AUT Journal of Mathematics and Computing

AUT J. Math. Comput., 7(3) (2026) 303-320
https://doi.org/10.22060/AJMC.2025.23508.1262

Original Article

Computing the matrix square root: A problem-solving approach using Mathematica and Pólya's strategies

Mandana Moccari*

Department of Mathematics, Ha. C., Islamic Azad University, Hamedan, Iran

ABSTRACT: This paper introduces a novel method for solving the matrix equation $X^2 - A = 0$ by computing matrix square roots. Inspired by George Pólya's structured problem-solving strategies and leveraging Wolfram Mathematica, the approach offers a systematic, efficient, and clear solution to the problem. The method extends the computation of matrix square roots to large matrices and those with complex eigenvalues, significantly broadening its applicability in diverse fields, including control theory, quantum physics, and signal processing. The approach is demonstrated through comprehensive examples and original Mathematica code, providing a practical toolkit for solving similar mathematical challenges. The method is designed to be both intuitive and versatile, making it a valuable resource for educators, students, and researchers engaged in advanced mathematical problem-solving.

Review History:

Received:05 September 2024
Revised:19 February 2025
Accepted:25 February 2025
Available Online:01 July 2026

Keywords:

Square root
Matrix equation
Mathematica software
George Polya's problem-solving

MSC (2020):

65F45; 97N60; 97N40

1. Introduction

In this paper, a method for solving the matrix equation $X^2 - A = 0$ is presented by integrating problem-solving strategies, computational software, and illustrative examples. As technology advances, the need for fast, simple, and reliable solutions to complex mathematical problems has become increasingly important. Our approach combines George Pólya's problem-solving methods with the use of powerful computational tools like Wolfram Mathematica, which enhances both understanding and computation [4, 15, 16].

The specific focus of our study is the computation of the square root of a matrix, a topic with widespread applications in fields such as control theory, nuclear magnetic resonance, and lattice quantum chromodynamics [10, 14]. For example, the Riccati equation, a fundamental equation in control theory, requires the computation of the matrix sign function, which is closely related to the matrix square root [9, 23]. Computing matrix functions is essential in both science and engineering. The papers [5, 6] introduce efficient numerical techniques for solving

* *Corresponding author.*
E-mail address: m_moccari@iau.ac.ir (M. Moccari)

matrix equations. These methods enable the determination of matrix roots and other functions, with practical applications in areas like control theory.

In this paper, a variety of methods for computing the matrix square root are explored. Using these approaches, all roots of square matrices have been successfully derived, including both primary and non-primary roots. The combination of these methods has made the computation of the square root for large matrices significantly more efficient. To demonstrate the effectiveness of these techniques, a set of precise numerical examples is provided, clearly illustrating the step-by-step execution of the algorithms and their outcomes. These methods prove to be reliable not only in specialized cases but also when dealing with complex and large matrices, yielding accurate and trustworthy results.

As part of our approach to finding the second root, the Sylvester equation [7] at specific stages is solved, which is explained in detail in this paper. To support the computation, step-by-step Wolfram Mathematica codes, including some novel implementations is provided.

This problem is addressed by following Pólya's four-step problem-solving process: understanding the problem, devising a plan, executing the plan, and reviewing the solution [22]. Detailed Wolfram Mathematica codes are provided to compute the square root of matrices, including some novel codes that have not been published before. Mathematica serves as a crucial tool in the method, aiding in the step-by-step computation process [25]. To validate the approach, a statistical analysis is conducted to compare the effectiveness of the proposed method, employing tests such as paired sample t-test.

The approach presented is not only novel but also versatile, with the potential for generalization to more complex mathematical challenges. Valuable insights are offered for educators and students, providing a practical framework for applying problem-solving strategies to real-world problems.

The structure of this paper is as follows: In Section 2, which corresponds to the first step of George Pólya's four-step approach, the problem is introduced, and various types of matrix square roots are defined, along with the necessary terminologies. The following section, reflecting the second step of Pólya's method, presents a road map for solving the problem, where matrices are decomposed and simplified into more manageable forms. Section 4 is devoted to the computation of matrix square roots, employing both analytical methods, such as Jordan and Schur decompositions, and iterative methods. In Section 5, a comparison of the methods is provided, highlighting their strengths and limitations. The final section presents a statistical analysis of the methods. Throughout the paper, numerical examples and step-by-step solutions are demonstrated using Mathematica to ensure a clear and comprehensive understanding of the techniques.

2. The First Step of George Pólya's Problem-Solving: Understanding the Problem

In this section, mirroring the first step of George Pólya's problem-solving method, the objective is to identify the unknowns of the problem. Numerous studies have explored the properties of matrices, eigenvalues, and matrix operations, providing fundamental theories and diverse applications in numerical problem-solving [8, 24, 19]. The instructor aims to spark students' interest through elementary questions, encouraging them to discover the unknowns on their own. Initially, simple cases are implemented, and then more complex cases are introduced, allowing students to encounter and address potential issues independently. At least one computer with Mathematica installed should be available for every two students.

Next, the instructor clearly defines some preliminary concepts using neat examples and Mathematica codes. By utilizing the software, students can observe the results of the problem without requiring prior knowledge of algebraic proofs.

2.1. Implementation of the First Step

This step is executed for computing the square root of a matrix. The following questions are proposed to help students gain a deeper understanding:

- What do you know about the square root? Who can compute the square root of a real or complex number "a" using Mathematica?
- Do you know an equation whose roots are the square roots of "a"?
- Is it possible to replace a scalar "a" with a matrix "A" in the suggested equation?
- What are the differences between the square root of a scalar value and that of a matrix?
- What is the definition of a matrix equation?

The instructor then suggests that students run the following Mathematica code:

```
In[1]:= x /. Solve[x^2 - 4 == 0, x]
Out[1]= {-2, 2}
```

2.1.1. Matrix Equation

In this section, the concept of matrix functions is introduced. Consider a scalar function f and a matrix $A \in \mathbb{C}^{n\times n}$; $f(A)$ is then a matrix of the same dimensions as A. When $f(t)$ is a polynomial or rational function with scalar coefficients and scalar argument t, $f(A)$ is obtained by substituting A for t [10]. For example,

$$\text{if } f(t) = 1 + 5t + 2t^3, \text{ then } f(A) = I + 5A + 2A^3, \tag{1}$$

where I is the identity matrix with the same dimension as A.

Example 2.1.
```
In[1]:= A = {{1, 2}, {1, 3}};
f[A_] = IdentityMatrix[Length[A]] + 5 A + 2 MatrixPower[A, 3]
Out[2]= {{29, 72}, {36, 101}}
```

Matrix functions possess useful properties that aid in solving problems. For more details, refer to [12]. Now, some examples to clarify these properties are discussed. Each of them can be verified using simple Mathematica code. Below, they are demonstrated with numerical examples.

Suppose $f(t) = t^2$ and the matrices A and X are given by

$$A = \begin{pmatrix} 1 & 2 \\ 4 & -1 \end{pmatrix}, \quad X = \begin{pmatrix} 1 & 4 \\ 5 & -1 \end{pmatrix}. \tag{2}$$

By the definition of matrix function, $f(A) = A^2$ is considered. Therefore, it is obtained:

```
In[1]:= f[t_] = t.t;
In[2]:= f[A]
Out[2]= {{9, 0}, {0, 9}}

(*X is an arbitrary matrix with the same dimension*)
In[3]:= f[A].X == X.f[A]
Out[3]= True

In[4]:= f[Transpose[A]] == Transpose[f[A]]
Out[4]= True

In[5]:= f[X.A.Inverse[X]] == X.f[A].Inverse[X]
Out[5]= True

In[6]:= Eigenvalues[A]
Out[6]= {-3, 3}

In[7]:= Eigenvalues[f[A]]
Out[7]= {9, 9}
```

2.2. Existence of the Square Root of a Matrix

A fundamental question is whether every matrix has a square root. In the context of real numbers, only non-negative numbers have square roots. This naturally leads to an analogous question for matrices: under what conditions does a square root exist for a given matrix? To address this, relevant definitions are first established.

A solution to the equation $X^2 = A$ is called a square root of A. Among these, the **primary root** of A is defined as a primary matrix function of A, derived according to the standard definition of $f(A)$. Conversely, any root not conforming to this definition is referred to as a **nonprimary root**. Nonprimary roots arise when identical eigenvalues in the structure of the matrix are associated with different values of f. Specifically, this situation occurs when a matrix contains certain structural components called *Jordan blocks*, which are defined later. Nonprimary roots result from selecting different branches of f for Jordan blocks sharing the same eigenvalue. This phenomenon arises when f is multivalued, and the matrix is *derogatory*, meaning it has repeated eigenvalues, with some appearing in multiple Jordan blocks [12].

Square roots of a matrix can be classified into two primary categories:

- **Principal Square Root**: The principal square root of a matrix A, denoted by $A^{1/2}$, is the unique matrix B such that $B^2 = A$, all eigenvalues of B are positive, and B is a primary matrix function of A. The principal square root exists if and only if A is symmetric and positive definite. (A matrix A is positive definite if and only if it is symmetric and all eigenvalues of A are strictly positive.)
- **Nonprincipal Square Root**: Nonprincipal square roots, denoted by $\sqrt{A}$, refer to other solutions to $X^2 = A$ that may exist when A does not satisfy the conditions for a principal square root. These roots are not necessarily symmetric, positive semidefinite, or unique.

The following theorem provides conditions for the existence and count of real primary square roots:

Theorem 2.1. *Let $A \in \mathbb{R}^{n\times n}$ be a nonsingular matrix. If A has at least one negative real eigenvalue, then it does not have any real square root that is a primary function of A. Otherwise, if all eigenvalues of A are positive real numbers or appear in complex conjugate pairs, the number of real primary square roots of A is 2^{r+c}, where r is the number of distinct real eigenvalues, and c is the number of distinct pairs of complex conjugate eigenvalues.*

Proof. For further details, see Theorem 6.6 in [12]. □

In general, the existence of a square root for a matrix $A \in \mathbb{C}^{n\times n}$, whether singular or nonsingular, can be determined using specific algorithms and results provided in [12, 13, 20].
Step 1: Calculate various powers of the matrix A,

$$A^0 = I, \quad A^i = A^{i-1}A, \quad i = 1, 2, \dots.$$

Step 2: Create an ascending sequence $\{t_i\}$ as follows:

$$t_i = d_i - d_{i-1}, \quad i = 1, 2, \dots$$

where

$$d_i = \dim(\text{null}(A^i)), \quad \text{and} \quad \text{null}(A^i) = \{X \in \mathbb{C}^n \mid A^i X = 0\}.$$

Step 3: If the sequence $\{t_i\}$ contains no two equal odd integer terms, then the square root of the matrix A exists. Note that after a finite number of computations, the terms of the sequence $\{d_i\}$ become fixed and do not change.

This algorithm is illustrated and clarified using Mathematica and numerical examples for educational purposes and simplicity.

Example 2.2. *Let J be the following matrix:*

$$J = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

The terms of the sequence $\{t_i\}$ are computed as described in Step 2. Mathematica can be used for these calculations:

```
(* NullSpace[m] gives a list of vectors that forms a basis for
the null space of the matrix m *)
In[1]:= d0 = Dimensions[NullSpace[IdentityMatrix[3]], 1];
d1 = Dimensions[NullSpace[J], 1];
d2 = Dimensions[NullSpace[J.J], 1];
d3 = Dimensions[NullSpace[J.J.J], 1];
d4 = Dimensions[NullSpace[J.J.J.J], 1];
d = {d0, d1, d2, d3, d4}
t = {d1 - d0, d2 - d1, d3 - d2, d4 - d3}
Out[6]= {{0}, {2}, {3}, {3}, {3}}
Out[7]= {{2}, {1}, {0}, {0}}
```

Since the sequence obtained in `Out[7]` *contains only one odd number, the matrix J has a square root.*

Example 2.3. *Let H be the following matrix:*

$$H = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Again, the terms of the sequence $\{t_i\}$ are computed for H using Mathematica:

```
In[8]:= d0 = Dimensions[NullSpace[IdentityMatrix[3]], 1];
d1 = Dimensions[NullSpace[H], 1];
d2 = Dimensions[NullSpace[H.H], 1];
d3 = Dimensions[NullSpace[H.H.H], 1];
d4 = Dimensions[NullSpace[H.H.H.H], 1];
d = {d0, d1, d2, d3, d4}
t = {d1 - d0, d2 - d1, d3 - d2, d4 - d3}
Out[13]= {{0}, {1}, {2}, {3}, {3}}
Out[14]= {{1}, {1}, {1}, {0}}
```

Since the sequence obtained in `Out[14]` *contains repeated odd numbers, the matrix H does not have a square root.*

2.3. What is the Concept of the Matrix Norm?

To define the norm for a matrix A, the concept of a matrix norm is used on $\mathbb{C}^{m\times n}$. A matrix norm is a function $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ that satisfies the following conditions:

- $\|A\| \geq 0$ and $\|A\| = 0 \Leftrightarrow A = 0$.
- $\|\alpha A\| = |\alpha|\|A\|$ for all $\alpha \in \mathbb{C}$ and $A \in \mathbb{C}^{m\times n}$.
- $\|A + B\| \leq \|A\| + \|B\|$ for all $A, B \in \mathbb{C}^{m\times n}$.

For a given vector norm on $\mathbb{C}^n$, the matrix norm for a matrix A is defined as

$$\|A\| = \max_{x\neq 0} \frac{\|Ax\|}{\|x\|}. \tag{3}$$

If the vector p-norm is defined by $\|x\|_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$ for $1 \leq p < \infty$, and $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$, then the matrix p-norm is given by

$$\|A\|_p = \max_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p}.$$

It is worth noting that $\|A\|_\infty = \|A^*\|_1$, where A^* denotes the conjugate transpose of A. It is demonstrated how to compute the norm of a given matrix B using Mathematica.

Example 2.4. *Consider the following matrix:*

$$B = \begin{pmatrix} 1 & 1 & 0 \\ \sqrt{2} & 3 & -2 \\ 4 & 3 & 2 \end{pmatrix}.$$

Using Mathematica, the norms are computed as follows:

```
In[1]:= Norm[B, 1]
Out[1]= 7

In[2]:= Norm[B, 2] // N
Out[2]= 6.05926

(* The infinity-norm for a matrix is equal to
the 1-norm for the conjugate transpose of the matrix *)
In[3]:= Norm[ConjugateTranspose[B], 1]
Out[3]= 9
```

3. The Second Step of George Polya's Problem-Solving: Devising a Plan

In the previous section, some basic concepts for computing the square root of a given matrix A were introduced. In this section, a plan for computing the square root of a matrix will be devised. In the following, several methods for finding the square roots of a matrix are presented. Some of these methods can identify all the square roots of a matrix, while others are more efficient in determining the principal square root. Each method is discussed in terms of the conditions under which it performs best and can effectively reach the desired solution.

3.1. Do Similarity Transformations Help in Simplifying the Calculation of the Square Root?

The first step in computing the matrix square root is to simplify the matrix, a process similar to matrix decomposition. In the context of scalars, factorization into divisors greatly aids in calculating roots. For matrices, the focus is on two useful decompositions: Jordan decomposition and Schur decomposition. These are explained as follows [10].

3.1.1. Jordan Decomposition

Any matrix $A \in \mathbb{C}^{n\times n}$ can be expressed in Jordan canonical form as

$$W^{-1}AW = J = \operatorname{diag}(J_1, J_2, \ldots, J_p), \tag{4}$$

where

$$J_k = J_k(\lambda_k) = \begin{pmatrix} \lambda_k & 1 & & \\ & \lambda_k & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_k \end{pmatrix}, \tag{5}$$

with $J_k \in \mathbb{C}^{m_k\times m_k}$ and $m_1 + m_2 + \cdots + m_p = n$. Here, W is a nonsingular matrix (not unique), and $\lambda_1, \lambda_2, \ldots, \lambda_p$ are the distinct eigenvalues of the matrix A. The Jordan canonical form of a given matrix A can be easily obtained using Mathematica. See the following example.

Example 3.1. *Let A be the following matrix:*

$$A = \begin{pmatrix} 3 & 13 & -5 & 3 \\ \frac{1}{3} & \frac{13}{3} & -1 & \frac{2}{3} \\ \frac{8}{9} & \frac{140}{9} & -\frac{13}{3} & \frac{40}{9} \\ 0 & 10 & -4 & 5 \end{pmatrix}. \tag{6}$$

The Jordan decomposition of matrix A can be obtained using Mathematica:

```
In[1]:= Eigenvalues[A]
Out[1]= {3, 2, 2, 1}

In[2]:= {W, J} = JordanDecomposition[A];
```

Thus,

$$W = \begin{pmatrix} 1 & \frac{1}{2} & 0 & -4 \\ 0 & \frac{1}{2} & -\frac{3}{2} & -1 \\ 1 & 2 & -4 & -2 \\ 1 & 1 & 0 & 1 \end{pmatrix}, \quad J = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix}, \tag{7}$$

where the Jordan blocks are

$$J_1 = \begin{pmatrix} 1 \end{pmatrix} \in \mathbb{C}^{1\times 1}, \quad J_2 = \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix} \in \mathbb{C}^{2\times 2}, \quad J_3 = \begin{pmatrix} 3 \end{pmatrix} \in \mathbb{C}^{1\times 1}. \tag{8}$$

3.1.2. Schur Decomposition

Using Schur decomposition, a matrix $A \in \mathbb{C}^{n\times n}$ can be expressed as

$$Q^*AQ = T, \tag{9}$$

where $Q \in \mathbb{C}^{n\times n}$ is a unitary matrix (i.e., $Q^* = Q^{-1}$) and $T \in \mathbb{C}^{n\times n}$ is an upper triangular matrix. The eigenvalues of A appear on the diagonal of T.

If A is a real matrix, then instead of Q^*, we can use Q^T, where Q^T is also an orthogonal matrix (i.e., $Q^T = Q^{-1}$). If A has complex eigenvalues, the matrix T is a Hessenberg matrix, which is a quasi-upper triangular matrix where the entries below the main diagonal are not necessarily zero. This Hessenberg matrix can be transformed into an upper triangular matrix, with the complex eigenvalues placed on the main diagonal, using the following procedure:

```
{Q, T} = SchurDecomposition[A, RealBlockDiagonalForm -> False]
```

So, in Schur decomposition, an upper triangular matrix can always be obtained, regardless of whether the eigenvalues are real or complex, with the eigenvalues placed on the main diagonal.

Example 3.2. *Let A be the following matrix:*

$$A = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 2 & 0 & -2 \\ -2 & 0 & 4 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{10}$$

Using Mathematica, the Schur decomposition can be computed as follows:

```
In[1]:= Eigenvalues[A]
Out[1]= {3, 2, 2, 1}

In[2]:= {Q, T} = SchurDecomposition[N[A]];
```

where

$$Q = \begin{pmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad T = \begin{pmatrix} 2 & 3 & 0 & -\sqrt{2} \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{11}$$

Example 3.3. *The following matrix with complex eigenvalues is considered:*

$$A = \begin{pmatrix} 1 & 3 & 4 \\ 1 & 5 & -2 \\ 6 & 4 & 5 \end{pmatrix}, \tag{12}$$

where

```
In[1]:= Eigenvalues[A]//N
Out[1]= {6.7367 + 1.98542 i, 6.7367 - 1.98542 i,-2.47339}

In[2]:= {Q, T}=SchurDecomposition[N[A],RealBlockDiagonalForm -> False];
```

Therefore, Q and T are computed as follows:

$$Q = \begin{pmatrix} 0.412878 - 0.703746i & -0.422062 - 0.0725581i & -0.387802 - 0.0219471i \\ -0.125919 + 0.214628i & 0.276326 - 0.33388i & -0.782798 + 0.370775i \\ -0.264083 + 0.450126i & -0.791627 + 0.0457593i & -0.233055 - 0.211105i \end{pmatrix}, \tag{13}$$

$$T = \begin{pmatrix} -2.47339 + 2.22044 \times 10^{-16}i & -0.497917 + 0.206369i & 1.16137 + 0.96249i \\ 0. + 0.i & 6.7367 + 1.98542i & 4.97465 - 0.958923i \\ 0. + 0.i & 0. + 0.i & 6.7367 - 1.98542i \end{pmatrix}, \tag{14}$$

where $||Q.T.Q^* - A|| = 1.55552 \times 10^{-14}$.

Therefore, if we let $f(t) = \sqrt{t}$, then $\sqrt{A} = W\sqrt{J}W^{-1}$ or $\sqrt{A} = Q\sqrt{T}Q^*$, depending on whether Jordan decomposition or Schur decomposition is used to factor A, respectively. In this context, Jordan decomposition is capable of finding all square roots of A, including both primary and non-primary roots, if they exist. On the other hand, Schur decomposition only finds the primary square roots. Both methods simplify the task to calculating the square root of a block diagonal or triangular matrix. Schur and Jordan decompositions are both effective techniques for determining the square roots of various types of matrices, including non-positive definite symmetric ones. These methods can find all the square roots of a matrix, but Schur decomposition is particularly advantageous for larger matrices, offering better computational efficiency. In contrast, Jordan decomposition involves more complex calculations due to the need for computing Jordan blocks, which increases its computational complexity. Thus, the choice between the two methods depends on the matrix size and available computational resources.

Although alternative methods, such as eigenvalue decomposition or LU factorization, are available, the Schur and Jordan decompositions were selected due to their superior numerical stability and their ability to account for both primary and non-primary roots in complex or non-diagonalizable matrices. While other methods may be suitable in certain scenarios, they often face difficulties when applied to large or ill-conditioned matrices, particularly those that are non-diagonalizable. The Schur and Jordan decompositions, on the other hand, are more effective in handling such matrices, making them the preferred choice for this analysis.

4. The third step of George Polya's problem-solving: Carrying out the plan

In the previous section, we discussed the necessity of computing the square root of a block diagonal matrix and a triangular matrix.

4.1. Finding the square root of a block diagonal matrix

The square root of a Jordan block is first computed. Suppose the Jordan block $J_k \in \mathbb{C}^{m_k \times m_k}$ and the arbitrary relation $f(t)^2 = t$. Then, the result is given as

$$f(J_k) = \begin{pmatrix} f(\lambda_k) & f'(\lambda_k) & \dots & \frac{f^{(m_k-1)}(\lambda_k)}{(m_k-1)!} \\ & f(\lambda_k) & \ddots & \vdots \\ & & \ddots & f'(\lambda_k) \\ & & & f(\lambda_k) \end{pmatrix}, \tag{15}$$

where f has two branches: $f(t) = \sqrt{t}$ and $f(t) = -\sqrt{t}$. If one eigenvalue λ_s of matrix A appears in two Jordan blocks of the Jordan decomposition of A, and f is chosen from only one branch for λ_s, the primary square root is obtained. Otherwise, a nonprimary square root for matrix A is derived [12]. In this example, the workflow is outlined clearly step by step. It should also be noted that the function in question must be differentiable at the eigenvalues. However, it is assumed that the reader is familiar with general mathematics, and these details are not elaborated further.

Example 4.1. *Consider the following Jordan block:*

$$J = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 2 \end{pmatrix}. \tag{16}$$

The square root for J is computed using Mathematica and the pattern given in (15):

```
f[t_] = Sqrt[t];

(* The function  g is defined to obtain the Jordan form *)
g[i_, j_] := 0 /; i > j
g[i_, j_] := f[J[[i, j]]] /; i == j
g[i_, j_] := 1/(j - i)! Derivative[j - i][f][J[[i, i]]] /; i < j

(* The matrix generated by the following Table corresponds to sqrt{J} *)
Table[g[i, j], {i, 1, Length[J]}, {j, 1, Length[J]}] // MatrixForm
```

It is deduced that

$$J^{\frac{1}{2}} = \begin{pmatrix} \sqrt{2} & \frac{1}{2\sqrt{2}} & -\frac{1}{16\sqrt{2}} & \frac{1}{64\sqrt{2}} \\ 0 & \sqrt{2} & \frac{1}{2\sqrt{2}} & -\frac{1}{16\sqrt{2}} \\ 0 & 0 & \sqrt{2} & \frac{1}{2\sqrt{2}} \\ 0 & 0 & 0 & \sqrt{2} \end{pmatrix}. \tag{17}$$

If $f(x) = -\sqrt{x}$, another primary root, $\sqrt{J}$, is obtained.

Example 4.2. *The computation of the square roots of the following diagonal matrix is considered. The matrix is defined as*

$$D = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix}, \tag{18}$$

which is

$$D = diag(J_1, J_2, J_3, J_4), \tag{19}$$

where

$$J_1 = \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix}, \quad J_2 = \begin{pmatrix} 3 \end{pmatrix}, \quad J_3 = \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix}, \quad J_4 = \begin{pmatrix} 3 \end{pmatrix}. \tag{20}$$

The diagonal matrix D has 8 square roots, of which four are primary and four are nonprimary. The primary square roots are

$$\begin{aligned} sqrt_1(D) &= diag(\sqrt{J_1}, \sqrt{J_2}, \sqrt{J_3}, \sqrt{J_4}) = D^{\frac{1}{2}}, \\ sqrt_2(D) &= diag(\sqrt{J_1}, -\sqrt{J_2}, \sqrt{J_3}, -\sqrt{J_4}) = \sqrt{D}, \\ sqrt_3(D) &= diag(-\sqrt{J_1}, \sqrt{J_2}, -\sqrt{J_3}, \sqrt{J_4}) = \sqrt{D}, \\ sqrt_4(D) &= diag(-\sqrt{J_1}, -\sqrt{J_2}, -\sqrt{J_3}, -\sqrt{J_4}) = \sqrt{D}, \end{aligned} \tag{21}$$

and the nonprimary roots are

$$\begin{aligned} sqrt_5(D) &= diag(\sqrt{J_1}, \sqrt{J_2}, -\sqrt{J_3}, -\sqrt{J_4}) = \sqrt{D}, \\ sqrt_6(D) &= diag(-\sqrt{J_1}, -\sqrt{J_2}, \sqrt{J_3}, \sqrt{J_4}) = \sqrt{D}, \\ sqrt_7(D) &= diag(-\sqrt{J_1}, \sqrt{J_2}, \sqrt{J_3}, -\sqrt{J_4}) = \sqrt{D}, \\ sqrt_8(D) &= diag(\sqrt{J_1}, -\sqrt{J_2}, -\sqrt{J_3}, \sqrt{J_4}) = \sqrt{D}, \end{aligned} \tag{22}$$

where

$$\sqrt{J_1} = \sqrt{J_3} = \begin{pmatrix} \sqrt{2} & \frac{1}{2\sqrt{2}} \\ 0 & \sqrt{2} \end{pmatrix}, \quad \sqrt{J_2} = \sqrt{J_4} = \begin{pmatrix} \sqrt{3} \end{pmatrix}. \tag{23}$$

4.2. Finding the Square Root of an Upper Triangular Matrix

This section explores an algorithm for computing the square root of an upper triangular matrix. For detailed proofs and algebraic properties, standard references such as [12] are recommended. A numerical example along with its implementation using Mathematica is provided to demonstrate the process.

Given an upper triangular matrix T, its square root is also an upper triangular matrix, with diagonal elements equal to $\sqrt{t_{ii}}$. The function f must be well-defined on the spectrum of T. The off-diagonal (superdiagonal) elements are determined using the following formula:

$$F_{ij} = \sum_{(k_0,\ldots,k_m)\in S_{ij}} t_{k_0,k_1} t_{k_1,k_2} \ldots t_{k_{m-1},k_m} f[\lambda_{k_0}, \ldots, \lambda_{k_m}], \tag{24}$$

where $\lambda_i = t_{ii}$, $f[\lambda_{k_0}, \ldots, \lambda_{k_m}]$ represents the m-th order divided difference of $f(t) = t^2$ at $\lambda_{k_0}, \ldots, \lambda_{k_m}$, and S_{ij} is the set of strictly increasing sequences of integers that begin at i and end at j. By following this method, all primary square roots of T can be computed.

Example 4.3. *To illustrate this formula, consider the following matrix:*

$$T = \begin{pmatrix} 1 & -1 & 3 \\ 0 & 2 & 2 \\ 0 & 0 & 2 \end{pmatrix}. \tag{25}$$

The square root of T is another triangular matrix. Its diagonal elements are 1, $\sqrt{2}$, and $\sqrt{2}$, corresponding to the branch $f(x) = \sqrt{x}$. Using alternative branches of $f(x)$, other primary square roots can be obtained. For the off-diagonal elements F_{12}, F_{23}, and F_{13}, the following computations are made using the formula above:

```
f1[x_] = Sqrt[x];
f2[x_]= -Sqrt[x];
f3[x_]= If[x==1,Sqrt[x],-Sqrt[x]];
f4[x_]= If[x==2,Sqrt[x],-Sqrt[x]];

(* The functions g is defined for divided difference of order one and two *)
g[x_, y_,f_] := (f[x] - f[y])/(x - y) /; x != y
g[x_, y_,f_] := f'[x] /; x == y
g[x_, y_, y_,f_] = (g[x, y,f] - f'[y])/(x - y);
g[x_, y_, z_,f_] = (g[x, y,f] - g[y, z,f])/(x - z);
```

```
(* Fij is ij-th element of superdiagonal elements of matrix F
according to previous formula *)
(* T[[i,j]] is ij-th element of matrix T *)
F12 = T[[1, 2]] g[T[[1, 1]], T[[2, 2]],f];
F23 = T[[2, 3]] g[T[[2, 2]], T[[3, 3]],f];
F13 = T[[1, 3]] g[T[[1, 1]], T[[3, 3]],f] +
T[[1, 2]] T[[2, 3]] g[T[[1, 1]], T[[2, 2]], T[[3, 3]],f];
```

where $S_{12} = \{\{1,2\}\}$, $S_{23} = \{\{2,3\}\}$, *and* $S_{13} = \{\{1,3\},\{1,2,3\}\}$.
Thus, the principal square root of T *is:*

$$T^{\frac{1}{2}} = \begin{pmatrix} 1 & 1-\sqrt{2} & -5+\frac{9}{\sqrt{2}} \\ 0 & \sqrt{2} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \sqrt{2} \end{pmatrix}. \tag{26}$$

If $f(x) = -\sqrt{x}$, *then:*

$$\sqrt{T} = \begin{pmatrix} -1 & \sqrt{2}-1 & 5-\frac{9}{\sqrt{2}} \\ 0 & -\sqrt{2} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & -\sqrt{2} \end{pmatrix}.$$

To compute other square roots, each eigenvalue is assigned a different branch. For example:

$$\sqrt{T} = \begin{pmatrix} 1 & \sqrt{2}+1 & -5-\frac{9}{\sqrt{2}} \\ 0 & -\sqrt{2} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & -\sqrt{2} \end{pmatrix}, \qquad \sqrt{T} = \begin{pmatrix} -1 & -\sqrt{2}-1 & 5+\frac{9}{\sqrt{2}} \\ 0 & \sqrt{2} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \sqrt{2} \end{pmatrix}.$$

Therefore, four primary square roots of the matrix T *are obtained. However, for large-sized matrices, it is necessary to define divided differences for higher dimensions, which increases computational complexity. Thus, proposing an alternative method or algorithm could be an effective approach to mitigate this issue.*

4.2.1. Parlett Algorithm

In this section, an alternative algorithm for computing the matrix square root is explored. Due to the high computational cost of Schur algorithm, Parlett introduced an algorithm for calculating $F = f(T)$ [21]. The algorithm leverages the fact that $f(T)$ commutes with T, and for the diagonal elements of T, we have $F_{ii} = f(T_{ii})$. In this paper, Parlett's algorithm is implemented in Mathematica. This method computes $F = f(T)$ for matrices T with distinct eigenvalues, where $f(t)^2 = t$.

```
f[x_] = Sqrt[x];

(* The diagonal elements of the square root of the matrix T are to be computed. *)
F[i_, j_] := f[T[[i, j]]] /; i == j

(* The elements appearing on the under diagonal of F are zero *)
F[i_, j_] := 0 /; i > j

(* Using the function A, the upperdiagonal elements are computed
by using diagonal elements and previously computed elements *)
For[j = 2, j <= Length[T], j++,
For[i = j - 1, i >= 1, i--,
A[i_, j_] :=  T[[i, j]]*(F[i, i] - F[j, j])/(T[[i, i]] - T[[j, j]]) +
Sum[(F[i,k]T[[k,j]] - T[[i,k]]F[k,j]), {k, i+1, j-1}]/(T[[i, i]] - T[[j, j]])]]

F[i_, j_] := A[i, j] /; i < j

(* The square root of the matrix T is denoted by sqrtT. *)
sqrtT = Table[F[i, j], {i, 1, Length[T]}, {j, 1, Length[T]}];
sqrtT // MatrixForm
```

4.2.2. Parlett's Block Algorithm and Computational Techniques for Solving the Sylvester Equation

However, if $T_{ii} = T_{jj}$ for some $i \neq j$, Parlett's algorithm may fail. In such cases, the block Parlett algorithm serves as an alternative. Consider a matrix $T \in \mathbb{C}^{n\times n}$ that is divided into blocks such that no two diagonal blocks share the same eigenvalues. Using this structure, we can compute $F = f(T)$. This method ensures that the square root of a triangular matrix remains triangular. Thus, the square roots of the diagonal blocks must be computed individually. Moreover, for any subdiagonal block F_{ij}, the following Sylvester equation must be solved, which results in the desired square root, $\sqrt{F_{ij}}$ (further details are in [12, 2, 3]):

```
T[[i, i]]F[[i, j]] - F[[i, j]]T[[j, j]] =T[[i, j]]F[[i, i]] - F[[j, j]]T[[i, j]] +
Sum[(F[[i,k]]T[[k,j]] - T[[i,k]]F[[k,j]]), {k, i+1, j-1}]
```

The above Sylvester equation represents a system where the right-hand side consists of known matrices that, upon computation, reduce to a singular matrix, denoted as C. On the left-hand side, the unknown matrix F_{ij} is scaled by the diagonal matrices T_{ii} and T_{jj}, which are both upper triangular.

To solve this system using the Bartels-Stewart method [1], the equation is first simplified using a Schur decomposition. Specifically, the matrices T_{ii} and T_{jj} are decomposed into upper triangular forms using unitary transformations: $T_{ii} = UTU^\star$ and $T_{jj} = VSV^\star$, where T and S are upper triangular matrices. Substituting these decompositions into the Sylvester equation, the equation becomes the following equivalent form:

$$TY - YS = F,$$

where $Y = U^\star F_{ij} V$ and $F = U^\star CV$. This transformation simplifies the original equation while preserving its structure, as the resulting triangular form allows for efficient forward or backward substitution.

In cases where the unitary matrices U and V are identity matrices, the decomposition simplifies. In this case, $F = C$ and $F_{ij} = Y$, and the equation reverts to its original form. These observations indicate that the simplified form obtained through the Schur decomposition is essentially equivalent to the original equation when no transformation is applied.

The Sylvester equations produced by the block Parlett algorithm are inherently simplified through the Bartels-Stewart method. This consistency highlights the robustness and efficiency of this method in practical applications.

Next, a few definitions are needed, which are described below.

Definition 4.1 ([12]). *The **Kronecker product** of two matrices $A \in \mathbb{C}^{m\times n}$ and $B \in \mathbb{C}^{p\times q}$ is defined as $A \otimes B = (a_{ij}B) \in \mathbb{C}^{mp\times nq}$.*

Definition 4.2 ([12]). *Let $A \in \mathbb{R}^{m\times n}$ be a matrix with columns $a_i \in \mathbb{R}^m$ for $i = 1, 2, \ldots, n$. The vector $vec[A]$ is a vector of dimension mn constructed by stacking the columns of A as follows:*

$$vec[A] = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \in \mathbb{R}^{mn}.$$

For matrices A, B, and X, the following identity holds:

$$vec[AXB] = (B^T \otimes A)\, vec[X].$$

These two definitions are illustrated with examples in Mathematica.

Example 4.4. *Consider the following matrices:*

$$A = \begin{pmatrix} 1 & 3 \\ 2 & 5 \end{pmatrix}, \qquad B = \begin{pmatrix} 6 & -1 \\ 0 & 2 \end{pmatrix}. \tag{27}$$

```
In[1]:= KroneckerProduct[A, B]
Out[1]= {{6, -1, 18, -3}, {0, 2, 0, 6}, {12, -2, 30, -5},
{0, 4, 0, 10}}

(* A matrix is a nested list and Flatten[list] flattens out nested
lists so that we can obtain the vec of the matrix *)
In[2]:= Flatten[Transpose[A]]
Out[2]= {1, 2, 3, 5}
```

To solve the Sylvester equation $AX - XB = C$, where $A \in \mathbb{C}^{m\times m}$, $B \in \mathbb{C}^{n\times n}$, $C \in \mathbb{C}^{m\times n}$, and $X \in \mathbb{C}^{m\times n}$, the problem can be rewritten as $P\text{vec}(X) = \text{vec}(C)$, where $P = I_n \otimes A - B^T \otimes I_m$, with $\otimes$ denoting the Kronecker product. Importantly, if A and B do not share any eigenvalues, P is guaranteed to be non-singular, ensuring the existence of a unique solution.

This method aids in solving the Sylvester equation presented by the block Parlett algorithm, where A and B are upper triangular matrices. Additionally, the Sylvester equation can also be solved using iterative methods such as the Smith iterative method [17], which is particularly efficient for large-dimensional matrices.

Example 4.5. *Consider the following triangular matrix:*

$$T = \begin{pmatrix} 2 & 1 & -1 & 2 \\ 0 & 2 & 1 & 1 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}, \tag{28}$$

where

$$T_{11} = \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix}, \quad T_{12} = \begin{pmatrix} -1 & 2 \\ 1 & 1 \end{pmatrix}, \quad T_{22} = \begin{pmatrix} 3 & 1 \\ 0 & 3 \end{pmatrix}. \tag{29}$$

The square roots of the diagonal blocks, F_{11} and F_{22}, are computed as:

$$F_{11} = \begin{pmatrix} \sqrt{2} & \frac{1}{2\sqrt{2}} \\ 0 & \sqrt{2} \end{pmatrix}, \quad F_{22} = \begin{pmatrix} \sqrt{3} & \frac{1}{2\sqrt{3}} \\ 0 & \sqrt{3} \end{pmatrix}. \tag{30}$$

To compute F_{12}, the off-diagonal block, the Sylvester equation

$$T_{11}F_{12} - F_{12}T_{22} = F_{11}T_{12} - T_{12}F_{22} \tag{31}$$

must be solved. This equation is rewritten as

$$P vec(F_{12}) = vec(F_{11}T_{12} - T_{12}F_{22}), \quad where \quad P = I_2 \otimes T_{11} - T_{22}^T \otimes I_2.$$

Using Mathematica, the solution is obtained as follows:

```
(* For solving the Sylvester equation, the matrix P that is specified must be computed. *)
P = KroneckerProduct[IdentityMatrix[2], T11] -
KroneckerProduct[ConjugateTranspose[T22], IdentityMatrix[2]];

(* The vec operator for the right-hand side of the
Sylvester equation is computed and defined as vecC. *)
vecC = Flatten[Transpose[F11.T12 - T12.F22]];

(* The vec of the superdiagonal matrix element F12
is the solution of the linear system
P.vec(F12) = vecC where vecC = F11.T12 - T12.F22 *)
vecF12 = LinearSolve[P, vecC];

(*
By using the concept of the vec operator, the matrix F12 is obtained. *)
F12 = {{vecF12[[1]], vecF12[[3]]}, {vecF12[[2]], vecF12[[4]]}};

(* The diagonal matrices F11 and F22 are concatenated, along with the superdiagonal
 matrix F12 and zero matrix, using Join[list1, list2, ...]. *)
(* ArrayFlatten[{{F11,F12},{F21,F22}}] creates a
single flattened matrix from a matrix of matrices Fij *)
sqrtT = ArrayFlatten[Join[{{F11, F12}, {ConstantArray[0, {2, 2}], F22}}]]
```

Then the principal square root of the triangular matrix T is:

$$T^{\frac{1}{2}} = \begin{pmatrix} \sqrt{2} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -2(\sqrt{2}-\sqrt{3}) \\ 0 & \sqrt{2} & -\sqrt{2}+\sqrt{3} & \frac{1}{2\sqrt{3}} \\ 0 & 0 & \sqrt{3} & \frac{1}{2\sqrt{3}} \\ 0 & 0 & 0 & \sqrt{3} \end{pmatrix}. \tag{32}$$

For this example, the eigenvalues are distinct and appear only in one block. If the square root were to be computed using other branches of f, or as in previous examples, by assigning each eigenvalue to a separate branch, all four primary square roots could be obtained. However, since no eigenvalue should appear in more than one block in this method, the nonprimary square roots are not obtained.

4.3. Finding the Square Root of a Matrix Using Iterative Methods

Iterative methods are a useful approach for approximating the square root of a matrix. It is important to remember that scalar iterative methods do not necessarily generalize to matrices.

A matrix iteration is defined by

$$X_{k+1} = g(X_k), \tag{33}$$

where X_0 is an initial guess, often chosen as a fixed function of A. The iteration function g is said to have an order of convergence p if the generated sequence of matrices $\{X_k\}$ converges to X_* with order p. This means

$$\|X_* - X_{k+1}\| \leq c\|X_* - X_k\|^p, \tag{34}$$

where $\{X_k\}$ converges to X_* for sufficiently large k, and c is a positive constant.

4.3.1. Newton's Method

Newton's method is an iterative technique for computing the matrix square root. Beginning with an initial guess X_0, the iteration is defined by the update formula:

Newton Iteration:
$X_{k+1} = \frac{1}{2}\left(X_k + X_k^{-1}A\right), \quad X_0 = A.$

This method is well-known for its quadratic convergence and simplicity, making it widely used. However, it struggles with ill-conditioned matrices or those with negative eigenvalues, leading to potential numerical instability or divergence. Preprocessing techniques like scaling can help, but they add complexity.

4.3.2. DB Iteration

To address challenges of the Newton method, the Denman-Beavers (DB) method is proposed, which is designed to offer superior numerical stability. It handles problematic matrices effectively without requiring complex preprocessing or stabilization.

The DB iteration refines the computation by introducing an auxiliary matrix $Y_k = A^{-1}X_k$. The iterative process is described as:

DB Iteration:
$X_{k+1} = \frac{1}{2}\left(X_k + Y_k^{-1}\right), \quad X_0 = A,$
$Y_{k+1} = \frac{1}{2}\left(Y_k + X_k^{-1}\right), \quad Y_0 = I.$

By defining $M_k = X_kY_k$, a product form of the DB iteration emerges, simplifying certain computational aspects:

Product Form of the DB Iteration:
$M_{k+1} = \frac{1}{2}\left(I + \frac{M_k + M_k^{-1}}{2}\right), \quad M_0 = A.$
$X_{k+1} = \frac{1}{2}X_k\left(I + M_k^{-1}\right), \quad X_0 = A.$
$Y_{k+1} = \frac{1}{2}Y_k\left(I + M_k^{-1}\right), \quad Y_0 = I.$

The DB iteration, through its coupling of X_k and Y_k, provides improved numerical stability and convergence behavior under certain conditions. The DB product method, using two auxiliary matrices, refines the matrix square root approximation through iterative matrix multiplication, offering greater stability, especially for ill-conditioned matrices. However, similar to the Newton method and the DB method, it requires matrix inversions in each iteration, which can be computationally expensive for large-scale problems.

4.3.3. Newton-Schulz Iteration

The Newton-Schulz iteration is another approach for computing the matrix square root, often used for matrices where the spectral radius of $A - I$ lies within the interval $(0, 1)$, i.e., $\rho(A - I) < 1$. Unlike the Newton and DB iterations, the Newton-Schulz method does not require explicit matrix inversions, which can make it more efficient in some cases. The iteration is based on the following formula:

Newton-Schulz Iteration:
$X_{k+1} = \frac{1}{2} X_k \left(3I - Y_K X_k\right), \quad X_0 = A,$
$Y_{k+1} = \frac{1}{2} \left(3I - Y_K X_k\right) Y_K, \quad Y_0 = I,$

where X_k converges to $A^{\frac{1}{2}}$ and Y_k converges to $A^{-\frac{1}{2}}$, such as in the DB method and the product form of the DB iteration.

This method is particularly advantageous for matrices that are well-conditioned and have eigenvalues within the specified range. Its cubic convergence ensures rapid computation in favorable cases. However, its performance still depends on the properties of the matrix A and the quality of the initial guess X_0.

In summary, while Newton, DB, and Newton-Schulz iterations each have unique advantages and limitations, they represent only a subset of iterative methods for computing the matrix square root. Other iterative techniques, such as those based on polar decomposition or scaling-and-squaring, also exist and are discussed extensively in texts like [12].

4.3.4. Stopping Criteria

To determine when to stop the iterative process, we use stopping criteria based on acceptable errors. The iterations are continued until either

$$\|X_{k+1} - X_k\| < \text{tolerance} \tag{35}$$

or

$$\|X_k^2 - A\| < \text{tolerance}. \tag{36}$$

These criteria help us decide when the approximation is sufficiently accurate.

4.3.5. Implementation of the Proposed Iterative Methods in Mathematica

The Mathematica code for the iterative methods presented in this article is provided as follows:

```
Method = Input["Method"];
X[0] = A; M[0] = A; Y[0] = IdentityMatrix[Length[A]];k = 0;
Timing[While[Norm[X[k]. X[k]- A, 2] > tolerance,
If[Method == "Netwon", X[k + 1] = 1/2 (X[k] + Inverse[X[k]] . A)];
If[Method == "DB", X[k + 1] = 1/2 (X[k] + Inverse[Y[k]]);
Y[k + 1] = 1/2 (Y[k] + Inverse[X[k]])];
If[Method == "ProductDB", M[k + 1] =
1/2 (IdentityMatrix[Length[A]] + (M[k] + Inverse[M[k]])/2);
X[k + 1] = 1/2 X[k] . (IdentityMatrix[Length[A]] + Inverse[M[k]]);
Y[k + 1] = 1/2 Y[k].(IdentityMatrix[Length[A]] + Inverse[M[k]])];
If[Method == "Netwon-Schulz",
X[k + 1] = 1/2 (X[k] . (3 IdentityMatrix[Length[A]] - Y[k] . X[k]));
Y[k + 1] = 1/2 (3 IdentityMatrix[Length[A]] - Y[k] . X[k]) . Y[k]];
k = k + 1];]
```

Here, $X[0]$, $M[0]$, and $Y[0]$ are chosen as the initial guesses for the selected iterative method, typically starting with the matrix A. The `while` loop continues until the error $\|X_k^2 - A\|$ is less than the specified tolerance. The output $X[k]$ provides an approximation for the square root of the matrix A.

Example 4.6. *Consider the following matrix A with a tolerance of 10^{-4}. The square root of A is computed using the iterative methods introduced in this paper. Table 1 summarizes the number of iterations, the error $\|X_k^2 - A\|$, and the computational time for each method. It is noteworthy that the Newton-Schulz method diverged for this example. Consequently, another example is provided later to evaluate its performance.*

$$A = \begin{pmatrix} 4 & 2 & 1 & 1 & 3 & 5 \\ 1 & -1 & 2 & -1 & 2 & 9 \\ 1 & -2 & 1 & 2 & 1 & 2 \\ 3 & 0 & 5 & 2 & 5 & 6 \\ -2 & 0 & 3 & 1 & 4 & 0 \\ 1 & 1 & 0 & 3 & 2 & 2 \end{pmatrix} \tag{37}$$

Table 1: Results for computing the square root of matrix A using the iterative methods

Method	Iterations	$\|X_k^2 - A\|$	Computation Time (s)
Newton Iteration	6	1.29653×10^{-5}	0.03125
DB Iteration	6	1.29653×10^{-5}	0.046875
DB product Iteration	6	1.29653×10^{-5}	0.046875

The computed square root matrix is as follows:

$$\sqrt{A} = \begin{pmatrix} 1.90407 & 0.610407 & -0.152974 & 0.311732 & 0.513736 & 0.919227 \\ 0.774544 & 0.773852 & 2.66055 & -2.52578 & 0.928596 & 3.79871 \\ 0.59904 & -0.800893 & 2.30266 & -0.761056 & 0.470849 & 1.61627 \\ 0.948325 & 0.184253 & 1.55629 & 0.996279 & 1.18826 & 1.13902 \\ -0.817412 & 0.202369 & 0.177149 & 0.864045 & 1.79804 & -0.438272 \\ 0.128034 & 0.0886207 & -0.605567 & 1.17197 & 0.25335 & 1.14042 \end{pmatrix}$$

The computed square root matrix has complex eigenvalues, indicating that it is not the principal square root of A. Since A does not have repeated eigenvalues, the obtained square root is a primary, non-principal square root of the matrix A.

Example 4.7. *The square root of the ill-conditioned Hilbert matrix is computed using the iterative methods presented in this paper. The results for computing the square roots of the matrix H for different sizes n are shown in Table 2. As observed in the table, the Newton-Schulz method succeeds in providing the square roots for larger matrices, while other methods encounter difficulties due to issues with matrix inversion.*

By utilizing the Schur decomposition, a matrix H can first be decomposed into the form $H = WTW^*$, where W is a unitary matrix and T is an upper triangular matrix. Subsequently, the square root of the triangular matrix T can be computed using any iterative method. Finally, the square root of the original matrix H can be reconstructed as $\sqrt{H} = W\sqrt{T}W^*$.

The square root of the matrix H was also computed using this approach. As shown in Table 2, the square roots of larger matrices were calculated in less time, demonstrating the efficiency of this method for larger dimensions.

5. The fourth step of George Polya's problem-solving: Review the solutions obtained

In this paper, several well-established methods for computing the square root of a matrix have been discussed, including Schur decomposition, Jordan decomposition, and four iterative methods: Newton's method, DB method, DB product method, and the Newton-Schulz iteration. Our primary goal was to create a structured and educational approach to help readers understand and compute matrix square roots effectively.

The selected methods provide a balanced combination of theoretical depth and computational strategies. For instance, the Jordan decomposition can compute all square roots of a matrix, including primary and nonprimary roots, but its accuracy diminishes for large matrices. In contrast, the Schur decomposition reliably computes all primary roots and remains accurate even for large-scale matrices.

Newton method is well-regarded for its simplicity and quadratic convergence, making it a widely favored choice across many applications. However, it faces notable challenges when dealing with ill-conditioned matrices or those with negative eigenvalues. In such cases, numerical instability or even divergence can occur, which limits the practical applicability of the method. While preprocessing steps, such as scaling or matrix conditioning, can improve the situation, they often add complexity to the problem.

To address these limitations, we propose an alternative method, known as the DB method, which is designed to offer superior numerical stability. Unlike traditional methods, the DB approach effectively handles challenging cases, such as matrices with negative eigenvalues or ill-conditioning, without requiring intricate preprocessing or stabilization techniques. Furthermore, the DB product method, which is employed in this framework, uses two

Table 2: The results for computing the square root of matrix H using the presented iterative methods.

Method	n	**Iterations**	$\|X_k^2 - H\|$	**Computation Time (s)**
Newton iteration	5	7	5.99508×10^{-5}	0.0156
Newton iteration	10	7	6.10352×10^{-5}	0.0313
Newton iteration	20	7	6.10352×10^{-5}	0.0313
DB iteration	20	7	6.10352×10^{-5}	0.0625
DB iteration	30	7	6.10352×10^{-5}	0.1094
DB product iteration	20	7	6.10352×10^{-5}	0.1094
DB product iteration	30	7	6.10352×10^{-5}	0.2656
Newton-Schulz iteration	50	10	8.97681×10^{-5}	0.1250
Newton-Schulz iteration	100	11	3.65037×10^{-5}	0.5469
Newton-Schulz iteration	400	10	9.51416×10^{-5}	25.9219
Newton-Schulz iteration	1000	11	5.07702×10^{-5}	435.266
Schur and Newton iteration	1000	7	6.10352×10^{-5}	4.2656
Schur and DB iteration	2000	7	6.10352×10^{-5}	51.0938
Schur and Newton-Schulz iteration	2000	11	4.85025×10^{-5}	84.4375
Schur and DB product iteration	4000	7	6.10352×10^{-5}	286.859
Schur and Newton-Schulz iteration	4000	11	4.45527×10^{-5}	371.875

auxiliary matrices in each iteration to refine the approximation of the matrix square root through matrix multiplication. This iterative process enhances the stability of the algorithm and yields more reliable results, even in the face of problematic matrices. Compared to other methods, the DB method provides a significant advantage in terms of both stability and efficiency, particularly when working with matrices that present common computational difficulties.

The iterative methods explored include three that require matrix inversion and one (Newton-Schulz) that avoids this step. These methods are particularly valuable for their computational flexibility. By combining the Schur decomposition with iterative methods, the challenges of computing square roots for large and ill-conditioned matrices were successfully addressed, demonstrating the effectiveness of this hybrid approach.

It is important to acknowledge that there are other methods available in the literature for computing matrix square roots. Techniques such as diagonalization, the matrix sign function, Padé approximations, and other advanced iterative algorithms provide alternative frameworks [12, 11]. Readers seeking a broader understanding are encouraged to explore these additional resources.

The methods discussed in this paper were chosen for their clarity, relevance to the educational goals, and practical utility. By presenting a variety of techniques and combining them strategically, A comprehensive framework for understanding and solving matrix square root problems is offered. This work not only addresses computational challenges but also aligns with George Pólya's problem-solving strategy, emphasizing clear reasoning, exploration of methods, and systematic problem-solving.

The paper also provides engaging examples that enhance the learning experience and make the process of understanding matrix square roots more efficient and intuitive.

6. Statistical Analysis of Learning Outcomes

In this study, two groups of ten graduate and doctoral students from various engineering disciplines, including mathematics, electrical engineering, mechanical engineering, and civil engineering, were selected. All participants had a prior familiarity with Mathematica and demonstrated a keen interest in learning the subject matter. The structure of the study was inspired by the approach outlined by Malmia et al. [18]. The students attended five in-person sessions designed to teach the new method presented in this paper. For those in the traditional learning group, relevant materials were provided, along with a single review session to address any questions. Due to their enthusiasm, participants were given the test questions and asked to submit their answers within one day.

To ensure the validity of the comparison, the normality of the test scores was first assessed using the Shapiro-Wilk test, yielding p-values of 0.08 for the first group and 0.12 for the second group, indicating that the data followed a normal distribution. Following this, an independent samples t-test was conducted to compare the mean scores between the two groups. The t-test resulted in a t-value of 2.45 with a p-value of 0.024, suggesting a statistically significant difference in performance between the two groups at the $\alpha = 0.05$ level.

These results indicate that the group taught using the new method outperformed the group that used traditional reference materials, confirming the efficacy of the proposed approach.

Acknowledgment

The author would like to thank the anonymous reviewers for their valuable feedback and insightful comments, which greatly helped in improving the quality of this research. Their constructive suggestions and recommendations are highly appreciated.

References

[1] R. H. Bartels and G. W. Stewart, *Solution of the matrix equation AX+XB=C [F4]*, Commun. ACM, 15 (1972), pp. 820–826.

[2] C. A. Bavely and G. W. Stewart, *An algorithm for computing reducing subspaces by block diagonalization*, SIAM J. Numer. Anal., 16 (1979), pp. 359–367.

[3] P. Benner, E. S. Quintana-Ortí, and G. Quintana-Ortí, *Solving stable Sylvester equations via rational iterative schemes*, J. Sci. Comput., 28 (2006), pp. 51–83.

[4] A.-L. Cirneanu and C.-E. Moldoveanu, *Use of digital technology in integrated mathematics education*, Appl. Syst. Innov., 7 (2024).

[5] M. Dehghan and M. Hajarian, *Determination of a matrix function using the divided difference method of Newton and the interpolation technique of Hermite*, J. Comput. Appl. Math., 231 (2009), pp. 67–81.

[6] ——, *Computing matrix functions using mixed interpolation methods*, Math. Comput. Modelling, 52 (2010), pp. 826–836.

[7] M. Dehghan, G. Karamali, and A. Shirilord, *An iterative scheme for a class of generalized Sylvester matrix equations*, AUT J. Math. Comput., 5 (2024), pp. 195–215.

[8] M. Dehghan and A. Shirilord, *Matrix multisplitting Picard-iterative method for solving generalized absolute value matrix equation*, Appl. Numer. Math., 158 (2020), pp. 425–438.

[9] ——, *On the Hermitian and skew-Hermitian splitting-like iteration approach for solving complex continuous-time algebraic Riccati matrix equation*, Appl. Numer. Math., 170 (2021), pp. 109–127.

[10] N. J. Higham, *Computing real square roots of a real matrix*, Linear Algebra Appl., 88/89 (1987), pp. 405–430.

[11] ——, *Stable iterations for the matrix square root*, Numer. Algorithms, 15 (1997), pp. 227–242.

[12] ——, *Functions of Matrices*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2008. Theory and Computation.

[13] R. A. Horn and C. R. Johnson, *Topics in matrix analysis*, Cambridge University Press, 1991.

[14] B. Iannazzo, *On the Newton method for the matrix pth root*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 503–523.

[15] M. Kim, J. Kim, and W. Lee, *Intellectual disabilities and programming: Improving computational thinking-based problem solving*, Educ. Inf. Technol., (2025).

[16] D. Lambić, *Presenting practical application of mathematics by the use of programming software with easily available visual components*, Teach. Math. Appl., 30 (2011), pp. 10–18.

[17] J. Li, Z. Mei, and X. Kong, *A new version of the Smith method for solving Sylvester equation and discrete-time Sylvester equation*, J. Appl. Anal. Comput., 6 (2016), pp. 582–599.

[18] W. Malmia, S. H. Makatita, S. Lisaholit, A. Azwan, I. Magfirah, H. Tinggapi, and M. C. B. Umanailo, *Problem-based learning as an effort to improve student learning outcomes*, Int. J. Sci. Technol. Res, 8 (2019), pp. 1140–1143.

[19] M. Moccari, *Smoothing the absolute value equations by the component-wise analysis*, J. Math. Model., 13 (2025), pp. 251–262.

[20] M. Moccari, T. Lotfi, and V. Torkashvand, *On the stability of a two-step method for a fourth-degree family by computer designs along with applications*, Int. J. Nonlinear Anal. Appl., 14 (2023), pp. 261–282.

[21] B. N. Parlett, *A recurrence among the elements of functions of triangular matrices*, Linear Algebra Appl., 14 (1976), pp. 117–121.

[22] G. Polya, *How to Solve It. A New Aspect of Mathematical Method*, Princeton University Press, Princeton, NJ, 1948.

[23] A. Shirilord and M. Dehghan, *Closed-form solution of non-symmetric algebraic Riccati matrix equation*, Appl. Math. Lett., 131 (2022), pp. Paper No. 108040, 10.

[24] M. Z. Ullah, V. Torkashvand, S. Shateyi, and M. Asma, *Using matrix eigenvalues to construct an iterative method with the highest possible efficiency index two*, Mathematics, 10 (2022), p. 1370.

[25] S. Wolfram, *Mathematica, Wolfram Media*, 1999.

Please cite this article using:

Mandana Moccari, Computing the matrix square root: A problem-solving approach using Mathematica and Pólya's strategies, AUT J. Math. Comput., 7(3) (2026) 303-320
https://doi.org/10.22060/AJMC.2025.23508.1262