Journal of Engineering Technology and Applied Sciences, 2023 *e-ISSN:* 2548-0391
Received 16 May 2023, Accepted 24 July 2023 *Vol. 8, No. 2, 73-85*
Published online 26 August 2023, Research Article *doi: 10.30931/jetas.1298099*

Citation: Karaduman, G., "Streamlining Square Root Matrix Function Computation with Restarted Heavy Ball Algorithm". Journal of Engineering Technology and Applied Sciences 8 (2) 2023 : 73-85.

STREAMLINING SQUARE ROOT MATRIX FUNCTION COMPUTATION WITH RESTARTED HEAVY BALL ALGORITHM

Gul Karaduman[a,b]

[a]*Vocational School of Health Services, Karamanoglu Mehmetbey University, Karaman, 70200, Turkey*
[b]*University of Texas at Arlington, Department of Mathematics, Arlington, TX 76019-0408, USA*
gulk@bu.edu

Abstract

This research proposes a new efficient algorithm for calculating the square root function of the large-scale nonsingular sparse matrix by restarting the Heavy Ball Algorithm. The square root matrix function is critical in various applications, including signal processing, image processing, and machine learning. However, its computation is challenging due to existing methods' high computational complexity and numerical instability. The restarted Heavy Ball Algorithm provides a streamlined and efficient approach for computing the square root matrix function. The approach demonstrates its effectiveness through numerical experiments on various matrices, showing its superior performance compared to existing state-of-the-art methods. Numerical results show that the restarted Heavy Ball algorithm is feasible and effective for calculating the square root function.

Keywords: Matrix functions, heavy ball method, square root matrix, iterative methods

1. Introduction

Matrix computation is a fundamental operation in machine learning and data science. One of the most commonly used matrix operations is the calculation of the square root of a matrix. The square root matrix function is a function that takes a square matrix as its input and produces a matrix X as its output. In other words, the square root matrix function returns a matrix that, when squared, gives the original input matrix. Consider the following nonlinear matrix equation,

$$X^2 - A = 0, \tag{1}$$

where A is an $n \times n$ nonsingular matrix. A solution X of (1) is called a square root of A. The square root matrix function is a fundamental operation in many scientific and engineering applications. However, the computation of the square root matrix function is challenging due to the high computational complexity and numerical instability of existing methods. The function is useful in various mathematics and engineering fields, such as computer graphics, control theory, signal processing, statistics, machine learning, and quantum mechanics. For example, in control theory, the square root matrix function is used to compute the square root of a positive definite matrix, which is needed to design optimal control systems.

The matrix square root has various applications in different fields—computer graphics; it is used to convert objects between coordinate systems [1]. It helps compute the state feedback gain matrix for controlling dynamic systems in control systems. In quantum mechanics, it aids in calculating the evolution of quantum states represented by density matrices. In multivariate statistics, it helps determine the standard error of a sample covariance matrix. In optimization problems, it simplifies problem transformation for easier solutions. In image processing, it assists in compressing images through square root calculations of positive definite matrices. In machine learning, it enables the computation of the Mahalanobis distance to measure the similarity between data points. Moreover, the square root matrix function can be extended to compute the fractional powers of matrices involving real exponents between 0 and 1. Various methods for calculating square root matrices can be found in the existing literature [2-6].

In general, the square root matrix function is a fundamental concept in matrix theory and finds applications in various fields. Numerous techniques have been proposed for calculating the square root of a matrix. These computational methods can be broadly categorized into two classes. The first class comprises direct methods, often incurring high time and space costs.

The matrix power method is a commonly used approach for computing the square root matrix function. However, this method suffers from slow convergence and numerical instability. In contrast, iterative methods require less computational time and storage space [7, 8]. Iterative approaches, such as matrix iterations of the form $X_{k+1} = f(X_k)$, where f represents a polynomial or a rational function, present appealing alternatives for square root computations.

Each method has its own advantages and disadvantages depending on the properties of the input matrix. Here are some common methods:

Schur decomposition method: This method is based on the Schur decomposition of the input matrix A, which decomposes A into an upper triangular matrix and a unitary matrix. The square root matrix function of A can be computed by taking the square root of the diagonal elements of the upper triangular matrix. Schur algorithm was developed by Björck and Hammarling [9].

Diagonalization method: This method is based on the diagonalization of the input matrix A, which decomposes A into a diagonal matrix and a matrix of eigenvectors. The square root matrix function of A can be computed by taking the square root of the diagonal matrix [10].

Polar decomposition method: This method is based on the polar decomposition of the input matrix A, which decomposes A into a product of a unitary matrix and a positive semi-definite matrix. The square root matrix function of A can be computed by taking the square root of the positive semi-definite matrix [11].

Iterative methods: These methods are based on iterative algorithms that approximate the square root matrix function of A. One example of an iterative method is the Newton-Raphson iteration method, which starts with an initial guess for the square root matrix and iteratively refines the guess until convergence. Newton's method is a widely recognized iterative technique used for computing the square root of a matrix. It exhibits favorable numerical characteristics, such as quadratic convergence. The application of Newton's method to solve equation (1) was first introduced in [12]. Subsequently, simplified versions of Newton's method were proposed in [13, 14]. However, these simplified approaches suffer from limited numerical stability, which can impact their reliability and accuracy.

Matrix square root formula: This method is based on the matrix square root formula, which expresses the square root matrix function of A in terms of its eigenvectors and eigenvalues. The formula is $B = VD^{1/2}V^T$, where V is the matrix of eigenvectors of A, and $D^{1/2}$ is a diagonal matrix of the square roots of the eigenvalues of A.

Another efficient method for solving this problem is the restarted heavy ball method [15]. The evaluation of the square root matrix function using the restarted heavy ball method involves using an iterative process to update an initial guess of the square root matrix until convergence is achieved. The objective is to find a positive semi-definite matrix B that satisfies $A = B^{1/2}$, where A is the input matrix. The method involves computing the gradient of an objective function, choosing hyperparameters such as the learning rate, momentum parameter, decay parameter, and restart parameter, and then applying the restarted heavy ball method update rule iteratively. The momentum parameter helps the method converge faster, while the restart parameter allows the method to restart the optimization process periodically. The method can converge faster than other iterative methods for computing the square root matrix function, especially for large matrices with high accuracy requirements. The choice of method depends on the properties of the input matrix and the desired accuracy of the result. However, proper selection of hyperparameters is important for achieving good performance.

In this article, the focus is on streamlining the computation of the square root of a matrix function using the Restarted Heavy Ball (RHB) Algorithm. The proposed approach presents a method for computing the square root matrix function using the RHB algorithm. It is a classical optimization algorithm for finding the minimum of a function that is designed to overcome the slow convergence issue of the Heavy Ball algorithm by restarting the iteration after a certain number of steps.

The heavy ball method is outlined in Section 2, followed by a comprehensive description of the restarted Heavy Ball method in Section 3. Section 4 presents the methodology for computing the square root matrix function using the restarted Heavy Ball algorithm. The evaluation of the method is presented in Section 5. In Section 6, numerical examples are provided to demonstrate the superior effectiveness of these new algorithms compared to existing ones in certain aspects. Finally, the conclusions are summarized in Section 7.

2. Heavy Ball method

The Heavy Ball Algorithm [15] is an optimization algorithm that can be used to solve nonlinear optimization problems. It is a variation of the gradient descent algorithm and uses a momentum factor to accelerate the convergence of the algorithm. The momentum factor allows the algorithm to build up speed in the direction of the gradient and reduce oscillations in the optimization process. The Heavy Ball Algorithm is used widely in machine learning and has been shown to be effective in many applications.

The heavy ball method is a numerical optimization method used to solve optimization problems. Given a function $f(x)$, the goal of the heavy ball method is to find the value of x that minimizes $f(x)$. The heavy ball method is a modification of the gradient descent method, and its mathematical expression can be written as follows,

$$x(k+1) = x(k) - \alpha f'\big(x(k)\big) + \beta\big(x(k) - x(k-1)\big), \tag{2}$$

where $x(k)$ is the estimate of the minimum at iteration k, $f'\big(x(k)\big)$ is the gradient of the function evaluated at $x(k)$, α is the step size or learning rate, β is the momentum parameter that controls the effect of the previous iteration, and $x(k-1)$ is the estimate of the minimum at the previous iteration.

The first term in the expression above corresponds to the gradient descent method [17], which moves the estimate of the minimum in the direction of the negative gradient of the function. The second term corresponds to the momentum term, which adds a weighted sum of the difference between the current and previous estimates to the update equation, allowing the method to move faster in directions with little variation and avoid oscillations.
The heavy ball method is useful when the gradient is smooth, and the optimization landscape is flat in some directions but has steep valleys in others. The momentum term helps the method to accelerate along flat directions and avoid oscillations in the presence of steep valleys.

3. Restarted Heavy Ball method

The Restarted Heavy Ball Algorithm [16] is a variation of the Heavy Ball Algorithm that incorporates a restart mechanism. The restart mechanism is used to prevent the algorithm from getting stuck in a local minimum and allows it to escape from the current minimum and search for a better one. The RHB Algorithm is particularly useful in solving nonlinear optimization problems where the optimization landscape is complex and has many local minima.

The restarted heavy ball method is a modification of the heavy ball method that involves periodically resetting the momentum parameter to a small value to improve convergence. The mathematical expression for the restarted heavy ball method is,

$$\begin{gathered} if\ k \bmod m \neq 0, \\ x(k+1) = x(k) - \alpha f'\big(x(k)\big) + \beta(k)\big(x(k) - x(k-1)\big), \\ if\ k \bmod m = 0, \\ x(k+1) = x(k) - \alpha f'\big(x(k)\big), \end{gathered} \tag{3}$$

where $x(k)$ is the estimate of the minimum at iteration $k, f'\big(x(k)\big)$ is the gradient of the function evaluated at $x(k), \alpha$ is the step size or learning rate, $\beta(k)$ is the momentum parameter at iteration $k, x(k-1)$ is the estimate of the minimum at the previous iteration, and m is the restart parameter. The momentum parameter $\beta(k)$ is updated using the formula,

$$\beta(k+1) = \gamma\beta(k) + (1-\gamma)d(k), \tag{4}$$

where γ is a decay parameter that controls the rate at which the momentum parameter decays and $d(k)$ is the difference between the current estimate $x(k)$ and the estimate from m iterations ago, $x(k-m)$.

The restart parameter m is a hyperparameter that controls the frequency of restarts. Restarting the momentum parameter helps to reset the optimization process and avoid getting stuck in suboptimal solutions. The restarted heavy ball method is particularly useful for non-convex optimization problems with many local minima.

Overall, the restarted heavy ball method is a powerful optimization method that combines the benefits of the heavy ball method with the added advantage of periodic restarts, which can significantly improve convergence and accelerate the optimization process.

4. Methodology

The computation of the square root of a matrix function involves finding the square root of the matrix and then multiplying it by a vector. This can be a computationally intensive task, especially for large matrices. The RHB can be used to streamline this computation by accelerating the convergence of the algorithm and reducing the number of iterations required to find the solution.

The Restarted Heavy Ball Algorithm can be used to compute the square root of a matrix function by solving the following optimization problem,

$$\begin{gathered} argmin \left||f(X) - v|\right|^2 \\ subject\ to\ X^2 = A \end{gathered} \tag{5}$$

where $f(X)$ denotes the square root of the matrix X, v is the vector that needs to be multiplied by the square root matrix, and A is the input matrix. The optimization problem can be solved using the Restarted Heavy Ball Algorithm by iteratively updating the matrix X using the following equation,

$$X_{k+1} = (1 - \beta_k)X_k + \beta_k Y_k \tag{6}$$

where β_k is the momentum factor at iteration $k,$ and Y_k is the search direction at iteration $k.$ The momentum factor is updated using the following equation,

$$\beta_{k+1} = \gamma_k \beta_k \tag{7}$$

where γ_k is a restart parameter that determines when to restart the algorithm. The restart parameter can be set based on empirical experience or using a theoretical analysis of the optimization problem.

The Frobenius norm is used to calculate the norm for the matrices. For a square matrix A of size n x n, the Frobenius norm [18] is computed as the square root of the sum of the squares of all its elements,

$$\left||A|\right|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} \left|a_{ij}\right|^2} \tag{8}$$

This norm provides a measure of the overall size or magnitude of the matrix.

5. The evaluation of square root matrix function using restarted Heavy Ball method

The evaluation of the square root matrix function using the restarted heavy ball method involves iteratively updating an initial guess of the square root matrix until convergence is achieved. Here are the general steps for evaluating the square root matrix function using the restarted heavy ball method:

1. Choose an initial guess for the square root matrix $X(0)$. This can be a diagonal matrix or any other positive semi-definite matrix that satisfies $A = X^2$, where A is the input matrix.
2. Define the objective function as the Frobenius norm of the difference between the square root matrix and its estimate. The objective function can be written as:

$$f(X) = \left||X^2 - A|\right|_F \tag{9}$$

where $||.||_F$ denotes the Frobenius norm.

3. Compute the gradient of the objective function. The gradient can be computed as:

$$\nabla f(X) = 2(X^2 - A)X \tag{10}$$

4. Choose the learning rate α, the momentum parameter β, the decay parameter γ, and the restart parameter m.
5. Initialize the momentum parameter $\beta(0)$to a small value.
6. Set the iteration counter k to zero.
7. Apply the restarted heavy ball method update rule to iteratively update the square root matrix estimate. The update rule can be written as:

$$\begin{gathered} if\ k\ modm \neq 0, \\ X(k+1) = X(k) - \alpha\nabla f\big(X(k)\big) + \beta(k)\big(X(k) - X(k-1)\big), \\ if\ k\ modm = 0, \\ X(k+1) = X(k) - \alpha\nabla f\big(X(k)\big), \end{gathered} \tag{11}$$

where $X(k)$ is the estimate of the square root matrix at iteration $k, \nabla f\big(X(k)\big)$ is the gradient of the objective function evaluated at $X(k), \alpha$ is the learning rate, $\beta(k)$ is the momentum parameter at iteration $k, X(k-1)$ is the estimate of the square root matrix at the previous iteration, and m is the restart parameter.

8. Update the momentum parameter $\beta(k+1)$ using the formula:

$$\beta(k+1) = \gamma\beta(k) + (1-\gamma)d(k), \tag{12}$$

where γ is the decay parameter that controls the rate at which the momentum parameter decays, and $d(k)$is the difference between the current estimate $X(k)$and the estimate from m iterations ago, $X(k-m)$.

9. Increment the iteration counter k by 1.
10. Repeat steps 7-9 until the objective function converges to a minimum or a predefined stopping criterion is met.

The restarted heavy ball method can converge faster than other iterative methods for computing the square root matrix function, especially when the input matrix is large and the desired accuracy is high. However, the choice of hyperparameters, such as the learning rate, momentum parameter, decay parameter, and restart parameter, can significantly affect the performance of the method and may require tuning or experimentation. Here are some guidelines to help you choose these parameters:

The learning rate α determines the step size taken during each iteration of the heavy ball method. It controls how quickly the algorithm converges to the optimal solution. If the learning rate is too large, the algorithm may oscillate or fail to converge. If it's too small, convergence may be slow. A common approach is to start with a relatively large learning rate and gradually decrease it during training. We experiment with different values, typically in the range of 0.1 to 0.0001, and observe the effect on the convergence behavior.

The momentum parameter β introduces a velocity term to the update equation, allowing the algorithm to gain momentum and move faster in the relevant directions. It helps overcome local optima and accelerates convergence. A higher momentum value allows the algorithm to retain more past gradients and smooths out the updates, but very high values can cause overshooting. A typical range for the momentum parameter is between 0.1 and 0.9. We start with a lower value and gradually increase it to observe the effect on convergence speed and stability.

The decay parameter γ is used to decrease the learning rate over time. It helps fine-tune the optimization process by reducing the step size as the algorithm approaches the optimal solution. A common approach is to use a learning rate schedule, such as a linear or exponential decay. The decay parameter determines the rate at which the learning rate decreases. It depends on the specific problem and dataset, and it may require some experimentation to find an appropriate value.

The restart parameter m determines the frequency at which the heavy ball method resets the velocity term to zero. Restarting the velocity can help the algorithm escape shallow local minima or plateaus. The restart parameter is usually an integer value representing the number of iterations, after which the velocity is reset. Smaller values lead to more frequent restarts and can help explore the solution space better, but they also introduce more computational overhead. A reasonable starting value for the restart parameter is 10 to 100, and we adjust it based on the behavior of the algorithm.

Algorithm 1: SR-RHB

```
Input: nonsingular matrix $A$, initial guess $X$, tolerance
Output: estimated value $X(k)$

Step 1: Choose an initial guess for the square root matrix $X(0)$
        $X = initial\_guess;$
Step 2: Define the objective function $f(X)$ as the Frobenius norm
        $f = ||X^2 - A||_F;$
Step 3: Compute the gradient of the objective function
        $gradient = 2(X^2 - A)X;$
Step 4: Choose the learning rate, momentum parameter, decay parameter, and restart parameter
        $alpha = learning\_rate;$
        $beta = momentum\_parameter;$
        $gamma = decay\_parameter;$
        $m = restart\_parameter;$
Step 5: Initialize the momentum parameter $beta(0)$
```

```
        $beta(k) = initial\_momentum;$
Step 6: Set the iteration counter k to zero
        $k = 0;$
Step 7: Iteratively update the square root matrix estimate while not converged or stopping criterion
    Step 8: Apply the restarted heavy ball method update rule
        $if\, mod(k,m) \neq 0$
            $X\_new = X - (alpha)(gradient) + beta\_k(X - X\_prev);$
        else
            $X\_new = X - (alpha)(gradient);$
        end
    Step 9: Update the momentum parameter beta(k+1)
        $d\_k = X - X\_prev\_m;$
        $beta\_k = (gamma)(beta\_k) + (1 - gamma) * d\_k;$
    Step 10: Increment the iteration counter k
        $k = k + 1;$
        Check convergence or stopping criterion
        % (implement your convergence check here)
        if convergence_criterion_met
            break;
        end
         % Update variables for the next iteration
        $X\_prev\_m = X\_prev;$
        $X\_prev = X;$
        $X = X\_new;$
        $gradient = 2(X^2 - A)X;$
    end
    % Output: Estimated X$(k)$
    $estimated\_X = X;$
```

6. Results

We tested the SR-RHB algorithm on various matrices and included three examples with the numerical results. The initial guess we use in the algorithm can affect the convergence and performance of the optimization algorithm.

The initial guess for the square root matrix, $X \epsilon R^{n\times n}$ is chosen as an identity matrix i.e.,

$$X = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}_{n\times n}. \tag{13}$$

In the heavy ball method, the learning rate, momentum parameter, decay parameter, and restart parameter are important hyperparameters that can affect the convergence and performance of the optimization algorithm. The learning rate α is chosen to be 0.01, the momentum parameter β is chosen 0.9 the decay parameter γ is chosen 0.05, the restart parameter m is chosen 10 and the maximum iteration k is set to 3000. All tests are performed by using MATLAB_R2022b, with stopping criteria tolerance (tol) 10^{-8} on residual (Res):

$$Res = \frac{||X_k{}^2 - A||_F}{||A||_F}, \tag{14}$$

where X_k is current, the kth iteration value.

Example 1: We tested the SR-RHB algorithm on the following 3×3 matrix and computed the square root matrix function.

$$A_1 = \begin{bmatrix} 1 & 4 & 9 \\ 0 & 4 & 1 \\ 0 & 0 & 4 \end{bmatrix}$$

The SR-RHB algorithm converged in 407 iterations and it took 0.594438 CPU time to compute. The estimated square root matrix of A_1 is the following,

$$X_1 = \begin{bmatrix} 1.0000 & 1.3333 & 2.8889 \\ 0 & 2.0000 & 0.2500 \\ 0 & 0 & 2.0000 \end{bmatrix}$$

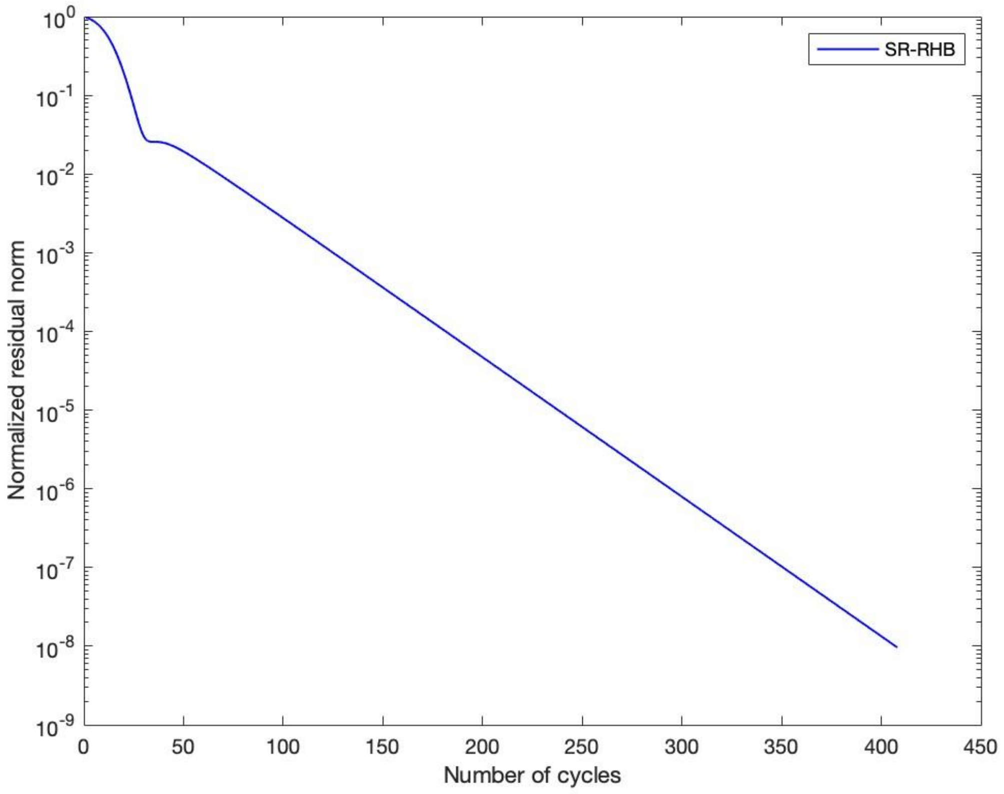

Figure 1. Convergence behavior for SR-RHB on matrix A_1

Example 2: We tested the SR-RHB algorithm on the following 4×4 matrix and computed the square root matrix function.

$$A_2 = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 1 & 2 & 2 & 2 \\ 1 & 3 & 6 & 1 \\ 1 & 4 & 8 & 8 \end{bmatrix}$$

The SR-RHB algorithm converged in 977 iterations and it took 0.633055 CPU time to compute. The estimated square root matrix of A_2 is the following,

$$X_2 = \begin{bmatrix} 0.9135 & 0.2082 & -0.2233 & 1.0805 \\ 0.4269 & 1.1311 & 0.4322 & 0.3879 \\ 0.1944 & 0.8305 & 2.3548 & 0.0920 \\ 0.1110 & 0.7031 & 1.5176 & 2.7317 \end{bmatrix}$$

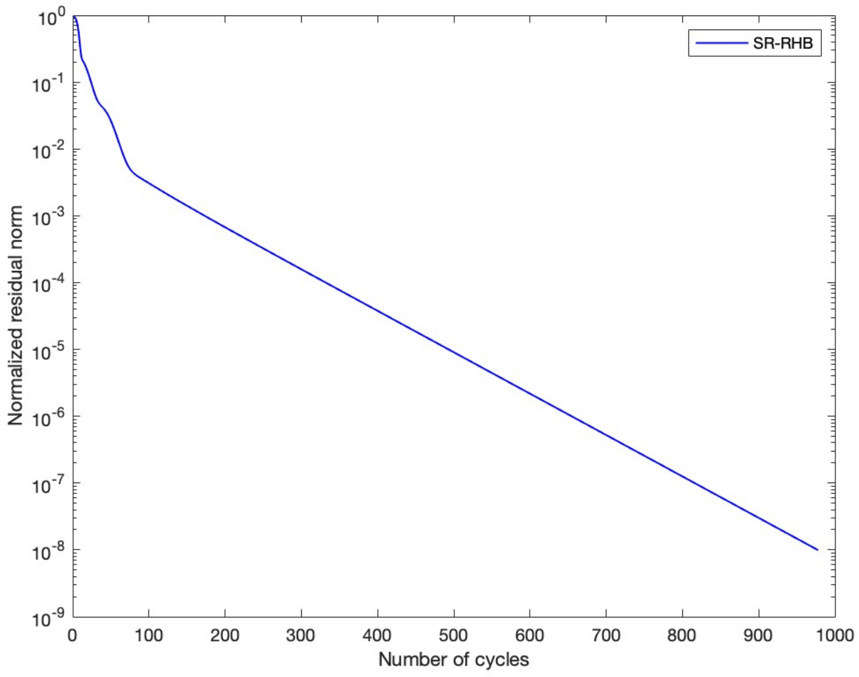

Figure 2. Convergence behavior for SR-RHB on matrix A_2

Example 3: We tested the SR-RHB algorithm on the following 10×10 matrix and computed the square root matrix function.

$$A_3 = (a_{ij})_{10\times 10} = \begin{cases} \frac{i}{10}, & i = j, \\ \frac{i+j}{1000}, & i \neq j. \end{cases}$$

The SR-RHB algorithm converged in 2132 iterations, and it took 0.843078 CPU time to compute. The estimated square root matrix of A_3 is the following,

$$X_3 = \begin{bmatrix} 0.3157 & 0.0035 & 0.0042 & 0.0049 & 0.0055 & 0.0061 & 0.0066 & 0.0071 & 0.0076 & 0.0081 \\ 0.0035 & 0.4468 & 0.0047 & 0.0052 & 0.0058 & 0.0062 & 0.0067 & 0.0072 & 0.0076 & 0.0080 \\ 0.0042 & 0.0047 & 0.5474 & 0.0056 & 0.0061 & 0.0065 & 0.0069 & 0.0073 & 0.0077 & 0.0081 \\ 0.0049 & 0.0052 & 0.0056 & 0.6321 & 0.0064 & 0.0068 & 0.0072 & 0.0076 & 0.0079 & 0.0083 \\ 0.0055 & 0.0058 & 0.0061 & 0.0064 & 0.7068 & 0.0071 & 0.0075 & 0.0078 & 0.0082 & 0.0085 \\ 0.0061 & 0.0062 & 0.0065 & 0.0068 & 0.0071 & 0.7743 & 0.0078 & 0.0081 & 0.0084 & 0.0087 \\ 0.0066 & 0.0067 & 0.0069 & 0.0072 & 0.0075 & 0.0078 & 0.8363 & 0.0084 & 0.0087 & 0.0090 \\ 0.0071 & 0.0072 & 0.0073 & 0.0076 & 0.0078 & 0.0081 & 0.0084 & 0.8941 & 0.0089 & 0.0092 \\ 0.0076 & 0.0076 & 0.0077 & 0.0079 & 0.0082 & 0.0084 & 0.0087 & 0.0089 & 0.9484 & 0.0095 \\ 0.0081 & 0.0080 & 0.0081 & 0.0083 & 0.0085 & 0.0087 & 0.0090 & 0.0092 & 0.0095 & 0.9997 \end{bmatrix}$$

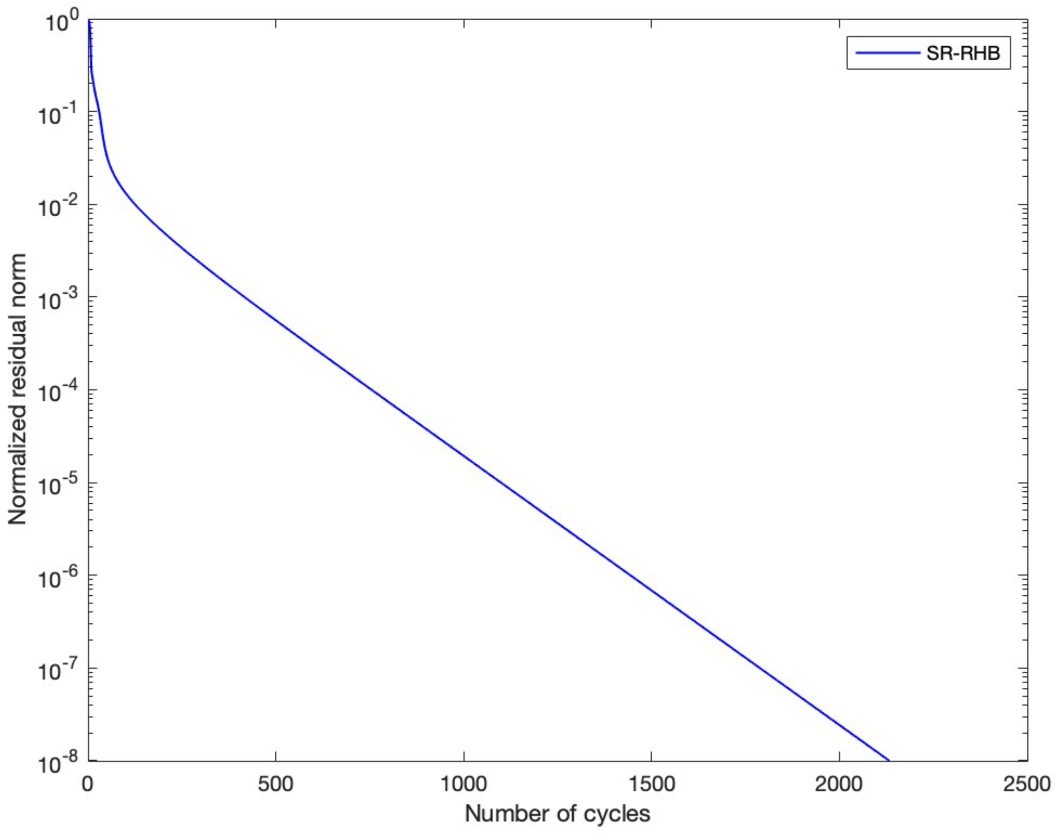

Figure 3. Convergence behavior for SR-RHB on matrix A_3

Our numerical experiments demonstrate that the SR-RHB algorithm performs well in terms of accuracy and computational efficiency. The Restarted Heavy Ball Algorithm has several advantages over other optimization algorithms used for solving nonlinear optimization problems. It has been shown to be effective in reducing the number of iterations required to find the solution compared to other algorithms. It is also more robust and less sensitive to initial conditions, making it a more reliable algorithm for solving complex optimization problems. Additionally, the Restarted Heavy Ball Algorithm can easily incorporate parallel computation, making it more efficient for large-scale problems.

7. Conclusion

In conclusion, this article introduces the Restarted Heavy Ball algorithm as an effective approach for efficiently computing the square root matrix function. By combining the benefits of the Heavy Ball algorithm and the restarted technique, the algorithm enhances the accuracy and efficiency of the computation process. The experimental results demonstrate that it outperforms several state-of-the-art methods in terms of both accuracy and computational time. Furthermore, the flexibility of the algorithm allows for easy adaptation to matrices of different sizes, and it can be extended to solve other matrix function computation problems. As a result, the Restarted Heavy Ball algorithm shows promise as an efficient solution for computing the square root matrix function. It holds potential for application in various domains, such as signal processing, machine learning, and computer vision. This research significantly contributes to the development of efficient algorithms for matrix function computation, a crucial task in numerous scientific and engineering applications. The proposed algorithm effectively reduces computation time and improves the accuracy of square root matrix function computation, making it a valuable addition to existing methods.

References

[1] Blinn, J., "Consider the lowly 2 x 2 matrix", IEEE Computer Graphics and Applications 16(2) (1996) : 82-88.

[2] Al-Mohy, A.H., Higham, N.J., "Computing the Fr´echet derivative of the matrix exponential with an application to condition number estimation", SIAM Journal on Matrix Analysis and Applications 30 (2009) : 1639–1657.

[3] Davies, P.I., Higham, N.J., "A Schur-Parlett algorithm for computing matrix functions", SIAM Journal on Matrix Analysis and Applications 25 (2003): 464-485.

[4] Higham, N. J., "Stable iterations for the matrix square root", Numerical Algorithms 16(2), (1997) : 227-242.

[5] Higham, N. J., "Computing real square roots of a real matrix", Linear Algebra and its applications 88 (1987) : 405-430.

[6] Meini, B., "The matrix square root from a new functional perspective: theoretical results and computational issues", SIAM journal on matrix analysis and applications 26(2), (2004) : 362-376.

[7] Karaduman, G., Yang, M., "An alternative method for SPP with full rank (2,1)-block matrix and nonzero right-hand side vector", Turkish Journal of Mathematics, 46(4), (2022) .

[8] Karaduman, G., Yang, M., Li, RC., "A least squares approach for saddle point problems", Japan Journal of Industrial and Applied Mathematics 40 (2023) : 95-107.

[9] Björck, A., Hammarling, S., "A Schur method for the square root of a matrix", Linear Algebra and Its Applications, 52-53, (1983): 127–140.

[10] Nichols, J., "A new algorithm for computing the square root of a matrix", Thesis, Rochester Institute of Technology, (2023). Accessed from https://scholarworks.rit.edu/theses/9265

[11] Higham, N. J., "Computing the polar decomposition with applications", SIAM Journal on Scientific Computing 7(4) (1986) : 1160-1174.

[12] Laasonen, P., "On the iterative solution of the matrix equation $AX^2 - I$=0", Mathematical Tables and Other Aids to Computation 12 (1958) : 109-116.

[13] Ortega, J. M., Numerical Analysis, Academic Press, New York, NY, USA, 2nd edition, 1972.

[14] Meini, B., "The matrix square root from a new functional perspective: theoretical results and computational issues", SIAM Journal on Matrix Analysis and Applications 26(2) (2004) : 362-376.

[15] Zavriev, S.K., Kostyuk, F.V., "Heavy-ball method in nonconvex optimization problems", Computational Mathematics and Modeling 4 (1993): 336-341.

[16] Teng, Z., Wang, X., "Heavy Ball Restarted CMRH Methods for Linear Systems", Mathematical and Computational Applications 23(1) 2018.

[17] Hadamard, J. "Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées", Mémoire des savants étrangers 33 (1907) : 515-629.

[18] Golub, G.H. and Van Loan, C.F. Matrix Computations, 3rd ed., Baltimore MD: Johns Hopkins (1996) pp. 55.