

FAULT OBLIVIOUS EIGENVALUE SOLVER*

JAYANTA MUKHERJEE[†], XUEJIAO KANG[‡], DAVID F. GLEICH[§], AHMED SAMEH[¶],
AND ANANTH GRAMA^{||}

Abstract. Eigenvalue problems serve as fundamental substrates for applications in large-scale scientific simulations and machine learning, often requiring computation on massively parallel platforms. As these platforms scale to hundreds of thousands of cores, hardware failures become a significant challenge to reliability and efficiency. In this paper, we propose and analyze a novel fault-tolerant eigenvalue solver based on erasure-coded computations – a technique that enhances resilience by augmenting the system with redundant data **a priori**. This transformation reformulates the original eigenvalue problem as a generalized eigenvalue problem, enabling fault-oblivious computation while preserving numerical stability and convergence properties. We formulate the augmentation scheme, establish the necessary conditions for the encoded blocks, and prove the relationship between the original and transformed problems. We implement an erasure-coded TraceMin eigensolver and demonstrate its effectiveness in extracting eigenvalues in the presence of faults. Our experimental results show that the proposed solver incurs minimal computational overhead, maintains robust convergence, and scales efficiently with the number of faults, making it a practical solution for resilient eigenvalue computations in large-scale systems.

Key words. Fault tolerance, Eigenvalue computation, Eigensolver, TraceMin

MSC codes. 15A18, 65F15, 93B60

1. Introduction. Eigenvalue problems are computationally intensive and arise in various domains, often requiring solutions on scalable parallel and distributed platforms. The high complexity and massive scale of such platforms makes fault tolerance a critical consideration. Traditional parallel computations typically rely on checkpoint-restart mechanisms for fault tolerance. However, these techniques present two major challenges: (i) they require consistent checkpoints, which may incur significant overhead due to rollback – especially in scalable parallel programs that aim to minimize global synchronization; and (ii) they demand substantial I/O capacity and bandwidth to store checkpoints in persistent storage or sufficient interconnect bandwidth for in-memory checkpoints.

An alternative approach to fault tolerance involves detecting and mitigating failures using active replicas in conjunction with a consensus procedure. Active replicas are typically used in real-time systems, where worst-case execution times must be guaranteed, and rollback/ replay schemes may violate such guarantees. However, active replicas have high resource overhead, as each computation must be executed by $s + 1$ replicas to tolerate s faults.

Erasur coding is frequently used in storage systems to provide efficient and scalable fault tolerance by adding appropriately coded redundancies to overcome data erasures. These codes can be conceptualized as multiplying a data vector of size n by a coding matrix consisting of n columns and $m > n$ rows. If any n of the m rows of the coding matrix are guaranteed to be linearly independent, the vector re-

*

Funding: This work was funded by the Department of Energy(DOE DE-SC0023162 Sparstitute) and the NSF (IIS-2007481).

[†]Department of Computer Science, Purdue University, West Lafayette, IN (jmukher@purdue.edu).

[‡]Facebook (xue@meta.com)

[§]Department of Computer Science, Purdue University, West Lafayette, IN (dgleich@purdue.edu)

[¶]Department of Computer Science, Purdue University, West Lafayette, IN (sameh@purdue.edu)

^{||}Department of Computer Science, Purdue University, West Lafayette, IN (ayg@cs.purdue.edu)

sulting from this matrix-vector product contains sufficient redundancies to tolerate up to $m - n$ erasures. Specifically, in the event of up to $m - n$ erasures, the remaining n elements can be used with the corresponding $n \times n$ non-singular matrix corresponding to the non-erased rows of the coding matrix to recover the original n data items. Building on this concept of erasure-coded storage, in prior work, we introduced the notion of erasure-coded computation for solving linear systems. In this work, the input problem is augmented with suitably coded blocks, the augmented problem instance is solved on a faulty parallel platform in a fault-oblivious manner, and the solution is recovered from the results on the non-faulty processors using an inexpensive procedure [10, 11, 20].

In this paper, we present a novel formulation of erasure-coded computations for solving eigenvalue problems. In contrast to solving linear systems, naively augmenting the input matrix with row and column blocks alters the spectrum, with no known inexpensive methods of recovering the original eigenvalues.

We focus here on faults that cause erasure (or deletion) of data, along with fail-stop failure of the processors on which the corresponding part of the computation may execute. We present a novel erasure-coded computation scheme for fault-tolerant solution of eigenvalue problems, $Ax = \lambda x$, for given matrix A . Unlike linear systems, naively adding a coding block to a given matrix A changes its eigenvalues, and there are no known computationally inexpensive ways of recovering the original eigenvalues from these perturbed eigenvalues. To address this, we transform the original eigenvalue problem to an equivalent (in terms of eigenvalues) generalized eigenvalue problem $\tilde{A}\tilde{x} = \lambda\tilde{B}\tilde{x}$, where $\tilde{A}$ is an augmented form of matrix A and $\tilde{B}$ is an encoded identity matrix. The resulting augmented problem can be solved using most off-the-shelf eigensolvers in a fault-oblivious manner; i.e., in the event of a fail-stop failure, the remaining processors simply proceed with their computation oblivious to the faults.

We present detailed proofs establishing the equivalence between the eigenvalues of the original problem and those of a reformulated generalized eigenvalue problem (Theorem-3.3). We analyze the impact of faults on the generalized problem and provide methods to recover eigenvalues in the presence of such faults. Although our primary fault model assumes fail-stop failures - analogous to erasures in storage - other fault types (e.g., transient or soft faults) can be handled similarly using predicates to detect and isolate faulty states.

We solve the reformulated problem using TraceMin [12, 17, 18] and show that our approach achieves a convergence behavior comparable to the fault-free case, even under random erasures (randomly selected row-column removals).

Our experiments quantify the impact of different fault models on convergence, the overhead introduced by our erasure-coded eigensolver, the benefits of optimizations, and sensitivity to various parameters, including approximation levels in adaptive code construction. These results demonstrate the efficacy of our approach in solving eigenvalue problems in fault-prone computing environments.

In summary, our contributions are as follows.

- (i) A novel reformulation of the eigenvalue problem as an equivalent generalized eigenvalue problem with augmented matrices, enabling fault-oblivious computation.
- (ii) Low-overhead erasure-coding schemes for fault-tolerant eigenvalue computations.
- (iii) Application of erasure coding to both the Power Method and TraceMin, demonstrating the generality of our approach.

- (iv) Performance comparison with checkpoint-restart techniques, highlighting the advantages of our erasure-coded solver.
- (v) Analysis of both single- and multifault scenarios, demonstrating robustness across diverse fault conditions.

2. Related Research. Accurate computation of eigenvalues is an essential part of ML applications operating in hardware environments ranging from embedded devices in harsh environments to data-center scale solvers. Fault tolerance techniques in these environments can be classified into two broad categories: system-supported and algorithm-based. System-supported methods include checkpoint-restart [1], active replicas [16], and deterministic replay [3]. Checkpoint-restart techniques periodically save the application state into persistent storage (disks or replicated in-memory). This requires the identification of consistent checkpoints and the capacity for persistent storage in terms of space and bandwidth. Active replicas execute computations on multiple processors – these replicas are monitored for potential faults, and a consensus protocol identifies fault-free executions. Algorithm-based fault tolerance (ABFT) methods modify the base algorithm to embed redundant computations to render the overall computation resilient to faults [2, 4, 5, 6, 7, 9, 15]. While ABFT methods often have advantages over system-supported methods in resource overheads, they must be specifically designed for each algorithm, leverage specific aspects of the algorithm and fault characteristics, and typically require intricate correctness proofs. Our method can be viewed as the first method for fault-tolerant eigenvalue computations in the broad class of ABFT methods. It leverages results from coding theory (sparse codes), linear algebra (augmented problem formulation, correctness proofs), randomized techniques (leverage score sampling), and efficient solvers (conditioning, convergence), to deliver a novel high-performance fault-tolerant eigensolver.

3. Erasure Coded Eigenvalue Solver. Our proposed solution adds redundant rows and columns to the matrix to render it tolerant to faults with any solver. In contrast to linear system solvers, adding a row (or column) to a matrix, even if it is in the row-subspace of the matrix, changes its eigenvalues. For this reason, a key challenge for us is the reformulation of the eigenvalue problem so that the addition of a coding block still allows for inexpensive recovery of the original eigenvalues. In this section, we present a novel reformulation of the eigenvalue problem, along with recovery algorithms.

3.1. Formulating an Erasure Coded Eigensolver. A standard symmetric eigenvalue problem can be written as:

$$(3.1) \quad Ax = \lambda x,$$

where $A \in \mathbb{R}^{n \times n}$ is a symmetric matrix, λ is an eigenvalue, and x is the corresponding eigenvector. A generalized eigenvalue problem is

$$(3.2) \quad Ax = \lambda Bx,$$

which reduces to the original problem when B is the identity matrix (I_n) of size $n \times n$.

Let E be an $n \times k$ matrix we call the *coding matrix*. In our design, the coding matrix E should have Kruskal row rank of k [14] to ensure tolerance of up to k faults that occur anywhere in the system. We derive the *recovery equation* for our erasure coding as:

$$(3.3) \quad x^* = x + Er.$$

Here, x is the eigenvector of the original eigenvalue problem 3.1 of dimension n , and r is the redundant part (related to fault-tolerance) of dimension k . We can substitute the recovery equation 3.3 into original eigenvalue system 3.1 to get

$$(3.4) \quad A(x + Er) = \lambda(x + Er)$$

Recall that, Kruskal row rank k of a matrix implies that any subset of k rows of the matrix is guaranteed to be linearly independent. Now, as we want some redundancy, we add another set of linear constraints.

$$(3.5) \quad E^T[Ax^* = \lambda x^*]$$

Putting 3.4 and 3.5 together we have,

$$(3.6) \quad \begin{cases} A(x + Er) = \lambda(x + Er) \\ E^T A(x + Er) = \lambda E^T(x + Er) \end{cases}$$

After grouping the terms of the above 3.6, we arrive at the augmented eigenvalue problem:

$$(3.7) \quad \begin{bmatrix} A & AE \\ E^T A & E^T AE \end{bmatrix} \begin{bmatrix} x \\ r \end{bmatrix} = \lambda \begin{bmatrix} I & E \\ E^T & E^T E \end{bmatrix} \begin{bmatrix} x \\ r \end{bmatrix}$$

We more compactly write this as the following generalized eigenvalue problem:

$$(3.8) \quad \tilde{A}\tilde{x} = \lambda\tilde{B}\tilde{x},$$

where, $\tilde{A}$, $\tilde{B}$ are the augmented matrices

$$(3.9) \quad \tilde{A} = \begin{bmatrix} A & AE \\ E^T A & E^T AE \end{bmatrix}, \quad \tilde{B} = \begin{bmatrix} I & E \\ E^T & E^T E \end{bmatrix}, \quad \tilde{x} = \begin{bmatrix} x \\ r \end{bmatrix}$$

and $\tilde{x}$ is the eigenvector of the augmented system.

This generalized eigenvalue problem is singular. In this case, this is because there exists a vector $\tilde{x}$ such that $\tilde{A}\tilde{x} = 0$ and $\tilde{B}\tilde{x} = 0$ for any vector $\tilde{x}$ in the joint null-space of $\tilde{A}$ and $\tilde{B}$.

LEMMA 3.1 (Null Space in Augmented Generalized Eigenvalue System). *The matrices $\tilde{A}$ and $\tilde{B}$ from (3.9) have a joint null space $\begin{bmatrix} E \\ -I_k \end{bmatrix}$*

Proof. Recall $\tilde{A}$ and $\tilde{B}$ are $(n+k) \times (n+k)$. Note that $\tilde{B}$ has rank n since the last k rows are a linear combination of the previous n , and the first n rows have an identity block. Therefore, the joint null space must have rank at most k . The matrix $\begin{bmatrix} E \\ -I_k \end{bmatrix}$ has rank k and

$$(3.10) \quad \tilde{A} \begin{bmatrix} E \\ -I_k \end{bmatrix} = 0, \quad \tilde{B} \begin{bmatrix} E \\ -I_k \end{bmatrix} = 0. \quad \square$$

To make the structure inside the singular pencil clear, we next show that the pencil $(\tilde{A}, \tilde{B})$ is strictly equivalent to a simple case related to the eigenvalues of A alone.

THEOREM 3.2 (Matrix Pencil Equivalence). *Let $\tilde{A}, \tilde{B}$ be from (3.7). Then the pencil $\tilde{A} - \lambda \tilde{B}$ is strictly equivalent to the pencil*

$$\begin{bmatrix} 0 & 0 \\ 0 & A(I + EE^T) \end{bmatrix} - \lambda \begin{bmatrix} 0 & 0 \\ 0 & (I + EE^T) \end{bmatrix}.$$

Since the matrix $(I + EE^T)$ is non-singular for any E , a subset of eigenvalues and eigenvectors of the generalized eigenvalues of $(\tilde{A}, \tilde{B})$ will be related to the eigenvalues of A .

Proof. Let $M = \begin{bmatrix} E & I \\ -I_k & E \end{bmatrix}$, M is invertible with the last n columns orthogonal to the first k . Then $M^{-1}(\tilde{A} - \lambda \tilde{B})M$ is a strict equivalence transformation and

$$M^{-1}(\tilde{A} - \lambda \tilde{B})M = \begin{bmatrix} 0 & 0 \\ 0 & A(I + EE^T) \end{bmatrix} - \lambda \begin{bmatrix} 0 & 0 \\ 0 & (I + EE^T) \end{bmatrix}.$$

For a quick understanding of why this form follows, note that $M = \begin{bmatrix} Z & Y \end{bmatrix}$ where Z is the basis for the joint null space of A and B and $Z^T Y = 0$, so these are orthogonal subspaces. A straightforward elementary justification is possible through the closed form calculation of

$$M^{-1} = \begin{bmatrix} E^T(I + EE^T)^{-1} & -(I + E^T E)^{-1} \\ (I + EE^T)^{-1} & E(I + E^T E)^{-1} \end{bmatrix}.$$

The two trickier steps are showing $E^T(I + EE^T)^{-1} + (I + E^T E)^{-1}E^T = 0$ (which shows up in the 2, 1 block) and $(I + EE^T)^{-1} + E(I + E^T E)^{-1}E^T = I$ (which shows up in the 2, 2 block). It suffices to look at the computation for $\tilde{A}$ alone as the structure of $\tilde{A}$ and $\tilde{B}$ are the same with A replaced by an identity block.

For the trailing size n block, we then have the generalized problem $A(I + EE^T)y = \lambda(I + EE^T)y$, which is a similarity transform of the eigenvalues of A through $(I + EE^T)^{-1}A(I + EE^T)$. $\square$

Since the singular subspace of the generalized eigenvalue problem can have *any* eigenvalues when treated computationally the augmented eigenvalue system (as shown in 3.8) will have some spurious eigenvalues that correspond to the vectors in the null-space of $\begin{bmatrix} I & E \end{bmatrix}$. These can be easily detected.

Example. We illustrate this encoding and detection of spurious eigenvalues using a simple example. Let:

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \quad E = \begin{bmatrix} 0.98 & 0.42 \\ 0.13 & 0.39 \\ 0.53 & 0.85 \\ 0.87 & 0.93 \end{bmatrix}$$

The eigenvalues of A are (to four digits)

$$0.382 \quad 1.382 \quad 2.618 \quad 3.618$$

One computation of the generalized eigenvalues of eigenvectors of $\tilde{A}$ and $\tilde{B}$ yielded:

$$0.382 \quad 0.6012 \quad 1.382 \quad 2.618 \quad 3.6031 \quad 3.618$$

$$\begin{bmatrix} -0.1696 & 0.1728 & -0.2473 & -0.258 & -0.4414 & 0.4409 \\ -0.5019 & 0.3572 & 0.2104 & 0.1836 & 0.2757 & -0.2745 \\ -0.8198 & 0.7163 & 0.7065 & 0.3035 & 0.3842 & -0.3852 \\ -0.7187 & 0.7106 & 0.6918 & 0.3455 & 0.1653 & -0.1643 \\ -0.3552 & 0.2522 & 0.4895 & 0.8502 & 0.879 & -0.8793 \\ 1.0 & -1.0 & -1.0 & -1.0 & -1.0 & 1.0 \end{bmatrix}$$

(each column of the matrix is an eigenvector with the associated eigenvalue above it). The two spurious eigenvalues can be detected by computing $[I \ E] r$ for each eigenvector and checking for zero (or a very small norm, more generally). For example,

$$\begin{bmatrix} 0.1728 \\ 0.3572 \\ 0.7163 \\ 0.7106 \end{bmatrix} + E \begin{bmatrix} 0.2522 \\ -1 \end{bmatrix} = 0$$

We summarize as follows. Given any generalized eigenvector $\tilde{x} = \begin{bmatrix} x \\ r \end{bmatrix}$ of $(\tilde{A}, \tilde{B})$, then compute $x + Er$ and if this is zero, then it is a spurious solution.

In principle, the singularity gives us the fault tolerance we seek. The eigenvalue problem remains valid with up to k elements of x set to arbitrary values. However, the singular pencil poses a variety of computational issues that are difficult to resolve. Consequently, we use this idea for inspiration and consider ideas more closely related to those used in [11]. This involves replacing the erased row-column pairs with the corresponding coding blocks from the augmented matrices $\tilde{A}$ and $\tilde{B}$ when a fault occurs.

3.2. A More Practical Fault Handling and Solution Recovery Idea. The idea for our more practical solver is that we treat the blocks $R = AE$, $S = E^T AE$, $T = E^T E$ as *redundancy* that can be used whenever it is needed by a fault. (Recall that we target symmetric A so $(AE)^T = E^T A$.) Then, when fault occur, we substitute these into the system for the missing parts. Then we initiate an eigensolver on A . In the event of a fault, a subset of rows or columns in the matrices A and B are “erased” in that we lose access to them. Note that we consider only fail-stop failures; that is, in the event of a failure, a compute node halts both computation and communication.

After erasures, we reconstitute the matrices A' and B' by substituting the erased rows and columns with the corresponding rows and columns from the blocks of E, R, S, T , as shown in Figure 1, and proceed with the eigenvalue solver. For example, if rows i_1 to i_2 of matrix A are deleted, we replace these rows and their corresponding columns in the erased system (represented as A_r) with the pre-computed erasure-coded blocks from $E^T A$ and AE , respectively, to preserve symmetry. Similarly, the erased rows and columns of matrix B , are replaced with the corresponding rows and columns of $E^T B$ and BE . The intersecting square-block from (i_1, i_1) to (i_2, i_2) of B is replaced by the square block of size $(i_2 - i_1) \times (i_2 - i_1)$ of $E^T BE$ of the augmented matrix $\tilde{B}$ as illustrated in Figure 1. Once the matrices have been reconstituted, the solver resumes execution.

Example. Continuing our previous example, we have A and E as before. We use the recovery information

$$R^T = \begin{bmatrix} 1.83 & -1.25 & 0.06 & 1.21 \\ 0.45 & -0.49 & 0.38 & 1.01 \end{bmatrix} \quad S = \begin{bmatrix} 2.7154 & 1.4574 \\ 1.4574 & 1.2602 \end{bmatrix} \quad T = \begin{bmatrix} 2.0151 & 1.7219 \\ 1.7219 & 1.9159 \end{bmatrix}.$$

Suppose as we were solving an eigenvalue problem $Ax = \lambda x$, we had a failure in the 3rd row. Then we’d assemble the new matrix by replacing the 3rd row and column

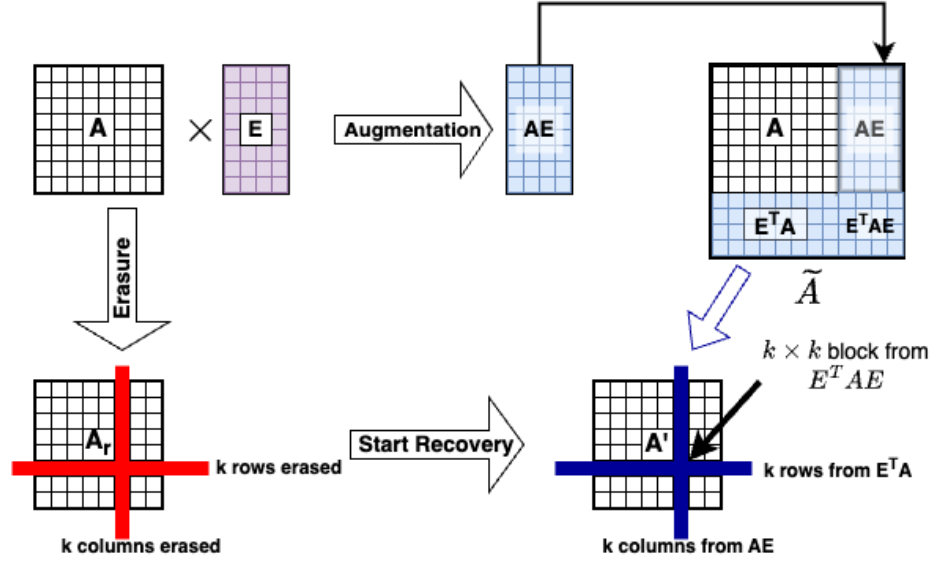


Fig. 1: Erasure-Coded TraceMin Solver Flow.

with information from the 1st row of R^T . (There is nothing special about this row, we could have used the 2nd as well, but our convention is to use the first unused row.) This yields the new *generalized* eigenvalue problem with $A'x = \lambda B'x$

$$A' = \begin{bmatrix} 2.0 & -1.0 & 1.83 & 0.0 \\ -1.0 & 2.0 & -1.25 & 0.0 \\ 1.83 & -1.25 & 2.7154 & 1.21 \\ 0.0 & 0.0 & 1.21 & 2.0 \end{bmatrix} \quad B' = \begin{bmatrix} 1.0 & 0.0 & 0.98 & 0.0 \\ 0.0 & 1.0 & 0.13 & 0.0 \\ 0.98 & 0.13 & 2.0151 & 0.87 \\ 0.0 & 0.0 & 0.87 & 1.0 \end{bmatrix}$$

The generalized eigenvalues and vectors of this new system are

$$\begin{matrix} 0.382 & 1.382 & 2.618 & 3.618 \\ \begin{bmatrix} 0.7405 & 1.2889 & 1.2889 & -0.7405 \\ -0.454 & 0.4629 & -0.2806 & -0.749 \\ -1.1349 & -0.7014 & -0.7014 & 1.1349 \\ 0.6156 & 0.0087 & 1.2117 & -1.3591 \end{bmatrix} \end{matrix}.$$

To recover the eigenvectors of the original system, we need to use the “recovery equation $x + Er$ ”. In this case – when applied to each eigenvector as a matrix equation – this becomes

$$\begin{bmatrix} 0.7405 & 1.2889 & 1.2889 & -0.7405 \\ -0.454 & 0.4629 & -0.2806 & -0.749 \\ 0 & 0 & 0 & 0 \\ 0.6156 & 0.0087 & 1.2117 & -1.3591 \end{bmatrix} + \begin{bmatrix} 0.98 \\ 0.13 \\ 0.53 \\ 0.87 \end{bmatrix} \begin{bmatrix} -1.1349 & -0.7014 & -0.7014 & 1.1349 \end{bmatrix} \\ = \begin{bmatrix} -0.3717 & 0.6015 & 0.6015 & 0.3717 \\ -0.6015 & 0.3717 & -0.3717 & -0.6015 \\ -0.6015 & -0.3717 & -0.3717 & 0.6015 \\ -0.3717 & -0.6015 & 0.6015 & -0.3717 \end{bmatrix}$$

Thus, we *recover* the original eigenvectors from the generalized eigensystem.

To describe and analyze this process formally, we will permute the system so that C is the part of A that is non-faulty and c are the corresponding coefficients of x that were not erased. The faulty components will be placed at the *end* of the matrix for analysis as in

$$(3.11) \quad Ax = \lambda x \quad \implies \quad \begin{bmatrix} C & F_1 \\ F_1^T & F_2^T \end{bmatrix} \begin{bmatrix} c \\ f \end{bmatrix} = \lambda \begin{bmatrix} c \\ f \end{bmatrix}.$$

The matrices F_1^T and F_2^T correspond to the erased rows, as well as the components f . Suppose there are $\ell \leq k$ failed or erased rows. The matrix C just consists of the rows and columns of A that are still correctly executing. We are going to replace the rows and columns corresponding to F_1 with F_2 with the redundant data. Let

$$(3.12) \quad R^T = E^T A = \begin{bmatrix} Z^T \\ Y^T \end{bmatrix} = \begin{bmatrix} Z_C^T & Z_F^T \\ Y_C^T & Y_F^T \end{bmatrix} \quad \text{and} \quad E^T = \begin{bmatrix} E_Z^T \\ E_Y^T \end{bmatrix} = \begin{bmatrix} E_{ZC}^T & E_{ZF}^T \\ E_{YC}^T & E_{YF}^T \end{bmatrix}$$

be a partition of the redundancy data into the first ℓ rows of R^T and the later rows of R , along with a corresponding partition of the first ℓ rows of the matrix E^T ; the column partition is given by the correct and faulty rows and columns of the matrix A . Also, let S_Z and T_Z be the first ℓ rows and columns of S and T , respectively. Then the reconstituted eigensystem is

$$(3.13) \quad \begin{bmatrix} C & Z_C \\ Z_C^T & S_Z \end{bmatrix} \begin{bmatrix} c \\ r \end{bmatrix} = \lambda \begin{bmatrix} I & E_{CZ} \\ E_{CZ}^T & T_Z \end{bmatrix} \begin{bmatrix} c \\ r \end{bmatrix}.$$

Here r are components of the solution that are not part of the original solution but need to be included into the solve. (The values in r correspond to the 3rd row of our eigenvector matrix above.)

One may question the need for a coding block, as opposed to replacing erased parts of matrix A with corresponding blocks of A itself. The problem with this approach is that the matrix A will have to be replicated $(k+1)$ -fold to tolerate k faults. This is infeasible for problems that operate close to limit of memory.

We now show that this process leaves the eigenvalues unchanged and permits us to recover the eigenvectors. We also simplify notation and assume we use *all* of the redundant data. We can do this without loss of generality because Z_C, S_Z , and T_Z *only* use the information corresponding to the first ℓ columns of E .

THEOREM 3.3 (Eigenvalue Equivalence). *Consider a eigenvalue problem $Ax = \lambda x$ where A is symmetric. Let E be a coding matrix with Kruskal row rank k . Let $R = AE$, $S = E^T AE$, and $T = E^T E$. Suppose there are k faults, which result in loss of k -rows and columns of A . Without loss of generality, we have the problem permuted so these are the last k rows and columns as in 3.11. Let*

$$E = \begin{bmatrix} E_C \\ E_F \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} R_C \\ R_F \end{bmatrix}$$

correspond to a partition of the data in E and Z into the same set of correct and faulty rows. Then the generalized eigenvalues of $A'y = \lambda' B'y$, that is

$$(3.14) \quad \begin{bmatrix} C & R_C \\ R_C^T & S \end{bmatrix} \begin{bmatrix} c \\ r \end{bmatrix} = \lambda' \begin{bmatrix} I & E_C \\ E_C^T & T \end{bmatrix} \begin{bmatrix} c \\ r \end{bmatrix},$$

are identical to those of A . Also, the eigenvectors are related by a simple linear transformation.

Proof. We will show that if $y = \begin{bmatrix} c \\ r \end{bmatrix}$ is a generalized eigenvector of 3.14 with eigenvalue λ' then

$$(3.15) \quad v = \begin{bmatrix} v_c \\ v_f \end{bmatrix} = \begin{bmatrix} c + E_C r \\ E_F r \end{bmatrix} = \underbrace{\begin{bmatrix} I & E_C \\ 0 & E_F \end{bmatrix}}_{=M} \begin{bmatrix} c \\ r \end{bmatrix}$$

is an eigenvector of A with the same eigenvalue. Because E has Kruskal rank k , the matrix E_F is $k \times k$ and invertible, so the transformation matrix M is invertible, which is the crux of the argument. Let $A'y = \lambda' B'y$ be the generalized eigenvalue problem (3.14). Since $My = v$ with $A' = M^T A M$ and $B' = M^T B M$ then we arrive at $Av = \lambda v$. $\square$

3.3. Implementing An Erasure-Coded Power Method. For the erasure-coded variant of the Generalized Power Method, we compute the erasure coding matrix E . We then derive the augmented coding blocks R, S, T from matrix A and E . We now show two different implementations of a power method. The first is one that exhibits the ideas clearly. The second is one that illustrates how a practical implementation might function. To be more concrete, in the first, we use matrices A and B to represent the operations that would be performed with modification to the matrices. In the second, we'll illustrate how these operations can be optimized using the structure of the problem.

The preceding theory dealt with the *eigensystems* that arise. When we move to these algorithms, we need to consider how to reinitialize the values of X_F after a fault occurs. We assume that the values X_F are totally lost. First, if an implementation periodically computes and retains the quantity $E^T X$, then we can always extract the faulty missing elements from this matrix – this is a simple calculation related to the ideas from [19]. However, since this has the flavor of checkpointing, we do not do it. Although it may seem like we should be able to utilize the non-faulty values X_C to derive improved values for X_F , we were unable to make this idea successful. Consequently, when a fault occurs, we replace the values X_F with random data. This appears to work the best in our experiments.

The power method for solving the generalized eigenvalue problem for matrices A and B involves multiplying both on the left and right by an orthogonal matrix Q , as outlined in Algorithm 3.1. At each iteration, a QR factorization is performed to compute the orthonormal matrix Q . Similar to the standard power method, the sequence of vectors X is updated by multiplying the orthogonal matrix Q with the approximate eigenvectors U obtained from solving the projected eigenproblem defined by the matrices $A_Q = Q^T A Q$ and $B_Q = Q^T B Q$. Convergence is assessed by computing the relative residual r_{rel} and comparing it against a predefined tolerance ε for the eigenvector corresponding to the dominant eigenvalue of the matrix pencil (A, B) .

In the generalized power method, the key step is an iteration is computing $B^{-1}AX$. The structure of our matrix B makes this easy in the event of any fault. Recall that

$$B = \begin{bmatrix} I & E_C \\ E_C^T & E^T E \end{bmatrix}.$$

This means it is – at most – a rank k update to the identity matrix B . Consequently, the matrix has a simple closed form inverse that can be used to implement an efficient solve

$$B^{-1} = \begin{bmatrix} I + E_C(E^T E - E_C^T E_C)^{-1} E_C^T & -E(E^T E - E_C^T E_C)^{-1} \\ -(E^T E - E_C^T E_C)^{-1} E^T & (E^T E - E_C^T E_C)^{-1} \end{bmatrix}.$$

Algorithm 3.1 Erasure-Coded Generalized Block Power Method with QR Subspace Iteration

Require: Symmetric matrix $A \in \mathbb{R}^{n \times n}$, initial matrix $X_0 \in \mathbb{R}^{n \times k}$, tolerance ε , redundancy information E, R, S, T

Ensure: Approximate eigenpairs (Λ, X) where $AX \approx \Lambda BX$

```

1: Initialize  $X \leftarrow X_0$ 
2: Initialize  $B \leftarrow I$ 
3: for  $i = 1, 2, \dots$  until convergence do
4:   if Fault Occurs then
5:     Replace the faulty rows and columns of  $A$  with any unused rows and columns of  $R$  and the intersection of rows and columns with block  $S$  to reconstitute  $A'$ 
6:     Replace the faulty rows and columns of  $B$  with any unused rows and columns of  $E$  and the intersection of rows and columns with block  $T$  to reconstitute  $B'$ .
7:     Use the reconstituted  $A'$  and  $B'$  as  $A$  and  $B$  respectively and continue the solver;  $A \leftarrow A'$ ;  $B \leftarrow B'$ 
8:     Set any failed rows of  $X$  to random data and orthogonalize with a QR factorization.
9:   end if
10:   $Y \leftarrow AX$ 
11:   $Z \leftarrow B^{-1}Y$  ▷ Solve  $BZ = AX$ 
12:   $[Q, \sim] \leftarrow \text{qr}(Z)$  ▷ Thin QR factorization
13:   $A_Q \leftarrow Q^T A Q$ ,  $B_Q \leftarrow Q^T B Q$ 
14:   $[U, D] \leftarrow \text{eig}(A_Q, B_Q)$  ▷ Solve small generalized eigenproblem
15:  Sort eigenvalues  $\Lambda$  in  $D$  in descending order; reorder  $U$  accordingly
16:   $X_{\text{new}} \leftarrow Q U$ 
17:  Compute residual  $r \leftarrow \|AX_{\text{new}} - BX_{\text{new}}\Lambda\|_F$ 
18:  Compute relative residual  $r_{\text{rel}} \leftarrow \frac{r}{\|A\|_F}$ 
19:  if  $r_{\text{rel}} < \varepsilon$  then
20:    break
21:  end if
22:   $X \leftarrow X_{\text{new}}$ 
23: end for return  $X, \Lambda$ 

```

Since $E = \begin{bmatrix} E_C \\ E_F \end{bmatrix}$, we have $E^T E - E_C^T E_C = E_F^T E_F$. This matrix is always invertible because E has Kruskal row rank k . Also, the inverse here only occurs for a small $k \times k$ block, $(E^T E - E_C^T E_C)^{-1}$, which is repeated through all the factors and could be treated practically with either a small inverse or a small LU factorization.

The result is a set of operations that implement these operations without explicitly reforming the matrix A . The auxiliary procedures **Aop**, **Bop**, and **invBop**, which are invoked in Algorithm 3.2, are specified in Algorithm 3.3. The key step is compute a Cholesky factor with all the solves.

3.4. Implemented An Erasure-Coded TraceMin. Our erasure-coded eigensolver builds upon the TraceMin algorithm described in [17], and is detailed in Algorithm 3.4, which outlines the procedure for handling faults.

The eigenvectors X of the original eigenvalue problem are recovered from the eigenvectors X' of the reconstituted system, computed as X in Algorithm 3.4.

Algorithm 3.2 Erasure-Coded Generalized Block Power Method with QR Subspace Iteration

Require: Symmetric matrix $A \in \mathbb{R}^{n \times n}$, initial orthogonal matrix $X_0 \in \mathbb{R}^{n \times k}$, tolerance ε , redundancy information $E^{(0)}, Z^{(0)}, S^{(0)}, T^{(0)}$ up to k faults.

Ensure: Approximate eigenpairs (Λ, X) where $AX \approx \Lambda BX$

```

1: Initialize  $X \leftarrow X_0$ ,  $\mathcal{F} = \emptyset$  (the set of faulty indices),
2:  $E \leftarrow \text{getErasureCodedMatrix}(n, k)$  ▷ generate encoding matrix
3:  $R \leftarrow A \cdot E$ ,  $S \leftarrow E^\top A \cdot E$ ,  $T \leftarrow E^\top E$  ▷ generate coding blocks
4: for  $i = 1, 2, \dots$  until convergence do
5:   if a new fault occurs then
6:     Let  $\mathcal{F}_{\text{new}}$  be the set of new faulty indices
7:     Add the faulty indices to  $\mathcal{F}$  (unless  $\mathcal{F}$  exceeds  $k$  indices, in which case
        terminate with an error that the fault capacity was exceeded).
8:     Set  $E_C, Z_C$  to the first  $|\mathcal{F}|$  columns of  $E^{(0)}, Z^{(0)}$ 
9:     Set  $S, T$  to be the first  $|\mathcal{F}| \times |\mathcal{F}|$  block of  $S^{(0)}, T^{(0)}$ 
10:    Compute a factorization of  $T - E_C^\top E_C$  for applications of  $(T - E_C^\top E_C)^{-1}$ 
11:    Set  $X(\mathcal{F}_{\text{new}}, :) =$  random normal entries
12:    Set  $A(\mathcal{F}_{\text{new}}, :) = A(:, \mathcal{F}_{\text{new}}) = 0$  (or implicitly via the fault)
13:  end if
14:  if  $\mathcal{F}_{\text{new}} \neq \emptyset$  then
15:    Update fault set  $\mathcal{F} \leftarrow \text{sort}(\mathcal{F} \cup \mathcal{F}_{\text{new}})$ ;  $EF \leftarrow E[\mathcal{F}, :]$ ,  $G \leftarrow \text{chol}(EF^\top EF)$ 
16:  end if
17:   $Y \leftarrow \text{Aop}(A, R, S, X, \mathcal{F})$ 
18:   $Z \leftarrow \text{invBop}(G, E, EF, Y, \mathcal{F})$  ▷ Solve  $BZ = AX$ 
19:   $[Q, \sim] \leftarrow \text{qr}(Z)$  ▷ Thin QR factorization
20:   $AQ \leftarrow Q^\top \cdot \text{Aop}(A, R, S, Q, \mathcal{F})$     $BQ \leftarrow Q^\top \cdot \text{Bop}(E, T, Q, \mathcal{F})$ 
21:   $[U, D] \leftarrow \text{eig}(AQ, BQ)$  ▷ Solve small generalized eigenproblem
22:  Sort eigenvalues  $\Lambda$  in  $D$  in descending order; reorder  $U$  accordingly
23:   $X_{\text{new}} \leftarrow QU$ 
24:  Compute residual  $r \leftarrow \|AX_{\text{new}} - BX_{\text{new}}\Lambda\|_F$ 
25:  Compute relative residual  $r_{\text{rel}} \leftarrow \frac{r}{\|A\|_F}$ 
26:  if  $r_{\text{rel}} < \varepsilon$  then
27:    break
28:  end if
29:   $X \leftarrow X_{\text{new}}$ 
30: end for return  $X, \Lambda$ 

```

The TraceMin algorithm computes a few of the smallest eigenvalues by reducing the generalized Rayleigh quotient $(\frac{X^T AX}{X^T BX})$ step by step. In [18], a simultaneous iteration method was introduced to address this problem. In each iteration, the previous approximation X_k , which satisfies $X_k^T B X_k = I_s$ and $X_k^T A X_k = \Theta_k$, is updated with a correction term Δ_k , calculated as follows:

$$\begin{aligned}
 (3.16) \quad & \min \quad \text{tr}(X_k - \Delta_k)^T A (X_k - \Delta_k), \\
 & \text{subject to} \quad X_k^T B \Delta_k = 0
 \end{aligned}$$

for any B-orthonormal basis X_{k+1} of the subspace spanned by Z_{k+1} , we construct our erasure-coded scheme on top of the TraceMin algorithm, as described in Algorithm 2 of [17]. In Step 1 of Algorithm 3.4, we augment the matrices A and B with erasure-

Algorithm 3.3 Erasure-Coded Operator Application and Inversion

Require: Mode $\in \{\text{Aop}, \text{Bop}, \text{invBop}\}$; matrices A, E, Z, S, T, EF ; input X ; faulty indices F ; Cholesky factor $G = \text{chol}(EF^\top EF, \text{upper})$

Ensure: Output Y

```

1: function AOP( $A, Z, S, X, \mathcal{F}$ )
2:    $Y \leftarrow AX$ 
3:    $Y \leftarrow Y + Z X(\mathcal{F}, :)$ 
4:    $Y(\mathcal{F}, :) \leftarrow Y(\mathcal{F}, :) + Z^\top X$ 
5:    $Y(\mathcal{F}, :) \leftarrow Y(\mathcal{F}, :) + (S - Z(\mathcal{F}, :) - Z(\mathcal{F}, :)^{\top}) X(\mathcal{F}, :) \triangleright$  New post-failure values
6:   return  $Y$ 
7: end function
8: function BOP( $E, T, X, F$ )
9:    $Y \leftarrow X$ ;  $Y(F, :) \leftarrow 0$ 
10:   $Y \leftarrow Y + E X(F, :)$ 
11:   $Y(F, :) \leftarrow Y(F, :) + E^\top X$ 
12:   $Y(F, :) \leftarrow Y(F, :) + (T - E(F, :) - E(F, :)^{\top}) X(F, :)$ 
13:  return  $Y$ 
14: end function
15: function INV BOP( $G, E, EF, X, F$ )
16:   $Y \leftarrow X$ ;  $Y(F, :) \leftarrow 0$ 
17:   $W \leftarrow E^\top X$ ;  $V \leftarrow X(F, :) + EF^\top X(F, :)$ 
18:   $Y \leftarrow Y + E (G^{-1}W) - E (G^{-1}V)$ 
19:   $Y(F, :) \leftarrow Y(F, :) + (I_F + EF) (G^{-1}(V - W))$ 
20:  return  $Y$ 
21: end function

```

coded rows and columns. In the event of a fault, the system is reconstituted from the erasure coding blocks, as outlined in Step 5 of the Algorithm 3.4. The remainder of the implementation follows Algorithm 2 in [17].

3.5. Construction of Erasure Coding Matrix. Although in theory a Kruskal rank of k is required to guarantee recovery from every possible fault pattern, a weaker but more practical requirement significantly reduces computational complexity. This approach, referred to as *recovery at random* in [11], assumes that faults occur randomly and can be recovered with exceptionally high probability. Furthermore, this recovery method leverages a structured sparse matrix, introducing minimal computational overhead while maintaining high efficiency.

3.6. Coding Matrix Structure. The coding matrix must be sparse to optimize both computation and storage. To achieve this, we generate a random matrix of dimensions $n \times p$ and arrange it in a staggered pattern to preserve the sparsity of the erasure coding block. Specifically, we ensure that each row of size k contains p nonzero entries, as proposed in [11] (Figure 2).

More specifically, the coding matrix E from [11] is an $n \times k$ matrix constructed using a staggered nonzero pattern with p randomly chosen entries for each occurrence of this pattern. [11] showed that selecting p to be larger than $\frac{\log k}{\log \log k}$ ensures that the probability of a random set of k rows of matrix E drawn from the $n \times k$ p-staggered distribution being linearly dependent is less than $(\frac{e}{p+1})^{p+1}$. Furthermore, they demonstrated that, with high probability, the maximum number of rows (out

Algorithm 3.4 Erasure-coded TraceMin

Input: A symmetric matrix, subspace dimension $s > 0$, number of erasures tolerated $k > s$

Output: $\Theta(\text{eigenvalues}), X(\text{eigenvectors})$

- 1: Generate the sparse coding matrix E which can tolerate up to k row/ column failures using coding matrix.
 - 2: Compute augmented blocks (Z, R) and (Q, S) for matrix A and B respectively.
 - 3: Choose a block size $s_2 = 2 \times s$ and an $n \times s_2$ random matrix V_1 of full rank such that $V_1^T B V_1 = I$.
 - 4: **for** $i = 1, 2, \dots$ until convergence **do**
 - 5: **if** Fault Occurs **then**
 - 6: **if** Fault Occurs at a Node (say F) **then**
 - 7: Replace the faulty rows and columns of A with the Z and Z' blocks and the intersection of rows and columns with block R .
 - 8: Replace the same rows and columns of B with the Q and Q' blocks and the intersection of rows and columns with block S .
 - 9: Replacing the faulty rows and columns with coded-blocks to reconstitute A' and B' ; $A = A'$; $B = B'$
 - 10: **end if**
 - 11: **end if**
 - 12: Compute $W_i = A V_i$ and the interaction matrix $H_i = V_i^T W_i$.
 - 13: Compute the eigenpairs of $(Y_i; \Theta_i)$ of H_i . Sort eigenvalues in ascending order and rearrange the corresponding eigenvectors.
 - 14: Compute the Ritz Vectors $X_i = V_i Y_i$.
 - 15: Compute the residuals $R_i = A X_i - B X_i \Theta_i$.
 - 16: Test for Convergence.
 - 17: Solve the following linear system approximately via the CG.
 - 18: $A Z_{i+1} = B X_i$
 - 19: B-orthonormalize Z_{i+1} into V_{i+1} .
 - 20: **end for**
-

of k randomly chosen rows) in matrix E that share the same nonzero structure is approximately $\frac{\ln k}{\ln \ln k} (1 + o(1))$.

We illustrate this $n \times k$ coding matrix, using the staggered non-zero pattern with p non-zeros entries, in Figure 2. We adapt this coding matrix to improve its design for eigenvalues and eigenvectors, which are more sensitive than linear systems. The erasure of critical row-column pairs can significantly affect the stability and convergence of the eigen-solver. In the case of adversarial deletions, the rows that contribute most to the eigenvalues may be erased. There-

fore, assigning more weight to the rows based on their importance is crucial to preserve the integrity of the data. We used leverage scores to assign appropriate importance to specific rows. In addition to random coding schemes, we also evaluated Leverage

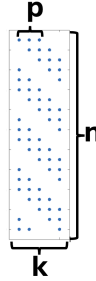


Fig. 2: Coding Matrix

Score-weighted Coding, as described below.

4. Experimental Results. We present detailed experimental validation of our erasure-coded fault-tolerant TraceMin eigensolver on both dense and sparse benchmark datasets described in Table 1. The goal of these experiments is to demonstrate the the stability and robustness of our approach. All methods and plots are implemented in MATLAB on an Apple M1 Pro with 8 cores with 16 GB RAM.

Type	Dataset	Dims.	Nonzeros	Description
Dense	MNIST - Train	$15K \times 15K$	$225M$	Training Dataset
	MNIST - Test	$10K \times 10K$	$100M$	Test Dataset
	CIFAR-10 - Train	$15K \times 15K$	$225M$	Training Dataset
	CIFAR-10 - Test	$10K \times 10K$	$225M$	Test Dataset
Sparse	bcstk17	$11K \times 11K$	$429K$	Stiffness Matrix - Elevated Pressure Vessel
	bcstk25	$15K \times 15K$	$252K$	Stiffness Matrix - 76 Story Skyscraper
	gyro	$17K \times 17K$	$1M$	Model Reduction Problem
	msc23052	$23K \times 23K$	$1.15M$	MSC/NASTRAN Shock Problem
	cbuckle	$14K \times 14K$	$677K$	Cylindrical Shell Stiffness Matrix
	Pres_Poisson	$15K \times 15K$	$716K$	Computational Fluid Dynamics Problem
	jnlbrng1	$40K \times 40K$	$199K$	Quadratic Journal Bearing Problem
	torsion1	$40K \times 40K$	$198K$	Optimization Problem

Table 1: Benchmark Datasets

4.1. Power Method vs TraceMin. We compare the performance results of the Power Iteration Method and TraceMin across various sparse and dense datasets, as summarized in Table 1. Two sets of experiments are conducted to analyze the performance impact: (1) as the problem size increases and (2) as the subspace size expands.

As the problem size increases, the runtime is expected to grow. However, it is crucial to compare the two methods to quantify their performance impact, enabling us to identify the most efficient approach for computing eigenvalues. Additionally, this comparison helps evaluate the overhead introduced by erasure coding relative to the best-performing method.

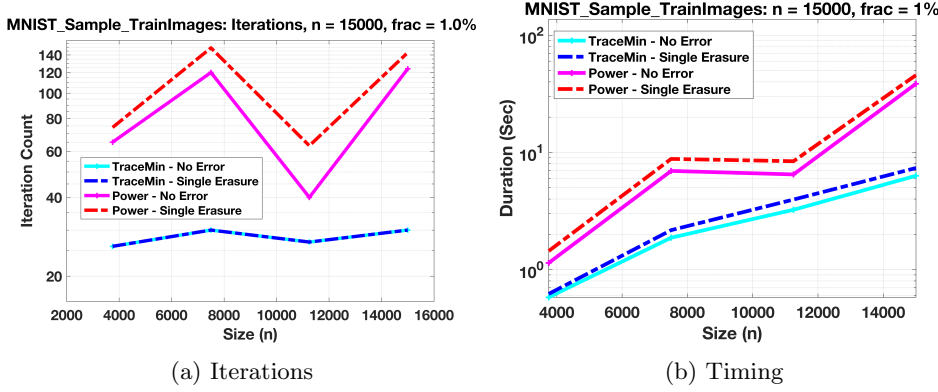


Fig. 3: MNIST Train Dataset with 0.1% Erased

Figures 3a and 3b compare the convergence behavior and computational cost of the **TraceMin** and **Power Method** algorithms under both **no-error** and **single-fault** scenarios for varying problem sizes on the MNIST dataset.

Figure 3b clearly demonstrates that while erasure coding introduces some overhead for both the Power Method and TraceMin, the TraceMin algorithm consistently outperforms the Power Method, achieving significantly faster runtimes—often by an order of magnitude—across all matrix sizes. Although the runtime of both methods increases with problem size, the performance degradation is far more pronounced for the Power Method. The Single Erasure cases for both algorithms exhibit higher runtimes than their no-error counterparts due to the additional fill-in introduced by erasure coding—nonzero coding rows and columns that increase the effective matrix density and, consequently, the computational cost per iteration. Overall, TraceMin demonstrates superior scalability and robustness to erasures, whereas the Power Method shows a steeper increase in both runtime and iteration count, underscoring its inefficiency for large-scale or fault-tolerant eigenvalue computations.

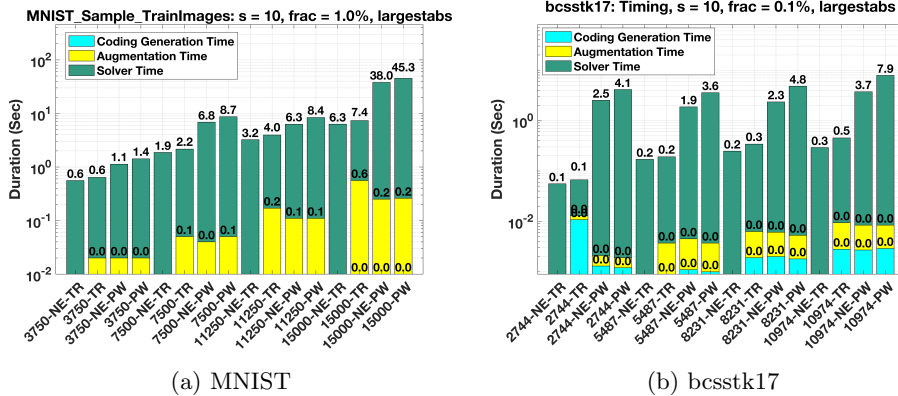


Fig. 4: Timing Breakdown

In Figure 4, the runs for Power Iteration with optimized solver (Algorithm 3.2) and TraceMin (Algorithm 3.4) are labeled with the suffixes “PW” and “TR”, respectively, while “NE” represents the “No Erasure” scenario. The figure illustrates that in both dense matrix scenarios (e.g., MNIST) and sparse matrix scenarios (e.g., bcsstk17), TraceMin is significantly faster than the Power Iteration method. This disparity arises because the solver time is substantially higher for the Power Iteration method compared to TraceMin.

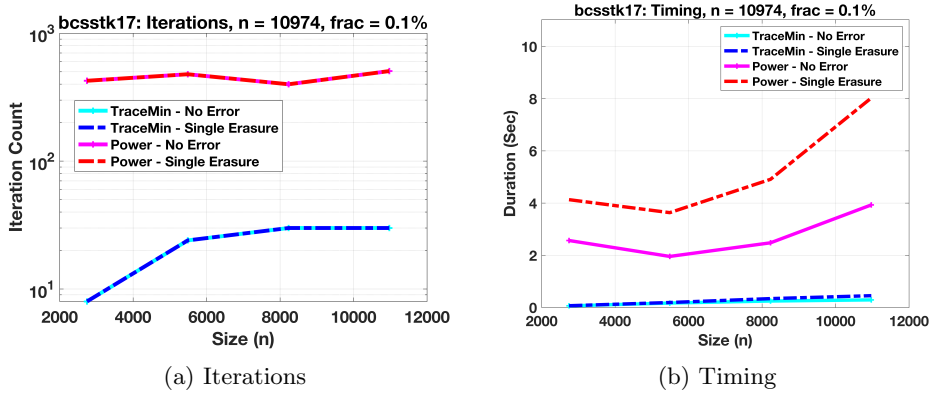


Fig. 5: Sparse bcsstk17 Train Dataset with 0.1% Erased

As shown in Figure 5a, the Power Method (Algorithms 3.1 and 3.1) requires significantly more iterations to converge for sparse dataset **bcsstk17** compared to Erasure-Coded TraceMin (Algorithm 3.4), leading to a higher overall execution time, as illustrated in Figure 5b. Furthermore, Figure 5b demonstrates that the erasure scenario incurs a higher runtime than its no-erasure counterpart. This increase in computational cost arises because erasure coding introduces additional fill-in in the sparse system due to nonzero coding rows and columns, thereby increasing the time required for each iteration and ultimately resulting in a longer end-to-end execution time. The results clearly indicate that the TraceMin algorithm substantially outperforms the Power Method across all problem sizes. The runtime of TraceMin remains consistently low and scales modestly with increasing matrix size, demonstrating its computational efficiency and scalability. In contrast, the Power Method exhibits significantly higher runtimes, particularly under the Single Erasure scenario, where the cost increases sharply with problem size.

The Power Method is designed to compute the largest eigenvalue and its corresponding eigenvector by using subspace iterations. As the subspace size increases, the size of eigenvectors increases as well and hence, the computational requirements grow significantly, making the method progressively more resource-intensive and costly to execute.

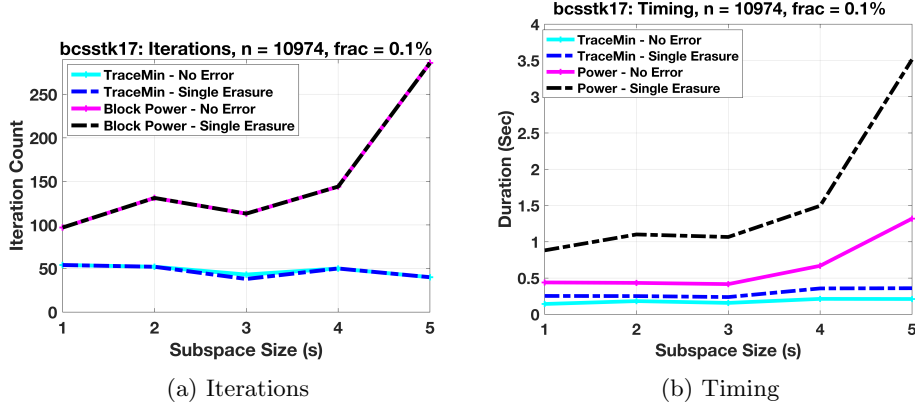


Fig. 6: Convergence for bcsstk17 Dataset for varying subspace size with 0.1% Erased

Figure 6 illustrates that as the subspace size increases, the runtime of TraceMin remains relatively stable, whereas the Power Method becomes progressively more expensive due to the sequential computation of eigenvalues. Furthermore, we observe that the erasure-coded scenarios incur slightly higher runtimes than their no-erasure counterparts, primarily because erasure coding introduces additional fill-in arising from the added coding rows and columns.

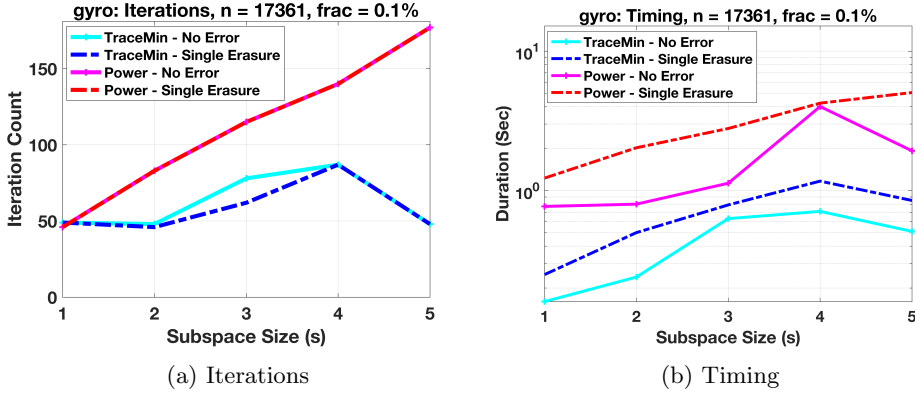


Fig. 7: Convergence of Eigensolvers for gyro Dataset for varying subspace size

Similarly, as shown in Figure 7a, the Power Method requires more iterations to converge on the **gyro** dataset when computing the dominant eigenvalues, resulting in significantly higher execution times compared to TraceMin (as illustrated in Figure 7b). Since TraceMin consistently outperforms the Power Iteration method, the remainder of this paper focuses on analyzing the overheads introduced by erasure coding and comparing other relevant benchmark results with respect to TraceMin.

4.2. Convergence Results. We present here convergence results for covariance matrices computed from covariance matrix based on MNIST [8] and CIFAR-10 [13]

Train and Test datasets.

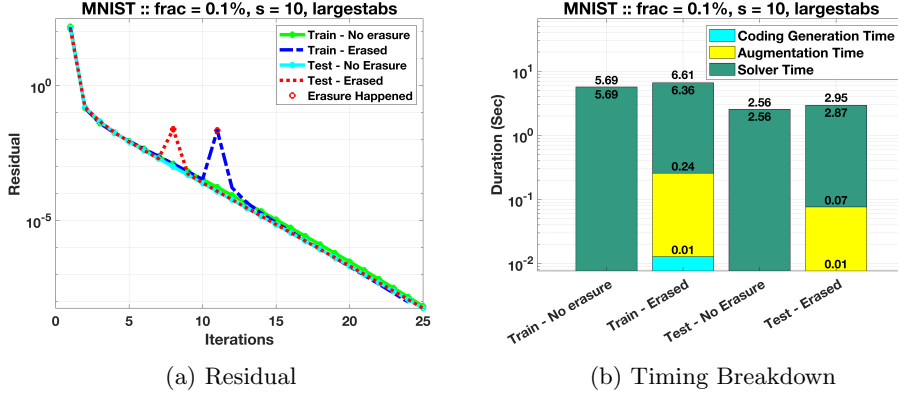


Fig. 8: Computing Largest Few Eigenvalue for MNIST Dataset with 0.1% Erased

Figure 8a shows that the Erasure Coded TraceMin successfully recovers from faults and, within a few iterations, follows a residual pattern similar to its “No Erasure” counterparts. This indicates that Erasure-Coded TraceMin exhibits excellent convergence characteristics for MNIST Train and Test datasets. Furthermore, as shown in Figure 8b, Erasure Coding introduces minimal additional computational time compared to the “No Erasure” scenario, resulting in a significantly low overhead for the Erasure Coding scheme.

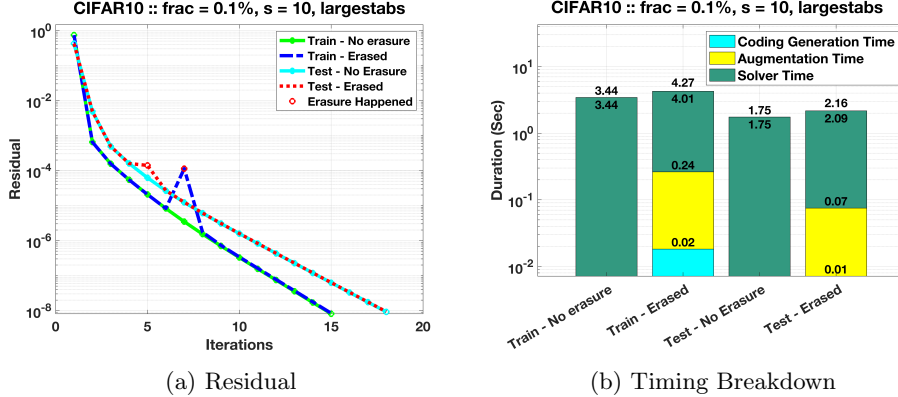


Fig. 9: Computing Largest Few Eigenvalue for CIFAR10 Dataset with 0.1% Erased

Figures 8 and 9 illustrate the TraceMin iteration errors for the MNIST and CIFAR-10 training and test datasets. We observe that while the residual increases in the event of an erasure, the Erasure-Coded TraceMin still converges efficiently without requiring significantly more iterations. This demonstrates the excellent convergence properties of the Erasure-Coded Eigensolver when computing the largest eigenvalues.

Furthermore, Figure 9b highlights the minimal overhead introduced by Erasure-Coded TraceMin, making it a highly efficient and robust choice.

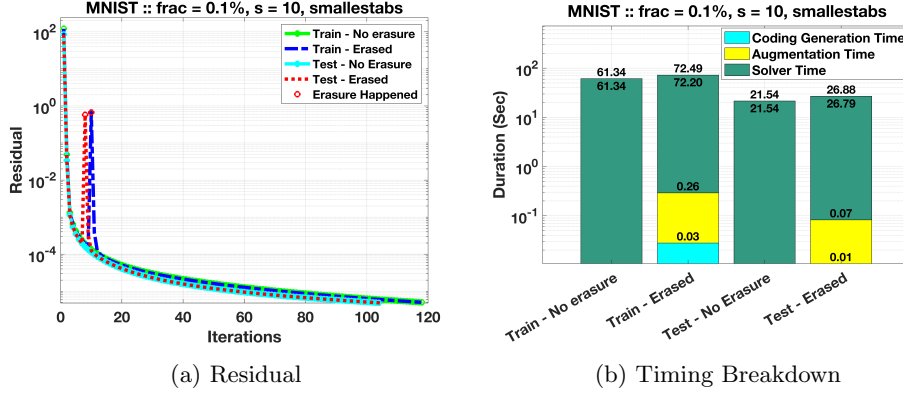


Fig. 10: Computing Smallest Few Eigenvalue for MNIST Dataset with 1% Erased

The Figure 10 shows that at the event of failure, erasure-coded TraceMin converges to its solution without taking many more iterations making the erasure coding an obvious choice. Erasure-Coded TraceMin showing impeccable convergence when computing smallest eigenvalues as well as shown in Figure 10.

We observe from Figure 10 that erasure-coded TraceMin recovers from 0.1% erasures (150 row-column pairs) of MNIST Training Dataset to compute exact eigen-pairs (tested up to 15 largest eigen-pairs). The erasure-coded TraceMin takes less than 20% additional iterations to converge in case of an erasure than the “No Erasure” scenario.

4.3. Multifault. We implement a Random Multifault model in which multiple faults can occur at any point during execution. These faults are simulated using a random number distribution, where a predefined set of iterations is selected to simulate fault occurrences, resulting in data erasure (loss).

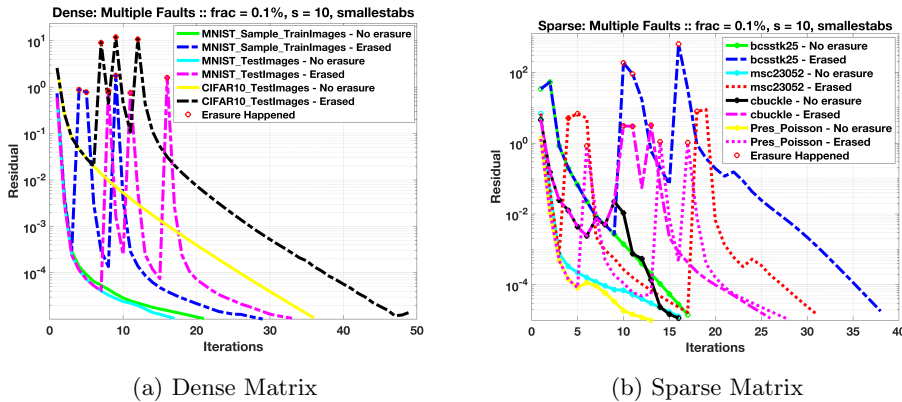


Fig. 11: Erasure Coded TraceMin in Multifault Scenario

In Figure 11, we observe that Erasure-Coded TraceMin successfully computes eigenvalues even in the presence of multiple failures, requiring only a modest increase in iterations. This excellent convergence behavior remains consistent across both sparse and dense datasets. To demonstrate the generalizability of our Erasure-Coded scheme, we select a representative set of dense and sparse datasets. Furthermore, Figure 11a shows that Erasure-Coded TraceMin requires significantly fewer iterations than the product of the number of faults and the iterations needed by TraceMin in the “No Erasure” scenario. In other words, Erasure Coding imposes substantially lower overhead compared to restarting the solver in the event of multiple failures.

5. Conclusion. In this work, we introduce a novel erasure-coded, fault-tolerant eigenvalue solver, establishing its correctness and convergence properties. Through extensive evaluations, we demonstrate that our solver incurs minimal overhead across different fault types while maintaining excellent convergence behavior. Our approach exhibits strong robustness and performance across various fault rates and standard benchmark datasets, highlighting its effectiveness in real-world scenarios. By enabling reliable eigenvalue computations in faulty environments, our methods provide a crucial computational foundation for a wide range of scientific applications where resilience and efficiency are paramount.

Acknowledgments. We would like to acknowledge NSF for supporting the research by grant. DFG would like to acknowledge DOE DE-SC0023162 Sparstitute MMICC center for partial support as well as NSF Nonlinear graph IIS-2007481.

REFERENCES

- [1] W. BLAND, P. DU, A. BOUTEILLER, T. HERAULT, G. BOSILCA, AND J. DONGARRA, *A checkpoint-on-failure protocol for algorithm-based recovery in standard mpi*, in Proceedings of the 18th International Conference on Parallel Processing, Euro-Par’12, Berlin, Heidelberg, 2012, Springer-Verlag, p. 477–488, https://doi.org/10.1007/978-3-642-32820-6_48, https://doi.org/10.1007/978-3-642-32820-6_48.
- [2] G. BOSILCA, R. DELMAS, J. DONGARRA, AND J. LANGOU, *Algorithm-based fault tolerance applied to high performance computing*, Journal of Parallel and Distributed Computing, 69 (2009), pp. 410–416, <https://doi.org/https://doi.org/10.1016/j.jpdc.2008.12.002>, <https://www.sciencedirect.com/science/article/pii/S0743731508002141>.
- [3] A. BOUTEILLER, G. BOSILCA, AND J. DONGARRA, *Retrospect: Deterministic replay of mpi applications for interactive distributed debugging*, in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)—Lect. Notes Comput. Sci., vol. 4757 of Lecture Notes in Computer Science, United States, 2007, Springer Nature, pp. 297–306, https://doi.org/10.1007/978-3-540-75416-9_41, <http://dblp.uni-trier.de/db/conf/pvm/pvm2007.html#BouteillerBD07><http://dblp.uni-trier.de/rec/bibtex/conf/pvm/BouteillerBD07.xml><http://dblp.uni-trier.de/rec/bibtex/conf/pvm/BouteillerBD07>. 14th European PVM/MPI Users’ Group Meeting on Parallel Virtual Machine and Message Passing Interface ; Conference date: 01-07-2007.
- [4] P. G. BRIDGES, K. B. FERREIRA, M. HEROUX, AND M. HOEMMEN, *Fault-tolerant linear solvers via selective reliability*, ArXiv, abs/1206.1390 (2012), <https://api.semanticscholar.org/CorpusID:16963719>.
- [5] Z. CHEN, *Optimal real number codes for fault tolerant matrix operations*, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC ’09, New York, NY, USA, 2009, Association for Computing Machinery, <https://doi.org/10.1145/1654059.1654089>, <https://doi.org/10.1145/1654059.1654089>.
- [6] Z. CHEN AND J. DONGARRA, *Algorithm-based fault tolerance for fail-stop failures*, IEEE Transactions on Parallel and Distributed Systems, 19 (2008), pp. 1628–1641, <https://doi.org/10.1109/TPDS.2008.58>.
- [7] Z. CHEN, G. E. FAGG, E. GABRIEL, J. LANGOU, T. ANGSKUN, G. BOSILCA, AND J. DONGARRA, *Fault tolerant high performance computing by a coding approach*, in Proceedings of the

- Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP '05, New York, NY, USA, 2005, Association for Computing Machinery, p. 213–223, <https://doi.org/10.1145/1065944.1065973>, <https://doi.org/10.1145/1065944.1065973>.
- [8] L. DENG, *The mnist database of handwritten digit images for machine learning research*, IEEE Signal Processing Magazine, 29 (2012), pp. 141–142.
- [9] K.-H. HUANG AND J. A. ABRAHAM, *Algorithm-based fault tolerance for matrix operations*, IEEE Transactions on Computers, C-33 (1984), pp. 518–528, <https://doi.org/10.1109/TC.1984.1676475>.
- [10] X. KANG, D. F. GLEICH, A. SAMEH, AND A. GRAMA, *Distributed fault tolerant linear system solvers based on erasure coding*, in 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 2478–2485, <https://doi.org/10.1109/ICDCS.2017.261>.
- [11] X. KANG, D. F. GLEICH, A. SAMEH, AND A. GRAMA, *Adaptive erasure coded fault tolerant linear system solver*, ACM Trans. Parallel Comput., 8 (2021), <https://doi.org/10.1145/3490557>, <https://doi.org/10.1145/3490557>.
- [12] A. KLINVEX, F. SAIED, AND A. SAMEH, *Parallel implementations of the trace minimization scheme tracemin for the sparse symmetric eigenvalue problem*, Computers & Mathematics with Applications, 65 (2013), pp. 460–468, <https://doi.org/https://doi.org/10.1016/j.camwa.2012.06.011>, <https://www.sciencedirect.com/science/article/pii/S0898122112004415>. Efficient Numerical Methods for Scientific Applications.
- [13] A. KRIZHEVSKY, *Learning multiple layers of features from tiny images*, tech. report, Department of Computer Science, University of Toronto, 2009.
- [14] J. B. KRUSKAL, *Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics*, Linear Algebra and its Applications, 18 (1977), pp. 95–138, <https://api.semanticscholar.org/CorpusID:121354144>.
- [15] F. T. LUK AND H. PARK, *An analysis of algorithm-based fault tolerance techniques*, Journal of Parallel and Distributed Computing, 5 (1988), pp. 172–184, [https://doi.org/https://doi.org/10.1016/0743-7315\(88\)90027-5](https://doi.org/https://doi.org/10.1016/0743-7315(88)90027-5), <https://www.sciencedirect.com/science/article/pii/0743731588900275>.
- [16] T. ROPARS, A. LEFRAY, D. KIM, AND A. SCHIPER, *Efficient process replication for mpi applications: Sharing work between replicas*, in 2015 IEEE International Parallel and Distributed Processing Symposium, 2015, pp. 645–654, <https://doi.org/10.1109/IPDPS.2015.29>.
- [17] A. SAMEH AND Z. TONG, *The trace minimization method for the symmetric generalized eigenvalue problem*, Journal of Computational and Applied Mathematics, 123 (2000), pp. 155–175, [https://doi.org/10.1016/S0377-0427\(00\)00391-5](https://doi.org/10.1016/S0377-0427(00)00391-5).
- [18] A. H. SAMEH AND J. A. WISNIEWSKI, *A Trace Minimization Algorithm for the Generalized Eigenvalue Problem*, SIAM Journal on Numerical Analysis, 19 (1982), pp. 1243–1259, <https://doi.org/10.1137/0719089>.
- [19] Y. ZHU, D. F. GLEICH, AND A. GRAMA, *Erasure coding for fault-oblivious linear system solvers*, SIAM Journal on Scientific Computing, 39 (2017), pp. C48–C64, <https://doi.org/10.1137/15M1041511>, <https://doi.org/10.1137/15M1041511>, <https://arxiv.org/abs/https://doi.org/10.1137/15M1041511>.
- [20] Y. ZHU, D. F. GLEICH, AND A. Y. GRAMA, *Erasure coding for fault-oblivious linear system solvers*, SIAM J. Sci. Comput., 39 (2014), <https://api.semanticscholar.org/CorpusID:16958418>.