Comparative Analysis of Algorithms for Sparse PCA

Chaeeun Ryu Rebeka Bojboi Rayaan Attari

May 13, 2025

Abstract

Sparse Principal Component Analysis (Sparse PCA) seeks to identify a low-dimensional representation of data using a small subset of the original features, balancing interpretability and variance preservation. In this report, we implement and compare five algorithms for the k-Sparse PCA problem: brute-force search, greedy forward selection, thresholding, random sampling, and DSPCA with rounding. These methods span a range of algorithmic paradigms—from exact combinatorial search to convex relaxation and randomized heuristics. We evaluate each approach on synthetic datasets under varying sparsity levels and feature dimensionalities. Our experiments highlight the trade-offs between computational efficiency and explained variance. While brute-force offers optimal solutions for small-scale problems, approximate methods such as greedy selection and DSPCA deliver strong performance with significant scalability. The results validate known theoretical insights and offer practical guidance for applying Sparse PCA in different settings.

1 Introduction

Principal Component Analysis (PCA) is a fundamental technique in statistics and machine learning for reducing the dimensionality of high-dimensional data while preserving as much variance as possible. Given a zero-mean data matrix $X \in \mathbb{R}^{n \times p}$, standard PCA seeks to find a direction $v \in \mathbb{R}^p$ that maximizes the variance of the projected data Xv, subject to the constraint $\|v\|_2 = 1$. The solution is given by the leading eigenvector of the empirical covariance matrix $A = X^\top X$.

However, the principal components obtained by PCA are typically dense, i.e., all entries of v may be nonzero. In applications such as gene expression analysis, financial modeling, and image processing, interpretability and computational efficiency are improved by enforcing sparsity in the principal components. This motivates the study of Sparse PCA (SPCA), where the goal is to find a principal component that has only a small number of nonzero entries.

Formally, the k-Sparse PCA problem can be stated as the following non-convex optimization problem:

$$\begin{aligned} \max_{v \in \mathbb{R}^p} \quad & v^\top A v \\ \text{subject to} \quad & \|v\|_2 = 1, \quad \|v\|_0 \leq k, \end{aligned}$$

where $\|v\|_0$ denotes the number of nonzero entries in v. This problem is known to be NP-hard, and approximating it to within any constant factor is Small Set Expansion (SSE)-hard in the worst case.

To address the intractability of the exact problem, a range of heuristic and approximation algorithms have been proposed in the literature. These include greedy selection methods, spectral heuristics based on thresholding, randomized sampling approaches, and convex relaxations such as semidefinite programming (SDP). Some methods offer theoretical guarantees on variance explained or approximation quality, while others are motivated by empirical performance.

In this project, we investigate and compare the following five algorithmic approaches to the k-Sparse PCA problem:

1. **Brute-Force Search:** Enumerates all $\binom{p}{k}$ possible supports of size k and computes the top eigenvector of the corresponding submatrix to identify the optimal sparse component.

2. **Greedy Algorithm:** Builds the support set iteratively by selecting the feature that leads to the greatest marginal increase in variance at each step.

3. **Thresholding Method:** Computes the top eigenvector of A and retains the k coordinates with largest absolute values.

4. **Random Sampling:** Randomly samples subsets of k coordinates and evaluates the leading eigenvalue of the associated submatrix to approximate the best support.

5. **DSPCA with Rounding:** Solves the semidefinite relaxation proposed by d'Aspremont et al. d'Aspremont et al. [2007] and applies a randomized rounding procedure to recover a k-sparse vector.

We implement each algorithm and evaluate their performance on synthetic datasets by measuring the explained variance, sparsity of the solution, and runtime. The rest of this report is organized as follows: Section 2 introduces the formal problem statement and background. Section 3 describes the algorithms in detail. Section 4 outlines our experimental setup, and Section 5 presents the empirical results. We conclude in Section 6 with a discussion and future directions.

2 Problem Formulation and Background

Let $X \in \mathbb{R}^{n\times p}$ denote a zero-mean data matrix, where each row corresponds to an observation and each column to a feature. The empirical covariance matrix is defined as $A := X^\top X \in \mathbb{R}^{p\times p}$. Classical PCA seeks a unit-norm vector $v \in \mathbb{R}^p$ that maximizes the variance of the projected data, i.e.,

$$\max_{v\in\mathbb{R}^p} \quad v^\top A v \quad \text{subject to} \quad \|v\|_2 = 1,$$

which has a closed-form solution given by the leading eigenvector of A.

Sparse PCA modifies this formulation by imposing an additional sparsity constraint on v, typically by limiting the number of nonzero entries. The most common variant is the k-Sparse PCA problem, defined as:

$$\begin{aligned} \max_{v\in\mathbb{R}^p} \quad & v^\top A v \\ \text{subject to} \quad & \|v\|_2 = 1, \quad \|v\|_0 \le k, \end{aligned} \tag{1}$$

where $\|v\|_0$ denotes the ℓ_0 pseudo-norm (i.e., the number of nonzero entries in v), and $\|v\|_2$ is the standard Euclidean norm. This formulation seeks the unit-norm vector supported on at most k coordinates that maximizes the variance of the projected data.

Computational Hardness

The problem in Eq. (1) is combinatorial and non-convex. It has been shown to be NP-hard, even to approximate within any constant factor Chan et al. [2016]. In fact, under standard complexity assumptions, no polynomial-time algorithm can approximate the optimal explained variance to within a multiplicative factor unless P = NP. Moreover, the problem is SSE-hard (Small Set Expansion), further implying that the hardness persists even under relaxed assumptions.

Alternative Formulations

Several convex and non-convex relaxations of Sparse PCA have been proposed in the literature. One notable approach is the semidefinite programming (SDP) relaxation introduced by d'Aspremont et al. d'Aspremont et al. [2007], which reformulates the problem as:

$$\max_{X \in \mathbb{R}^{p \times p}} \quad \mathrm{Tr}(AX) \quad \text{subject to} \quad \mathrm{Tr}(X) = 1, \quad \|X\|_1 \le k, \quad X \succeq 0,$$

where X is a rank-one approximation to $vv^\top$, and the ℓ_1 norm $\|X\|_1 = \sum_{i,j} |X_{ij}|$ promotes sparsity. This relaxation can be solved efficiently via convex optimization, and a sparse vector can be extracted from the solution using randomized rounding or support thresholding.

Other heuristic approaches include greedy forward selection, thresholding the leading eigenvector, and random sampling of candidate supports. While these methods lack worst-case guarantees, they are often effective in practice and scale well to larger problem sizes.

Evaluation Metric

Across all methods, the primary performance metric is the proportion of variance explained by the sparse component. Given a solution vector v, this is quantified as $v^\top Av$, normalized by the leading eigenvalue λ_1 of A. That is, we compute the explained variance ratio:

$$\text{Explained Variance Ratio} = \frac{v^\top Av}{\lambda_1}.$$

This metric allows for fair comparison across algorithms, regardless of their sparsity strategies or assumptions.

3 Algorithms and Methods

We implement and evaluate five distinct algorithms for solving the k-Sparse Principal Component Analysis (PCA) problem. These methods span the spectrum from exact solutions to efficient approximations, including exhaustive search approaches, greedy algorithms, spectral methods, randomized techniques, and convex relaxations. Each algorithm offers different trade-offs between computational efficiency and solution quality. Below, we provide a detailed description of each method, highlighting their theoretical foundations, implementation details, computational complexity, and the nature of the solutions they provide.

3.1 Brute-Force Search

The brute-force method guarantees an exact solution by systematically evaluating all possible sparse supports of size k. This exhaustive approach serves as a ground truth benchmark for assessing the performance of more efficient approximate methods.

For each possible subset $S \subseteq \{1, \ldots, p\}$ with cardinality $|S| = k$, we extract the corresponding principal submatrix A_S from the covariance matrix A, and compute its leading eigenvector and associated eigenvalue. The eigenvalue represents the variance explained by the projection of the data onto the subspace defined by the support S.

Formally, for each subset S, we solve:

$$\lambda_1^{(S)} := \max_{\substack{v \in \mathbb{R}^p \\ \mathrm{supp}(v) \subseteq S \\ \|v\|_2 = 1}} v^\top A v = \lambda_1(A_S),$$

where $\lambda_1(A_S)$ denotes the largest eigenvalue of the submatrix A_S. The optimal solution is the pair (S^*, v^*) that achieves the maximum eigenvalue across all subsets.

Algorithm 1 Brute-Force Search for k-Sparse PCA

```
1: **Input:** Covariance matrix $A \in \mathbb{R}^{p \times p}$, sparsity level $k \in \mathbb{N}$
2: **Output:** Leading sparse component $v^*$ and explained variance $\lambda^*$
3: Initialize $\lambda^* \leftarrow -\infty$, $v^* \leftarrow \mathbf{0}$
4: **for** each subset $S \subseteq \{1, \ldots, p\}$ with $|S| = k$ **do**
5:     Extract submatrix $A_S \leftarrow A[S, S]$
6:     Compute leading eigenvector $v_S$ and eigenvalue $\lambda_S \leftarrow v_S^\top A_S v_S$
7:     **if** $\lambda_S > \lambda^*$ **then**
8:         Update $\lambda^* \leftarrow \lambda_S$, $v^* \leftarrow v_S$
9:     **end if**
10: **end for**
11: **return** $v^*, \lambda^*$
```

Computational Complexity: The algorithm requires evaluating $\binom{p}{k}$ different subsets. For each subset, we compute the principal eigenvector of a $k \times k$ matrix, which takes $O(k^3)$ time using standard eigendecomposition techniques. This results in an overall time complexity of $O\left(\binom{p}{k} \cdot k^3\right)$, which quickly becomes infeasible for large p or even moderate values of k.

Despite its computational cost, the brute-force approach is critical in our study as it provides a reliable reference point for evaluating the quality of approximate solutions. To make this approach tractable in small-to-moderate dimensions, we implement a parallelized version that distributes the subset evaluations across multiple CPU cores. This exact algorithm is inspired by problem formulations presented in d'Aspremont et al. d'Aspremont et al. [2007] and serves as a foundational baseline for our comparative analysis.

3.2 Greedy Forward Selection

The greedy forward selection algorithm incrementally constructs a sparse support set by adding one coordinate at a time. Starting from the empty set, it iteratively selects the feature that offers the greatest marginal increase in the variance of the projected data. This approach significantly reduces computational burden compared to exhaustive search, while often yielding near-optimal solutions in practice.

Let S_t denote the partial support set after t iterations, where $|S_t| = t$. At iteration $t + 1$, the algorithm selects the coordinate $j \notin S_t$ that maximizes the leading eigenvalue of the submatrix $A_{S_t \cup \{j\}}$:

$$j^* = \arg\max_{j \notin S_t} \lambda_1 \left(A_{S_t \cup \{j\}} \right).$$

The support is then updated as $S_{t+1} = S_t \cup \{j^*\}$, and the process continues until $|S_k| = k$. The final sparse principal component is the leading eigenvector of A_{S_k}, normalized to unit ℓ_2 norm.

Algorithm 2 Greedy Forward Selection for k-Sparse PCA

```
1: Input: Covariance matrix $A \in \mathbb{R}^{p \times p}$, sparsity level $k \in \mathbb{N}$
2: Output: Greedy approximate component $v$ and explained variance $\lambda$
3: Initialize selected indices $S \leftarrow \emptyset$, $\lambda \leftarrow 0$
4: for $t = 1$ to $k$ do
5:     Initialize $\lambda_{\text{best}} \leftarrow \lambda$, $j_{\text{best}} \leftarrow$ null
6:     for each $j \in \{1, \dots, p\} \setminus S$ do
7:         Let $T \leftarrow S \cup \{j\}$
8:         Extract submatrix $A_T \leftarrow A[T, T]$
9:         Compute leading eigenvalue $\lambda_T \leftarrow \lambda_1(A_T)$
10:        if $\lambda_T > \lambda_{\text{best}}$ then
11:            Update $\lambda_{\text{best}} \leftarrow \lambda_T$, $j_{\text{best}} \leftarrow j$
12:        end if
13:    end for
14:    Update $S \leftarrow S \cup \{j_{\text{best}}\}$, $\lambda \leftarrow \lambda_{\text{best}}$
15: end for
16: Compute leading eigenvector $v$ of $A[S, S]$ and pad with zeros to form $v \in \mathbb{R}^p$
17: return $v, \lambda$
```

Computational Complexity: At each of the k iterations, the algorithm evaluates at most p candidate features. For each candidate, it computes the leading eigenvalue of a $(t+1) \times (t+1)$ matrix, which requires $O((t+1)^3)$ operations. Summing over all iterations results in an overall complexity of:

$$O\left(p \sum_{t=0}^{k-1} (t+1)^3\right) = O(pk^4).$$

This method is adapted from the greedy heuristic proposed by d'Aspremont et al. d'Aspremont et al. [2007] and follows similar principles to forward selection methods used in classical variable selection. While it lacks worst-case approximation guarantees, its efficiency and simplicity make it a competitive baseline for sparse PCA in practical settings.

3.3 Thresholding Method

The thresholding method is an extremely efficient spectral heuristic motivated by a simple yet powerful observation: the coordinates of the leading principal component (i.e., the top eigenvector of the covariance matrix) with the largest magnitudes often correspond to the most informative features in the sparse setting.

The algorithm proceeds as follows:

1. Compute the leading eigenvector v_1 of the full covariance matrix A.

2. Identify the indices of the k coordinates with the largest absolute values in v_1, denoted as Top-$k(|v_1|)$.

3. Construct a sparse approximation $\hat{v}$ by retaining only these k coordinates and zeroing out the rest:
$$\hat{v}_i = \begin{cases} (v_1)_i & \text{if } i \in \text{Top-}k(|v_1|), \\ 0 & \text{otherwise,} \end{cases}$$

4. Normalize the result to unit ℓ_2 norm: $v = \hat{v}/\|\hat{v}\|_2$.

Algorithm 3 Thresholding Method for k-Sparse PCA

```
1: **Input:** Covariance matrix $A \in \mathbb{R}^{p \times p}$, sparsity level $k \in \mathbb{N}$
2: **Output:** Thresholded sparse principal component $v \in \mathbb{R}^p$
3: Compute the leading eigenvector $v_1$ of $A$
4: Let $S \subseteq \{1, \ldots, p\}$ be the indices of the $k$ largest entries of $|v_1|$
5: Initialize $\hat{v} \leftarrow \mathbf{0} \in \mathbb{R}^p$
6: for each $i \in S$ do
7:     Set $\hat{v}_i \leftarrow (v_1)_i$
8: end for
9: Normalize: $v \leftarrow \hat{v}/\|\hat{v}\|_2$
10: return $v$
```

Computational Complexity: The dominant cost lies in computing the leading eigenvector of the $p \times p$ matrix A, which requires $O(p^2)$ time using standard power iteration methods. The thresholding and normalization steps require only $O(p)$ additional operations, making this method extremely efficient.

Due to its simplicity, scalability, and empirical effectiveness, the thresholding method is widely used in practice. It also serves as a foundational component in several provable approximation algorithms for sparse PCA. In particular, it has been theoretically analyzed by Chan et al. Chan et al. [2016] and Berk and Bertsimas Berk and Bertsimas [2019], where it plays a central role in certifying approximate optimality under relaxed assumptions.

3.4 Random Sampling

The random sampling approach employs a Monte Carlo strategy to explore the exponentially large space of possible support sets efficiently. While conceptually simple, this method can yield surprisingly strong approximations with sufficiently many samples and has appealing probabilistic guarantees under mild assumptions.

The algorithm proceeds as follows:

1. Sample N distinct support sets $\{S_1, S_2, \ldots, S_N\}$ uniformly at random from the set of all k-sized subsets of $\{1, \ldots, p\}$.

2. For each sampled support S_i, extract the submatrix A_{S_i} and compute its leading eigenvalue $\lambda_1(A_{S_i})$ and corresponding eigenvector v_i.

3. Return the support set S^* and vector v^* that yield the highest explained variance among all samples.

Algorithm 4 Random Sampling for k-Sparse PCA

```
1: **Input:** Covariance matrix $A \in \mathbb{R}^{p \times p}$, sparsity level $k \in \mathbb{N}$, number of samples $s \in \mathbb{N}$
2: **Output:** Approximate variance $\lambda^*$ of the $k$-sparse principal component
3: Initialize $\lambda^* \leftarrow 0$
4: for $i \leftarrow 1$ to $s$ do
5:     Sample a support set $S \subseteq \{1, \ldots, p\}$ uniformly at random with $|S| = k$
6:     Extract submatrix $A_S \leftarrow A[S, S]$
7:     Compute top eigenvalue $\lambda_S \leftarrow \|A_S\|_2$
8:     Update $\lambda^* \leftarrow \max(\lambda^*, \lambda_S)$
9: end for
10: Return: $\lambda^*$
```

Computational Complexity: For each of the N randomly sampled support sets, the algorithm computes the leading eigenvalue of a $k \times k$ matrix, which takes $O(k^3)$ time. Thus, the total time complexity is $O(N \cdot k^3)$. The number of samples N acts as a fidelity parameter that trades off computational cost against approximation quality.

This method has been analyzed by Berk and Bertsimas Berk and Bertsimas [2019], who showed that with a sufficient number of independent samples, the probability of recovering a near-optimal support increases. Although the method lacks deterministic guarantees, its simplicity and probabilistic approximation bounds make it a practical and scalable alternative to more expensive approaches, particularly when a rough but fast solution is acceptable.

3.5 DSPCA with Rounding

The Direct Sparse PCA (DSPCA) method employs semidefinite programming (SDP) to obtain a convex relaxation of the original non-convex problem. This approach provides a principled upper bound on the optimal value and often yields high-quality sparse components after appropriate rounding.

We implement the semidefinite relaxation introduced by d'Aspremont et al. d'Aspremont et al. [2007]:

$$\begin{aligned} \max_{X \in \mathbb{R}^{p \times p}} \quad & \mathrm{Tr}(AX) \\ \text{subject to} \quad & \mathrm{Tr}(X) = 1, \quad \|X\|_1 \leq k, \quad X \succeq 0. \end{aligned}$$

Here, X is a positive semidefinite matrix that approximates the outer product $vv^\top$ of the desired sparse vector. The constraint $\|X\|_1 \leq k$ promotes sparsity, while $\mathrm{Tr}(X) = 1$ enforces the unit-norm constraint.

Since the optimal solution X^* may not be rank-one (which would correspond to a single vector v), we employ a rounding procedure to extract a sparse vector. We either:

1. Sample $v \sim \mathcal{N}(0, X^*)$ from a multivariate Gaussian distribution with covariance matrix X^*
2. Compute the eigendecomposition of X^* and use the leading eigenvector.

We then identify the k coordinates with the largest absolute values, retain only these coordinates, and normalize the resulting vector.

Algorithm 5 DSPCA with Rounding for k-Sparse PCA

```
1: Input: Covariance matrix $A \in \mathbb{R}^{p \times p}$, sparsity level $k \in \mathbb{N}$
2: Output: Approximate $k$-sparse principal component $v \in \mathbb{R}^p$
3: Solve the semidefinite relaxation:
       $\max_{X \in \mathbb{R}^{p \times p}} \operatorname{Tr}(AX) \quad \text{s.t.} \quad \operatorname{Tr}(X) = 1,\ \|X\|_1 \leq k,\ X \succeq 0$
4: Let $X^*$ be the optimal solution
5: Compute the leading eigenvector $v_1$ of $X^*$
6: Identify index set $S \subseteq \{1, \dots, p\}$ of the top $k$ entries (in absolute value) of $v_1$
7: Initialize $\hat{v} \leftarrow \mathbf{0} \in \mathbb{R}^p$
8: for each $i \in S$ do
9:     Set $\hat{v}_i \leftarrow (v_1)_i$
10: end for
11: Normalize: $v \leftarrow \hat{v}/\|\hat{v}\|_2$
12: return $v$
```

Computational Complexity: The dominant cost is solving the SDP, which requires $O(p^3)$ operations using interior-point methods. The subsequent rounding procedure adds only $O(p^2)$ operations.

This convex relaxation approach aligns with the relaxation and rounding pipeline proposed in d'Aspremont et al. d'Aspremont et al. [2007] and further analyzed in Berk and Bertsimas Berk and Bertsimas [2019], providing a principled framework with theoretical guarantees.

4 Experiments

In this section, we describe the experimental setup used to evaluate the performance of the five algorithms presented in Section 3. Our goals are to assess the explained variance achieved by each method as a function of the sparsity level k, compare their behavior across varying data dimensionalities, and examine how closely each approximation method approaches the brute-force optimum.

Dataset Generation

Following prior work Chatziafratis et al. [2021], we generate synthetic datasets by constructing random overcomplete dictionaries. Specifically, for a fixed number of rows $K = 30$ and varying number of columns $p \in \{2, 5, 10, 20\}$, we generate a matrix $D \in \mathbb{R}^{K \times p}$ with i.i.d. standard Gaussian entries. Each row of D is then normalized to unit ℓ_2 norm. We form the covariance matrix as:

$$B := \frac{DD^\top}{\|DD^\top\|_2},$$

where $\|\cdot\|_2$ denotes the spectral norm (largest eigenvalue). This construction ensures that B is symmetric and positive semidefinite with maximum eigenvalue 1, allowing fair comparisons across settings.

Algorithms Evaluated

We evaluate the following five algorithms:

- **Brute-Force Search** (exact baseline)
- **Greedy Forward Selection**
- **Thresholding Method**
- **Random Sampling**
- **DSPCA with Rounding**

The brute-force method computes the optimal k-sparse principal component by exhaustively enumerating all $\binom{p}{k}$ subsets of features. Due to the exponential cost of this approach, we restrict the maximum sparsity level to $k = 20$ in practice. To make brute-force computation feasible, we parallelize subset evaluation using `joblib`, as detailed in our implementation.

Each algorithm is implemented independently, and we ensure that the explained variance is non-decreasing with k by enforcing monotonicity during post-processing (i.e., replacing each value with the maximum of all previous values).

Evaluation Metric

For each method, we measure the variance explained by the returned k-sparse unit vector v on the normalized matrix B. To facilitate meaningful comparisons across datasets and algorithms, we normalize this value by the top eigenvalue λ_1 of B, which is 1 by construction. That is, we report the *explained variance ratio*:

$$\frac{v^\top Bv}{\lambda_1(B)}.$$

This ratio lies in the range $[0, 1]$ and allows us to assess how closely each algorithm approximates the optimal variance captured by standard (dense) PCA.

Experimental Parameters

Unless otherwise specified, all experiments are conducted with:

- Number of observations: $K = 30$
- Dimensionalities: $p \in \{2, 5, 10, 20\}$
- Sparsity levels: $k \in \{1, \ldots, 30\}$ (capped at $k = 20$ for brute-force)
- Random seed: 42 (for reproducibility)
- Number of random samples for sampling-based methods: $N = 200$

Implementation Details

All algorithms are implemented in Python 3 using NumPy and SciPy for matrix operations. Eigenvalue computations are performed using `numpy.linalg.eigh`, which exploits the symmetry of the input matrix. The DSPCA relaxation is implemented using CVXPY with the SCS solver. The brute-force implementation uses `joblib.Parallel` to process subsets in parallel. All experiments are executed on a standard laptop CPU without GPU acceleration.

5 Experimental Results

This section presents the results of our experiments comparing the performance of the greedy sparse PCA lower bound and the sampling-based lower bound under varying conditions of sparsity and data dimensionality.

5.1 Effect of Data Dimensionality (Brute-Force Method)

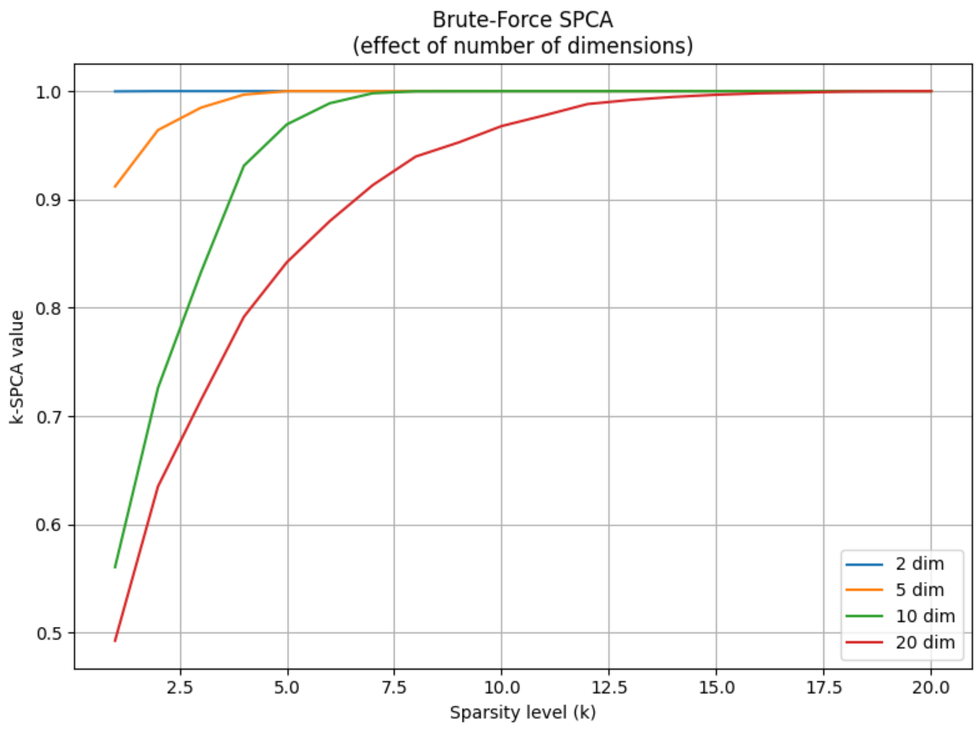

Figure 1: Brute-force SPCA value as a function of sparsity level for different data dimensionalities (2, 5, 10, and 20 dimensions). The plot illustrates how the optimal explained variance changes as sparsity increases across varying feature spaces.

Figure 1 presents the exact optimal variance achieved using the brute-force method, plotted against the sparsity level k for several data dimensionalities. As expected, the explained variance increases monotonically with k for all values of p. For low-dimensional data (e.g., 2 or 5 features), the maximum variance is captured quickly—often within a small number of active features—resulting in rapid early saturation of the curve.

In higher-dimensional settings (10 and 20 features), the brute-force method continues to discover better sparse supports as k increases, with the explained variance improving more gradually. These curves rise more slowly at first but reach higher variance values before eventually plateauing. This pattern suggests that when more features are available, the number of sparse directions capable of capturing variance increases, and brute-force can identify the globally best support among them.

Due to the combinatorial complexity of this method, we limit our evaluation to a maximum of $k = 20$ in all experiments. Even with parallelization, runtime becomes prohibitive beyond this threshold for $p = 20$, making exact computation infeasible. Despite this, the brute-force results serve as a valuable upper bound, allowing us to benchmark the performance of approximate algorithms presented in previous sections.

5.2 Effect of Data Dimensionality (Greedy Method)

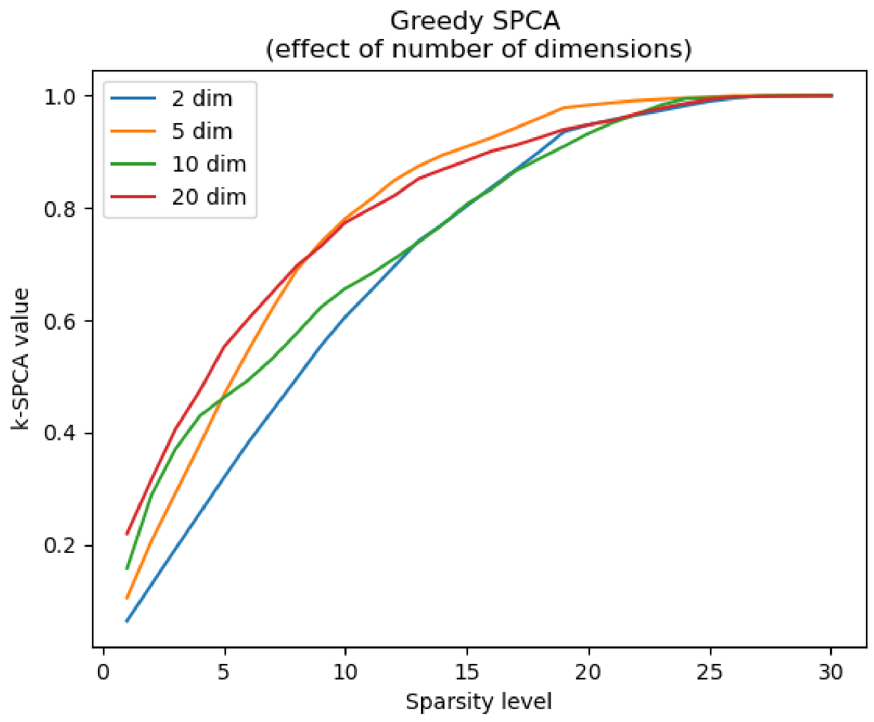

Figure 2: Greedy SPCA lower bound as a function of sparsity level for different data dimensionalities (2, 5, 10, and 20 dimensions). The plot illustrates how the greedy method performs with increasing sparsity across varying numbers of features.

Figure 2 illustrates that the greedy SPCA lower bound increases consistently with the sparsity level k across all tested dimensions. In lower-dimensional datasets (2 and 5 features), the bound tends to saturate at smaller values of k, indicating that a limited number of features is sufficient to explain most of the variance. In contrast, for higher-dimensional datasets (10 and 20 features), the bound continues to rise more sharply in early stages of sparsity, suggesting that the greedy selection process identifies more informative sparse subspaces when more features are available.

Additionally, while the rate of increase is more pronounced for higher dimensions at smaller k, all curves tend to converge toward a similar value as the sparsity level approaches the dimensional limit. This convergence indicates that beyond a certain sparsity threshold, the marginal gain in explained variance diminishes as more coordinates are added.

5.3 Effect of Data Dimensionality (Random Sampling Method)

Figure 3 illustrates the behavior of the sampling-based lower bound across varying data dimensionalities. As with the greedy approach, the explained variance increases with the sparsity level k for all values of p. However, the sampling curves exhibit less divergence across different dimensions, indicating a more stable and uniform performance regardless of feature count.

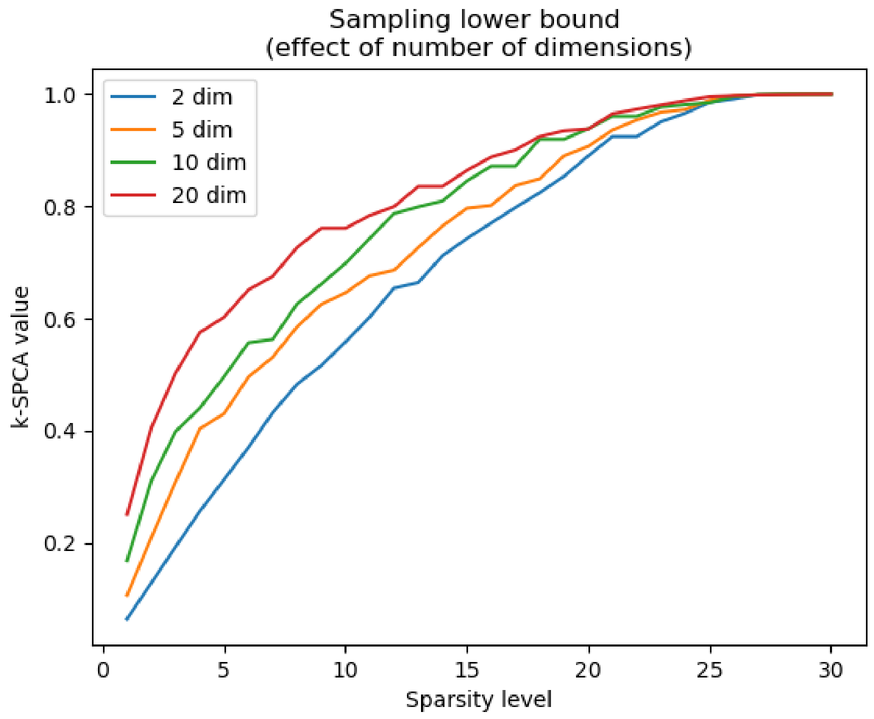

Figure 3: Sampling lower bound as a function of sparsity level for different data dimensionalities (2, 5, 10, and 20 dimensions). The plot shows the impact of feature dimensionality on the lower bound obtained via the random sampling method.

Notably, higher-dimensional datasets (e.g., 10 and 20 dimensions) achieve marginally better variance bounds than lower-dimensional ones at the same sparsity levels. Nevertheless, this gap is smaller than in the greedy method (Figure 2), suggesting that random sampling is less sensitive to the structural richness introduced by increasing dimensionality. This robustness makes it a useful baseline when computational resources are constrained.

5.4 Effect of Data Dimensionality (Thresholding Method)

Figure 4 presents the performance of the thresholding algorithm across different values of p. As the sparsity level k increases, the explained variance rises consistently for all dimensionalities. Similar to the greedy method, higher-dimensional data yields more substantial gains in the early stages, likely due to the increased availability of informative features.

However, a key distinction lies in the smoothness and tight clustering of the thresholding curves across dimensions. Unlike the greedy method, which exhibits more variation between dimensional settings, the thresholding method demonstrates more stable behavior. This suggests that its selection mechanism—based purely on the magnitude of the unconstrained principal component—generalizes well even as dimensionality changes.

These results underscore the thresholding method's strength as a fast and reliable heuristic. Despite its algorithmic simplicity, it consistently captures a substantial portion of the total variance, particularly when k is small. This confirms its role in the literature as a strong baseline for sparse PCA and supports its use in large-scale or low-resource settings where more complex methods may be impractical.

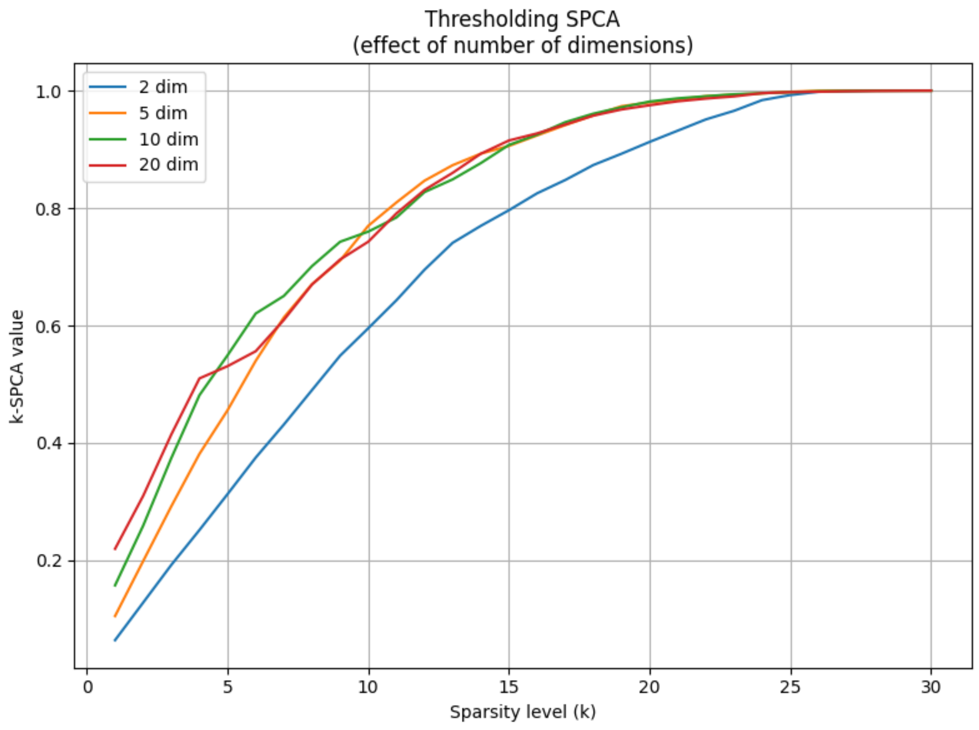

Figure 4: Thresholding SPCA value as a function of sparsity level for different data dimensionalities (2, 5, 10, and 20 dimensions). The plot illustrates the performance of the thresholding method across varying feature dimensions.

5.5 Effect of Data Dimensionality (DSPCA with Rounding)

Figure 5 displays the performance of the DSPCA algorithm with rounding across different data dimensionalities. As the sparsity level k increases, the explained variance grows steadily across all tested settings, indicating that the method effectively identifies high-variance directions even under sparsity constraints.

The curves exhibit a similar upward trend to the thresholding method, with rapid gains for small k and eventual saturation near the full variance as k approaches $K = 30$. Notably, DSPCA appears to outperform thresholding slightly at lower sparsity levels, particularly for higher-dimensional datasets ($p = 10$ and $p = 20$), suggesting that the semidefinite relaxation better captures the global structure of the covariance matrix.

Despite its higher computational complexity, DSPCA with rounding maintains consistent performance across dimensionalities, with smooth convergence and minimal sensitivity to feature count. These results confirm its theoretical appeal as a principled convex relaxation for sparse PCA d'Aspremont et al. [2007], and demonstrate that the randomized rounding procedure yields robust sparse approximations in practice.

6 Conclusion

In this study, we have evaluated five different algorithms for solving the k-Sparse Principal Component Analysis (k-Sparse PCA) problem: **Brute-Force Search**, **Greedy Forward Selection**, **Thresholding Method**, **Random Sampling**, and **DSPCA with Rounding**. Each algorithm

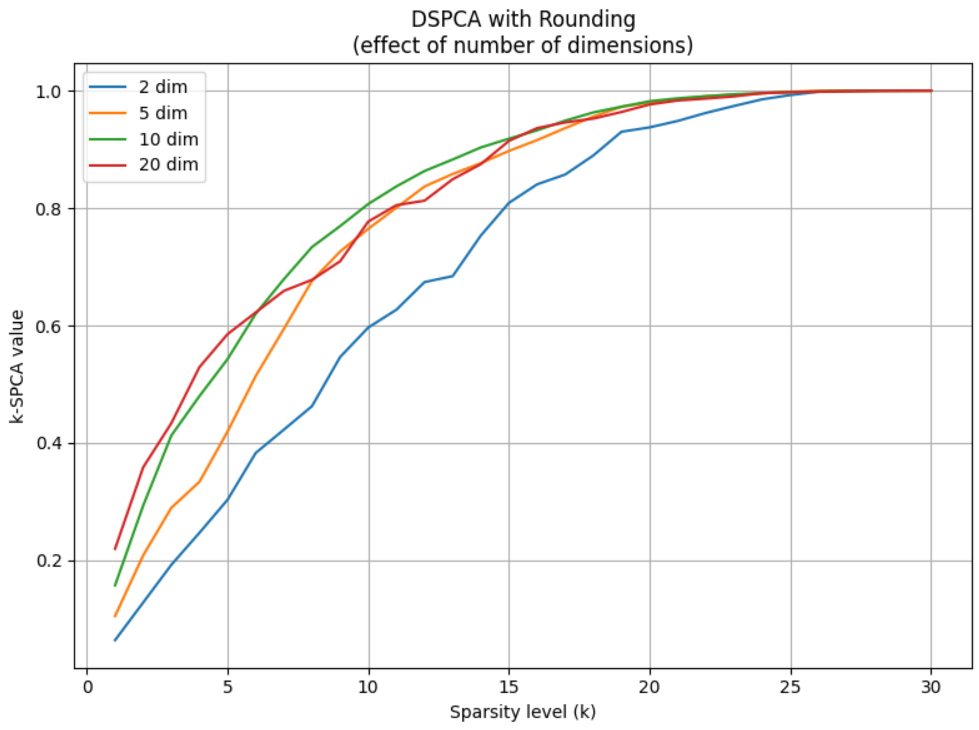

Figure 5: DSPCA with rounding: explained variance as a function of sparsity level for different data dimensionalities (2, 5, 10, and 20 dimensions). The plot illustrates how the semidefinite relaxation method scales across varying feature dimensions.

offers a different trade-off between **computational efficiency** and **solution quality**, and their performance has been assessed across a range of data dimensionalities and sparsity levels.

6.1 Brute-Force Search

As the **gold standard**, the brute-force method guarantees the **optimal solution** but becomes computationally infeasible for larger datasets. For small-to-moderate dimensionalities, it provides an exact benchmark for comparing approximate methods, highlighting the potential variance explained by the **optimal k-sparse principal component**. However, its *O(p choose k* $\cdot$ *k^3)* time complexity limits its scalability, especially for larger k and p values. Despite its computational burden, the brute-force approach offers valuable insights into the problem's theoretical bounds.

6.2 Greedy Forward Selection

The greedy algorithm offers a significant **reduction in computational complexity** compared to brute force while often providing near-optimal solutions. The results show that it is particularly effective for lower-dimensional datasets but still performs well in higher dimensions. While it lacks worst-case guarantees, it strikes a balance between **efficiency** and **accuracy**, making it a practical choice for large-scale applications.

6.3 Thresholding Method

The thresholding approach, while simple, is highly **efficient** and **scalable**. The method identifies sparse components by retaining the **largest eigenvalues** from the unconstrained principal component, making it an attractive option in **low-resource settings**. Though it may not always capture the highest possible variance, it provides a **good approximation** with minimal computational cost.

6.4 Random Sampling

Random sampling offers a **probabilistic approximation** of sparse PCA, where **multiple random subsets** of features are evaluated. While this method tends to be more **stable** across dimensions, its performance heavily relies on the **number of samples**. The results show that random sampling provides a robust solution with **moderate computational cost**, particularly when dealing with large datasets and higher dimensions.

6.5 DSPCA with Rounding

The DSPCA with rounding method employs **semidefinite programming (SDP)** to **relax** the problem into a convex optimization framework. By leveraging **rounding techniques**, it recovers a sparse vector that approximates the principal component. This method provides strong performance and theoretical guarantees, making it an effective tool for sparse PCA problems. However, the **complexity of solving SDP** can become prohibitive in extremely high-dimensional settings, though it still outperforms the greedy and sampling methods in terms of **variance explained**.

7 Future Directions

While the algorithms presented here offer promising results for sparse PCA, there are several opportunities for further improvements:

- **Improving Brute-Force Search Efficiency**: One potential avenue is to develop more **efficient combinatorial optimization techniques** or approximate methods that can reduce the complexity of brute-force search without sacrificing accuracy.
- **Hybrid Algorithms**: Combining elements from different methods, such as **greedy search** and **random sampling**, may offer further improvements in performance, especially in scenarios where both **speed** and **accuracy** are crucial.
- **Higher-Order Relaxations**: Exploring **higher-order relaxations** for SDP and other convex programming approaches may yield more scalable solutions while maintaining theoretical guarantees.
- **Parallel Computing and Distributed Architectures**: For methods like brute-force and DSPCA with rounding, utilizing **distributed computing frameworks** could significantly reduce the execution time for large datasets, making these methods more practical in real-world applications.

In summary, the choice of algorithm for k-Sparse PCA should depend on the **trade-off between computational cost** and the **desired accuracy**. While brute-force and DSPCA with rounding provide the most accurate results, the greedy, thresholding, and random sampling methods offer scalable alternatives with good performance, making them suitable for high-dimensional datasets where computational resources are constrained.

References

Lauren Berk and Dimitris Bertsimas. Certifiably optimal sparse principal component analysis. *Mathematical Programming Computation*, 11(3):381–420, 2019.

Siu On Chan, Dimitris Papailiopoulos, and Aviad Rubinstein. On the approximability of sparse pca. In *Proceedings of the 29th Conference on Learning Theory (COLT)*, volume 49, pages 1–24. PMLR, 2016.

Vaggos Chatziafratis, Rasmus Kyng, Sushant Sachdeva, and Luca Zanetti. Approximation algorithms for sparse principal component analysis. In *International Conference on Learning Representations (ICLR)*, 2021. URL `https://openreview.net/forum?id=trj4iYJpIvy`.

Alexandre d'Aspremont, Francis Bach, and Laurent El Ghaoui. A direct formulation for sparse pca using semidefinite programming. *Journal of Machine Learning Research*, 9:1269–1294, 2007.