
EE364B Final Report

Nipun Agarwala
nipuna1@stanford.edu

Kao Kitichotkul
rk22@stanford.edu

Shaked Regev
sregev@stanford.edu

1 Introduction

Solving symmetric positive definite (SPD) sparse $n \times n$ linear systems $Ax = b$ is common in fields such as computational fluid dynamics, power networks, economics, material analysis, structural analysis, statistics, circuit simulation, computer vision, model reduction, electromagnetics, acoustics, combinatorics, undirected graphs, and heat or mass transfer. Solving directly (e.g., with Cholesky factorization) can be computationally prohibitive. Iterative methods are then preferred. The rate of convergence of these methods depends on the relative magnitude and the clustering of the eigenvalues of A [2, 7]. Preconditioning is a process that transforms the system into an equivalent one with more favorable properties.

$$MAx = Mb$$

is left preconditioning,

$$AMy = b \quad My = x$$

is right preconditioning, and

$$CAC^T y = Cb \quad C^T y = x$$

is left-right preconditioning. If we know $M = C^T C$, we can rearrange the conjugate gradient (CG) method to preconditioned CG (PCG) solve without knowledge of C . We seek M such that $CAC^T \approx I$ which can multiply vectors cheaply. One such M can be computed as a sparse approximate inverse (SPAI) [1]. However, barring substantial cancellation, the inverse of a sparse matrix with a connected graph is dense. This is because the connectivity of A^{-1} is the same as of A^n (which is complete if A is connected). For this reason, our sparse approximation alone is virtually guaranteed to be missing a lot of relevant data about the problem. It should be noted however that it's possible that these entries are very small, in which case a sparse approximation alone is appropriate. This motivates us to perhaps do better by adding a low rank approximation giving

$$M = C^T C = M^* + UU^T,$$

where M^* is a sparse approximation of A^{-1} and UU^T is an $n \times r$ low rank correction (with $r \ll n$). Note that we will use methods such as CVX that scale as $r^3 \rightarrow r^6$, so $r \approx n^{1/6} \rightarrow n^{1/3}$ will keep these calculations at most $O(n^{1.5})$ which should be reasonable compared to other computations in the problem (such as finding the sparse part of the inverse or solving the system with PCG). We therefore wish to solve

$$\min_{\text{rank}(U) \leq r} \kappa(C^T AC) \quad (1)$$

such that $M = C^T C = M^* + UU^T$ (with given M^*). This problem is non-convex, but we will write a tractable relaxation.

2 Method

We use SSAI [4] to compute M^* . This is a version of SPAI guaranteed to produce an SPD matrix, computed with coordinate descent. Consider the left and right preconditioning $(M^* + UU^T)A \approx I$ and $A(M^* + UU^T) \approx I$. To get a symmetric expression, sum them:

$$\begin{aligned} (M^* + UU^T)A + A(M^* + UU^T) &\approx 2I \\ \Rightarrow UU^T A + AUU^T &\approx S := 2I - M^* A - AM^*. \end{aligned} \quad (2)$$

The system is too large to be solved for the optimal value of U directly, so we project each row of S such that the matrix structure is preserved. Specifically, we used Extremely Sparse Johnson-Lindenstrauss Transform (ESE), which can be computed very efficiently for sparse matrices; to embed n points of dimension m , ESE requires the embedding dimension of $\mathcal{O}(\log(n) \log(m)/\epsilon)$ to achieve a distortion bounded by $\pm\epsilon$ [8]. To project each row of a data matrix $X \in \mathbb{R}^{n \times m}$ to a dimension r using ESE, we construct a projection matrix as follow.

1. Build hash function $h : \{1, \dots, r\} \mapsto \{1, \dots, m\}$
2. Build hash function $\sigma : \{1, \dots, m\} \mapsto \{-1, +1\}$
3. Build $\Omega \in \mathbb{R}^{m \times r}$ with

$$\Omega_{ij} = \begin{cases} \sqrt{\frac{m}{r}} \sigma(i) & \text{for } i = h(j) \\ 0 & \text{for } i \neq h(j) \end{cases}$$

4. The projection is $\hat{X} = X\Omega$.

We project each row of S from n dimension to r dimension using FJLT and then perform a thin QR factorization.

$$S\Omega = QR \tag{3}$$

Note that Q has dimension $n \times r$. Suppose $U = Q\tilde{U}$ where $\tilde{U} \in \mathbb{R}^{r \times r}$. Then, we project the left hand side of equation (2) using Q .

$$\begin{aligned} Q^T(UU^T A + AUU^T)Q &= Q^T(Q\tilde{U}\tilde{U}^T Q^T A + AQ\tilde{U}\tilde{U}^T Q^T)Q \\ &= \tilde{U}\tilde{U}^T Q^T A Q + Q^T A Q \tilde{U}\tilde{U}^T \\ &= \tilde{U}\tilde{U}^T \tilde{A} + \tilde{A}\tilde{U}\tilde{U}^T, \end{aligned}$$

where $\tilde{A} := Q^T A Q$. Then, from equation (2), we obtain

$$\begin{aligned} \tilde{U}\tilde{U}^T \tilde{A} + \tilde{A}\tilde{U}\tilde{U}^T &\approx Q^T(2I - M^* A - AM^*)Q \\ &= Q^T S Q := E \end{aligned}$$

Let $P := \tilde{U}\tilde{U}^T$. We will solve the following problem.

$$\begin{aligned} &\operatorname{argmin}_P \|E - P\tilde{A} - \tilde{A}P^T\|_F. \\ &\text{subject to } P \in \mathcal{S}_{++}^n \end{aligned} \tag{4}$$

We can recover $\tilde{U}$ from the optimal P with Cholesky factorization, and finally we have $U = Q\tilde{U}$.

3 Test matrix families

3.1 Baseline

We started by creating a "baseline" matrix family to see whether our method performs well, if our assumptions are correct. We created this family by taking a Wathen matrix, adding a low rank Gaussian noise matrix to it and inverting it. This gives us a dense matrix, which we know certainly has an inverse that can be represented as sparse and low rank. The motivation for choosing this matrix is to verify that there are no errors in our method for finding the low rank approximation. If our method works on this matrix but not the others, we can deduce that the issue may be with the assumption that the inverse can be presented as sparse and low rank, or the method in which we derived the sparse approximation.

3.2 Trefethen and Wathen

The first family is the Trefethen challenge matrices [5, 6] which are considered challenging because systems with them are impossible to solve with dense solvers, due to substantial fill-in in the factorization regardless of the ordering of the unknowns. Sparse approximate inverses for these matrices are also generally good approximations, making it challenging to get a good low rank

addition. Let p_i denote the i -th prime number then, $T_{ij} = \begin{cases} p_i & i = j \\ 1 & |i - j| = 2^N \\ 0 & o.w. \end{cases} \quad n \in \mathcal{N} \cup \{0\}$

The second family is the Wathen matrices, which are finite element matrices with random density. Finite elements is an important tool for solving almost any multidimensional physics problem numerically [3].

4 Implementation

We use Python 3 and CVXPY to find the solution to the problem stated in (4). The implementation can be found at https://github.com/nipunagarwala/ee364b_final_project. The test matrices generated in MATLAB needed to be read into Python before being utilized. To get a workable solution in a reasonable runtime, we had to apply some tricks described below.

4.1 Maintaining P as PSD

The P matrix solution provided by CVXPY often had k extremely small negative eigenvalues λ_n , $1 \leq n \leq k$, where $k < r$. These negative values were often $\sim 10^{-6}$ or smaller. As a result, finding $P = \tilde{U}\tilde{U}^T$ using Cholesky factorization in Numpy or Scipy is not possible since the algorithm expects eigenvalues $\lambda_i \geq 0$, $1 \leq i \leq r$. We came up with a technique called *Eigenvalue Magnitude Preservation* to ensure $P \in \mathcal{S}_{++}^r$. If the eigenvalues are negative, we take its absolute value and reconstruct matrix P as P' . Specifically, from the Eigenvalue decomposition we see that that:

$$\begin{aligned} P &= Q\Sigma Q^T \\ \Sigma &= \text{diag}(\lambda_1, \dots, \lambda_r) \\ \Sigma'_{ij} &= \begin{cases} |\Sigma_{ii}|, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

We can reconstruct a $P' = Q\Sigma'Q^T \in \mathcal{S}_{++}^n$ which can now be used in a Cholesky factorization. The advantage of this method is that we do not need to throw away eigenvectors and the eigenvalues are relatively close to the actual eigenvalues of P because the negative eigenvalues are very small in magnitude. The disadvantage is that we still change the eigenvalues more than absolutely necessary.

5 Results

5.1 Evaluation Metric

We measure the performance of our approach by calculating how well the resulting matrix approximates the inverse. Precisely, for a sparse + low rank approximate inverse $M^* + UU^T$ of a matrix A , we define the error as

$$\|(M^* + UU^T)A + A(M^* + UU^T) - 2I\|_F = \|UU^T A + AUU^T - S\|_F,$$

where S is defined in equation (2). For the sake of symmetry, we multiply the approximate inverse to both the left and the right of the matrix A . We also compare the sparse and low rank approximate inverse to the sparse-only approximate inverse obtained from SSAI without the low rank correction. In this case, the error is reduced to $\|S\|_F$. In the plots in this section, the error of the sparse-only approximation is referred to as $r = 0$.

5.2 Baseline

As baselines, we apply our approximate inverse approach to our baseline family in section 3.1, which we know to have sparse and low-rank inverse. We experimented with matrices of size $n = 4033$ and Gaussian noise rank $r = 2, 4, 6, 8$. We know that the rank of the Gaussian noise should be the eventual rank of the low rank $\tilde{U}$ in the approximate inverse. Since we have access to the sparse part of the exact inverse by the construction of the matrices in section 3.1, we use this sparse part as M^* instead of obtaining M^* by applying SSAI to the matrix A . The error of the sparse and low rank approximate inverse for $r = 1, \dots, 20$ is shown in Figure 1.

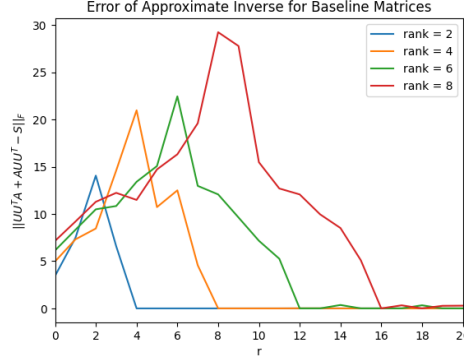


Figure 1: Error of the sparse + low rank approximate inverse for the baseline matrices. Each line belongs to one such matrix where the labels specify the rank of the Gaussian noise. The x-axis is the chosen rank r of the low rank part of the approximate inverse obtained by our approach.

We observe that the approximate inverse is accurate only when the r is at least twice of the rank of the corresponding low rank part of the exact inverse. For example, the error of the approximate inverse for the baseline matrix with Gaussian noise of rank 2 drops to almost zero when $r \geq 4$. We think that our approach requires higher rank than necessary because finding the low rank part sampled from the Gaussian distribution is a hard problem. Nevertheless, this result shows that our approach can approximate the inverse very well if the sparse and low rank inverse exists.

5.3 Test Matrices

We apply our approximate inverse approach to a Trefethen matrix of size $n = 32768$ and a Wathen matrix of size $n = 43681$. We choose $r = 1, 2, 4, 6, 8, 12, 16, 20, 24, 28$. The results are shown in Figure 2.

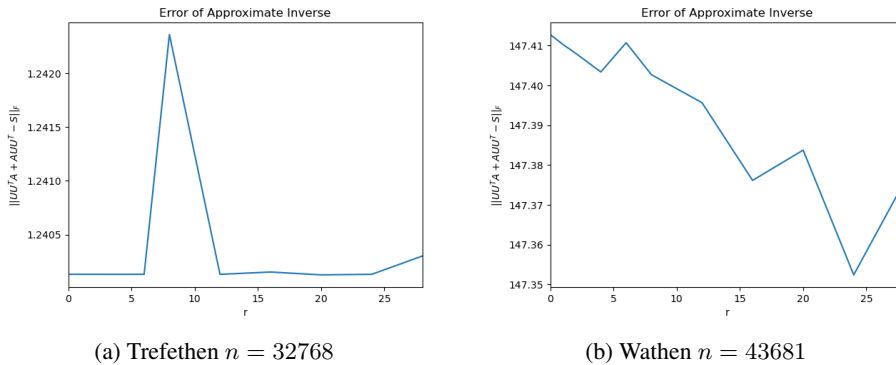


Figure 2: Error of the sparse + low rank approximate inverse for the test matrices. (a) The error for the Trefethen matrix. Note that CVXPY gave a warning that it might give an inaccurate solution at $r = 8$ where there is a peak in the error plot. (b) The error for the Wathen matrix.

Note that CVXPY gave a warning that it might give an inaccurate solution only for the Trefethen matrix when $r = 8$, at which we observe a peak in error. On using MOSEK as our solver,

we do not see this peak, suggesting that this error was particular to the SCS solver in CVXPY due to numerical instability.

For the Trefethen matrix, we do not observe any improvement from adding the low rank correction to the sparse approximate inverse. Since the error when there is no low rank correction ($r = 0$) is already low, we interpret this result as SSAI performs very well for this matrix that it is difficult to find a low rank correction to improve the inverse. For the Wathen matrix, we observe a trend that the low rank correction reduces the error as r increases.

6 Analysis

6.1 Inverse Matrix Improvement

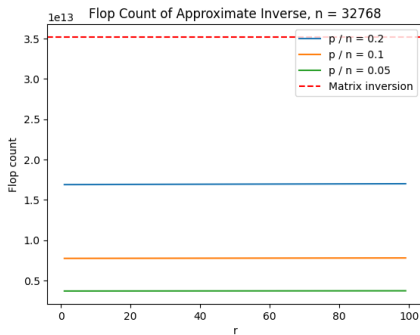
The results in section 5.2 show that if our assumption that the inverse of a matrix can be represented as sparse and low rank holds, and we have a good approximation of the sparse part (in our case exact), our method works well. The results in section 5.3 show that if the sparse part is already very close to the exact inverse, it is difficult to improve. Furthermore, they show that if the sparse part is not exact, the low rank approximation helps, even when $r \ll n$.

A way to improve this would be to add the low rank approximations one at a time (or in batches) and adapt the sparse part to suit them. This is because the sparse approximation of the inverse does not "know" of the upcoming low rank approximation, but the low rank approximation needs a good sparse approximation first for the method to make sense. Once we have the low rank approximation, it's possible that some entries in the sparse approximation need to change, which will affect the next low-rank update.

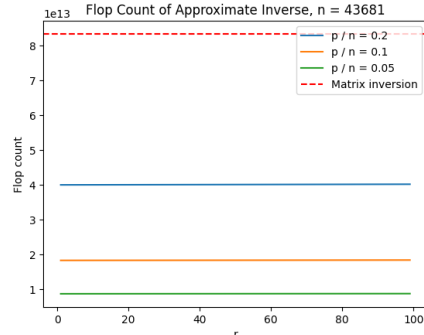
6.2 Computational Complexity

Let $A \in \mathcal{R}^{n \times n}$. We want to find $U \in \mathcal{R}^{n \times r}$, where r is the desired projected rank. Let us also assume that the CVXPY solver has a convergence rate of $\log\left(\frac{1}{\epsilon}\right)$, which is reasonable for algorithms like Newton's Method and Proximal Gradient. We define p to be the average number of non-zeros per row in A (and assume that the actual distribution is close to this average). For the proposed algorithm, we can now see the breakdown of computational complexity in the table below that we sum together in [3]

Complexity breakdown		Complexity breakdown	
Algorithm	FLOPs	Algorithm	FLOPs
M^* calculation	$2np^2$	$\tilde{A}$ calculation	$2npr$
Sparse FJLT transformation	$npr + nr$	E matrix calculation	$2npr$
QR factorization	$2(n - \frac{r}{3})r^2$	CVXPY Solver	$O\left(\frac{1}{\epsilon}\right)r^3 = 10^3r^3$
S Matrix calculation	$2np^2$	Cholesky factorization	$\frac{1}{3}r^3$
U calculation	nr^2		



(a) Trefethen $n = 32768$



(b) Wathen $n = 43681$

Figure 3: Plots for test matrices across rank r and sparsity p

From the plots, we see that sparsity influences the FLOP count trend lines the most. Specifically, we get the biggest benefit from having lower NNZs in the matrix i.e. a smaller p . Since $r \ll p$ and $r \ll n$, it is obvious that changing r would not increase the FLOP count significantly. We can thus conclude that our method makes computational sense only when we can guarantee at most 20% NNZ in the original A and M^* matrices.

6.3 Non-monotonicity of the low rank approximation usefulness

Our results show a seemingly confusing result of non-monotonicity of the error with respect to r . The problem formulation allows eigenvalues and eigenvectors that are 0, so there should be no degradation in the objective when the rank increases. However, finding solutions at the boundaries is very difficult numerically and in practice the rank of P is always r . An example can help illustrate why a higher rank is not always better. Consider the matrix $B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$. The best rank 0 approximation of B is $B_0 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$. The best rank 1 approximation $B_1 = B$. The best rank 2 approximation is $B = \begin{pmatrix} 1+u & 1 \\ 1 & 1+u \end{pmatrix}$ where u = machine precision. Clearly the quality of the approximation is not monotonic in r , even in exact arithmetic. A sudden increase in the error with an increase in r means that either there is no good approximation of the low-rank part of the matrix with that rank, or it is too difficult to find numerically.

7 Conclusion

In this paper, we show that given an original matrix which can be successfully decomposed into a sparse and low rank matrix, we can find an approximate inverse successfully that is sparse and low rank. This process is much more computationally scalable. We also show that for the Trefethen and Wathen matrix families, the addition of the low rank approximation does not provide much benefit because the sparse approximation is good enough and the method to find the sparse approximation does not have prior knowledge of the upcoming low rank approximation.

References

- [1] M. Benzi, C. D. Meyer, and M. Tuma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17:1135–1149, 1996.
- [2] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. The Johns Hopkins University Press, Baltimore, fourth edition, 2013.
- [3] T. J.R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Courier Corporation, North Chelmsford, 2012.
- [4] S. Regev, 2019. <https://github.com/shakedregev/SSAI>.
- [5] L. N. Trefethen. A hundred-dollar, hundred-digit challenge. *SIAM News*, 35:65, 2002.
- [6] L. N. Trefethen. The SIAM 100-dollar, 100-digit challenge, 2002. <https://people.maths.ox.ac.uk/trefethen/hundred.html>.
- [7] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, Philadelphia, PA, 1983.
- [8] Rong Yin, Yong Liu, Weiping Wang, and Dan Meng. Extremely sparse johnson-lindenstrauss transform: From theory to algorithm. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1376–1381, 2020.