Transactions on Engineering and Computing Sciences - Vol. 11, No. 1
Publication Date: February 25, 2023
DOI:10.14738/tecs.111.13894.
Ehsaei, M., Nguyen-Thoi, T., Dinh-Cong, D., & Nguyen, D. T. (2023). Parallel Finite Element Procedures for Generalized Eigen-Solution with Improved Damage Detection of Bridge-Type Structures. Transactions on Engineering and Computing Sciences, 11(1). 143-167.

SOCIETY FOR SCIENCE AND EDUCATION

Parallel Finite Element Procedures for Generalized Eigen-Solution with Improved Damage Detection of Bridge-Type Structures

Maryam Ehsaei
Old Dominion University (ODU), CEE Department,
135 KH, Norfolk, VA 23529 (USA)

Trung Nguyen-Thoi
Division of Computational Mathematics
and Engineering, Institute for Computational Science,
Ton Duc Thang University, Ho Chi Minh City, Vietnam
Faculty of Civil Engineering Ton Duc Thang University,
Ho Chi Minh City, Vietnam

Du Dinh-Cong
Division of Computational Mathematics and
Engineering, Institute for Computational Science,
Ton Duc Thang University, Ho Chi Minh City, Vietnam
Faculty of Civil Engineering Ton Duc Thang University,
Ho Chi Minh City, Vietnam

Duc T. Nguyen
Old Dominion University (ODU), CEE Department,
135 KH, Norfolk, VA 23529 (USA)

Abstract

Serial Jacobi transformation algorithm for the solution of standard eigen-problems is re-visited to facilitate the explanation of the proposed parallel transformation algorithm, for which computational efficiency can be realized in this study through pattern recognition for the development and explanation of explicit formulas to avoid costly matrix time matrix operations. The proposed parallel Subspace Iteration and Jacobi Transformation (SI-JT) algorithm for the solution of generalized eigen-problems has also been incorporated into the improved damage identification algorithm. Computational efficiency and robust behaviors for the entire proposed procedures (eigen-solution, damaged localization, and damaged quantification) are validated through several academic and real-life numerical examples. Numerical results obtained in this research work reveal that parallel SI-JT algorithm is more robust as compared to MATLAB eigen-solver. Furthermore, the proposed damage identification method for

damage localization and quantification of bridge-type structures also gives better results than existing methods.

Keywords: Damage identification, Jacobi Transformation, Subspace Iteration, parallel computation, two-phase method, genetic algorithm.

INTRODUCTION

During the past decades, substantial research efforts have been devoted to the development of damage identification techniques for civil engineering structures with both simulation and experimental studies. Based on the comprehensive literature reviews [Carden (2004); Wei Fan and Pizhong Qiao (2011); Das, Saha, and Patro (2016)], vibration-based damage identification (VBDI) approaches have been widely developed and become an important research topic in the fields of civil, mechanical and aerospace engineering. Model-based techniques, a class of VBDI approaches, can be utilized effectively for both damage localization and quantification. In the techniques, an analytical or a numerical model (e.g. finite element methods) is generally required to give eigen-solutions of the monitored structure. As a result, performing eigen-analysis with computational efficiency becomes one of the important factors affecting the effectiveness of this kind of model-based techniques.

For an undamped vibrating structure with N degrees-of-freedoms, the generalized eigen-problem [Bathe (2006); Reddy (2004); Goluh and Van Loan (1996); Nguyen (2006); Heath (2018)] can be described by the following equation:

$$\boldsymbol{K}_{N\times N}\emptyset = \lambda \boldsymbol{M}_{N\times N}\emptyset \quad (1)$$

For solving the above-generalized eigen-problem, efficient solutions such as Subspace Iteration [Bathe (2006); Nguyen (2006)] and Lanczos algorithms [Bathe (2006); Goluh and Van Loan (1996); Nguyen (2006); Heath (2018)] have been well documented in the literature. It should also be mentioned that if the above *N*x*N* "Mass" matrix [**M**] becomes an Identity matrix [Bathe (2006)], then the above-generalized eigen-problem would be simplified to the standard eigen-problem:

$$\mathbf{K}_{N\times N}\emptyset = \lambda\emptyset \quad (2)$$

In Eqs. (1-2), **K**, λ and ∅ represent the system stiffness, eigen-values, and eigen-vectors matrices, respectively. Jacobi transformation/rotation family of algorithms [Bathe (2006); Reddy (2004); Goluh and Van Loan (1996); Nguyen (2006); Heath (2018)] basically transforms the standard/generalized eigen-problem into the diagonal matrix for easily computing all eigen-pairs.

Many researchers [Bathe (2006); Goluh and Van Loan (1996); Nguyen (2006); Heath (2018)] have considered the classical Jacobi rotation algorithm to transform the symmetrical, standard eigen-problem into the diagonal matrix with all eigen-values appeared on its diagonal locations. Sameh and other researchers have extended the above classical (Jacobi rotation) procedure into parallel Jacobi algorithm [Sameh (1971)] by simply demanding several (instead of only

one) off-diagonal terms be driven to zero in each transformation. In Sameh's prior works [Sameh (1971)], however, all eigen-pairs of the standard eigen-problem need to be computed. Bathe and other researchers have incorporated the classical Jacobi transformation into the subspace iteration algorithm [Bathe (2006)] so that only the first few (or all) eigen-pairs can be found for the generalized eigen-problem. Using the subspace iteration algorithm, one can also easily exploit the "sparse" matrix operations [Bathe (2006); Nguyen (2006); Pissanetzky (1984)]. Nevertheless, in Bathe's prior works [Bathe (2006)], only one off-diagonal term at a time can be driven to zero.

The objective of this work is not only to extend the capability of the stand-alone, generalized eigen-solver [as shown in Table 1] but also to incorporate the parallel generalized eigen-solver into practical (real-life) engineering applications such as structural health monitoring. In this present work, first, the Jacobi transformation algorithm is embedded inside the subspace iteration algorithm to calculate the generalized eigen-problem of the monitored structure.

To provide an effective computational procedure, a parallel computing strategy based on the idea of making several off-diagonal terms to be simultaneously driven to zero is used for the Jacobi transformation algorithm, which is the so-called parallel subspace iteration and Jacobi transformation (SI-JT) algorithm. Then, the parallel SI-JT algorithm is incorporated into a two-phase damage identification method for improving the quality of damage assessment results in terms of the accurate solution and computational time. Finally, 2D and 3D bridge-type structures are presented to validate the superior performance of the proposed damage identification approach.

The remaining of this paper will be organized as follows. After the introduction, the classical Jacobi transformation for the solution of the generalized eigen-problem is briefly reviewed in Sec. 2.1. Next, in Sec. 2.2, explicit formulas (based on observed pattern recognitions) for the triple products (matrix times matrix) operations are developed and explained. Parallel computing strategies are presented in Sec. 2.3. Subspace iteration algorithm is summarized in Sec. 2.4 so that only a few lowest eigen-pairs specified by the user can be computed for the generalized eigen-problem, which also shows that the

Table 1. Motivations for this research work

	Standard Eigen-Problem	**Generalized Eigen-Problem**	**Parallel Computation***	**All Eigen-Pairs**	**Few Lowest Eigen-Pairs**	**Sparse**	**Dense**
[Sameh (1971)]	Yes	No	Yes	Yes	No	No	Yes
[Bathe (2006)]	Yes	Yes	No	Yes	Yes	Some	Yes
The present work	Yes	Yes	Yes	Yes	Yes	Yes	Yes

* Several off-diagonal terms can be driven to zero in each transformation

stand-alone "Jacobi transformation" algorithm (presented in Sec. 2.2, and Sec. 2.3) are embedded inside the subspace iteration algorithm. In Sec. 2.5, the superior performance of the parallel SI-JT algorithm is investigated by comparing it to the well-established MATLAB built-in eigen-solver such as the EIG function. Existing damaged detection and damaged quantification are discussed in Sec. 3.1 and Sec. 3.2, for which a simple rule of thumb is proposed in this work to improve the quality of damaged detection in bridge structures. Several numerical examples are presented in Sec. 3.3 to validate our claim for improving the quality of damaged identification as compared to recently published algorithms. Finally, some conclusions are highlighted in Sec. 4.

CLASSICAL JACOBI TRANSFORMATION AND GENERALIZED EIGEN-PROBLEM

In the well-documented (classical) Jacobi transformation method, the original "stiffness" matrix [**K**] and "mass" matrix [**M**] in Eq. (1) can be repeatedly transformed into diagonal matrices, [**K***] and [**M***] respectively, through the Jacobi transformation as shown in Eqs. (3-4)

$$\mathbf{K}^* = \mathbf{P_1}^T \mathbf{K}\, \mathbf{P_1} \tag{3}$$

$$\mathbf{M}^* = \mathbf{P_1}^T \mathbf{M}\, \mathbf{P_1} \tag{4}$$

where **K** and **M** matrices are symmetrical. The rotation (or transformation) matrix [$\mathbf{P}_1$] can be defined as:

$$\mathbf{P}_1{}^T = \begin{bmatrix} 1 & \theta_1 & 0 & 0 \\ \theta_2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

In Eq. (5), we have assumed that the new off-diagonal terms for matrix $\mathbf{K}^*$ at the location (p, q) = (1, 2) to be driven to zero through the transformation shown in Eqs. (3-4). θ_1 and θ_2 are the 2 unknowns, which can be solved by applying the following equation:

$$K_{12}{}^* = 0, \text{ and } M_{12}{}^* = 0 \tag{6}$$

A Review of Jacobi Transformation for the Solution of the Generalized Eigen-Problem

The following derivation is valid, when K_{12} is intended to become zero. For the general case, two unknowns should be placed in K_{ij} and K_{ji} locations.

$$\mathbf{K}^* = \mathbf{P}_1{}^T \mathbf{K}\, \mathbf{P}_1 = \begin{bmatrix} 1 & \theta_1 & 0 & 0 \\ \theta_2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix} \begin{bmatrix} 1 & \theta_2 & 0 & 0 \\ \theta_1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$= \begin{bmatrix} 1 & \theta_1 & 0 & 0 \\ \theta_2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_{11} + k_{12}\theta_1 & k_{11}\theta_2 + k_{12} & k_{13} & k_{14} \\ k_{21} + k_{22}\theta_1 & k_{21}\theta_2 + k_{22} & k_{23} & k_{24} \\ k_{31} + k_{32}\theta_1 & k_{31}\theta_2 + k_{32} & k_{33} & k_{34} \\ k_{41} + k_{42}\theta_1 & k_{41}\theta_2 + k_{42} & k_{43} & k_{44} \end{bmatrix} \tag{8}$$

After performing the triple products shown in Eq. (8), one obtains:

$\mathbf{K}^*$ =

$(\ k_{11}+k_{12}\theta_1\ )\ +\ \theta_1(k_{21}+k_{22}\theta_1)$	$(\ k_{11}\theta_2+k_{12}\ )\ +\ \theta_1(k_{21}\theta_2+k_{22})$	$k_{13}+\theta_1 k_{23}$	$k_{14}+\theta_1 k_{24}$
$sym.$	$\theta_2\ (\ k_{11}\theta_2+k_{12}\ )\ +(k_{21}\theta_2+k_{22})$	$\theta_2\ k_{13}+k_{23}$	$\theta_2 k_{14}+k_{24}$
$sym.$	$sym.$	k_{33}	k_{34}
$sym.$	$sym.$	$sym.$	k_{44}

(9)

Thus,

$$K^*_{1,2} = 0 = (k_{11}\theta_2 + k_{12}) + \theta_1(k_{21}\theta_2 + k_{22}) \quad (10)$$

$$M^*_{1,2} = 0 = (M_{11}\theta_2 + M_{12}) + \theta_1(M_{21}\theta_2 + M_{22}) \quad (11)$$

From Eqs. (10) and (11), one can find:

$$\theta_1 = \frac{-(k_{11}\theta_2+k_{12})}{(k_{21}\theta_2+k_{22})} = \frac{-(M_{11}\theta_2+M_{12})}{(M_{21}\theta_2+M_{22})} \quad (12)$$

From Eq. (12), θ_2 can be computed as shown in the following

$$(k_{11}\theta_2 + k_{12})\,(M_{21}\theta_2 + M_{22}) = (k_{21}\theta_2 + k_{22})\,(M_{11}\theta_2 + M_{12}) \quad (13)$$

$$\begin{aligned} &k_{11}M_{21}\theta_2{}^2 + (k_{11}M_{22}+k_{12}M_{21})\,\theta_2 + (k_{12}M_{22}) \\ &= k_{21}M_{11}\theta_2{}^2 + (k_{21}M_{12}+k_{22}M_{11})\,\theta_2 + (k_{22}M_{12})\,(k_{11}M_{21} - k_{21}M_{11})\,\theta_2{}^2 \\ &+ (k_{11}M_{22}+k_{12}M_{21} - k_{21}M_{12} - k_{22}M_{11})\,\theta_2 + (k_{12}M_{22} - k_{22}M_{12}) = 0 \end{aligned} \quad (14)$$

The above Eq. (14) can be expressed as:

$$(A_1)\theta_2{}^2 + (B_1)\theta_2 + (C_1) = 0 \quad (15)$$

Hence,

$$\theta_2 = \frac{-B_1 \pm \sqrt{B_1{}^2 - 4A_1C_1}}{2A_1} \qquad \text{(assuming } A_1 \neq 0) \quad (16)$$

In Eq. (16), if the denominator $A_1 = 0$, from (Eq. (15) one obtains:

$$\theta_2 = -C_1 / B_1 \quad (17)$$

Finally, θ_1 can be found from Eq. (12). The sign in front of the SQRT of Eq. (16) will be based on the sign of $\bar{k}$, defined as below

$$\bar{k} = B_1 = (k_{11}M_{22}+k_{12}M_{21} - k_{21}M_{12} - k_{22}M_{11}) \quad (18)$$

After computing θ_2 [see Eq. (16), or Eq. (17)], and θ_1 [see Eq. (12)], matrix $\boldsymbol{P}_1^T$ can be generated as described in Eq. (5).

In the following steps, explicit formulas for the matrices **K*** in Eq. (3) and **M*** in Eq. (4) should be developed. In the transformed matrices, we assume the selected off-diagonal terms ($k_{12} = k_{21}, and\ m_{12} = m_{21}$) should be driven to zero. The above procedure will be repeated until all the off-diagonal terms become zero.

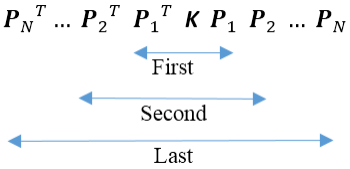
From Ref. [Bathe (2006)], one has:

$$\boldsymbol{P}_N^T \dots \boldsymbol{P}_2^T \boldsymbol{P}_1^T \boldsymbol{K} \boldsymbol{P_1} \boldsymbol{P_2} \dots \boldsymbol{P_N} = \boldsymbol{K}^* \quad (19)$$

In Eq. (19), the matrix **K*** eventually becomes a diagonal (eigen-value) matrix, where N is the size of $\mathbf{K}_{N\times N}$. Furthermore, the eigen-vectors matrix can also be identified from Eq. (19) [Bathe (2006); Nguyen (2006)]:

$$\boldsymbol{P_1} \boldsymbol{P_2} \dots \boldsymbol{P_N} = \emptyset \quad (20)$$

Based on Ref. [Sameh (1971)], more than one off-diagonal terms can be driven to zero, which will also be adopted in this work. The most time-consuming part of the Jacobi Rotation procedure is the computation which involves repeated matrix times matrix operations.



In this work, however, expensive matrix times matrix operations can be avoided by recognizing the pattern of "explicit formulas" for $\boldsymbol{P}_{i+1}^T \boldsymbol{K} \boldsymbol{P}_{i+1}$, which will be explained in greater detail in the next section.

Development of Explicit Formulas for Triple Matrix Times Matrix Operations

We have observed that there are specific patterns in the result of $\mathbf{K}^* = \boldsymbol{P}_1^T \mathbf{K} \boldsymbol{P}_1$ [see Eq. (9)], which will be repeated in every step of the procedure. First of all, the changes in matrix **K*** (as compared to matrix **K**) only happens in the terms associated with the related rows and columns of matrix **K** (i^{th} row and j^{th} column for the selected K_{ij}), which will become zero, after the Jacobi transformation step $\mathbf{K}^* = \boldsymbol{P}_1^T \mathbf{K} \boldsymbol{P}_1$ is completed.

For better explanation, assuming that K_{12} [or K_{pq}, where p=1, and q=2] is selected to become zero after the Jacobi transformation. For the pairs (p, q), it can be described as. The "companion" row for "row p" is "row q", and the "companion" row for "row q" is "row p". The "companion" column for "column p" is "column q", and the "companion" column for "column q" is "column p". Recalled Eqs. (7-8), $\boldsymbol{P}_1$ can be defined as Eq. (5). Based on Eqs. (3), (4) and (9), $\mathbf{K}^*$ can be computed as described in Eq. (9). In general, there are three types of terms in the transformed matrix $K_{\mathrm{ij}}{}^*$, as shown in Eq. (9). The explicit formulas for each term $K_{\mathrm{ij}}{}^*$ can be developed based on the observed patterns, which can be described in the following paragraphs:

Type 1: All K_{ij} terms, which none of the indexes are either p=1 or q=2 (such as K_{33}, K_{34} and K_{44}). In other words, all terms $K_{\mathrm{ij}}{}^*$ for which $i \neq p$, and $j \neq q$. It should be noted that these (type 1) terms will not change after the triple product matrix multiplications ($\mathbf{K}^* = \boldsymbol{P}_1{}^T \mathbf{K} \boldsymbol{P}_1$) and remain the same values.

Type 2: All K_{ij} terms which only one of the indexes are either p=1 or q=2 (such as K_{13}, K_{14}, K_{23}, and K_{24}). In other words, all terms $K_{\mathrm{ij}}{}^*$ for which i is either p or q and $j \neq p$ and $j \neq q$. These $K_{\mathrm{ij}}{}^*$ terms can be computed based on the following explicit formula:

$$K_{\mathrm{ij}}{}^* = K_{ij} + \theta_{\mathrm{m}} * \mathbf{K} \tag{21}$$

where the subscript m can be found by looking at the "companion" row for ith row of the rotation matrix $\boldsymbol{P}_1$. Based on the explicit formula shown in Eq. (21), we can compute:

$$K_{13}{}^* = K_{13} + \theta_{\mathrm{m}} * K_{23} \tag{22}$$

From Eq. (22), we can easily identify $\theta_{\mathrm{m}} = \theta_1$. Thus,

$$K_{13}{}^* = K_{13} + \theta_1 * K_{23} \tag{23}$$

Similarly, we can compute:

$$K_{24}{}^* = K_{24} + \theta_{\mathrm{m}} * K_{14} \tag{24}$$

Also, we can easily find $\theta_{\mathrm{m}} = \theta_2$, and thus,

$$K_{24}{}^* = K_{24} + \theta_2 * K_{14} \tag{25}$$

Type 3: All K_{ij} terms, which both indexes are either p=1 or q=2 (such as K_{11}, K_{12} and K_{22}). In other words, all terms $K_{\mathrm{ij}}{}^*$ for which i is either p or q and j = either p or q. These $K_{\mathrm{ij}}{}^*$ terms can be computed based on the following 2 steps.

Step 3.1: In this step, we follow the same procedure as explained in Type 2.

$$K_{12}{}^{*} = K_{12} + \theta_{\mathrm{m}} * K_{22} \quad (26)$$

By referring to the 2nd row of the matrix $\boldsymbol{P}_1$, we obtain the proper subscript m for θ_{m}, and thus

$$K_{12}{}^{*} = K_{12} + \theta_1 * K_{22} \quad (27)$$

Step 3.2: By replacing K_{12}(as shown in Eq. (27)), one can find as

$$\begin{aligned} K_{12} &= K_{12} + \theta_r * \mathbf{K} \\ &= K_{12} + \theta_r * K_{11} \end{aligned} \quad (28)$$

where the subscript r of θ can be obtained by referring to the 1st column of the matrix $\boldsymbol{P_1}^T$. Thus,

$$K_{12} = K_{12} + \theta_2 * K_{11} \quad (29)$$

Similarly, by replacing K_{22} (as shown in Eq. (27)), we have the following formulas

$$K_{22} = K_{22} + \theta_{\mathrm{s}} * \mathbf{K} \quad (30)$$

$$K_{22} = K_{22} + \theta_{\mathrm{s}} * K_{21} \quad (31)$$

where the subscript s of θ can be obtained by referring to the 1st column of the matrix $\boldsymbol{P_1}^T$. Thus,

$$K_{22} = K_{22} + \theta_2 * K_{21} \quad (32)$$

Finally, substituting Eqs. (29, 32) into Eq. (27), one obtains

$$K_{12}{}^{*} = \{ K_{12} + \theta_2 * K_{11} \} + \theta_1 * \{ K_{22} + \theta_2 * K_{21} \} \quad (33)$$

Similar procedures can be used to compute $K_{11}{}^{*}$, and $K_{22}{}^{*}$ for these type 3 terms of matrix [**K***].

Parallel Computing Strategies for Jacobi Transformation Algorithm

Sameh (1971) presented an algorithm that can zero-out several off-diagonal terms (row "*p*", column "*q*") simultaneously for the standard eigen-problem. This idea can also be applied for the generalized eigen-problem", where p and q are sequences defined by Ref. [Sameh (1971)], for which p & q can be swapped so that p is less than q. The algorithm (to systematically identify all the off-diagonal locations (p, q) of matrix [**K**]) presented in Ref. [Sameh (1971)] can be conveniently summarized here as follows

For k = 1, 2, …, m -1 [where $m = n / 2$; and k = step #]

$$q = m - k + 1,\ m - k + 2,\ \ldots,\ n - k,$$

$p = (2m - 2k + 1) - q$, if $m - k + 1 \le q \le 2m - 2k$
$p = (4m - 2k) - q$, if $2m - 2k < q \le 2m - k - 1$
$p = n$, if $2m - k - 1 < q$

For k = m, m+1, … , 2m-1

$q = 4m - n - k, 4m - n - k + 1, \dots, 3m - k - 1,$
$p = n,$ if $q < 2m - k + 1$
$p = (4m - 2k) - q,$ if $2m - k + 1 \le q \le 4m - 2k - 1$
$p = (6m - 2k - 1) - q,$ if $4m - 2k - 1 < q$

The extension of Ref. [Sameh (1971)] for the parallel transformation of generalized eigen-problems is described in the following.

The generalized eigen-equations, see Eq. (1), can be re-stated as

$$[\mathbf{K}_{N\times N}]\,[\emptyset] = [\lambda]\,[\mathbf{M}_{N\times N}][\phi] \tag{34}$$

where **K** is a symmetrical positive definite (SPD) stiffness matrix.

Assuming we want to make the off-diagonal terms of matrices **K*** in Eq. (3) and **M*** in Eq. (3), at locations $(p, q) = (1, 2)$ & $(p, q) = (3, 4)$, to be driven to zero. Thus,

$$P_1{}^T = \begin{bmatrix} 1 & \theta_1 & 0 & 0 \\ \theta_2 & 1 & 0 & 0 \\ 0 & 0 & 1 & \theta_3 \\ 0 & 0 & \theta_4 & 1 \end{bmatrix} \tag{35}$$

In Eq. (37), the 4 unknowns θ_1, θ_2, θ_3 and θ_4 can be solved by employing 4 associated equations $K_{12}{}^* = 0 = K_{34}{}^* = M_{12}{}^* = M_{34}{}^*$, and using similar explicit formulas developed in Sec. 2.2.

Subspace Iteration

Subspace Iteration and Lanczos Algorithms [Bathe (2006); Reddy (2004); Goluh and Van Loan (1996); Nguyen (2006); Heath (2018)] have been used extensively in the engineering communities for solving the generalized eigen-problem. The details of Subspace Iteration algorithm are presented in the following step-by-step procedure.

Step 1: Guess $[\boldsymbol{X}_k]_{N\times L}$, where $L \ll N$ and $L \approx$ (2 to 4) × (# lowest eigen-pairs desired)

Step 2:

$$[\,\mathbf{K}\,]\,\overline{\boldsymbol{X}}_{k+1} = [\mathbf{M}]\,\boldsymbol{X}_k \tag{36}$$

One can solve $[\overline{\boldsymbol{X}}_{k+1}]$ by sparse equation solver [Goluh and Van Loan (1996); Heath (2018); Pissanetzky (1984)].

Step 3: Solve the following reduced stiffness and reduced mass matrices

$$[\mathbf{K}_{Reduce}]_{L\times L} = [\overline{\boldsymbol{X}}_{k+1}^{T}]_{L\times N}\,[\mathbf{K}]_{N\times N}\,[\overline{\boldsymbol{X}}_{k+1}]_{N\times L}$$
$$[\mathbf{M}_{Reduce}]_{L\times L} = [\overline{\boldsymbol{X}}_{k+1}^{T}]_{L\times N}\,[\mathbf{M}]_{N\times N}\,[\overline{\boldsymbol{X}}_{k+1}]_{N\times L} \quad (37)$$

Step 4: Solve for all eigen-pairs of the generalized reduced eigen-problem [see Jacobi transformation with explicit formulas in Sec. 2.2]

$$[\mathbf{K}_{\mathrm{R}}]_{L\times L}[\mathrm{E_Vectors}]_{L\times L} = [\mathrm{E_Values}]_{L\times L}\,[\mathbf{M}_{\mathrm{R}}]_{L\times L}\,[\mathrm{E_Vectors}]_{L\times L} \quad (38)$$

Step 5: Updating the guessed (eigen-vector) matrix [**X**]

$$[\mathbf{X}_{\mathrm{k+1}}]_{N\times L} = [\overline{\mathbf{X}}_{\mathrm{k+1}}]_{N\times L} \times [\mathrm{E_Vectors}]_{L\times L} \quad (39)$$

In this process, the convergence criteria are based on both the absolute error norm, which has to be less than 10^{-7}, and the difference of eigen pairs in two consecutive iterations to be less than 10^{-3}. If converged, the subspace iteration process will be stopped. Otherwise, it will be returned to Step 2, with $\mathbf{X}_{\mathrm{k}}$ to be replaced by $\mathbf{X}_{\mathrm{k+1}}$.

Numerical Examples Using Parallel Subspace Iteration and Jacobi Transformation for Eigen- Problems

In this Section, several illustrative test examples are used to evaluate the performance of the proposed parallel subspace iteration and Jacobi transformation (SI-JT) algorithm. All the examples are real-world eigenvalue problems. The results for eigen-solutions, and wall-clock time are reported in Tables 2-5. From the Tables, the results indicate that the performance of the parallel SI-JT algorithm is superior to MATLAB built-in function.

Table 2. Size (2000x2003) Fluid Flow "EIG" solution time and solution accuracy

Requested Eigenvalue	Parallel SI-JT algorithm	MATLAB "EIG"
2	2.449043 (9 subspace iteration)	2.458226 (9 subspace iteration)
4	2.742210 (8 subspace iteration)	2.454659 (8 subspace iteration)
10	6.689142 (8 subspace iteration)	Not converged
27	102.968629 (8 subspace iteration)	Not converged
63	2027.442472 (7 subspace iteration)	Not converged

Table 3. Size (1086x1086) Buckling of Hot Washer "EIG" solution time and solution accuracy

Requested Eigenvalue	Parallel SI-JT algorithm	MATLAB "EIG"
2	1.916372 (9 subspace iteration)	1.650959 (27 subspace iteration)
4	2.348291 (11 subspace iteration)	1.449550 (20 subspace iteration)
10	5.795341 (8 subspace iteration)	Not converged
27	98.628914 (8 subspace iteration)	Not converged
63	2016.937281 (7subspaceiteration)	Not converged

Table 4. Size (420x420) Lumped Mass "EIG" solution time and solution accuracy

Requested Eigenvalue	Parallel SI-JT algorithm	MATLAB "EIG"
2	0.199795 (9 subspace iteration)	0.092030 (11 subspace iteration)
4	0.376640 (7 subspace iteration)	0.091801 (7 subspace iteration)
10	4.459468 (7 subspace iteration)	Not converged
20	1405.341492 (7 subspace iteration)	Not converged

Table 5. Size (153x153) Transmission Tower "EIG" solution time and solution accuracy

Requested Eigenvalue	Parallel SI-JT algorithm	MATLAB "EIG"
2	0.078787 (4 subspace iteration)	0.010875 (4 subspace iteration)
4	0.316909 (7 subspace iteration)	0.020544 (9 subspace iteration)
10	3.417312 (8 subspace iteration)	Not converged
20	30.654872 (7 subspace iteration)	Not converged
27	1602.563036 (8 subspace iteration)	Not converged
28	2690.135469 (8 subspace iteration)	Not converged

For further investigation, parallel performance and time comparison for the different number of requested eigen-pairs using a various number of processors is shown in Tables 6-9. This example is a real-world fluid flow eigen-value problem, in which the stiffness matrix is a module of an offshore platform found at the website https://www.cise.ufl.edu/research/sparse/matrices/HB/.

The stiffness matrix has about 4000 rows and 4000 columns. It is a sparse, symmetric positive definite matrix that is a structural full rank matrix. A high number of components make it time-consuming for non-parallel algorithms to solve and order the eigen-pairs such matrix. However,

by using the proposed algorithm a few numbers of eigen-pairs can be found in a reasonable amount of time.

Table 6. Fluid flow generalized eigenvalues, symmetric stiffness matrix

Requested Eigenvalue	ESI 1 processor (sec)	1 processor time (sec)	ESI 2 processors (sec)	2 processor time (sec)	ESI 3 processors (sec)	3 processors time (sec)	ESI 4 processors (sec)	4 processors time (sec)
66	896.456	895.378	603.306	602.346	599.162	598.197	542.308	541.303
	1102.823	1101.811	730.748	729.746	725.860	724.858	667.135	666.160
	1261.488	1260.468	856.725	855.684	815.687	814.700	763.603	762.586
	895.841	894.852	608.521	607.506	591.797	590.831	541.680	540.677
	590.352	589.356	389.603	388.606	386.428	385.435	350.564	349.564
	422.939	421.939	280.349	279.366	281.493	280.537	254.538	253.555
Average	861.650	860.634	578.209	577.209	566.738	565.760	519.971	518.974
Speed Ratio		1		1.491		1.521		1.658
Efficiency		100%		75%		51%		41%

* ESI : Entire Subspace Iteration time in conjunction with MATLAB eigen-solver.
1 (or 2, or 3, or 4) processors time: Entire Subspace Iteration time in conjunction with our proposed Jacobi solver.

Table 7. Fluid flow generalized eigenvalues, symmetric stiffness matrix

Requested Eigenvalue	ESI 1 processor (sec)	1 processor time (sec)	ESI 2 processors (sec)	2 processor time (sec)	ESI 3 processors (sec)	3 processors time (sec)	ESI 4 processors (sec)	4 processors time (sec)
77	831.959	830.658	713.884	712.550	681.975	680.614	606.456	605.202
	1051.925	1050.620	880.492	879.019	817.824	816.489	752.660	751.401
	1048.577	1047.275	865.130	863.854	863.766	862.428	770.184	768.925
	743.937	742.619	607.148	605.874	617.981	616.640	550.740	549.484
	523.077	521.781	426.450	425.162	415.185	413.828	390.863	389.598
	436.999	435.712	348.093	346.781	343.968	342.706	326.717	325.461
	393.060	391.798	334.650	333.348	309.103	307.825	293.609	292.370
Average	718.505	717.209	596.550	595.227	578.543	577.219	527.318	526.063
Speed Ratio		1		1.205		1.243		1.363
Efficiency		100%		60%		41%		34%

Table 8. Fluid flow generalized eigenvalues, symmetric stiffness matrix

Requested Eigenvalue	ESI 1 processor (sec)	1 processor time (sec)	ESI 2 processors (sec)	2 processor time (sec)	ESI 3 processors (sec)	3 processors time (sec)	ESI 4 processors (sec)	4 processors time (sec)
100	1789.793	1788.016	1183.647	1182.114	1089.224	1087.593	1057.478	1055.889
	2187.909	2186.397	1482.388	1480.841	1304.323	1302.712	1346.470	1344.895
	2011.821	2010.273	1363.752	1362.213	1320.356	1318.706	1222.838	1221.249
	1751.386	1749.810	1144.917	1143.398	1056.208	1054.590	1066.127	1064.553
	1235.838	1234.151	801.335	799.792	840.833	839.215	758.246	756.672
	982.869	981.287	631.880	630.311	679.718	678.120	586.748	585.172
	805.975	804.442	517.551	515.989	571.278	569.618	485.437	483.871
Average	1537.942	1536.339	1017.924	1016.380	980.277	978.650	931.906	930.329
Speed Ratio		1		1.512		1.570		1.651
Efficiency		100%		75%		52%		41%

* ESI : Entire Subspace Iteration time in conjunction with MATLAB eigen-solver.
1 (or 2, or 3, or 4) processors time: Entire Subspace Iteration time in conjunction with our proposed Jacobi solver.

Table 9. Fluid flow generalized eigenvalues, symmetric stiffness matrix

Requested Eigenvalue	ESI 1 processor (sec)	1 processor time (sec)	ESI 2 processors (sec)	2 processor time (sec)	ESI 3 processors (sec)	3 processors time (sec)	ESI 4 processors (sec)	4 processors time (sec)
130	4674.505	4672.471	2914.734	2912.711	2798.486	2796.439	2694.348	2692.321
	5317.979	5315.945	3324.582	3322.546	3211.935	3209.894	3022.486	3020.436
	5758.927	5756.906	3607.436	3605.413	3525.006	3522.958	3302.339	3300.250
	4693.046	4691.039	2946.656	2944.638	2912.528	2910.519	2711.995	2709.959
	3862.507	3860.491	2458.863	2456.850	2474.853	2472.808	2289.757	2287.743
	3195.745	3193.679	2019.590	2017.598	2006.355	2004.330	1869.879	1867.834
	2561.291	2559.252	1633.518	1631.485	1656.406	1654.390	1523.150	1521.121
	2349.782	2347.763	1482.918	1480.926	1519.862	1517.850	1373.882	1371.865
	2138.590	2136.548	1360.015	1358.009	1359.840	1357.800	1276.555	1274.509
Average	3839.152	3837.122	2416.479	2414.464	2385.030	2382.999	2229.377	2227.338
Speed Ratio		1		1.589		1.610		1.723
Efficiency		100%		79.50%		53.60%		43%

* ESI : Entire Subspace Iteration time in conjunction with MATLAB eigen-solver.
1 (or 2, or 3, or 4) processors time: Entire Subspace Iteration time in conjunction with our proposed Jacobi solver.

A TWO-PHASE METHOD FOR STRUCTURAL DAMAGE IDENTIFICATION

Damage detection in structures, specifically bridge-type structures, is an important subject. Its important application has been drawn a lot of old and new scholars to research on this topic. A substantial amount of research works has been reported for damage detection or health monitoring problems [Koh and Dyke (2007); Wang et al. (2012); Moaveni, Hurlebaus, and Moon (2013); Liu et al. (2014); D. Dinh-Cong et al. (2019); D. Dinh-Cong, Vo-Duy, and Nguyen-Thoi (2018); Naveen and Balu (2019)]

In this Section, a two-phase method is presented with a "simple rule of thumb" for structural damage detection and quantification. The merit of the present two-phase method over other exiting two-phase methods [Vo-Duy et al. (2015); Du Dinh-Cong et al. (2017)] is that a simple but efficient "rule of thumb" is proposed for the improvement in damage detection, together with the parallel SI-JT algorithm that is incorporated to effectively compute for the generalized eigen-problem.

To illustrate damage localization and quantification process of the two-phase method, an example on an 11-bar 2D truss structure, as shown in Fig. 1, is used herein. In Fig. 1, the lengths of each member, the cross-sectional area of each member, the material density and Young's modulus are the user's input parameters. In general, the finite element method (FEM) will be based on the type of structure we wish to analyze. This illustrative example is based on a 2D truss structure. Using FEM, one can analyze a structure under (a) undamaged (original) condition, and (b) damaged condition.

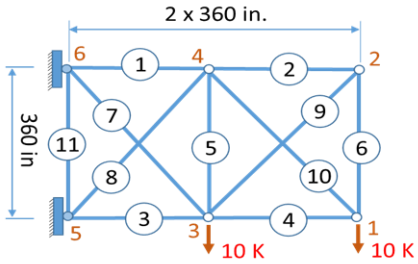


Fig. 1. 11-bar 2D truss structure

Once the frequencies (related to eigen-values) & mode-shapes (eigen-vectors) of the damaged structure can be measured (via optimal locations of sensors), the proposed method can robustly detect the "location (Phase ½) and the severity (Phase 2/2)" of damaged members. The step-by-step numerical procedures of this two-phase method can be summarized in the following Section.

Phase 1/2: Identifying Potentially Damaged Members

Step 1: Finite element analysis of monitored structure

In this step, first the element stiffness $[\mathbf{k}_L^{(e)}]$ matrices, and the element diagonal/lumped mass $[\mathbf{m}_L^{(e)}]$ matrices are computed. Then, the global stiffness $[\mathbf{K}] = \sum\left[\mathbf{k}_G^{(e)}\right]$ matrix, and the global $[\mathbf{M}] = \sum\left[\mathbf{m}_G^{(e)}\right]$ diagonal/ lumped mass matrix is assembled. After that, MATLAB "EIG" command will solve the "generalized" eigen-equation:

$$[\mathbf{K}_{bc}]\varphi_i^* = \omega_i^2[\mathbf{M}_{bc}]\,\varphi_i^* \tag{40}$$

Next, the mass-orthonormalized scalar of each eigen-vector is computed.

$$\{\varphi_i^*\}^T[\mathbf{M}_{bc}]\{\varphi_i^*\} = \text{scalar} = c_i \tag{41}$$

$$\{\varphi_i\} = \frac{\{\varphi_i^*\}}{\sqrt{c_i}} \tag{42}$$

The modal flexibility matrix of the healthy structure can be expressed as follows [Pandey and Biswas (1995); Bernal (2002)]

$$\widetilde{\mathbf{F}_{UD}} = \widetilde{\mathbf{F}_{UnDamaged}} = \sum_{i=1}^{NLM} \frac{1}{\omega_i}\varphi_i\,\varphi_i^T \tag{43}$$

where *NLM* denotes the number of vibration modes considered.

Step 2: Formulating damage indicator based on modal flexibility matrix with a "simple rule of thumb"

Using FEM, the associated damaged structure is analyzed. In real life structure, the measurements of frequencies & mode shapes would come from sensors installed on the structure at key locations. For our example, "artificial damage" is applied to elements #1, #5 and #10 of the mentioned example [see Fig. 1], with stiffness reduction of 80%, 70% and 90% for those 3 elements, respectively.

After obtaining the eigen-solution for damaged structure, the Mass-Orthonormalized scalar of each eigen-vector is computed as

$$\{\varphi_i^*\}^T[\mathbf{M}_{bc}]\{\varphi_i^*\} = \text{scalar} = c_i \tag{44}$$

$$\{\varphi_i\} = \frac{\{\varphi_i^*\}}{\sqrt{c_i}} \tag{45}$$

The modal flexibility matrix of the damaged structure is also expressed as follows [Pandey and Biswas (1995); Bernal (2002)]

$$\widetilde{\boldsymbol{F}_{UD}} = \mathbf{F}_{\widetilde{UnDamaged}} = \sum_{i=1}^{NLM} \frac{1}{\omega_i} \varphi_i \, \varphi_i^T \tag{46}$$

Next, the flexibility matrix changes before and after the existence of damage are obtained by

$$\tilde{\mathbf{F}}_{\Delta} = \tilde{\mathbf{F}}_{UD} - \tilde{\mathbf{F}}_{D} \tag{47}$$

Then by using MATLAB "SVD" command, the given matrix [see Eq. (47)] into its triple products is decomposed, where the second (or middle) matrix is a diagonal matrix, and the first and third matrices are orthogonal matrices as [Bernal (2002)]

$$\tilde{\mathbf{F}}_{\Delta} = [\boldsymbol{U}][\varepsilon][\boldsymbol{V}]^T = \begin{bmatrix} [\boldsymbol{U}_1] & [\boldsymbol{U}_0] \end{bmatrix} \begin{bmatrix} [\varepsilon_1] & [0] \\ [0] & [0] \end{bmatrix} \begin{bmatrix} [\boldsymbol{V}_1] & [\boldsymbol{V}_0] \end{bmatrix}^T \tag{48}$$

If the column vectors in the matrix $[V_0]$ are treated like different loading conditions/vectors [Bernal (2002); Gao, Spencer, and Bernal (2007)], then the stresses of damaged elements will be equal to zero. In practical application, we should use "strain energy" $E_i^{(e)}$ associated with the e^{th} element, such as:

$$E_i^{(e)} = \frac{1}{2}\left\{d_L^{(e)}\right\}^T \left[\boldsymbol{k}_L^{(e)}\right] d_L^{(e)} = \text{scalar;} \tag{49}$$

where i = 1,2, 3,… *ndlv* = number of damaged location vectors = # of columns of the sub-matrix $[V_0]$

It should be noted that the above element's strain energy is associated with the undamaged structure since the goal of Phase 1 is to find/identify which members are damaged.

$$\left\{d_L^{(e)}\right\} = \left[\boldsymbol{R}^{(e)}\right] d_G^{(e)} \tag{50}$$

Where

$$\left[\boldsymbol{R}^{(e)}\right] = \begin{bmatrix} C_x & S_x & 0 & 0 \\ -S_x & C_x & 0 & 0 \\ 0 & 0 & C_x & S_x \\ 0 & 0 & -S_x & C_x \end{bmatrix}; and \;\; C_x = \frac{x_j - x_i}{L^{(e)}} \;;\; C_y = \frac{y_j - y_i}{L^{(e)}} \tag{51}$$

in which

$$L^{(e)} = \sqrt{\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2} \tag{52}$$

Instead of using stress, or strain energy for each element, we use the normalized cumulative energy (NCE) for each element [Du Dinh-Cong et al. (2017); Nguyen-Thoi et al. (2018)], which can be expressed as

$$\bar{E}^{(e)} = \frac{\psi^{(e)}}{\max\limits_k\{\psi^k\}} \tag{53}$$

Where

$$\psi^{(e)} = \sum_{i=1}^{ndlv} \frac{E_i^{(e)}}{\max\limits_k\{E_i^k\}} \tag{54}$$

For each e^{th} element (corresponding to the i^{th} loading case), $E_i^{(e)}$ is computed as shown above for the undamaged case. Within each i^{th} loading case, the max value among all elements is found and the $\max\limits_k\{E_i^k\}$ is obtained. Then, Eq. (54) is applied to compute $\psi^{(e)}$.

Among all $\psi^{(e)}$ values computed, the max value = $\max\limits_k\{\psi^k\}$ is found and Eq. (53) is applied to compute $\bar{\boldsymbol{E}}^{(e)}$. Based on the computed "NCE" $= \bar{E}^{(e)}$, eq (55) is obtained.

$$\bar{\boldsymbol{E}}^{(e)} = \begin{Bmatrix} \bar{E}^{(1)} \\ \bar{E}^{(2)} \\ \bar{E}^{(3)} \\ \bar{E}^{(4)} \\ \bar{E}^{(5)} \\ \bar{E}^{(6)} \\ \bar{E}^{(7)} \\ \bar{E}^{(8)} \\ \bar{E}^{(9)} \\ \bar{E}^{(10)} \\ \bar{E}^{(11)} \end{Bmatrix} = \begin{Bmatrix} very\ close\ to\ zero\ (even\ when\ 3\ or\ 4\ lowest\ modes\ used) \\ \bar{E}^{(2)} \\ \bar{E}^{(3)} \\ \bar{E}^{(4)} \\ very\ close\ to\ zero\ (even\ when\ 3\ or\ 4\ lowest\ modes\ used) \\ \bar{E}^{(6)} \\ \bar{E}^{(7)} \\ \bar{E}^{(8)} \\ \bar{E}^{(9)} \\ very\ close\ to\ zero\ (when\ 6\ lowest\ modes\ used) \\ exactly\ zero\ (Element\ \#11\ is\ connected\ by\ 2\ pinned\ nodes) \end{Bmatrix} \tag{55}$$

It is also noted that element #11 having 2 end nodes which are fully constrained by 2 pinned (Dirichlet boundary condition) supports has its nodal displacements equal to zero. Thus, it has no stress and has a zero value of NCE.

In the present work, a simple rule of thumb is added for improving the damage localization phase. This rule of thumb basically states that if the NCE value of an element is less than or equal to a specific factor, say 10 (based on our numerical experience) times min ($\bar{E}^{(e)}$), then that member should also be considered as a damaged element. However, this rule of thumb should obviously not be applied for finding the minimum energy for any member that is fully constrained at its end nodes such as member 11 in Fig. 1).

$$E^{(e)} \leq 10 \times \min\left(\bar{\boldsymbol{E}}^{(e)}\right) \tag{56}$$

Phase 2/2: Determining the Damage Severity of Potentially Damaged Members

Optimization-based damage detection can be presented as an inverse problem [Du et al. (2018); Alkayem et al. (2018); D. Dinh-Cong et al. (2017)] solved by using optimization algorithms such as genetic algorithm (GA), or differential evolution (DE), etc. The approach aims to minimize a suitable objective function that represents the difference between the predicted and measured responses of a structure. The potentially damaged members identified in Phase 1/2 are considered as design variables.

Let $\vec{x}$ = the unknown amount of damaged in the truss elements # (1), # (5) and # (10).

$$\vec{x} = \begin{Bmatrix} x^{(1)} = [0.00 \rightarrow 1.00] \\ x^{(2)} = [0.00 \rightarrow 1.00] \\ x^{(3)} = [0.00 \rightarrow 1.00] \end{Bmatrix} \tag{57}$$

The unknown vector $\vec{x}$ is found, such that the objective function $\Gamma(\vec{x})$ [Nguyen-Thoi et al. (2018)], defined in Eq. (58), is minimized.

$$\text{Min. } \Gamma(\vec{x}) = 1 - MDLAC(\vec{x}) + \sum_{i=1}^{NLM} \frac{\left\|\phi_{DM,i} - \phi_{DA,i}(\vec{x})\right\|}{\left\|\phi_{DM,i}\right\|} \tag{58}$$

In Eq. (58), $\phi_{DM,i}$ is the i^{th} damaged mode shape, which can be obtained by measurements (using sensors at strategic/optimal locations). $\phi_{DA,i}(\vec{x})$ is the i^{th} analytical (damaged) mode-shape, associated with the current amount of damaged vector $\vec{x}$, found by the optimization (GA, or DE, etc…) process. MDLAC (Multiple Damage Location Assurance Criterion) is calculated as [Messina, Williams, and Contursi (1998)]

$$MDLAC(\vec{x}) = \frac{\left|\Delta f^T \delta f(\vec{x})\right|^2}{(\Delta f^T \Delta f)\left(\delta f^T(\vec{x}) \delta f(\vec{x})\right)} \leq 1 \tag{59}$$

The right-hand side of the above inequality can be easily verified by Cauchy's inequality, and

$$\Delta f = \frac{\vec{f}_{ud} - \vec{f}_{DM}}{\vec{f}_{ud}} \quad (60)$$

$$\delta f(\vec{x}) = \frac{\vec{f}_{ud} - \vec{f}_{DA}(\vec{x})}{\vec{f}_{ud}} \quad (61)$$

If $\vec{f}_{DM} = \vec{f}_{DA}(\vec{x})$, as the measured frequency vector of the damaged structure is equal to the analytical frequency vector, the Eqs. (62-63) will lead to $\Delta f = (\vec{x})$, and $MDLAC(\vec{x}) = 1$. Thus, the minimum value for the objective function will become min. $\Gamma(\vec{x}) = 0$.

Numerical Examples for Damage Localization and Quantification Problems

In this Section, several numerical examples are used to evaluate the performance of the proposed "simple rule of thumb", which basically modify the existing algorithms for damaged detection and damaged quantification of 2D and 3D bridge-type structures. For the purpose, different sizes for 2D and 3D bridge-type structures have been investigated, using the proposed damage identification method. In this work, the actual measurements have not been taken. Instead, artificially created damaged conditions to validate the numerical procedures. In each example, some elements are damaged with different levels of severity. The number of modes used in each example for both phases is much less than the problem size. For 11-bar truss, the first 3 modes, for 48-bar truss, the first 8 modes, and for larger truss the first 32 modes have been used to compute the flexibility matrices.

Comparisons between existing algorithms [Vo-Duy et al. (2015); Du Dinh-Cong et al. (2017)], and the proposed "simple/inexpensive rule of thumb" are reported in Table 10, and in Figs. 2-4. From the Table, it is clear that the existing algorithms [Vo-Duy et al. (2015); Du Dinh-Cong et al. (2017)] are unable to detect all of the damaged members while the improved algorithm can easily recognize the damaged elements and their severities (either low or high), regardless of the input amount of severities on damaged elements.

All the figures are the last iteration results, in which the meaning of each diagram is explained as follows. In all figures, the upper diagram, *x*-axes show the number of variations, which represents the number of damaged elements (for instance, the number of bars shows the number of damaged elements), and *y*-axes named as "current best individual" shows the severity of damage elements. Herein, the *x*-axis shows "score" that indicates the fitness function value, and this *y*-axis also shows the number of populations, which falls within the score ranges. As can be seen, these figures show the convergence of the problem to the results including the number of damage members and their damage severities, the value of the objective function and number population, which falls within the range. Specifically, 30 individuals in the population converged with the score range of [0.2-0.5]x10^{-3}, as seen in Fig. 2.

Table 10. Three different damage cases of 11-bar 2D truss examples with different damaged elements

Case 1: 3 damage elements			Case 2: 4 damage elements			Case 3: 5 damage elements		
Damage Element	Damage Severity	Detected elements by existed alg.	Damage Element	Damage Severity	Detected elements by existed alg.	Damage Element	Damage Severity	Detected elements by existed alg.
1	80%	detected	1	20%	Not detected	1	70%	detected
5	70%	detected	7	10%	Not detected	3	50%	Not detected
10	90%	detected	5	30%	Not detected	6	70%	detected
			10	50%	detected	7	20%	Not detected
						9	40%	Not detected

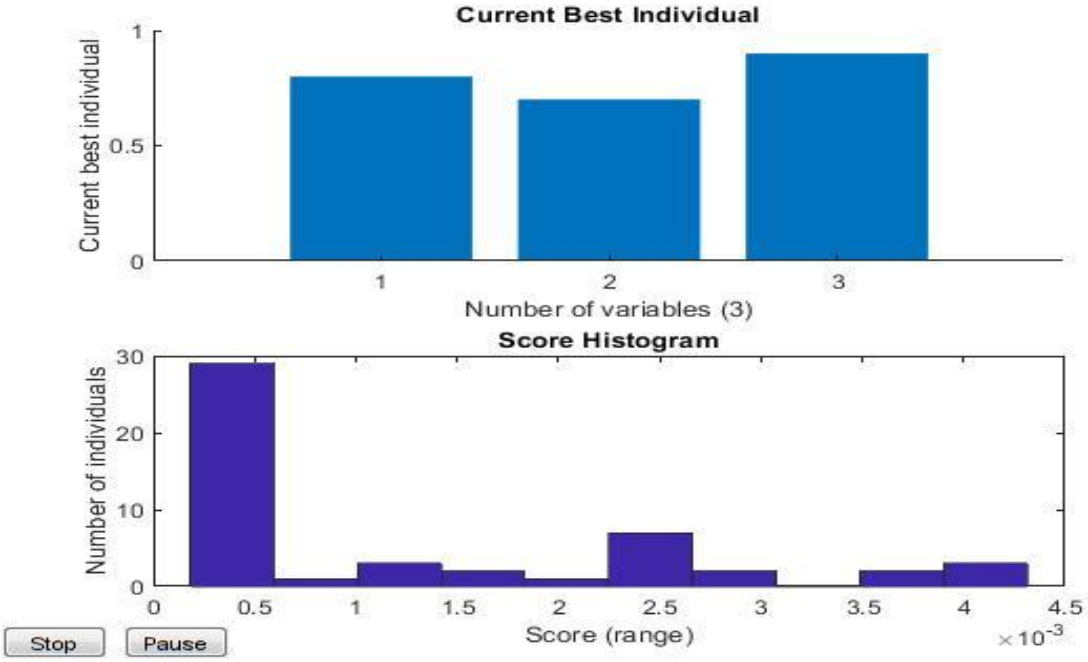


Fig. 2. Damage severity result for case 1 of the 11-bar 2D truss structure

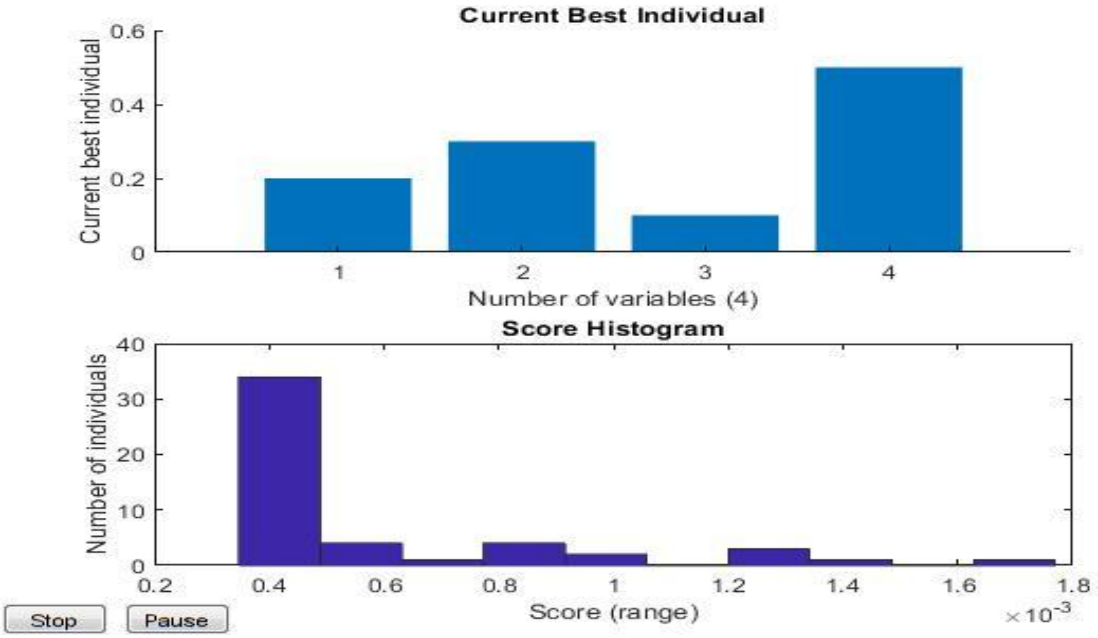


Fig. 3. Damage severity result for case 2 of the 11-bar 2D truss structure

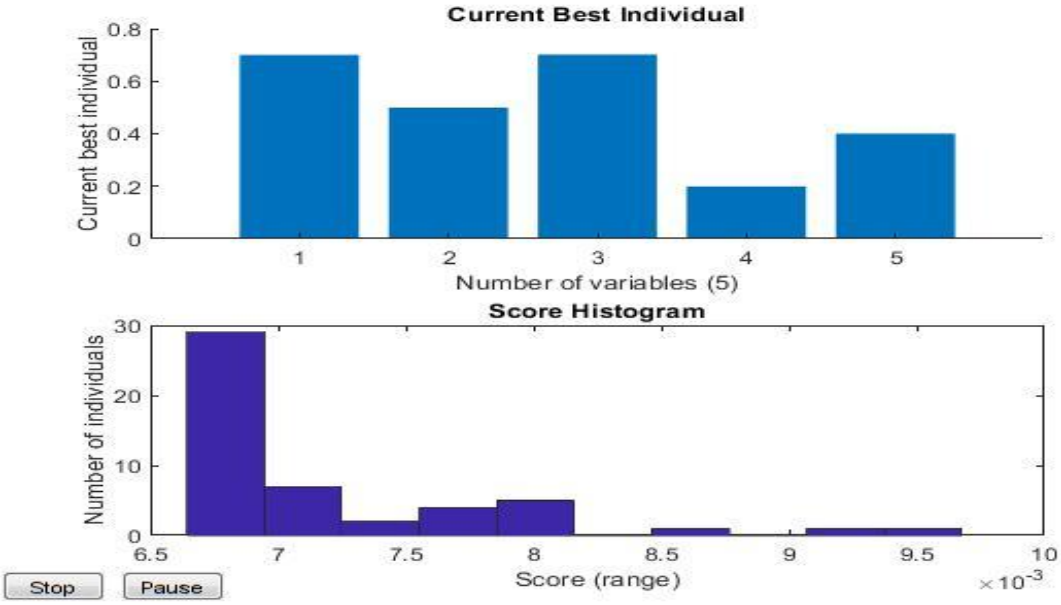


Fig. 4. Damage severity result for case 3 of the 11-bar 2D truss structure

The second example considered is a 48-bar 3D truss which contains 1 bay, 3 stories, and 2 frames, as depicted in Fig. 5. Each frame consists of columns, beams and x brace frames in each bay and stories, including the connecting bays. A damage scenario with five damage locations is assumed in this 3D truss, as listed in Table 11.

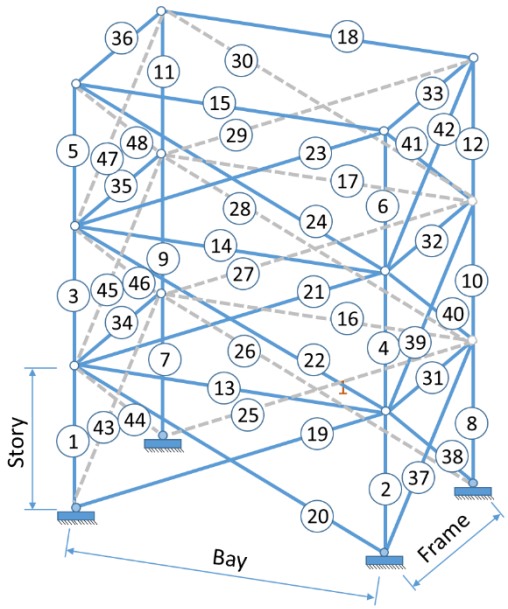


Fig. 5. 48-bar 3D truss structure

Table 11. A damage scenario with five damage locations of the 48-bar 3D truss

Damage Element	Damage Severity	Detected elements by existed alg.
5	90%	Detected
13	80%	Not detected
20	60%	Not detected
35	90%	Detected
37	20%	Not detected

The last example is a bridge with 10 bays, each 8 feet, 8 stories, each 8 feet and 6 frames, each 8 feet. This example has 1782 degrees of freedom, 594 nodes and 3288 members, which is a larger size problem. This structure has been used to show the time efficiency as well as the accuracy of the proposed method.

In this example, 5 elements have been identified as damaged by the proposed algorithm correctly. The damage severity of members is varied, which has been detected by the program correctly. Also, the computing time is reduced by using 2 processors in parallel computation. Computation time using a various number of processors is reported in Table 12. The results can be found in Table 13 and Fig. 7.

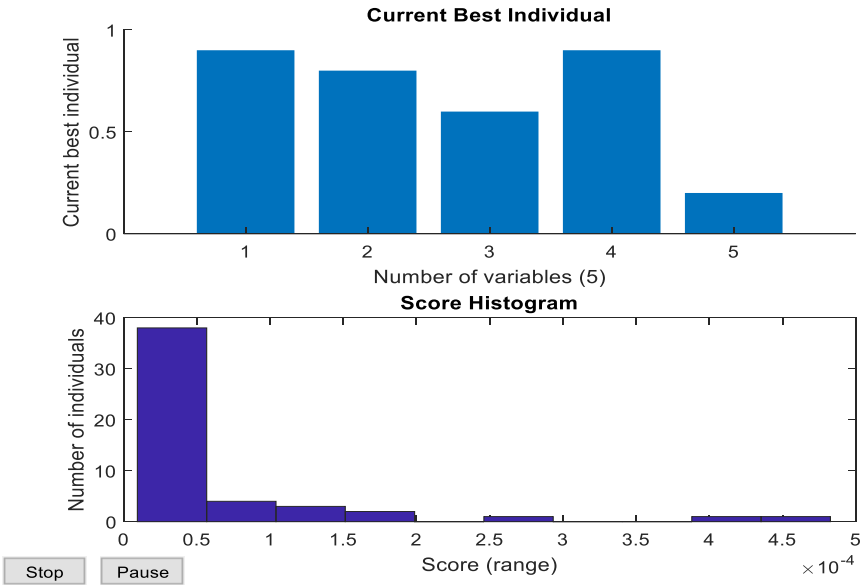


Fig. 6. Damage severity result for damage case of the 48-bar truss 3D structure

Table 12. Computation time for damage estimation of the 3D bridge with 10 bays

Number of processors	Time (second)
1	1375.4501
2	941.3579
3	898.6813
4	867.9016

Table 13. A damage scenario with five damage locations of the 3D bridge with 10 bays

Damage Element	Damage Severity	Detected elements by existed alg.
10	80%	Detected
37	70%	Not detected
55	90%	Not detected
529	75%	Not detected
705	40%	Not detected

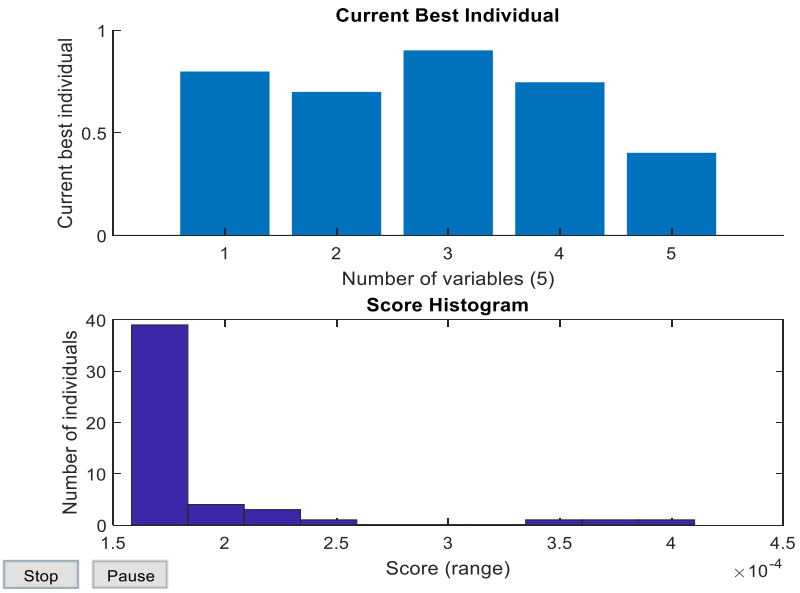


Fig. 7. Damage severity result for damage case of the 3D bridge with 10 bays

CONCLUSIONS

Serial Jacobi transformation algorithm for the solution of standard eigen-problems is revisited to facilitate the explanation of the proposed parallel transformation algorithm, for which computational efficiency can be realized in this study through "pattern recognition" for the development of explicit formulas to avoid matrix times matrix operations.

The parallel SI-JT method for the solution of generalized eigen-problems has been incorporated into the improved damage identification algorithm. Computational efficiency and robust behaviors for the entire proposed procedures (eigen-solution, damage localization and quantification) have been validated through several academic and real-life numerical examples. All numerical results obtained from this study have indicated that the parallel SI-JT method is more robust as compared to the combined subspace iteration and MATLAB eigen-solver. Furthermore, the parallel SI-JT method incorporated into the improved damage identification algorithm for damage assessment of 2D and 3D bridge-type structures also gives better results than existing algorithms.

Acknowledgements

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant number 107.02-2019.330.

References

Alkayem, Nizar Faisal, Maosen Cao, Yufeng Zhang, Mahmoud Bayat, and Zhongqing Su, *Structural Damage Detection Using Finite Element Model Updating with Evolutionary Algorithms: A Survey*. Neural Comput., Springer London on, 1994. 30(2): p. 389–411.

Bathe, Klaus-Jürgen, *Clinical ophthalmology*. Klaus-Jurgen Bath

Bernal, Dionisio, *Load Vectors for Damage Localization.* J. Eng. Mech., American Society of Civil Engineers, 2002. 128(1): p. 7-14.

Carden, E. P., *Vibration Based Condition Monitoring: A Review.* Struct. Heal. Monit., 2004. 3(4): p. 355–377.

Das, Swagato, P. Saha, and S. K. Patro, *Vibration-Based Damage Detection Techniques Used for Health Monitoring of Structures: A Review.* J. Civ. Struct. Heal. Monit., Springer Berlin Heidelberg, 2016. 6(3): p. 477–507.

Dinh-Cong, D., T. Nguyen-Thoi, M. Vinyas, and Duc T. Nguyen, *Two-Stage Structural Damage Assessment by Combining Modal Kinetic Energy Change with Symbiotic Organisms Search.* J. Civ. Struct. Heal. Monit., Int. J. Struct. Stab. Dyn., 2019. 19(10): p. 1950120.

Dinh-Cong, D., T. Vo-Duy, V. Ho-Huu, H. Dang-Trung, and T. Nguyen-Thoi, *An Efficient Multi-Stage Optimization Approach for Damage Detection in Plate Structures*. Adv. Eng. Softw., Elsevier Ltd, 2017. 112(October): p. 76–87.

Dinh-Cong, D., T. Vo-Duy, V. Ho-Huu, H. Dang-Trung, and T. Nguyen-Thoi, *An Efficient Multi-Stage Optimization Approach for Damage Detection in Plate Structures*. Adv. Eng. Softw., Elsevier Ltd, 2017. 112(October): p. 76–87.

Dinh-Cong, D., T. Vo-Duy, and T. Nguyen-Thoi, *Damage Assessment in Truss Structures with Limited Sensors Using a Two-Stage Method and Model Reduction*. Appl. Soft Comput, 2018. 66(May): p. 264–277.

Dinh-Cong, Du, Trung Vo-Duy, Nhan Nguyen-Minh, Vinh Ho-Huu, and Trung Nguyen-Thoi, *A Two-Stage Assessment Method Using Damage Locating Vector Method and Differential Evolution Algorithm for Damage Identification of*
Cross-Ply Laminated Composite Beams. *Adv. Struct. Eng.*, 2017. 20(12): p. 1807–1827.

Du, Dinh-Cong, Ho-Huu Vinh, Vo-Duy Trung, Ngo-Thi Hong Quyen, and Nguyen-Thoi Trung, *Efficiency of Jaya Algorithm for Solving the Optimization-Based Structural Damage Identification Problem Based on a Hybrid Objective Function*. Eng. Optim., 2018. 50(8): p. 1233–1251.

Gao, Y, B. F. Spencer, and D Bernal, *Experimental Verification of the Flexibility-Based Damage Locating Vector Method*. J. Eng. Mech., 2007. 133(10): p. 1043–1049.

Goluh, Gene H, and Charles F Van Loan, *Matrix Computations*. Johns Hopkins.

Heath, Michael T, *Scientific Computing: An Introductory Survey*. Vol. 80. SIAM.

Kohh, B. H., and S. J. Dyke, *Structural Health Monitoring for Flexible Bridge Structures Using Correlation and Sensitivity of Modal Data*. Comput. Struct., 2007. 85(3-4): p. 117–130.

Liu, Hanbing, Gang Song, Yubo Jiao, Peng Zhang, and Xianqiang Wang, *Damage Identification of Bridge Based on Modal Flexibility and Neural Network Improved by Particle Swarm Optimization*. Math. Probl. Eng. 2014.

Messina, a., E.J. Williams, and T. Contursi, *Structural Damage Detection by A Sensitivity and Statistical-Based Method*. J. Sound Vib., 1998. 216(5): p. 791–808.

Moaveni, Babak, Stefan Hurlebaus, and Franklin Moon, *Special Issue on Real-World Applications of Structural Identification and Health Monitoring Methodologies*. J. Struct. Eng., 2013. 139(10): p. 1637–1638.

Naveen, B. O., and A. S. Balu. Contursi, *HDMR-Based Model Update in Structural Damage Identification*. Int. J. Comput. Methods, 2019. 16(5): p. 1–14.

Nguyen-Thoi, T., A. Tran-Viet, N. Nguyen-Minh, T. Vo-Duy, and V. Ho-Huu, *A Combination of Damage Locating Vector Method (DLV) and Differential Evolution Algorithm (DE) for Structural Damage Assessment*. Civ. Eng., 2018. 12(1): p. 92–108.

Nguyen, Duc Thai, *Finite Element Methods: Parallel-Sparse Statics and Eigen-Solutions*. Springer Science & Business Media.

Pandey, A.K., and M Biswas, *Experimental Verification of Flexibility Difference Method for Locating Damage in Structures*. J. Sound Vib, 1995. 184(2): p. 311–328.

Pissanetzky, Sergio, *Sparse Matrix Technology-Electronic Edition*. Academic Press.

Reddy, J N, *An Introduction to the Finite Element Method*. Vol. 1221. McGraw-Hill New York, USA.

Sameh, Ahmed H, *On Jacobi and Jacobi-like Algorithms for a Parallel Computer*. Math. Comput., 1971. 25(115): p. 579–590.

Vo-Duy, T., N. Nguyen-Minh, H. Dang-Trung, A. Tran-Viet, and T. Nguyen-Thoi, *Damage Assessment of Laminated Composite Beam Structures Using Damage Locating Vector (DLV) Method*. J. Contemp. Phys., 2015. 50(4): p. 457–465.

Wang, Frank L, Tommy H T Chan, David P Thambiratnam, Andy C C Tan, and Craig J L Cowled, *Correlation-Based Damage Detection for Complicated Truss Bridges Using Multi-Layer Genetic Algorithm*. Adv. Struct. Eng., 2012. 15(5): p. 693–706.

Wei Fan, and Pizhong Qiao, *Vibration-Based Damage Identification Methods: A Review and Comparative Study*. Struct. Heal. Monit., 2011. 10(1): p. 83–111.