A New Preconditioner that Exploits Low-Rank Approximations to Factorization Error

Higham, Nicholas J. and Mary, Theo

2018

MIMS EPrint: **2018.10**

ISSN 1749-9097

A New Preconditioner that Exploits Low-Rank Approximations to Factorization Error[*]

Nicholas J. Higham Theo Mary

April 23, 2018

Abstract

We consider ill-conditioned linear systems $Ax = b$ that are to be solved iteratively, and assume that a low accuracy LU factorization $A \approx \widehat{L}\widehat{U}$ is available for use in a preconditioner. We have observed that for ill-conditioned matrices A arising in practice, A^{-1} tends to be numerically low rank, that is, it has a small number of large singular values. Importantly, the error matrix $E = \widehat{U}^{-1}\widehat{L}^{-1}A - I$ tends to have the same property. To understand this phenomenon we give bounds for the distance from E to a low-rank matrix in terms of the corresponding distance for A^{-1}. We then design a novel preconditioner that exploits the low-rank property of the error to accelerate the convergence of iterative methods. We apply this new preconditioner in three different contexts fitting our general framework: low floating-point precision (e.g., half precision) LU factorization, incomplete LU factorization, and block low-rank LU factorization. In numerical experiments with GMRES-based iterative refinement we show that our preconditioner can achieve a significant reduction in the number of iterations required to solve a variety of real-life problems.

1 Introduction

We consider the iterative solution of a linear system $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ is nonsingular. A widely used approach is to compute a low accuracy LU factorization $A = \widehat{L}\widehat{U} + \Delta A$, and use the approximate inverse factors $\widehat{U}^{-1}\widehat{L}^{-1}$ as a preconditioner. However, the rate of convergence of the iterative method strongly depends on the matrix properties, such as the distribution of its singular values. If A is ill conditioned the preconditioned iteration may converge slowly or not at all. In such a situation, we may have no other choice than to compute a more accurate LU factorization, which is likely to be too expensive for large-scale problems.

The objective of this article is to present a novel and yet general preconditioner that builds on a given approximate LU factorization and can be effective even for ill-conditioned systems. This preconditioner is based on the following key observation: ill-conditioned matrices that arise in practice often have a *small number of small singular values*. The inverse of such a matrix has a small number of large singular values and so is numerically low rank. This observation suggests that the error matrix

$$E = \widehat{U}^{-1}\widehat{L}^{-1}A - I = \widehat{U}^{-1}\widehat{L}^{-1}\Delta A \approx A^{-1}\Delta A$$

is of interest, because we may expect E to retain the numerically low-rank property of A^{-1}. The main contributions of this article are to specify conditions under which E is indeed numerically low rank and to describe how to exploit this property to accelerate the convergence of iterative methods by building a preconditioner based on a low-rank approximation to E.

We begin, in section 2, by describing the general framework for our analysis and providing three examples of algorithms that fit within the framework: low floating-point precision (for example, half precision) LU factorization, incomplete LU factorization, and block low-rank LU factorization.

[*]Version of April 23, 2018. **Funding**: This work was supported by Engineering and Physical Sciences Research Council grant EP/P020720/1, The MathWorks, and the Royal Society. The opinions and views expressed in this publication are those of the authors, and not necessarily those of the funding bodies.

We also describe the experimental setting of the following sections. In section 3 we derive sufficient conditions for the error matrix to be low rank. In section 4 we propose a new preconditioner based on a low-rank approximation to the error. In section 5 we analyze experimentally how this preconditioner can accelerate the solution of linear systems in the three contexts mentioned above. We use GMRES-based iterative refinement (GMRES-IR) as our iterative method. Concluding remarks are given in section 6.

Throughout this article, the unsubscripted norm $\|\cdot\|$ denotes the 2-norm.

2 General framework and three applications

We first describe the general framework to which the theory and algorithms developed in this article apply. We then provide three examples of widely used algorithms that fit within the framework: low floating-point precision LU factorization, incomplete LU factorization, and block low-rank LU factorization. We finally describe the experimental setting used in the following sections.

2.1 General framework description

We consider a linear system $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ is nonsingular. We assume an approximate LU factorization

$$A = \widehat{L}\widehat{U} + \Delta A \tag{2.1}$$

can be computed, but we do not make any assumption on how the factorization is computed, nor do we assume ΔA to have any special structure. We define the error matrix as

$$E = \widehat{U}^{-1}\widehat{L}^{-1}A - I = \widehat{U}^{-1}\widehat{L}^{-1}\Delta A.$$

We will show theoretically and experimentally that E is likely to have low numerical rank[1] when A is ill conditioned (that is, $\kappa(A) = \|A\|\|A^{-1}\| \gg 1$). Note that A being ill conditioned is not a strict requirement of our framework, but the algorithms we design can cope with such matrices; therefore, in this article, we mostly consider ill-conditioned A.

2.2 Low floating-point precision LU factorization

The use of half or single precision floating-point arithmetic in mixed-precision algorithms is becoming increasingly common. In particular, half-precision arithmetic is attracting growing interest now that it has started to become available in hardware [13].

A natural way to exploit a low precision LU factorization is with iterative refinement. Carson and Higham [6] investigate iterative refinement in three precisions. They show that if the working precision u is double precision, the LU factors are computed in half precision, and residuals are computed in quadruple precision then convergence of the refinement process is guaranteed if $\kappa(A) \le 10^4$ and backward errors and forward errors of order u will be produced. They also show that, by using GMRES preconditioned with the LU factors to solve for the correction term, the limit on $\kappa(A)$ can be relaxed to 10^{12} or 10^{16} in order to ensure a forward error or backward error of order u, respectively. Algorithm 1 summarizes this GMRES-based iterative refinement (GMRES-IR), which was originally proposed in a form using two precisions [5].

While GMRES-IR requires only a small number of iterative refinement steps (outer iterations) when $\kappa(A)$ satisfies the required bounds, the number of iterations in the GMRES solves (inner iterations) can be large. In this article, we will demonstrate how the new preconditioner proposed in section 4 can help to reduce the number of iterations and therefore widen the range of tractable problems.

[1]The term "numerical rank" will be defined in Definition 3.1.

Algorithm 1 GMRES-based iterative refinement with precisions u_f, u, and u_r.

```
1: Compute LU factorization of $A$ at precision $u_f$.
2: Solve $Ax_1 = b$ at precision $u_f$ using the LU factors and store $x_1$ at precision $u$.
3: while not converged do
4:     Compute $r_i = b - Ax_i$ at precision $u_r$ and round $r_i$ to precision $u$.
5:     Solve $\widetilde{A}d_i \equiv \widehat{U}^{-1}\widehat{L}^{-1}Ad_i = \widehat{U}^{-1}\widehat{L}^{-1}r_i$ at precision $u$ using GMRES, with matrix–vector
       products with $\widetilde{A}$ computed at precision $u_r$.
6:     $x_{i+1} = x_i + d_i$ at precision $u$.
7: end while
```

2.3 Incomplete LU factorization

The LU factors of a sparse matrix can be much less sparse than the matrix, because of fill-in, potentially making the factorization too expensive. This problem can be alleviated by using fill-reducing reorderings of the matrix, such as minimum degree, minimum fill, and nested dissection. However, the amount of fill-in can still be quite large in many practical applications.

A widely used alternative approach is to compute an incomplete LU (ILU) factorization [26, sec. 10.3], in which the sparsity of the LU factors is kept under a given threshold. For example, ILU(0) forces $\widehat{L}\widehat{U}$ to have the same sparsity pattern as A. More generally, the sparsity of the computed factors can be controlled by a tolerance τ, where filled entries of magnitude less than τ (relative to the norm of A) are dropped. For large values of τ, ILU-based preconditioners may yield slow convergence of the iterative method. We will show how our new preconditioner, used in conjunction with an ILU preconditioner, can overcome this obstacle.

2.4 Block low-rank LU factorization

In numerous scientific applications, such as the solution of partial differential equations, the matrices resulting from the discretization of the physical problem have been shown to possess a low-rank property [4]: suitably defined off-diagonal blocks of their Schur complements can be approximated by low-rank matrices. This property can be exploited to provide a substantial reduction of the complexity of matrix factorizations.

Several matrix representations—so-called low-rank formats—have been proposed in the literature. Most of them fit within our general framework, but we will focus on the block low-rank (BLR) format [1], [2], [3]. The BLR format is based on a flat 2D blocking of the matrix that is defined by conveniently clustering the associated unknowns. A BLR representation $\widetilde{A}$ of a dense matrix A has the form

$$\widetilde{A} = \begin{bmatrix} \widetilde{A}_{11} & \widetilde{A}_{12} & \cdots & \widetilde{A}_{1p} \\ \widetilde{A}_{21} & \cdots & \cdots & \vdots \\ \vdots & \cdots & \ddots & \vdots \\ \widetilde{A}_{p1} & \cdots & \cdots & \widetilde{A}_{pp} \end{bmatrix}, \tag{2.2}$$

where each block A_{ij} of size $m_i \times n_j$ and numerical rank k_{ij}^{τ} is approximated by a low-rank product $\widetilde{A}_{ij} = X_{ij}Y_{ij}^T$, where X_{ij} is $m_i \times k_{ij}^{\tau}$ and Y_{ij} is $n_j \times k_{ij}^{\tau}$.

The $\widetilde{A}_{ij}$ approximation of each block can be computed in different ways. We have chosen to use a truncated QR factorization with column pivoting, which is a QR factorization with column pivoting that is truncated as soon as a diagonal coefficient of the R factor falls below a prescribed threshold τ, referred to as the BLR threshold. The BLR threshold τ controls the accuracy of the factorization.

In order to perform the LU factorization of a dense BLR matrix, the standard LU factorization has to be modified so that the numerically low-rank blocks can be exploited to reduce the number of operations. Many such algorithms can be defined, depending on where the compression step (the introduction of the low-rank approximations) is performed. In this article, we will consider the CUFS variant, introduced in [2]. As described in [2], the CUFS variant achieves the lowest complexity of all BLR variants by performing the compression as early as possible.

Using the BLR factorization as an approximate LU factorization has been shown to make an efficient preconditioner for iterative methods such as GMRES, outperforming both traditional iterative and direct methods for many applications of practical interest [21]. We will show that our low-rank approximation to the factorization error can improve the performance of the BLR preconditioner.

2.5 Experimental setting

The numerical results have been obtained in MATLAB R2017b on a laptop computer equipped with 8 GB of memory and a four-core Intel i5-6300U running at 2.40GHz.

Our experiments use four different precisions of IEEE standard arithmetic: half-precision, with unit roundoff $u = 4.9 \times 10^{-4}$, for which we used the MATLAB fp16 class from the Cleve Laboratory toolbox [24]; single precision ($u = 6.0 \times 10^{-8}$) and double precision ($u = 1.1 \times 10^{-16}$); and quadruple precision ($u = 9.6 \times 10^{-35}$), for which we use the Advanpix Multiprecision Computing Toolbox [25].

We will consider a large set of both random dense and real-life sparse matrices coming from a variety of applications. The randsvd matrices were generated with the MATLAB `gallery` function, using `rng(1)` to seed the random number generator. All the other matrices come from the SuiteSparse Matrix Collection (previously called the University of Florida Sparse Matrix Collection) [9]. The full list is provided in Table 2.1.

The matrices are of relatively small size. We are mainly interested in the theoretical and numerical behavior of the algorithms; their high performance implementation is not our focus here. We will, however, explain why the proposed algorithms are expected to perform well on large-scale problems and in parallel computing environments.

Throughout the rest of this article, we will specify between parentheses after each matrix name which type of approximate LU factorization was performed: half precision LU factorization (fp16), incomplete LU factorization (ILU), or block low-rank LU factorization (BLR). For ILU, we used the MATLAB `ilu` function with threshold partial pivoting, with `setup.type = 'ilutp'` and drop tolerance τ.

Table 2.1 indicates which type of factorization was tested on which matrices. The fp16 factorization was tested on the full set; the ILU and BLR factorizations were tested on a subset of matrices with values of τ varying between 10^{-1} and 10^{-5} for ILU and between 0.99 and 10^{-5} for BLR. This leads to a total of 163 tests on 40 matrices.

3 Bounds for the numerical rank of the error matrix

We begin with some definitions.

Definition 3.1. *Let $A \in \mathbb{R}^{n\times n}$ be nonzero. For $k \le n$, the rank-k accuracy of A is*

$$\varepsilon_k(A) = \min_{W_k} \left\{ \frac{\|A - W_k\|}{\|A\|} : \operatorname{rank} W_k \le k \right\}. \tag{3.1}$$

We call W_k of rank k an optimal rank-k approximation to A if W_k achieves the minimum in (3.1). The numerical rank of A at accuracy ε, denoted by $k_\varepsilon(A)$, is

$$k_\varepsilon(A) = \min\{k : \varepsilon_k(A) \le \varepsilon\}.$$

The matrix A is of low numerical rank if $\varepsilon_k(A) \ll 1$ for some $k \ll n$.

For brevity, we will sometimes shorten "low numerical rank" to "low rank"; this should cause no confusion as we will not need to refer to exactly low-rank matrices.

Table 2.1: The test matrices, indicating for each one which of ILU and BLR was tested and the corresponding values of the drop tolerance or the BLR threshold τ.

Matrix	n	$\kappa(A)$	Values of τ tested		Application
			ILU	BLR	
randsvd(1e4,2)	100	1.0e+04	—	—	Random dense
randsvd(1e7,2)	100	1.0e+07	—	—	Random dense
randsvd(1e10,2)	100	1.0e+10	—	—	Random dense
randsvd(1e7,1)	100	1.0e+07	—	—	Random dense
randsvd(1e7,3)	100	1.0e+07	—	—	Random dense
d_dyn1	87	7.4e+06	$10^{-\{1,3,5\}}$	—	Chemical Process Simulation
d_ss	53	6.1e+08	$10^{-\{1,3,5\}}$	—	Chemical Process Simulation
west0132	132	4.2e+11	$10^{-\{1,3,5\}}$	10^{-2}	Chemical Process Simulation
west0167	167	4.8e+10	10^{-1}	10^{-5}	Chemical Process Simulation
impcol_a	207	1.4e+08	$10^{-\{1,3,5\}}$	—	Chemical Process Simulation
impcol_e	225	1.4e+08	$10^{-\{1,3\}}$	—	Chemical Process Simulation
rajat11	135	9.2e+05	$10^{-\{1,3,5\}}$	10^{-1}	Circuit Simulation
rajat14	180	3.2e+08	$10^{-\{1,3,5\}}$	10^{-1}	Circuit Simulation
494_bus	494	2.4e+06	$10^{-\{1,3,5\}}$	0.9, 10^{-1}	Power Network
bfwa398	398	3.0e+03	$10^{-\{1,3\}}$	0.9	Electromagnetics
utm300	300	8.5e+05	$10^{-\{3,5\}}$	$10^{-\{1,2,3\}}$	Electromagnetics
arc130	130	6.1e+10	$10^{-\{1,3,5\}}$	—	Materials
robot	120	4.3e+08	$10^{-\{1,3\}}$	—	Robotics
rotor1	100	2.4e+12	$10^{-\{1,3\}}$	—	Structural
lund_a	147	2.8e+06	$10^{-\{1,3,5\}}$	0.9	Structural
nos1	237	2.0e+07	$10^{-\{1,3\}}$	0.99	Structural
nos5	468	1.1e+04	$10^{-\{1,3\}}$	0.99,0.9	Structural
lshp_406	406	1.1e+03	10^{-1}	10^{-1}	Thermal
ex1	216	3.3e+04	$10^{-\{2,3,5\}}$	$10^{-\{3,4,5\}}$	Computational Fluid Dynamics
saylr1	238	7.8e+08	$10^{-\{1,3,5\}}$	0.9, $10^{-\{3,4,5\}}$	Computational Fluid Dynamics
steam1	240	2.8e+07	$10^{-\{1,3,5\}}$	10^{-1}	Computational Fluid Dynamics
steam3	80	5.0e+10	$10^{-\{1,3,5\}}$	—	Computational Fluid Dynamics
cavity01	317	3.5e+04	$10^{-\{1,3\}}$	$10^{-\{1,2,3\}}$	Computational Fluid Dynamics
fs_183_1	183	2.2e+13	$10^{-\{1,3,5\}}$	—	2D/3D Problem
plskz362	362	4.7e+05	10^{-3}	10^{-3}	2D/3D Problem
cz308	308	1.4e+04	$10^{-\{1,3\}}$	0.99, 0.9	2D/3D Problem
cz400	400	2.6e+05	$10^{-\{1,3\}}$	10^{-3}	2D/3D Problem
tumor_1	205	2.6e+05	$10^{-\{1,3,5\}}$	10^{-1}	Optimal Control
hangGlider_1	360	1.1e+10	$10^{-\{3,5\}}$	10^{-1}	Optimal Control
orbitRaising_1	442	1.1e+08	$10^{-\{3,5\}}$	10^{-3}	Optimal Control
str_600	363	1.9e+05	$10^{-\{1,3,5\}}$	$10^{-\{1,2\}}$	Optimization
ww_36_pmec_36	66	3.0e+11	$10^{-\{1,3,5\}}$	—	Eigenvalue/Model Reduction
lop163	163	2.8e+07	$10^{-\{1,3,5\}}$	10^{-3}	Statistical/Mathematical
rw136	136	2.5e+05	$10^{-\{1,3,5\}}$	10^{-1}	Statistical/Mathematical
CAG_mat364	364	1.4e+05	10^{-3}	10^{-1}	Combinatorial

Let $X\Sigma Y^T$ denote the singular value decomposition (SVD) of A, and let $\sigma_i(A)$ be the ith largest singular value. By the Eckart–Young–Mirsky theorem [11], [19, p. 468], [23], $\|A - W_k\|$ is minimized[2] for $W_k = X_{:,1:k}\Sigma_{1:k,1:k}Y^T_{:,1:k}$, which yields

$$\varepsilon_k(A) = \frac{\sigma_{k+1}(A)}{\sigma_1(A)}. \tag{3.2}$$

As stated in the introduction, we have observed that ill-conditioned matrices often have a (numerically) low-rank inverse. We now assume that A^{-1} is low rank and seek conditions under which the error $E = \widehat{U}^{-1}\widehat{L}^{-1}\Delta A$ retains this low-rank property. To that aim, we have to answer two questions: is $\widehat{U}^{-1}\widehat{L}^{-1}$ low rank if A^{-1} is low rank? And is E low rank if $\widehat{U}^{-1}\widehat{L}^{-1}$ is low rank?

To answer the first question, we consider $\widehat{L}\widehat{U}$ as an additive perturbation of A. We need the following lemma.

Lemma 3.1. *Let $X \in \mathbb{R}^{n\times n}$ and $X + \Delta X \in \mathbb{R}^{n\times n}$ be nonsingular. Then*

$$\sigma_i(X + \Delta X) \le \sigma_i(X)\big(1 + \|X^{-1}\Delta X\|\big), \quad 1 \le i \le n. \tag{3.3}$$

Proof. Apply inequality (3.3.26) from [18] to X and $X(I + X^{-1}\Delta X)$. □

We apply the lemma twice. First, recalling (2.1) and taking $X = \widehat{L}\widehat{U}$ and $\Delta X = \Delta A$ in (3.3) yields (for all $i \le n$)

$$\sigma_i(A) \le \sigma_i(\widehat{L}\widehat{U})\big(1 + \|\widehat{U}^{-1}\widehat{L}^{-1}\Delta A\|\big). \tag{3.4}$$

Second, taking $X = A$ and $\Delta X = -\Delta A$ in (3.3) yields

$$\sigma_i(\widehat{L}\widehat{U}) \le \sigma_i(A)\big(1 + \|A^{-1}\Delta A\|\big). \tag{3.5}$$

We can now answer the first question with the following theorem.

Theorem 3.1. *Let $A = \widehat{L}\widehat{U} + \Delta A \in \mathbb{R}^{n\times n}$ be nonsingular. The rank-k accuracy of $\widehat{U}^{-1}\widehat{L}^{-1}$ satisfies*

$$\varepsilon_k(\widehat{U}^{-1}\widehat{L}^{-1}) \le \beta_g\beta_s\varepsilon_k(A^{-1}), \quad 1 \le k \le n, \tag{3.6}$$

with $\beta_g = 1 + \|A^{-1}\Delta A\|$ and $\beta_s = 1 + \|\widehat{U}^{-1}\widehat{L}^{-1}\Delta A\|$.

Proof. For all $k \le n$, by (3.2) we have

$$\frac{\varepsilon_k(\widehat{U}^{-1}\widehat{L}^{-1})}{\varepsilon_k(A^{-1})} = \frac{\sigma_{k+1}(\widehat{U}^{-1}\widehat{L}^{-1})\sigma_1(A^{-1})}{\sigma_1(\widehat{U}^{-1}\widehat{L}^{-1})\sigma_{k+1}(A^{-1})} = \frac{\sigma_n(\widehat{L}\widehat{U})\sigma_{n-k}(A)}{\sigma_{n-k}(\widehat{L}\widehat{U})\sigma_n(A)}.$$

Bounding the numerator with (3.4) and (3.5) yields the result. □

Theorem 3.1 states that if $\|A^{-1}\Delta A\|$ and $\|\widehat{U}^{-1}\widehat{L}^{-1}\Delta A\|$ are not too large then $\widehat{U}^{-1}\widehat{L}^{-1}$ will retain the low-rank property of A^{-1}. The quantity $\beta_g = 1 + \|A^{-1}\Delta A\|$ bounds how much the "perturbed" singular values of $\widehat{L}\widehat{U}$ can grow with respect to those of A by (3.5). Conversely, $\beta_s = 1 + \|\widehat{U}^{-1}\widehat{L}^{-1}\Delta A\|$ bounds how much they can shrink by (3.4). The inequality (3.6) is sharp in a scenario where $\sigma_n(A) = \sigma_1(A^{-1})^{-1}$ grows by a factor β_g and $\sigma_{n-k}(A) = \sigma_{k+1}(A^{-1})^{-1}$ shrinks by a factor β_s. It is not clear whether this scenario is attainable. In any case the bound (3.6) is very pessimistic in practice.

Numerical experiments are reported in Table 3.1 for a subset of the matrices, with the LU factors computed either in half precision or in double precision as an incomplete LU factorization or BLR factorization. For all these matrices, A^{-1} has low (numerical) rank. We see that the singular values of $\widehat{L}\widehat{U}$ are usually of the same order of magnitude as those of A, so that $\widehat{U}^{-1}\widehat{L}^{-1}$ remains of low rank. The bound (3.6) is usually weak by three orders of magnitude or more, but depending on the matrix, it can still imply a low rank. A possible explanation for the bound (3.6) being weak is that $\beta_g\beta_s$ is the same for all k, whereas the value $\varepsilon_k(\widehat{U}^{-1}\widehat{L}^{-1})/\varepsilon_k(A^{-1})$ itself depends on k. It may thus not matter if $\varepsilon_k(\widehat{U}^{-1}\widehat{L}^{-1})/\varepsilon_k(A^{-1})$ is large for a large value of k, since we are only interested in small k.

We now turn to our second question: assuming $\widehat{U}^{-1}\widehat{L}^{-1}$ is low rank, when can we expect $E = \widehat{U}^{-1}\widehat{L}^{-1}\Delta A$ also to be low rank? We answer this question with the following theorem.

[2]This result is usually stated for the Frobenius norm, but actually holds for any unitarily invariant norm.

Table 3.1: Ratios and corresponding bounding quantities for a selection of matrices, where the LU factorization is computed in half precision or as an incomplete LU factorization or BLR factorization.

Matrix	$\max_k \frac{\sigma_k(\widehat{L}\widehat{U})}{\sigma_k(A)}$	β_g	$\max_k \frac{\sigma_k(A)}{\sigma_k(\widehat{L}\widehat{U})}$	β_s	$\max_k \frac{\varepsilon_k(\widehat{U}^{-1}\widehat{L}^{-1})}{\varepsilon_k(A^{-1})}$	$\beta_g\beta_s$
rotor1 (fp16)	1.9e+00	3.7e+04	1.0e+00	3.6e+04	9.7e−01	1.3e+09
robot (fp16)	1.0e+00	7.4e+02	1.0e+00	7.1e+02	1.0e+00	5.3e+05
rajat14 (fp16)	2.2e+00	4.6e+02	8.3e+00	4.6e+02	7.7e+00	2.1e+05
tumor_1 (fp16)	4.2e+00	3.5e+02	1.2e+00	4.1e+02	5.1e+00	1.4e+05
west0132 (fp16)	1.1e+00	4.4e+03	1.0e+00	4.6e+03	1.1e+00	2.0e+07
west0167 (fp16)	1.1e+00	1.6e+02	1.0e+00	1.5e+02	1.1e+00	2.5e+04
ww_36_pmec_36 (fp16)	2.1e+04	3.4e+06	1.2e+00	2.0e+02	2.5e+04	6.9e+08
impcol_a (fp16)	1.0e+00	6.5e+02	1.0e+00	6.4e+02	1.0e+00	4.1e+05
impcol_e (fp16)	1.0e+00	4.8e+00	1.0e+00	4.8e+00	1.0e+00	2.3e+01
nos1 (fp16)	2.9e+00	8.7e+03	1.0e+00	6.5e+00	1.2e+00	5.7e+04
steam1 (fp16)	1.7e+00	2.2e+05	1.4e+00	2.9e+05	1.4e+00	6.3e+10
steam3 (fp16)	1.0e+00	1.2e+04	1.0e+00	1.2e+04	1.0e+00	1.4e+08
randsvd(1e4,2) (fp16)	2.1e+02	1.0e+04	1.5e+00	4.8e+01	3.3e+02	4.9e+05
randsvd(1e7,2) (fp16)	7.1e+03	1.0e+07	1.4e+00	1.4e+03	1.0e+04	1.4e+10
randsvd(1e10,2) (fp16)	1.4e+07	9.9e+09	1.4e+00	7.2e+02	1.9e+07	7.1e+12
lund_a (fp16)	2.2e+00	1.1e+04	1.7e+00	1.1e+04	6.0e−01	1.2e+08
lund_a (ILU, $\tau = 10^{-1}$)	1.1e+00	3.7e+04	2.8e+05	5.7e+08	1.9e+01	2.1e+13
lund_a (ILU, $\tau = 10^{-3}$)	4.7e+00	2.2e+03	1.6e+00	4.8e+02	7.5e+00	1.0e+06
lund_a (ILU, $\tau = 10^{-5}$)	1.0e+00	5.4e+00	1.0e+00	5.4e+00	1.0e+00	2.9e+01
cavity01 (fp16)	4.9e+00	3.6e+03	3.2e+01	5.3e+03	1.4e+00	1.9e+07
cavity01 (ILU, $\tau = 10^{-3}$)	1.2e+00	3.5e+01	1.0e+00	2.9e+01	1.3e+00	1.0e+03
cavity01 (BLR, $\tau = 10^{-2}$)	1.2e+00	6.5e+01	5.2e+00	4.0e+01	6.3e+00	2.6e+03
cavity01 (BLR, $\tau = 10^{-3}$)	1.0e+00	1.1e+00	1.0e+00	1.2e+00	1.0e+00	1.3e+00

Theorem 3.2. *Let $A = \widehat{L}\widehat{U} + \Delta A \in \mathbb{R}^{n\times n}$ be nonsingular. The error $E = \widehat{U}^{-1}\widehat{L}^{-1}\Delta A$ satisfies*

$$\varepsilon_k(E) \le \mu\varepsilon_k(\widehat{U}^{-1}\widehat{L}^{-1}), \quad 1 \le k \le n, \tag{3.7}$$

with

$$\mu = \frac{\|\widehat{U}^{-1}\widehat{L}^{-1}\|\,\|\Delta A\|}{\|\widehat{U}^{-1}\widehat{L}^{-1}\Delta A\|}. \tag{3.8}$$

Proof. Let W_k be an optimal rank-k approximation of $\widehat{U}^{-1}\widehat{L}^{-1}$, so that $\|\widehat{U}^{-1}\widehat{L}^{-1} - W_k\| = \varepsilon_k(\widehat{U}^{-1}\widehat{L}^{-1})\|\widehat{U}^{-1}\widehat{L}^{-1}\|$. Then $E_k = W_k\Delta A$ has rank at most k and therefore

$$\varepsilon_k(E) \le \frac{\|E - E_k\|}{\|E\|} = \frac{\|(\widehat{U}^{-1}\widehat{L}^{-1} - W_k)\Delta A\|}{\|E\|} \le \varepsilon_k(\widehat{U}^{-1}\widehat{L}^{-1})\frac{\|\widehat{U}^{-1}\widehat{L}^{-1}\|\,\|\Delta A\|}{\|E\|},$$

as required. □

Combining our two theorems we obtain the following corollary.

Corollary 3.1. *Let $A = \widehat{L}\widehat{U} + \Delta A \in \mathbb{R}^{n\times n}$ be nonsingular. The error $E = \widehat{U}^{-1}\widehat{L}^{-1}\Delta A$ satisfies*

$$\varepsilon_k(E) \le \mu\beta_g\beta_s\varepsilon_k(A^{-1}), \quad k \le n. \tag{3.9}$$

Corollary 3.1 tells us that the error E retains the low-rank property of $\widehat{U}^{-1}\widehat{L}^{-1}$ as long as μ is not too large, that is, $\|\widehat{U}^{-1}\widehat{L}^{-1}\|\|\Delta A\|$ is not too much larger than $\|\widehat{U}^{-1}\widehat{L}^{-1}\Delta A\|$, and β_g and β_s are not too large. Here again, inequalities (3.7) and thus (3.9) are pessimistic in practice: experiments reported in Table 3.2 show that E remains in most cases low rank.

Explaining the role of μ is less straightforward than for β_g and β_s. To clarify the role of μ, we compute an upper bound $\bar{\mu} = \min_k \bar{\mu}_k$ whose mathematical meaning is easier to grasp. The following theorem states that unless ΔA is very special, μ should be of moderate size.

Table 3.2: The bounds from Theorem 3.2 and Corollary 3.1 compared with the quantities they are bounding, using the same matrices and factorizations as in Table 3.1.

matrix	$\max_k \frac{\varepsilon_k(E)}{\varepsilon_k(\widehat{U}^{-1}\widehat{L}^{-1})}$	μ	$\max_k \frac{\varepsilon_k(E)}{\varepsilon_k(A^{-1})}$	$\beta_g\beta_s\mu$
rotor1 (fp16)	7.6e+00	8.5e+03	7.3e+00	1.1e+13
robot (fp16)	2.1e+00	1.3e+02	2.1e+00	7.0e+07
rajat14 (fp16)	7.9e−01	2.8e+03	2.4e+00	6.0e+08
tumor_1 (fp16)	4.3e−01	4.9e+00	1.8e+00	6.9e+05
west0132 (fp16)	4.4e+00	1.6e+04	4.7e+00	3.3e+11
west0167 (fp16)	5.1e+00	6.3e+04	5.4e+00	1.6e+09
ww_36_pmec_36 (fp16)	2.0e+00	5.7e+01	4.3e+04	3.9e+10
impcol_a (fp16)	2.7e−01	2.6e+01	2.7e−01	1.1e+07
impcol_e (fp16)	1.4e+00	1.4e+02	1.4e+00	3.2e+03
nos1 (fp16)	5.0e+01	5.0e+05	5.9e+01	2.8e+10
steam1 (fp16)	6.1e−01	8.8e+00	6.1e−01	5.6e+11
steam3 (fp16)	4.9e−01	1.1e+03	5.1e−01	1.5e+11
randsvd(1e4,2) (fp16)	1.8e+00	2.2e+00	4.7e+02	1.1e+06
randsvd(1e7,2) (fp16)	1.8e+00	2.2e+00	1.5e+04	3.0e+10
randsvd(1e10,2) (fp16)	1.6e+00	2.1e+00	3.0e+07	1.5e+13
lund_a (fp16)	2.6e+00	6.5e+00	1.2e+00	7.8e+08
lund_a (ILU, $\tau = 10^{-1}$)	5.6e+00	1.8e+01	1.9e+01	3.7e+14
lund_a (ILU, $\tau = 10^{-3}$)	1.1e+00	2.9e+00	6.7e+00	3.0e+06
lund_a (ILU, $\tau = 10^{-5}$)	1.3e+00	5.4e+00	1.3e+00	1.6e+02
cavity01 (fp16)	2.3e+00	7.2e+01	2.1e+00	1.4e+09
cavity01 (ILU, $\tau = 10^{-3}$)	1.8e+00	4.0e+00	2.2e+00	4.1e+03
cavity01 (BLR, $\tau = 10^{-2}$)	5.2e−01	1.3e+02	3.2e+00	3.3e+05
cavity01 (BLR, $\tau = 10^{-3}$)	7.1e+00	1.3e+01	7.1e+00	1.7e+01

Theorem 3.3. *The quantity $\mu_F = \|\widehat{U}^{-1}\widehat{L}^{-1}\|\|\Delta A\|/\|E\|_F$, which differs from μ only in that the Frobenius norm of E is taken rather than the 2-norm, is bounded by*

$$\mu_F \le \bar{\mu}_k = \frac{\sigma_{n+1-k}(\widehat{L}\widehat{U})}{\sigma_n(\widehat{L}\widehat{U})} \frac{\|\Delta A\|}{\|P_k \Delta A\|}, \quad 1 \le k \le n, \tag{3.10}$$

where $P_k = X_k X_k^T$, $X_k = [x_{n+1-k}, \dots, x_n]$, and x_j is the left singular vector corresponding to the jth largest singular value of $\widehat{L}\widehat{U}$.

Proof. The proof draws its inspiration from [7]. Let $X\Sigma Y^T$ be an SVD of $\widehat{L}\widehat{U}$, with $\sigma_1 \ge \dots \ge \sigma_n$. Then, with x_j and y_j denoting the jth columns of X and Y, respectively,

$$E = Y\Sigma^{-1}X^T \Delta A = \sum_{i=1}^{n} \frac{1}{\sigma_i} y_i (x_i^T \Delta A).$$

Thus, for all $k \le n$,

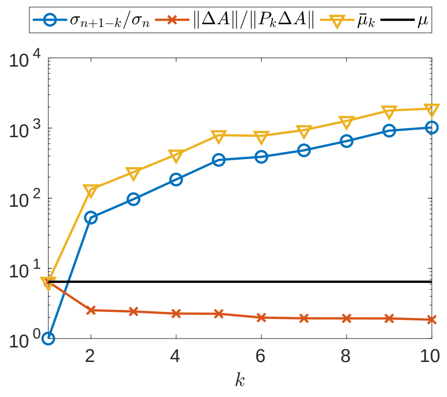
$$\|E\|_F^2 \ge \sum_{i=n+1-k}^{n} \frac{1}{\sigma_i^2} \|y_i\|^2 \|x_i^T \Delta A\|^2 \ge \frac{1}{\sigma_{n+1-k}^2} \sum_{i=n+1-k}^{n} \|x_i^T \Delta A\|^2 = \frac{1}{\sigma_{n+1-k}^2} \|P_k \Delta A\|^2.$$

We can therefore bound μ_F for all $k \le n$ by

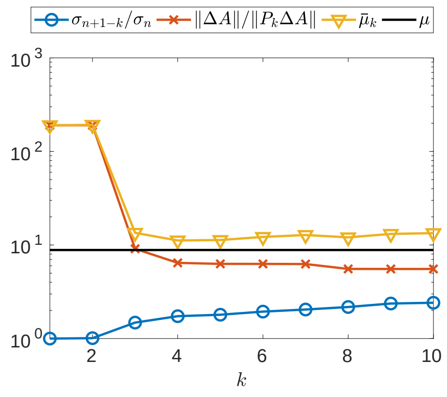
$$\mu_F = \frac{\|\Delta A\|}{\sigma_n \|E\|_F} \le \frac{\sigma_{n+1-k}}{\sigma_n} \frac{\|\Delta A\|}{\|P_k \Delta A\|},$$

as required. □

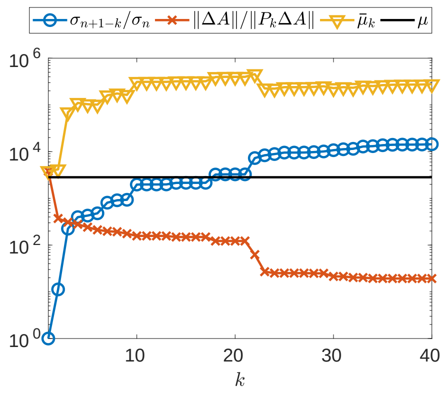
Theorem 3.3 tells us that μ_F will be small when ΔA is a “typical” matrix: one having a significant component in the subspace span(X_k) for some k such that $\sigma_{n+1-k}(\widehat{L}\widehat{U}) \approx \sigma_n(\widehat{L}\widehat{U})$. Note that the proof requires us to take the Frobenius norm of E, which is in general greater than its 2-norm (and thus $\mu_F \le \mu$). However, since E is expected to be numerically low rank, $\|E\| \approx \|E\|_F$ should



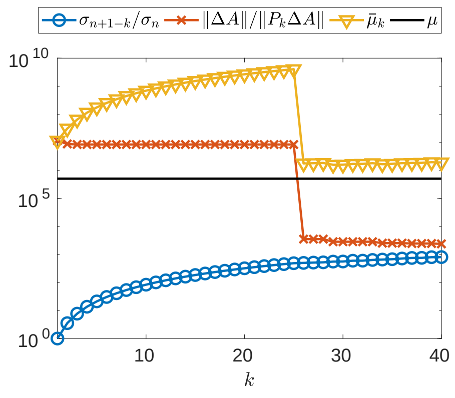
(a) Matrix lund_a (fp16).



(b) Matrix steam1 (fp16).



(c) Matrix rajat14 (fp16).



(d) Matrix nos1 (fp16).

Figure 3.1: Quantities μ (defined in (3.8)) and $\bar{\mu}_k$ (defined in (3.10)), and factors in upper bound (3.10). μ is small if ΔA is typical (top two matrices) but can be large for special ΔA (bottom two matrices).

hold. Thus, we can also expect $\mu \approx \mu_F$ to be small. Figure 3.1 plots the values of $\|\Delta A\|/\|P_k \Delta A\|$, $\sigma_{n+1-k}(\widehat{L}\widehat{U})/\sigma_n(\widehat{L}\widehat{U})$, and $\bar{\mu}_k$ as a function of k for four example matrices. In the first two cases (matrices lund_a and steam1), ΔA is a typical matrix and thus μ is small. However, for some matrices (e.g., rajat14 and nos1 in Figure 3.1), ΔA turns out to be special and leads to a large μ. Nevertheless, for all the matrices studied, the bound (3.7) is never sharp when μ is large (see the second column of Table 3.2).

We conclude this section by building a matrix for which the μ bound is both large and sharp, to prove it cannot be improved without further assumptions on the matrix. Generating a matrix A for which the bound is sharp is difficult, because we do not control the matrix ΔA of rounding errors. To build such a matrix, we adopt the approach of Higham [17] and use a direct search optimization procedure to maximize the ratio $\max_k \varepsilon_k(E)/\varepsilon_k(A^{-1})$, where the $\widehat{L}\widehat{U}$ factors are computed with a half-precision LU factorization. We obtained the 5×5 matrix

$$A = \begin{bmatrix} 0.70262059 & 0.10163234 & -0.42912567 & -0.09693864 & 0.25863816 \\ -0.56142448 & 0.09716073 & -0.79799236 & 0.15351272 & 0.14026396 \\ -0.07776207 & -0.25317885 & 0.27700267 & 0.60226171 & 0.68688885 \\ 0.23039340 & 0.44918023 & -0.07276241 & -0.05928910 & 0.43898931 \\ 0.31561248 & -0.71961953 & -0.31980583 & 0.21672135 & -0.19520101 \end{bmatrix}, \tag{3.11}$$

for which the bound is almost sharp: $\max_k(\varepsilon_k(E)/\varepsilon_k(\widehat{U}^{-1}\widehat{L}^{-1})/) \approx 9.6e{+}03$ and $\mu \approx 1.0e{+}04$ (for $k = 1$). As shown in Figure 3.2, A^{-1} and $\widehat{U}^{-1}\widehat{L}^{-1}$ are indeed numerically low rank, but the error E

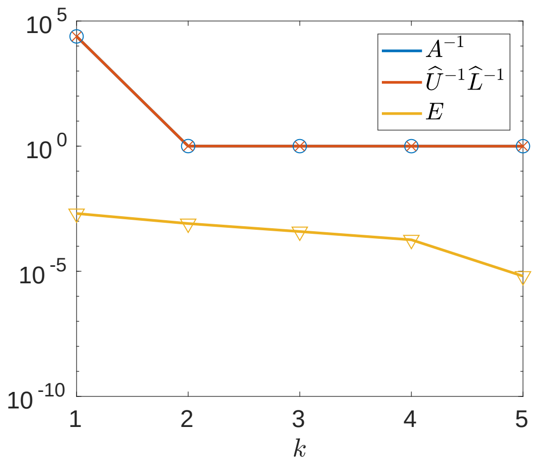


Figure 3.2: Singular values distribution of A^{-1}, $\widehat{U}^{-1}\widehat{L}^{-1}$, and E corresponding to example matrix A defined in (3.11) for which the bound of Theorem 3.2 is almost sharp. A^{-1} and $\widehat{U}^{-1}\widehat{L}^{-1}$ are numerically low rank but the error E is not.

is not.

4 A novel preconditioner based on the low-rank error

Now we consider the solution of the linear system $Ax = b$ by means of an iterative method. A classical approach to accelerate the convergence of the iterative method is to use the preconditioner based on the computed LU factors

$$\Pi_{LU} = \widehat{U}^{-1}\widehat{L}^{-1} \tag{4.1}$$

to solve the preconditioned system $\Pi_{LU}Ax = \Pi_{LU}b$. However, when the LU factors have been computed at low accuracy, and when the matrix A is ill conditioned, convergence may still be slow. To overcome this obstacle, we propose a novel preconditioner

$$\Pi_{E_k} = (I + E_k)^{-1}\widehat{U}^{-1}\widehat{L}^{-1}, \tag{4.2}$$

which is based on a rank-k approximation E_k to the error $E = \widehat{U}^{-1}\widehat{L}^{-1}A - I$. We expect the factor $(I + E_k)^{-1}$ to improve the quality of the preconditioner. In the extreme case where $E_k = E$, Π_E is exactly equal to A^{-1} and thus yields a perfect preconditioner, but this is obviously too expensive. However, if $k \ll n$ then the solve with $I + E_k$ can be cheaply done with the Sherman–Morrison–Woodbury formula [15]. Since we have shown that E is often numerically low rank, we may expect Π_{E_k}, with some suitable small k, to be almost as good a preconditioner as Π_E.

4.1 Computing E_k, a low-rank approximation to E

It would be too expensive to compute E_k explicitly, so we develop a matrix-free approach, in which we only need to perform matrix-vector products with E_k.

Although other methods could be considered, we use the randomized sampling algorithm [14], [20] which has been shown to be efficient for computing low-rank approximations to dense matrices [22]. We build E_k as a truncated SVD of E. We consider two versions of the randomized SVD algorithm, described in Algorithms 2 and 3.

Both algorithms begin by sampling the columns of E with a random matrix Ω of size $n \times (k + p)$, where p is a small integer parameter that provides oversampling. A small amount of oversampling is usually enough to ensure a good accuracy of the low-rank approximation [14]. We then build an orthonormal basis V of S; note that V captures the range of E: $E \approx VV^TE$. In Algorithm 2,

Algorithm 2 Randomized SVD algorithm from [14, Alg. 5.1] via direct SVD of V^TE.

```
Input: matrix $A$, its computed LU factors $\widehat{L}\widehat{U}$, and an $n \times (k+p)$ random matrix $\Omega$.
1: Sample $E$: $S = E\Omega = \widehat{U}^{-1}\big(\widehat{L}^{-1}(A\Omega)\big) - \Omega$.
2: Orthonormalize $S$: $V = \mathrm{qr}(S)$.
3: Form $V^TE = \big((V^T\widehat{U}^{-1})\widehat{L}^{-1}\big)A - V^T$ and compute an SVD $V^TE = X\Sigma Y^T$.
4: Truncate $X$, $\Sigma$, $Y$ into $X_k$, $\Sigma_k$, $Y_k$ to keep only $k$ singular vectors/values.
5: An SVD of $E_k$ is given by $(VX_k)\Sigma_k Y_k^T$.
```

Algorithm 3 Randomized SVD algorithm from [14, Alg. 5.2] via row extraction.

```
Input: matrix $A$, its computed LU factors $\widehat{L}\widehat{U}$, and an $n \times (k+p)$ random matrix $\Omega$.
1: Sample $E$: $S = E\Omega = \widehat{U}^{-1}\big(\widehat{L}^{-1}(A\Omega)\big) - \Omega$.
2: Orthonormalize $S$: $V = \mathrm{qr}(S)$.
3: Compute the interpolative decomposition $V = (I_\ell \;\; W)^T V_{(L,:)}$.
4: Extract $E_{(L,:)}$ and compute a QR factorization $E_{(L,:)}^T = QR$.
5: Form $(I_\ell \;\; W)^T R^T$ and compute an SVD $(I_\ell \;\; W)^T R^T = X\Sigma Y^T$.
6: Truncate $X$, $\Sigma$, $Y$ into $X_k$, $\Sigma_k$, $Y_k$ to keep only $k$ singular vectors/values.
7: An SVD of $E_k$ is given by $X_k\Sigma_k(QY_k)^T$.
```

based on this observation, we compute a rank-k approximation of V^TE by means of a deterministic truncated SVD $X_k\Sigma_kY_k^T$, which then yields the truncated SVD of the original matrix E as $E_k = (VX_k)\Sigma_kY_k^T$. This however requires us to form the product V^TE which, as analyzed in the next section, can be expensive. To overcome this issue, Algorithm 3 builds instead an interpolative decomposition (ID) [8] of V:

$$V = (I_\ell \;\; W)^T V_{(L,:)},$$

where I_ℓ denotes the identity matrix of order $\ell = k+p$ and $V_{(L,:)}$ is a subset of ℓ rows of V. Such a decomposition can be computed by means of a pivoted QR factorization $V^TP = QR$ and by defining $W = R_{1:\ell,1:\ell}^{-1}R_{:,\ell+1:n}$ and $V_{(L,:)} = P_{:,1:\ell}^TV$ [14]. We then have, defining $\widehat{E} = VV^TE$,

$$E \approx \widehat{E} = VV^TE = (I_\ell \;\; W)^T V_{(L,:)}V^TE = (I_\ell \;\; W)^T \widehat{E}_{(L,:)} \approx (I_\ell \;\; W)^T E_{(L,:)}. \tag{4.3}$$

Therefore, we can build the truncated SVD of E based on that of $(I_\ell \;\; W)^TE_{(L,:)}$. The second approximation in (4.3) makes algorithm 3 less accurate than Algorithm 2 by a factor up to $1+\sqrt{1+4k(n-k)}$ [14, Lem. 5.1]. To maintain a unified presentation, we have formulated Algorithm 3 working on the orthonormal basis V. However, as explained in [14], for this second algorithm it is not necessary to orthonormalize the sample, i.e., we can work on S rather than V. This is what we will do in practice.

4.2 The four variants of the Π_{E_k} preconditioner

In the rest of this article, we will analyze four distinct variants of Π_{E_k}, which differ in how E_k is computed:

- $\Pi_{E_k}^{(1)}$: compute E_k with Algorithm 2 and with Ω a random Gaussian matrix;
- $\Pi_{E_k}^{(2)}$: compute E_k with Algorithm 2 and with Ω an SRFT matrix;
- $\Pi_{E_k}^{(3)}$: compute E_k with Algorithm 3 and with Ω a random Gaussian matrix;
- $\Pi_{E_k}^{(4)}$: compute E_k with Algorithm 3 and with Ω an SRFT matrix.

An SRFT matrix is a subsampled random Fourier transform matrix, defined as

$$\Omega = FR,$$

Table 4.1: Cost in flops of the setup and solve phases for the two preconditioners: classical Π_{LU} and new Π_{E_k}, where $\ell = k + p$. For the sake of simplicity, we assume here that A is a dense, full-rank matrix.

	setup	solve
Π_{LU}	$\frac{2}{3}n^3$	$2n^2$
$\Pi_{E_k}^{(1)}$	$\frac{2}{3}n^3 + 8n^2\ell + O(n\ell^2)$	$2n^2 + O(nk)$
$\Pi_{E_k}^{(2)}$	$\frac{2}{3}n^3 + 6n^2\ell + 4n^2\log\ell + O(n\ell^2)$	$2n^2 + O(nk)$
$\Pi_{E_k}^{(3)}$	$\frac{2}{3}n^3 + 4n^2\ell + O(n\ell^2)$	$2n^2 + O(nk)$
$\Pi_{E_k}^{(4)}$	$\frac{2}{3}n^3 + 2n^2\ell + 4n^2\log\ell + O(n\ell^2)$	$2n^2 + O(nk)$

where $F \in \mathbb{C}^{n\times n}$ is the discrete Fourier transform (DFT) matrix and $R \in \mathbb{R}^{n\times\ell}$ is a matrix that randomly selects ℓ distinct columns of F, i.e., it consists of ℓ random distinct columns of the $n \times n$ identity matrix.

4.3 Computational cost analysis

We now analyze the computational cost of our new Π_{E_k} preconditioner and compare it with that of the classical Π_{LU} preconditioner. In the following analysis, for the sake of simplicity, we assume that the matrix A is dense and that the LU factorization is performed in low precision but without reducing the $2n^3/3$ flop cost of standard LU (via, e.g., BLR factorization). We briefly comment on how to extend this analysis to other contexts at the end of this section.

The cost of the algorithm is driven by two main tasks: the setup phase, in which we build Π, and the solve phase, in which we solve the system $\Pi Ax = \Pi b$ via an iterative method, where Π is one of the two preconditioners Π_{LU} or Π_{E_k}. Table 4.1 summarizes the cost of these two phases for Π_{LU} and the four variants of the Π_{E_k} preconditioners. We only keep track of the constants for the $O(n^3)$ and $O(n^2)$ terms, and define $\ell = k + p$.

Both the setup and solve phases are more expensive for the new Π_{E_k} preconditioner. The cost of the solve phase is independent of the variant of Π_{E_k} considered. At each iteration, we must perform a solve with $I + E_k$, which can be achieved via the Sherman–Morrison–Woodbury formula in $O(nk)$ flops. This cost should be small compared with the $2n^2$ flops cost of the two triangular solves with $\widehat{L}$ and $\widehat{U}$.

The cost of the setup phase strongly depends on which variant of Π_{E_k} is used. Computing E_k using Algorithm 2 requires us to perform the products $E\Omega$ (line 1) and V^TE (line 3). Because it would be too expensive to build E explicitly, the products must be computed implicitly, i.e., what we actually compute is $\widehat{U}^{-1}\big(\widehat{L}^{-1}(A\Omega)\big) - \Omega$ and similarly for the V^TE product. Therefore both products cost $4n^2\ell + O(n\ell)$ flops (two triangular solves with $\widehat{L}$ and $\widehat{U}$, one matrix multiplication with A, and one addition). Thus, the setup overhead cost of $\Pi_{E_k}^{(1)}$ is $8n^2\ell + O(n\ell^2)$ flops. While this overhead cost is expected to be small with respect to the initial $O(n^3)$ LU factorization, it can still be quite significant. In fact, for the use of $\Pi_{E_k}^{(1)}$ to be beneficial (in terms of flops) with respect to Π_{LU}, the number of iterations N_{iter} should be reduced by at least 4ℓ, regardless of the problem size n. We do not achieve an asymptotic gain because the setup overhead cost is of order $O(n^2)$, which is the cost of performing one iteration. The preconditioner $\Pi_{E_k}^{(2)}$ samples E with a SRFT matrix Ω instead; this reduces the cost of the sampling from $4n^2\ell + O(n\ell)$ down to $2n^2\ell + 4n^2\log\ell + O(n\ell)$, to which we must still add $4n^2\ell + O(n\ell)$ for computing V^TE, which leads to a total setup cost of $6n^2\ell + 4n^2\log\ell + O(n\ell)$. Thus, for $\Pi_{E_k}^{(2)}$ to be beneficial compared with Π_{LU}, N_{iter} must be reduced by at least $3\ell + 2\log\ell$. On the other hand, $\Pi_{E_k}^{(3)}$ avoids forming V^TE explicitly and therefore replaces a factor $4n^2\ell$ by only $O(n\ell^2)$ flops. This therefore makes $\Pi_{E_k}^{(3)}$ beneficial compared with Π_{LU} if N_{iter} is reduced by at least 2ℓ. Finally, $\Pi_{E_k}^{(4)}$ combines the cost reductions of the previous two variants, which makes it beneficial compared with Π_{LU} if N_{iter} is reduced by at least $\ell + 2\log\ell$.

The four variants of the Π_{E_k} preconditioner are thus decreasingly expensive. However, they are also decreasingly accurate. Indeed, the theoretical properties of SRFT sampling are less well understood (e.g., how to choose the amount of oversampling). Perhaps more concerningly, the row extraction SVD (Algorithm 3) leads to an error that is up to a factor $1+\sqrt{1+4k(n-k)}$ larger than that of Algorithm 2 [14, Lem. 5.1]. Since we are only building a preconditioner, this might not be too problematic if it does not significantly increase the number of iterations. In the following, we will therefore compare all four variants of Π_{E_k}.

Regardless of the variant of Π_{E_k} used, the new preconditioner might in some cases perform more flops than the original Π_{LU} preconditioner if k is large or if the number of iterations is only reduced by a small quantity. Nevertheless, we still expect it to perform better for the following three reasons.

- The solve phase achieves in general a low execution rate because it uses BLAS 2 kernels (in the case of a single right-hand side). On the contrary, for the setup phase, the LU factorization is a BLAS 3 kernel, while computing E_k may also be achieved with BLAS 3 kernels (or “BLAS 2.5” if k is very small).

- The solve phase is performed at working precision, while the setup phase may be performed at lower precision. This includes the LU factorization but also the computation of E_k. The influence of u_{E_k}, the precision at which E_k is computed, will be analyzed in the next section.

- Several applications require the solution for multiple right-hand sides. In this case, the setup overhead cost of the Π_{E_k} preconditioner is amortized by the necessity of performing more solves.

We conclude this cost analysis by briefly indicating how it could be extended to other contexts such as when A is sparse and/or block low-rank. The important question is whether the extra work necessary to build the Π_{E_k} preconditioner remains reasonably small compared with the cost of one iteration (i.e., at least of the same order as in the dense case). For example, if A is a sparse matrix arising from a 3D regular problem, the cost of one triangular solve is only $O(n^{4/3})$ flops [12]. Fortunately, the computation of E_k can also exploit the sparsity of A, and the setup overhead cost is of order $O(n^{4/3}\ell)$ flops, where the constant depends on which variant of Π_{E_k} is considered. Therefore, the extra work required to build Π_{E_k} will be compensated as long as the number of iterations is reduced by some value of order $O(\ell)$, just as in the dense case. Thus we expect our preconditioner to be applicable to large-scale problems, both dense and sparse. A similar argument can be made in the case where A is block low-rank.

5 Numerical experiments with GMRES-IR

In this section, we analyze how our new Π_{E_k} preconditioner can improve the convergence of GMRES-IR (Algorithm 1) [5], which uses iterative refinement with the solves for the correction carried out by preconditioned GMRES. We use three precisions, as proposed in [6].

- The LU factorization of A is computed at precision u_f, which is half precision for a full factorization or double precision when ILU and BLR are used.

- The working precision is double precision for all experiments.

- The residual is computed in quadruple precision for all experiments. Computing the residuals in extended precision improves the forward error for ill-conditioned problems, though it has no effect on the convergence of iterative refinement [6].

We set the maximum number of iterative refinement steps to 10 and the maximum number of GMRES iterations per step of iterative refinement to 100 (hence a maximum of 1000 total iterations). The GMRES stopping criterion is set to a relative tolerance of 10^{-8}.

To assess the effectiveness of each preconditioner we will measure both the number of performed iterations and the associated flops. Since our new Π_{E_k} preconditioner can and often does

Table 5.1: Five matrices representative of typical scenarios. We used a half precision LU factorization for matrices randsvd(1e7,3) and lund_a, ILU factorization with $\tau = 10^{-1}$ for matrices west0167 and rajat14, and BLR factorization with $\tau = 10^{-2}$ for matrix utm300. The seventh and eighth columns of the table show the number of GMRES iterations, with the number of iterative refinement steps in parentheses.

Matrix	ε	k	$\kappa(A)$	$\kappa(\Pi_{LU}A)$	$\kappa(\Pi_{E_k}A)$	Π_{LU} Iterations	Π_{E_k} Iterations	Π_{E_k}/Π_{LU} Flops	Π_{E_k}/Π_{LU} Time
randsvd(1e7,3)	10^{-2}	53	1.0e+07	3.3e+06	5.2e+05	200 (2)	99 (2)	161%	84%
lund_a	10^{-3}	6	2.8e+06	1.2e+08	1.7e+04	37 (3)	18 (2)	106%	55%
west0167	10^{-2}	1	4.8e+10	1.3e+18	1.8e+14	403 (7)	300 (5)	88%	76%
utm300	10^{-3}	10	8.5e+05	1.9e+06	5.3e+03	52 (3)	35 (3)	124%	76%
rajat14	10^{-2}	44	3.2e+08	9.3e+04	1.2e+03	47 (2)	26 (2)	280%	95%

perform more flops that the traditional Π_{LU} preconditioner, we also estimate the time for solution. Since a high-performance implementation and analysis is not our focus, we use a simple model, assuming BLAS 3 computations are 10 times faster than their BLAS 2 counterparts. We also assume that computations in single and half precision are twice and four times faster than in double precision, respectively.

5.1 Analysis of typical scenarios

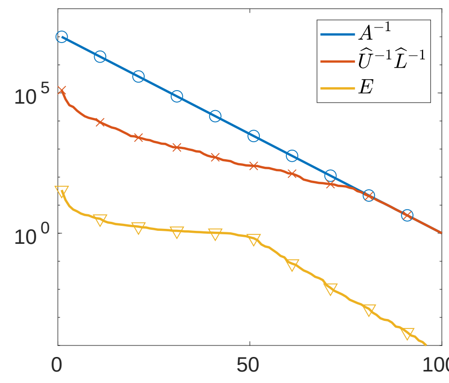
In this section, we focus on the first variant $\Pi^{(1)}_{E_k}$ and refer to it simply as Π_{E_k}. In Table 5.1, we consider five matrices that are representative of five typical scenarios. We report the condition numbers of A, $\Pi_{LU}A$, and $\Pi_{E_k}A$. For both preconditioners, we compare the total number of GMRES iterations (the number in parentheses corresponds to the number of iterative refinement steps), and the associated flops and estimated time. We also indicate the low-rank threshold ε that we used to compute E_k, and its corresponding rank k. In Figure 5.1, we plot the singular value distribution of A^{-1}, $\widehat{U}^{-1}\widehat{L}^{-1}$, and E for each of these five matrices.

We recall that our preconditioner targets ill-conditioned matrices that have a small number of small singular values and therefore a numerically low-rank inverse. We have observed that *all* ill-conditioned matrices that we have tested from the SuiteSparse Matrix Collection fulfill that requirement. The only matrices in our set that do not are the mode 1 and 3 randsvd matrices, which are artificially created problems. They constitute what we will call Scenario 1. Mode 3 randsvd is analyzed in Figure 5.1a. Interestingly, the SVD of the inverse factors shows a slightly faster singular value decay and therefore some improvement is observed with Π_{E_k} over Π_{LU}. However, the rank k is about $n/2$, leading to an important flop overhead and thus only a modest time gain.

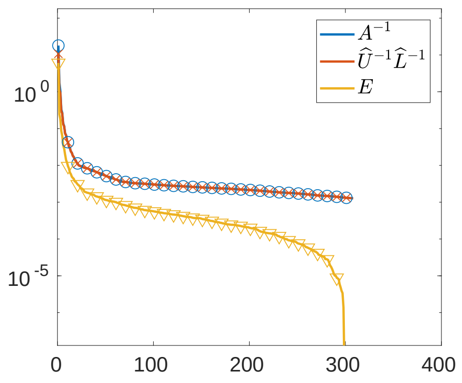
We emphasize that we selected the matrices based on their condition number only; we did not specifically select matrices for which A^{-1} is numerically low rank. While one can surely find an ill-conditioned matrix from the SuiteSparse collection that does not fulfill this requirement, we believe that the fact that one does not easily come upon one of them demonstrates that our preconditioner's scope is extremely general.

While A^{-1} is thus numerically low rank for nearly all matrices in the test set, the performance of our Π_{E_k} preconditioner is heavily dependent on the extent to which the error E is numerically low rank. In the following Scenarios 2, 3, and 4, E is numerically low rank and thus Π_{E_k} performs well. In Scenario 2, the SVD of E closely follows that of A^{-1} (Figure 5.1b); in other cases, the SVD of E shows an even faster decay than that of A^{-1}, either because $\widehat{U}^{-1}\widehat{L}^{-1}$ is more low rank than A^{-1} (Scenario 3, Figure 5.1c), or because E is more low rank than $\widehat{U}^{-1}\widehat{L}^{-1}$ (Scenario 4, Figure 5.1d). Scenario 3 generally happens when the approximate $\widehat{L}\widehat{U}$ factors are nearly singular, thus leading to a very ill-conditioned $\Pi_{LU}A$ matrix. By using Π_{E_k}, we can correct the ill conditioning of $\Pi_{LU}A$. We conjecture that Scenario 4 is due to a ΔA that possesses some kind of structure, and we have in fact observed it to be especially frequent for the test cases with BLR factorization.

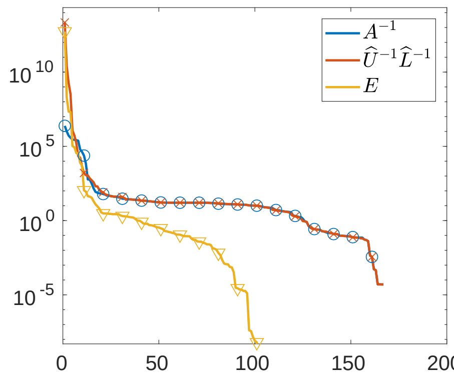
Finally, Scenario 5 contains the unfortunate cases for which E loses the low-rank property of



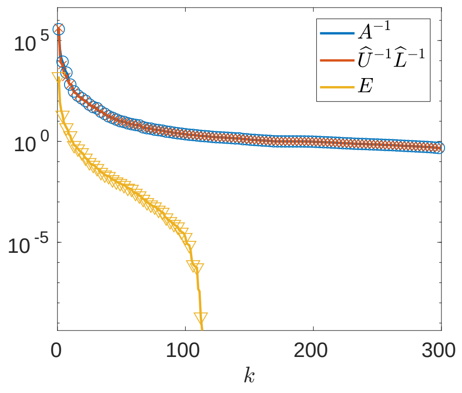
(a) Matrix randsvd(1e7,3) (fp16).



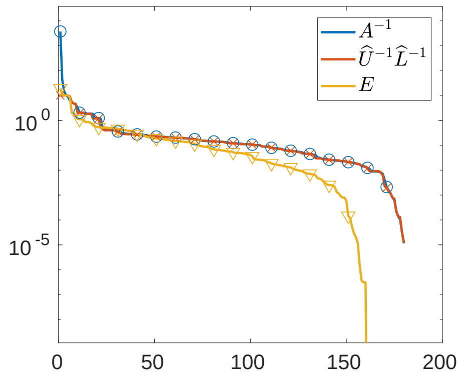
(b) Matrix cz308 (fp16).



(c) Matrix west0167 (ILU, $\tau = 10^{-1}$).



(d) Matrix utm300 (BLR, $\tau = 10^{-2}$).



(e) Matrix rajat14 (ILU, $\tau = 10^{-1}$).

Figure 5.1: Singular value distribution of the five matrices in Table 5.1.

A^{-1} (Figure 5.1e). In our set of matrices, this is always due to $\widehat{U}^{-1}\widehat{L}^{-1}$ not being low rank (i.e., the bound from Theorem 3.1 is sharp and β_g or β_s is large). We recall that in section 3 we built a matrix for which E loses the low-rank property due to a special ΔA (see (3.11)), but this did not occur on any of the matrices of our set.

The numerical rank k_ε of E at accuracy ε can be quite large for matrices falling into Scenarios 1 and 5. To limit the cost of Π_{E_k}, we can limit k to be no larger than a given k_{max}. For example, for rajat14, using $k_{max} = n/10$, the flop cost of Π_{E_k} is reduced from 280% to 170% of that of Π_{LU}, and the time gain is increased from 95% to 92%.

This diversity of scenarios shows that the optimal choice of the preconditioner parameters will be heavily matrix dependent. However, we would like to design a "black box" version of the preconditioner that has default settings for which it performs well on a wide range of problems. This is the aim of the next section.

5.2 Finding a black box setting

Three main parameters influence the cost and accuracy of the preconditioner Π_{E_k}: the precision u_{E_k} at which E_k is computed, the low-rank threshold ε, and the amount of oversampling p. In this section, we analyze how to set these parameters to produce good performance on a wide range of problems. In order to do that we seek the best value for each parameter separately, using performance profiles [10], [16, sec. 26.4]. Each performance profile corresponds to a preconditioner, a selection of three or four parameters, and a chosen performance measure for which smaller is better. Each curve on a performance profile shows, for a range of values of $\alpha \geq 1$, the proportion of problems $p \in [0,1]$ for which the performance measure for a particular parameter was within a factor α of the smallest performance measure over all the parameter values. The performance measures are the number of iterations, the number of flops, and the time predicted by our model.

Note that if the iteration fails to converge for some problems for a given parameter then the corresponding curve in the performance profile never reaches $p = 1$; thus the value of p at which a curve levels off is a measure of robustness.

Naturally, the parameters are interdependent: for example, a high oversampling parameter will increase the weight of the sampling operation, which is performed at precision u_{E_k}, thus increasing the importance of the latter parameter. While the approach of studying each parameter independently is thus possibly not optimal, it allows us to find a suitable setting without getting lost into the combinatorics of the parameters.

We first analyze the influence of the oversampling parameter p. From Figure 5.2, it is clear that a larger oversampling leads to a greater reduction of the number of iterations, but also to a greater flop overhead due to the larger subspace size $\ell = k+p$. We must therefore find a compromise aiming at minimizing the time estimated by our model. Interestingly, the time performance profiles suggest that the value for p should be set differently depending on which Π_{E_k} variant is considered. Indeed, $\Pi^{(1)}_{E_k}$ and $\Pi^{(2)}_{E_k}$ require us to form the product V^TE (see section 4.3) and their cost is thus very sensitive to the choice of p; setting it to a small value works best. Conversely, $\Pi^{(3)}_{E_k}$ and $\Pi^{(4)}_{E_k}$ avoid forming V^TE, and we can thus afford to take much higher values of p, since building the preconditioner is much cheaper. This is visible from the time plots (right column) in Figure 5.2, where the curves corresponding to small p tend to be above those corresponding to large p for $\Pi^{(1)}_{E_k}$ (top row), while the opposite is true for $\Pi^{(4)}_{E_k}$ (bottom row). Results for $\Pi^{(2)}_{E_k}$ and $\Pi^{(3)}_{E_k}$ variants (not shown in Figure 5.2), lie in the middle ground. In the following, we will therefore use $p = 0$ for $\Pi^{(1)}_{E_k}$ and $\Pi^{(2)}_{E_k}$, and $p = 10$ for $\Pi^{(3)}_{E_k}$ and $\Pi^{(4)}_{E_k}$.

We now turn to the low-rank threshold parameter ε, whose effect is plotted in Figure 5.3. The trend is again clear: a smaller value of ε makes the preconditioner more robust but more costly. The role of ε is also strongly dependent on which variant of Π_{E_k} is considered, for the same reasons than the oversampling parameter. In the following, we will use $\varepsilon = 10^{-3}$ for $\Pi^{(1)}_{E_k}$ and $\Pi^{(2)}_{E_k}$, and $\varepsilon = 10^{-5}$ for $\Pi^{(3)}_{E_k}$ and $\Pi^{(4)}_{E_k}$.

Finally, in Figure 5.4 we study the role of the u_{E_k} precision parameter on the subset of tests

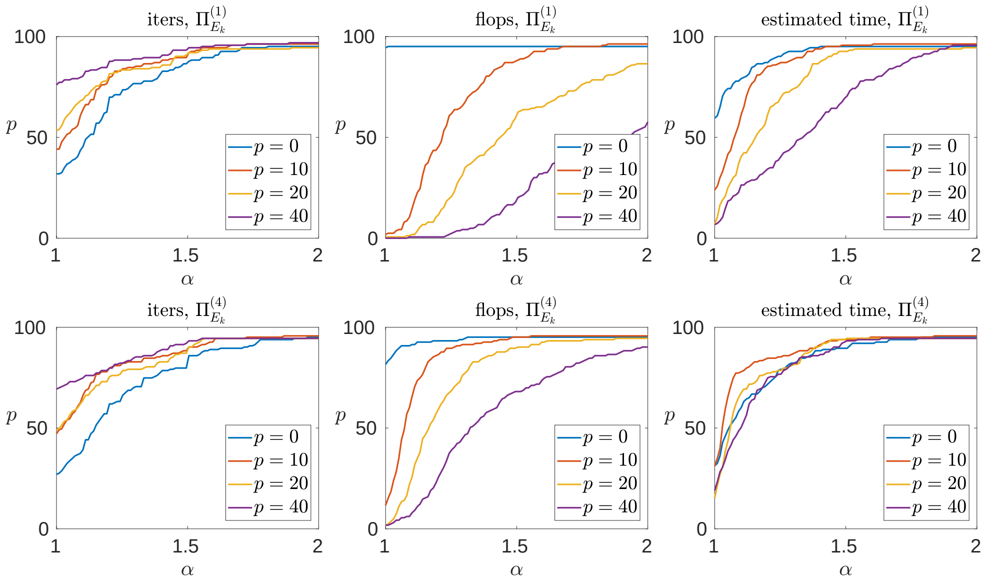


Figure 5.2: Performance profile of the Π_{E_k} preconditioner for different oversampling parameters p. The other parameters were set to $\varepsilon = 10^{-5}$ and u_{E_k} = single.

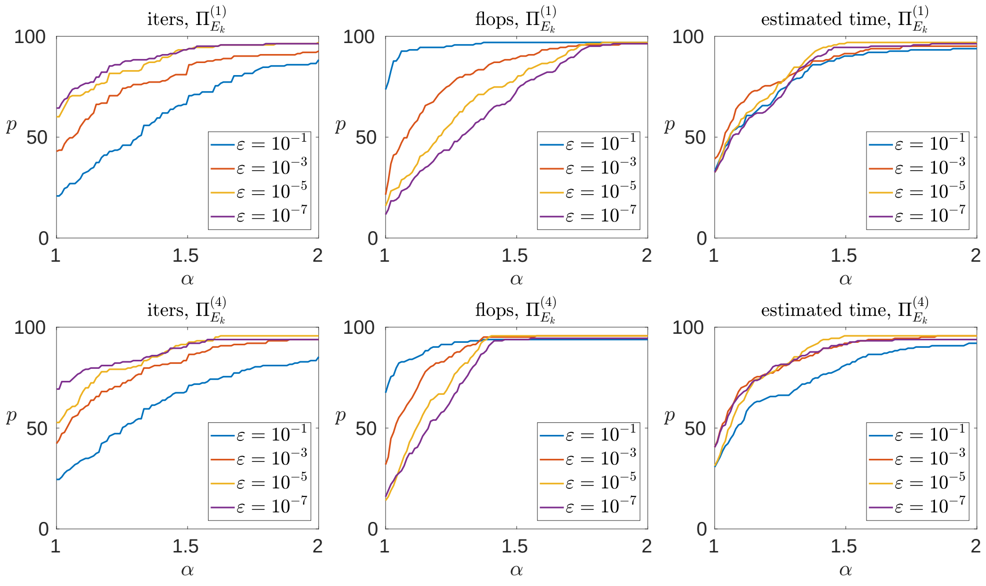


Figure 5.3: Performance profile of the Π_{E_k} preconditioner for different low-rank threshold ε parameters. The other parameters were fixed to $p = 10$ and u_{E_k} = single.

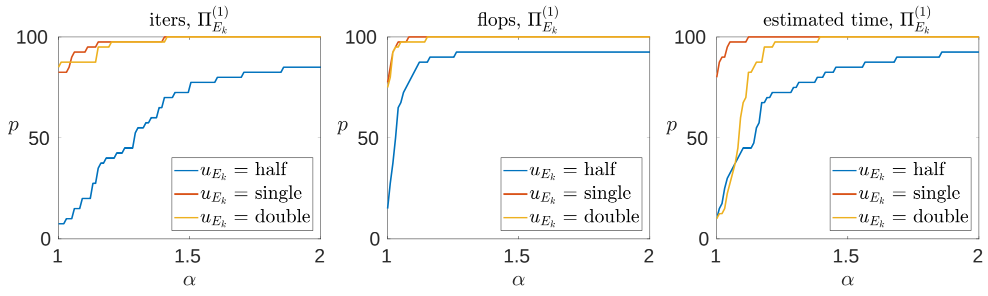


Figure 5.4: Performance profile of the $\Pi_{E_k}^{(1)}$ preconditioner for an LU factorization computed in half precision for different u_{E_k} precision parameters, with $\varepsilon = 10^{-5}$ and $p = 10$. Results with $\Pi_{E_k}^{(2,3,4)}$ are similar.

Table 5.2: Black box settings devised in section 5.2.

	ε	p	u_{E_k}
$\Pi_{E_k}^{(1)}$	10^{-3}	0	single
$\Pi_{E_k}^{(2)}$	10^{-3}	0	single
$\Pi_{E_k}^{(3)}$	10^{-5}	10	single
$\Pi_{E_k}^{(4)}$	10^{-5}	10	single

performed with half precision LU factorization (fp16). Computing E_k in half precision leads to a preconditioner that is less accurate than when E_k is computed in higher precision: in particular, in about 8% of the cases the preconditioner fails when E_k is built in half precision, whereas it succeeds with a higher precision u_{E_k}. On the other hand, computing E_k in single or double precision makes little difference on this set of problems, and since single precision is twice as fast as double precision, the time performance profile shows that setting u_{E_k} to single is the best strategy overall, for all four variants of Π_{E_k}.

5.3 Results on the full set of problems with the black box setting

In this section we report numerical experiments on the full set of problems using the black box settings chosen in the previous section, which we summarize in Table 5.2.

We emphasize that the results were obtained without tuning the preconditioner parameters on a case-by-case basis, thereby demonstrating the generality and versatility of the preconditioner.

We first compare the four variants of the Π_{E_k} preconditioner. Figure 5.5 shows the time performance profile of each variant. Note that we do not provide the iterations and flop profiles, since comparing the four variants in terms of iterations or flops is not meaningful, because they are used with different values of ε and/or p. We must compare their time performance to assess which variant finds the best cost/accuracy compromise. The preconditioner $\Pi_{E_k}^{(4)}$ ranks first on the largest number of problems (about 50% of them); it is, however, less robust than the other variants, failing to converge in three cases where the other variants converged. We therefore choose to reject it. While $\Pi_{E_k}^{(1)}$ and $\Pi_{E_k}^{(2)}$ are significantly slower, $\Pi_{E_k}^{(3)}$ achieves a good performance overall, very close to that of $\Pi_{E_k}^{(4)}$. Interestingly, it is also the most robust variant; recalling that it is less accurate than $\Pi_{E_k}^{(1)}$ by a factor up to $1+\sqrt{1+4k(n-k)}$, this means that it compensates its lesser accuracy by being able to afford a much smaller threshold ($\varepsilon = 10^{-5}$ instead of 10^{-3}). We conclude that $\Pi_{E_k}^{(3)}$ is the choice that leads to the best performance overall on this set of problems.

We now compare $\Pi_{E_k}^{(3)}$ with the classical Π_{LU} preconditioner. In Figure 5.6, we plot the relative

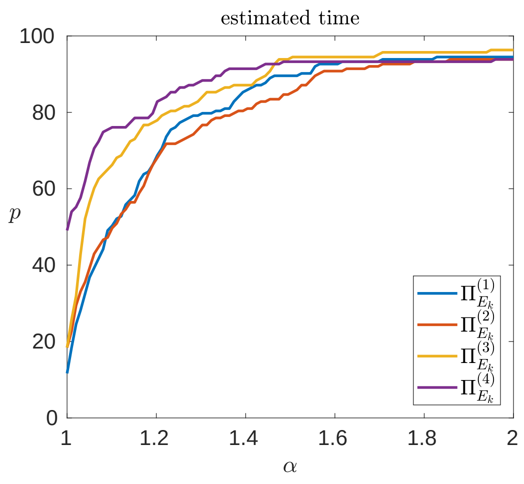


Figure 5.5: Time performance profile of the four variants of the Π_{E_k} preconditioner, obtained with the black box settings described in Table 5.2.

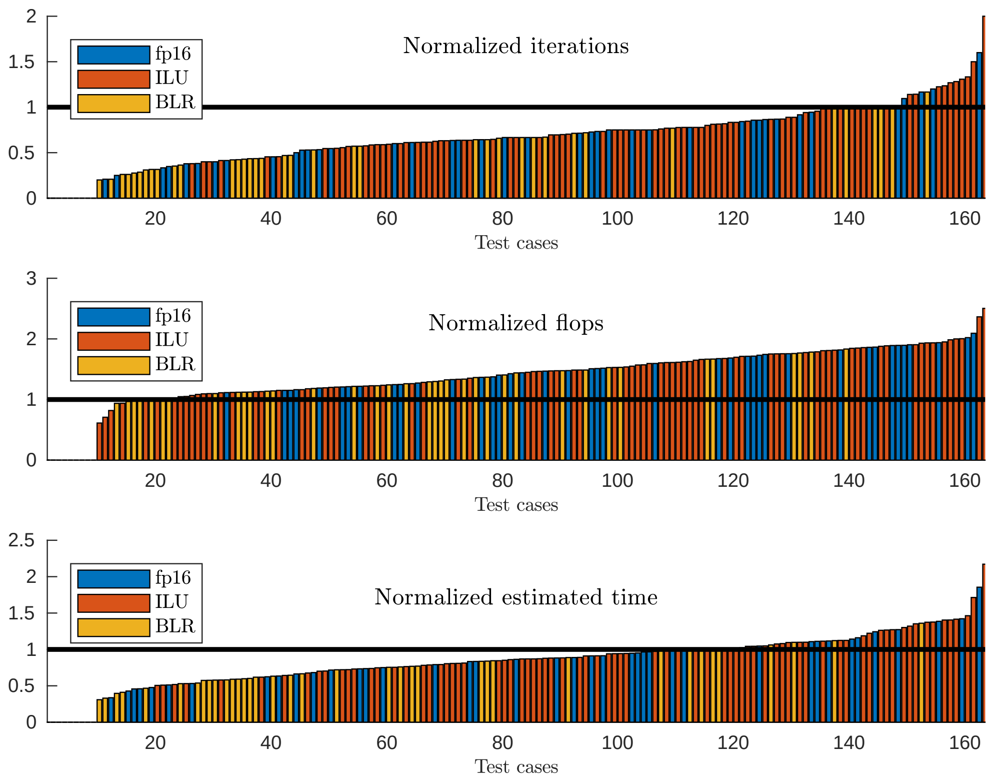


Figure 5.6: Performance comparison between the Π_{LU} preconditioner and the best variant of the Π_{E_k} preconditioner ($\Pi_{E_k}^{(3)}$). Each bar corresponds to one of the 163 test cases, its color indicating which type of approximate factorization is considered (fp16, ILU, or BLR). The y-axis corresponds to the normalized performance of Π_{E_k} with respect to that of Π_{LU}: thus, Π_{E_k} performs better than Π_{LU} when the bar is under the black line. These results were obtained with the black box settings described in Table 5.2. The white gap on the left side of the plots corresponds to the test cases for which Π_{LU} did not converge whereas Π_{E_k} did.

performance of $\Pi_{E_k}^{(3)}$ with respect to Π_{LU}. Each bar corresponds to a different test case, its color indicating which type of approximate factorization is considered (fp16, ILU, or BLR). The colors are evenly distributed, which means that the numerical behavior of $\Pi_{E_k}^{(3)}$ is comparable for all three types of factorization. The preconditioner $\Pi_{E_k}^{(3)}$ leads to a lower number of iterations than Π_{LU} in about 80% of the test cases. Moreover, this reduction of the number of iterations is often important: Π_{LU} performs more than 50% more iterations on 30% the test cases. Interestingly, in about 5% of the cases, Π_{LU} fails to converge whereas $\Pi_{E_k}^{(3)}$ successfully solves the problem (indicated by the white gap on the left side of the plots). Therefore, even though $\Pi_{E_k}^{(3)}$ leads to a flop overhead compared with Π_{LU} in about 90% of the cases, that overhead is often limited (less than 50% overhead in half the cases) and using our simple performance model, the estimated time results suggest significant gains can be expected.

6 Conclusion

We have presented a new and very general preconditioner for ill-conditioned linear systems $Ax = b$.

The key idea is to exploit the low numerical rank structure that is typically present in the error arising in approximate matrix factorizations. We have defined a general framework in which a low accuracy LU factorization $A = \widehat{L}\widehat{U} + \Delta A$ is computed. This allows for many different types of approximate LU factorizations, among which in our experiments we have used half precision LU, incomplete LU, and block low-rank LU.

We have used theoretical results from singular value perturbation analysis to bound the distance from $E = \widehat{U}^{-1}\widehat{L}^{-1}A - I = \widehat{U}^{-1}\widehat{L}^{-1}\Delta A$ to a numerically low-rank matrix by the distance from A^{-1} to a numerically low-rank matrix. These bounds are pessimistic and we have found E to be almost always numerically low rank in practice when A is ill conditioned.

Our novel preconditioner improves the traditional preconditioner $\widehat{U}^{-1}\ \widehat{L}^{-1}$ based on the approximate LU factors by premultiplying it by a correction term $(I+E_k)^{-1}$, exploiting the numerical low rank of E. Because building E explicitly is too expensive, our algorithm uses a matrix-free approach based on randomized sampling to compute a rank-k matrix E_k as a truncated SVD of E. We have compared four variants of the algorithm theoretically, by performing a computational cost analysis, and experimentally.

After experimenting with the internal parameters of the preconditioner, in order to better understand its practical behavior, we chose a set of parameters that we applied in a black box manner to a large set of real-life problems coming from a variety of applications. Our numerical results show the capacity of the new preconditioner to accelerate the solution of a wide range of ill-conditioned problems, thereby demonstrating its generality and versatility.

We conclude by mentioning some possible directions for future work. Our preconditioner could be coupled with other iterative methods than GMRES-IR, such as GMRES. The LU framework that we have described could also be naturally adapted to symmetric problems. We believe our work could even be extended to preconditioners that are not based on matrix factorizations, such as Jacobi, Gauss-Seidel, approximate inverse, or multigrid approaches.

Most importantly, while out of the scope of this article, a high-performance implementation of the proposed preconditioner will be of interest both to assess the performance gains that can be achieved and to study its numerical behavior on large-scale problems. In particular, an important question is whether large ill-conditioned matrices still possess an inverse that is numerically low rank, and whether the numerical rank k_ε of A^{-1} remains small or on the contrary increases with n. In Figure 6.1, we compare k_ε for different matrix sizes for two families of matrices from the SuiteSparse Matrix Collection that contain cz308 and utm300 in Table 2.1. The plots show that the numerical rank remains almost constant with respect to n if the required ε is not too small. While there may of course be some other matrices for which this property is not true, the figure suggests that, at least for some problem classes, our preconditioner should perform well, or even better, on large-scale problems.

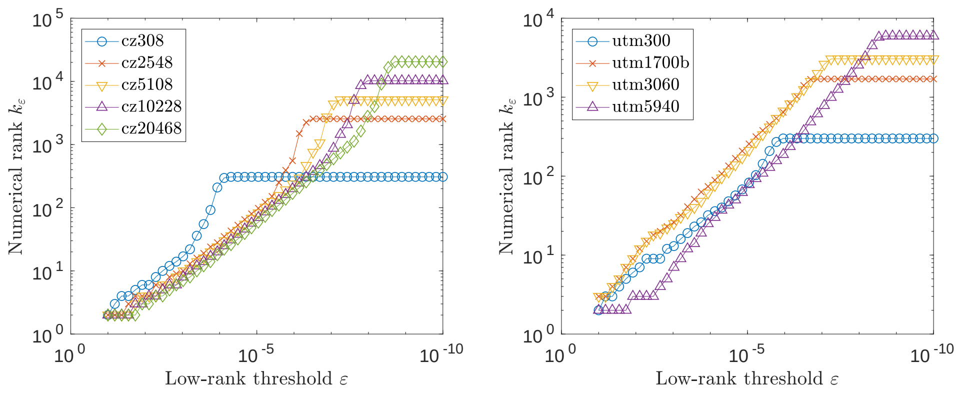


Figure 6.1: Numerical rank k_ε of A^{-1} at accuracy ε for different matrix sizes. The matrices are from the SuiteSparse Matrix Collection and the digits in the name denote the matrix size.

Acknowledgements

We thank Patrick R. Amestoy and Alfredo Buttari for useful discussions.

References

[1] Patrick Amestoy, Cleve Ashcraft, Olivier Boiteau, Alfredo Buttari, Jean-Yves L'Excellent, and Clément Weisbecker. Improving multifrontal methods by means of block low-rank representations. *SIAM J. Sci. Comput.*, 37(3):A1451–A1474, 2015.

[2] Patrick Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. On the complexity of the block low-rank multifrontal factorization. *SIAM J. Sci. Comput.*, 39(4):A1710–A1740, 2017.

[3] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Trans. Math. Software*, 2017. Submitted.

[4] Mario Bebendorf. *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, volume 63 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, Berlin, 2008. xvi+290 pp. ISBN 978-3-540-77146-3.

[5] Erin Carson and Nicholas J. Higham. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM J. Sci. Comput.*, 39(6): A2834–A2856, 2017.

[6] Erin Carson and Nicholas J. Higham. Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM J. Sci. Comput.*, 40(2):A817–A847, 2018.

[7] Tony F. Chan and David E. Foulser. Effectively well-conditioned linear systems. *SIAM J. Sci. Statist. Comput.*, 9(6):963–969, 1988.

[8] H. Cheng, Z. Gimbutas, P. G. Martinsson, and V. Rokhlin. On the compression of low rank matrices. *SIAM J. Sci. Comput.*, 26(4):1389–1404, 2005.

[9] Timothy A. Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Software*, 38(1):1:1–1:25, 2011.

[10] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Programming*, 91:201–213, 2002.

[11] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[12] J. A. George. Nested dissection of a regular finite-element mesh. *SIAM J. Numer. Anal.*, 10 (2):345–363, 1973.

[13] Azzam Haidar, Panruo Wu, Stanimire Tomov, and Jack Dongarra. Investigating half precision arithmetic to accelerate dense linear system solvers. In *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA '17, November 2017, pages 10:1–10:8.

[14] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, 2011.

[15] H. V. Henderson and S. R. Searle. On deriving the inverse of a sum of matrices. *SIAM Rev.*, 23(1):53–60, 1981.

[16] Desmond J. Higham and Nicholas J. Higham. *MATLAB Guide*. Third edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2017. xxvi+476 pp. ISBN 978-1-61197-465-2.

[17] Nicholas J. Higham. Optimization by direct search in matrix computations. *SIAM J. Matrix Anal. Appl.*, 14(2):317–333, 1993.

[18] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1991. viii+607 pp. ISBN 0-521-30587-X.

[19] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Second edition, Cambridge University Press, Cambridge, UK, 2013. xviii+643 pp. ISBN 978-0-521-83940-2.

[20] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

[21] Théo Mary. *Block Low-Rank Multifrontal Solvers: Complexity, Performance, and Scalability*. PhD thesis, Université de Toulouse, Toulouse, France, November 2017.

[22] Théo Mary, Ichitaro Yamazaki, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Jack Dongarra. Performance of random sampling for computing low-rank approximations of a dense matrix on GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, New York, NY, USA, 2015, pages 60:1–60:11. ACM.

[23] L. Mirsky. Symmetric gauge functions and unitarily invariant norms. *Quart. J. Math.*, 11: 50–59, 1960.

[24] Cleve B. Moler. Cleve Laboratory. `http://mathworks.com/matlabcentral/fileexchange/59085-cleve-laboratory`.

[25] Multiprecision Computing Toolbox. Advanpix, Tokyo. `http://www.advanpix.com`.

[26] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003. xviii+528 pp. ISBN 0-89871-534-2.