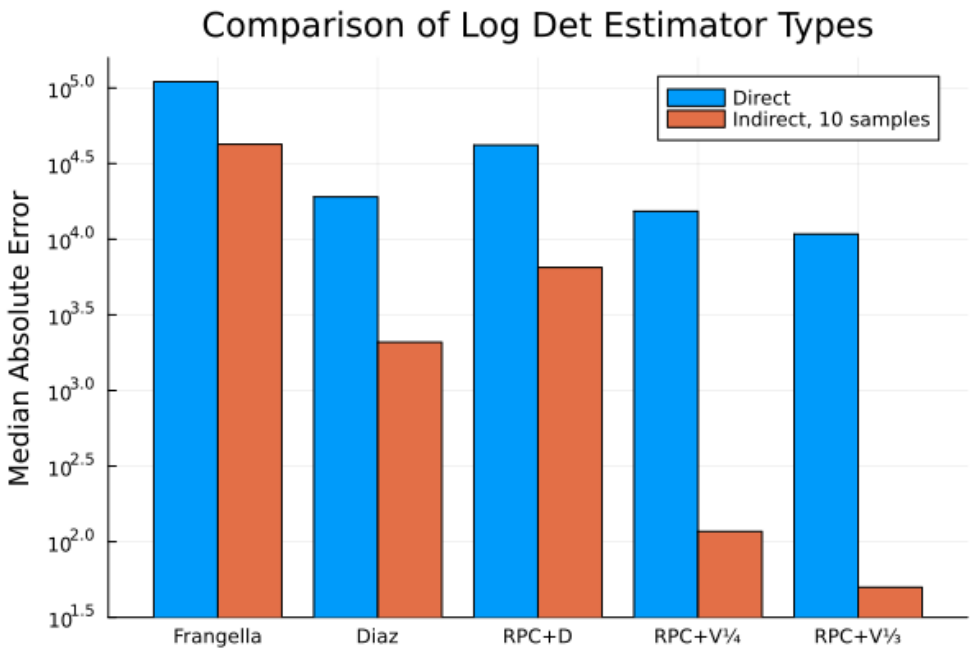

FIG. 9. *Comparison of [11]'s and [9]'s RPC-based preconditioners (which use a Kaporin-suboptimal diagonal), RPC plus diagonal, RPC plus Vecchia with $s = n^{1/4}$, and RPC plus Vecchia with $s = n^{1/3}$, as applied to log determinant estimation.*

[2] S. ARYA, D. M. MOUNT, N. S. NETANYAHU, R. SILVERMAN, AND A. Y. WU, *An optimal algorithm for approximate nearest neighbor searching fixed dimensions*, Journal of the ACM, 45 (1998), p. 891–923, https://doi.org/10.1145/293347.293348.

[3] O. AXELSSON AND I. KAPORIN, *On the sublinear and superlinear rate of convergence of conjugate gradient methods*, Numerical Algorithms, 25 (2000), p. 1–22, https://doi.org/10.1023/a:1016694031362.

[4] M. BLUM, R. W. FLOYD, V. PRATT, R. L. RIVEST, AND R. E. TARJAN, *Time bounds for selection*, Journal of Computer and System Sciences, 7 (1973), pp. 448–461, https://doi.org/10.1016/S0022-0000(73)80033-9.

[5] D. CAI, E. CHOW, AND Y. XI, *Posterior covariance structures in gaussian processes*, SIAM Journal on Matrix Analysis and Applications, 46 (2025), pp. 1640–1673, https://doi.org/10.1137/24M1684918.

[6] Y. CHEN, E. N. EPPERLY, J. A. TROPP, AND R. J. WEBBER, *Randomly pivoted Cholesky: Practical approximation of a kernel matrix with few entry evaluations*, Communications on Pure and Applied Mathematics, 78 (2025), pp. 995–1041, https://doi.org/10.1002/cpa.22234.

[7] A. P. DEMPSTER, *Covariance selection*, Biometrics, 28 (1972), pp. 157–175, https://doi.org/https://doi.org/10.2307/2528966.

[8] A. DESHPANDE AND K. VARADARAJAN, *Sampling-based dimension reduction for subspace approximation*, in Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing, 2007, https://doi.org/10.1145/1250790.1250884.

[9] M. DÍAZ, E. N. EPPERLY, Z. FRANGELLA, J. A. TROPP, AND R. J. WEBBER, *Robust, randomized preconditioning for kernel ridge regression*, 2024, https://arxiv.org/abs/2304.12465.

[10] H. ENGLER, *The behavior of the QR-factorization algorithm with column pivoting*, Applied Mathematics Letters, 10 (1997), pp. 7–11, https://doi.org/10.1016/S0893-9659(97)00098-0.

[11] Z. FRANGELLA, J. A. TROPP, AND M. UDELL, *Randomized nyström preconditioning*, SIAM Journal on Matrix Analysis and Applications, 44 (2023), pp. 718–752, https://doi.org/10.1137/21M1466244.

name	quantity	Frangella	RPC+D	RPC+V$^{\frac{1}{2}}$	RPC+V^{1}	CPC+D	FPS+D	GKL+D	SDS+D	CPC+V$^{\frac{1}{2}}$	FPS+V$^{\frac{1}{2}}$	GKL+V$^{\frac{1}{2}}$	SDS+V$^{\frac{1}{2}}$	Diaz
	direct	334.845	-6.2864e+04	-1.1253e+05	-1.1754e+05	-3.1689e+04	-3.8907e+04	-7.6956e+04	-6.7110e+04	-1.0960e+05	-1.1035e+05	-1.1344e+05	-1.1289e+05	-1.3681e+05
ACSIncome	mean	-2.3689e+04	-1.0827e+05	-1.2515e+05	-1.2525e+05	-7.7495e+04	-9.0769e+04	-1.1251e+05	-1.0836e+05	-1.2486e+05	-1.2529e+05	-1.2526e+05	-1.2517e+05	-1.2246e+05
(Truth: -1.2527e+05)	stdev	681.065	631.001	190.929	126.609	657.369	780.049	414.095	430.400	206.647	249.839	242.794	274.351	524.556
	direct	369.476	-6.4580e+04	-1.1333e+05	-1.1958e+05	-4.6907e+04	-5.3631e+04	-7.3915e+04	-6.6741e+04	-1.1060e+05	-1.1170e+05	-1.1449e+05	-1.1373e+05	-1.3681e+05
Airlines_DepDelay_1M	mean	-4.1623e+04	-1.1215e+05	-1.2823e+05	-1.2866e+05	-1.0671e+05	-1.1120e+05	-1.1777e+05	-1.1346e+05	-1.2824e+05	-1.2828e+05	-1.2840e+05	-1.2838e+05	-1.2657e+05
(Truth: -1.2863e+05)	stdev	1401.108	544.001	248.538	186.685	1046.949	426.923	282.680	660.750	141.401	192.545	346.960	274.493	189.154
	direct	544.274	-1.2677e+05	-1.3512e+05	-1.3589e+05	-9.5746e+04	-9.8808e+04	-1.3035e+05	-1.2811e+05	-1.3482e+05	-1.3486e+05	-1.3526e+05	-1.3520e+05	-1.3711e+05
COMET_MC_SAMPLE	mean	-1.3159e+05	-1.3631e+05	-1.3654e+05	-1.3652e+05	-1.3610e+05	-1.3604e+05	-1.3618e+05	-1.3617e+05	-1.3653e+05	-1.3653e+05	-1.3654e+05	-1.3655e+05	-1.3643e+05
(Truth: -1.3653e+05)	stdev	1826.516	157.678	75.074	53.732	536.307	426.772	116.022	134.259	64.210	118.108	78.734	47.944	63.070
	direct	365.986	-8.2071e+04	-1.2416e+05	-1.2674e+05	-3.7946e+04	-3.9718e+04	-9.8787e+04	-8.7935e+04	-1.2192e+05	-1.2219e+05	-1.2464e+05	-1.2444e+05	-1.3687e+05
Click_prediction_small	mean	-5.8934e+04	-1.2247e+05	-1.3019e+05	-1.3010e+05	-1.0320e+05	-1.0532e+05	-1.2559e+05	-1.2312e+05	-1.3008e+05	-1.3018e+05	-1.3016e+05	-1.3018e+05	-1.2925e+05
(Truth: -1.3017e+05)	stdev	1869.905	467.448	190.691	142.414	644.305	1319.048	468.703	478.371	218.865	160.467	126.979	148.108	313.733
	direct	307.176	-3.5339e+04	-5.7627e+04	-6.3177e+04	-1.4202e+04	-1.3908e+04	-3.9166e+04	-3.5332e+04	-5.0668e+04	-5.0663e+04	-5.8817e+04	-5.7623e+04	-1.3682e+05
HIGGS	mean	-4.3235e+04	-7.7894e+04	-8.9008e+04	-8.9476e+04	-5.3704e+04	-5.3657e+04	-8.3727e+04	-7.7353e+04	-8.7082e+04	-8.6740e+04	-8.9060e+04	-8.8816e+04	-8.4939e+04
(Truth: -8.9494e+04)	stdev	1045.022	633.831	616.038	273.716	500.336	654.686	553.356	714.033	511.948	362.210	288.936	590.065	666.023
	direct	334.847	-3.2156e+04	-5.1422e+04	-5.5705e+04	-4445.350	-3655.401	-3.7606e+04	-3.1760e+04	-4.7410e+04	-4.7407e+04	-5.3137e+04	-5.1495e+04	-1.3686e+05
MNIST	mean	-4.6163e+04	-6.7671e+04	-7.0694e+04	-7.0737e+04	-4.8851e+04	-4.8573e+04	-7.0307e+04	-6.6813e+04	-7.0399e+04	-7.0361e+04	-7.0609e+04	-7.0590e+04	-6.8281e+04
(Truth: -7.0740e+04)	stdev	494.214	388.971	231.871	216.753	540.955	584.384	361.600	310.046	339.559	386.977	156.285	166.022	707.193
	direct	344.702	-4.8448e+04	-9.5010e+04	-1.0252e+05	-3.2529e+04	-3.7509e+04	-5.5835e+04	-4.9297e+04	-9.1787e+04	-9.2628e+04	-9.6599e+04	-9.5284e+04	-1.3678e+05
Medical-Appointment	mean	-2.4469e+04	-9.4161e+04	-1.1734e+05	-1.1765e+05	-6.7118e+04	-7.5606e+04	-1.0107e+05	-9.4446e+04	-1.1638e+05	-1.1684e+05	-1.1751e+05	-1.1733e+05	-1.1087e+05
(Truth: -1.1774e+05)	stdev	735.489	409.146	294.861	264.997	665.003	593.038	565.881	411.805	211.164	216.395	362.523	346.267	602.909
	direct	434.227	-1.1491e+05	-1.2906e+05	-1.3132e+05	-8.4264e+04	-9.2031e+04	-1.2033e+05	-1.1643e+05	-1.2681e+05	-1.2701e+05	-1.2987e+05	-1.2924e+05	-1.3714e+05
MiniBooNE	mean	-1.3090e+05	-1.3439e+05	-1.3494e+05	-1.3492e+05	-1.3394e+05	-1.3421e+05	-1.3460e+05	-1.3447e+05	-1.3497e+05	-1.3488e+05	-1.3493e+05	-1.3487e+05	-1.3479e+05
(Truth: -1.3493e+05)	stdev	1535.619	376.637	177.803	114.152	303.253	331.832	258.444	251.644	136.893	171.195	174.572	214.645	141.837
	direct	300.073	-3.3351e+04	-4.8600e+04	-5.3492e+04	-1.2935e+04	-1.2945e+04	-4.2656e+04	-3.4734e+04	-4.0707e+04	-4.0708e+04	-5.3255e+04	-4.9211e+04	-1.3688e+05
YearPredictionMSD	mean	-3.6243e+04	-6.8724e+04	-7.4172e+04	-7.5143e+04	-4.8014e+04	-4.8183e+04	-7.3891e+04	-7.0087e+04	-7.1299e+04	-7.1423e+04	-7.5190e+04	-7.4431e+04	-7.2954e+04
(Truth: -7.5234e+04)	stdev	464.650	525.411	444.036	294.478	520.207	431.839	245.356	507.840	519.697	238.316	407.982	313.404	1169.069
	direct	390.617	-3.6516e+04	-6.9339e+04	-7.5355e+04	-1.0481e+04	-1.2382e+04	-4.3540e+04	-3.7320e+04	-6.3403e+04	-6.3606e+04	-7.1366e+04	-6.9454e+04	-1.3580e+05
a9a	mean	-2.6995e+04	-7.7022e+04	-9.0674e+04	-9.0963e+04	-3.8335e+04	-4.0240e+04	-8.6246e+04	-7.8684e+04	-8.8259e+04	-8.8514e+04	-9.0766e+04	-9.0681e+04	-8.6520e+04
(Truth: -9.0819e+04)	stdev	525.440	730.245	436.426	214.374	451.557	367.468	531.301	658.115	408.269	336.019	329.394	271.758	606.337
	direct	396.911	-8.4913e+04	-1.0746e+05	-1.0888e+05	-4.8752e+04	-6.0195e+04	-9.3483e+04	-8.8055e+04	-1.0636e+05	-1.0672e+05	-1.0778e+05	-1.0758e+05	-1.1298e+05
cadata	mean	-9.6039e+04	-1.0882e+05	-1.1062e+05	-1.1060e+05	-1.0767e+05	-1.0827e+05	-1.0886e+05	-1.0849e+05	-1.1066e+05	-1.1063e+05	-1.1062e+05	-1.1060e+05	-1.1026e+05
(Truth: -1.1061e+05)	stdev	1015.349	338.949	108.997	82.596	698.542	479.771	191.300	334.357	108.104	142.108	109.371	106.229	46.488
	direct	412.645	-9.3538e+04	-1.2893e+05	-1.3092e+05	-5.6889e+04	-7.0179e+04	-1.0452e+05	-9.7116e+04	-1.2767e+05	-1.2818e+05	-1.2942e+05	-1.2915e+05	-1.3688e+05
cod-rna	mean	-8.2082e+04	-1.2752e+05	-1.3327e+05	-1.3331e+05	-1.2605e+05	-1.2758e+05	-1.2791e+05	-1.2647e+05	-1.3340e+05	-1.3328e+05	-1.3323e+05	-1.3336e+05	-1.3271e+05
(Truth: -1.3330e+05)	stdev	1693.477	430.720	173.391	140.765	489.192	194.441	210.147	280.583	141.838	168.978	149.338	106.182	119.019
	direct	436.692	-3.6931e+04	-7.3114e+04	-8.0005e+04	-2.3851e+04	-2.6701e+04	-4.1805e+04	-3.8151e+04	-6.9031e+04	-6.9733e+04	-7.4614e+04	-7.3558e+04	-1.3680e+05
connect-4	mean	-3.0513e+04	-7.9013e+04	-9.6382e+04	-9.6718e+04	-5.7403e+04	-6.1399e+04	-8.7759e+04	-8.1265e+04	-9.5190e+04	-9.5873e+04	-9.6722e+04	-9.6508e+04	-9.2736e+04
(Truth: -9.6767e+04)	stdev	519.676	439.614	456.823	377.811	491.442	451.741	474.423	769.840	444.849	500.929	383.780	236.957	669.897
	direct	454.444	-7.3306e+04	-1.2538e+05	-1.2765e+05	-4.4468e+04	-5.5897e+04	-8.5953e+04	-7.4830e+04	-1.2436e+05	-1.2477e+05	-1.2585e+05	-1.2541e+05	-1.3682e+05
covtype_binary	mean	-4.2024e+04	-1.1700e+05	-1.2984e+05	-1.2981e+05	-1.0074e+05	-1.1224e+05	-1.1918e+05	-1.1657e+05	-1.2992e+05	-1.2986e+05	-1.2981e+05	-1.2984e+05	-1.2776e+05
(Truth: -1.2985e+05)	stdev	1065.614	553.854	167.341	82.286	544.802	858.753	641.038	347.044	140.456	93.190	145.128	176.941	231.861
	direct	341.541	-4.7290e+04	-9.0957e+04	-9.6467e+04	-5475.087	-8029.605	-6.1002e+04	-4.9405e+04	-8.5242e+04	-8.5376e+04	-9.3187e+04	-9.1324e+04	-1.3685e+05
creditcard	mean	-3.3639e+04	-9.1483e+04	-1.0921e+05	-1.0942e+05	-3.9204e+04	-4.2008e+04	-1.0021e+05	-9.1323e+04	-1.0755e+05	-1.0765e+05	-1.0938e+05	-1.0938e+05	-9.9553e+04
(Truth: -1.0955e+05)	stdev	919.532	551.257	353.856	201.626	2067.358	942.980	497.463	386.728	230.956	190.024	187.160	257.150	289.465
	direct	420.947	-9.4323e+04	-1.2907e+05	-1.3115e+05	-5.3694e+04	-6.9985e+04	-1.0474e+05	-9.8605e+04	-1.2772e+05	-1.2831e+05	-1.2927e+05	-1.2912e+05	-1.3691e+05
diamonds	mean	-9.1277e+04	-1.2995e+05	-1.3373e+05	-1.3370e+05	-1.2550e+05	-1.2790e+05	-1.3038e+05	-1.2905e+05	-1.3373e+05	-1.3358e+05	-1.3370e+05	-1.3376e+05	-1.3330e+05
(Truth: -1.3372e+05)	stdev	1523.201	459.712	147.119	95.742	855.481	786.498	320.502	374.067	202.342	163.998	96.625	95.143	143.682
	direct	458.793	-1.0550e+05	-1.2988e+05	-1.3207e+05	-8.2491e+04	-9.0911e+04	-1.1175e+05	-1.0578e+05	-1.2906e+05	-1.2941e+05	-1.3006e+05	-1.2982e+05	-1.3700e+05
hls4ml_lhc_jets_hlf	mean	-1.2062e+05	-1.3324e+05	-1.3458e+05	-1.3454e+05	-1.3269e+05	-1.3310e+05	-1.3333e+05	-1.3270e+05	-1.3453e+05	-1.3452e+05	-1.3460e+05	-1.3457e+05	-1.3430e+05
(Truth: -1.3455e+05)	stdev	1642.532	309.136	63.948	122.235	651.027	431.501	194.544	267.382	211.056	136.712	126.157	114.718	118.022
	direct	383.848	-6.1634e+04	-1.1392e+05	-1.1832e+05	-2.9214e+04	-3.6693e+04	-7.7972e+04	-6.2816e+04	-1.1239e+05	-1.1273e+05	-1.1420e+05	-1.1389e+05	-1.3680e+05
ijcnn1	mean	-3.3873e+04	-1.0553e+05	-1.2385e+05	-1.2376e+05	-7.1030e+04	-8.5248e+04	-1.1340e+05	-1.0493e+05	-1.2371e+05	-1.2373e+05	-1.2387e+05	-1.2382e+05	-1.1926e+05
(Truth: -1.2383e+05)	stdev	643.282	679.864	262.984	162.527	962.454	528.583	220.525	614.284	242.612	312.572	147.162	232.451	291.496
	direct	296.254	-2.7846e+04	-4.1473e+04	-4.5695e+04	-1.5077e+04	-1.6102e+04	-3.1524e+04	-2.7863e+04	-3.6655e+04	-3.7092e+04	-4.2590e+04	-4.1457e+04	-1.3684e+05
jannis	mean	-5.3977e+04	-6.5018e+04	-6.7421e+04	-6.7760e+04	-5.8763e+04	-5.9286e+04	-6.6624e+04	-6.5532e+04	-6.6918e+04	-6.6839e+04	-6.7387e+04	-6.7549e+04	-6.7112e+04
(Truth: -6.7629e+04)	stdev	581.731	493.705	394.852	580.738	470.684	657.624	382.370	423.126	446.733	428.016	298.098	413.465	950.700
	direct	151.107	-7411.038	-1.5058e+04	-1.6366e+04	-4627.919	-4866.678	-8934.265	-7484.812	-1.4249e+04	-1.4308e+04	-1.5535e+04	-1.5078e+04	-3.4623e+04
phishing	mean	-8713.223	-1.8569e+04	-2.0118e+04	-2.0123e+04	-1.2750e+04	-1.2991e+04	-1.9654e+04	-1.8633e+04	-2.0051e+04	-1.9968e+04	-1.9998e+04	-2.0042e+04	-1.9272e+04
(Truth: -2.0079e+04)	stdev	141.840	366.919	187.682	149.784	439.265	341.913	212.769	281.780	131.203	115.444	100.918	164.168	252.101
	direct	307.760	-1.3522e+04	-1.7143e+04	-1.9643e+04	-1.2910e+04	-1.2919e+04	-1.4392e+04	-1.3536e+04	-1.6657e+04	-1.6675e+04	-1.7793e+04	-1.7160e+04	-1.3679e+05
santander	mean	-3.0250e+04	-3.0372e+04	-3.0410e+04	-3.0375e+04	-3.0313e+04	-3.0441e+04	-3.0390e+04	-3.0528e+04	-3.0480e+04	-3.0579e+04	-3.0503e+04	-3.0452e+04	-3.0421e+04
(Truth: -3.0466e+04)	stdev	451.504	177.149	203.206	139.763	313.523	260.562	148.539	232.805	214.325	292.871	137.495	216.849	1015.891
	direct	302.999	-3.9264e+04	-6.2419e+04	-6.7501e+04	-1.6818e+04	-1.6759e+04	-5.4236e+04	-3.9336e+04	-5.6922e+04	-5.6897e+04	-6.6423e+04	-6.2698e+04	-1.3689e+05
sensit_vehicle	mean	-2.3976e+04	-7.6516e+04	-8.1223e+04	-8.1364e+04	-4.6982e+04	-4.7200e+04	-8.0872e+04	-7.6814e+04	-7.9724e+04	-7.9703e+04	-8.1460e+04	-8.1130e+04	-7.6592e+04
(Truth: -8.1473e+04)	stdev	777.125	426.125	152.798	298.921	583.034	489.374	377.015	368.662	345.798	395.643	216.860	231.766	765.183
	direct	324.221	-5.7404e+04	-9.3336e+04	-1.0056e+05	-2.7673e+04	-3.1935e+04	-6.6285e+04	-5.8435e+04	-8.8029e+04	-8.8682e+04	-9.4726e+04	-9.3669e+04	-1.3684e+05
sensorless	mean	-4.7981e+04	-1.0458e+05	-1.1687e+05	-1.1717e+05	-7.0026e+04	-7.4891e+04	-1.1025e+05	-1.0543e+05	-1.1538e+05	-1.1560e+05	-1.1712e+05	-1.1719e+05	-1.1398e+05
(Truth: -1.1736e+05)	stdev	790.815	367.840	391.474	228.070	388.794	259.302	314.499	464.837	308.558	303.069	297.195	320.440	353.842
	direct	519.161	-7.2587e+04	-7.5642e+04	-7.5950e+04	-7.1532e+04	-7.2538e+04	-7.3642e+04	-7.2533e+04	-7.5514e+04	-7.5569e+04	-7.5656e+04	-7.5639e+04	-7.6293e+04
skin_nonskin	mean	-7.4990e+04	-7.6242e+04	-7.6289e+04	-7.6288e+04	-7.6096e+04	-7.6204e+04	-7.6288e+04	-7.6205e+04	-7.6279e+04	-7.6262e+04	-7.6278e+04	-7.6274e+04	-7.6280e+04
(Truth: -7.6281e+04)	stdev	801.067	86.027	74.116	44.680	76.143	49.154	46.108	89.715	79.217	71.462	63.574	48.515	3.255
	direct	338.316	-4.1925e+04	-6.0368e+04	-6.6963e+04	-2.2793e+04	-2.4114e+04	-4.7360e+04	-4.1880e+04	-5.4507e+04	-5.4682e+04	-6.4142e+04	-6.0279e+04	-1.3684e+05
volkert	mean	-5.0125e+04	-8.3528e+04	-8.8681e+04	-8.9992e+04	-6.6925e+04	-6.8103e+04	-8.7689e+04	-8.3626e+04	-8.6014e+04	-8.6165e+04	-9.0027e+04	-8.8841e+04	-8.7235e+04
(Truth: -9.0445e+04)	stdev	749.510	534.948	355.853	506.511	564.416	667.478	595.732	314.501	539.299	255.391	326.941	553.132	521.088
	direct	361.182	-2.6315e+04	-4.7931e+04	-5.2613e+04	-1.1726e+04	-1.1751e+04	-3.6964e+04	-2.6193e+04	-4.2663e+04	-4.2703e+04	-5.3215e+04	-4.7827e+04	-1.2284e+05
w8a	mean	-4.0155e+04	-5.6918e+04	-6.4267e+04	-6.4539e+04	-4.6567e+04	-4.6376e+04	-6.1324e+04	-5.6914e+04	-6.3259e+04	-6.3396e+04	-6.4413e+04	-6.4307e+04	-6.0613e+04
(Truth: -6.4648e+04)	stdev	654.053	472.312	235.256	175.029	992.436	975.766	228.100	462.505	323.408	382.396	272.378	421.683	558.045
	direct	332.002	-3.0460e+04	-4.1784e+04	-4.5952e+04	-1016.972	-1200.986	-3.7687e+04	-3.0366e+04	-3.3334e+04	-3.3358e+04	-4.5418e+04	-4.1687e+04	-1.3686e+05
yolanda	mean	-3.8924e+04	-6.3560e+04	-6.6313e+04	-6.6997e+04	-3.9408e+04	-3.9066e+04	-6.6583e+04	-6.3998e+04	-6.3285e+04	-6.3543e+04	-6.6981e+04	-6.6548e+04	-6.5738e+04
(Truth: -6.7057e+04)	stdev	614.046	384.050	451.450	361.899	1102.380	1251.893	263.187	329.048	319.041	387.431	306.098	506.957	460.553

TABLE 2

Log determinant estimation data for direct and indirect preconditioners on each dataset, for $\mu = 10^{-3}$

[12] A. FROMMER AND M. SCHWEITZER, *Error bounds and estimates for Krylov subspace approximations of Stieltjes matrix functions*, 56 (2016), https://doi.org/10.1007/s10543-015-0596-3.

[13] G. GOLUB, *Numerical methods for solving linear least squares problems*, Numerische Mathematik, 7 (1965), p. 206–216, https://doi.org/10.1007/bf01436075.

[14] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations - 4th Edition*, Johns Hopkins University Press, Philadelphia, PA, 4 ed., 2013, https://doi.org/10.1137/1.9781421407944.

[15] T. F. GONZALEZ, *Clustering to minimize the maximum intercluster distance*, Theoretical Computer Science, 38 (1985), pp. 293–306, https://doi.org/10.1016/0304-3975(85)90224-5.

[16] J. GUINNESS, *Permutation and grouping methods for sharpening gaussian process approximations*, Technometrics, 60 (2018), pp. 415–429.

[17] N. J. HIGHAM, *Analysis of the Cholesky decomposition of a semi-definite matrix*, in Reliable Numerical Computation, M. G. Cox and S. J. Hammarling, eds., Oxford University Press, 1990, pp. 161–185, https://eprints.maths.manchester.ac.uk/id/eprint/1193.

[18] S. HUAN, J. GUINNESS, M. KATZFUSS, H. OWHADI, AND F. SCHÄFER, *Sparse Cholesky factorization by greedy conditional selection*, 2023, https://arxiv.org/abs/2307.11648.

[19] I. E. KAPORIN, *New convergence results and preconditioning strategies for the conjugate gradient method*, Numerical Linear Algebra with Applications, 1 (1994), pp. 179–210, https://doi.org/https://doi.org/10.1002/nla.1680010208.

name	quantity	Frangella	RPC+D	RPC+V$\frac{1}{4}$	RPC+V$\frac{1}{2}$	CPC+D	FPS+D	GKL+D	SDS+D	CPC+V$\frac{1}{4}$	FPS+V$\frac{1}{4}$	GKL+V$\frac{1}{4}$	SDS+V$\frac{1}{4}$	Diaz
	direct	334.925	-6.6992e+04	-1.4427e+05	-1.6320e+05	-3.1831e+04	-3.9126e+04	-8.5191e+04	-6.7841e+04	-1.3516e+05	-1.3680e+05	-1.5152e+05	-1.4448e+05	-2.7399e+05
ACSIncome	mean	-3268.924	-7.1199e+04	-1.7354e+05	-1.9756e+05	-3.4612e+04	-4.2135e+04	-9.2591e+04	-7.2192e+04	-1.5944e+05	-1.6208e+05	-1.8419e+05	-1.7421e+05	-2.3247e+05
(Truth: -2.1403e+05)	stdev	317.645	184.908	436.648	444.975	248.157	183.212	270.306	299.461	317.979	458.527	357.319	479.340	948.544
	direct	369.589	-6.6740e+04	-1.3091e+05	-1.4844e+05	-4.7225e+04	-5.4099e+04	-7.7293e+04	-6.9605e+04	-1.2415e+05	-1.2628e+05	-1.3502e+05	-1.3220e+05	-2.7399e+05
Airlines_DepDelay_1M	mean	-2172.741	-6.9737e+04	-1.5572e+05	-1.8993e+05	-4.9436e+04	-5.6695e+04	-8.1512e+04	-7.2927e+04	-1.4345e+05	-1.4757e+05	-1.6295e+05	-1.5806e+05	-2.4262e+05
(Truth: -2.2360e+05)	stdev	128.847	177.521	530.649	760.602	224.668	270.464	257.910	174.131	227.897	395.783	511.491	495.315	902.888
	direct	547.855	-2.0346e+05	-2.6598e+05	-2.6867e+05	-1.0295e+05	-1.0845e+05	-2.5023e+05	-2.4246e+05	-2.6426e+05	-2.6493e+05	-2.6703e+05	-2.6631e+05	-2.7429e+05
COMET_MC_SAMPLE	mean	-8.3203e+04	-2.4517e+05	-2.7074e+05	-2.7102e+05	-1.0586e+05	-1.1242e+05	-2.5950e+05	-2.5237e+05	-2.7025e+05	-2.7056e+05	-2.7086e+05	-2.7076e+05	-2.2887e+05
(Truth: -2.7101e+05)	stdev	2.0279e+04	404.252	170.426	97.918	1205.659	1883.810	171.093	190.794	128.086	158.429	148.999	253.683	954.794
	direct	366.155	-8.8015e+04	-1.9584e+05	-2.1138e+05	-3.8238e+04	-4.0002e+04	-1.2584e+05	-1.0006e+05	-1.8499e+05	-1.8580e+05	-2.0221e+05	-1.9832e+05	-2.7405e+05
Click_prediction_small	mean	-3053.869	-9.0084e+04	-2.2101e+05	-2.3505e+05	-3.9592e+04	-4.1379e+04	-1.3771e+05	-1.0348e+05	-2.0764e+05	-2.0891e+05	-2.2752e+05	-2.2384e+05	-2.5100e+05
(Truth: -2.4120e+05)	stdev	278.677	165.415	317.801	539.764	127.587	74.074	265.943	183.803	276.564	357.236	384.792	350.907	590.354
	direct	307.249	-3.5030e+04	-5.8029e+04	-6.3897e+04	-1.4254e+04	-1.3959e+04	-3.9827e+04	-3.5727e+04	-5.1104e+04	-5.1096e+04	-5.9604e+04	-5.8374e+04	-2.7400e+05
HIGGS	mean	-3.2567e+04	-6.8550e+04	-9.3617e+04	-9.6118e+04	-4.3267e+04	-4.2890e+04	-7.7451e+04	-6.9525e+04	-8.7790e+04	-8.8048e+04	-9.4790e+04	-9.4025e+04	-1.0906e+05
(Truth: -9.6956e+04)	stdev	678.839	513.307	323.742	498.868	757.878	619.386	297.815	251.946	682.111	425.105	349.786	611.421	1692.906
	direct	334.968	-3.1735e+04	-5.2120e+04	-5.6860e+04	-1475.040	-3685.436	-3.7995e+04	-3.2518e+04	-4.8036e+04	-4.8033e+04	-5.4115e+04	-5.2449e+04	-2.7404e+05
MNIST	mean	-4.2797e+04	-6.4045e+04	-7.3831e+04	-7.4447e+04	-4.5515e+04	-4.5305e+04	-7.0893e+04	-6.4710e+04	-7.3053e+04	-7.3109e+04	-7.4354e+04	-7.3906e+04	-8.2306e+04
(Truth: -7.4651e+04)	stdev	391.498	457.497	308.236	260.081	683.829	833.240	504.378	506.625	254.668	301.245	409.419	319.472	2074.463
	direct	344.779	-4.8381e+04	-1.0381e+05	-1.1648e+05	-3.2677e+04	-3.7706e+04	-5.6973e+04	-4.8880e+04	-9.9462e+04	-1.0051e+05	-1.0676e+05	-1.0405e+05	-2.7397e+05
Medical-Appointment	mean	-7863.124	-5.8412e+04	-1.3389e+05	-1.5333e+05	-4.0463e+04	-4.5940e+04	-6.9628e+04	-5.8998e+04	-1.2652e+05	-1.2790e+05	-1.4007e+05	-1.3443e+05	-2.0437e+05
(Truth: -1.7067e+05)	stdev	347.785	324.926	298.101	649.913	284.663	156.877	268.785	323.524	622.214	369.176	397.115	343.315	913.384
	direct	436.213	-1.3317e+05	-1.7701e+05	-1.9326e+05	-8.6621e+04	-9.5533e+04	-1.5398e+05	-1.4180e+05	-1.6236e+05	-1.6352e+05	-1.8781e+05	-1.8183e+05	-2.7433e+05
MiniBooNE	mean	-1.0976e+04	-1.4178e+05	-2.1366e+05	-2.3582e+05	-8.8736e+04	-9.8078e+04	-1.7703e+05	-1.5448e+05	-1.9122e+05	-1.9266e+05	-2.2850e+05	-2.2110e+05	-2.5011e+05
(Truth: -2.5169e+05)	stdev	2885.110	303.743	375.604	634.117	319.867	274.811	348.341	458.840	471.799	328.082	487.580	682.160	418.726
	direct	300.172	-3.2956e+04	-4.8939e+04	-5.4265e+04	-1.2999e+04	-1.3009e+04	-4.3459e+04	-3.4469e+04	-4.1146e+04	-4.1148e+04	-5.4490e+04	-4.9791e+04	-2.7406e+05
YearPredictionMSD	mean	-3.0988e+04	-6.2187e+04	-7.6926e+04	-7.9996e+04	-4.1818e+04	-4.1986e+04	-7.5328e+04	-6.4631e+04	-7.1072e+04	-7.0857e+04	-8.0555e+04	-7.7224e+04	-8.9637e+04
(Truth: -8.1141e+04)	stdev	469.526	381.288	575.546	458.809	404.891	379.834	682.798	400.727	378.032	469.249	448.637	371.432	1555.579
	direct	390.780	-3.6299e+04	-7.1624e+04	-7.9463e+04	-1.0522e+04	-1.2427e+04	-4.4065e+04	-3.6688e+04	-6.5667e+04	-6.5888e+04	-7.4596e+04	-7.1920e+04	-2.7193e+05
a9a	mean	-2.2254e+04	-5.9859e+04	-1.0854e+05	-1.0412e+05	-3.1113e+04	-3.3183e+04	-7.3280e+04	-6.0063e+04	-9.3385e+04	-9.3594e+04	-1.0229e+05	-1.0089e+05	-1.3495e+05
(Truth: -1.0681e+05)	stdev	385.449	255.444	513.462	395.157	440.785	361.225	463.507	411.840	460.235	496.559	419.359	368.032	1883.872
	direct	397.433	-9.5782e+04	-1.7194e+05	-1.8733e+05	-4.9534e+04	-6.1418e+04	-1.2481e+05	-1.0575e+05	-1.5769e+05	-1.6039e+05	-1.8118e+05	-1.7598e+05	-2.2616e+05
cadata	mean	-1.8143e+04	-9.8206e+04	-1.9711e+05	-2.0810e+05	-5.4319e+04	-6.3254e+04	-1.3997e+05	-1.1411e+05	-1.8295e+05	-1.8618e+05	-2.0342e+05	-2.0006e+05	-2.1511e+05
(Truth: -2.1249e+05)	stdev	4174.340	234.610	567.249	323.322	825.211	336.002	303.583	485.779	387.897	361.191	362.794	280.979	277.213
	direct	412.913	-1.0154e+05	-2.0426e+05	-2.2305e+05	-5.7483e+04	-7.1388e+04	-1.3327e+05	-1.1230e+05	-1.9223e+05	-1.9539e+05	-2.1449e+05	-2.0823e+05	-2.7407e+05
cod-rna	mean	-2145.394	-1.0287e+05	-2.3169e+05	-2.4745e+05	-5.8141e+04	-7.1681e+04	-1.4665e+05	-1.1592e+05	-2.1770e+05	-2.2163e+05	-2.4070e+05	-2.3551e+05	-2.5973e+05
(Truth: -2.5420e+05)	stdev	429.133	209.086	359.058	420.114	52.705	96.386	300.256	235.355	367.632	452.731	349.753	407.336	219.823
	direct	436.867	-3.7352e+04	-7.4977e+04	-8.2746e+04	-2.3940e+04	-2.6807e+04	-4.2144e+04	-3.7768e+04	-7.0527e+04	-7.1249e+04	-7.6531e+04	-7.5026e+04	-2.7398e+05
connect-4	mean	-2.3699e+04	-6.4899e+04	-1.0610e+05	-1.0840e+05	-4.6986e+04	-5.0457e+04	-7.4927e+04	-6.5703e+04	-1.0101e+05	-1.0174e+05	-1.0698e+05	-1.0611e+05	-1.2762e+05
(Truth: -1.0908e+05)	stdev	568.906	421.356	400.485	369.317	503.563	497.894	452.002	295.002	578.151	367.587	597.873	345.889	1409.019
	direct	454.685	-7.5471e+04	-1.8300e+05	-2.0549e+05	-4.4821e+04	-5.6541e+04	-9.9056e+04	-7.9149e+04	-1.7718e+05	-1.7942e+05	-1.9256e+05	-1.8470e+05	-2.7400e+05
covtype_binary	mean	-1573.697	-7.7408e+04	-2.1320e+05	-2.2940e+05	-4.5849e+04	-5.7678e+04	-1.0744e+05	-8.1358e+04	-2.0682e+05	-2.0887e+05	-2.2227e+05	-2.1466e+05	-2.5115e+05
(Truth: -2.3453e+05)	stdev	145.385	266.982	444.006	292.190	165.233	236.740	318.370	202.985	294.547	437.749	385.139	411.452	746.069
	direct	341.659	-4.8310e+04	-1.1263e+05	-1.2348e+05	-5505.843	-8065.583	-6.6485e+04	-5.1014e+04	-1.0455e+05	-1.0472e+05	-1.1730e+05	-1.1328e+05	-2.7403e+05
creditcard	mean	-1.1073e+04	-6.0359e+04	-1.4129e+05	-1.5376e+05	-1.5881e+04	-1.7998e+04	-8.4080e+04	-6.3407e+04	-1.3001e+05	-1.3049e+05	-1.4893e+05	-1.4219e+05	-1.8841e+05
(Truth: -1.6062e+05)	stdev	434.569	403.245	311.751	410.784	332.249	270.735	393.787	312.231	619.893	524.833	422.812	490.900	993.545
	direct	421.273	-1.0306e+05	-2.0277e+05	-2.2146e+05	-5.4183e+04	-7.1190e+04	-1.2937e+05	-1.1286e+05	-1.9334e+05	-1.9592e+05	-2.0948e+05	-2.0520e+05	-2.7409e+05
diamonds	mean	-5727.127	-1.0447e+05	-2.3077e+05	-2.4728e+05	-5.4827e+04	-7.1434e+04	-1.4079e+05	-1.1623e+05	-2.1795e+05	-2.2186e+05	-2.3681e+05	-2.3335e+05	-2.5959e+05
(Truth: -2.5535e+05)	stdev	1516.542	135.963	521.442	349.360	74.216	143.469	406.337	203.733	279.037	416.968	369.196	529.177	519.439
	direct	459.655	-1.2060e+05	-1.9345e+05	-2.1402e+05	-8.5792e+04	-9.4616e+04	-1.4038e+05	-1.2498e+05	-1.8243e+05	-1.8454e+05	-2.0046e+05	-1.9544e+05	-2.7418e+05
hls4ml_lhc_jets_hlf	mean	-1.4217e+04	-1.2444e+05	-2.2878e+05	-2.4669e+05	-8.6584e+04	-9.5549e+04	-1.5501e+05	-1.3128e+05	-2.1627e+05	-2.1944e+05	-2.3418e+05	-2.3048e+05	-2.5693e+05
(Truth: -2.5519e+05)	stdev	5180.807	168.923	428.815	521.642	181.025	286.255	348.394	187.203	395.036	490.716	591.908	509.586	557.662
	direct	383.970	-6.3419e+04	-1.4837e+05	-1.6726e+05	-2.9363e+04	-3.6906e+04	-8.6161e+04	-6.5058e+04	-1.4354e+05	-1.4448e+05	-1.5183e+05	-1.4858e+05	-2.7398e+05
ijcnn1	mean	-4437.291	-6.8540e+04	-1.7942e+05	-1.9834e+05	-3.3060e+04	-4.0752e+04	-9.5012e+04	-7.0601e+04	-1.7317e+05	-1.7441e+05	-1.8380e+05	-1.7979e+05	-2.2905e+05
(Truth: -2.0690e+05)	stdev	275.867	279.797	311.374	542.992	242.672	198.186	240.138	226.032	547.534	508.348	341.781	429.496	987.895
	direct	296.325	-2.7786e+04	-4.1640e+04	-4.5929e+04	-1.5128e+04	-1.6156e+04	-3.1709e+04	-2.8248e+04	-3.6832e+04	-3.7273e+04	-4.2850e+04	-4.1805e+04	-2.7402e+05
jannis	mean	-5.2013e+04	-6.5125e+04	-6.8943e+04	-6.9009e+04	-5.7267e+04	-5.7942e+04	-6.7747e+04	-6.5652e+04	-6.8008e+04	-6.8185e+04	-6.8912e+04	-6.9032e+04	-7.0266e+04
(Truth: -6.9136e+04)	stdev	886.299	379.169	369.218	346.723	352.154	410.528	550.154	511.594	415.911	611.777	350.670	393.567	2208.360
	direct	151.154	-7546.272	-1.5399e+04	-1.6919e+04	-4644.935	-4884.715	-9031.648	-7203.401	-1.4539e+04	-1.4607e+04	-1.5948e+04	-1.5278e+04	-6.9404e+04
phishing	mean	-1.0225e+04	-1.4283e+04	-2.1815e+04	-2.1966e+04	-1.2081e+04	-1.2248e+04	-1.7629e+04	-1.3705e+04	-2.1023e+04	-2.1340e+04	-2.2070e+04	-2.1750e+04	-2.7897e+04
(Truth: -2.2161e+04)	stdev	271.929	165.892	236.294	124.941	173.470	161.635	324.359	221.586	168.240	235.960	270.588	270.082	1218.773
	direct	307.826	-1.3566e+04	-1.7202e+04	-1.9709e+04	-1.2955e+04	-1.2964e+04	-1.4444e+04	-1.3584e+04	-1.6714e+04	-1.6733e+04	-1.7858e+04	-1.7221e+04	-2.7397e+05
santander	mean	-3.0536e+04	-3.0530e+04	-3.0590e+04	-3.0602e+04	-3.0356e+04	-3.0665e+04	-3.0547e+04	-3.0496e+04	-3.0505e+04	-3.0482e+04	-3.0614e+04	-3.0596e+04	-2.9627e+04
(Truth: -3.0587e+04)	stdev	349.817	256.391	233.420	78.077	263.342	346.744	158.995	262.360	144.463	197.558	216.051	194.757	2077.336
	direct	303.107	-3.5026e+04	-6.6638e+04	-7.4849e+04	-1.6938e+04	-1.6880e+04	-5.8840e+04	-3.9104e+04	-6.0524e+04	-6.0504e+04	-7.4383e+04	-6.7209e+04	-2.7407e+05
sensit_vehicle	mean	-1.6869e+04	-5.0615e+04	-9.1377e+04	-1.0125e+05	-3.2075e+04	-3.2006e+04	-8.4627e+04	-5.5723e+04	-8.3499e+04	-8.3569e+04	-1.0121e+05	-9.2444e+04	-1.3117e+05
(Truth: -1.0693e+05)	stdev	648.447	381.913	466.330	408.605	493.197	433.066	453.620	424.658	326.877	519.073	388.681	209.741	1947.397
	direct	324.318	-5.8628e+04	-9.9759e+04	-1.1045e+05	-2.7787e+04	-3.2078e+04	-6.8436e+04	-6.0556e+04	-9.2545e+04	-9.3327e+04	-1.0204e+05	-1.0027e+05	-2.7402e+05
sensorless	mean	-1.4062e+04	-7.3828e+04	-1.3309e+05	-1.4804e+05	-3.8055e+04	-4.2556e+04	-8.7991e+04	-7.6170e+04	-1.2119e+05	-1.2270e+05	-1.3715e+05	-1.3381e+05	-1.9029e+05
(Truth: -1.6041e+05)	stdev	481.322	375.226	505.657	509.003	418.061	466.170	526.298	336.009	537.470	415.620	516.629	558.935	612.741
	direct	670.659	-1.3720e+05	-1.5112e+05	-1.5163e+05	-1.2492e+05	-1.3410e+05	-1.4324e+05	-1.3751e+05	-1.5108e+05	-1.5113e+05	-1.5116e+05	-1.5110e+05	-1.5242e+05
skin_nonskin	mean	-1.2153e+05	-1.5100e+05	-1.5209e+05	-1.5210e+05	-1.5121e+05	-1.5057e+05	-1.5094e+05	-1.4701e+05	-1.5208e+05	-1.5209e+05	-1.5210e+05	-1.5209e+05	-1.5188e+05
(Truth: -1.5209e+05)	stdev	3331.136	216.693	88.393	46.649	460.113	376.684	103.969	177.445	59.609	82.396	62.354	75.228	35.722
	direct	338.427	-4.2166e+04	-6.0891e+04	-6.7960e+04	-2.2879e+04	-2.4206e+04	-4.7951e+04	-4.3183e+04	-5.4971e+04	-5.5151e+04	-6.5374e+04	-6.1785e+04	-2.7402e+05
volkert	mean	-4.1148e+04	-7.8323e+04	-9.1063e+04	-9.4739e+04	-5.8289e+04	-5.9336e+04	-8.7125e+04	-7.9970e+04	-8.5530e+04	-8.5871e+04	-9.4870e+04	-9.1754e+04	-1.0588e+05
(Truth: -9.6886e+04)	stdev	579.315	507.691	341.673	484.464	591.008	309.115	416.266	333.149	185.948	607.340	288.294	575.722	2461.194
	direct	361.319	-2.6552e+04	-4.9498e+04	-5.4942e+04	-1.1777e+04	-1.1802e+04	-4.0392e+04	-2.6333e+04	-4.4252e+04	-4.4287e+04	-5.6812e+04	-4.9758e+04	-2.4603e+05
w8a	mean	-3.4601e+04	-5.0667e+04	-6.8623e+04	-7.0755e+04	-4.0437e+04	-3.9649e+04	-6.3680e+04	-5.0483e+04	-6.6417e+04	-6.6171e+04	-7.1034e+04	-6.8944e+04	-8.7256e+04
(Truth: -7.1276e+04)	stdev	774.386	368.411	373.839	138.973	557.760	897.720	389.723	326.609	343.511	511.357	192.012	270.281	1624.215
	direct	332.117	-3.0807e+04	-4.2182e+04	-4.6458e+04	-1038.983	-1223.322	-3.8153e+04	-3.0646e+04	-3.3514e+04	-3.3538e+04	-4.6017e+04	-4.2153e+04	-2.7404e+05
yolanda	mean	-3.5868e+04	-6.2223e+04	-6.8410e+04	-6.9708e+04	-3.6599e+04	-3.6674e+04	-6.8768e+04	-6.2522e+04	-6.3272e+04	-6.3059e+04	-6.9614e+04	-6.8360e+04	-7.2320e+04
(Truth: -6.9879e+04)	stdev	520.728	321.563	325.097	298.217	882.671	909.311	563.883	642.617	356.761	479.674	338.272	437.642	2335.977

TABLE 3

Log determinant estimation data for direct and indirect preconditioners on each dataset, for $\mu = 10^{-6}$

[20] L. Y. KOLOTILINA AND A. Y. YEREMIN, *Factorized sparse approximate inverse preconditionings i. theory*, SIAM Journal on Matrix Analysis and Applications, 14 (1993), pp. 45–58, https://doi.org/10.1137/0614004.

[21] K. MAKARYCHEV, A. REDDY, AND L. SHAN, *Improved guarantees for k-means++ and k-means++ parallel*, in Proceedings of the 34th International Conference on Neural Information Processing Systems, 2020, https://dl.acm.org/doi/10.5555/3495724.3497078.

[22] B. K. NATARAJAN, *Sparse approximate solutions to linear systems*, SIAM Journal on Computing, 24 (1995), pp. 227–234, https://doi.org/10.1137/S0097539792240406.

[23] F. SCHÄFER, M. KATZFUSS, AND H. OWHADI, *Sparse Cholesky factorization by Kullback–Leibler minimization*, SIAM Journal on Scientific Computing, 43 (2021), pp. A2019–A2046, https://doi.org/10.1137/20M1336254.

[24] B. SCHÖLKOPF AND A. J. SMOLA, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, The MIT Press, 2001, https://doi.org/10.7551/mitpress/4175.001.0001.

[25] Y. SUN AND M. L. STEIN, *Statistically and computationally efficient estimating equations for large spatial datasets*, Journal of Computational and Graphical Statistics, 25 (2016), pp. 187–208, https://doi.org/10.1080/10618600.2014.975230.

[26] K.-C. TOH AND L. N. TREFETHEN, *The Chebyshev polynomials of a matrix*, SIAM Journal

name	quantity	Frangella	RPC+D	RPC+V$\frac{1}{4}$	RPC+V$\frac{1}{2}$	CPC+D	FPS+D	GKL+D	SDS+D	CPC+V$\frac{1}{4}$	FPS+V$\frac{1}{4}$	GKL+V$\frac{1}{4}$	SDS+V$\frac{1}{4}$	Diaz
	direct	334.926	-6.6994e+04	-1.4523e+05	-1.6630e+05	-3.1831e+04	-3.9126e+04	-8.5102e+04	-6.7843e+04	-1.3577e+05	-1.3742e+05	-1.5314e+05	-1.4541e+05	-4.5689e+05
ACSIncome	mean	-3339.010	-7.1076e+04	-1.5813e+05	-1.8458e+05	-3.4848e+04	-4.2211e+04	-9.0301e+04	-7.2100e+04	-1.4658e+05	-1.4879e+05	-1.6818e+05	-1.5829e+05	-3.7556e+05
(Truth: -2.6234e+05)	stdev	177.103	225.613	334.239	370.668	315.055	226.360	221.193	161.919	291.461	418.628	402.563	285.111	1764.066
	direct	369.589	-6.6741e+04	-1.3696e+05	-1.4857e+05	-4.7225e+04	-5.4100e+04	-7.7299e+04	-6.9607e+04	-1.2418e+05	-1.2631e+05	-1.3508e+05	-1.3225e+05	-4.5690e+05
Airlines_DepDelay_1M	mean	-2219.980	-6.9684e+04	-1.4299e+05	-1.6670e+05	-4.9562e+04	-5.6529e+04	-8.1076e+04	-7.2810e+04	-1.3461e+05	-1.3728e+05	-1.4778e+05	-1.4445e+05	-3.9373e+05
(Truth: -2.5610e+05)	stdev	157.861	189.842	320.845	417.094	324.034	168.595	189.575	244.076	288.810	350.053	429.548	319.384	1040.703
	direct	547.858	-1.9591e+05	-4.2275e+05	-4.2967e+05	-1.0301e+05	-1.0851e+05	-3.9891e+05	-3.7083e+05	-4.1893e+05	-4.1994e+05	-4.0490e+05	-3.6210e+05	-4.5720e+05
COMET_MC_SAMPLE	mean	-8.8374e+04	-2.1578e+05	-4.2558e+05	-4.3465e+05	-1.6477e+05	-1.5557e+05	-4.0923e+05	-3.9051e+05	-4.2090e+05	-4.2212e+05	-4.1510e+05	-3.7666e+05	-4.4882e+05
(Truth: -4.4539e+05)	stdev	2.2368e+04	3445.151	188.708	326.777	1.0672e+04	9146.089	252.291	463.163	101.984	134.920	374.004	258.163	1453.342
	direct	366.155	-8.8884e+04	-2.1700e+05	-2.4397e+05	-3.8238e+04	-4.0002e+04	-1.2878e+05	-1.0010e+05	-2.0278e+05	-2.0377e+05	-2.2686e+05	-2.2049e+05	-4.5696e+05
Click_prediction_small	mean	-3235.334	-9.0198e+04	-2.2572e+05	-2.5900e+05	-3.9596e+04	-4.1348e+04	-1.3150e+05	-1.0154e+05	-2.0966e+05	-2.1066e+05	-2.3686e+05	-2.3007e+05	-4.1120e+05
(Truth: -3.3888e+05)	stdev	563.732	176.713	286.028	346.910	139.272	99.851	163.150	151.423	304.025	380.168	339.326	282.470	907.353
	direct	307.249	-3.5030e+04	-5.8030e+04	-6.3898e+04	-1.4254e+04	-1.3959e+04	-3.9828e+04	-3.5727e+04	-5.1104e+04	-5.1097e+04	-5.9605e+04	-5.8375e+04	-4.5691e+05
HIGGS	mean	-3.2232e+04	-6.8211e+04	-9.3872e+04	-9.6006e+04	-4.3265e+04	-4.2915e+04	-7.7337e+04	-6.9536e+04	-8.7908e+04	-8.7871e+04	-9.4975e+04	-9.3767e+04	-1.3850e+05
(Truth: -9.6970e+04)	stdev	502.085	535.646	301.267	226.850	559.836	605.775	510.127	567.065	224.794	460.271	478.466	697.309	2760.891
	direct	334.968	-3.1735e+04	-5.2121e+04	-5.6861e+04	-1475.069	-1685.466	-3.7995e+04	-3.2519e+04	-4.8036e+04	-4.8034e+04	-5.4116e+04	-5.2450e+04	-4.5694e+05
MNIST	mean	-4.2762e+04	-6.4101e+04	-7.3891e+04	-7.4531e+04	-4.5171e+04	-4.5515e+04	-7.0901e+04	-6.4913e+04	-7.2903e+04	-7.3068e+04	-7.4278e+04	-7.4009e+04	-9.6383e+04
(Truth: -7.4662e+04)	stdev	557.866	356.698	358.959	279.699	645.415	454.630	204.412	490.663	394.362	370.578	336.795	238.108	3240.331
	direct	344.779	-4.8381e+04	-1.0388e+05	-1.1662e+05	-3.2677e+04	-3.7707e+04	-5.6975e+04	-4.8880e+04	-9.9519e+04	-1.0057e+05	-1.0684e+05	-1.0412e+05	-4.5687e+05
Medical-Appointment	mean	-7746.806	-5.8306e+04	-1.3083e+05	-1.4936e+05	-4.0540e+04	-4.5905e+04	-6.9541e+04	-5.8959e+04	-1.2373e+05	-1.2550e+05	-1.3570e+05	-1.3100e+05	-3.1756e+05
(Truth: -1.7561e+05)	stdev	358.068	353.878	620.335	495.720	295.938	267.868	324.816	273.675	407.336	358.542	464.972	613.920	2255.523
	direct	436.215	-1.3320e+05	-1.7755e+05	-1.9434e+05	-8.6624e+04	-9.5537e+04	-1.5447e+05	-1.4190e+05	-1.6252e+05	-1.6369e+05	-1.8882e+05	-1.8234e+05	-4.5723e+05
MiniBooNE	mean	-9048.227	-1.3624e+05	-1.8619e+05	-2.0865e+05	-8.8510e+04	-9.7397e+04	-1.5905e+05	-1.4534e+05	-1.6853e+05	-1.6977e+05	-2.0059e+05	-1.9216e+05	-4.1286e+05
(Truth: -3.1142e+05)	stdev	1452.024	134.491	367.541	384.217	113.615	301.521	219.938	285.084	220.091	259.763	332.723	211.822	852.092
	direct	300.172	-3.2956e+04	-4.8943e+04	-5.4269e+04	-1.2999e+04	-1.3009e+04	-4.3460e+04	-3.4469e+04	-4.1149e+04	-4.1151e+04	-5.4494e+04	-4.9795e+04	-4.5697e+05
YearPredictionMSD	mean	-3.1253e+04	-6.2062e+04	-7.6970e+04	-7.9810e+04	-4.1773e+04	-4.1668e+04	-7.5364e+04	-6.4397e+04	-7.1107e+04	-7.1185e+04	-8.0519e+04	-7.7458e+04	-1.0941e+05
(Truth: -8.1160e+04)	stdev	480.687	578.636	360.989	353.342	570.798	472.019	259.345	324.944	353.551	315.151	397.085	256.223	3129.016
	direct	390.780	-3.6104e+04	-7.1521e+04	-7.9387e+04	-1.0522e+04	-1.2427e+04	-4.4065e+04	-3.6689e+04	-6.5671e+04	-6.5892e+04	-7.4602e+04	-7.1924e+04	-4.5344e+05
a9a	mean	-2.2477e+04	-5.9566e+04	-1.0027e+05	-1.0451e+05	-3.1107e+04	-3.2823e+04	-7.3046e+04	-6.0120e+04	-9.3371e+04	-9.3642e+04	-1.0268e+05	-1.0068e+05	-1.9266e+05
(Truth: -1.0701e+05)	stdev	409.895	484.529	519.729	258.743	458.462	413.731	464.814	193.123	485.078	602.851	265.019	403.412	3923.541
	direct	397.433	-9.4360e+04	-1.7490e+05	-1.9781e+05	-4.9535e+04	-6.1419e+04	-1.2621e+05	-1.0582e+05	-1.5929e+05	-1.6220e+05	-1.9140e+05	-1.8131e+05	-3.7706e+05
cadata	mean	-2.1145e+04	-9.4556e+04	-1.7684e+05	-2.0203e+05	-5.4495e+04	-6.3669e+04	-1.2655e+05	-1.0603e+05	-1.6058e+05	-1.6358e+05	-1.9500e+05	-1.8354e+05	-3.5621e+05
(Truth: -3.1817e+05)	stdev	6594.422	95.269	208.381	201.983	615.619	525.068	167.193	154.528	155.223	245.662	218.657	198.393	718.995
	direct	412.913	-9.8517e+04	-2.1388e+05	-2.4219e+05	-5.7484e+04	-7.1390e+04	-1.3440e+05	-1.1233e+05	-2.0075e+05	-2.0429e+05	-2.3121e+05	-2.2031e+05	-4.5697e+05
cod-rna	mean	-2170.166	-9.8774e+04	-2.1810e+05	-2.5110e+05	-5.8303e+04	-7.1719e+04	-1.3511e+05	-1.1273e+05	-2.0393e+05	-2.0775e+05	-2.3604e+05	-2.2506e+05	-4.2909e+05
(Truth: -3.7399e+05)	stdev	257.941	155.469	122.478	328.796	152.917	107.675	140.256	132.764	134.915	170.341	382.938	267.321	744.061
	direct	436.867	-3.7352e+04	-7.4978e+04	-8.2749e+04	-2.3940e+04	-2.6807e+04	-4.2145e+04	-3.7768e+04	-7.0529e+04	-7.1250e+04	-7.6533e+04	-7.5028e+04	-4.5689e+05
connect-4	mean	-2.3712e+04	-6.4730e+04	-1.0593e+05	-1.0839e+05	-4.6918e+04	-5.0381e+04	-7.4903e+04	-6.5520e+04	-1.0065e+05	-1.0168e+05	-1.0692e+05	-1.0597e+05	-1.6868e+05
(Truth: -1.0911e+05)	stdev	460.095	346.412	460.874	512.058	369.482	479.355	519.039	318.451	376.953	512.281	361.080	334.645	3180.749
	direct	454.685	-7.5474e+04	-1.8452e+05	-2.1173e+05	-4.4821e+04	-5.6541e+04	-9.9192e+04	-7.9153e+04	-1.7828e+05	-1.8066e+05	-1.9643e+05	-1.8642e+05	-4.5691e+05
covtype_binary	mean	-1515.947	-7.7275e+04	-1.9290e+05	-2.2375e+05	-4.5772e+04	-5.7766e+04	-1.0202e+05	-8.1048e+04	-1.8582e+05	-1.8852e+05	-2.0700e+05	-1.9478e+05	-4.1187e+05
(Truth: -3.0128e+05)	stdev	106.080	333.920	347.207	410.092	138.760	210.485	141.386	232.473	284.242	215.556	339.521	414.900	875.215
	direct	341.659	-4.8310e+04	-1.1903e+05	-1.3213e+05	-5505.874	-8065.619	-6.7241e+04	-5.1016e+04	-1.1077e+05	-1.1088e+05	-1.2440e+05	-1.1964e+05	-4.5693e+05
creditcard	mean	-1.1342e+04	-6.0106e+04	-1.4371e+05	-1.6122e+05	-1.6027e+04	-1.8136e+04	-8.2936e+04	-6.3388e+04	-1.3286e+05	-1.3315e+05	-1.4961e+05	-1.4473e+05	-2.9186e+05
(Truth: -1.7941e+05)	stdev	343.457	348.329	448.763	672.172	292.922	330.411	351.531	321.906	510.583	289.976	393.057	448.742	1990.670
	direct	421.274	-1.0300e+05	-2.1157e+05	-2.4149e+05	-5.4183e+04	-7.1191e+04	-1.2991e+05	-1.1290e+05	-2.0168e+05	-2.0439e+05	-2.2164e+05	-2.1453e+05	-4.5699e+05
diamonds	mean	-5266.104	-1.0321e+05	-2.1585e+05	-2.5042e+05	-5.4970e+04	-7.1427e+04	-1.3048e+05	-1.1329e+05	-2.0502e+05	-2.0816e+05	-2.2727e+05	-2.1907e+05	-4.2920e+05
(Truth: -3.7438e+05)	stdev	1752.397	97.815	237.677	281.260	141.189	80.403	138.377	114.769	232.663	249.523	239.662	237.197	574.169
	direct	459.656	-1.1882e+05	-1.9447e+05	-2.1939e+05	-8.5794e+04	-9.4620e+04	-1.4113e+05	-1.2505e+05	-1.8330e+05	-1.8545e+05	-2.0471e+05	-1.9761e+05	-4.5709e+05
hls4ml_lhc_jets_hlf	mean	-1.5747e+04	-1.1979e+05	-1.9946e+05	-2.2821e+05	-8.6371e+04	-9.5245e+04	-1.4243e+05	-1.2594e+05	-1.8728e+05	-1.8966e+05	-2.1089e+05	-2.0298e+05	-4.2478e+05
(Truth: -3.5574e+05)	stdev	5824.100	150.941	294.245	295.252	146.448	217.592	147.496	131.183	249.787	267.296	266.668	239.378	1066.903
	direct	383.971	-6.5152e+04	-1.4894e+05	-1.6942e+05	-2.9363e+04	-3.6906e+04	-8.6202e+04	-6.4701e+04	-1.4400e+05	-1.4496e+05	-1.5275e+05	-1.4926e+05	-4.5689e+05
ijcnn1	mean	-4422.527	-7.0536e+04	-1.6449e+05	-1.9035e+05	-3.3057e+04	-4.0633e+04	-9.3186e+04	-6.9805e+04	-1.5850e+05	-1.5976e+05	-1.6974e+05	-1.6510e+05	-3.6750e+05
(Truth: -2.4187e+05)	stdev	349.688	178.829	342.235	397.451	262.999	199.097	147.766	271.532	527.452	209.524	379.608	238.699	1854.956
	direct	296.325	-2.7786e+04	-4.1640e+04	-4.5930e+04	-1.5128e+04	-1.6157e+04	-3.1709e+04	-2.8248e+04	-3.6832e+04	-3.7273e+04	-4.2850e+04	-4.1805e+04	-4.5693e+05
jannis	mean	-5.2033e+04	-6.5210e+04	-6.8858e+04	-6.8964e+04	-5.7427e+04	-5.7999e+04	-6.7692e+04	-6.5661e+04	-6.8255e+04	-6.7953e+04	-6.9049e+04	-6.9016e+04	-7.0502e+04
(Truth: -6.9138e+04)	stdev	665.885	518.825	395.415	247.973	483.120	565.781	259.353	493.460	392.679	216.097	246.503	363.405	2999.995
	direct	151.154	-7546.314	-1.5399e+04	-1.6919e+04	-4644.952	-4884.733	-9031.730	-7203.437	-1.4539e+04	-1.4607e+04	-1.5949e+04	-1.5278e+04	-1.1578e+05
phishing	mean	-1.0322e+04	-1.4124e+04	-2.1811e+04	-2.2140e+04	-1.2047e+04	-1.2236e+04	-1.7579e+04	-1.3675e+04	-2.1162e+04	-2.1158e+04	-2.1934e+04	-2.1679e+04	-3.6837e+04
(Truth: -2.2171e+04)	stdev	231.541	238.905	200.222	250.855	234.340	233.184	271.942	323.619	169.899	213.493	142.976	234.861	2344.356
	direct	307.826	-1.3566e+04	-1.7202e+04	-1.9709e+04	-1.2955e+04	-1.2964e+04	-1.4444e+04	-1.3584e+04	-1.6714e+04	-1.6733e+04	-1.7858e+04	-1.7221e+04	-4.5688e+05
santander	mean	-3.0440e+04	-3.0576e+04	-3.0654e+04	-3.0551e+04	-3.0514e+04	-3.0704e+04	-3.0675e+04	-3.0533e+04	-3.0528e+04	-3.0502e+04	-3.0564e+04	-3.0591e+04	-2.8391e+04
(Truth: -3.0587e+04)	stdev	433.564	169.721	276.448	128.227	290.090	261.343	250.641	229.985	243.896	254.088	174.946	197.552	3990.427
	direct	303.107	-3.5026e+04	-6.6665e+04	-7.4920e+04	-1.6938e+04	-1.6880e+04	-5.8878e+04	-3.9105e+04	-6.0544e+04	-6.0525e+04	-7.4480e+04	-6.7239e+04	-4.5698e+05
sensit_vehicle	mean	-1.6931e+04	-5.0503e+04	-8.9697e+04	-9.9955e+04	-3.1806e+04	-3.1871e+04	-8.3055e+04	-5.5631e+04	-8.2297e+04	-8.2274e+04	-9.9841e+04	-9.0643e+04	-1.9737e+05
(Truth: -1.0905e+05)	stdev	720.674	234.577	551.212	359.062	312.934	553.820	363.404	507.369	229.431	277.419	443.732	359.519	4303.481
	direct	324.318	-5.8629e+04	-9.9769e+04	-1.1047e+05	-2.7787e+04	-3.2078e+04	-6.8439e+04	-6.0557e+04	-9.2551e+04	-9.3333e+04	-1.0205e+05	-1.0028e+05	-4.5693e+05
sensorless	mean	-1.4185e+04	-7.3525e+04	-1.3148e+05	-1.4621e+05	-3.7980e+04	-4.2453e+04	-8.7908e+04	-7.6230e+04	-1.2005e+05	-1.2151e+05	-1.3552e+05	-1.3238e+05	-2.8707e+05
(Truth: -1.6200e+05)	stdev	518.689	258.822	670.245	619.212	191.474	376.271	427.821	283.802	484.448	337.021	689.318	684.920	1861.642
	direct	671.056	-1.5940e+05	-2.3108e+05	-2.3197e+05	-1.2799e+05	-1.4601e+05	-2.0719e+05	-1.8567e+05	-2.4367e+05	-2.3394e+05	-2.0811e+05	-1.9043e+05	-2.5390e+05
skin_nonskin	mean	-1.2103e+05	-1.7151e+05	-2.3337e+05	-2.3362e+05	-2.1310e+05	-1.8955e+05	-2.1772e+05	-2.0079e+05	-2.4914e+05	-2.4071e+05	-2.1783e+05	-2.0295e+05	-2.2800e+05
(Truth: -2.5183e+05)	stdev	3247.826	3765.408	117.245	226.841	2038.242	6774.323	555.751	414.794	184.460	318.226	163.896	626.779	4170.811
	direct	338.427	-4.2167e+04	-6.0892e+04	-6.7961e+04	-2.2879e+04	-2.4206e+04	-4.7952e+04	-4.3184e+04	-5.4971e+04	-5.5151e+04	-6.5375e+04	-6.1786e+04	-4.5693e+05
volkert	mean	-4.0987e+04	-7.8266e+04	-9.1088e+04	-9.5084e+04	-5.8127e+04	-5.9161e+04	-8.7457e+04	-8.0018e+04	-8.5714e+04	-8.5752e+04	-9.4722e+04	-9.1562e+04	-1.2494e+05
(Truth: -9.6896e+04)	stdev	563.502	391.531	379.599	401.135	700.989	603.241	476.920	393.490	491.795	430.155	634.315	471.677	1534.453
	direct	361.320	-2.6552e+04	-4.9510e+04	-5.4966e+04	-1.1777e+04	-1.1803e+04	-4.0401e+04	-2.6333e+04	-4.4262e+04	-4.4297e+04	-5.6846e+04	-4.9767e+04	-4.1028e+05
w8a	mean	-3.4250e+04	-5.0760e+04	-6.8614e+04	-7.0929e+04	-4.0526e+04	-4.0468e+04	-6.3483e+04	-5.0441e+04	-6.6345e+04	-6.6174e+04	-7.1049e+04	-6.8909e+04	-1.1607e+05
(Truth: -7.1363e+04)	stdev	655.734	252.284	295.598	221.693	1352.892	590.025	494.250	462.422	348.973	320.349	347.014	390.710	1691.824
	direct	332.117	-3.0807e+04	-4.2183e+04	-4.6458e+04	-1039.005	-1223.344	-3.8153e+04	-3.0646e+04	-3.3514e+04	-3.3538e+04	-4.6017e+04	-4.2153e+04	-4.5695e+05
yolanda	mean	-3.5430e+04	-6.2058e+04	-6.8527e+04	-6.9528e+04	-3.6266e+04	-3.6470e+04	-6.8629e+04	-6.2204e+04	-6.3228e+04	-6.3035e+04	-6.9517e+04	-6.8353e+04	-7.9750e+04
(Truth: -6.9883e+04)	stdev	603.970	588.432	404.264	435.399	820.022	506.223	445.619	363.215	356.771	254.707	261.273	308.942	2929.062

TABLE 4

Log determinant estimation data for direct and indirect preconditioners on each dataset, for $\mu = 10^{-10}$

on Matrix Analysis and Applications, 20 (1998), pp. 400–419, https://doi.org/10.1137/S0895479896303739.

[27] J. TROPP, *Greed is good: Algorithmic results for sparse approximation*, IEEE Transactions on Information Theory, 50 (2004), pp. 2231–2242, https://doi.org/10.1109/TIT.2004.834793.

[28] S. UBARU, J. CHEN, AND Y. SAAD, *Fast estimation of $tr(f(a))$ via stochastic lanczos quadrature*, SIAM Journal on Matrix Analysis and Applications, 38 (2017), pp. 1075–1099, https://doi.org/10.1137/16M1104974.

[29] A. V. VECCHIA, *Estimation and model identification for continuous spatial processes*, Journal of the Royal Statistical Society Series B: Statistical Methodology, 50 (1988), pp. 297–312, https://doi.org/10.1111/j.2517-6161.1988.tb01729.x.

[30] A. Y. YEREMIN, L. Y. KOLOTILINA, AND A. NIKISHIN, *Factorized sparse approximate inverse preconditionings. III. Iterative construction of preconditioners*, Journal of Mathematical Sciences, 101 (2000), pp. 3237–3254.

[31] S. ZHAO, T. XU, H. HUANG, E. CHOW, AND Y. XI, *An adaptive factorized Nyström preconditioner for regularized kernel matrices*, SIAM Journal on Scientific Computing, 46 (2024), pp. A2351–A2376, https://doi.org/10.1137/23M1565139.

Appendix A. Proof of Kaporin optimality theory.

This section provides

a few proofs deferred from the main text.

A.1. Sparse approximation characterization. This section provides a linear algebraic characterization of the Vecchia approximation. We are not aware of this characterization appearing elsewhere.

PROPOSITION A.1 (Vecchia characterization). *For any positive-semidefinite matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$, the Vecchia approximation $\hat{\boldsymbol{A}} = \hat{\boldsymbol{C}}^{-1}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-*}$ with sparsity pattern $\{\mathsf{S}_i\}_{i=1}^n$ is characterized by*

$$\begin{cases} \boldsymbol{e}_i - \hat{\boldsymbol{C}}(i,\cdot)^* \in \operatorname*{argmin}_{\boldsymbol{v}\in\operatorname{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{S}_i}} d_{\boldsymbol{A}}(\boldsymbol{e}_i,\boldsymbol{v}), & 1\le i\le n,\\ \hat{\boldsymbol{D}}(i,i) = \min_{\boldsymbol{v}\in\operatorname{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{S}_i}} d_{\boldsymbol{A}}(\boldsymbol{e}_i,\boldsymbol{v}), & 1\le i\le n. \end{cases} \tag{A.1}$$

In contrast, the inverse Cholesky decomposition $\boldsymbol{A} = \boldsymbol{C}^{-1}\boldsymbol{D}\boldsymbol{C}^{-*}$ is characterized by

$$\begin{cases} \boldsymbol{e}_i - \boldsymbol{C}(i,\cdot)^* \in \operatorname*{argmin}_{\boldsymbol{v}\in\operatorname{span}\{\boldsymbol{e}_j\}_{j<i}} d_{\boldsymbol{A}}(\boldsymbol{e}_i,\boldsymbol{v}), & 1\le i\le n,\\ \boldsymbol{D}(i,i) = \min_{\boldsymbol{v}\in\operatorname{span}\{\boldsymbol{e}_j\}_{j<i}} d_{\boldsymbol{A}}(\boldsymbol{e}_i,\boldsymbol{v}), & 1\le i\le n. \end{cases} \tag{A.2}$$

Proof. For any $\mathsf{J}\subseteq\{1,\dots,i-1\}$ and $\boldsymbol{v}\in\operatorname{span}\{\boldsymbol{e}_j\}_{j\in\mathsf{J}}$, we can write

$$\begin{aligned} d_{\boldsymbol{A}}(\boldsymbol{e}_i,\boldsymbol{v})^2 &= \begin{bmatrix}-\boldsymbol{v}(\mathsf{J})\\ 1\end{bmatrix}^* \begin{bmatrix}\boldsymbol{A}(\mathsf{J},\mathsf{J}) & \boldsymbol{A}(\mathsf{J},i)\\ \boldsymbol{A}(i,\mathsf{J}) & \boldsymbol{A}(i,i)\end{bmatrix}\begin{bmatrix}-\boldsymbol{v}(\mathsf{J})\\ 1\end{bmatrix}\\ &= \boldsymbol{A}(i,i) - \boldsymbol{A}(i,\mathsf{J})\boldsymbol{A}(\mathsf{J},\mathsf{J})^+\boldsymbol{A}(\mathsf{J},i)\\ &\quad + \big[\boldsymbol{v}(\mathsf{S})^*\boldsymbol{A}(\mathsf{J},\mathsf{J}) - \boldsymbol{A}(i,\mathsf{J})\big]\boldsymbol{A}(\mathsf{J},\mathsf{J})^+\big[\boldsymbol{A}(\mathsf{J},\mathsf{J})\boldsymbol{v}(\mathsf{J}) - \boldsymbol{A}(\mathsf{J},i)\big]. \end{aligned}$$

Here we are using the fact that $\boldsymbol{A}(\mathsf{J},i)\in\operatorname{range}\boldsymbol{A}(\mathsf{J},\mathsf{J})$ and consequently

$$\boldsymbol{A}(\mathsf{J},\mathsf{J})\boldsymbol{A}(\mathsf{J},\mathsf{J})^+\boldsymbol{A}(\mathsf{J},i) = \boldsymbol{A}(\mathsf{J},i)$$

because $\boldsymbol{A}$ is positive-semidefinite. The minimizers of $d(\boldsymbol{e}_i,\boldsymbol{v})^2$ are characterized by

$$\boldsymbol{v}(\mathsf{J})^*\boldsymbol{A}(\mathsf{J},\mathsf{J}) = \boldsymbol{A}(i,\mathsf{J}), \tag{A.3}$$

and the minimum value of $d(\boldsymbol{e}_i,\boldsymbol{v})^2$ can be written using (A.3) as

$$\|\boldsymbol{e}_i - \boldsymbol{v}\|_{\boldsymbol{A}}^2 = \boldsymbol{A}(i,i) - \boldsymbol{v}(\mathsf{J})^*\boldsymbol{A}(\mathsf{J},i). \tag{A.4}$$

Thus, we have characterized the minimizers and minimum value of the distance.

Next we check that formulas (A.3) and (A.4) apply to the Vecchia approximation and inverse Cholesky decomposition. Indeed, the vector $\boldsymbol{v} = \boldsymbol{e}_i - \hat{\boldsymbol{C}}(i,\cdot)^*$ and diagonal entry $\hat{\boldsymbol{D}}(i,i)$ in the Vecchia approximation perfectly match the characterization (A.3)-(A.4) with $\mathsf{J} = \mathsf{S}_i$. Now observe the matrix $\boldsymbol{D}\boldsymbol{C}^{-*}$ in the inverse Cholesky decomposition is upper triangular, so

$$\begin{aligned} \boldsymbol{0} &= (\boldsymbol{D}\boldsymbol{C}^{-*})(i,1\!:\!i-1)\\ &= (\boldsymbol{C}\boldsymbol{A})(i,1\!:\!i-1)\\ &= \boldsymbol{C}(i,1\!:\!i-1)\boldsymbol{A}(1\!:\!i-1,1\!:\!i-1) + \boldsymbol{A}(i,1\!:\!i-1). \end{aligned} \tag{A.5}$$

Using (A.5), each diagonal entry satisfies

$$\boldsymbol{D}(i,i) = \boldsymbol{C}(i,\cdot)\boldsymbol{A}\boldsymbol{C}(i,\cdot)^* = \boldsymbol{A}(i,i) + \boldsymbol{C}(i,1\!:\!i-1)\boldsymbol{A}(1\!:\!i-1,i)$$

Hence, the vector $\boldsymbol{v} = \boldsymbol{e}_i - \boldsymbol{C}(i,\cdot)^*$ and diagonal entry $\boldsymbol{D}(i,i)$ in the inverse Cholesky decomposition match the characterization (A.3) and (A.4) with $\mathsf{J} = \{1,\dots,i-1\}$.

Last, if $\boldsymbol{D}$ and $\boldsymbol{C}$ are generated according to (A.2), then (A.3) showes that

$$\boldsymbol{C}(i,\cdot)\boldsymbol{A}(\cdot,1\!:\!i-1) = \boldsymbol{0}, \qquad \text{for each } i = 1,\dots,n,$$

so $\boldsymbol{CA}$ must be upper triangular. By multiplication with another upper triangular matrix, $\boldsymbol{CAC}^*$ is upper triangular also. By symmetry, $\boldsymbol{CAC}^*$ must be diagonal, and by (A.2) the diagonal entries are given by

$$\boldsymbol{D}(i,i) = \boldsymbol{C}(i,\cdot)\boldsymbol{AC}(i,\cdot)^*, \qquad \text{for each } i = 1,\dots,n,$$

We conclude that $\boldsymbol{A} = \boldsymbol{C}^{-*}\boldsymbol{D}\boldsymbol{C}^{-1}$ is an inverse Cholesky decomposition. ∎

Proposition A.1 gives a sparse variational characterization of the Vecchia approximation. Identifying each row $\boldsymbol{C}(i,\cdot)$ and diagonal entry $\boldsymbol{D}(i,i)$ in the inverse Cholesky decomposition via (A.2) is possible but expensive — it requires examining and processing a large submatrix $\boldsymbol{A}(1\!:\!i,1\!:\!i)$. It is cheaper to restrict the optimization to a sparsity set $\mathsf{S}_i \subseteq \{1,\dots,i-1\}$ so the solution only depends on a submatrix $\boldsymbol{A}(\{i\}\cup\mathsf{S}_i,\{i\}\cup\mathsf{S}_i)$. This sparse optimization results in the Vecchia approximation.

A.2. Volume and trace formulas. The inverse Cholesky decomposition $\boldsymbol{A} = \boldsymbol{C}^{-*}\boldsymbol{D}\boldsymbol{C}^{-1}$ leads to pseudoinverse and volume formulas given in the next lemma.

LEMMA A.2 (Pseudoinverse and volume formulas). *Fix a positive-semidefinite matrix with inverse Cholesky decomposition $\boldsymbol{A} = \boldsymbol{C}^{-*}\boldsymbol{D}\boldsymbol{C}^{-1}$. Let $\boldsymbol{Q}$ be an orthonormal basis for* range$(\boldsymbol{A})$ *and set $\boldsymbol{R} = \mathbf{I}(\cdot,\mathsf{N})$, where $\mathsf{N} \subseteq \{1,\dots,n\}$ picks out the nonzero elements of $\boldsymbol{D}$. Then*

$$\boldsymbol{A}^+ = \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{C}\boldsymbol{D}^+\boldsymbol{C}^*\boldsymbol{Q}\boldsymbol{Q}^* \quad \textit{and} \quad \operatorname{vol}(\boldsymbol{A}) = \frac{\operatorname{vol}(\boldsymbol{D})}{\det(\boldsymbol{R}^*\boldsymbol{Q})^2}. \tag{A.6}$$

Proof. The pseudoinverse $\boldsymbol{A}^+$ is characterized as the unique positive-semidefinite matrix that satisfies $\boldsymbol{A}^+\boldsymbol{x} = \boldsymbol{0}$ if $\boldsymbol{x} \perp \text{range}(\boldsymbol{Q})$ and $\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{x} = \boldsymbol{x}$ if $\boldsymbol{x} \in \text{range}(\boldsymbol{Q})$. We can check that these conditions are satisfied for $\boldsymbol{A}^+ = \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{C}\boldsymbol{D}^+\boldsymbol{C}^*\boldsymbol{Q}\boldsymbol{Q}^*$. If $\boldsymbol{x} \perp \text{range}(\boldsymbol{Q})$, we verify

$$\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{C}\boldsymbol{D}^+\boldsymbol{C}^*\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{x} = \boldsymbol{0}.$$

If $\boldsymbol{x} \in \text{range}(\boldsymbol{Q})$, then $\boldsymbol{x} = \boldsymbol{C}^{-*}\boldsymbol{D}\boldsymbol{y}$ for some $\boldsymbol{y} \in \mathbb{C}^n$ and we verify

$$\boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{C}\boldsymbol{D}^+\boldsymbol{C}^*\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{x} = \boldsymbol{C}^{-*}\boldsymbol{D}\boldsymbol{D}^+\boldsymbol{C}^*\boldsymbol{x} = \boldsymbol{C}^{-*}\boldsymbol{D}\boldsymbol{D}^+\boldsymbol{C}^*\boldsymbol{C}^{-*}\boldsymbol{D}\boldsymbol{y} = \boldsymbol{C}^{-*}\boldsymbol{D}\boldsymbol{y} = \boldsymbol{x}.$$

This confirms the pseudoinverse formula in (A.6).

Next observe that $\boldsymbol{R}^*\boldsymbol{C}^{-*}\boldsymbol{R}$ is an upper triangular matrix with ones on the diagonal; hence, it has determinant one. Using the definition of $\boldsymbol{Q}$ as an orthonormal basis for range$(\boldsymbol{A})$ = range$(\boldsymbol{C}^{-*}\boldsymbol{R})$, it follows

$$1 = \det\big(\boldsymbol{R}^*\boldsymbol{C}^{-*}\boldsymbol{R}\big) = \det\big(\boldsymbol{R}^*\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{C}^{-*}\boldsymbol{R}\big) = \det(\boldsymbol{R}^*\boldsymbol{Q})\det(\boldsymbol{Q}^*\boldsymbol{C}^{-*}\boldsymbol{R}).$$

The volume of $\boldsymbol{A}$ is given by

$$\begin{aligned}
&\operatorname{vol}(\boldsymbol{A}) = \operatorname{vol}\big(\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{C}^{-*}\boldsymbol{R}\boldsymbol{R}^*\boldsymbol{D}\boldsymbol{R}\boldsymbol{R}^*\boldsymbol{C}^{-1}\boldsymbol{Q}\boldsymbol{Q}^*\big) \\
&= \det\big(\boldsymbol{Q}^*\boldsymbol{C}^{-*}\boldsymbol{R}\boldsymbol{R}^*\boldsymbol{D}\boldsymbol{R}\boldsymbol{R}^*\boldsymbol{C}^{-1}\boldsymbol{Q}\big) = \det\big(\boldsymbol{Q}^*\boldsymbol{C}^{-*}\boldsymbol{R}\big)^2\det\big(\boldsymbol{R}^*\boldsymbol{D}\boldsymbol{R}\big) \\
&= \operatorname{vol}(\boldsymbol{D})/\det(\boldsymbol{R}^*\boldsymbol{Q})^2.
\end{aligned}$$

This confirms the volume formula in (A.6) and completes the proof. ∎

The next lemma gives explicit formulas for the trace and volume of $\boldsymbol{A}\hat{\boldsymbol{A}}^+$, where $\hat{\boldsymbol{A}}$ is an approximation with the same range as $\boldsymbol{A}$. In partial, when $\hat{\boldsymbol{A}}$ is the Vecchia approximation with the same range as $\boldsymbol{A}$, this guarantees $\operatorname{tr}\big(\hat{\boldsymbol{A}}^+\boldsymbol{A}\big) = \operatorname{rank}(\boldsymbol{A})$.

LEMMA A.3 (Trace and volume formulas). *When two inverse Cholesky decompositions $\boldsymbol{A} = \boldsymbol{C}^{-1}\boldsymbol{D}\boldsymbol{C}^{-*}$ and $\hat{\boldsymbol{A}} = \hat{\boldsymbol{C}}^{-1}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-*}$ share the same range, it follows*

$$\operatorname{vol}\big(\hat{\boldsymbol{A}}^+\boldsymbol{A}\big) = \frac{\operatorname{vol}\big(\boldsymbol{A}\big)}{\operatorname{vol}\big(\hat{\boldsymbol{A}}\big)} = \frac{\operatorname{vol}\big(\boldsymbol{D}\big)}{\operatorname{vol}\big(\hat{\boldsymbol{D}}\big)} \quad \textit{and} \quad \operatorname{tr}\big(\hat{\boldsymbol{A}}^+\boldsymbol{A}\big) = \operatorname{tr}\big(\hat{\boldsymbol{D}}^+\hat{\boldsymbol{C}}^*\boldsymbol{A}\hat{\boldsymbol{C}}\big).$$

Proof. First we argue that $\operatorname{range}(\hat{\boldsymbol{D}}) = \operatorname{range}(\boldsymbol{D})$. To that end, introduce the affine subspace

$$\mathsf{A}_i = \{(\boldsymbol{a}(1), \dots, \boldsymbol{a}(i-1), 1, 0, \dots, 0) \,|\, \boldsymbol{a} \in \mathbb{C}^{i-1}\}$$

for $i = 1, \dots, n$. Recall that $\boldsymbol{C}^{-*}$ is upper triangular, and fix an index $i \in \{1, \dots, n\}$ for which $\boldsymbol{D}(i,i) > 0$. Then the range of $\hat{\boldsymbol{A}}$ contains the vector $\boldsymbol{C}^{-*}(\cdot, i) \in \mathsf{A}_i$ and it is also spanned by vectors $\hat{\boldsymbol{C}}^{-*}(\cdot, j) \in \mathsf{A}_j$ for which $\hat{\boldsymbol{D}}(j,j) > 0$. Any set of vectors $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ with $\boldsymbol{x}_j \in \mathsf{A}_j$ is linearly independent, so it must be the case that $\hat{\boldsymbol{D}}(i,i) > 0$. This argument shows $\operatorname{range}(\boldsymbol{D}) \subseteq \operatorname{range}(\hat{\boldsymbol{D}})$ and reversing the argument shows $\operatorname{range}(\hat{\boldsymbol{D}}) \subseteq \operatorname{range}(\boldsymbol{D})$ also.

At this point, take an orthonormal basis $\boldsymbol{Q}$ for $\operatorname{range}\big(\hat{\boldsymbol{A}}\big) = \operatorname{range}(\boldsymbol{A})$ and an orthonormal basis $\boldsymbol{R}$ for $\operatorname{range}\big(\hat{\boldsymbol{D}}\big) = \operatorname{range}(\boldsymbol{D})$. To calculate $\operatorname{vol}\big(\hat{\boldsymbol{A}}^+\boldsymbol{A}\big)$, observe

$$\begin{aligned}
&\operatorname{vol}\big(\hat{\boldsymbol{A}}^+\boldsymbol{A}\big) = \operatorname{vol}\big(\boldsymbol{Q}\boldsymbol{Q}^*\hat{\boldsymbol{A}}^+\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*\big) \\
&= \det\big(\boldsymbol{Q}^*\hat{\boldsymbol{A}}^+\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\big) = \det\big(\boldsymbol{Q}^*\hat{\boldsymbol{A}}^+\boldsymbol{Q}\big)\det\big(\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\big) \\
&= \operatorname{vol}\big(\boldsymbol{Q}\boldsymbol{Q}^*\hat{\boldsymbol{A}}^+\boldsymbol{Q}\boldsymbol{Q}^*\big)\operatorname{vol}\big(\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*\big) = \operatorname{vol}\big(\hat{\boldsymbol{A}}^+\big)\operatorname{vol}\big(\boldsymbol{A}\big) = \frac{\operatorname{vol}\big(\boldsymbol{A}\big)}{\operatorname{vol}\big(\hat{\boldsymbol{A}}\big)}.
\end{aligned}$$

The last line uses the fact that $\operatorname{vol}\big(\hat{\boldsymbol{A}}^+\big)$ is the inverse of $\operatorname{vol}\big(\hat{\boldsymbol{A}}\big)$. Then apply the exact formula for $\operatorname{vol}\big(\boldsymbol{A}\big)$ and $\operatorname{vol}\big(\hat{\boldsymbol{A}}\big)$ given in Lemma A.2:

$$\frac{\operatorname{vol}\big(\boldsymbol{A}\big)}{\operatorname{vol}\big(\hat{\boldsymbol{A}}\big)} = \frac{\operatorname{vol}\big(\boldsymbol{D}\big)/\det(\boldsymbol{R}^*\boldsymbol{Q})^2}{\operatorname{vol}\big(\hat{\boldsymbol{D}}\big)/\det(\boldsymbol{R}^*\boldsymbol{Q})^2} = \frac{\operatorname{vol}\big(\boldsymbol{D}\big)}{\operatorname{vol}\big(\hat{\boldsymbol{D}}\big)}.$$

Similarly, apply the exact formula for the pseudoinverse $\hat{\boldsymbol{A}}$ in Lemma A.2 to calculate $\operatorname{tr}(\hat{\boldsymbol{A}}^+\boldsymbol{A})$ as follows:

$$\begin{aligned}
&\operatorname{tr}\big(\hat{\boldsymbol{A}}^+\boldsymbol{A}\big) = \operatorname{tr}\big(\boldsymbol{Q}\boldsymbol{Q}^*\hat{\boldsymbol{C}}\hat{\boldsymbol{D}}^+\hat{\boldsymbol{C}}^*\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\big) \\
&= \operatorname{tr}\big(\hat{\boldsymbol{D}}^+\hat{\boldsymbol{C}}^*\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}\boldsymbol{Q}\boldsymbol{Q}^*\hat{\boldsymbol{C}}\big) = \operatorname{tr}\big(\hat{\boldsymbol{D}}^+\hat{\boldsymbol{C}}^*\boldsymbol{A}\hat{\boldsymbol{C}}\big)
\end{aligned}$$

This completes the proof. ∎

A.3. Proof of Theorem 3.2. We introduce an inverse Cholesky decomposition $\boldsymbol{A} = \boldsymbol{C}^{-*}\boldsymbol{D}\boldsymbol{C}^{-1}$, and we will derive from first principles an inverse Cholesky approximation $\hat{\boldsymbol{A}} = \hat{\boldsymbol{C}}^{-*}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-1}$ that minimizes κ_{Kap}, assuming $\kappa_{\text{Kap}} < \infty$.

To that end, we partition the indices $\{1, \dots, n\}$ into a set of "good" indices $\mathsf{G} = \{i \,|\, \boldsymbol{D}(i,i) > 0\}$ and a set of "bad" indices $\mathsf{B} = \{i \,|\, \boldsymbol{D}(i,i) = 0\}$. The number of

good indices is $r = \operatorname{rank}(\boldsymbol{A})$ and the number of bad indices is $n - r$. We first consider what happens to the bad indices $i \in \mathsf{B}$. If $\boldsymbol{A}$ and $\hat{\boldsymbol{A}}$ have the same nullspace, we establish the following chain of implications:

(i) $\boldsymbol{D}(i,i) = 0$;
(ii) $\hat{\boldsymbol{D}}(i,i) = 0$;
(iii) $\boldsymbol{A}\hat{\boldsymbol{C}}(\cdot, i) = \boldsymbol{0}$.

We can check the implications as follows:

- (i) $\implies$ (ii). If $\boldsymbol{D}(i,i) = 0$, the vector $\boldsymbol{C}(\cdot, i)$ is in the shared nullspace of $\boldsymbol{A} = \boldsymbol{C}^{-*}\boldsymbol{D}\boldsymbol{C}^{-1}$ and $\hat{\boldsymbol{A}} = \hat{\boldsymbol{C}}^{-*}\hat{\boldsymbol{D}}\hat{\boldsymbol{C}}^{-1}$, so consequently the vector $\hat{\boldsymbol{C}}^{-1}\boldsymbol{C}(\cdot, i)$ is in the nullspace of $\hat{\boldsymbol{D}}$. Now recall that a lower triangular matrix with ones on the diagonal is a mapping from $\mathcal{V}_i$ into $\mathcal{V}_i$, where

$$\mathcal{V}_i = \{\boldsymbol{v} \in \mathbb{C}^n \,|\, \boldsymbol{v}(i) = 1, \boldsymbol{v}(j) = 0 \text{ for } j < i\}.$$

  Since $\boldsymbol{C}(\cdot, i) \in \mathcal{V}_i$ and $\hat{\boldsymbol{C}}^{-1}$ is lower triangular, it follows that $\hat{\boldsymbol{C}}^{-1}\boldsymbol{C}(\cdot, i) \in \mathcal{V}_i$. Since $\hat{\boldsymbol{C}}^{-1}\boldsymbol{C}(\cdot, i) \in \mathcal{V}_i$ lies in the nullspace of the nonnegative-valued diagonal matrix $\hat{\boldsymbol{D}}$, it follows that $\hat{\boldsymbol{D}}(i,i) = 0$.
- (ii) $\implies$ (iii). If $\hat{\boldsymbol{D}}(i,i) = 0$, the vector $\hat{\boldsymbol{C}}(\cdot, i)$ is in the shared nullspace of $\boldsymbol{A}$ and $\hat{\boldsymbol{A}}$.

Statements (ii) and (iii) imply that we must choose $\hat{\boldsymbol{C}}(\cdot, i)$ so that $\hat{\boldsymbol{C}}(\cdot, i)^* \boldsymbol{A}\hat{\boldsymbol{C}}(\cdot, i) = 0$ and we must set $\hat{\boldsymbol{D}}(i,i) = 0$. By the variational characterization in Proposition A.1, $\hat{\boldsymbol{C}}(\cdot, i)$ and $\hat{\boldsymbol{D}}(i,i)$ are constructed in exactly this way in the Vecchia approximation.

Now we focus on the good indices $i \in \mathsf{G}$. To that end, we rewrite κ_{Kap} using the volume and trace formulas from Lemma A.3 in the appendix.

$$\kappa_{\text{Kap}} = \frac{\left(\frac{1}{r}\operatorname{tr}\left(\boldsymbol{A}\hat{\boldsymbol{A}}^+\right)\right)^r}{\operatorname{vol}\left(\boldsymbol{A}\hat{\boldsymbol{A}}^+\right)} = \frac{\operatorname{vol}\left(\hat{\boldsymbol{D}}\right)\operatorname{tr}\left(\hat{\boldsymbol{D}}^+\hat{\boldsymbol{C}}^*\boldsymbol{A}\hat{\boldsymbol{C}}\right)^r}{r^r \operatorname{vol}\left(\boldsymbol{D}\right)}.$$

When we optimize $\hat{\boldsymbol{C}}(\cdot, i)$ to make $\left(\hat{\boldsymbol{C}}^*\boldsymbol{A}\hat{\boldsymbol{C}}\right)(i,i)$ as small as possible, Proposition A.1 shows that we are reproducing the column $\hat{\boldsymbol{C}}(\cdot, i)$ in the Vecchia approximation. Next consider the impact of varying $\hat{\boldsymbol{D}}(i,i)$ after $\hat{\boldsymbol{C}}(\cdot, i)$ is selected.

$$\partial_{\hat{\boldsymbol{D}}(i,i)} \log(\kappa_{\text{Kap}}) = \frac{1}{\hat{\boldsymbol{D}}(i,i)} - \frac{r\left(\hat{\boldsymbol{C}}^*\boldsymbol{A}\hat{\boldsymbol{C}}\right)(i,i)}{\hat{\boldsymbol{D}}(i,i)^2 \operatorname{tr}\left(\hat{\boldsymbol{D}}^+\hat{\boldsymbol{C}}^*\boldsymbol{A}\hat{\boldsymbol{C}}\right)}.$$

The logarithmic derivative is negative for small $\hat{\boldsymbol{D}}(i,i)$ and positive for large $\hat{\boldsymbol{D}}(i,i)$, and it achieves a zero value when $\hat{\boldsymbol{D}}(i,i) = \left(\hat{\boldsymbol{C}}^*\boldsymbol{A}\hat{\boldsymbol{C}}\right)(i,i)$. This matches the Vecchia definition as well, due to Proposition A.1.

With the Vecchia parameter settings, it follows that $\kappa_{\text{Kap}} = \operatorname{vol}(\hat{\boldsymbol{D}})/\operatorname{vol}(\boldsymbol{D})$. We can use the distance formulas in Proposition A.1 to rewrite κ_{Kap} as given in (3.1) and complete the proof.

Appendix B. Applications of the Kaporin condition number. The section proves several upper bounds for linear algebra calculations in terms of the Kaporin condition number.

B.1. Proof of Proposition 3.3. We can use the fact that $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_\star$ is orthogonal to range$(\boldsymbol{A})$ = range$(\hat{\boldsymbol{A}})$ to calculate

$$\begin{aligned}\|\hat{\boldsymbol{x}} - \boldsymbol{x}_\star\|_{\boldsymbol{A}} &= \|\boldsymbol{x}_0 + \hat{\boldsymbol{A}}^+[\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0] - \boldsymbol{x}_\star\|_{\boldsymbol{A}} \\ &= \big\|\big[\mathbf{I} - \hat{\boldsymbol{A}}^+\boldsymbol{A}\big]\big[\boldsymbol{x}_0 - \boldsymbol{x}_\star\big]\big\|_{\boldsymbol{A}} \\ &= \big\|\big[\mathbf{I} - \boldsymbol{A}^{1/2}\hat{\boldsymbol{A}}^+\boldsymbol{A}^{1/2}\big]\boldsymbol{A}^{1/2}\big[\boldsymbol{x}_0 - \boldsymbol{x}_\star\big]\big\| \\ &\le \|\mathbf{I} - \boldsymbol{A}^{1/2}\hat{\boldsymbol{A}}^+\boldsymbol{A}^{1/2}\|\|\hat{\boldsymbol{x}} - \boldsymbol{x}_0\|_{\boldsymbol{A}}\end{aligned}$$

The eigenvalues of $\boldsymbol{A}^{1/2}\hat{\boldsymbol{A}}^+\boldsymbol{A}^{1/2}$ are the eigenvalues of $\boldsymbol{A}\hat{\boldsymbol{A}}^+$, which we write as $\lambda_1 \ge \cdots \ge \lambda_r > 0$. Therefore,

$$\frac{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}{\|\hat{\boldsymbol{x}} - \boldsymbol{x}_0\|_{\boldsymbol{A}}} \le \|\mathbf{I} - \boldsymbol{A}^{1/2}\hat{\boldsymbol{A}}^+\boldsymbol{A}^{1/2}\| = \max_{1\le i\le r}|1 - \lambda_i|,$$

which gives a sharp upper bound.

By the concavity of $x \mapsto \log x$, $\sum_{j=1}^r \log(\lambda_j)$ is maximized when the eigenvalues λ_j for $j \ne i$ are equal. By the normalization $\sum_{j=1}^r \lambda_j = r$, we obtain

$$-\log(\kappa_{\text{Kap}}) = \sum_{j=1}^r \log(\lambda_j) \le \log(\lambda_i) + (r-1)\log\left(\frac{r-\lambda_i}{r-1}\right).$$

Over the interval $(0, r]$, we can bound $\log(x)$ from above by a concave quadratic that passes through $(1, 0)$ and $(r, \frac{r}{2} - \frac{1}{2r})$, so

$$\log(\lambda_i) \le -1 + \lambda_i - \frac{1}{2r}(1-\lambda_i)^2.$$

Also, $\log(1+x) \le x$ holds globally, so

$$(r-1)\log\left(\frac{r-\lambda_i}{r-1}\right) = (r-1)\log\left(1 + \frac{1-\lambda_i}{r-1}\right) \le 1 - \lambda_i.$$

It follows that

$$-\log(\kappa_{\text{Kap}}) \le \log(\lambda_i) + (r-1)\log\left(\frac{r-\lambda_i}{r-1}\right) \le -\frac{1}{2r}(1-\lambda_i)^2.$$

We conclude that $(1-\lambda_i)^2 \le 2r\log(\kappa_{\text{Kap}})$ for each $i = 1, \ldots, r$, which completes the proof.

B.2. Proof of Proposition 3.4. The starting point is a generic error bound for PCG iterates [3, eq. (3.4)].

$$\frac{\|\boldsymbol{x}_t - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_{\boldsymbol{A}}} \le \max_{1\le i\le r}|p_t(\lambda_i)|.$$

Here, p_t is any degree-t polynomial satisfying $p_t(0) = 1$, and $\lambda_1 \ge \cdots \ge \lambda_r$ are the sorted positive eigenvalues of $\boldsymbol{A}\hat{\boldsymbol{A}}^+$. In the case of even t, we construct

$$p_t(\lambda) = \prod_{i=1}^{t/2}\left(1 - \frac{\lambda}{\lambda_i}\right)\left(1 - \frac{\lambda}{\lambda_{r+1-i}}\right).$$

In the case of odd t, we set $t_\star = (t+1)/2$ and construct

$$p_t(\lambda) = \left(1 - \frac{2\lambda}{\lambda_{t_\star} + \lambda_{r+1-t_\star}}\right) \prod_{i=1}^{(t-1)/2} \left(1 - \frac{\lambda}{\lambda_i}\right)\left(1 - \frac{\lambda}{\lambda_{r+1-i}}\right).$$

Now observe that

$$\max_{1 \le i \le r} \big|p_t(\lambda_i)\big| = \max_{\lfloor t/2 \rfloor \le i \le r+1-\lfloor t/2 \rfloor} \big|p_t(\lambda_i)\big|,$$

and the right-hand side is bounded by the product of terms

$$\max_{\lambda_i \le \lambda \le \lambda_{r+1-i}} \left(\frac{\lambda}{\lambda_i} - 1\right)\left(1 - \frac{\lambda}{\lambda_{r+1-i}}\right) = \frac{(\lambda_i - \lambda_{r+1-i})^2}{4\lambda_i \lambda_{r+1-i}}$$

and potentially a term

$$\max_{\lambda_{t_\star} \le \lambda \le \lambda_{r+1-t_\star}} \left|1 - \frac{2\lambda}{\lambda_{t_\star} + \lambda_{r+1-t_\star}}\right| = \frac{\lambda_{t_\star} - \lambda_{r+1-t_\star}}{\lambda_{t_\star} + \lambda_{r+1-t_\star}} \le \frac{\lambda_{t_\star} - \lambda_{r+1-t_\star}}{2(\lambda_{t_\star}\lambda_{r+1-t_\star})^{1/2}},$$

where the last line follows from the arithmetic-geometric mean inequality

$$(\lambda_{t_\star}\lambda_{r+1-t_\star})^{1/2} \le \frac{\lambda_{t_\star} + \lambda_{r+1-t_\star}}{2}.$$

In the case of even t, we can use the arithmetic-geometric mean inequality twice to write

$$\begin{aligned}
&\prod_{i=1}^{t/2}\left(\frac{4\lambda_i\lambda_{r+1-i}}{(\lambda_i + \lambda_{r+1-i})^2}\right)^{2/t} + \prod_{i=1}^{t/2}\left(\frac{(\lambda_i - \lambda_{r+1-i})^2}{(\lambda_i + \lambda_{r+1-i})^2}\right)^{2/t} \\
&\le \frac{2}{t}\sum_{i=1}^{t/2} \frac{4\lambda_i\lambda_{r+1-i} + (\lambda_i - \lambda_{r+1-i})^2}{(\lambda_i + \lambda_{r+1-i})^2} = 1,
\end{aligned}$$

and consequently

$$1 + \left[\frac{\|\boldsymbol{x}_t - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}\right]^{2/t} \le 1 + \prod_{i=1}^{t/2}\left(\frac{(\lambda_i - \lambda_{r+1-i})^2}{4\lambda_i\lambda_{r+1-i}}\right)^{2/t} \le \prod_{i=1}^{t/2}\left(\frac{(\lambda_i + \lambda_{r+1-i})^2}{4\lambda_i\lambda_{r+1-i}}\right)^{2/t}.$$

In the case of odd t, we can use the generalized arithmetic-geometric mean inequality with weights $1/t, 2/t, \ldots, 2/t$ twice to write

$$\begin{aligned}
&\left(\frac{4\lambda_{t_\star}\lambda_{r+1-t_\star}}{(\lambda_{t_\star} + \lambda_{r+1-t_\star})^2}\right)^{1/t} \prod_{i=1}^{(t-1)/2}\left(\frac{4\lambda_i\lambda_{r+1-i}}{(\lambda_i + \lambda_{r+1-i})^2}\right)^{2/t} \\
&\quad + \left(\frac{(\lambda_{t_\star} - \lambda_{r+1-t_\star})^2}{(\lambda_{t_\star} + \lambda_{r+1-t_\star})^2}\right)^{1/t} \prod_{i=1}^{(t-1)/2}\left(\frac{(\lambda_i - \lambda_{r+1-i})^2}{(\lambda_i + \lambda_{r+1-i})^2}\right)^{2/t} \\
&\le \frac{1}{t}\frac{4\lambda_{t_\star}\lambda_{r+1-t_\star} + (\lambda_{t_\star} - \lambda_{r+1-t_\star})^2}{(\lambda_{t_\star} + \lambda_{r+1-t_\star})^2} + \frac{2}{t}\sum_{i=1}^{(t-1)/2} \frac{4\lambda_i\lambda_{r+1-i} + (\lambda_i - \lambda_{r+1-i})^2}{(\lambda_i + \lambda_{r+1-i})^2} = 1,
\end{aligned}$$

and consequently

$$\begin{aligned}
&1 + \left[\frac{\|\boldsymbol{x}_t - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}\right]^{2/t} \\
&\le 1 + \left(\frac{(\lambda_{t_\star} - \lambda_{r+1-t_\star})^2}{4\lambda_{t_\star}\lambda_{r+1-t_\star}}\right)^{1/t} \prod_{i=1}^{(t-1)/2} \left(\frac{(\lambda_i - \lambda_{r+1-i})^2}{4\lambda_i\lambda_{r+1-i}}\right)^{2/t} \\
&\le \left(\frac{(\lambda_{t_\star} + \lambda_{r+1-t_\star})^2}{4\lambda_{t_\star}\lambda_{r+1-t_\star}}\right)^{1/t} \prod_{i=1}^{t/2} \left(\frac{(\lambda_i + \lambda_{r+1-i})^2}{4\lambda_i\lambda_{r+1-i}}\right)^{2/t}.
\end{aligned}$$

Last, when r is even, we can use the arithmetic-geometric mean inequality two more times to write

$$\left(1 + \left[\frac{\|\boldsymbol{x}_t - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}\right]^{2/t}\right)^{t/2} \le \frac{1}{\prod_{i=1}^r \lambda_i} \prod_{i=1}^{r/2} \left(\frac{\lambda_i + \lambda_{r+1-i}}{2}\right)^2 \le \kappa_{\text{Kap}}.$$

Similarly when r is odd, we can use the generalized arithmetic-geometric mean inequality two more times to obtain

$$\left(1 + \left[\frac{\|\boldsymbol{x}_t - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}\right]^{2/t}\right)^{t/2} \le \frac{1}{\prod_{i=1}^r \lambda_i} \lambda_{(r+1)/2} \prod_{i=1}^{(r-1)/2} \left(\frac{\lambda_i + \lambda_{r+1-i}}{2}\right)^2 \le \kappa_{\text{Kap}}.$$

We conclude by recalling that

$$\begin{aligned}
2x &= \int_0^x 2\mathrm{d}y \le \int_0^x \frac{2\mathrm{d}y}{1-y^2} = \int_0^x \left[\frac{1}{1-y} + \frac{1}{1+y}\right]\mathrm{d}y \\
&= \Big[-\log(1-y) + \log(1+y)\Big]_{y=0}^{y=x} = \log\left(\frac{1+x}{1-x}\right)
\end{aligned}$$

and therefore $\mathrm{e}^{2x} - 1 \le \frac{2x}{1-x}$. Therefore, we make the calculation

$$\begin{aligned}
\frac{\|\boldsymbol{x}_t - \boldsymbol{x}_\star\|_{\boldsymbol{A}}}{\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_{\boldsymbol{A}}} &\le \left(\kappa_{\text{Kap}}^{2/t} - 1\right)^{t/2} \le \left(\frac{2\log(\kappa_{\text{Kap}})/t}{1 - \log(\kappa_{\text{Kap}})/t}\right)^{t/2} \\
&= \left(\frac{2\log(\kappa_{\text{Kap}})}{2t/3 + [t/3 - \log(\kappa_{\text{Kap}})]}\right)^{t/2} \le \left(\frac{3\log(\kappa_{\text{Kap}})}{t}\right)^{t/2}
\end{aligned}$$

on the event that $t/3 \ge \log(\kappa_{\text{Kap}})$. On the other hand, if $t/3 < \log(\kappa_{\text{Kap}})$, the result (3.2) holds vacuously. This completes the proof.

B.3. Proof of Proposition 3.6. Consider the matrix $\boldsymbol{M} = \hat{\boldsymbol{A}}^{-1/2}\boldsymbol{A}\hat{\boldsymbol{A}}^{-1/2}$ which has eigendecomposition

$$\boldsymbol{M} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^* \quad \text{where} \quad \boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n).$$

We need to prove the mean square error bound

$$\mathbb{E}\left|\operatorname{tr}(\log \boldsymbol{M}) - \frac{1}{t}\sum_{i=1}^t \boldsymbol{z}_i^*(\log \boldsymbol{M})\boldsymbol{z}_i\right|^2 \le \frac{4\log(\kappa_{\text{Kap}})}{t}.$$

To that end, we introduce a complex Gaussian vector $\boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and set $\boldsymbol{z} = \sqrt{n}\,\boldsymbol{\omega}/\|\boldsymbol{\omega}\|$. It suffices to show

$$\mathbb{E}[\boldsymbol{z}^*(\log \boldsymbol{M})\boldsymbol{z}] = \operatorname{tr}(\log \boldsymbol{M}) \quad \text{and} \quad \operatorname{Var}[\boldsymbol{z}^*(\log \boldsymbol{M})\boldsymbol{z}] \le 4 \log(\kappa_{\mathrm{Kap}}) \tag{B.1}$$

when the Kaporin condition number is $\log(\kappa_{\mathrm{Kap}}) \le n$.

We first observe that rotational invariance implies $\boldsymbol{z} \overset{\mathcal{D}}{=} \boldsymbol{Q}^*\boldsymbol{z}$ so

$$\boldsymbol{z}^*(\log \boldsymbol{M})\boldsymbol{z} \overset{\mathcal{D}}{=} \boldsymbol{z}^*(\log \boldsymbol{\Lambda})\boldsymbol{z} = \sum_{i=1}^n \log(\lambda_i)|\boldsymbol{z}(i)|^2.$$

Then we check the stochastic trace estimator is unbiased.

$$\mathbb{E}\left[\sum_{i=1}^n \log(\lambda_i)|\boldsymbol{z}(i)|^2\right] = \sum_{i=1}^n \log(\lambda_i)\,\mathbb{E}|\boldsymbol{z}(i)|^2 = \sum_{i=1}^n \log(\lambda_i) = \operatorname{tr}(\log \boldsymbol{M}). \tag{B.2}$$

This confirms the first part of (B.2).

To bound the variance, we make a calculation using the Gaussian vector $\boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

$$\begin{aligned}
&\mathbb{E}\left|\sum_{i=1}^n \log(\lambda_i)|\boldsymbol{\omega}(i)|^2\right|^2 - \left|\sum_{i=1}^n \log(\lambda_i)\right|^2 = \operatorname{Var}\left[\sum_{i=1}^n \log(\lambda_i)|\boldsymbol{\omega}(i)|^2\right] \\
&= \sum_{i=1}^n (\log \lambda_i)^2 \operatorname{Var}\big[|\boldsymbol{\omega}(i)|^2\big] = \sum_{i=1}^n (\log \lambda_i)^2.
\end{aligned}$$

Here we have used the independence of $\boldsymbol{\omega}(i)$ variables and the identity $\operatorname{Var}\big[|\boldsymbol{\omega}(i)|^2\big] = 1$, which holds for complex Gaussians. Since $\boldsymbol{\omega}$ has independent length and direction, we calculate

$$\mathbb{E}\left|\sum_{i=1}^n \log(\lambda_i)|\boldsymbol{\omega}(i)|^2\right|^2 = \frac{\mathbb{E}\|\boldsymbol{\omega}\|^4}{n^2}\,\mathbb{E}\left|\sum_{i=1}^n \log(\lambda_i)|\boldsymbol{z}(i)|^2\right|^2 = \frac{n^2+n}{n^2}\,\mathbb{E}\left|\sum_{i=1}^n \log(\lambda_i)|\boldsymbol{z}(i)|^2\right|^2,$$

where again we have used the fact that $\operatorname{Var}\big[|\boldsymbol{\omega}(i)|^2\big] = 1$. By rearrangement, it follows

$$\mathbb{E}\left|\sum_{i=1}^n \log(\lambda_i)|\boldsymbol{z}(i)|^2\right|^2 = \frac{n}{n+1}\left[\sum_{i=1}^n (\log \lambda_i)^2 + \left(\sum_{i=1}^n \log(\lambda_i)\right)^2\right].$$

Subtracting the square mean (B.2) shows that

$$\begin{aligned}
&\operatorname{Var}\left[\sum_{i=1}^n \log(\lambda_i)|\boldsymbol{z}(i)|^2\right] \\
&\le \frac{n}{n+1}\left[\frac{1}{n}\sum_{i=1}^n (\log \lambda_i)^2 - \left(\sum_{i=1}^n \log(\lambda_i)\right)^2\right] = \frac{1}{n+1}\sum_{i=1}^n (\log \lambda_i')^2,
\end{aligned}$$

where we have introduced $\lambda_i' = \lambda_i / \prod_{j=1}^n \lambda_j$ and we observe that $\prod_{i=1}^n \log(\lambda_i') = 0$ and $\frac{1}{n}\sum_{i=1}^n \lambda_i' = \kappa_{\mathrm{Kap}}^{1/n}$. Then recall the standard identity that

$$1 + t + \frac{1}{2}t^2 \le \mathrm{e}^t \quad \text{and therefore} \quad (\log u)^2 \le 2(u - 1 - \log u).$$

We obtain the variance bound

$$\operatorname{Var}\left[\sum_{i=1}^{n} \log(\lambda_i)|\boldsymbol{z}(i)|^2\right] \leq \frac{2n}{n+1}\sum_{i=1}^{n}\big(\lambda_i' - 1 - \log\lambda_i'\big) = \frac{2n^2}{n+1}\big(\kappa_{\mathrm{Kap}}^{1/n} - 1\big).$$

Last observe that $t \mapsto (\mathrm{e}^t - 1)/t$ is strictly increasing for $t > 0$ and therefore

$$\frac{\kappa_{\mathrm{Kap}}^{1/n} - 1}{\log(\kappa_{\mathrm{Kap}})/n} \leq 2 \quad \text{if} \quad \log(\kappa_{\mathrm{Kap}}) \leq 1.42n.$$

This completes the proof.