A bi-orthogonal Jacobi-Davidson method to solve the left and right eigenvectors of a non-Hermitian system

1 Introduction

Eigenvalue problems is one of the most essential and valuable problems in linear algebra. It deals with finding special values and vectors of a square matrix that satisfy a certain equation. The equation is of the form $Ax = \lambda x$, where A is a given matrix, x is an unknown vector, and λ is an unknown scalar. The vector x is called an eigenvector of A, and the scalar λ is called an eigenvector A. The word 'eigen' comes from the German word for 'own' or 'characteristic', meaning that the eigenvectors and eigenvalues capture some essential properties of the matrix.

The eigenvalue problem arises in many applications, such as modelling vibrations, stability, and data analysis. For example, suppose we have a system of masses and springs, and we want to find the natural frequencies and modes of vibration of the system. we can represent the system by a matrix A, where each entry a_{ij} indicate the stiffness of the spring connecting the i^{th} and j^{th} mass. Then, the equation $Ax = \lambda x$ describes how the displacements of the masses x change over time, and the eigenvalues λ are the modes of vibration. By finding the eigenvalues and eigenvectors of A, we can understand how the system behaves and how to control it.

To solve the eigenvalue problem, we must find all the possible values of λ and the corresponding vector of x that make the equation true. There are various methods to do this, depending on the type and the size of the matrix A. One of the most common methods is finding the roots of the characteristic polynomial of A, defined as $p(\lambda) = det(\lambda I - A)$. The determinant of $\lambda I - A$ is a polynomial function of λ, and its roots are precisely the eigenvalues of A. Once we find the eigenvalues, we can find the roots by plugging them into the equation $Ax = \lambda x$ and solving for x. However, this method can be difficult or impossible to carryout by hand, especially for large or complicated matrices. Therefore, we often use numerical algorithms, such as the power method, the QR algorithm, or the Jacobi method, to approximate the eigenvalues and eigenvectors of A.

The eigenvalue problem reveals important properties and insights about the matrix and the system it represents. For example, the number of eigenvalues of A is equal to the dimension of A, and the sum of the eigenvalues of A is equal to the trace of A, which is the sum of the diagonal entries of A. The product of the eigenvalues of A equals the determinant of A, which measures the volume change of A. The eigenvalues and eigenvectors of A also determine the rank, the nullity, the inverse, and the powers of A. Moreover, some special matrices have special eigenvalues and eigenvectors. For example, a symmetric matrix A has only real eigenvalues and orthogonal eigenvectors, meaning that the eigenvectors are perpendicular to each other. A positive definite matrix A has only positive eigenvalues and positive definite eigenvectors, meaning that the eigenvectors point in the same direction as A. These properties make the eigenvalue problem a powerful tool for analyzing and solving linear systems.

Solving the characteristic polynomial, however, has a drawback : the solutions of the characteristic polynomial can change drastically if the polynomial coefficients are slightly altered, even for eigenvalues that are not ill-conditioned. Instead of using this approach, we will use different techniques. Iterative methods help solve eigenvalue problems for large matrices, which are matrices that have a large number of rows and columns. Large matrices are often sparse[1], meaning that most entries are zero and have unique structures, such as symmetry, positive definiteness, or diagonal dominance. These properties can be explained by iterative methods to reduce the computational cost and storage requirements of finding the eigenvalues and eigenvectors of large matrices.

Some of the most common iterative methods for eigenvalue problems are as follows.

1.0.1 Power Method

This is the simplest iterative method. It computes the largest eigenvalue in absolute value and its corresponding eigenvector[2] by repeatedly multiplying a random vector by the matrix and normalizing it. The power method is easy to implement and only requires matrix-vector multiplication, but it converges slowly and may fail if there are multiple eigenvalues of the same magnitude.

1.0.2 Inverse Iteration Method

This is a variant of the power method[3], which computes eigenvalue closest to a given value and its corresponding eigenvector by repeatedly solving a linear system with the matrix shifted by the given value and normalizing the solution. The inverse iteration method can be used to find any eigenvalue, but it requires solving a linear system at each step, which can be costly and unstable.

1.0.3 Rayleigh Quotient Iteration Method

This is another variant of the power method, which computes the eigenvalue and eigenvector of a symmetric matrix by using the Rayleigh Quotient[4], which is the ratio of $x^T Ax$ and $x^T x$, as the shift value for the inverse iteration method. The Rayleigh Quotient iteration method can converge faster than the inverse iteration method, but it is not guaranteed to converge to the desired eigenvalue and may oscillate or diverge.

1.0.4 Arnoldi Method

This is a generalization of the power method, which computes a few eigenvalues and eigenvectors of a matrix by constructing an orthogonal basis for a Krylov subspace, which is the span of successive powers of the matrix applied to a random vector, and then finding the eigenvalues and eigenvectors of a smaller matrix that preserves the action of the original matrix on the subspace. The Arnoldi method[5] can find eigenvalues of any magnitude and multiplicity, but it requires storing and orthogonalizing the basis vectors, which can be expensive and ill-conditioned.

1.0.5 Davidson Method

This is an improvement of the Arnoldi method, which computes a few eigenvalues and eigenvectors of a symmetric matrix by using a pre-conditioner, which is a matrix that approximates the inverse of the matrix, to accelerate the convergence of the Krylov subspace and reduce the size of the smaller matrix. The Davidson method[6] can be more efficient and robust than the Arnoldi method, but it depends on the choice of the pre-conditioner, which can be challenging to construct and apply.

1.0.6 Jacobi-Davidson Method

This is a further improvement of the Davidson method, which computes a few eigenvalues and eigenvectors of a symmetric matrix by using a correction equation[7], which is a linear system that updates the approximate eigenvector by minimizing the residual, to refine the Krylov subspace and the pre-conditioner. The Jacobi-Davidson method can be more accurate and flexible than the Davidson method, but it requires solving a correction equation at each step, which can be challenging and time consuming.

1.0.7 Lanczos Method

This is a special case of the Arnoldi method, which computes a few eigenvalues and eigenvectors of a symmetric matrix by constructing a tri-diagonal matrix that preserves the action of the original matrix on the Krylov subspace. The Lanczos method[8] can be faster and more stable than the Arnoldi method, but it may suffer from loss of orthogonality and spurious eigenvalues due to round-off errors.

2 Davidson Method

The Davidson method is a popular technique to compute a few of the smallest (or largest) eigenvalues of a large sparse real symmetric matrix. It is effective when the matrix is nearly diagonal, that is if the matrix of eigenvectors is close to the identity matrix. It is mainly used for problems of theoretical chemistry (abinitio calculations in quantum chemistry) where the matrices are strongly diagonal dominant. Similar to the Lanczos method, the Davidson method is an iterative method which however does not take advantage of Krylov subspace but uses the Rayleigh-Ritz procedure with non-Krylov spaces and expands the search spaces in a different way. The Block Davidson method is an iterative algorithm for efficiently computing

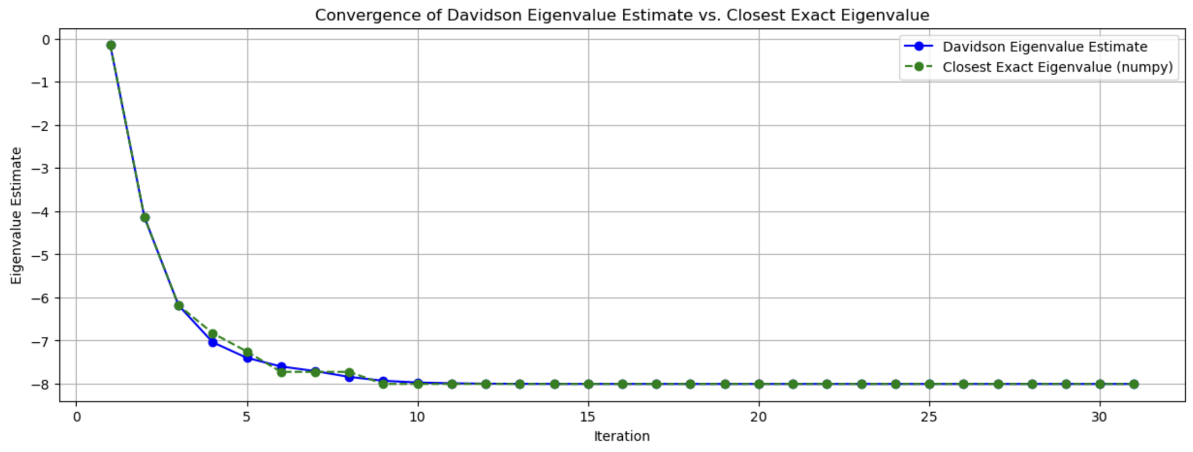


Figure 1: The convergence of the JD eigenvalue to the closest eigenvalue computed by numpy.eig.

a subset of the eigenvalues and corresponding eigenvectors of a large Hermitian matrix A. It starts with a small trial subspace spanned by k initial basis vectors and iteratively expands this subspace to refine the approximation of eigenvalues. At each iteration, the method constructs a reduced projected matrix T by projecting A onto the current subspace and solves for its eigenvalues and eigenvectors. These eigenvalues approximate the desired eigenvalues of A. Residual vectors are computed to assess convergence, and if the residual norms fall below a tolerance ϵ, the process terminates. Otherwise, new basis vectors, derived from the residuals, are orthogonalized and added to the subspace. This approach reduces computational cost

Algorithm 1 Block Davidson Method for Eigenvalue Computation

```
 1: procedure DAVIDSON($A$, $k$, $n_{\text{eigen}}$, $\epsilon$, $N_{\max}$)
 2:     Input:
          • $A \in \mathbb{C}^{n\times n}$: Hermitian matrix
          • $k$: Dimension of trial subspace
          • $n_{\text{eigen}}$: Number of eigenvalues to compute
          • $\epsilon$: Convergence tolerance
          • $N_{\max}$: Maximum number of iterations
 3:     $n \leftarrow \dim(A)$
 4:     if $N_{\max} =$ None then
 5:         $N_{\max} \leftarrow \lfloor \frac{n}{2} \rfloor$
 6:     end if
 7:     if $k =$ None then
 8:         $k \leftarrow 2n_{\text{eigen}}$
 9:     end if
10:     $V \leftarrow [v_1, v_2, \ldots, v_k] \in \mathbb{C}^{n\times k}$: initial basis vectors
11:     $R \leftarrow \text{zeros}(k)$: residual norms
12:     $E \leftarrow$ empty list: eigenvalue estimates
13:     ExactEigenvalues $\leftarrow$ numpy.eigh($A$)
14:     ExactEigenvalues $\leftarrow$ sort(ExactEigenvalues)
15:     for $m = 1, 2, \ldots, N_{\max}$ do
16:         $\{v_j\} \leftarrow$ Orthogonalize($V$)
17:         $T \leftarrow V^*AV \in \mathbb{C}^{k\times k}$: projected matrix
18:         $E, U \leftarrow \text{eig}(T)$: compute eigenvalues and eigenvectors of $T$
19:         $E \leftarrow \text{sort}(E)$: sort eigenvalues
20:         closest_exact_value $\leftarrow$ argmin($|E[1] -$ ExactEigenvalues$|$)
21:         $R[m] \leftarrow |E[1] -$ ExactEigenvalues[closest_exact_value]$|$
22:         $E_{\text{est}}[m] \leftarrow E[1]$
23:         if $R[m] < \epsilon$ then
24:             break: convergence achieved
25:         end if
26:         for $j = 1, \ldots, k$ do
27:             $w \leftarrow (A - E[j]I)VU[:, j]$
28:             $V_{new} \leftarrow \frac{w}{E[j]-A[j,j]}$
29:             $V \leftarrow [V, V_{new}]$                    ▷ Add new vector to basis
30:         end for
31:     end for
32:     Output: Estimated eigenvalues $E[1 : n_{\text{eigen}}]$, residuals $R$
33: end procedure
```

by focusing only on a few eigenvalues, making it particularly suitable for large matrices. Figures 1 and 2 show the convergence of the eigenvalue and the residual with the number of iterations computed using the Jacobi-Davidson algorithm.

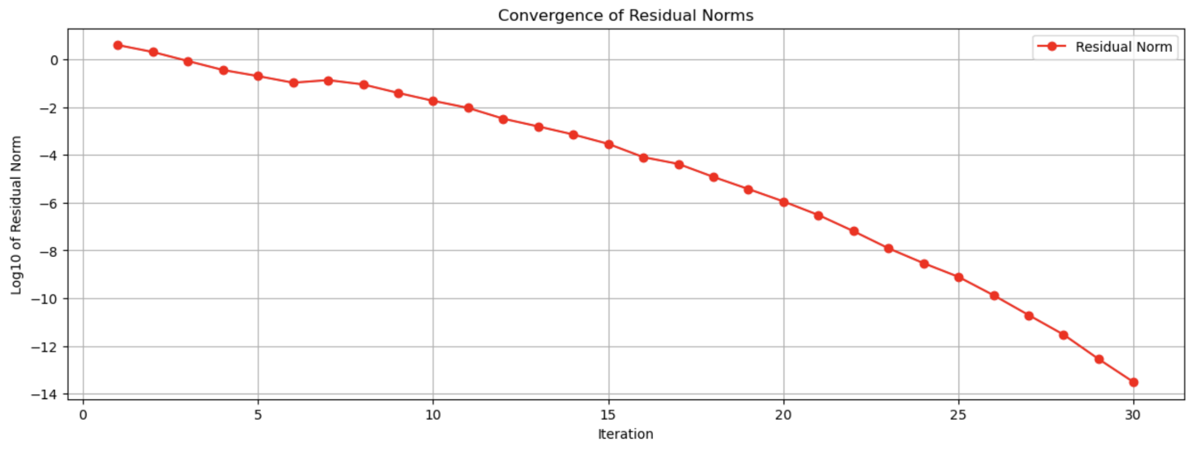


Figure 2: The convergence of the residuals computed by the JD algorithm.

3 Biorthogonal Jacobi Davidson Method

The Biorthogonal Jacobi-Davidson method represents a robust iterative approach[9] for computing selected eigenvalues and eigenvectors of large sparse matrices. Originating from the realm of linear algebra, this method offers superior convergence properties compared to traditional eigenvalue solvers, making it well-suited for large matrices.

At its core, the Bi-Orthogonal Jacobi-Davidson algorithm builds upon the traditional Jacobi-Davidson method, extending it to effectively solve non-Hermitian eigenvalue problems. The principal innovation of BIJD is its ability to maintain bi-orthogonality between the left and right eigenvectors at every stage of the algorithm. Specifically, the algorithm enforces the condition

$$v_L^H v_R = 1 \tag{1}$$

where v_L is the left eigenvector and v_R is the right eigenvector. This requirement ensures that the left and right eigenvectors remain bi-orthogonal throughout the iterative process.

Algorithm 2 Bi-Orthogonal Jacobi-Davidson (BIJD) Algorithm for Non-Hermitian Eigenvalue Problems

```
Require: Matrix $A \in \mathbb{C}^{n\times n}$, initial right eigenvector $v_R \in \mathbb{C}^n$, initial left eigenvector $v_L \in \mathbb{C}^n$, tolerance $\epsilon$, shift $\sigma$, regularization $\rho$, maximum iterations $k_{\max}$
Ensure: Approximate eigenvalue $\lambda$, right eigenvector $v_R$, left eigenvector $v_L$
 1: Normalize the initial guesses: $v_R \leftarrow \frac{v_R}{\|v_R\|}$, $v_L \leftarrow \frac{v_L}{\|v_L\|}$
 2: for $k = 1$ to $k_{\max}$ do
 3:     Compute the bi-orthogonality condition: $s \leftarrow v_L^H v_R$
 4:     if $|s| < \epsilon$ then
 5:         Re-initialize $v_L$ and normalize: $v_L \leftarrow \frac{v_L}{\|v_L\|}$
 6:     end if
 7:     Bi-orthogonalize $v_L$: $v_L \leftarrow \frac{v_L}{s}$
 8:     Compute Rayleigh quotient (eigenvalue estimate): $\lambda \leftarrow v_L^H A v_R$
 9:     Compute right residual: $r_R \leftarrow Av_R - \lambda v_R$
10:     Compute left residual: $r_L \leftarrow A^H v_L - \overline{\lambda} v_L$
11:     if $\|r_R\| < \epsilon$ and $\|r_L\| < \epsilon$ then
12:         Return $\lambda, v_R, v_L$                          ▷ Converged solution
13:     end if
14:     Solve correction equation for right eigenvector:
            $$M_R \leftarrow A - \lambda I + \sigma I + \rho I$$
15:     Apply orthogonal projection:
            $$M_{R,\text{proj}} \leftarrow M_R - v_R(v_R^H M_R)$$
16:     Solve for correction $\delta_R$: $M_{R,\text{proj}}\delta_R = -r_R$
17:     Update right eigenvector: $v_R \leftarrow v_R + \delta_R$
18:     Normalize $v_R$: $v_R \leftarrow \frac{v_R}{\|v_R\|}$
19:     Solve correction equation for left eigenvector:
            $$M_L \leftarrow A^H - \overline{\lambda} I + \sigma I + \rho I$$
20:     Apply orthogonal projection:
            $$M_{L,\text{proj}} \leftarrow M_L - v_L(v_L^H M_L)$$
21:     Solve for correction $\delta_L$: $M_{L,\text{proj}}\delta_L = -r_L$
22:     Update left eigenvector: $v_L \leftarrow v_L + \delta_L$
23:     Normalize $v_L$: $v_L \leftarrow \frac{v_L}{v_L^H v_R}$
24: end for
25: Return $\lambda, v_R, v_L$                                  ▷ Max iterations reached
```

Preconditioning presents challenges when applied to iterative solvers for eigenvalue problems. Initial efforts included variations of the Davidson method and the shift-and-invert method, but Jacobi-Davidson methods have emerged as an approximate preconditioning framework.

Eigenvalue solvers also face significant storage requirements. While linear systems can mitigate storage concerns with the three term recurrence methods like CG and BCG, the Lanczos method for eigenproblems requires storing basis vectors to get back the approximate eigenvectors. Consequently, eigenvalue methods using preconditioning typically necessitate Arnoldi-type methods like Jacobi Davidson because of the lack of Krylov subspace. Due to these factors, the more intricate and resource-intensive procedures like Arnoldi and Jacobi Davidson methods are typically favored over the computationally simpler Lanczos method.

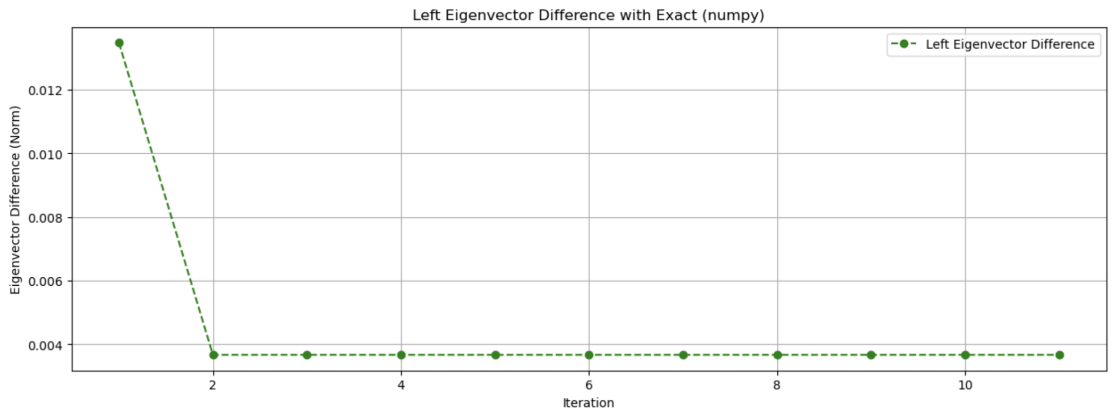
Despite enhanced convergence, Jacobi Davidson may still require numerous steps and extensive storage (as is shown by Fig 1), prompting the use of restarting techniques, triggered when the basis size exceeds a predefined threshold set by the user. These techniques include implicit restarting with various shift strategies and thick restarting, which retains Ritz or Schur vectors[10]. The non-symmetric Lanczos method[11] offers appealing characteristics that could enhance a Jacobi Davidson framework, notably its maintenance of both left and right bi-orthogonal bases generated by A^* and A respectively. The biJD algorithm combines Lanczos two-sided iterations with solving the correction equation for both left and right Ritz pairs, offering faster convergence and an effective restarting scheme.

The BIJD algorithm[12] begins with the initialization of the right and left eigenvector approximations. The user provides an initial guess for the right eigenvector $v_R^{(0)}$ and the left eigenvector $v_L^{(0)}$, which are then normalized to ensure they have unit length. This normalization is crucial because it sets a consistent starting point for the iterations. The algorithm then proceeds to check the bi-orthogonality condition by calculating the dot product between the left and right eigenvectors. If this condition is not satisfied—meaning the dot product deviates from unity—the algorithm will reinitialize the left eigenvector in order to restore the bi-orthogonality.

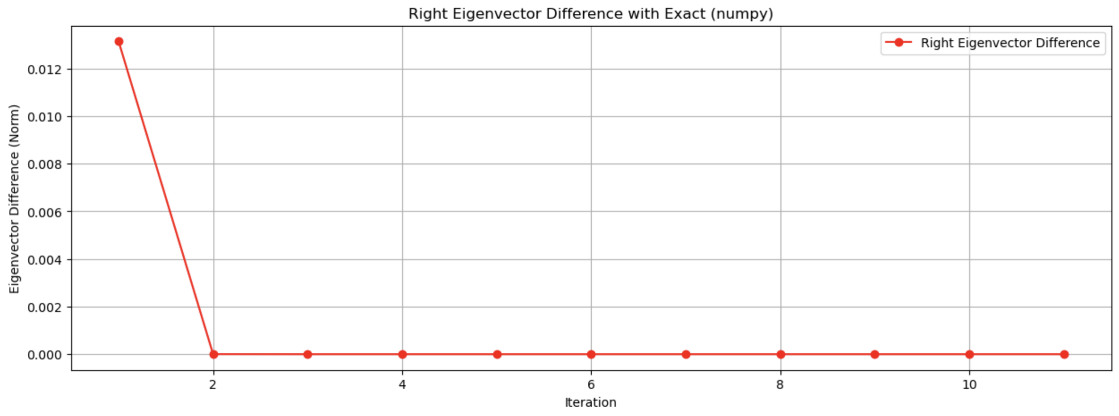
Next, the algorithm computes an estimate of the eigenvalue using the Rayleigh quotient, defined as

$$\lambda^{(k)} = v_L^{(k)H} A v_R^{(k)}. \tag{2}$$

This estimation provides a starting point for the corrections that will be made to the eigenvectors in subsequent iterations. The Rayleigh quotient essentially reflects how well the current approximations for the eigenvectors are capturing the eigenvalue associated with the matrix A. After estimating the eigenvalue, the algorithm calculates the right residual and



(a)



(b)

Figure 3: Convergence of the left and right eigenvectors to that computed from numpy.

the left residual to quantify the accuracy of the current approximations. The right residual is computed as

$$r_R^{(k)} = Av_R^{(k)} - \lambda^{(k)} v_R^{(k)}, \tag{3}$$

and the left residual is given by

$$r_L^{(k)} = A^H v_L^{(k)} - \lambda^{\overline{(k)}} v_L^{(k)}. \tag{4}$$

These residuals serve as measures of how far the current eigenvector approximations deviate from satisfying the eigenvalue equations, and they are tracked throughout the iterations to monitor the algorithm's convergence. Note the convergence of the left and right eigenvector in Fig.3. The convergence is plotted as the difference norm between the eigenvector produced by the biJD algorithm and numpy.eig.

If the residual norms exceed a predetermined tolerance level, indicating that the current approximations are insufficiently accurate, the algorithm proceeds to correct the eigenvector approximations. To accomplish this, the BIJD algorithm sets up correction equations based on the Jacobi-Davidson correction scheme. These correction equations can be expressed for the right eigenvector as

$$M_R \delta_R = -r_R, \tag{5}$$

where M_R is the modified matrix defined as

$$M_R = A - \lambda^{(k)} I + \sigma I + \rho I \tag{6}$$

In this expression, σ serves as a shift parameter intended to prevent singularities, while ϕ is a regularization term that enhances the numerical stability of the algorithm.

A similar equation is established for the left eigenvector:

$$M_L \delta_L = -r_L, \tag{7}$$

where M_L follows the same conceptual framework as M_R but accounts for the left eigenvector.

Once the correction equations are set up, the algorithm solves them to obtain the updates δ_R and δ_L for the right and left eigenvectors, respectively. These updates are then applied to the current approximations, resulting in new eigenvector estimates. To ensure that the bi-orthogonality condition is maintained, the algorithm incorporates an orthogonal projection step, adjusting the updates to prevent loss of the bi-orthogonal structure. After updating the eigenvectors, they are re-normalized to restore their unit length. The process is repeated

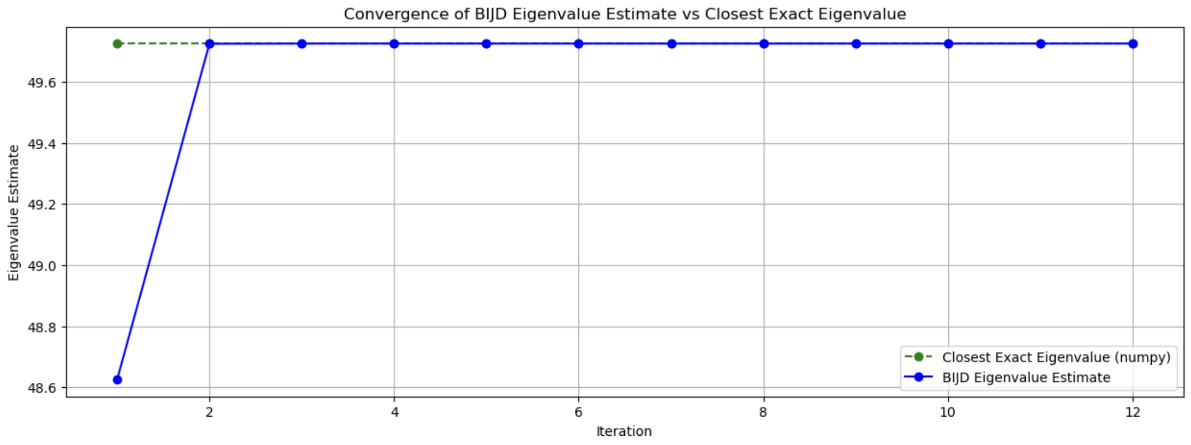


Figure 4: The convergence of the biJD eigenvalue to the closest eigenvalue computed by numpy.eig.

iteratively, with the algorithm checking for convergence at each step. The iterative nature of the BIJD algorithm allows it to refine its eigenvalue and eigenvector estimates gradually until the residual norms fall below the specified tolerance level.

BiJD, however, is computationally more demanding than Jacobi Davidson, requiring matrix-vector multiplication with A^*. While it uses twice as much storage due to the left space W and its image $L = A^*W$, savings can be made by computing residuals explicitly.

One notable advantage of biJD is its ability to obtain left eigenvectors almost effortlessly and with similar accuracy as right ones. These left eigenvectors can be valuable, aiding in deflating converged eigenpairs[13] and estimating the condition number of required eigenvalues. Also, the Rayleigh quotient, easily computed in biJD, serves as a valuable means of assessing eigenvalue convergence. BiJD inherits several characteristics advantageous over

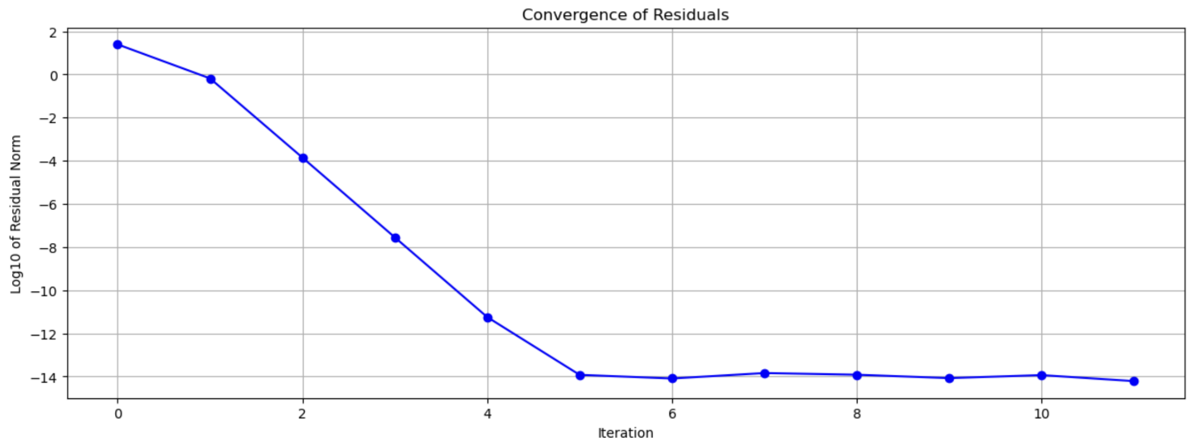


Figure 5: The convergence of the residuals computed from the biJD algorithm.

conventional JD, including explicit biorthogonalization and flexibility in accommodating various restarting techniques.

The biJD algorithm offers the capability to employ spectral projection for deflating converged eigenpairs. By utilizing both left and right eigenpairs, it provides an estimate of the condition number of the target eigenvalue, indicating the reliability of the computed eigenpair. Even before reaching convergence, identifying an ill-conditioned eigenvalue can potentially accelerate the correction equation by approximately mitigating this ill-conditioning through a similarity transformation.

Finally, the code stores eigenvalue estimates at each iteration, which are then compared with the exact eigenvalues computed by numpy.eig. This comparison is valuable, as convergence to the correct eigenvalue is not always guaranteed in non-Hermitian systems. To approximate the left eigenvector for comparison, the code uses $v_L = inv(A).Tv_R$ based on an adjoint relation. While this is a practical approach, a more accurate result may be achievable by calculating the left eigenvector directly when feasible.

Overall, while biJD's use of twice the storage and its computational demands pose challenges, its ability to obtain left eigenvectors and flexibility in restarting techniques make it a promising approach for certain problems. Using the $tracemalloc$ and $process_time$ packages in python we see that biJD is considerably faster (0.01s vs 0.25s) than JD but uses up almost twice the memory to run the program (1630 kB vs 2055 kB). The most expensive step determined by running a memory profiling code was updating and normalizing the right and left eigenvectors.

4 Conclusion

The BIJD method is a robust approach for tackling non-Hermitian eigenvalue problems, especially when standard methods fail to converge or struggle with non-Hermitian matrices. One of the main strengths of this algorithm is its maintenance of bi-orthogonality between the left and right eigenvectors. This property is crucial in non-Hermitian problems, where the left and right eigenvectors may not naturally be orthogonal, which can lead to inaccurate results if not handled properly. Additionally, the algorithm presented in this article uses the Rayleigh quotient as an eigenvalue estimate, leveraging the current eigenvector approximations to refine the eigenvalue estimate at each step. This approach can be effective in reducing the number of iterations required for convergence. The BIJD algorithm can handle large-scale eigenvalue problems by exploiting the Jacobi-Davidson framework, iteratively solving correction equations that make it less sensitive to poor initial guesses compared to basic power or inverse iteration methods. The focus on both right and left residuals further ensures that the algorithm accurately converges to an eigenpair, an important consideration in non-Hermitian problems where residuals for left and right eigenvectors may behave differently.

However, the BIJD algorithm does have some limitations. The Jacobi-Davidson framework requires solving correction equations with projections for both right and left eigenvectors at each iteration, a step that can become computationally intensive for very large matrices due to the need for inversion (or approximate inversion) of matrices near the size of the original problem. Although BIJD is less sensitive than basic methods to initial guesses, its performance still depends on the quality of the initial guesses for v_R and v_L. Poor initial guesses may lead to slower convergence or even convergence to the wrong eigenpair. Additionally, the algorithm relies on the shift σ and regularization ρ parameters to stabilize the correction equation, but these parameters are non-trivial to choose optimally and can vary depending on the problem. Poor choices for these parameters can result in slower convergence or instability. Moreover, the bi-orthogonalization step can be numerically sensitive, particularly when v_L and v_R approach orthogonality (i.e., $v_L^H v_R \approx 0$). While the algorithm mitigates this by re-initializing v_L when bi-orthogonality becomes too weak, this can disrupt convergence and add extra computational overhead.

Some possible improvements to the BIJD algorithm could enhance its performance and stability. Implementing an adaptive shift strategy for the parameters σ and ρ could help stabilize the algorithm and improve convergence rates. For example, dynamically adjusting these values based on the residuals at each step might better handle varying spectral properties of the matrix. Additionally, incorporating a more stable bi-orthogonalization procedure, perhaps through a modified Gram-Schmidt process or QR factorization, could improve numerical robustness in cases where $v_L^H v_R$ is close to zero. For very large matrices, preconditioning techniques could be applied in solving the correction equations, which would improve convergence rates, particularly for matrices with widely spread eigenvalues. Lastly, there are opportunities to parallelize certain steps, such as the bi-orthogonal projections and residual computations. This would make the BIJD algorithm more suitable for high-performance computing applications, particularly for very large non-Hermitian matrices.

References

[1] Owe Axelsson. *Iterative solution methods*. Cambridge university press, 1996.

[2] Xiao-Tong Yuan and Tong Zhang. "Truncated Power Method for Sparse Eigenvalue Problems." In: *Journal of Machine Learning Research* 14.4 (2013).

[3] Elias Jarlebring, Simen Kvaal, and Wim Michiels. "An inverse iteration method for eigenvalue problems with eigenvector nonlinearities". In: *SIAM Journal on Scientific Computing* 36.4 (2014), A1978–A2001.

[4] Benyamin Ghojogh, Fakhri Karray, and Mark Crowley. "Eigenvalue and generalized eigenvalue problems: Tutorial". In: *arXiv preprint arXiv:1903.11240* (2019).

[5] Walter Edwin Arnoldi. "The principle of minimized iterations in the solution of the matrix eigenvalue problem". In: *Quarterly of applied mathematics* 9.1 (1951), pp. 17–29.

[6] Michel Crouzeix, Bernard Philippe, and Miloud Sadkane. "The davidson method". In: *SIAM Journal on Scientific Computing* 15.1 (1994), pp. 62–76.

[7] Gerard LG Sleijpen and Henk A Van der Vorst. "A Jacobi–Davidson iteration method for linear eigenvalue problems". In: *SIAM review* 42.2 (2000), pp. 267–293.

[8] Cornelius Lanczos. "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators". In: (1950).

[9] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[10] Christopher Beattie. "Harmonic Ritz and Lehmann bounds". In: *Electron. Trans. Numer. Anal* 7 (1998), pp. 18–39.

[11] Beresford N Parlett, Derek R Taylor, and Zhishun A Liu. "A look-ahead Lanczos algorithm for unsymmetric matrices". In: *Mathematics of computation* 44.169 (1985), pp. 105–124.

[12] Andreas Stathopoulos. “A case for a biorthogonal Jacobi–Davidson method: Restarting and correction equation”. In: *SIAM Journal on Matrix Analysis and Applications* 24.1 (2002), pp. 238–259.

[13] Andrew Chapman and Yousef Saad. “Deflated and augmented Krylov subspace techniques”. In: *Numerical linear algebra with applications* 4.1 (1997), pp. 43–66.