RustNLA: Randomized Linear Algebra Project

CS-IS-3077-1: Rustification - An Introduction to Rust

Project Report

Divij Khaitan, Saptarishi Dhanuka

November 25, 2024

Contents

1 Introduction & Outline

Numerical linear algebra (NLA) is the study of algorithms for matrix operations that can run efficiently on computers. It is an indispensable part of most aspects of science and engineering. However, with extremely large numerical problems come extremely large matrices, such as those in machine learning. This slows down regular deterministic NLA algorithms by a large extent. However, if we are willing to accept some degree of error in our answers to get approximate solutions to our problems, then we can get much faster algorithms through the use of randomization, due to operations on smaller representations of large matrices, with few good properties. [5]

We develop a package for randomized numerical linear algebra (RandNLA) in Rust mostly based on this [5] paper and this C++ codebase. This wasn't an explicit translation project since we implemented a few components that weren't mentioned in the existing code. Most of the mathematical background needed came from the paper.

The primary motivation behind doing this project for the Rust Programming course is that the original codebase is in C++, which is famous for it's issues with memory safety. Rust bypasses these issues while offering comparable performance. Moreover, to the best of our knowledge, no such package exists in Rust yet.

2 Main steps

The broad steps we carried out during our implementation were as follows:

1. **Random number generation:** `rust-random123`
   We looked at rust-random123 which ported portions of DE Shaw's library to Rust. We spent some time in debugging the poorly maintained library but eventually made it work.

2. **Sketching Operators:** `sketch.rs`
   A core part of RandNLA, these operators transform a high dimensional space into a lower dimensional space by pre or post multiplication, producing smaller matrices. The types outlined in the C++ library are iid sampled entries from the rademacher, gaussian and uniform distributions, which are the ones we implemented. Along with data-oblivious operators, we also implement a data-aware operator `tsog1` which takes into account the matrix to be sketched.

3. **Deterministic Solvers:** `cg.rs & solvers.rs`
   For making randomized solvers, we first needed to implement deterministic solvers that can work with the smaller matrices that would be produced by sketching and randomization methods. For this we implemented the following:

   - Common Conjugate Gradient Method (CG) [2]
   - CGLS (a more stable method on finite precision arithmetic) [2]
   - LSQR (more stable than CG and CGLS for ill-conditioned problems) (we translated the implementation given in the scipy library
   - Upper triangular and diagonal solvers, for special classes of matrices which also work for least squares problems

4. **Randomized Least Squares:** `sketch_and_precondition.rs & sketch_and_solve.rs`
We implemented various algorithms for solving linear systems and least squares problems utilizing the benefits of sketching operators. The first two are sketch-and-solve methods that solve a sketch of the system of equations. The next three sketch-and-precondition methods use the structure of the problem and some randomisation to solve similar problems in a much smaller amount of time. [5]

   - `sketched_least_squares_qr` : Using QR decomposition to solve least squares.
   - `sketched_least_squares_svd` : Using SVD to solve least squares.
   - `blendenpik_overdetermined` : Implements Blendenpik algorithm for solving overdetermined linear systems [1]
   - `lsrn_overdetermined` : Implements LSRN algorithm for solving overdetermined linear systems [4]
   - `sketch_saddle_point_precondition` : Preconditions and solves the dual and primal forms of the saddle-point problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2 + \mu\|\mathbf{x}\|_2^2 + 2\mathbf{c}^*\mathbf{x}.$$

5. **Randomised QR Decomposition:** `cqrrpt.rs`
`CQRRPT:` (CholeskyQR with Randomization and Pivoting for Tall Matrices) [3]: This computes the QR Decomposition of a matrix using the choleskyQR method. It uses the triangular QR factor from a skecth of A as a preconditioner and returns the upper triangular part from the Cholesky Decomposition of $A_{pre}^* A_{pre}$. Randomised preconditioning extends this to work for tall rank-deficient matrices. This can also be extended to matrices with arbitrary structure by calling it as a subroutine in a more complex algorithm called Householder QR with randomisation for column pivoting (HQRRP).

6. **Low-rank matrix decompositions:** `lora_drivers.rs & id.rs`
The original matrix A is approximated by some low-rank version A_r such some loss between the two is minimized. Standard decompositions are then applied to A_r. The low-rank approximation is computed using subroutines like the QB decomposition, ultimately based on data-aware sketching operators. Below are the specific decompositions implemented:

   - `Randomized SVD:` Constructs the SVD of A_r, a low-rank approximation of A, reducing computation by projecting onto a smaller subspace.
   - `Randomized EVD1:` Approximates dominant eigenpairs of Hermitian matrices.
   - `Randomized EVD2:` Approximates dominant eigenpairs of positive semi-definite matrices.
   - `CUR Decompositions:` Constructs $A_r \approx CUR$, where C and R are subsets of columns and rows of A, respectively. The linking matrix U ensures the approximation accuracy while preserving sparsity.
   - `One-sided & Two-sided Interpolative Decomposition (ID):`
   - One-sided ID: Produces $A \approx CX$, with C as a small subset of columns and X interpolating others.
   - Two-sided ID: Extends one-sided ID to both rows and columns, forming $A \approx ZA[I,J]X$, where $A[I,J]$ is a submatrix, and Z, X are interpolation matrices.

3 Results

3.1 Benchmarks

After comprehensively testing our code, we benchmarked our code against the best deterministic algorithms of Rust and observed clear speedups with acceptable error, as shown in Figures 1-10.

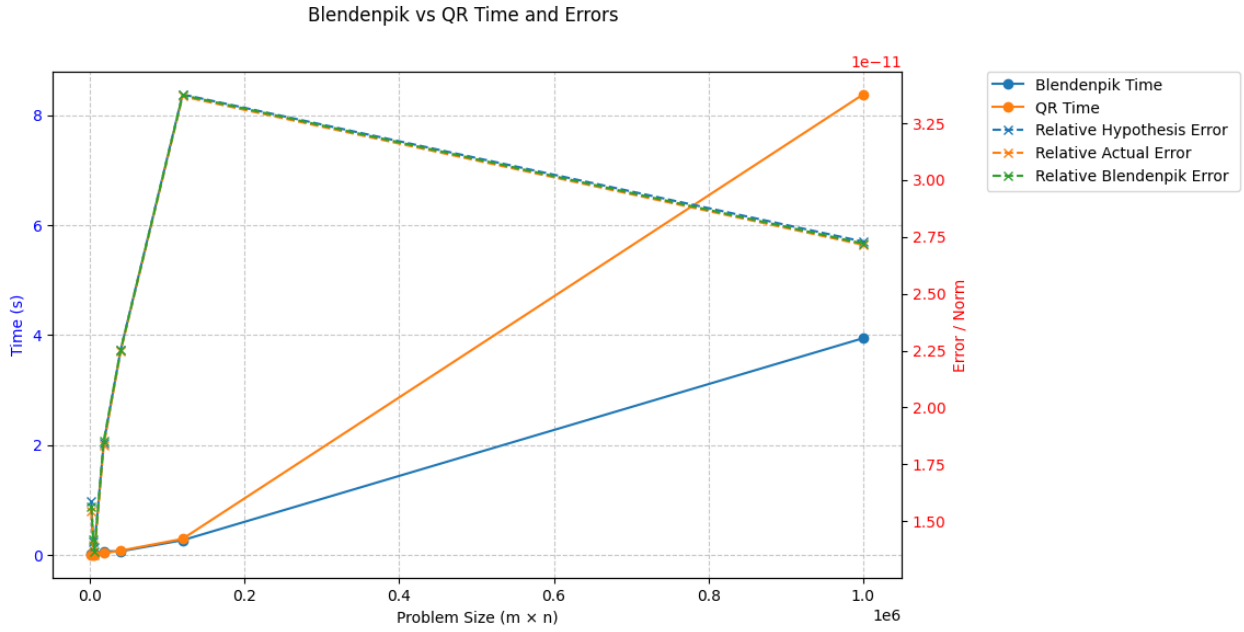


Figure 1: Blendenpik Overdetermined Least Squares Benchmark

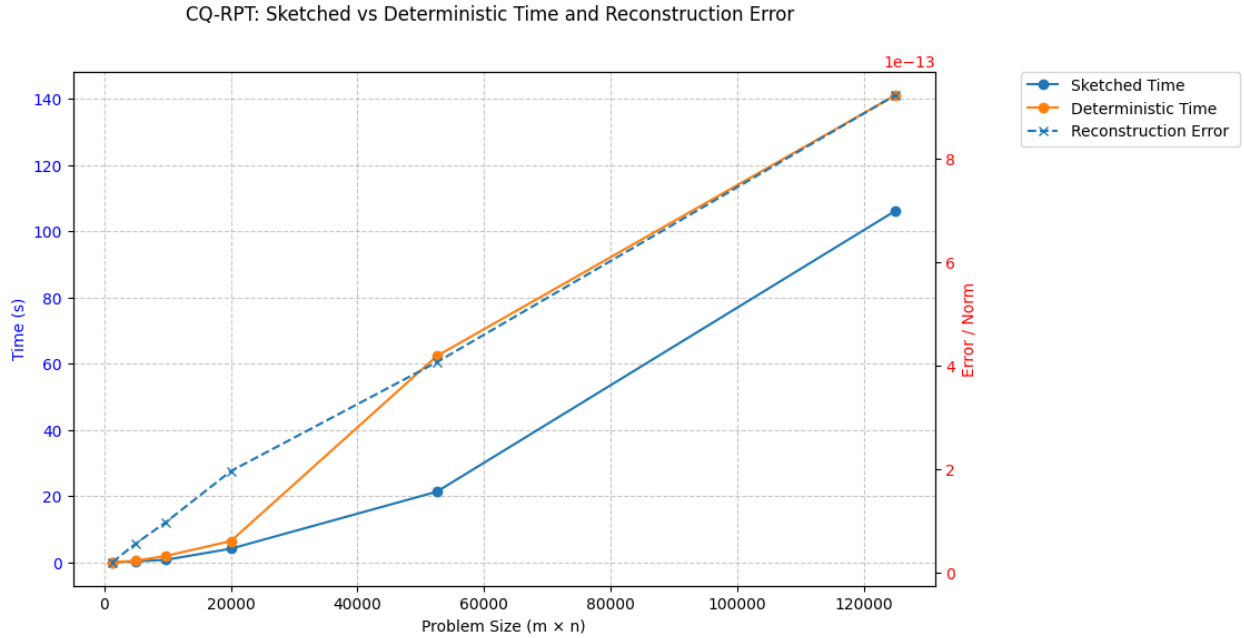


Figure 2: CQRRPT Benchmark

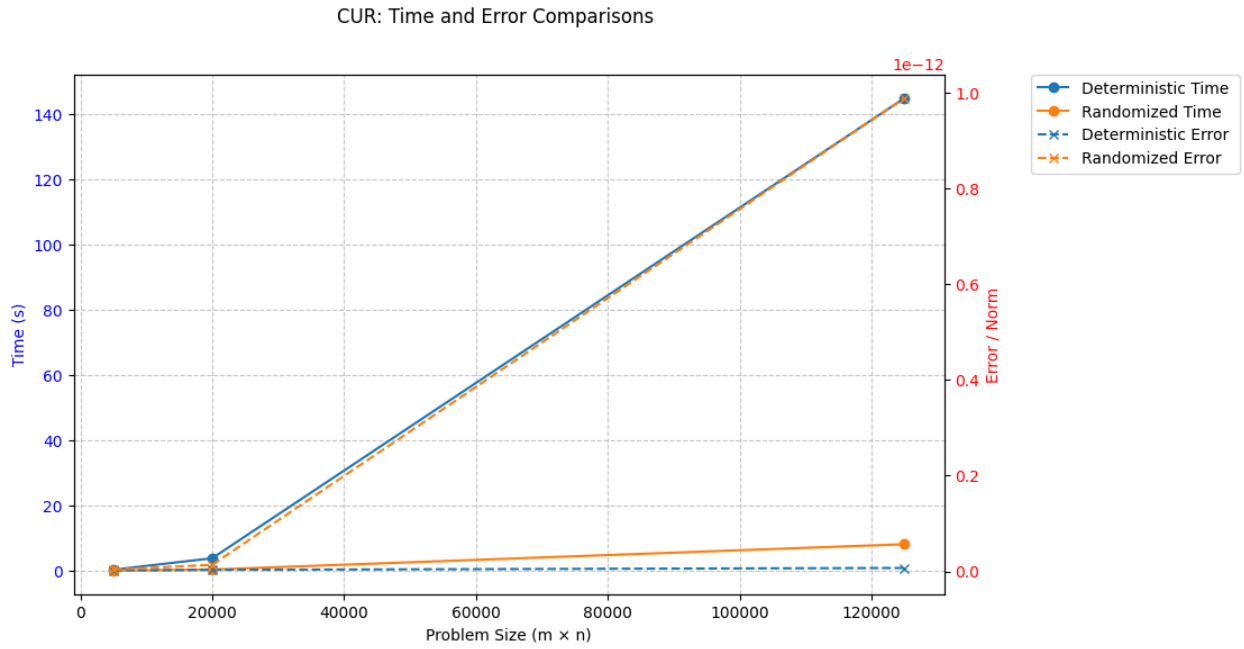


Figure 3: CUR Decomposition Benchmark

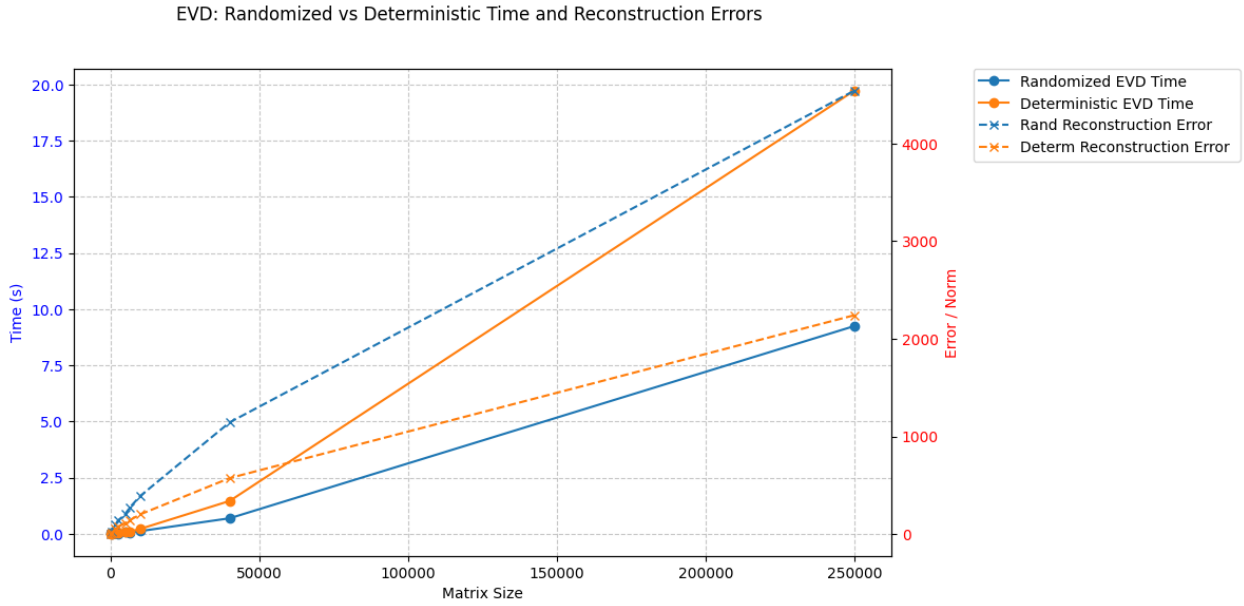


Figure 4: Eigendecomposition Benchmark

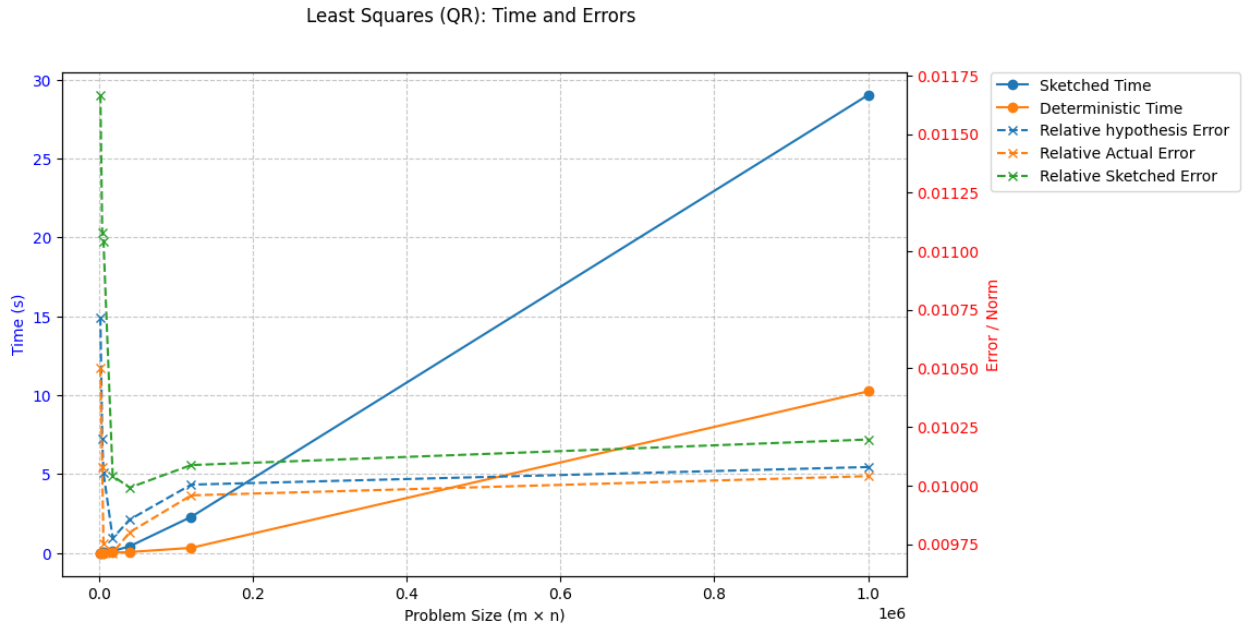


Figure 5: Least Square QR Benchmark

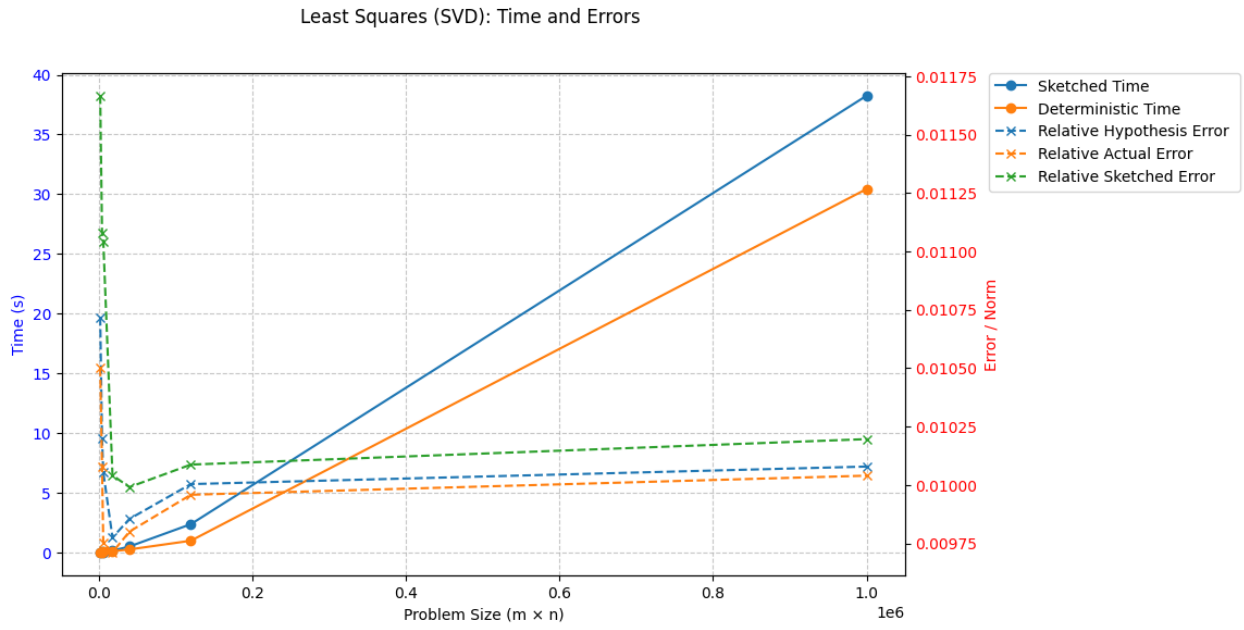


Figure 6: Least Squares SVD Benchmark

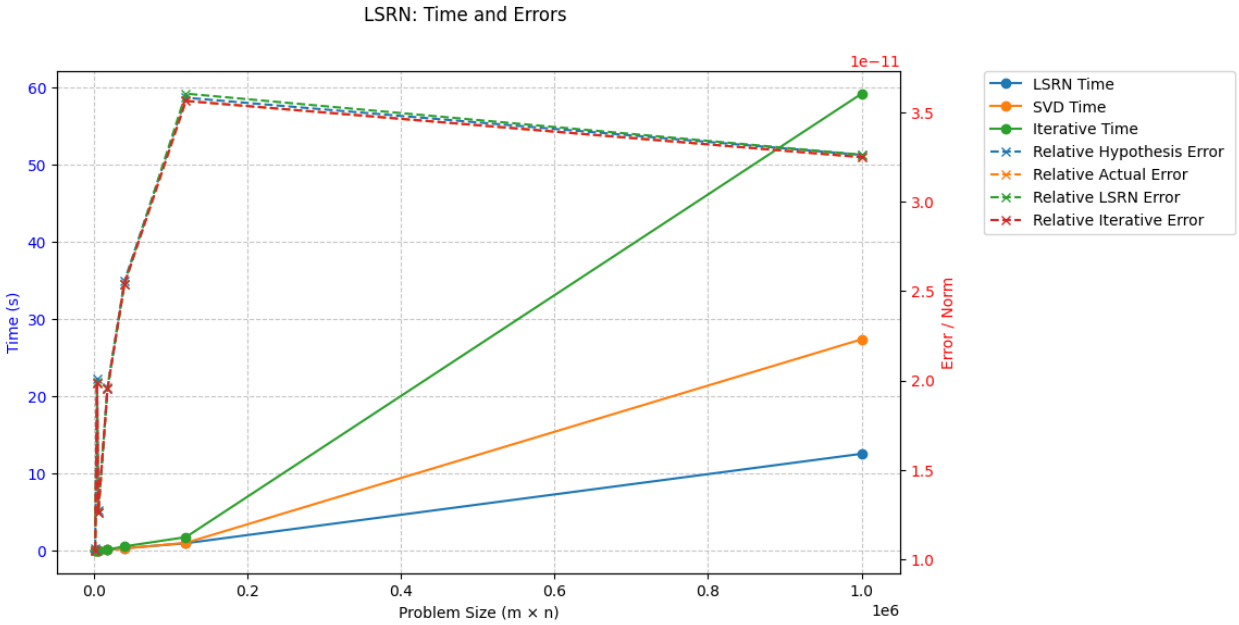


Figure 7: LSRN Overdetermined Least Squares Benchmark

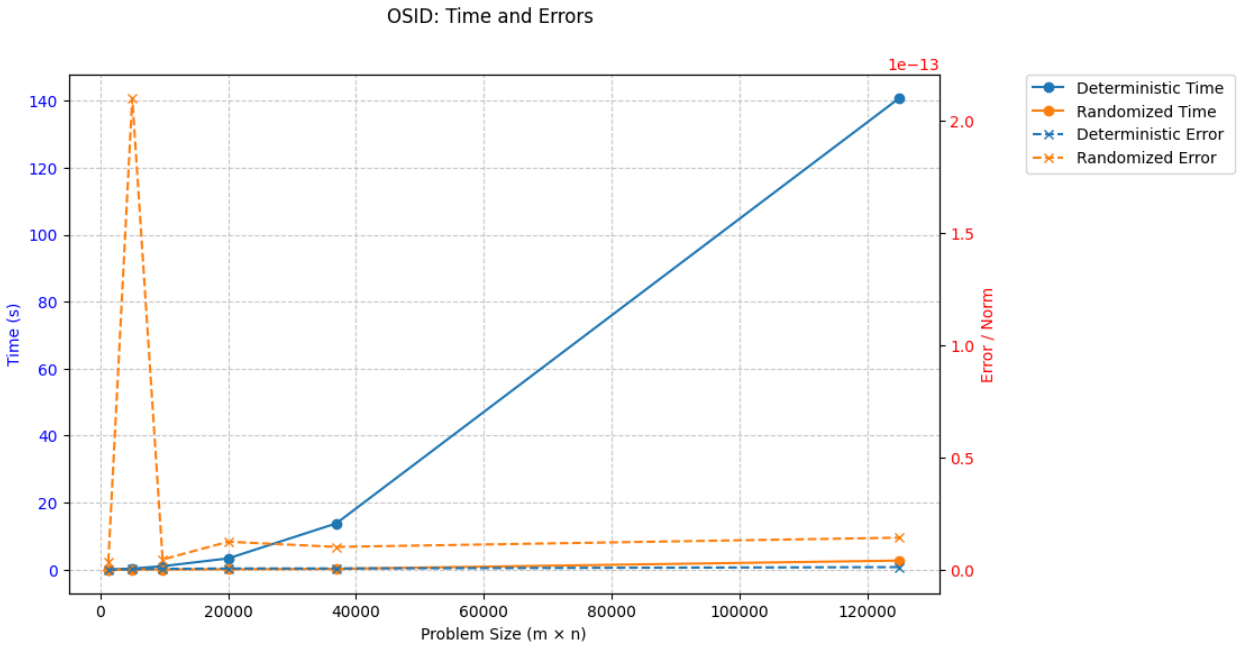


Figure 8: OSID Interpolative Decomposition Benchmark

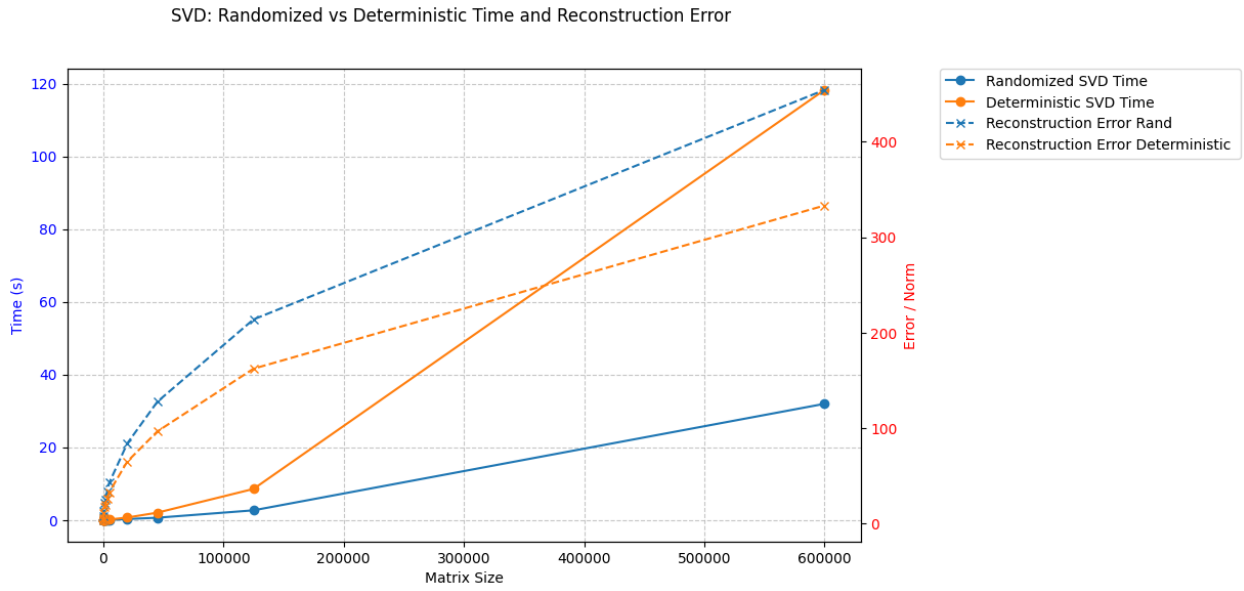


Figure 9: RandSVD Benchmark

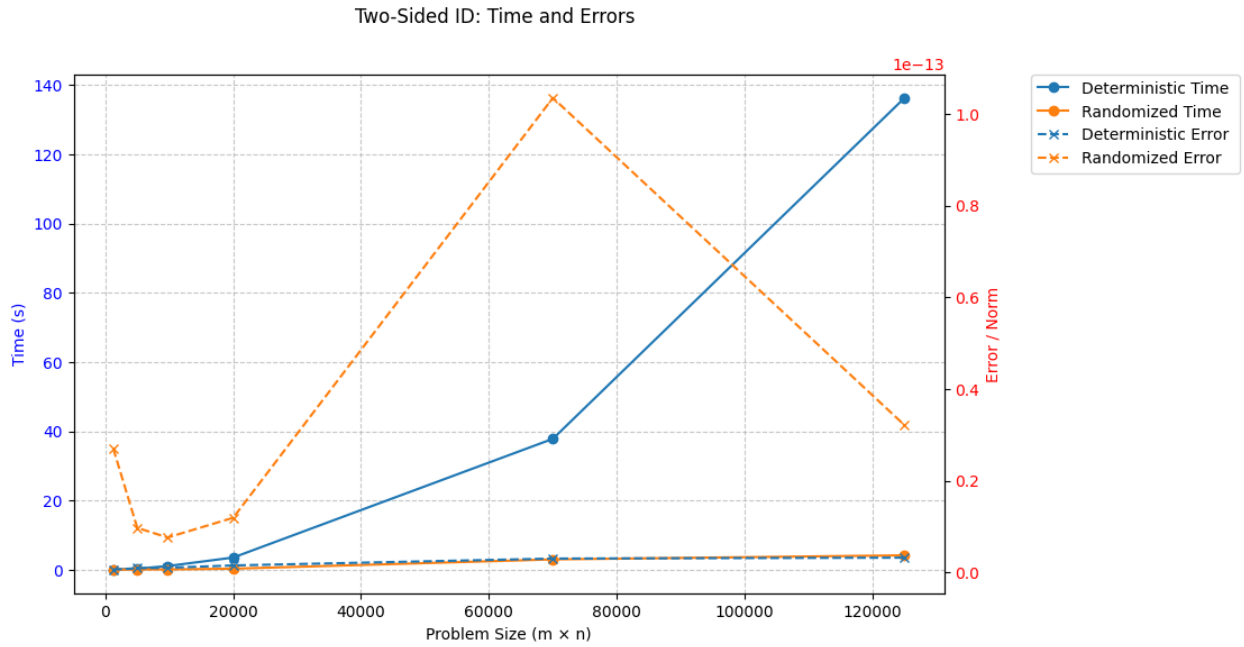


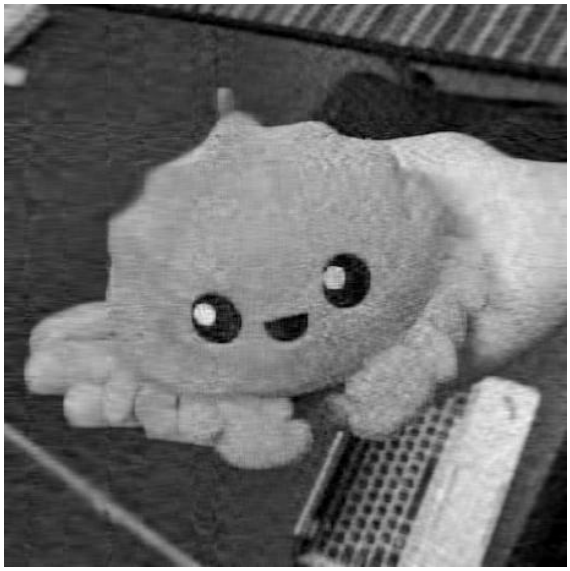
Figure 10: Two-Sided Interpolative Decomposition Benchmark

3.2 Downstream Tasks

We also use our algorithms on downstream tasks and find that the results are comparable to those of a deterministic algorithm, with large speedups as mentioned above.
We used the `nalgebra` deterministic `svd` and our randomized SVD for performing image compression on a test image as shown in Figures 11b and 11c

(a) Original image of Ferris the Rust mascot

(b) Compression with `nalgebra` SVD.



(c) Compression with randomized SVD.

Figure 11: Comparison of image compression methods: The original image (top) compared with compressed results using `nalgebra`'s SVD (left) and randomized SVD (right).

References

[1] Haim Avron, Petar Maymounkov, and Sivan Toledo. Blendenpik: Supercharging lapack's least-squares solver. *SIAM Journal on Scientific Computing*, 32(3):1217–1236, 2010.

[2] Magnus Rudolph Hestenes, Eduard Stiefel, et al. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC, 1952.

[3] Maksim Melnichenko, Oleg Balabanov, Riley Murray, James Demmel, Michael W Mahoney, and Piotr Luszczek. Choleskyqr with randomization and pivoting for tall matrices (cqrrpt). *arXiv preprint arXiv:2311.08316*, 2023.

[4] Xiangrui Meng, Michael A Saunders, and Michael W Mahoney. Lsrn: A parallel iterative solver for strongly over-or underdetermined systems. *SIAM Journal on Scientific Computing*, 36(2):C95–C118, 2014.

[5] Riley Murray, James Demmel, Michael W Mahoney, N Benjamin Erichson, Maksim Melnichenko, Osman Asif Malik, Laura Grigori, Piotr Luszczek, Michał Dereziński, Miles E Lopes, et al. Randomized numerical linear algebra: A perspective on the field with an eye to software. *arXiv preprint arXiv:2302.11474*, 2023.