Randomly Pivoted Cholesky: Analysis and Applications

Aryan Jain* Siddarth Menon† Aditya Ramabadran‡

Fall 2023

Abstract

This project will concern the *randomly pivoted Cholesky* algorithm (RPCHOLESKY), a randomized algorithm (based on the classical Cholesky decomposition) for computing low rank approximations for large positive semidefinite matrices. We will explore this algorithm from three points of view: theoretical (explaining the algorithm and the proofs of its theoretical bounds), experimental (performing numerical/empirical experiments to assess its performance), and practice (explaining its applicationa and performing experiments to show its use in a real-world problem).

First, we will introduce and motivate this algorithm as part of a family of column Nyström approximation algorithms. Next, we will discuss the theoretical error bounds for the implementation of RPCHOLESKY as presented in [2]. Once we've seen how Chen et al. prove these bounds, we will implement our own naive and efficient implementations of RPCHOLESKY, and test them on moderately sized matrices. Using our implementations, we will empirically compare how well RPCHOLESKY performs (in relative trace norm error) compared to other methods such as uniform pivot sampling, and the best rank-k approximation (truncated eigendecomposition) of the matrix.

We then discuss applications of this randomized procedure in scientific computing contexts, starting with a brief discussion of [4]. This generalizes the procedures above from finite kernel matrices to continuous setting of reproducing kernel Hilbert spaces, towards providing a scheme for *kernel quadrature*, a fundamental problem in machine learning and statistics.

Finally, we discuss the kernel ridge regression problem, and how it can be optimized using RPCHOLESKY . We provide an implementation of using RPCHOLESKY for kernel ridge regression in the real-world problem of handwritten digit classification (using the popular MNIST dataset [5]).

Contents

*aryanjain@berkeley.edu
†sidmenon@berkeley.edu
‡aditya.ramabadran@berkeley.edu

1 Introduction

Positive semidefinite matrices are ubiquitous and are found in a variety of fields, including convex optimization, high-dimensional statistics, quantum chemistry and numerical simulations. Our primary motivation for finding efficient schemes for such approximations are *kernel methods* in machine learning, which often involve large $N \times N$ kernel matrices measuring similarity between data points.

Given an $N \times N$ positive semidefinite matrix $\mathbf{A}$, we want to compute a rank k-approximation $\widehat{\mathbf{A}}$. We will work in the setting of *column Nyström approximations*. We choose a subset of k indices $\mathcal{S} \subseteq \{1, 2, \ldots, N\}$ of indices, and we consider the matrix

$$\widehat{\mathbf{A}} = \mathbf{A}(:, \mathcal{S})\ \mathbf{A}(\mathcal{S}, \mathcal{S})^{\dagger}\ \mathbf{A}(\mathcal{S}, :)$$

where $\mathbf{A}(:, \mathcal{S})$ is the $N \times k$ submatrix of columns of $\mathbf{A}$ from $\mathcal{S}$, $\mathbf{A}(\mathcal{S}, \mathcal{S})^{\dagger}$ is the Moore-Penrose pseudoinverse of the $k \times k$ submatrix where we take rows and columns from $\mathcal{S}$, and $\mathbf{A}(\mathcal{S}, :)$ is the $N \times k$ submatrix of columns of $\mathbf{A}$ from $\mathcal{S}$. $\widehat{\mathbf{A}}$ admits the following nice properties:

1. it's the best approximation for which $\text{range}(\widehat{\mathbf{A}}) \subseteq \text{range}(\mathbf{A}(:, \mathcal{S}))$.
2. the residual $\mathbf{A} - \widehat{\mathbf{A}}$ is also positive semidefinite.

The implementation details of this family of algorithms all depend on the choice of subset $\mathcal{S}$. The authors of [2] present RPCHOLESKY as a simple and efficient randomized algorithm for choosing this subset $\mathcal{S}$. Each distinguished index s_i is sampled as follows

$$\Pr[s_i = j] = \frac{a_{jj}^{(i-1)}}{\operatorname{tr} \mathbf{A}^{(t-1)}}$$

thus each pivot is sampled proportional to the diagonal of the residual matrix at that step.

2 Theoretical Error Bounds

As stated in the introduction, we present the mathematical error bounds for RPCHOLESKY , towards determining which theoretical properties distinguish it from other column Nystroöm low rank approximations, and providing a benchmark against which we may test empirical results of our own implementations.

For the remainder of this section, we will denote the error of $\widehat{\mathbf{A}}$ approximating $\mathbf{A}$ by $\operatorname{tr}(\mathbf{A} - \widehat{\mathbf{A}})$ (the trace norm of the residual). In general we benchmark our low rank approximation against the best rank r, positive semidefinite approximation, given by the r-truncated eigenvalue decomposition, which we denote as in [2], by $[\![\mathbf{A}]\!]_r$.

Definition 2.1. For $r \in \mathbb{N}$ and $\varepsilon > 0$, we say a randomized Nyström approximation $\widehat{\mathbf{A}}$ is an $(r, \varepsilon)-$**approximation** of $\mathbf{A}$ if

$$\mathbf{E}\left[\operatorname{tr}(\mathbf{A} - \widehat{\mathbf{A}})\right] \leq (1 + \varepsilon) \cdot \operatorname{tr}\left(\mathbf{A} - [\![\mathbf{A}]\!]_r\right). \tag{1}$$

Here, the expectation averages over randomly sampled columns in the approximation $\widehat{\mathbf{A}}$.

From this definition, it's clear that the efficiency of some approximation scheme is dependent on how many columns we take to form the approximation (we will denote the number of accessed columns by k).

2.1 Main theorem

The one of the main novelties in Chen, Epperly, Tropp, and Webber's paper is the error bound for RPCHOLESKY , as it demonstrates near-optimality in the regime of column Nyström approximation schemes. Broadly speaking, the main theorem tells us how many columns we must sample before guaranteeing an $(r, \varepsilon)-$approximation.

Theorem 2.2. *Fix $r \in \mathbb{N}$ and $\varepsilon > 0$, and let $\mathbf{A}$ be a positive, semidefinite matrix. Then the approximation given by RPCHOLESKY obtained by accessing k columns is an $(r, \varepsilon)-$approximation of $\mathbf{A}$ provided that*

$$k \geq \frac{r}{\varepsilon} + \min\left\{ r \log_+\left(\frac{1}{\varepsilon\eta}\right), r + r\log_+\left(\frac{2^r}{\varepsilon}\right)\right\}.$$

Here, η is the relative error, given by $\eta := \frac{\operatorname{tr}(\mathbf{A}-[\![\mathbf{A}]\!]_r)}{\operatorname{tr}(\mathbf{A})}$, and $\log_+(x) = \max\{\log(x), 0\}$.

To understand how one should interpret the logarithmic term, there are two primary consequences (one for each regime).

1. In the first bound, we note that the $\log\left(\frac{1}{\eta}\right)$ term distinguishing algorithms like RPCHOLESKY from other column Nyström algorithms based on things like determinental point processes (DPPs), is effectively negligible in practice. If $\eta > 10^{-16}$, the machine precision for double-based arithmetic, then this factor is < 37 anyways. Thus, this algorithm is handedly better than uniformly sampling the diagonal, as well as greedily choosing the columns.
2. In the second regime, $r + r\log_+\left(\frac{2^r}{\varepsilon}\right)$ eliminates the dependence on η entirely, and when r is manageably small (so that the dependence on r^2 is not too significant), RPCHOLESKY is actually competitive with DPP based methods.

Also as a remark, if $\mathbf{A}$ is rank deficient (i.e. $\operatorname{rank}(\mathbf{A}) \leq r$), then we certainly have that $\mathbf{A} = \widehat{A}^{(k)}$ for any $k \geq r$. Intuitively, if we select enough columns then we can guarantee a perfect approximation.

The remainder of the section proceeds as follows: we present three lemmas which capture elements of the proof of Theorem 2.2, and then present the proof of the main theorem assuming each of the lemmas. Then we will sketch (for brevity) proofs of each lemma, and then finally end with some more detailed comparisons of theoretical error bounds to other approximation schemes in the column Nyström family.

Definition 2.3. Define the **expected residual function** Φ as follows:

$$\Phi(\mathbf{A}) = \mathbf{E}\left[\mathbf{A}^{(1)} \mid \mathbf{A}\right] = \mathbf{A} - \frac{\mathbf{A}^2}{\operatorname{tr}\mathbf{A}}$$

This corresponds to the expectation of residual $\mathbf{A}^{(1)}$ after one iteration of RPCHOLESKY .

Before presenting the first lemma, recall the *Loewner order* on positive, semidefinite (PSD) matrices: $\mathbf{A} \preceq \mathbf{B}$ if $\mathbf{B} - \mathbf{A}$ is PSD. We can now present the first lemma concerning some desirable properties of the function Φ.

Lemma 2.4. *The map Φ is positive, monotone, and concave with respect to the Loewner order. That is, for all PSD $\mathbf{A}, \mathbf{H}$ with the same dimensions,*

$$0 \preceq \Phi(\mathbf{A}) \preceq \Phi(\mathbf{A} + \mathbf{H})$$

and for all weights $\theta \in [0, 1]$,

$$\theta\Phi(\mathbf{A}) + (1-\theta)\Phi(\mathbf{H}) \preceq \Phi(\theta\mathbf{A} + (1-\theta)\mathbf{H})$$

As we iterate the RPCHOLESKY algorithm, we'd like to understand how the residual under the trace norm decays (as the algorithm converges). This type of convergence is also known as the *contraction rate*. In what follows, let $\Phi^{\circ k}$ denote Φ composed with itself k times.

Lemma 2.5. *Fix $r \in \mathbb{N}$. For each PSD $\mathbf{H}$ and $\Delta > 0$, we have that*

$$\operatorname{tr}\Phi^{\circ k}(\mathbf{H}) \leq \operatorname{tr}\left(\mathbf{H} - [\![\mathbf{H}]\!]_r\right) + \Delta\operatorname{tr}(\mathbf{H})$$

whenever

$$k \geq \frac{r\operatorname{tr}\left(\mathbf{H} - [\![\mathbf{H}]\!]_r\right)}{\Delta\operatorname{tr}(\mathbf{H})} + \log_+\left(\frac{1}{\Delta}\right).$$

The last lemma that we will use in the proof of the main theorem shows that the error after k iterations is competitive with the error in the optimal rank k approximation. In particular, we show that at each iteration the error increases by a factor of at most 2:

Lemma 2.6. *For each PSD matrix* $\mathbf{A}$ *and each* $k \in \mathbb{N}$*, the residual* $\mathbf{A}^{(k)}$ *after* k *steps of RPCHOLESKY satisfies*

$$\mathbf{E}\left[\operatorname{tr}\mathbf{A}^{(k)}\right] \leq 2^k \operatorname{tr}\left(\mathbf{A} - [\![\mathbf{A}]\!]_k\right).$$

At this point, the broad idea of the proof (or at least the bound in the first, η-dependent regime) should be clear.

Pf. of Theorem 2.2. By hypothesis, $\mathbf{A}$ is PSD. Recall that by Lemma 2.4, Φ is concave and thus we may apply a matrix analogue of Jensen's inequality (see Appendix A, Lemma A.1) as well as the law of iterated expectation. The residual matrix for every $j = 1, \ldots, k$ satisfies

$$\mathbf{E}\left[\mathbf{A}^{(j)}\right] = \mathbf{E}\left[\mathbf{E}\left[\mathbf{A}^{(j)} \mid \mathbf{A}^{(j-1)}\right]\right] = \mathbf{E}\left[\Phi\left(\mathbf{A}^{(j-1)}\right)\right] \preceq \Phi\left(\mathbf{E}\left[\mathbf{A}^{(j-1)}\right]\right).$$

Using monotonicity (also a consequence of Lemma 2.4), we have that $\mathbf{E}\left[\mathbf{A}^{(k)}\right] \preceq \Phi\left(\mathbf{E}\left[\mathbf{A}^{(j-1)}\right]\right) \preceq \cdots \preceq \Phi^{\circ k}(\mathbf{A})$. Thus, as tr is a linear and order-preserving operator, we have that

$$\mathbf{E}\left[\operatorname{tr}(\mathbf{A} - \mathbf{A}^{(k)})\right] = \mathbf{E}\left[\operatorname{tr}\mathbf{A}^{(k)}\right] \leq \operatorname{tr}\left(\Phi^{\circ k}(\mathbf{A})\right). \tag{2}$$

Now we can just apply Lemma 2.5 with $\mathbf{H} = \mathbf{A}$ and $\Delta = \varepsilon\eta$, at which point we immediately obtain the η-dependent bound in Theorem 2.2:

$$\mathbf{E}\left[\operatorname{tr}(\mathbf{A} - \mathbf{A}^{(k)})\right] \leq (1+\varepsilon) \cdot \operatorname{tr}(\mathbf{A} - [\![\mathbf{A}]\!]_r) \tag{3}$$

for

$$k \geq \frac{r}{\varepsilon} + r \log_+\left(\frac{1}{\varepsilon\eta}\right)$$

The other regime is a bit more delicate. Using the same monotonicity reasoning as we used in (2), only iterated r times, we have that $\mathbf{E}\left[\operatorname{tr}(\mathbf{A} - \mathbf{A}^{(k)})\right] \leq \operatorname{tr}\left(\Phi^{\circ(k-r)}(\mathbf{E}\,\mathbf{A}^{(r)})\right)$. As before, we apply Lemma 2.5 with $\mathbf{H} = \mathbf{E}\,\mathbf{A}^{(r)}$ and $\Delta = \varepsilon\eta \cdot \frac{\operatorname{tr}(\mathbf{A})}{\mathbf{E}[\operatorname{tr}(\mathbf{A})]}$. Before presenting the bounds, we make the key observation that although the residual $\mathbf{A}^{(r)}$ is random (as it's sampled by RPCHOLESKY), by construction it's a Schur complement of $\mathbf{A}$, and therefore $\mathbf{E}\,\mathbf{A}^{(r)} \preceq \mathbf{A}$. Now we will need the Ky Fan variational principle (see Appendix A, Lemma A.2) which can be applied to show that the the error metric we defined earlier (trace norm of the residual $\mathbf{A} - [\![\mathbf{A}]\!]_r$) is monotone with respect to the Loewner order. Thus, 3 holds when

$$k - r \geq \frac{r}{\varepsilon} + r \log_+\left(\frac{\mathbf{E}\left[\operatorname{tr}(\mathbf{A}^{(r)})\right]}{\varepsilon \operatorname{tr}(\mathbf{A} - [\![\mathbf{A}]\!]_r)}\right)$$

At this point we just apply Lemma 2.6 to the numerator inside the logarithmic term, and obtain the desired bound in the second (η-independent) case. This concludes the proof. □

2.2 Proof sketches of supporting lemmas

For brevity and to avoid reiterating details which may be found in [2], we'll only provide the main details relevant to the proofs of Lemmas 2.4 - 2.6.

The proof of the expected residual lemma is very straightforward, effectively leveraging the definitions of Φ as an operator as well as natural properties of the Loewner order $\preceq$.

Pf. Sketch of Lemma 2.4. Recall that by Definition 2.3, $\left(\mathbf{I} - \frac{\mathbf{H}}{\operatorname{tr}\mathbf{H}}\right)\mathbf{H} \succeq \mathbf{0}$, which guarantees positivity. To show concavity, we just have to show $\Phi(\theta\mathbf{A} + (1-\theta)\mathbf{H}) - \theta\Phi(\mathbf{A}) - (1-\theta)\Phi(\mathbf{H}) \succeq 0$ for arbitrary parameter $0 \leq \theta \leq 1$. We just observe that the LHS is precisely

$$\frac{\theta(1-\theta)}{\theta \operatorname{tr}\mathbf{A} + (1-\theta)\operatorname{tr}\mathbf{H}}\left(\sqrt{\frac{\operatorname{tr}\mathbf{H}}{\operatorname{tr}\mathbf{A}}}\mathbf{A} - \sqrt{\frac{\operatorname{tr}\mathbf{A}}{\operatorname{tr}\mathbf{H}}}\mathbf{H}\right)^2$$

(this is just algebra exploiting the symmetry of the original LHS). This is clearly non-negative because $\mathbf{A}, \mathbf{H}$ are both PSD. Proving monotonicity is just a consequence of applying concavity and positivity to $\Phi(\mathbf{A} + \mathbf{H}) = 2\Phi\left(\frac{\mathbf{A}+\mathbf{H}}{2}\right)$, so we're taking $\theta = 1 - \theta = \frac{1}{2}$. □

The proof for the contraction rate is notably more involved. The basic idea involves realizing that it suffices to consider diagonal matrices, defining a function used to upper bound $\operatorname{tr}\Phi^{\circ k}(\mathbf{H})$, and then converting the discrete process to a continuous process which we may solve classically using ordinary differential equations.

Proof. Pf. Sketch of Lemma 2.4 As mentioned earlier, if we have that $\mathbf{H}$ has eigendecomposition $\mathbf{H} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^*$, the definition of Φ gives us that $\Phi(\mathbf{H}) = \mathbf{V}\Phi(\mathbf{\Lambda}\mathbf{V}^*$, and therefore

$$\Phi^{\circ k}(\mathbf{H}) = \mathbf{V}\Phi^{\circ k}(\mathbf{\Lambda})\mathbf{V}^*$$

Thus, without loss of generality we may assume $\mathbf{H} = \mathbf{\Lambda}$ is diagonal. As stated earlier, we'd like to bound $\operatorname{tr}\Phi^{\circ k}(\mathbf{H})$. Define

$$f_k(a,b,r,q) = \operatorname{tr}\Phi^{\circ k}\left(\operatorname{diag}\left(\frac{a}{r},\dots,\frac{a}{r},\frac{b}{q},\dots\frac{b}{q}\right)\right)$$

Here, $\frac{a}{r}$ is repeated r times and $\frac{b}{q}$ is repeated q times. The clever observation is that using concavity as well as the fact that the trace is a symmetric function (as addition is commutative), we obtain the upper bound

$$\operatorname{tr}\Phi^{\circ k}(\mathbf{H}) \leq f_k(\operatorname{tr}[\![\mathbf{H}]\!]_r, \operatorname{tr}(\mathbf{H} - [\![\mathbf{H}]\!]_r), r, N-r) \tag{4}$$

The next observation uses the fact that Φ is defined on matrices of arbitrary dimension, to show that f_k is weakly increasing in q (so $f_k(a,b,r,q) \leq f_k(a,b,r,q+1)$. The details can be found in the paper and it's a routine computation. It suffices to determine a worst case bound for f_k that we can apply to (4). For each k we define the quantity $a^{(k)}$ and $b^{(k)}$ such that

$$\operatorname{diag}\left(\frac{a^{(k)}}{r},\dots,\frac{a^{(k)}}{r},\frac{b^{(k)}}{q},\dots\frac{b^{(k)}}{q}\right) := \Phi^{\circ k}\left(\operatorname{diag}\left(\frac{a}{r},\dots,\frac{a}{r},\frac{b}{q},\dots\frac{b}{q}\right)\right)$$

We again refer to Definition 2.3 to obtain the recurrence relations (again, leaving the details to the paper)

$$a^{(k)} - a^{(k-1)} = -\frac{\left(a^{(k-1)}\right)^2}{r\left(a^{(k-1)} + b^{(k-1)}\right)}$$

$$b^{(k)} - b^{(k-1)} = -\frac{\left(b^{(k-1)}\right)^2}{q\left(a^{(k-1)} + b^{(k-1)}\right)}$$

Observe that $f_k(a,b,r,q) = a^{(k)} + b^{(k)}$ and also that when we take the limit as $q \to \infty$, we get the limiting values $\overline{a}^{(k)}$ and $\overline{b}^{(k)}$. Note that we take $\overline{b}^{(k)}$ to be constant in the limit we will refer to as b. We get the discrete evolutionary process

$$\overline{a}^{(k)} - \overline{a}^{(k-1)} = -\frac{\left(\overline{a}^{(k-1)}\right)^2}{r\left(\overline{a}^{(k-1)} + b\right)}$$

with initial condition $\overline{a}^{(0)} = a$, and therefore the upper bound $f_k(a,b,r,q) \leq a+b$ holds. We upper bound the discrete system $\overline{a}^{(t)}$ with the continuous system $x(t)$ with

$$\frac{d}{dt}x(t) = -\frac{x(t)^2}{r(x(t)+b)}$$

Then we use separation of variables and compute an integral (the explicit computation is omitted from the sketch) to obtain that the time t^* at which $x(t^*) = \Delta(a+b)$ satisfies

$$t^* \leq \frac{rb}{\Delta(a+b)} + r\log_+\left(\frac{1}{\Delta}\right)$$

Translating to the discrete system, and substituting $a = \operatorname{tr}[\![\mathbf{H}]\!]_r$ and $b = \operatorname{tr}(\mathbf{H} - [\![\mathbf{H}]\!]_r)$ yields the desired conclusion. □

The proof, although a bit long, doesn't make use of any non-obvious results or principles, and is effectively just a clever construction that also makes extensive use of desirable properties of the expected residual function Φ. Further, the particular shape of k in the main theorem becomes apparent, as it's just an evaluation of a very explicit integral.

The last lemma is the least intuitive, but the idea is essentially as follows:

Pf. Sketch of Lemma 2.6. First, we orthogonally project $\mathbf{A}$ onto k eigenvectors with largest corresponding eigenvalues, which we call $\mathbf{P}$. We let $\mathbf{P}_\perp = \mathbb{I} - \mathbf{P}$. We define the following sampling procedure, which will ultimately be equivalent to sampling the first pivot s_1 using RPCHOLESKY :

1. **Case I:** With probability $\mathrm{tr}(\mathbf{AP})$, select $s_i = i$ with probability $\frac{(\mathbf{AP})ii}{\mathrm{tr}(\mathbf{AP})}$
2. **Case II:** With probability $\mathrm{tr}(\mathbf{AP}_\perp)$, select $s_i = i$ with probability $\frac{(\mathbf{AP}_\perp)ii}{\mathrm{tr}(\mathbf{AP}_\perp)}$

Aside from distinguishing the two cases in which the part of the partition our sampled element lands, this is precisely the same procedure as in RPCHOLESKY . In Case I, we use the bound:

$$\mathrm{tr}(\mathbf{A}^{(1)} - [\![\mathbf{A}^{(1)}]\!]_{k-1}) = \sum_{j=k}^{N} \lambda_j(\mathbf{A}^{(1)}) \leq \lambda_k(\mathbf{A}) + \mathrm{tr}(\mathbf{A} - [\![\mathbf{A}]\!]_k)$$

where we make use of the Weyl monotonicity principle [6]. In the Case II, the idea is to choose $N - k$ specific eigenvectors orthonormal to the ones from $\mathbf{P}$, apply the definition of $\mathbf{A}^{(1)}$ to obtain an inequality, and take conditional expectations on both sides (the rigorous details of these computations are omitted in this sketch). We obtain

$$\mathbf{E}\left[\mathrm{tr}\left(\mathbf{A}^{(1)} - [\![\mathbf{A}^{(1)}]\!]_{k-1}\right) \mid \text{Case II}\right] \leq \left(1 + \frac{\lambda_1(\mathbf{A})}{\mathrm{tr}(\mathbf{AP})}\right) \mathrm{tr}(\mathbf{A} - [\![\mathbf{A}]\!]_k)$$

Combining the bounds for both cases using the fact that Case I and Case II are complementary events and applying the law of total probability yields

$$\mathbf{E}\left[\mathrm{tr}\left(\mathbf{A}^{(1)} - [\![\mathbf{A}^{(1)}]\!]_{k-1}\right)\right] \leq \left(1 + \frac{\lambda_1(\mathbf{A}) + \lambda_k(A)}{\mathrm{tr}\,\mathbf{A}}\right) \mathrm{tr}(\mathbf{A} - [\![\mathbf{A}]\!]_k)$$

The pre-factor can be upper bounded by 2 since $\mathrm{tr}\,\mathbf{A} = \sum_k \lambda_k(\mathbf{A})$, and then we can just iterate this procedure repeatedly to obtain

$$\mathbf{E}\left[\mathrm{tr}\left(\mathbf{A}^{(k-m)} - [\![\mathbf{A}^{(k-m)}]\!]_m\right)\right] \leq 2\,\mathbf{E}\left[\mathrm{tr}\left(\mathbf{A}^{(k-(m+1))} - [\![\mathbf{A}^{(k-(m+1))}]\!]_{m+1}\right)\right]$$

Thus, across $m = 0$ to j, we have

$$\mathbf{E}\,\mathrm{tr}(\mathbf{A}^{(k)}) \leq 2^k\,\mathrm{tr}(\mathbf{A} - [\![\mathbf{A}]\!]_k)$$

□

This completes the proof of the last lemma, and therefore establishes the main theorem, Theorem 2.2.

2.3 The $k \geq \frac{r}{\varepsilon}$ threshold is sharp

Having presented theoretical error bounds, it's natural to ask how well these bounds are expected to hold. Note that this is different than an empirical analysis of generic matrices, as the sharp example is in many ways contrived. We present the following theorem, and will outline the construction of the sharp example matrix:

Theorem 2.7 (4.3(c) in [2])**.** *There exists a PSD matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ such that any rank−k column Nyström approximation satisfies*

$$\mathrm{tr}(\mathbf{A} - \mathbf{A}^{(k)}) > (1 + \varepsilon) \cdot \mathrm{tr}(A - [\![\mathbf{A}]\!]_r)$$

for any $k < r/\varepsilon$.

The proof is just an explicit construction, where we take $\mathbf{A}$ to be the $M \cdot r \times M \cdot r$ matrix given by:

$$\mathbf{A} = \mathrm{diag}(C_{M,\delta}, \ldots, C_{M,\delta})$$

where each C_δ looks like

$$C_{M,\delta} := \begin{bmatrix} 1 & \delta & \cdots & \delta \\ \delta & 1 & \cdots & \vdots \\ \vdots & & \ddots & \\ \delta & \cdots & & 1 \end{bmatrix}$$

Upon choosing M very large and δ sufficiently close to 1, it holds that

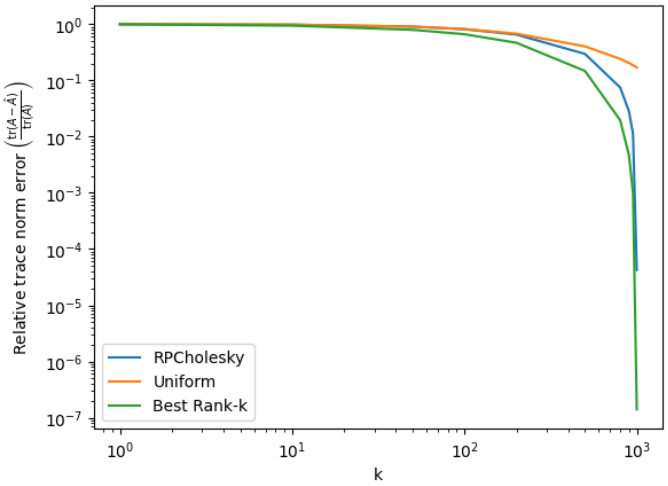
$$\operatorname{tr}(\mathbf{A} - \mathbf{A}^{(k)}) > (1 + \varepsilon) \cdot \operatorname{tr}(A - [\![\mathbf{A}]\!]_r)$$

so the column Nyström approximation deviates from the optimal low rank approximation by a nontrivial amount.

3 Implementation and Empirical Performance of RPCHOLESKY

In this section, we will empirically compare the performance of RPCHOLESKY against uniform pivot sampling and the best low rank approximation taken as the truncated eigendecomposition of a PSD matrix. We sample a random PSD matrix $A \in \mathbb{R}^{N \times N}$ by sampling its random eigendecomposition QDQ^T, i.e., we uniformly pick eigenvalues $\lambda_i \in [0.01, 100]$, construct $D = \operatorname{diag}(\lambda_1, \ldots, \lambda_N)$, and randomly sample an orthonormal matrix Q. The sampling code is attached in the appendix.

We sample 1000 matrices, compute their best k-rank approximation and Nyström approximations using k iterations of both strategies above. The code for computing Nyström approximations is also attached in the appendix. We average the relative trace norm error for each method across all 1000 matrices and plot them for different values of k below:



Observe that in the $[100, 1000]$ interval, RPCHOLESKY converges faster than uniform sampling of pivots, and is still fairly competitive with the truncated eigenvalue decomposition. We further remark that we observed all low-rank approximation schemes were roughly competitive when we chose our matrix completely randomly (likely due to high condition numbers with high probability).

4 Application to Kernel Quadrature

In this section, we will discuss [4], which introduces an interesting application of RPCHOLESKY to the Kernel Quadrature problem.

4.1 Understanding Kernel Quadrature

Kernel quadrature is a mathematical method used to approximate integrals. In lots of real-world scenarios, calculating integrals exactly can be very challenging or impossible. Kernel quadrature provides a solution by approximating the integral as a weighted sum of the function's values at specific points, called "nodes."

More formally, suppose we are given a function g, and we are looking for weights $(w_1, \ldots, w_n)$ and nodes $S = (s_1, \ldots, s_n)$ such that our weighted sum is a good approximation of the integral. We choose these weights and nodes to minimize the largest possible difference between the true integral and our approximation. However, we want to restrict ourselves to only functions f such that $\|f\|_{\mathcal{H}} \leq 1$, which means that these functions shouldn't be too wiggly or rough.

The authors derive the following formula for our approximation error:

$$\mathrm{Err}(S, w; g) = \left\| Tg - \sum_{i=1}^{n} w_i k(\cdot, s_i) \right\|_{\mathcal{H}}, \tag{5}$$

where Tg represents the true integral we want to approximate (since T is defined at the integral operator), and $k(\cdot, s_i)$ represents a kernel function evaluated at each node s_i. This objective is simply the least squares objective for approximating Tg by a linear combination $\sum_{i=1}^{n} w_i k(\cdot, s_i)$.

In fact, if we fix our nodes S, then the optimal weights $w^\star$ satisfy $k(S, S)w^\star = Tg(S)$, a linear system of equations. So, it suffices to just find the optimal nodes S, since the optimal weights can easily be derived from them. The paper shows that these optimal nodes can simply be found through our RPCHOLESKY algorithm.

4.2 Applying a reformulated version of RPCHOLESKY

One thing to note is that RPCHOLESKY (as we have covered so far) is a discrete algorithm applied to finite matrices. Therefore, its application to selecting nodes on integrable functions may not immediately be clear. However, we can extend RPCHOLESKY in a simple way to the continuous domain. Below, we will show how to do this for the naive implementation of RPCHOLESKY (although it can also be done more efficiently through the use of rejection sampling).

To make this transition, we consider the kernel function $k(x, y)$ that defines the geometry of our function space. This kernel function, which must be integrable along the diagonal, provides a means to sample potential nodes for kernel quadrature. The first node, s_1, is selected according to a probability distribution proportional to the kernel's diagonal, just as in regular RPCHOLESKY :

$$s_1 \sim \frac{k(x, x)}{\int_X k(z, z) d\mu(z)} \tag{6}$$

Once s_1 is chosen, its influence on the kernel is subtracted from the entire kernel function, updating it to a new kernel $k'(x, y)$, which is also analogous to the residual subtraction step in regular RPCHOLESKY as well:

$$k'(x, y) = k(x, y) - \frac{k(x, s_1)k(s_1, y)}{k(s_1, s_1)} \tag{7}$$

This update can be interpreted as a form of Gaussian elimination on an infinite matrix. Or, if we view our kernel as a Gaussian process, this represents conditioning on the value of the process at s_1. We then use the updated kernel k' to select the next node s_2, and continue onwards until we've selected as many nodes as we want.

By repeating this process, we keep iteratively refining our kernel and selecting nodes that are optimally spread according to the geometry induced by our kernel function. As shown in the previous subsection, once we have our nodes, we can easily find the optimal weights through solving a simple linear system, completing the algorithm.

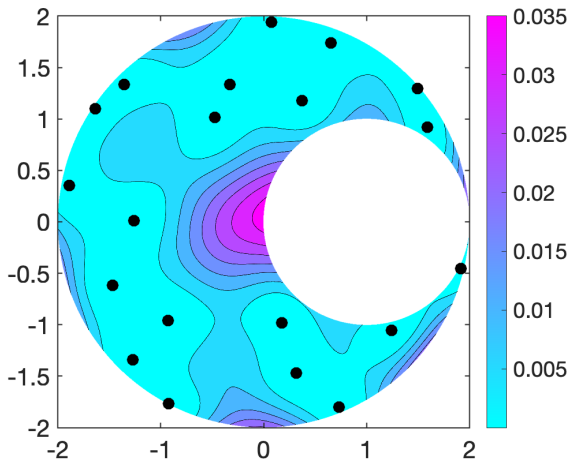
4.3 Intuitive Explanation of RPCHOLESKY 's Application

One of the main findings in this kernel quadrature problem is that the worst case error with the ideal weights $w^\star$ can be written in terms of $\hat{k}$, the Nystrom approximation to the kernel function:

$$\mathrm{Err}(w^*, s) = \left\| \int_\Omega (k(\cdot, x) - \hat{k}_s(\cdot, x)) g(x) d\mu(x) \right\|.$$

This implies that it would be reasonable to select nodes to make the Nystrom approximation as accurate as possible. This reduced problem has many solutions in randomized linear algebra, including ridge leverage score sampling, but the simplest one (and the subject of our project) is the RPCHOLESKY algorithm.

The intuition for why RPCHOLESKY is good for node selection is that it tends to select nodes that are far from the nodes it has previously selected.



This diagram [3] shows a region where RPCHOLESKY selects nodes. It has already selected 20 nodes (labeled in black), and the probability distribution of the next node is colored (with purple being the most likely location). We can see that the regions near previously selected nodes are mostly cyan (unlikely to be the next location) while the center is very purple since it's far from any previously selected nodes. This depicts why RPCHOLESKY is a good algorithm for selecting nodes for the kernel quadrature problem.

5 Application to Kernel Ridge Regression

In this section, we will cover the application of RPCHOLESKY to speeding up kernel ridge regression.

5.1 Kernel Ridge Regression

Kernel Ridge Regression (KRR) is an extension of Ridge Regression that allows it to deal with non-linear data. It combines ridge regression (which is linear least squares with ℓ_2-norm regularization), with the kernel trick. This allows the method to find linear relationships in the feature space transformed through the kernel, rather than the original input space (where relationships might be nonlinear).

In regular ridge regression, the objective is to minimize the sum of the squares of the residuals, with a penalty on the size of the coefficients. The penalty is the ℓ_2 norm of the coefficients multiplied by a regularization parameter λ:

$$\min_{\beta} \left\{ \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i^\top \beta)^2 + \lambda \|\beta\|_2^2 \right\}. \tag{8}$$

KRR incorporates the regularization term from ridge regression, mitigating overfitting by shrinking the coefficients, and adds a positive definite kernel function $K : \mathbb{R}^d \times \mathbb{R}^d \to R$ to allow handling non-linear relationships in the original data. Given this kernel function K and a regularization parameter λ, KRR solves the following optimization problem:

$$\min_{\beta} \left\{ \frac{1}{N} \sum_{j=1}^{N} \left(y_j - \sum_{i=1}^{N} \beta_i K(x^{(i)}, x^{(j)}) \right)^2 + \lambda \sum_{i,j=1}^{N} \beta_i \beta_j K(x^{(i)}, x^{(j)}) \right\}. \tag{9}$$

If $A \in \mathbb{R}^{N \times N}$ is the kernel matrix, then $\beta^\star = (A + \lambda N I)^{-1} y$ is the optimal value of β. Note that if we were to find $\beta^\star$ directly, we need to solve a dense linear system. This would take $O(N^3)$ time.

Once $\beta^\star$ is found, KRR gives us the prediction function $f(x; \beta^\star) = \sum_{i=1}^{N} \beta_i K(x^{(i)}, x)$. Note that evaluating this prediction function takes $O(N)$ time.

5.2 Speeding up with RPCHOLESKY

Using RPCHOLESKY , we can significantly accelerate kernel ridge regression to only require $O(k^2N)$ time, where k is our approximation rank (the number of pivots). The steps are as follows:

1. First, we apply our RPCHOLESKY algorithm to kernel matrix A. This produces a set of pivots $S = \{s_1, \ldots, s_k\}$.
2. Then, restrict ourselves to only these pivots, and solve a smaller kernel ridge regression problem:

$$\min_{\hat{\beta} \in \mathbb{R}^k} \left\{ \frac{1}{N} \sum_{j=1}^{N} |\hat{f}(x^{(j)}; \hat{\beta}) - y_j|^2 + \lambda \sum_{i,j=1}^{k} \hat{\beta}_i \hat{\beta}_j K(x^{(s_i)}, x^{(s_j)}) \right\}. \tag{10}$$

   As before, the $\hat{\beta}$ here can be found by solving the following linear system:

$$\hat{\beta} = (A(S,:)^\top A(S,:) + \lambda N A(S,S))^{-1} A(S,:)^\top y. \tag{11}$$

   The trick is that now, this linear system is much smaller and only involves a matrix in $\mathbb{R}^{k \times k}$, so it is pretty cheap to solve. It takes $O(k^2N)$ time to find the matrix product in the first term, and $O(k^3)$ time to solve for $\hat{\beta}$ in this linear system.

3. Lastly, as before, this $\hat{\beta}$ allows us to define a prediction function

$$\hat{f}(\cdot; \hat{\beta}) = \sum_{i=1}^{k} \hat{\beta}_i K(x^{(s_i)}, \cdot). \tag{12}$$

   Note that we can evaluate this prediction function in just $O(k)$ time, rather than $O(N)$ before.

Of course, RPCHOLESKY is a randomized algorithm (based on sampling pivots proportionally to the diagonal entries), so this prediction function is only as good as the low-rank approximation RPCHOLESKY gives.

5.3 Implementation

We consider the problem of handwritten digit recognition from the MNIST dataset [5] The goal is to train a 10-way classification model that can categorize each of the digits from a test set of 10,000 images. While this dataset contains 60,000 training samples, we only consider the first 30,000 since that is the biggest data subset whose kernel matrix would fit within the memory of the compute resources available to us.

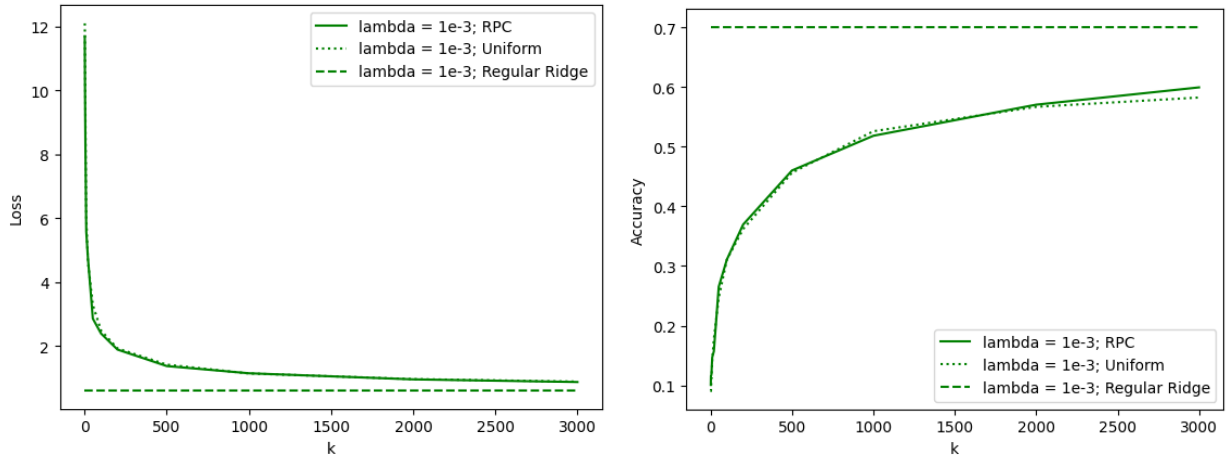
We approach this problem by training a kernel ridge regression model. This choice may seem unorthodox since ridge regression is typically not used for classification problems but a classification problem can be framed as a regression problem. Namely, we simply take each of the discrete class labels as scalar values that a regression model must predict; during inference, the model outputs are rounded to the nearest integer. See the appendix for the code implementation.

We flatten each 28×28 image in the MNIST dataset to 784-dimensional feature vectors. We standardize them from a range of $[0, 255]$ to $[0, 1]$ and employ the radial basis function (RBF) kernel:

$$K(x1, x_2; \sigma) = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

We found $\sigma = 5$ and $\lambda_{ridge} = 10^{-5}$ to work well with our experiments.

We perform KRR with the entire kernel matrix which, as per the discussion above, takes $O(N^3)$ time to run. We also run KRR with Nyström approximations of the kernel matrix, per the approach described in section 5.2, yielded by the RPCHOLESKY and uniform pivoting strategies. In the figure below, we plot the testing loss and testing accuracy of different rank-k Nyström approximations, along with the baseline expected from the full kernel matrix.



As expected, the full kernel matrix outperforms either of its approximations, achieving a test accuracy of exactly 70%, but this comes at a computational cost: it is also $\sim 3x$ slower than either of the Nyström approximations at $k = 3,000$, which can achieve a very modest test accuracy of 63% (averaged across multiple Nyström approximations, to keep uncertainties regarding their randomized nature small). In addition, we note that the performance of RPCHOLESKY and uniform pivoting is very similar up to $k = 1,500$, after which the former method starts to outperform by a small margin.

The theoretical error bounds we previously covered, as well as our own implementation and testing of RPCHOLESKY , show that RPCHOLESKY can give competitive and accurate low-rank approximations, supporting its use in speeding up kernel ridge regression.

A Appendix: Additional lemmas

We present the operator theoretic analogue of Jensen's inequality found in [1]

Lemma A.1. *Let Φ be any operator convex function. Then for a $*$-subalgebra $\mathfrak{U}$ of M_n,*

$$\Phi(\underset{\mathfrak{U}}{\mathbf{E}}[\mathbf{A}]) \leq \underset{\mathfrak{U}}{\mathbf{E}}[\Phi(\mathbf{A})]$$

In our applications we shouldn't need to consider any proper subalgebra. Next we present the Ky Fan variational principle used for the η-independent bound in Theorem 2.2 found in [6]:

Lemma A.2. *Let H be an $n \times n$ Hermitian matrix. Denote by S_k a set of k orthonormal vectors $x_1, \ldots, x_k$. Then*

$$\sum_{i=1}^{k} \lambda_i(H) = \max_{S_k} \sum_{i=1}^{k} x_i^* H x_i$$

B Appendix: Implementation

B.1 Randomly Sampling PSD Matrices

Here is the code snippet we use for generating random PSD matrices:

```
def get_random_psd(n: int) -> np.ndarray:
    Q = ortho_group.rvs(n)
    D = np.random.uniform(0.01, 100, size=n)
    D = np.sort(D)[::-1]
    A = Q @ np.diag(D) @ Q.T
    return A, Q, D
```

B.2 Efficient Implementation for computing Nyström Approximations

The efficiency strategy in which the implementation below differs from the general column Nyström algorithm outlined in the introduction lies in the fact that we keep track of our approximation through a factor matrix $\mathbf{F}$, and return $\mathbf{F}\mathbf{F}^*$ at the end rather than $\mathbf{A}(:,\mathcal{S})\ \mathbf{A}(\mathcal{S},\mathcal{S})^\dagger\ \mathbf{A}(\mathcal{S},:)$.

For comparison, we also include the efficient approximation method given by uniformly sampling pivots:

```
def efficient_nystrom(A: np.ndarray, k: Union[int, list[int]],
                      method: str = 'rpc', return_pivots: bool = False):
    if isinstance(k, int):
        k_list = [k]
    else:
        k_list = sorted(k)

    n = A.shape[0]
    if method == 'uniform':
        pivots = np.random.choice(np.arange(n), size=max(k_list))
        for k in k_list:
            pivot = np.unique(pivots[:k])
            S = set(pivot)
            L = A[:, pivot]
            R = A[pivot, :]
            C = A[np.ix_(pivot, pivot)]
            A_hat = L @ np.linalg.pinv(C) @ R
            yield A_hat if not return_pivots else S
    else:
        assert max(k_list) <= n
        F = np.zeros((n, max(k_list)))
        d = np.diagonal(A).copy()
        pivots = set()
        for i in range(max(k_list)):
            if method == 'rpc':
                d_ = np.clip(d, a_min=0, a_max=None)
                weights = d_ / np.sum(d_)
                pivot = np.random.choice(np.arange(n), p=weights)
            else:
                raise ValueError(f"Unknown method: {method}")
            pivots.add(pivot)
            g = A[:, pivot] - F[:, :i] @ F[pivot, :i].T
            F[:, i] = g / np.sqrt(g[pivot])
            d -= np.square(F[:, i])
            if i + 1 in k_list:
                A_hat = F @ F.T
                yield A_hat if not return_pivots else pivots.copy()
```

B.3 Implementation of Kernel Ridge Regression

We first define a method for computing the RBF kernel:

```
from sklearn.metrics.pairwise import euclidean_distances

def rbf_kernel(X, Y, sigma):
    dsts = euclidean_distances(X, Y) # faster
    return np.exp(-dsts ** 2 / (2 * sigma ** 2))
```

We also provide our data preprocessing code:

```
from keras.datasets import mnist

(train_X, train_y), (test_X, test_y) = mnist.load_data()
train_X = train_X.astype(np.float32) / 255.0
test_X = test_X.astype(np.float32) / 255.0

train_Xs = []
train_ys = []
for i in range(10):
    idxs = np.where(train_y == i)[0][:3000]
    train_Xs.append(train_X[idxs])
    train_ys.append(train_y[idxs])

train_X = np.concatenate(train_Xs, axis=0)
train_y = np.concatenate(train_ys, axis=0)

train_X = train_X.reshape(train_X.shape[0], -1)
test_X = test_X.reshape(test_X.shape[0], -1)

train_y = train_y.astype(np.float32)
test_y = test_y.astype(np.float32)
```

Here is our implementation of KRR that leverages Nyström approximations:

```
import scipy

train_losses = []
test_losses = []
train_accs = []
test_accs = []
k_list = [1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 3000]

K = rbf_kernel(train_X, train_X, 5)
K_test = rbf_kernel(test_X, train_X, 5)

N = train_X.shape[0]
LAMBDA = 0.00001

train_losses.append([])
test_losses.append([])
train_accs.append([])
test_accs.append([])

for method in ['rpc', 'uniform']:

    train_losses[-1].append([])
    test_losses[-1].append([])
    train_accs[-1].append([])
    test_accs[-1].append([])

    pivots = efficient_nystrom(K, k_list, method, return_pivots=True)

    for k in k_list:
        S = np.array(list(next(pivots)))
        beta = scipy.linalg.solve(K[S, :] @ K[:, S] + LAMBDA * N * K[np.ix_(S, S)], \
```

```
            K[S, :] @ train_y)
        train_y_hat = K[:, S] @ beta
        test_y_hat = K_test[:, S] @ beta
        train_loss = np.mean((train_y_hat - train_y) ** 2)
        test_loss = np.mean((test_y_hat - test_y) ** 2)
        train_pred = np.clip(np.rint(train_y_hat), a_min=0, a_max=9)
        test_pred = np.clip(np.rint(test_y_hat), a_min=0, a_max=9)
        train_acc = np.mean(train_pred == train_y)
        test_acc = np.mean(test_pred == test_y)

        train_losses[-1][-1].append(train_loss)
        test_losses[-1][-1].append(test_loss)
        train_accs[-1][-1].append(train_acc)
        test_accs[-1][-1].append(test_acc)
```

Finally, here is an implementation that performs KRR with the full kernel matrix:

```
N = train_X.shape[0]
krr_beta = scipy.linalg.solve(K + LAMBDA * N * np.eye(N), train_y)
reg_ridge_train_y_hat = K @ krr_beta
reg_ridge_test_y_hat = K_test @ krr_beta
train_loss = np.mean((reg_ridge_train_y_hat - train_y) ** 2)
test_loss = np.mean((reg_ridge_test_y_hat - test_y) ** 2)
train_pred = np.clip(np.rint(reg_ridge_train_y_hat), a_min=0, a_max=9)
test_pred = np.clip(np.rint(reg_ridge_test_y_hat), a_min=0, a_max=9)
train_acc = np.mean(train_pred == train_y)
test_acc = np.mean(test_pred == test_y)
```

References

[1] Eric A. Carlen. Trace inequalities and quantum entropy: An introductory course. 2009.

[2] Yifan Chen, Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. Randomly pivoted cholesky: Practical approximation of a kernel matrix with few entry evaluations, 2023.

[3] Ethan N. Epperly. Five interpretations of kernel quadrature, 2023. Accessed: 2023-12-05.

[4] Ethan N. Epperly and Elvira Moreno. Kernel quadrature with randomly pivoted cholesky, 2023.

[5] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[6] Fuzhen Zhang. Matrix theory: Basic results and techniques. 1999.