

Software Project

EE25BTECH11007- Aniket

(1) SUMMARY OF STRANG'S SVD VIDEO

Let $A \in \mathbb{R}^{m \times n}$. The singular value decomposition (SVD) factors any matrix into

$$A = U \Sigma V^T, \quad U \in \mathbb{R}^{m \times m}, \quad V \in \mathbb{R}^{n \times n} \text{ orthonormal}, \quad \Sigma = \text{diag}(\sigma_1 \geq \dots \geq \sigma_r > 0),$$

where $r = \text{rank}(A)$. Geometrically, V rotates (or reflects) coordinates to the directions of action of A , Σ stretches by the nonnegative singular values, and U rotates to the output axes. Strang emphasizes:

- $A^T A$ is symmetric positive semidefinite. Its eigenpairs satisfy $A^T A v_i = \sigma_i^2 v_i$; thus $\sigma_i = \sqrt{\lambda_i(A^T A)}$ and $u_i = \frac{1}{\sigma_i} A v_i$.
- Formula Given

$$A_k = \sum_{i=1}^k \sigma_i u_i v_i^T$$

(2) ONE-SIDED JACOBI SVD: MATH AND PSEUDOCODE

GENERAL IDEA: Orthogonalize the columns of A by applying Givens rotations on the right so that $A^T A$ becomes (nearly) diagonal **without forming it**. For a column pair (p, q) , form their Gram matrix

$$G = \begin{bmatrix} \alpha & \gamma \\ \gamma & \beta \end{bmatrix}, \quad \alpha = \langle a_p, a_p \rangle, \quad \beta = \langle a_q, a_q \rangle, \quad \gamma = \langle a_p, a_q \rangle.$$

Choose a plane rotation

$$R = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}, \quad t = \frac{\beta - \alpha}{2\gamma}, \quad \tau = \text{sign}(t)/(|t| + \sqrt{1+t^2}), \quad c = \frac{1}{\sqrt{1+\tau^2}}, \quad s = c\tau,$$

that diagonalizes $R^T G R$. Apply this **on the right** to the two columns:

$$\begin{bmatrix} a_p & a_q \end{bmatrix} \leftarrow \begin{bmatrix} a_p & a_q \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}, \quad V \leftarrow V \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \text{ on columns } (p, q).$$

After enough sweeps over all pairs (p, q) , the off-diagonal entries of $A^T A$ are tiny, so the columns $\{a_j\}$ are mutually orthogonal. Then

$$\sigma_j = \|a_j\|_2, \quad u_j = \frac{a_j}{\sigma_j}, \quad \text{and the columns of } V \text{ are the } v_j.$$

Constructing U, Σ, V from one-sided Jacobi and ordering

After the final sweep, the method has applied a right orthogonal transform V so that

$$A_{\text{final}} = A V = [a'_1 \ a'_2 \ \dots \ a'_n],$$

whose columns are (numerically) mutually orthogonal.

a) *Column-norms-squared vector and ordering.*: Compute

$$d_j = \|a'_j\|_2^2, \quad \mathbf{d} = [d_1, \dots, d_n].$$

Let π be a permutation that sorts \mathbf{d} in **descending** order: $d_{\pi(1)} \geq d_{\pi(2)} \geq \dots \geq d_{\pi(n)}$, and let Π be the associated permutation matrix. Permute the columns of A_{final} and V simultaneously:

$$\tilde{A} = A_{\text{final}}\Pi, \quad \tilde{V} = V\Pi.$$

b) *Forming Σ , U , and (optionally) truncation.*: Define singular values by

$$\sigma_j = \|\tilde{a}_j\|_2 = \sqrt{d_{\pi(j)}} \quad (j = 1, \dots, n),$$

drop any j with $\sigma_j \approx 0$, and set

$$\Sigma = \text{diag}(\sigma_1, \dots, \sigma_r), \quad U = [\tilde{a}_1/\sigma_1 \ \dots \ \tilde{a}_r/\sigma_r], \quad V = \tilde{V}.$$

Because the columns of \tilde{A} are orthogonal, $U^\top U = I_r$. The SVD is

$$A = U\Sigma V^\top.$$

For the truncated SVD, keep only the first k indices:

$$U_k = [\tilde{a}_1/\sigma_1 \ \dots \ \tilde{a}_k/\sigma_k], \quad \Sigma_k = \text{diag}(\sigma_1, \dots, \sigma_k), \quad V_k = \tilde{V}(:, 1:k).$$

c) *Pseudocode (one-sided Jacobi, truncated to top k):*

- 1) Input: $A \in \mathbb{R}^{m \times n}$, target k , tolerance `tol`, max sweeps S .
- 2) Set $V \leftarrow I_n$.
- 3) For $s = 1, \dots, S$ (a “sweep”):
 - a) For all pairs $1 \leq p < q \leq n$:
 - i) $\alpha \leftarrow \langle a_p, a_p \rangle$, $\beta \leftarrow \langle a_q, a_q \rangle$, $\gamma \leftarrow \langle a_p, a_q \rangle$.
 - ii) If $|\gamma|$ is small relative to $\sqrt{\alpha\beta}$, continue.
 - iii) Compute (c, s) using the formulas above.
 - iv) Update the two columns of A : $(a_p, a_q) \leftarrow (a_p, a_q) \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$.
 - v) Accumulate in V : $(v_p, v_q) \leftarrow (v_p, v_q) \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$.
- 4) If convergence test satisfied, break.
- 5) Compute $\sigma_j = \|a_j\|_2$. Sort indices by decreasing σ_j .
- 6) Output $U_k = [a_{j_1}/\sigma_{j_1} \ \dots \ a_{j_k}/\sigma_{j_k}]$, $\Sigma_k = \text{diag}(\sigma_{j_1}, \dots, \sigma_{j_k})$, $V_k = [v_{j_1} \ \dots \ v_{j_k}]$.

(3) ALGORITHM COMPARISON AND CHOICE

Golub–Reinsch (bidiagonalization + QR). Householder reductions drive A to bidiagonal form in $O(mn^2)$ (for $m \geq n$), then a specialized QR iteration diagonalizes it in $O(n^3)$. This is the standard high-performance approach (LAPACK), very fast and robust, but the implementation from scratch is longer and uses Level-2/3 BLAS-style kernels.

Randomized SVD (RSVD). Projects A to a low-dimensional subspace via random sketching, then computes a small SVD. Excellent for very large, sparse, or data-streaming

settings; accuracy depends on oversampling/power iters. Requires more moving parts (random matrices, QR/LU) and careful parameter choices.

Power/Block Orthogonal Iteration. Iteratively applies A and A^\top with reorthogonalization (e.g., MGS). Simple and effective for leading k , but convergence slows when singular values are clustered; maintaining orthogonality needs care.

One-sided Jacobi (used here).

- **Pros:** conceptually clear (implicitly diagonalizes $A^\top A$), extremely good orthogonality of U, V , embarrassingly parallel over column pairs, and easy to truncate—after convergence, just sort column norms. Implementation is compact and uses only dot products and plane rotations (good for a from-scratch C codebase with no LAPACK/BLAS).
- **Cons:** higher operation count per level of accuracy than Golub–Reinsch; per-sweep cost $O(mn^2)$ with nontrivial constants, so full SVD on large square images is slower.

Why this choice. For this assignment’s goals (clarity, numerical stability, minimal dependencies, and easy top- k reconstruction for images), one-sided Jacobi is a great fit. It delivers highly orthogonal singular vectors and accurate singular values; truncation to different k just selects the largest column norms at the end, and the code stays short and dependency-free.

(4) DISCUSSION OF TRADE-OFFS AND REFLECTIONS ON IMPLEMENTATION CHOICE

What I optimized for

My priorities were: (i) **orthogonality and numerical clarity** (easy to explain and verify), (ii) **few dependencies** (no BLAS/LAPACK), and (iii) **simple truncation to top- k** for image compression. The one-sided Jacobi method aligns well with all three.

Accuracy vs. speed

- **Accuracy/orthogonality.** Jacobi implicitly diagonalizes $A^\top A$ by right-rotations, yielding very small off-diagonals and highly orthogonal U, V in practice. This is excellent for downstream tasks (e.g., stability of $U_k \Sigma_k V_k^\top$ and error analysis).
- **Runtime.** A sweep processes all $\binom{n}{2}$ column pairs and touches m rows \Rightarrow about $O(mn^2)$ work per sweep (for $m \geq n$). Convergence usually needs multiple sweeps. For square images, this is slower than bidiagonalization + QR (Golub–Reinsch), which attains near-optimal constants in tuned libraries.
- **When it shines.** Small-to-medium n (typical lab images), when code simplicity and robustness matter more than peak speed; when exact orthogonality is valued (teaching, diagnostics).

Memory, implementation complexity, and portability

- **Memory.** I store A and accumulate V ; U is formed at the end by normalizing columns of A . Peak memory is $O(mn + n^2)$ (if all of V is kept). For **truncated** output, I can retain only the k most energetic columns after sorting, reducing storage to $k(m + n + 1)$ numbers.

- **Complexity of code.** The core is dot products and 2×2 Givens rotations, so the implementation is compact, branch–light, and dependency–free. This also makes it easy to reason about correctness and add instrumentation (off–diagonal norms, sweep counters).
- **Portability.** Uses only standard C and `math.h`; no platform–specific intrinsics. That meets the assignment’s “barebones” requirement.

Convergence control and tuning

- **Stopping rules.** I use a relative orthogonality test

$$\max_{p < q} \frac{|\langle a_p, a_q \rangle|}{\|a_p\|_2 \|a_q\|_2} \leq \text{tol},$$

or a cap on the number of sweeps. Tight tolerances improve orthogonality but increase runtime.

- **Pair ordering.** I used cyclic (p, q) pairs. Greedy/adaptive schedules (pick the largest $|\langle a_p, a_q \rangle|$) can reduce sweeps but add overhead.
- **Scaling.** Column–wise scaling (or simple pre–normalization) can help when columns have very different norms, but I avoided extra passes to keep code minimal.

Numerical stability

- **Pros.** Jacobi never forms $A^T A$ explicitly, avoiding the squaring of condition numbers that can hurt eigen–based methods. Givens rotations are norm–preserving, which limits error growth and preserves column energies.
- **Cons.** Very clustered singular values can slow convergence (more sweeps). In such cases, Golub–Reinsch or block power methods with robust reorthogonalization may reach the same accuracy faster.

When I would choose a different method

- **Large square/tall problems with strict time budgets:** Prefer Golub–Reinsch (LAPACK) for full SVD, or **randomized SVD** / block power iterations for top- k only.
- **Very large sparse matrices / streaming data:** Randomized methods (with power iterations) are usually superior in wall time and memory.

(4) RECONSTRUCTED IMAGE

EINSTEIN
GLOBE GREYSCALE

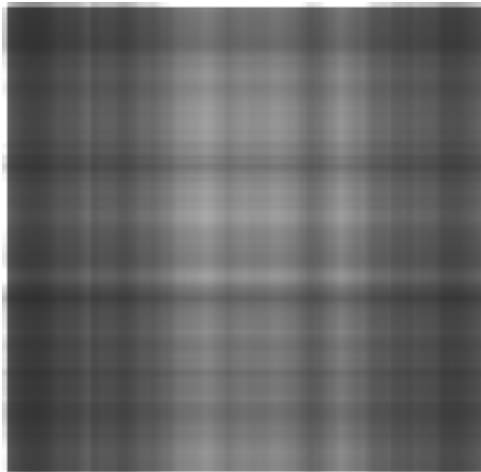


Fig. 1: Caption



Fig. 2: Caption

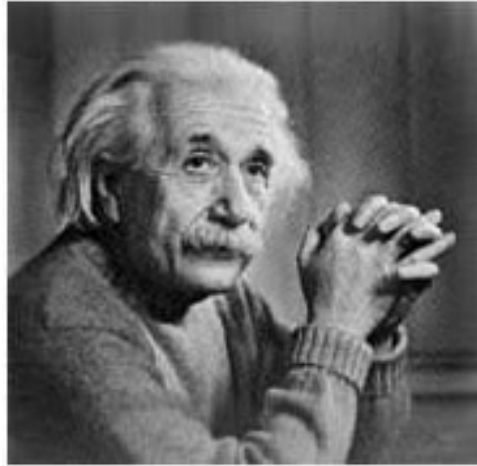


Fig. 3: Caption

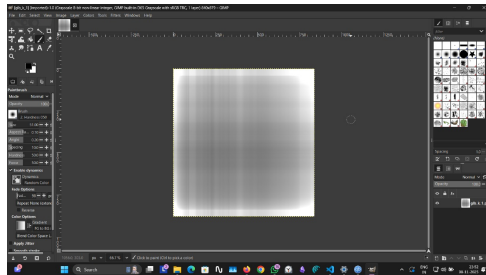


Fig. 4: Caption

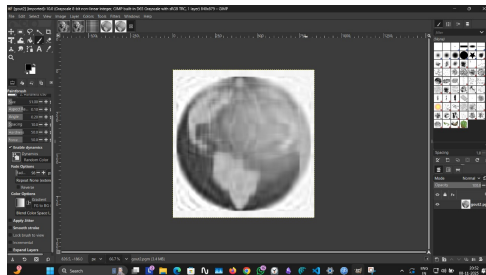


Fig. 5: Caption



Fig. 6: Caption

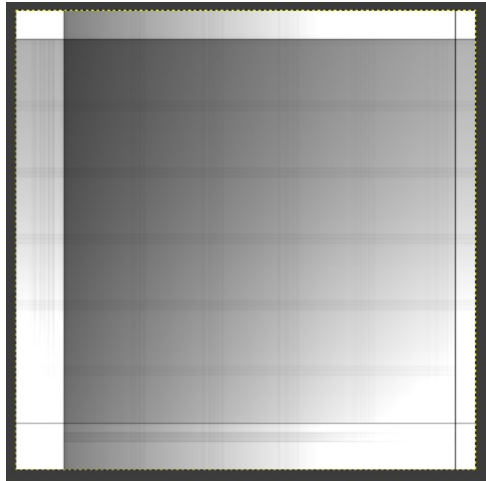


Fig. 7: Caption

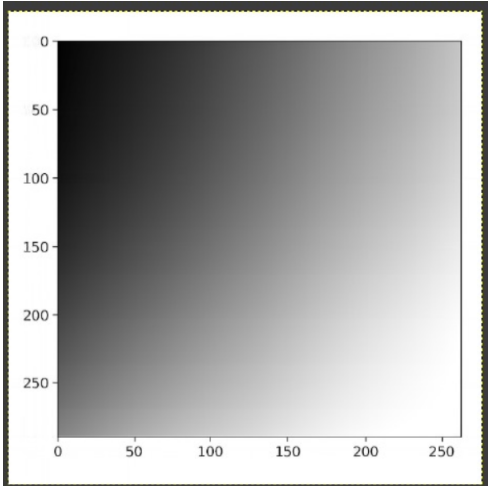


Fig. 8: Caption

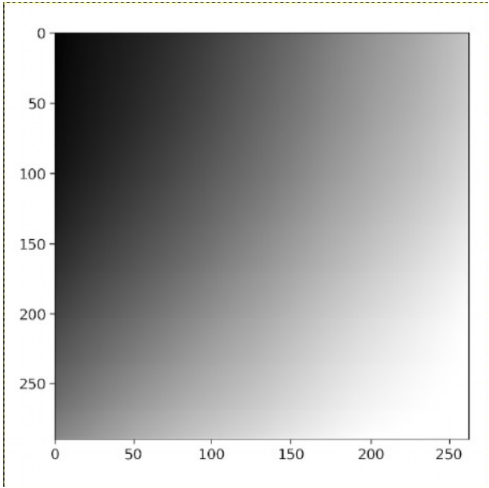


Fig. 9: Caption