A Memory-Efficient Two-Pass Lanczos Algorithm

Luca Lombardo

Abstract

This work investigates the two-pass Lanczos algorithm for computing matrix functions $f(\mathbf{A})\mathbf{b}$. We analyze its memory and computational complexities, showing a reduction in memory from $O(nk)$ to $O(n)$ at the cost of doubling matrix-vector products. Numerical experiments validate this model. The observed wall-clock time demonstrates that while computational work increases, actual performance can be comparable to, or even exceed, the one-pass method in memory-bound scenarios due to favorable cache utilization. Furthermore, the two-pass algorithm maintains numerical accuracy and stability equivalent to the standard approach, even for challenging ill-conditioned problems.

Contents

1 Introduction

The computation of the action of a matrix function on a vector, an operation denoted as $\mathbf{x} = f(\mathbf{A})\mathbf{b}$, represents a fundamental task in numerical linear algebra and scientific computing [1, 2]. For large, sparse Hermitian matrices, methods based on Krylov subspaces are standard. A foundational algorithm in this class is the Lanczos process, first introduced as a *method of minimized iterations* for eigenvalue problems [3]. Its standard implementation, however, encounters a severe practical limitation: the necessity of storing an ever-growing basis of Lanczos vectors. This memory requirement often becomes prohibitive for the large-scale problems encountered in practice [1].

To address this memory bottleneck, we examine a simple and effective variant known as the two-pass Lanczos method [2]. This approach fundamentally alters the standard algorithm by separating the computation into two distinct phases. It first generates the projection of the matrix without storing the basis, and then regenerates the basis in a second pass to construct the final solution. This report provides an analysis of this method, focusing on the explicit trade-off it presents: a significant reduction in memory storage at the cost of an increased computational workload.

1.1 The Problem of Computing Matrix Functions

We consider the problem of computing the vector $\mathbf{x} \in \mathbb{C}^n$ defined by the action of a matrix function on a vector $\mathbf{b} \in \mathbb{C}^n$,

$$\mathbf{x} = f(\mathbf{A})\mathbf{b}, \tag{1}$$

where $\mathbf{A} \in \mathbb{C}^{n\times n}$ is a large, sparse Hermitian matrix and $f : \Omega \subseteq \mathbb{C} \to \mathbb{C}$ is a function defined on the spectrum of $\mathbf{A}$, $\sigma(\mathbf{A}) \subset \Omega$. The explicit computation of the matrix $f(\mathbf{A})$ is generally infeasible due to its high computational cost and the fact that $f(\mathbf{A})$ is typically a dense matrix even when $\mathbf{A}$ is sparse [2]. Our focus, therefore, is on methods that compute $\mathbf{x}$ directly.

This problem is fundamental in many areas of scientific computing [1]. A primary example is the solution of a large system of linear equations $\mathbf{Ax} = \mathbf{b}$, which corresponds to the case where $f(z) = z^{-1}$ and $\mathbf{A}$ is nonsingular [4]. Another critical application arises in the numerical solution of systems of linear ordinary differential equations. The solution to the initial value problem $\mathbf{y}'(t) = \mathbf{Ay}(t)$, with $\mathbf{y}(0) = \mathbf{y}_0$, is given by $\mathbf{y}(t) = \exp(t\mathbf{A})\mathbf{y}_0$. This computation is an instance of our problem where the function is the matrix exponential, $f(z) = \exp(tz)$ [2].

1.2 Krylov Subspace Methods: The Standard Approach

The standard framework for computing an approximation to $\mathbf{x} = f(\mathbf{A})\mathbf{b}$ for large matrices is based on projection onto a Krylov subspace [2, 4]. We begin by formally defining this subspace.

Definition 1.1 (Krylov Subspace)**.** Given a matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$ and a vector $\mathbf{b} \in \mathbb{C}^n$, the k-th Krylov subspace generated by $\mathbf{A}$ and $\mathbf{b}$ is the vector space

$$\mathcal{K}_k(\mathbf{A}, \mathbf{b}) = \operatorname{span}\{\mathbf{b}, \mathbf{Ab}, \ldots, \mathbf{A}^{k-1}\mathbf{b}\}. \tag{2}$$

The fundamental principle of a Krylov subspace method is to seek an approximate solution $\mathbf{x}_k$ within the low-dimensional subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$, where $k \ll n$. This is achieved through a projection process that can be summarized in three distinct stages. First, we construct an orthonormal basis, typically via the Arnoldi or Lanczos iteration, for the subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$. Let this basis be the columns of the matrix $\mathbf{V}_k \in \mathbb{C}^{n \times k}$.

Second, we project the high-dimensional operator $\mathbf{A}$ onto $\mathcal{K}_k(\mathbf{A}, \mathbf{b})$, which yields a small $k \times k$ matrix representation of the operator, typically a Hessenberg or tridiagonal matrix $\mathbf{T}_k = \mathbf{V}_k^H \mathbf{A} \mathbf{V}_k$. The original problem is thereby reduced to the evaluation of $f(\mathbf{T}_k)$, a task which is computationally feasible for small k.

Finally, the solution to the projected problem is lifted back to the original n-dimensional space to form the final approximation. Assuming the starting vector for the basis construction is $\mathbf{v}_1 = \mathbf{b}/\|\mathbf{b}\|_2$, the approximation $\mathbf{x}_k$ to $\mathbf{x}$ is given by

$$\mathbf{x}_k = \mathbf{V}_k f(\mathbf{T}_k) \mathbf{e}_1 \|\mathbf{b}\|_2, \tag{3}$$

where $\mathbf{e}_1$ is the first canonical basis vector in $\mathbb{C}^k$ [2].

1.3 The Memory Bottleneck of the Standard Lanczos Process

The practical utility of approximating the solution via $\mathbf{x}_k = \mathbf{V}_k f(\mathbf{T}_k) \mathbf{e}_1 \|\mathbf{b}\|_2$ is conditioned on our ability to construct and store the basis matrix $\mathbf{V}_k$. In a standard one-pass implementation, the Lanczos vectors $\{\mathbf{v}_j\}_{j=1}^k$ are generated sequentially and must all be retained in memory. This is because the final step involves forming a linear combination of these vectors, weighted by the components of the vector $\mathbf{y}_k = f(\mathbf{T}_k) \mathbf{e}_1 \|\mathbf{b}\|_2$.

This requirement imposes a memory cost that scales as $O(nk)$. For many problems of practical interest, particularly those arising from the discretization of partial differential equations, the dimension n can be on the order of millions or larger. Concurrently, achieving a desired accuracy may require a number of iterations k that is substantial, leading to a storage demand that exceeds the capacity of modern computing systems [1]. This memory constraint is the primary bottleneck of the standard Lanczos process.

To overcome this limitation, memory-efficient strategies are necessary. The two-pass Lanczos method is a simple but effective approach designed explicitly to address this problem of memory consumption [2]. The method is structured to avoid storing the entire basis $\mathbf{V}_k$ by decoupling the generation of the projected matrix $\mathbf{T}_k$ from the final synthesis of the solution vector $\mathbf{x}_k$.

2 Symmetric Lanczos Process

To analyze the two-pass variant, we first take a step back and review the standard symmetric Lanczos process. The derivation begins from the method of minimized iterations, as originally formulated by Lanczos [3]. This approach demonstrates that the process of generating orthogonal

vectors through successive applications of a symmetric operator yields a three-term recurrence relation. We then formalize this recurrence using modern matrix notation [1].

2.1 Derivation via Successive Orthogonalization

We derive the Lanczos recurrence following the method of minimized iterations for a Hermitian operator $\mathbf{A} \in \mathbb{C}^{n\times n}$ [3]. The procedure constructs a sequence of mutually orthogonal vectors $\{\mathbf{b}_k\}_{k=0}^{m-1}$ from an arbitrary starting vector $\mathbf{b}_0 \in \mathbb{C}^n$, where $\mathbf{b}_0 \neq \mathbf{0}$.

The sequence is generated iteratively. At each step $k \geq 1$, a new vector $\mathbf{b}_k$ is formed by minimizing the Euclidean norm of a vector obtained by applying $\mathbf{A}$ to the previous vector $\mathbf{b}_{k-1}$ and removing its components along all previously generated vectors. For the first step, we define $\mathbf{b}_1$ as

$$\mathbf{b}_1 = \mathbf{A}\mathbf{b}_0 - \alpha_0 \mathbf{b}_0,$$

where the scalar α_0 is chosen to minimize $\|\mathbf{b}_1\|_2$. This minimization is equivalent to enforcing the orthogonality condition $\mathbf{b}_1 \perp \mathbf{b}_0$. The inner product $(\mathbf{A}\mathbf{b}_0 - \alpha_0\mathbf{b}_0, \mathbf{b}_0) = 0$ yields the coefficient

$$\alpha_0 = \frac{(\mathbf{A}\mathbf{b}_0, \mathbf{b}_0)}{(\mathbf{b}_0, \mathbf{b}_0)}.$$

The general step for $k \geq 2$ is to construct $\mathbf{b}_k$ from $\mathbf{A}\mathbf{b}_{k-1}$ by ensuring it is orthogonal to the entire set $\{\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{k-1}\}$. This is achieved by defining $\mathbf{b}_k$ as

$$\mathbf{b}_k = \mathbf{A}\mathbf{b}_{k-1} - \sum_{i=0}^{k-1} c_i \mathbf{b}_i.$$

The coefficients c_i are determined by the orthogonality conditions $(\mathbf{b}_k, \mathbf{b}_i) = 0$ for $i = 0, \ldots, k-1$. Due to the mutual orthogonality of the basis vectors $\{\mathbf{b}_j\}$, this simplifies to

$$c_i = \frac{(\mathbf{A}\mathbf{b}_{k-1}, \mathbf{b}_i)}{(\mathbf{b}_i, \mathbf{b}_i)}.$$

A key property emerges when the operator $\mathbf{A}$ is Hermitian. For any i, we can write

$$(\mathbf{A}\mathbf{b}_{k-1}, \mathbf{b}_i) = (\mathbf{b}_{k-1}, \mathbf{A}\mathbf{b}_i).$$

From the construction of the sequence, the vector $\mathbf{b}_{i+1}$ is a linear combination of $\mathbf{A}\mathbf{b}_i$ and the preceding vectors $\mathbf{b}_i, \ldots, \mathbf{b}_0$. It follows that $\mathbf{A}\mathbf{b}_i$ lies in the span of $\{\mathbf{b}_0, \ldots, \mathbf{b}_{i+1}\}$. By the orthogonality of the sequence, the inner product $(\mathbf{b}_{k-1}, \mathbf{A}\mathbf{b}_i)$ must be zero if $k-1 > i+1$, or equivalently, if $i < k-2$. Consequently, the coefficients c_i are zero for all $i < k-2$.

This establishes that the summation in the general step collapses, leaving a three-term recurrence relation. Following the notation in [3], we define $\alpha_{k-1} = c_{k-1}$ and $\beta_{k-2} = c_{k-2}$. The recurrence simplifies to

$$\mathbf{b}_k = \mathbf{A}\mathbf{b}_{k-1} - \alpha_{k-1}\mathbf{b}_{k-1} - \beta_{k-2}\mathbf{b}_{k-2},$$

which is the foundational recurrence of the symmetric Lanczos process.

2.2 Matrix Representation of the Lanczos Recurrence

The three-term recurrence relation derived from the principle of minimized iterations can be expressed in a compact matrix form. Let us define a sequence of orthonormal vectors $\{\mathbf{v}_j\}_{j=1}^k$ from the orthogonal vectors $\{\mathbf{b}_j\}_{j=1}^k$. We initialize the process with a unit-norm vector $\mathbf{v}_1 = \mathbf{b}_0/\|\mathbf{b}_0\|_2$, where we adopt the notation from [4]. The recurrence relation can be written as

$$\beta_j \mathbf{v}_{j+1} = \mathbf{A}\mathbf{v}_j - \alpha_j \mathbf{v}_j - \beta_{j-1}\mathbf{v}_{j-1},$$

where, following the convention in [4], we set $\mathbf{v}_0 = \mathbf{0}$. The orthonormality of the vectors $\{\mathbf{v}_j\}$ dictates the choice of the coefficients. Specifically, taking the inner product of the above relation with $\mathbf{v}_j$ yields

$$\alpha_j = \mathbf{v}_j^H \mathbf{A}\mathbf{v}_j.$$

The coefficient β_j is determined by the normalization condition $\|\mathbf{v}_{j+1}\|_2 = 1$, which gives $\beta_j = \|\mathbf{A}\mathbf{v}_j - \alpha_j\mathbf{v}_j - \beta_{j-1}\mathbf{v}_{j-1}\|_2$.

After k steps of this process, for $j = 1,\ldots,k$, we can write the set of recurrence relations in matrix form. Let $\mathbf{V}_k = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k] \in \mathbb{C}^{n\times k}$ be the matrix whose columns are the Lanczos vectors. The recurrences can be collected into a single matrix equation

$$\mathbf{A}\mathbf{V}_k = \mathbf{V}_k\mathbf{T}_k + \beta_k \mathbf{v}_{k+1}\mathbf{e}_k^T,$$

where $\mathbf{e}_k$ is the k-th canonical basis vector in $\mathbb{C}^k$, and $\mathbf{T}_k$ is the $k \times k$ real symmetric tridiagonal matrix

$$\mathbf{T}_k = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{pmatrix}$$

From the orthonormality of the columns of $\mathbf{V}_k$, i.e., $\mathbf{V}_k^H\mathbf{V}_k = \mathbf{I}_k$, it follows that the tridiagonal matrix $\mathbf{T}_k$ is the orthogonal projection of $\mathbf{A}$ onto the Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{v}_1)$, since

$$\mathbf{V}_k^H\mathbf{A}\mathbf{V}_k = \mathbf{V}_k^H(\mathbf{V}_k\mathbf{T}_k + \beta_k\mathbf{v}_{k+1}\mathbf{e}_k^T) = \mathbf{T}_k.$$

The tridiagonal and symmetric structure of $\mathbf{T}_k$ is a direct consequence of the three-term recurrence relation inherent to the orthogonalization process with a Hermitian operator.

3 The Two-Pass Lanczos Algorithm

Having established the theoretical framework for the standard Lanczos process and identified its principal limitation[1] we now present a memory-efficient variant known as the two-pass Lanczos algorithm [2, 5]. The algorithm resolves the memory constraint by decoupling the computation of the projected tridiagonal matrix $\mathbf{T}_k$ from the synthesis of the final solution vector $\mathbf{x}_k$.

[1] the prohibitive memory cost of storing the basis vectors

In this section, we provide a description of the two distinct phases of the algorithm: an initial pass to generate the coefficients of $\mathbf{T}_k$ without storing the basis, and a second pass to reconstruct the solution vector. We then present an analysis of the method's core trade-off, quantifying its memory savings against its increased computational cost.

3.1 Algorithmic Formulation

The two-pass Lanczos algorithm computes the standard Lanczos approximation $\mathbf{x}_k$ while avoiding the $O(nk)$ memory cost associated with storing the basis matrix $\mathbf{V}_k$. It achieves this by executing two distinct computational passes. The first pass computes the tridiagonal projection $\mathbf{T}_k$, and the second pass synthesizes the solution vector $\mathbf{x}_k$.

3.1.1 First Pass: Projection and Coefficient Generation

The objective of the first pass is to compute the scalar entries of the $k \times k$ symmetric tridiagonal matrix $\mathbf{T}_k$. This is accomplished by executing k steps of the Lanczos iteration. The process generates a sequence of orthonormal vectors $\{\mathbf{v}_j\}$ via a three-term recurrence, which we derive here following the formulation in [1].

The process is initialized with a starting vector of unit norm, $\mathbf{v}_1 = \mathbf{b}/\|\mathbf{b}\|_2$. The core of the algorithm is the following recurrence relation, where we define $\beta_0 = 0$ and $\mathbf{v}_0 = \mathbf{0}$:

$$\beta_j \mathbf{v}_{j+1} = \mathbf{A}\mathbf{v}_j - \alpha_j \mathbf{v}_j - \beta_{j-1}\mathbf{v}_{j-1}. \tag{4}$$

The scalars α_j and β_j are determined at each step $j = 1, \ldots, k$ by enforcing the orthonormality of the sequence $\{\mathbf{v}_j\}$. Taking the inner product of equation (4) with $\mathbf{v}_j$ and leveraging the orthogonality of the previously constructed vectors, we obtain the expression for the diagonal elements of $\mathbf{T}_k$:

$$\alpha_j = \mathbf{v}_j^H \mathbf{A}\mathbf{v}_j.$$

The off-diagonal elements, β_j, are determined by the normalization condition $\|\mathbf{v}_{j+1}\|_2 = 1$. Let $\mathbf{r}_j$ be the residual vector on the right-hand side of (4) before normalization:

$$\mathbf{r}_j = \mathbf{A}\mathbf{v}_j - \alpha_j \mathbf{v}_j - \beta_{j-1}\mathbf{v}_{j-1}.$$

Then, the coefficient β_j is its Euclidean norm,

$$\beta_j = \|\mathbf{r}_j\|_2.$$

If $\beta_j = 0$, the process terminates. Otherwise, the next Lanczos vector is computed as $\mathbf{v}_{j+1} = \mathbf{r}_j/\beta_j$.

The structure of the three-term recurrence in (4) is a critical property for memory-efficient implementations. In exact arithmetic, it ensures that the newly generated vector $\mathbf{v}_{j+1}$ is orthogonal to all preceding vectors $\mathbf{v}_1, \ldots, \mathbf{v}_j$, despite its construction using only $\mathbf{v}_j$ and $\mathbf{v}_{j-1}$ [1]. This allows for the sequential generation of the basis while only requiring storage for a constant number of

n-dimensional vectors at any given time, thus addressing the memory bottleneck of the standard Lanczos process [2].

After k iterations, the stored sequences of scalars $\{\alpha_j\}_{j=1}^{k}$ and $\{\beta_j\}_{j=1}^{k-1}$ are used to construct the real, symmetric tridiagonal matrix $\mathbf{T}_k$. The final operation of the first pass is the computation of the coefficient vector $\mathbf{y}_k \in \mathbb{C}^k$:

$$\mathbf{y}_k = f(\mathbf{T}_k)\mathbf{e}_1 \|\mathbf{b}\|_2. \tag{5}$$

This vector contains the coordinates of the approximate solution $\mathbf{x}_k$ with respect to the orthonormal basis $\mathbf{V}_k$. The computation of $f(\mathbf{T}_k)$ is a small-scale dense matrix problem whose cost is independent of n, and is thus considered negligible for $k \ll n$.

3.1.2 Second Pass: Solution Reconstruction

In the second pass, we construct the final solution vector $\mathbf{x}_k$. The vector $\mathbf{x}_k$ is the linear combination of the Lanczos basis vectors, where the coefficients are the components of the vector $\mathbf{y}_k$ computed in the first pass. We define the approximation as

$$\mathbf{x}_k = \mathbf{V}_k\mathbf{y}_k = \sum_{j=1}^{k} (\mathbf{y}_k)_j \mathbf{v}_j. \tag{6}$$

To compute this sum without storing the matrix $\mathbf{V}_k$, we re-generate the Lanczos vectors sequentially [2].

We re-initialize the process with the starting vector $\mathbf{v}_1 = \mathbf{b}/\|\mathbf{b}\|_2$. We then execute the Lanczos iteration a second time, using the scalars $\{\alpha_j\}_{j=1}^{k}$ and $\{\beta_j\}_{j=1}^{k-1}$ that we stored during the first pass. For each $j = 1, \ldots, k-1$, we reconstruct the subsequent Lanczos vectors via the same three-term recurrence from equation (4):

$$\mathbf{v}_{j+1} = \frac{1}{\beta_j}\left(\mathbf{A}\mathbf{v}_j - \alpha_j\mathbf{v}_j - \beta_{j-1}\mathbf{v}_{j-1}\right). \tag{7}$$

As we re-generate each vector $\mathbf{v}_j$, we immediately use it to form the sum in equation (6). We initialize the solution vector $\mathbf{x}_k$ to zero. After computing $\mathbf{v}_1$, we form the first term of the sum. Then, for each $j = 1, \ldots, k-1$, after computing $\mathbf{v}_{j+1}$ via (7), we update the solution as follows:

$$\mathbf{x}_k \leftarrow \mathbf{x}_k + (\mathbf{y}_k)_{j+1}\mathbf{v}_{j+1}. \tag{8}$$

The memory management is identical to that of the first pass. We use each vector $\mathbf{v}_j$ to compute $\mathbf{v}_{j+1}$ and to update the sum for $\mathbf{x}_k$. Afterwards, we retain it only as long as required for the next step of the recurrence. After k steps, the accumulation is complete. The procedure is formally described in Algorithm 1.

3.2 Computational and Memory Complexity

We now provide a formal analysis of the memory requirements and computational cost of the two-pass Lanczos algorithm. To formalize this analysis, we model the total peak memory usage,

Algorithm 1 The Two-Pass Lanczos Method for $\mathbf{x} = f(\mathbf{A})\mathbf{b}$

Input: Hermitian matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$, vector $\mathbf{b} \in \mathbb{C}^n$, function f, number of iterations k.
Output: Approximate solution $\mathbf{x}_k \in \mathbb{C}^n$.

▷ — *First Pass: Coefficient Generation* —

Store $\mathbf{b}$ for the second pass. Let $\|\mathbf{b}\|_2$ be its norm.
$\beta_0 \leftarrow 0$, $\mathbf{v}_{\text{prev}} \leftarrow \mathbf{0}$, $\mathbf{v} \leftarrow \mathbf{b}/\|\mathbf{b}\|_2$.
for $j = 1, \ldots, k$ **do**
  $\mathbf{w} \leftarrow \mathbf{A}\mathbf{v} - \beta_{j-1}\mathbf{v}_{\text{prev}}$
  $\alpha_j \leftarrow \mathbf{v}^H\mathbf{w}$
  $\mathbf{w} \leftarrow \mathbf{w} - \alpha_j\mathbf{v}$
  $\beta_j \leftarrow \|\mathbf{w}\|_2$
  Store α_j and β_j.
  if $\beta_j = 0$ **then** break **end if**
  $\mathbf{v}_{\text{prev}} \leftarrow \mathbf{v}$
  $\mathbf{v} \leftarrow \mathbf{w}/\beta_j$
end for
Let k be the final iteration count.
Construct $\mathbf{T}_k \in \mathbb{R}^{k\times k}$ from stored $\{\alpha_j\}_{j=1}^{k}$ and $\{\beta_j\}_{j=1}^{k-1}$.
Compute coefficient vector $\mathbf{y}_k \leftarrow f(\mathbf{T}_k)\mathbf{e}_1\|\mathbf{b}\|_2$.

▷ — *Second Pass: Solution Reconstruction* —

$\beta_0 \leftarrow 0$, $\mathbf{v}_{\text{prev}} \leftarrow \mathbf{0}$, $\mathbf{v} \leftarrow \mathbf{b}/\|\mathbf{b}\|_2$.
$\mathbf{x}_k \leftarrow (\mathbf{y}_k)_1\mathbf{v}$.
for $j = 1, \ldots, k-1$ **do**
  $\mathbf{w} \leftarrow \mathbf{A}\mathbf{v} - \alpha_j\mathbf{v} - \beta_{j-1}\mathbf{v}_{\text{prev}}$ ▷ Use stored α_j, β_{j-1}
  $\mathbf{v}_{\text{next}} \leftarrow \mathbf{w}/\beta_j$ ▷ Use stored β_j
  $\mathbf{x}_k \leftarrow \mathbf{x}_k + (\mathbf{y}_k)_{j+1}\mathbf{v}_{\text{next}}$
  $\mathbf{v}_{\text{prev}} \leftarrow \mathbf{v}$
  $\mathbf{v} \leftarrow \mathbf{v}_{\text{next}}$
end for
return $\mathbf{x}_k$.

$M(n,k)$, as a function of the problem dimension n and the number of iterations k. This total memory can be decomposed into a base cost required for problem representation and an additional cost specific to the algorithm's execution:

$$M(n,k) = M_{\text{base}}(n) + M_{\text{alg}}(n,k). \tag{9}$$

The base component, $M_{\text{base}}(n)$, is common to both algorithms. It includes the memory for storing the sparse matrix $\mathbf{A}$ and a constant number of work vectors. The term $M_{\text{alg}}(n,k)$ represents the additional memory specific to each algorithm's execution strategy, which is the source of the substantial difference between the two methods.

Proposition 3.1 (Memory Complexity)**.** *Let n be the dimension of the matrix $\mathbf{A}$ and k be the number of iterations. The standard one-pass Lanczos algorithm has a memory complexity of $O(nk)$, while the two-pass variant reduces this requirement to $O(n)$.*

Proof. The base memory component, $M_{\text{base}}(n)$, includes storage for the sparse matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ and a constant number of work vectors, c_v, each of dimension n. Let $\text{nnz}(\mathbf{A})$ be the number of non-zero entries in $\mathbf{A}$, and let s_d be the size of a double-precision float. The storage for $\mathbf{A}$ in a Compressed Sparse Column (CSC) format requires approximately $s_d(\text{nnz}(\mathbf{A}) + n) + s_i(\text{nnz}(\mathbf{A}))$, where s_i is the size of an integer index. For the class of sparse problems considered, $\text{nnz}(\mathbf{A}) \propto n$. The base memory cost is therefore strictly linear in n, so $M_{\text{base}}(n) = O(n)$.

The difference between the methods lies in the algorithm-specific term, $M_{\text{alg}}(n,k)$. The standard one-pass algorithm must store the entire basis matrix $\mathbf{V}_k \in \mathbb{R}^{n\times k}$ to synthesize the final solution. This imposes a significant memory overhead:

$$M_{\text{alg, std}}(n,k) = n \cdot k \cdot s_d = O(nk). \tag{10}$$

The total memory is therefore $M_{\text{std}}(n,k) = M_{\text{base}}(n) + M_{\text{alg, std}}(n,k) = O(n) + O(nk) = O(nk)$ [1].

The two-pass algorithm is designed to minimize this term. It stores only the scalar coefficients of $\mathbf{T}_k$, resulting in a negligible cost of $M_{\text{alg, TP}}(k) = O(k)$. The total memory requirement is thus:

$$M_{\text{TP}}(n,k) = M_{\text{base}}(n) + M_{\text{alg, TP}}(k) = O(n) + O(k)$$

Under the common assumption that $n \gg k$, this simplifies to $O(n)$. □

Proposition 3.2 (Computational Complexity)**.** *Let the dominant computational cost of the Lanczos process be the matrix-vector products. For k iterations, the standard one-pass Lanczos algorithm requires k matrix-vector products, whereas the two-pass algorithm requires $2k$.*

Proof. For large, sparse matrices, the matrix-vector product $\mathbf{A}\mathbf{v}_j$ is the most computationally expensive operation per iteration [4, Chap. 6]. The standard one-pass method performs one such product at each of its k iterations, for a total of k products.

The two-pass method executes two distinct passes. The first pass requires k matrix-vector products to generate the coefficients of $\mathbf{T}_k$. To synthesize the solution, the second pass must re-generate the Lanczos basis vectors, which forces the re-computation of the same sequence of k matrix-vector products. The total count is therefore $2k$. □

4 Numerical Experiments

In this section, we present a series of numerical experiments designed to validate the theoretical analysis and to assess the practical performance of the two-pass Lanczos algorithm. Our investigation is structured around three distinct objectives: to provide empirical evidence for the memory-versus-computation trade-off; to evaluate the scalability of both algorithms with respect to the problem dimension; and to conduct an analysis of the numerical stability and the potential for error propagation in the second pass.

4.1 Experimental Setup and Performance Metrics

All experiments were executed on a dual-socket machine running Ubuntu 22.04.2 LTS (GNU/Linux kernel 5.15.0-119-generic). The system is equipped with two Intel(R) Xeon(R) Gold 5318Y CPUs, each operating at a base frequency of 2.10GHz, for a total of 96 logical cores. The algorithms were implemented in Rust and compiled with `rustc` version 1.89.0 using release-level optimizations. All computations were performed using double-precision floating-point arithmetic.

We defined a set of quantitative metrics. The primary measure of computational work is the total number of matrix-vector products, which provides a hardware-independent basis for comparison [4], as it is the dominant operation in Krylov subspace methods for large, sparse matrices. We also report the total wall-clock time as a practical measure of performance. The memory footprint is quantified by the Peak Resident Set Size, corresponding to the `VmPeak` field in the `/proc/self/status` virtual file on our Linux system.

We measure the accuracy of a computed solution x_k by its relative error with respect to a known ground-truth solution x_{true}. We assess the stability of the basis generation by quantifying the loss of orthogonality of the Lanczos basis V_k via the Frobenius norm $\|I - V_k^H V_k\|_F$, a known phenomenon in finite-precision implementations of the process [1]. For the stability experiment, we also measure the numerical deviation, or *drift*, between the standard basis V_k and the re-generated basis V_k', as well as the drift in the scalar recurrence coefficients. We evaluate the ultimate impact by the relative difference between the final solution vectors obtained from the one-pass and two-pass methods.

4.2 Test Problem Generation

We derive our test problems from Karush-Kuhn-Tucker (KKT) systems associated with convex quadratic separable Min-Cost Flow problems. Such systems are sparse, symmetric, and exhibit the block saddle-point structure

$$A = \begin{pmatrix} D & E^T \\ E & 0 \end{pmatrix} \in \mathbb{R}^{n \times n}. \tag{11}$$

We construct the node-arc incidence matrix, E, from network topologies generated by the `netgen` utility, which allows for precise control over problem dimension and sparsity. The diagonal block D is defined as $D = \text{diag}(d_1, \ldots, d_m)$, with its entries drawn from a uniform random distribution, $d_i \sim U[1, C_D]$.

This construction allows us to control key properties of A. First, by setting $d_i \geq 1$, we ensure that D is a symmetric positive definite matrix, as all its eigenvalues are positive. This choice guarantees that the resulting KKT matrix A is indefinite. The indefiniteness can be directly verified by examining the quadratic form $x^T Ax$: vectors of the form $[x_1^T, 0]^T$ with $x_1 \neq 0$ yield positive values since $x_1^T D x_1 > 0$, while other choices of x can lead to non-positive values. Operating on indefinite matrices provides a robust test for the Lanczos algorithm. The process, as formulated for general symmetric matrices, does not require positive definiteness, a property that distinguishes it from methods such as the Conjugate Gradient algorithm [1, Sec. 10.3].

Second, the parameter C_D gives us control over the spectral properties of the problem. For a symmetric positive definite matrix, the 2-norm condition number is the ratio of its largest to its smallest eigenvalue, $\kappa_2(D) = \lambda_{\max}(D)/\lambda_{\min}(D)$ [4, Chap. 2]. Our construction thus yields $\kappa_2(D) \approx C_D$. The spectral properties of the operator are known to fundamentally govern the convergence rate of Krylov subspace methods. The error in the Lanczos approximation is closely tied to the problem of best polynomial approximation on the spectrum of the matrix, a central theme in the analysis of the method [1, Sec. 10.1.5]. By varying C_D, we can therefore generate both well-conditioned and ill-conditioned problem instances. Using a uniform random distribution for the entries of D ensures that our test instances are generic and not biased toward an artificially simple spectral structure.

In order to be able to have the exact computation of solution error, we adopt a known-solution methodology. We first define a ground-truth solution vector x_{true} and then construct the right-hand side vector as $b := Ax_{\text{true}}$.

To assess the robustness of the approach, we conduct experiments using two functions, f, chosen for their distinct analytical properties. The first is the matrix exponential, $f(z) = \exp(z)$. As an entire function, it is a canonical and well-behaved test case for algorithms computing matrix functions [2]. The Lanczos process itself is independent of f. Using a smooth function like the exponential therefore allows a clear analysis of the algorithm's intrinsic properties, such as the stability of the basis re-generation, without confounding factors from singularities in f.

The second function is the inverse, $f(z) = z^{-1}$, which recasts our problem as the solution of the linear system $Ax = b$. We select this function to evaluate the algorithm in the presence of a singularity at the origin. The convergence of Krylov methods for this problem is highly sensitive to the eigenvalues of A near zero, making it an essential and challenging test case for numerical robustness [4, Sec. 6.11.3].

4.3 Experiment 1: Memory and Computation Trade-off

The first experiment was designed to empirically validate the theoretical complexity analysis presented in Section 3.2. The goal was to demonstrate the trade-off between memory consumption and computational workload by comparing the one-pass and two-pass Lanczos algorithms.

Based on our model, we formulate two distinct hypotheses. First, concerning memory usage, we expect the peak resident set size of the standard algorithm to exhibit linear growth with respect to the iteration count k, consistent with its $O(nk)$ complexity derived from storing the basis $\mathbf{V}_k$. Conversely, we expect the two-pass method to maintain a nearly constant memory footprint, consistent with its $O(n)$ complexity. Second, regarding computational time, the model based on floating-point operations predicts that the wall-clock time for the two-pass method should be approximately double that of the one-pass method, due to the doubled number of matrix-vector products. However, this model neglects the cost of memory access. A more realistic hypothesis is that the actual performance will be governed by the relation between the additional arithmetic operations and the algorithms' memory access patterns. For problem sizes where the basis matrix $\mathbf{V}_k$ is too large to fit in processor caches, the cost of memory bandwidth may dominate, reducing

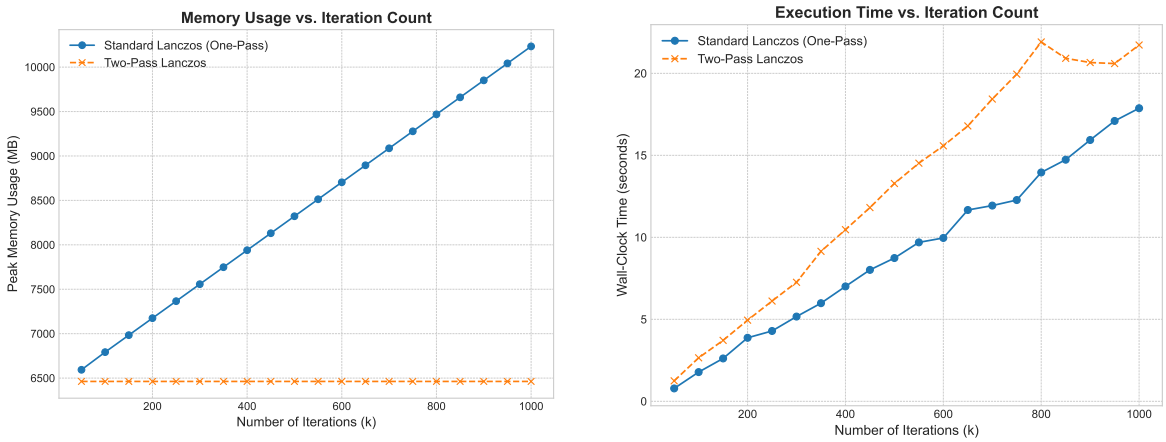
the observable time penalty for the two-pass method.

We conducted this experiment by selecting fixed-size matrices $\mathbf{A}$ of varying dimensions and executing both algorithms, varying the number of iterations k across a predefined range. For each value of k, we recorded the peak RSS and the total wall-clock time.

4.3.1 Results

The memory consumption results, presented in the left panels of Figures 1, 2, and 3, confirm the complexity analysis across all problem scales. The standard one-pass algorithm exhibits a linear growth in memory usage with respect to the iteration count k. This behavior stems directly from the need to store the entire basis matrix $\mathbf{V}_k \in \mathbb{C}^{n\times k}$, a requirement with a cost of $O(nk)$ [1]. In contrast, the two-pass algorithm maintains a constant memory footprint, independent of k. This aligns with its theoretical $O(n)$ memory complexity, as the algorithm only requires simultaneous storage for a small, constant number of n-dimensional vectors.

A closer analysis of Figure 2 (a), however, reveals a subtler dynamic. For a low number of iterations, the memory consumption of the two-pass method is slightly higher than that of the standard method, with a crossover point observed around $k \approx 300$. This phenomenon does not contradict the theoretical model but rather clarifies the constants it abstracts. Our model, $M(n,k) = M_{\text{base}}(n) + M_{\text{alg}}(n,k)$, remains valid, but the base cost, $M_{\text{base}}(n)$, is not identical for the two algorithms. The two-pass method incurs a slightly larger fixed memory overhead. This overhead arises from the need to store a copy of the initial vector $\mathbf{b}$ for the second pass, as well as the arrays for the scalar coefficients $\{\alpha_j\}$, $\{\beta_j\}$, and the solution vector $\mathbf{y}_k$. For small values of k, the storage cost for the basis in the standard method, $O(nk)$, is less than this fixed overhead. As k increases, however, the $O(nk)$ term rapidly dominates.



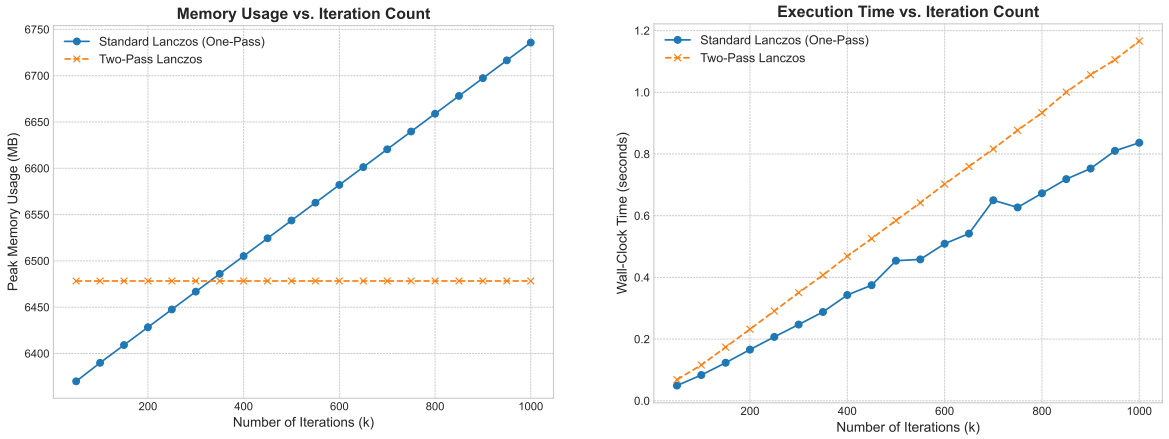
(a) Peak Memory Usage

(b) Wall-Clock Time

Figure 1: Performance results for a large-scale problem instance (500k arcs). The left panel shows peak memory usage, while the right panel shows wall-clock time.

For the large-scale instance with 500k arcs, shown in Figure 1 (right panel), the wall-clock time of

the two-pass method is only marginally greater than that of the one-pass method. This observation appears to contradict the expectation of doubled work [2]. The discrepancy arises because for very large n, the dominant cost shifts from sparse matrix-vector products to dense vector operations during solution reconstruction. Both the final matrix-vector product $\mathbf{x}_k = \mathbf{V}_k \mathbf{y}_k$ in the one-pass method and the incremental accumulation in the two-pass method have a complexity of $O(nk)$. When $\mathbf{V}_k$ is too large to fit in any CPU cache, these operations become limited by main memory bandwidth. This large, common cost term dominates the total execution time for both methods and masks the effect of the doubled sparse matrix-vector products.



(a) Peak Memory Usage

(b) Wall-Clock Time

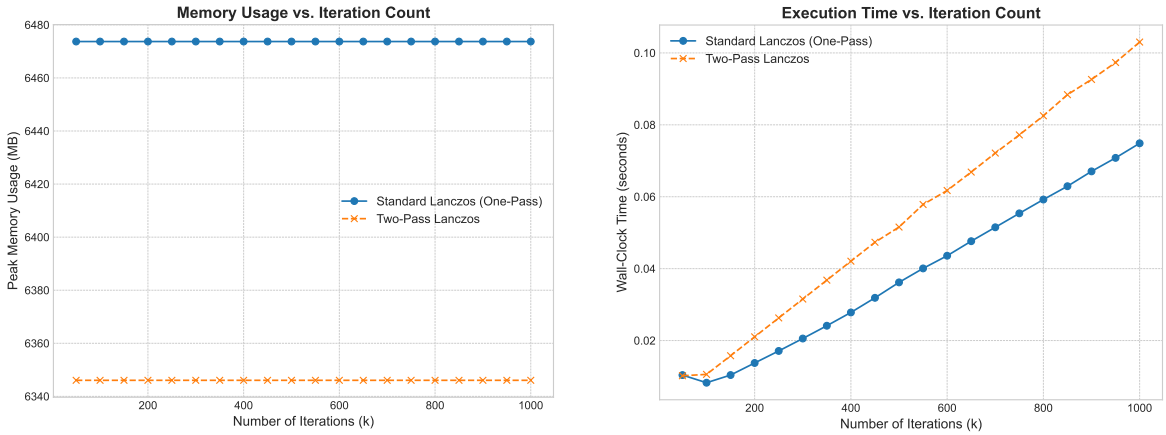
Figure 2: Performance results for a medium-scale problem instance (50k arcs). The left panel shows peak memory usage, while the right panel shows wall-clock time.

As we reduce the problem dimension, this dynamic changes. For the medium-scale instance with 50k arcs, the results in Figure 2 (right panel) show that the two-pass method is consistently faster than the standard one-pass method. This performance reversal originates from the memory access patterns of the algorithms. The standard algorithm's reconstruction step accesses a large contiguous block of memory for $\mathbf{V}_k$, generating an inefficient access pattern with poor data locality that leads to a high rate of cache misses. The two-pass algorithm, by contrast, operates on a small working set of vectors at each step. The processor's cache hierarchy manages this small working set effectively, resulting in a high cache hit rate and superior performance.

This effect becomes even more pronounced for the small-scale instance with 5k arcs, shown in Figure 3 (right panel). Here, the working set of the two-pass method can reside almost entirely within the fastest L1 and L2 caches, maximizing its performance advantage. In this cache-friendly regime, the performance gain from eliminating cache misses far outweighs the cost of the additional matrix-vector products.

4.4 Experiment 2: Scalability

The second experiment assesses how the resource requirements of both algorithms scale with the problem dimension n. For this test, we fix the number of iterations k to a constant value and gen-



(a) Peak Memory Usage

(b) Wall-Clock Time

Figure 3: Performance results for a small-scale problem instance (5k arcs). The left panel shows peak memory usage, while the right panel shows wall-clock time.

erate a family of test matrices of increasing dimension. The network topologies are constructed to maintain a comparable sparsity across all instances. We hypothesize that the memory advantage of the two-pass algorithm will become increasingly significant as n grows, which should manifest as a substantially steeper slope in the memory usage plot for the one-pass method compared to the two-pass method.

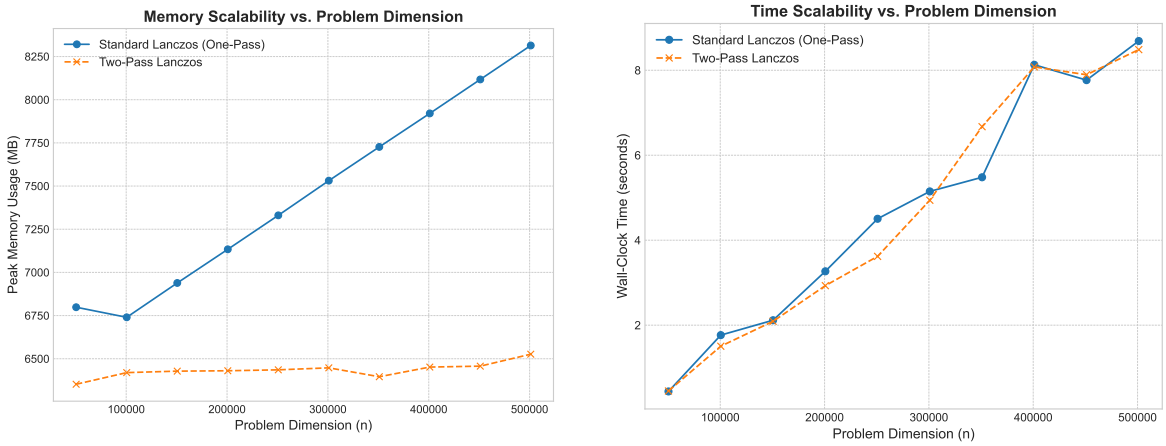
4.4.1 Results

We fixed the number of iterations at $k = 500$ and analyzed the performance of both algorithms as a function of the problem dimension, n. Figure 4 plots the peak memory usage and wall-clock time against n.

The memory consumption profiles in Figure 4 (a) show a stark contrast between the two methods, confirming our hypothesis. The memory usage of both methods grows linearly with n, but the slope of this growth differs significantly. The two-pass method exhibits a gentle slope, corresponding to the linear growth of the base memory requirement, $M_{\text{base}}(n)$. The standard method, in contrast, shows a much steeper slope, reflecting the combined cost of the base memory and the algorithm-specific term for storing the basis, $M_{\text{std}}(n,k) = M_{\text{base}}(n) + M_{\text{alg, std}}(n,k)$.

In Table 1 we provide the raw data underlying Figure 4 . The memory usage of the two-pass method provides a stable empirical baseline for $M_{\text{base}}(n)$. The final column in the table, representing the difference in memory consumption, effectively isolates the cost of storing the basis matrix $\mathbf{V}_k$, which corresponds to the term $M_{\text{alg, std}}(n,k)$. A linear regression performed on this column's data yields an empirical growth rate of approximately 3.97 MB per 1000 units of increase in n, which corresponds to 4065 bytes per unit n. This empirical rate aligns remarkably well with the theoretical rate predicted by our model, which is $k \cdot s_d = 500 \cdot 8 = 4000$ bytes per unit n.

The wall-clock time for both algorithms, shown in Figure 4 (b), scales linearly with the problem



(a) Peak Memory Usage

(b) Wall-Clock Time

Figure 4: Performance and memory scalability with respect to problem dimension n for a fixed number of iterations ($k = 500$).

Dimension (n)	**Standard (MB)**	**Two-Pass (MB)**	**Difference (MB)**
50,365	6798.4	6352.3	446.1
100,516	6739.9	6419.7	320.2
150,632	6938.8	6428.2	510.6
200,730	7133.7	6430.4	703.3
250,816	7330.5	6435.8	894.7
300,894	7531.2	6447.3	1083.9
350,966	7726.6	6395.8	1330.8
401,033	7920.7	6451.8	1468.9
451,095	8117.4	6457.1	1660.3
501,155	8314.1	6526.4	1787.7

Table 1: Peak memory usage (RSS) in megabytes for a fixed number of iterations ($k = 500$). The difference isolates the memory cost of the basis matrix $\mathbf{V}_k$ in the standard algorithm.

dimension n. This is consistent with the theoretical cost, which is dominated by k iterations of operations (such as sparse matrix-vector products) whose complexity is linear in n. With k held constant, the total time complexity is $O(nk)$, resulting in the observed linear scaling. The execution times for both methods remain close, which is consistent with the findings from 4.3 for large-scale problems where memory bandwidth becomes a significant factor in overall performance.

4.5 Experiment 3: Numerical Accuracy and Stability Analysis

In our final experiment, we investigate the numerical accuracy and stability of the two-pass Lanczos method. Our primary objective is to empirically verify that the solution computed via basis regeneration is numerically equivalent to the solution from the standard one-pass algorithm across a range of distinct scenarios. To achieve this, we move beyond relative performance metrics and

measure the absolute error of each method against a known, analytically computed ground-truth solution. This approach allows us to answer two fundamental questions: what is the absolute accuracy of the Lanczos approximation for a given problem, and does the two-pass method maintain this accuracy, or does the process of regenerating the basis vectors introduce a significant numerical degradation?

To isolate the factors governing algorithmic performance (namely, the analytic properties of the function f and the spectral properties of the operator $\mathbf{A}$), we designed four distinct experimental scenarios. We build these scenarios upon synthetic diagonal test matrices, allowing us to exercise precise control over the problem's characteristics [4]. For a diagonal matrix $\mathbf{A} = \text{diag}(\lambda_1, \ldots, \lambda_n)$, we can compute the ground-truth solution $\mathbf{x}_{\text{true}} = f(\mathbf{A})\mathbf{b}$ to machine precision via the element-wise product $(\mathbf{x}_{\text{true}})_i = f(\lambda_i)b_i$. This provides a verifiable, reference against which we measure the relative error of the computed solutions $\mathbf{x}_k$ (one-pass) and $\mathbf{x}'_k$ (two-pass). We also track the direct deviation $\|\mathbf{x}_k - \mathbf{x}'_k\|_2$ to quantify their numerical equivalence.

Exponential on a Well-Conditioned Spectrum. We first test $f(z) = \exp(z)$ on a matrix $\mathbf{A}$ whose spectrum is a compact interval on the negative real axis, e.g., $\sigma(\mathbf{A}) \subset [-C, -c]$ with $C, c > 0$. As an entire function, the exponential is analytic everywhere. The theory of polynomial approximation predicts that for such functions on a compact set, the error of the Lanczos approximation converges super-linearly to zero [2]. This scenario serves as a baseline control experiment. In the absence of any problem-induced instabilities, we hypothesize that both Lanczos variants will converge rapidly to the ground truth and that their solutions will be numerically indistinguishable.

Inverse on a Well-Conditioned Spectrum. Next, we test $f(z) = z^{-1}$ on a symmetric positive definite matrix $\mathbf{A}$ whose spectrum is strictly bounded away from zero, i.e., $\sigma(\mathbf{A}) \subset [c, C]$ with $0 < c$. This recasts the problem as the solution of a well-conditioned linear system. The classical theory of the conjugate gradient method, which is algebraically equivalent to Lanczos for this problem, predicts a monotonic error decrease, with a convergence rate governed by the condition number $\kappa(\mathbf{A})$ [4]. This experiment validates the standard convergence behaviour and verifies the numerical equivalence of the two methods for a function with a singularity, provided the spectrum is kept in a safe region.

Exponential on an Ill-Conditioned Spectrum. We then test $f(z) = \exp(z)$ on a matrix with a very wide spectrum, e.g., $\sigma(\mathbf{A}) \subset [-C, -c]$ where $C \gg c$. While the problem remains well-posed, approximating $\exp(z)$ with a single polynomial over a large interval is known to be difficult, requiring a high polynomial degree k. This scenario tests the robustness of the methods under conditions of slow convergence. We hypothesize that both algorithms will exhibit identical, albeit slower, convergence, demonstrating that the two-pass approach remains stable even over a large number of iterations.

Inverse on an Ill-Conditioned Spectrum. Finally, we test $f(z) = z^{-1}$ on a symmetric, indefinite matrix $\mathbf{A}$ constructed to be nearly singular, with an eigenvalue λ_j satisfying $|\lambda_j| \ll 1$. This provides

a profound stress test for the algorithm. The Lanczos process is known to approximate extremal eigenvalues of $\mathbf{A}$ well [1]. Consequently, the projected matrix $\mathbf{T}_k$ is expected to become severely ill-conditioned as one of its eigenvalues converges to λ_j. The computation of $\mathbf{y}_k = \mathbf{T}_k^{-1}\mathbf{e}_1\|\mathbf{b}\|_2$ is therefore expected to be numerically unstable. However, for $f(z) = z^{-1}$, the Lanczos method is a finite-termination method in exact arithmetic [4]. This experiment investigates these competing effects. We hypothesize that both methods will exhibit identical behaviour, characterized by a phase of instability followed by eventual convergence, thus demonstrating that the observed instability is an intrinsic property of the projected problem, not a flaw introduced by the two-pass reconstruction.

4.5.1 Results

The results of our accuracy and stability analysis are presented in Figures 5 through 8. Each figure provides a view of a single experimental scenario, displaying both the convergence of each method towards the ground-truth solution and the direct numerical deviation between the two methods.

Well-Conditioned Scenarios. Figure 5 displays the results for the matrix exponential on a well-conditioned spectrum. As predicted by the theory of polynomial approximation for entire functions on a compact set [2], the relative error for both methods converges super-linearly, reaching the level of machine precision in fewer than 30 iterations. The lower panel shows that the deviation between the two solutions, $\|\mathbf{x}_k - \mathbf{x}'_k\|_2$, remains stable and of the order of machine epsilon (10^{-15}), confirming that the basis regeneration process is intrinsically stable.

Figure 6 presents the case for the matrix inverse on a well-conditioned, positive definite matrix. The convergence is linear, appearing as a straight line on the log-linear scale, and the error decreases monotonically until it stagnates at the level of machine precision. This behaviour is consistent with the classical convergence theory for the Lanczos method applied to linear systems [4]. Again, the two methods produce nearly identical error curves, and their direct deviation is negligible, reinforcing the conclusion that the two-pass approach does not compromise numerical accuracy on well-posed problems.

Ill-Conditioned Scenarios. The experiments on ill-conditioned spectra reveal a more complex dynamic. In Figure 7, for the matrix exponential on a wide spectrum, we observe a significantly slower convergence rate compared to the well-conditioned case. This is an expected consequence of the difficulty of approximating e^z with a low-degree polynomial over a large interval. Nonetheless, the convergence remains monotonic, and the two methods continue to produce numerically close results. This demonstrates that the stability of the two-pass method holds even when a large number of iterations are required.

Figure 8 illustrates the most challenging scenario: the matrix inverse on a nearly singular, indefinite matrix. The error plot shows the three phases predicted by the theory. Initially, for small k, the error stagnates as the Krylov subspace fails to capture the problematic near-null space of $\mathbf{A}$. Subsequently, as the Lanczos process begins to approximate the near-zero eigenvalue, the projected

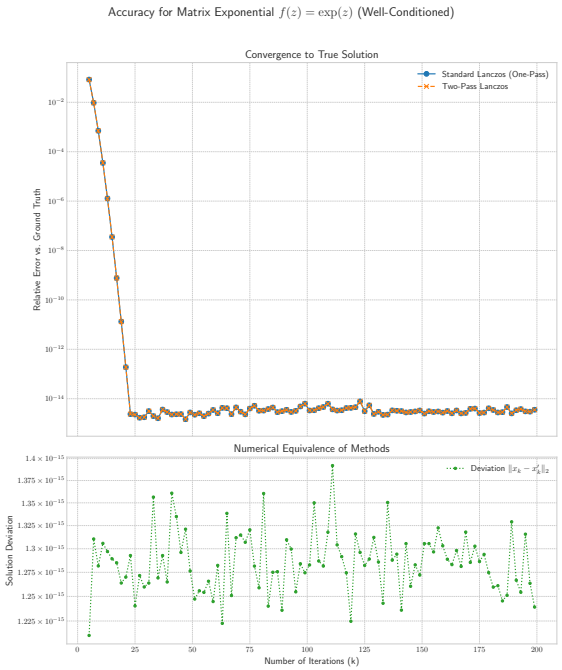


Figure 5: Accuracy for $f(z) = \exp(z)$ on a well-conditioned matrix.

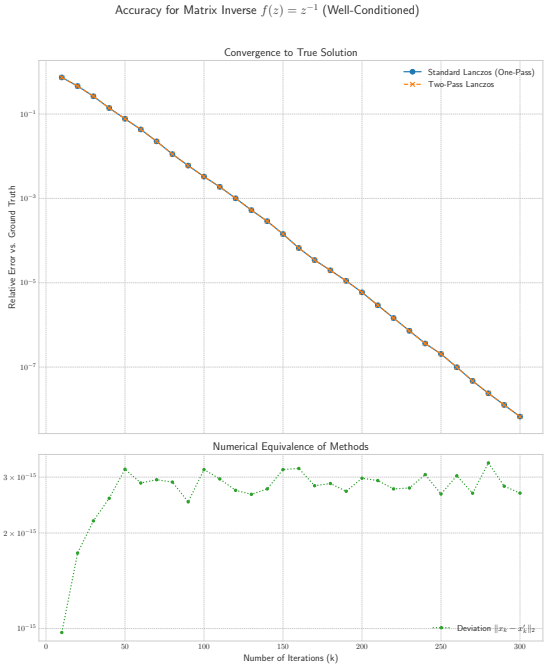


Figure 6: Accuracy for $f(z) = z^{-1}$ on a well-conditioned, positive definite matrix.

matrix $\mathbf{T}_k$ becomes severely ill-conditioned. This induces a period of erratic, non-monotonic error behaviour, as the inversion of $\mathbf{T}_k$ becomes numerically unstable [1]. Critically, this instability affects both methods identically. Finally, as k becomes sufficiently large, the finite-termination property of the Lanczos method for linear systems comes into effect [4]. The Krylov subspace becomes rich enough to construct a highly accurate polynomial approximant for z^{-1}, and the error converges abruptly to a high level of precision. Throughout this entire process, from stagnation to instability to convergence, the solutions from the one-pass and two-pass methods remain numerically close.

4.6 Experiment 4: Analysis of Basis Orthogonality

The previous experiment established the numerical equivalence of the final solution vectors. We now complete the analysis by investigating the stability of the basis regeneration process itself. This experiment is designed to measure the loss of orthogonality, a well-known phenomenon in finite-precision implementations of the Lanczos process [1]. While the three-term recurrence guarantees orthogonality in exact arithmetic, rounding errors accumulate and cause a gradual degradation of this property with respect to the more distant vectors in the sequence [2].

We use the same four experimental scenarios defined in 4.5 to study how orthogonality loss is affected by different spectral properties. We quantify the degradation using the Frobenius norm of the deviation of $\mathbf{V}_k^H \mathbf{V}_k$ from the identity matrix: $\|\mathbf{I} - \mathbf{V}_k^H \mathbf{V}_k\|_F$ [1]. This metric is computed for both the standard basis $\mathbf{V}_k$, stored during the one-pass algorithm, and the regenerated basis $\mathbf{V}'_k$,

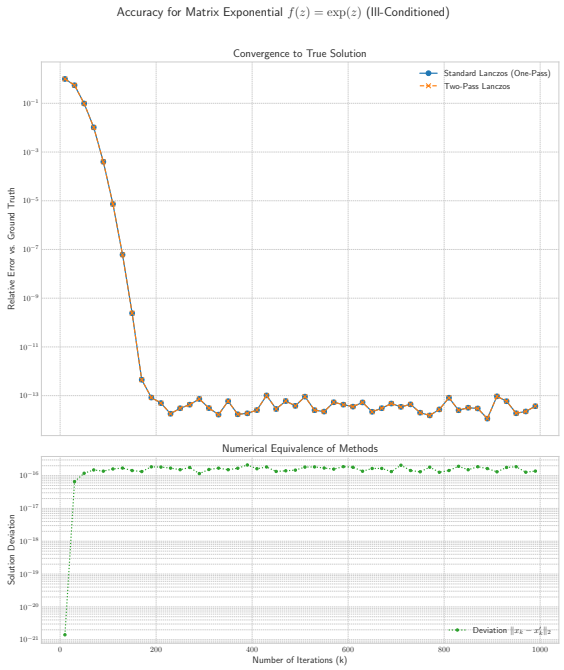


Figure 7: Accuracy for $f(z) = \exp(z)$ on a matrix with a wide spectrum.

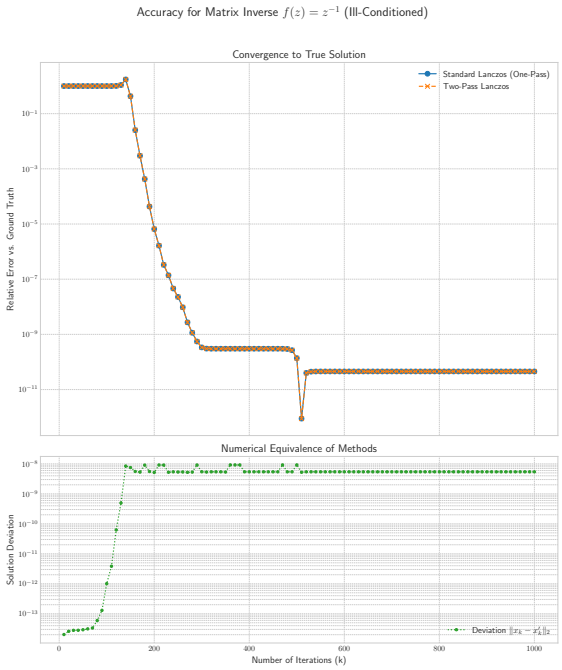


Figure 8: Accuracy for $f(z) = z^{-1}$ on a nearly singular, indefinite matrix.

recomputed during the second pass.

4.7 Results

Figures 9, 10, 11, and 12 present the results of this analysis. The data confirms that the loss of orthogonality increases with the number of iterations, a behavior consistent with the theory of the Lanczos process in finite-precision arithmetic [4]. The degradation is visibly more severe for the ill-conditioned problems (Figures 11 and 12), a phenomenon linked to the convergence of Ritz values to the eigenvalues of $\mathbf{A}$ [1].

The most significant result is the numerical identity of the orthogonality loss profiles for the standard and regenerated bases, observed across all test scenarios. This outcome is a direct consequence of the deterministic nature of the algorithm when executed in finite-precision arithmetic. The second pass reconstructs the basis using the recurrence from Equation (7), which is supplied with the exact scalar coefficients $\{\alpha_j, \beta_j\}$ stored during the first pass. Since the sequence of floating-point operations and their respective operands is identical in both the original generation and the regeneration, the resulting basis matrices $\mathbf{V}_k$ and $\mathbf{V}'_k$ are computationally indistinguishable. The two-pass method, therefore, introduces no additional numerical degradation; it precisely replicates the propagation of rounding errors inherent to the one-pass process.

The severe loss of orthogonality observed in Figures 12 and 11 for the ill-conditioned inverse problem coincides with the period of erratic error behavior in the corresponding solution accuracy

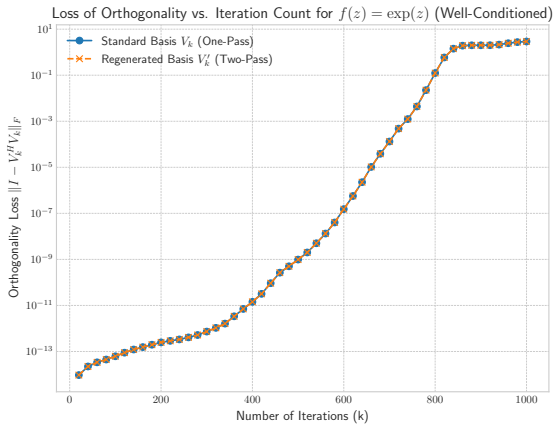


Figure 9: Orthogonality loss for $f(z) = \exp(z)$ on well-conditioned matrix

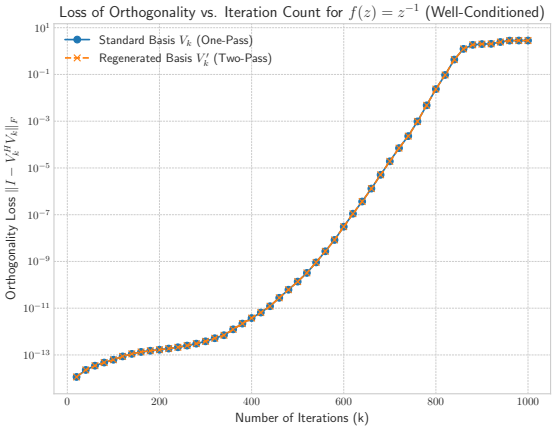


Figure 10: Orthogonality loss for $f(z) = z^{-1}$ on well-conditioned matrix

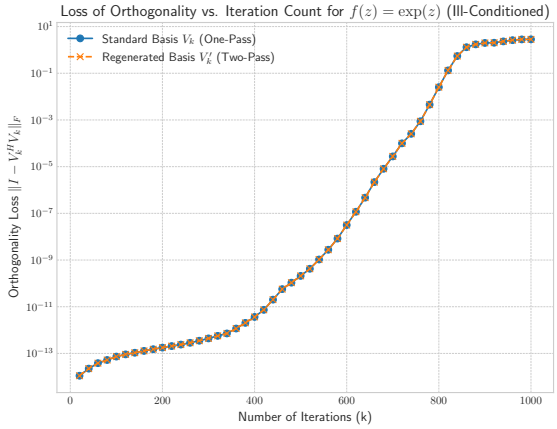


Figure 11: Orthogonality loss for $f(z) = \exp(z)$ on ill-conditioned matrix

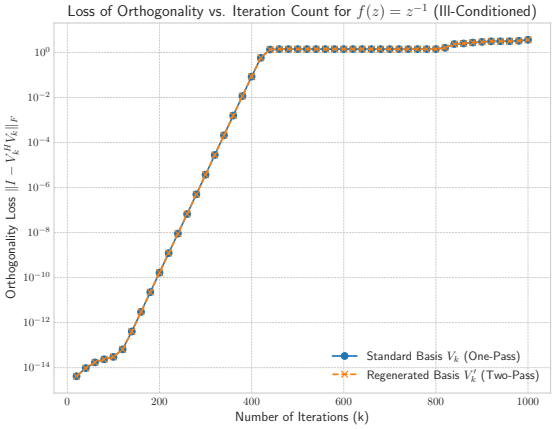


Figure 12: Orthogonality loss for $f(z) = z^{-1}$ on ill-conditioned matrix

plot (Figure 8). This confirms that the instability in the solution is an intrinsic part of the Lanczos method applied to a nearly singular problem, which is reproduced by the two-pass algorithm.

References

[1] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[2] Andreas Frommer and Valeria Simoncini. Matrix functions. In *Model order reduction: theory, research aspects and applications*, pages 275–303. Springer, 2008.

[3] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of research of the National Bureau of Standards*, 45(4):255–282, 1950.

[4] Yousef Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2 edition, 2003.

[5] Angelo A Casulli and Igor Simunec. A low-memory lanczos method with rational krylov compression for matrix functions. *SIAM Journal on Scientific Computing*, 47(3):A1358–A1382, 2025.

[6] Artan Boriçi. Fast methods for computing the neuberger operator. In *Numerical Challenges in Lattice Quantum Chromodynamics: Joint Interdisciplinary Workshop of John von Neumann Institute for Computing, Jülich, and Institute of Applied Computer Science, Wuppertal University, August 1999*, pages 40–47. Springer, 2000.

[7] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, 2008.