Randomly Pivoted Cholesky: Investigation and Applications

Ryan Divan

Princeton University,
rd2700@princeton.edu

Abstract. This is a brief report on "Randomly pivoted Cholesky" by Chen et al. (2024) for the MAT321 at Princeton University in Fall 2025, taught by Professor Marc Aurèle Gilles. This report includes an overview of key findings in the paper and is accompanied by a Github repository with algorithm implementations and interesting results arising from the algorithm and its applications.

1 Introduction

In modern applications, particularly data science and machine learning, high-dimensional matrices are commonly used for a variety of tasks. However, due to the structured nature of such matrices, especially kernel matrices, much of their structure and information can be captured by a low-rank approximation. This leads us to a motivating theorem that describes the best low-rank approximation to a real matrix:

Theorem 1 (Eckart-Young-Mirsky). *Let $A \in \mathbb{R}^{m\times n}$ and let A_t denote the truncated singular value decomposition (SVD) of A, i.e. $A_t = \sum_{i=1}^{t} \sigma_i u_i v_i^T$, where $\{u_i\}$ and $\{v_i\}$ are left and right singular vectors of A respectively. Then, where $||\cdot||$ denotes the 2-norm or the Frobenius norm, it follows that:*

$$||A - A_t|| = \min_{B\in\mathbb{R}^{m\times n}, rank(B)=t} ||A - B||$$

In other words, the truncated SVD A_t is the best rank t approximation of A under the 2-norm and the Frobenius norm. This has some brilliant effects on computational and storage efficiency; we need only store the singular values and the singular vectors, which is $\mathcal{O}(t(n+m))$, and for matrix-vector multiplication, we can calculate $A_t x$ in $\mathcal{O}(t(n+m))$ as opposed to $\mathcal{O}(nm)$. However, computing the SVD (and consequently the truncated SVD) is computationally expensive, on the order of $\mathcal{O}(m^2 n)$ (assuming $m > n$), which can outweigh its optimality as an approximation in many applications where lower precision can be accepted.

In many kernel methods in machine learning, we end up dealing with symmetric positive semi-definite (PSD) kernel matrices that exhibit significant spectral decay; in this case, Chen et al. (2024) propose the randomly pivoted Cholesky algorithm (RPCholesky), which exploits this common structure of kernel matrices

to avoid the computational burden of other factorization method while providing a reliable Nyström approximation to A that can be leveraged for efficient kernel methods in machine learning and scientific computing: In particular, leveraging the block RPCholesky variant, we are able to produce a theoretical guarantee in trace-norm error:

Theorem 2 (Chen et al.). *Fix a positive semi-definite matrix $A \in \mathbb{R}^{n\times n}$, approximation rank r, and tolerance $\epsilon \in (0,1)$. Then, let $T \geq r/\epsilon$ be the block size. After selecting k/T blocks of columns, block RPCholesky produces a random Nyström approximation $\hat{A}$ with rank k such that:*

$$\mathbb{E}tr(A - \hat{A}) \leq (1-\epsilon)^{-1} tr(A - [[A]]_r) + \epsilon^{k/T} tr(A)$$

where $[[A]]_r$ denotes the best rank-r positive semi-definite approximation with respect to trace-norm error $tr(A - [[A]]_r)$.

The RPCholesky algorithm is able to tremendously decrease time and storage complexity for kernel methods in scientific computing while suffering negligible loss in accuracy due to this theoretical guarantee. In this paper, we will explore several interesting features of this algorithm, experimental results, and applications in domains not explored in the original paper.

2 Algorithm Overview

Algorithm 1 RPCholesky

Input: PSD matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$; approximation rank k
Output: Pivot set $S = \{s_1, \ldots, s_k\}$; matrix $\mathbf{F} \in \mathbb{C}^{n\times k}$ such that $\hat{\mathbf{A}} = \mathbf{F}\mathbf{F}^*$
Initialize: $\mathbf{F} \leftarrow \mathbf{0}_{n\times k}$, $\mathbf{d} \leftarrow \mathrm{diag}(\mathbf{A})$
for $i = 1$ to k **do**
  Sample pivot $s_i \sim \boldsymbol{d}/\sum_{j=1}^{N} \boldsymbol{d}(j)$ ▷ Probability proportional to residual diagonal
  $\boldsymbol{g} \leftarrow \boldsymbol{A}(:, s_i)$ ▷ Evaluate column s of input matrix
  $\boldsymbol{g} \leftarrow \boldsymbol{g} - \boldsymbol{F}(:, 1:i-1)\boldsymbol{F}(s_i, 1:i-1)^*$ ▷ Remove overlap with previous columns
  $\boldsymbol{F}(:, i) \leftarrow \boldsymbol{g}/\sqrt{\boldsymbol{g}(s_i)}$ ▷ Update approximation
  $\boldsymbol{d} \leftarrow \boldsymbol{d} - |\boldsymbol{F}(:, i)|^2$ ▷ Track diagonal of residual matrix
  $\boldsymbol{d} \leftarrow \max\{\boldsymbol{d}, \mathbf{0}\}$ ▷ Ensure diagonal remains nonnegative
end for

The basic RPCholesky algorithm is a variant of the pivoted partial Cholesky algorithm, which only deviates at the sampling step; while RPCholesky samples with probability proportional to the residual diagonal, other strategies include uniform sampling and greedy sampling (which effectively makes the algorithm deterministic). These methods are united in producing a column Nyström approximation, which is defined as follows:

Definition 1 (Column Nyström Approximation). *The column Nyström approximation $\hat{A}$ of an SPD matrix $A \in \mathbb{R}^{n \times n}$ is defined as follows:*

$$\hat{A} = A[:, S]A[S, S]^{\dagger}A[S, :], \text{ where } S \subseteq [1, n]$$

Note that $A^{\dagger}$ denotes the Moore-Penrose pseudoinverse of a matrix A and S is a set of pivot indices. The Nyström approximation is usually given in the factorized form $\hat{A} = F^*F$.

There are a few key properties the Nyström approximation enjoys.

1. The Nyström approximation $\hat{A}$ agrees with the matrix A at the pivot columns S, i.e. $\hat{A}[:, S] = A[:, S]$.
2. The range of the approximation $\hat{A}$ is precisely the span of the selected columns of A, i.e. $\text{range}(\hat{A}) = \text{range}(A[:, S])$.
3. We have that $\hat{A}$ satisfies $0 \preceq \hat{A} \preceq A$, where for any Hermitian matrices X, Y, $Y \preceq X$ implies $X - Y$ is positive semi-definite.

In fact, we have that $\hat{A}$ is the maximal SPD matrix that satisfies properties 2 and 3, where maximal is with respect to the Loewner ordering $\preceq$ defined above. The goal of the original paper was to find a set of pivots S for which the trace norm is minimized (and certainly $\text{tr}(A - \hat{A}) \geq 0$ since $\hat{A} \preceq A$), since this norm is particularly relevant to and applicable in kernel learning contexts. In applications, the random selection proportional to the diagonal is an optimal choice, providing theoretical guarantees and practical results to be discussed in the following sections.

3 Main Theorem and Theoretical Guarantees

We can augment the RPCholesky algorithm with a key variant, block RPCholesky, with a block parameter $T > 1$ (pseudocode below). This is inspired by past work with pivoted QR decompositions, and convergence for similar methods for pivoted Cholesky can be shown. In particular, at step i, we select T pivot columns with probability proportional to the diagonal of $A^{(i-1)}$. Then, we update A^i by eliminating these columns, and repeat.

In particular, this algorithm is able to enjoy the trace-norm error guarantee from **Theorem 2**, since it has nontrivial block size T. One drawback is that T is determined by r and ϵ, and thus these factors must be determined a priori. However, we are able to find a strong bound that also provides a theoretical guarantee when we presume $T = 1$ a priori, which is the same as **Algorithm 1**:

Theorem 3 (Randomly pivoted Cholesky). *Let A be PSD. Fix $r \in \mathbb{N}, \epsilon > 0$. Then, the rank-$k$ column Nyström approximation $\hat{A}^{(k)}$ produced by RPCholesky (with block size 1, i.e. **Algorithm 1**) satisfies the following bound:*

$$\mathbb{E}tr(A - \hat{A}^{(k)}) \leq (1 + \epsilon) \cdot tr(A - [[A]]_r)$$

when the number of columns k satisfies the following inequality:

$$k \geq \frac{r}{\epsilon} + \min\left\{ r \log\left(\frac{1}{\epsilon\eta}\right), r + r\max\left\{ \log\left(\frac{2^r}{\epsilon}\right), 0 \right\} \right\}$$

Algorithm 2 Block RPCholesky

Input: PSD matrix $\boldsymbol{A} \in \mathbb{C}^{N \times N}$; block size T; tolerance η or approximation rank k which is a multiple of T
Output: Pivot set S; matrix $\boldsymbol{F}$ defining Nyström approximation $\widehat{\boldsymbol{A}} = \boldsymbol{F}\boldsymbol{F}^*$
Initialize $\boldsymbol{F} \leftarrow \boldsymbol{0}_{N \times k}$, $\mathrm{S} \leftarrow \emptyset$, and $\boldsymbol{d} \leftarrow \operatorname{diag} \boldsymbol{A}$ ▷ Evaluate diagonal of input matrix
for $i = 0$ to $k/T - 1$ **do** ▷ Alternatively, run until $\sum_{j=1}^{N} \boldsymbol{d}(j) \leq \eta \operatorname{tr} \boldsymbol{A}$
  Sample $s_{iT+1}, \dots, s_{iT+T} \overset{\text{iid}}{\sim} \boldsymbol{d} / \sum_{j=1}^{N} \boldsymbol{d}(j)$ ▷ Probability prop. to diagonal
  $\mathrm{S}' \leftarrow \textsc{Unique}(\{s_{iT+1}, \dots, s_{iT+T}\})$
  $\mathrm{S} \leftarrow \mathrm{S} \cup \mathrm{S}'$
  $\boldsymbol{G} \leftarrow \boldsymbol{A}(:, \mathrm{S}')$ ▷ Evaluate columns S' of input matrix
  $\boldsymbol{G} \leftarrow \boldsymbol{G} - \boldsymbol{F}\boldsymbol{F}(\mathrm{S}', :)^*$ ▷ Remove overlap with prev. columns
  $\boldsymbol{R} \leftarrow \textsc{Chol}(\boldsymbol{G}(\mathrm{S}', :) + \varepsilon_{\text{mach}} \operatorname{tr}(\boldsymbol{G}(\mathrm{S}', :))\mathbf{I})$ ▷ Stabilized Cholesky $\boldsymbol{G}(\mathrm{S}', :) \approx \boldsymbol{R}^* \boldsymbol{R}$
  $\boldsymbol{F}(:, iT + 1 : iT + |\mathrm{S}'|) \leftarrow \boldsymbol{G}\boldsymbol{R}^{-1}$ ▷ Update approximation
  $\boldsymbol{d} \leftarrow \boldsymbol{d} - \textsc{SquaredRowNorms}(\boldsymbol{G}\boldsymbol{R}^{-1})$ ▷ Track diagonal of residual matrix
  $\boldsymbol{d} \leftarrow \max\{\boldsymbol{d}, \boldsymbol{0}\}$ ▷ Ensure diagonal remains nonnegative
end for
Remove zero columns from $\boldsymbol{F}$

This is the main theorem of the paper, and there are a few key takeaways. Firstly, for any pair of $r \in \mathbb{N}$ and $\epsilon > 0$, this theoretical bound holds for all steps k. This replaces the vacuous statement from **Theorem 2** in the case that $T = 1$. We also do not need any prior knowledge of these parameters, allowing for a simpler implementation than the Block RPCholesky for similar guarantees.

The lower bound on k can be split into two conditions, both of which are sufficient. In the first case, if we have $k \geq \frac{r}{\epsilon} + r \log(1/\epsilon\eta)$, this gives us a linear dependence on r and is comparable to the bound $k \geq \frac{r}{\epsilon}$, as per the original paper. The second bound, $k \geq \frac{r}{\epsilon} + r + r \log(2^r/\epsilon)$ gives us a quadratic dependence on r but also shows us that convergence is not dependent on the tolerance parameter. Chen et al. also suggest that the quadratic dependence may be a symptom of the proof technique, rather than a concrete diagnosis of the algorithm.

This proof also gives a similar bound for randomly pivoted QR, which is promising for randomized numerical linear algebra method as a whole, although out of the scope of this review.

Given this theoretical bound, alongside the given features of **Algorithm 1**, we are able to see that RP Cholesky has the four desirable properties outlined in the original paper:

1. **Cheap entry evaluations.** We only need $O(kn)$ entries of the kernel matrix to compute a rank k approximation.
2. **Low time and storage complexity.** The method takes $O(k^2 n)$ times in terms of arithmetic operations. Furthermore, the storage cost is $O(kn)$, since the Nyström factor F has kn entries.
3. **Approximation quality.** The rank-k Nyström approximation produced by RPCholesky is competitive with the optimal rank r approximation, where $r \leq k$ but not by a significant factor.

4. **Simplicity.** RPCholesky is easy to implement, works for any suitable input, and does not require the user to modify parameters or hyperparameters.

These conditions are excellent for widespread use of RPCholesky, and pose significiant benefits for kernel methods. The random pivot selection further increases the advantages of this method over pivoted partial Cholesky with traditional methods like greedy and uniform selection, as will be demonstrated with numerical experiments in the next section.

4 Numerical Experiments

For this section, we begin with a brief implementation of the standard RPCholesky algorithm in Python, which is given below:

```
def RPCholesky(A: np.array, k: int):
'''
Args:
    A: SPD nxn matrix with complex entries
    k: Approximation rank
Outputs:
    S: set of k randomly chosen pivots
    F: nxk Nystroem approximation of A
'''
assert A.shape[0] == A.shape[1], "A must be square"
n = A.shape[0]
F = np.zeros((n, k), dtype= A.dtype)
d = np.diag(A)
S = []
for i in range(k):
    S.append(np.random.choice(range(n), p=d/np.sum(d)))
    g = A[:, S[i]]
    g = g - F[:,:i] @ np.conj(F[S[i], :i]).T
    F[:, i] = g / np.sqrt(g[S[i]])
    d = d - np.abs(F[:, i]) ** 2
    np.maximum(d,0,d)
return F, S
```

The rather concise implementation of this method makes it highly flexible for applications, allowing users and researchers to leverage it with neither a significant investment in implementing it nor significant time in tuning hyperparameters, thanks to the bounds established in **Theorem 3**. The key parameter to be tuned is the approximation rank, which can be numerically verified for many matrices with well-known structure. One example of such matrices is Hankel matrices with spectral decay, which will be analyzed in the next section.

4.1 Alternate Pivoting Strategies

To demonstrate some of the tangible numerical gains of the RPCholesky method, we first present a comparison of RPCholesky against pivoted partial Cholesky with other selection strategies, namely uniform and greedy selection. For this task, similarly to the original paper, we have chosen a smiley face kernel and a spiral kernel to highlight cases where RPCholesky captures a more faithful representation of the data.

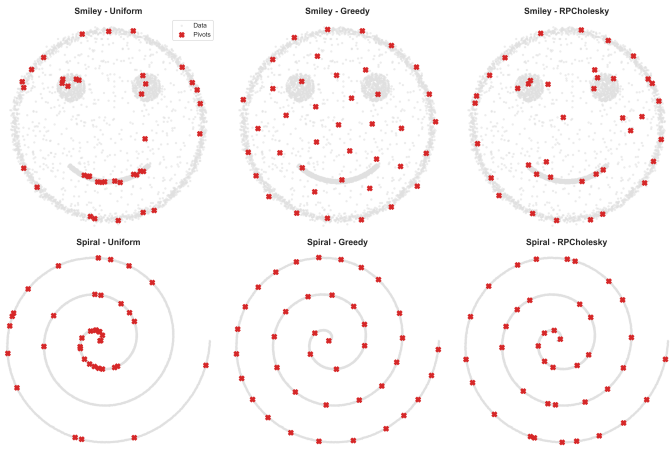

Fig. 1. Pivot selection on smiley face and spiral kernels from uniform, greedy, and RPCholesky strategies

For the smiley face, we notice that the greedy strategy does not choose points that entirely capture the features of the image. The uniform strategy has a better outcome, but the selections are more concentrated around the smile and eyes, which is reasonable due to the increased density of points in those regions. However, the RPCholesky method is able to balance selecting the high-density features, like the smile and eyes, while also sampling the sparser but more significant features like the outline of the face that the greedy algorithm prioritized.

As for the spiral, the uniform selection strategy did not produce particularly noteworthy results; the greedy algorithm chose roughly equispaced points on the spiral, but this gave higher weight to the outermost points and failed to capture finer detail like curvature. These shortcomings were avoided for the most part by RPCholesky; the algorithm was able to select many distinctive features but also chose more points at areas with more curvature, remaining more faithful to the features of the spiral.

Test cases such as these are particularly compelling for the use of sampling proportional to the diagonal, as the stochasticity introduced can avoid the pitfalls

of greedy methods in cases such as these while allowing for more representative data than a purely uniform strategy. It is important to note that the greedy algorithm allows for some marginal accuracy to be gained for most random matrices, i.e. those without particular special structure or spectral decay, as discovered from a few numerical tests. However, for those matrices which have structure that can be exploited, particularly PSD matrices with spectral decay, RPCholesky comes out ahead. Particularly for kernel methods, like kernel ridge regression, the ability for RPCholesky to capture features without succumbing to either of these shortcomings is a plausible explanation for its success over other sampling methods in the original paper.

4.2 Experimental Testing of Theoretical Bounds

To verify the theoretical bounds from **Theorem 2** and **Theorem 3**, we introduce a Python implementation of block RPCholesky:

```
def BlockRPCholesky(A: np.array, k: int, T: int, nu: float):
'''
Args:
    A: SPD nxn matrix with complex entries
    k: Approximation rank
    T: Number of blocks
    nu: Tolerance for stopping criterion

Outputs:
    F: nxk Nystroem approximation of A
    S: set of k randomly chosen pivots
'''

assert k % T == 0, "k must be divisible by T"
assert nu > 0, "nu must be positive"
assert A.shape[0] == A.shape[1], "A must be square"
n = A.shape[0]
eps = 1e-16
F = np.zeros((n, k), dtype=A.dtype)
d = np.diag(A)
S = []
print(F.shape)
for i in range(k // T - 1):
    Sp = np.random.choice(range(n), size=T, p=d/np.sum(d))
    Sp = np.unique(Sp).tolist()
    S.extend(Sp)
    G = A[:, Sp]
    G = G - F @ np.conj(F[Sp, :]).T
    shift = eps * np.linalg.trace(G[Sp, :]) * np.eye(len(Sp))
    R = np.linalg.cholesky(G[Sp, :] + shift, upper=True)
```

```
        F[:, i * T:i * T + len(Sp)] = np.linalg.solve(R.T, G.T).T
        d = d - np.sum(np.abs(F[:, i * T:i * T + len(Sp)]) ** 2, axis=1)
        np.maximum(d, 0, d)
        if np.sum(d) < nu:
            break
    F = F[:, ~np.all(F == 0, axis=0)]
    return F, S
```

To demonstrate the error bound as indicated by the literature in the prior theorems, we've implemented a Python method that plots the trace-norm error at several values of k for RPCholesky alongside the theoretical bound, with a fixed $r = 50$ for an RBF kernel:

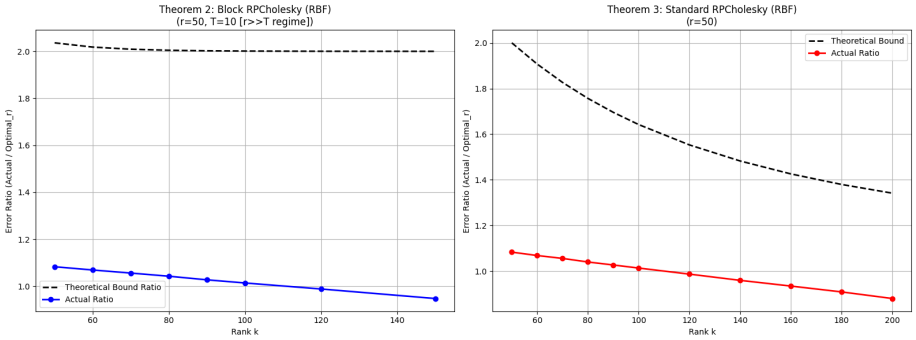

Fig. 2. Empirical testing of error bounds established by **Theorem 2** and **Theorem 3**. Plots RPCholesky approximation rank k against ratio of trace-norm error to best rank-r trace-norm error.

An initial survey of the results reveals that the theoretical bound established by Theorem 2 for block RPCholesky is not particularly tight. As indicated by the graph, the first few iterations (approximately 5, since T must divide k) give a fair estimate with about 10% trace-norm error in excess. A shortfall of this, however, is that for small epsilon, we have that the required block size and consequently approximation rank k grows dramatically. This is a consequence of the restriction that $T = r/\epsilon$; the above figure uses a presumed $\epsilon = 5$, and requiring further precision will increase the looseness of this bound.

However, for the empirical testing of **Theorem 3**, it can be seen that the bound is much tighter; here, the values of k are inferred from r and a varying ϵ parameter to fit the assumptions of the theorem. While the bound does remain somewhat loose, this may be a result of the rank of the matrix being used and can benefit from more numerical testing. Nonetheless, it is clear that Chen et al. provide a much tighter bound; as stated in the original paper, however, the experiments indicate that a bound tighter still may be achieved due to limitations of the proof technique in the original paper.

5 RPCholesky for SPD Hankel Matrices

In a recent paper from the Hazan Lab at Princeton University, Liu et al. (2025) proposed Flash Spectral Transform Units (STUs), which leverage spectral filters formed from the dominant eigenvectors of a Hankel matrix for long-term sequence modeling, providing several advantages over the traditional transformer architecture. While the Hankel matrix is fixed before training, computing its dominant eigenvectors is expensive with the standard np.linalg.eigh method; if $H \in \mathbb{R}^{n\times n}$, the standard method is $\mathcal{O}(n^3)$. While this computational tradeoff is acceptable when the Hankel eigenvectors need to be computed once, before training, this precludes the efficient recomputation of the Hankel matrix during inference, which may prove promising for model performance in linear dynamical systems, the focus of the original paper. Thus, we propose an RPCholesky improvement; real Hankel matrices exhibit significant spectral decay, providing prime conditions to apply RPCholesky.

To this end, we have implemented an implicit RPCholesky method that can decompose a Hankel matrix with the required properties:

Algorithm 3 Implicit Hankel RPCholesky

Input: PSD Hankel matrix $\mathbf{H} \in \mathbb{R}^{n\times n}$, represented by a vector $\mathbf{h} \in \mathbb{R}^{2n-1}$; approximation rank k
Output: Pivot set $S = \{s_1, \ldots, s_k\}$; matrix $\mathbf{F} \in \mathbb{R}^{n\times k}$ such that $\hat{\mathbf{H}} = \mathbf{F}\mathbf{F}^*$
Initialize: $\mathbf{F} \leftarrow \mathbf{0}_{n\times k}$, $\mathbf{d} \leftarrow \mathbf{h}[::2]$
for $i = 1$ to k **do**
  Sample pivot $s_i \sim \boldsymbol{d}/\sum_{j=1}^{N} \boldsymbol{d}(j)$ ▷ Probability proportional to residual diagonal
  $\boldsymbol{g} \leftarrow \boldsymbol{h}(s_i, s_i + n)$ ▷ Evaluate column s of input matrix
  $\boldsymbol{g} \leftarrow \boldsymbol{g} - \boldsymbol{F}(:, 1:i-1)\boldsymbol{F}(s_i, 1:i-1)^*$ ▷ Remove overlap with previous columns
  $\boldsymbol{F}(:, i) \leftarrow \boldsymbol{g}/\sqrt{\boldsymbol{g}(s_i)}$ ▷ Update approximation
  $\boldsymbol{d} \leftarrow \boldsymbol{d} - |\boldsymbol{F}(:, i)|^2$ ▷ Track diagonal of residual matrix
  $\boldsymbol{d} \leftarrow \max\{\boldsymbol{d}, \mathbf{0}\}$ ▷ Ensure diagonal remains nonnegative
end forr = |S| $\mathbf{F} = \mathbf{F}[:, :r]$

This is very similar to **Algorithm 1**, except we initialize only using a vector defining the entire Hankel matrix. The entries of this vector are the first column and the last row of the Hankel matrix, where they coincide at entry n of the vector. One apparent drawback is that instead of encoding the Hankel matrix in a vector with storage $\mathcal{O}(n)$, the resulting factor $\mathbf{F}$ takes up $\mathcal{O}(kn)$; however, in the context of FlashSTU, k is sufficiently small (typically $\tilde{2}4$) to make this storage effectively $\mathcal{O}(n)$.

There are two key applications of this method in this context; low-rank approximation of the Hankel matrix and faster eigenvector finding for the spectral filters used in the FlashSTU architecture. For the low-rank approximation, RPCholesky is able to compute an approximation incredibly quickly due to the relatively low rank of the Hankel matrices. However, it is important to note that we can

precisely calculate each entry of the Hankel matrix from its defining vector, so this may in fact be counterproductive. Should we need to repeatedly act on the Hankel matrix with other linear operators, the Nystöm factors may speed up computation, but it is reasonable to expect that we can exploit the Hankel structure for fast matmul similar to how it can be used for fast matvec.

However, we do see significant runtime gains when attempting to get a full eigendecomposition of the Hankel matrix; in particular, when running with a matrix in $\mathbb{R}^{n\times n}$ with $n \approx 10,000$, np.linalg.eigh takes approximately two minutes on an M4 Max CPU while running np.linalg.svd on the Nyström factor $\mathbf{F}$ and computing an estimate to the eigenvalues and eigenvectors from there takes approximately 0.5 seconds. This is a significant gain, but there are two key considerations: the accuracy and the efficacy of other methods.

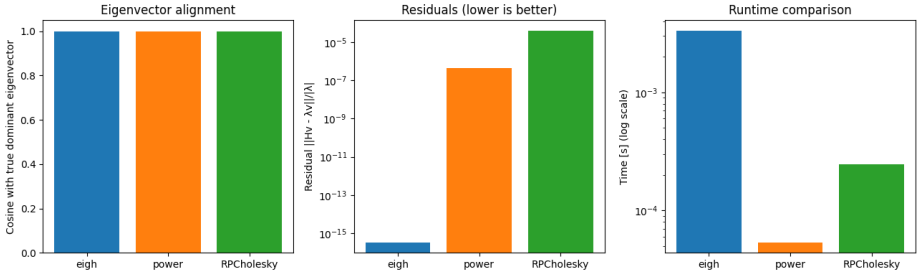

Fig. 3. Eigenvector accuracy (cosine similarity to true dominant eigenvector), residual, and runtime for np.linalg.eigh, Power method, and RPCholesky decomposition method

In terms of speed, we see that while RPCholesky dominates the built-in eigenvalue solving method (which is unoptimized) using the power and inverse methods is dramatically faster, due to the efficient matvec with the Hankel matrix. Not only that, but convergence is much nicer using the tighter theoretical bounds, as the Hankel matrix has a larger distance between its eigenvalues as a result of spectral decay. This, unfortunately, means that RPCholesky is not quite applicable to the eigenvalue and eigenvector problem in a practical sense. However, we can still examine its effect on the spectrum of the Hankel matrix, which is seen on the next page.

First, we note that the dominant eigenvector is very faithful to the original; this high fidelity is promising for using the spectra of an RPCholesky-generated Nyström approximation, and this may open up the opportunity for RPCholesky to be used to calculate a dominant eigenvector for matrices that are very poorly conditioned for the power method or inverse method. However, it is likely that such method already exist without the storage burden or stochasticity that are inherent to the RPCholesky factorization. From there, the eigenvector accuracy drops off dramatically, with the third vector having a 0.09 cosine similarity.

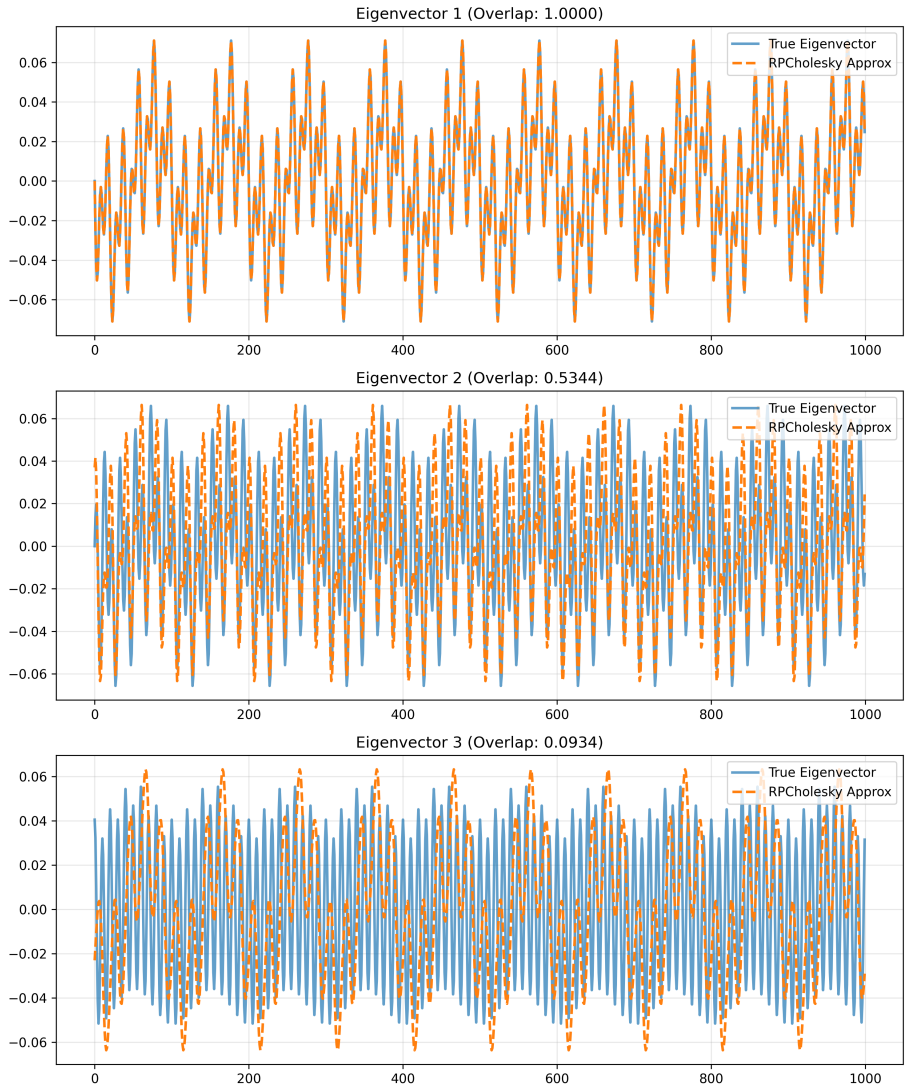

Fig. 4. RPCholesky approximation of Hankel matrix spectra versus built-in method np.linalg.eigh

6 Conclusion

This report investigated the Randomly Pivoted Cholesky (RPCholesky) algorithm, validating its theoretical guarantees and exploring its application across diverse domains, from standard kernel methods to structured linear dynamical systems. The numerical experiments confirm that RPCholesky is able to bridge between the efficiency of uniform sampling and the quality of greedy sampling, while avoiding their pitfalls in the "smiley" and "spiral" kernel examples.

A key contribution of this report was the novel application of RPCholesky to Hankel matrices within the FlashSTU architecture. While the results indicated that RPCholesky did not outperform the Power Method for eigenvector computation, this result offers insight into the algorithm's optimal use cases. RPCholesky performs exceptionally well in settings involving dense kernel matrices, as shown in the original paper, where entry evaluation is cheap but full matrix operations are expensive ($\mathcal{O}(n^2)$). However, for structured matrices like Hankel matrices which admit cheap matvec functions, the overhead of constructing a Nyström factor outweighs the benefits of the factorization. The results of the experimentation with Hankel matrices highlight that the RPCholesky algorithm is best-suited for data compression in dense kernel matrices rather than spectral problems in structured operators.

It is also important to note that the method still sees success in its applications to kernel methods, like kernel spectral clustering and kernel ridge regression, as emphasized in the original paper. However, given the Hankel application, there is much to be explored in terms of using the RPCholesky for specially structured matrices and leveraging other algorithms like the nonuniform FFT and specialized matvec operations to develop quick and simple factorizations of complicated kernel matrices.

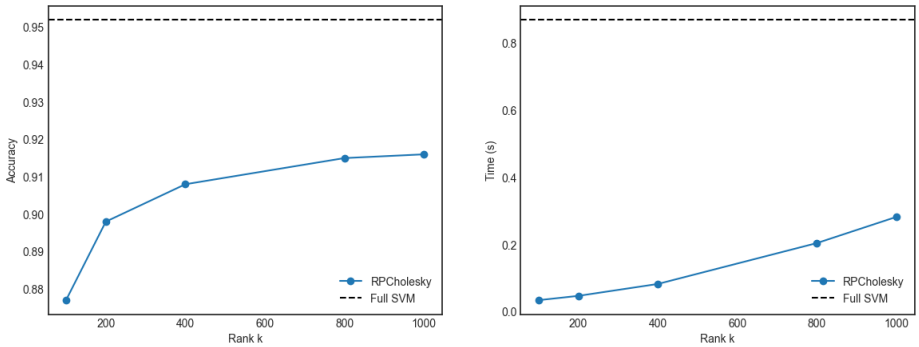

Fig. 5. RPCholesky vs. SVM for MNIST classification, in terms of time and accuracy

The machine learning implications in particular are promising; in the typical MNIST digit classifcation task, RPCholesky was able to produce similar accuracy

as the SVM method with a much lower time complexity. Although this accuracy plateaud and would only reach the SVM accuracy when the rank of the approximation matched the rank of the matrix, there are potential applications in terms of boosting algorithms that can use an ensemble of these "RPCholesky learners," which are not as accurate as a strong learner but would prove easier to boost than a weak learner. However, particular machine learning applications should be explored with more focus in another paper, and it would be prudent to particularly focus on more robust kernel and clustering methods that RPCholesky has proven to perform adequately in.

Overall, RPCholesky is a useful algorithm with clear applications to scientific computing, with its random aspects providing clear value in specific cases that the greedy and uniform methods cannot consistently match. While the theoretical bounds on the algorithm have room to improve, this is promising for the practical applications of the algorithm for a quick, efficient low-rank approximation for PSD matrices that may not be nice enough for other factorization methods.

Acknowledgments

I acknowledge the contribution of LLMs to this report, particularly Gemini 3 Pro via Cursor for generation of numerical tests and graphics used in this report. I also acknowledge consultation with the Github repositories associated with the original RPCholesky and FlashSTU papers, found at `github.com/eepperly/randomly-pivoted-cholesky/` and `github.com/hazan-lab/flash-stu/` respectively.

References

1. Chen, Y., Epperly, E., Tropp, J., Webber, R.: Randomly pivoted Cholesky: Practical approximation of a kernel matrix with few entry evaluations. (2024). https://arxiv.org/abs/2207.06503.
2. Y. Isabel Liu, Windsor Nguyen, Yagiz Devre, Evan Dogariu, Anirudha Majumdar, Elad Hazan: Flash STU: Fast Spectral Transform Units. (2025). https://arxiv.org/abs/2409.10489.

Further Results

A Jupyter notebook summarizing a few of the main experiments can be found at `https://colab.research.google.com/drive/1L6TusyctiyxKO-AWu6xsTkyvp5TKdJEi?usp=sharing`.