Sound & Vibration 2026, 60(2), 3923.
doi: https://doi.org/10.59400/sv3923

Article

Modal FEA-EMA CrossMAC analysis of a Bell UH-1H helicopter tail rotor blade using FEA software and open source algorithms

Daniel J. Winarski[1,*], Marc D. Lamparelli[2], Tyson Y. Winarski[3]

[1] Independent Researcher, Tucson, AZ 85710, USA

[2] Spectral Dynamics, Technical Support Contractor, Troy, MI 48098, USA

[3] Sandra Day O'Conner College of Law, Phoenix Campus, Arizona State University, Phoenix, AZ 85004, USA

*** Corresponding author:** Daniel J. Winarski, winarskifirm@gmail.com

CITATION

Winarski DJ, Lamparelli MD, Winarski TY. Modal FEA-EMA CrossMAC analysis of a Bell UH-1H helicopter tail rotor blade using FEA software and open source algorithms. Sound & Vibration. 2026; 60(2): 3923. https://doi.org/10.59400/sv3923

ARTICLE INFO

Received: 15 January 2026
Revised: 11 March 2026
Accepted: 18 March 2026
Available online: 8 April 2026

Abstract: Our challenge was the data integration between the output of two disparate commercial software packages, one for analytical finite element analysis and the other for experimental modal analysis. Our primary objective was the augmentation of two existing free open source subroutines with our own programming to create a versatile analysis tool capable of calculating the Cross-Application Modal Assurance Criterion (CrossMAC) comparison of mode shapes between experimental modal analysis and analytical finite element analysis of a stationary tail rotor blade of a Bell UH-1H helicopter. The key contribution of our work was the creation of an original GNU Octave main program with four original subroutines, along with the two existing open source subroutines and one modification to one of these existing subroutines, to integrate the experimental modal analysis done using Spectral Dynamics STAR7 software and the analytical finite element analysis using Mecway. Both the STAR7 and Mecway commercial software packages were chosen because each permitted access by our GNU Octave program to critical node locations and mode shape data for three out-of-plane flapping modes as well as a torsional mode of vibration. By changing from a fixed to an elastic support, and enabling grid refinement, our analytical modal frequencies agreed well with our experimental ones, giving a Pearson correlation of 99.1% between our experimental and analytical data. Our open source software framework and methodology offers an extensible and robust approach for validating experimental modal data against analytical finite element modeling, with direct applicability to a broad diversity of vibration applications.

Keywords: helicopter; tail rotor blade; STAR7; modal analysis; Mecway; finite element analysis; GNU Octave; modal assurance criterion

1. Introduction

Fundamental publications on the Modal Assurance Criterion (MAC) included Brown and Allemang [1], which included the development of the Modal Assurance Criterion in 1978 at the University of Cincinnati. Allemang [2], University of Cincinnati, subsequently summarized the basic MAC equation and discussed several alternative formulations. Equation (4c) by Allemang was used in our open source program written in GNU Octave. Allemang also discussed the historical representation of MAC values as a numerical table, then transitioning to colorized 2-D and 3-D presentations. The lead author of our manuscript received his first education on experimental modal analysis through Professor David Brown.

Pastor et al. [3] listed several important reasons why the main diagonal elements of the Modal Assurance Criterion matrix may be near unity while off-diagonal elements may be near zero.

Rezaifar et al. [4] used the Modal Assurance Criterion to compare three analytical modes of vibration with three experimental modes for a single story building constructed on a shake table. However, this reference did not disclose the code used.

Santos et al. [5] used the Coordinate Modal Assurance Criterion (COMAC) on a helicopter main rotor blade to detect experimental damage.

Grappasonni et al. [6] use the Modal Assurance Criterion to compare the Eigensystem Realization Algorithm (ERA) and the Hilbert Transform Method (HTM) of an AgustaWestland Lightweight Helicopter Tail using white noise excitation.

Safari Tarbozagh et al. [7] used the Modal Assurance Criterion to compare intact vs. debonded concrete-filled steel tubes.

Khanahmadi et al. [8] made use of the Logarithm of the Modal Assurance Criterion, LMAC = $-\log_{10}$(MAC), to investigate mode shapes of a column with axial load and without an axial load. This axial load varied between 0.1 and 0.5 times the critical axial load. This LMAC has a different look from the Modal Assurance Criterion. The LMAC main diagonal values are preferably zero, which is the logarithm of unity, the desired main diagonal value of the Modal Assurance Criterion. Additionally, the off-diagonal values are large, just the opposite of the traditional MAC.

Dai and Ji [9] modified the equation for the Modal Assurance Criterion to represent the MAC where modes of vibration are weighted-orthogonal to their respective masses. Our research did not need to accommodate this modification, as all of our calculations were on a unit-mass basis.

Our original study [10] benefited from Modal Assurance Criterion as applied solely within the STAR7 [11, 12] experimental modal analysis (EMA) package and, separately, solely within a finite element analysis (FEA) package. This follow-on paper introduces original open source code that enables direct computation of the Cross-Application Modal Assurance Criterion (CrossMAC) values between the otherwise independent analysis platforms of commercial Spectral Dynamics STAR7 software and the analytical finite element analysis using commercial Mecway [13–15]. This paper is devoted to that innovative GNU Octave open source programming.

2. Experimental modal analysis on UH-1H tail rotor blade

The Bell UH-1 [16], informally nicknamed “Huey”, was the beginning of a successful family of utility helicopters designed and produced by the American aerospace company Bell Helicopter. This helicopter was originally designated HU-1, as in Helicopter Utility, and hence the “Huey” nickname. The nickname “Huey” became ubiquitous and has stayed in common use for the life of this helicopter, despite the later and official redesignation to UH-1. The UH-1 remains in production today. Bell Helicopter was renamed as Bell Helicopter Textron in 1976 and is now simply known as Bell as of 2018. Bell installed the 1,000 kW (1,400 shaft horse power or shp) Lycoming T53-L-13 turbine engine to provide more power for the helicopter. The pitot tube, used to measure air speed, was relocated from the nose to the roof of the cockpit, as shown

in **Figure 1**, to prevent inadvertent damage during a landing. Production models in this configuration were designated as the UH-1H. The UH-1H was the most-produced version, and is representative of all "Huey" types. Hence, with careful consideration, it was the tail rotor blade of this UH-1H helicopter that was chosen for our study.

Figure 1. Bell UH-1H [17] showing the XYZ coordinate system of our tail rotor.

Figure 1 shows a Bell UH-1H along with the right-hand Cartesian coordinate system adopted by this study. When the helicopter was in flight, the tail rotor rotated about the Z-axis. The tail rotor of the UH-1H helicopter had two blades. The radius of the arc of the tail rotor was 1.295 m (4.25 feet). The Y-axis chord of the largely rectangular blade was 213.6 mm (8.41 inches). The mass of the one blade was approximately 3 kg and the overall blade length along the X-axis was 1.16 m. The center of mass was located 0.685 m from the outer edge, 0.475 m from the inner edge of the blade. The three flapping modes and one torsion mode which we are investigating are introduced as thumbnail screenshots in the lower right insert of **Figure 1**, matching the orientation of the pictured tail rotor blade of the Bell UH-1H.

Our experimental modal analysis was done in Lab D-18 in Mahan Hall at what is now known as the Department of Mechanical and Aerospace Engineering (MAE), United States Military Academy, West Point, New York. The tail-rotor blade of the UH-1H helicopter was loaned for this study by the Second Aviation Detachment stationed at the Steward Army Air Field, Newburgh, New York.

Our study utilized 45 nodes along the top surface of the UH-1H tail rotor blade, which was effectively a fixed-free cantilever beam. For comparison, Santos et al. [5] used 55 nodes to study a main rotor of a helicopter, each node having an accelerometer.

As previously described by Winarski et al. [10], our experimental modal analysis had a single Brüel & Kjær (B&K) 4393 accelerometer (fixed-location response, in m/s^2) [18] mounted at the radially outermost node along the trailing edge of the tail rotor blade (node 16) with beeswax. This location for a single accelerometer was chosen because the Euler-Bernoulli theory for a fixed-free cantilever beam presented by Blevins [19], Lima [20], Yoo and Shin [21], and Wright et al. [22] all indicated that node 16 was not a vibrational node and that there were no closely spaced modes. A B&K

8202 impact hammer and 8200 force transducer (excitation, in Newtons) [23] were used to gently tap the 45 nodes, and the fixed location accelerometer sensed the subsequent response. Using the hard-rubber tip, without the additional mass, on the impact hammer limited the excitation to 500 Hz, coinciding nicely with the 400 Hz bandwidth on the B&K 2034 signal analyzer. B&K 2644 line-driver charge-amplifiers [24] amplified the low-level signals from the accelerometer and force transducer before these signals were received by the signal analyzer. The resulting dataset comprised 45 experimental frequency response functions (*.FRF), each corresponding to a unique excitation-response pair of nodes. These *.FRF files were originally processed by SMS StarStruct software, which we then replaced by its most recent successor, Spectral Dynamics STAR7 Premier Modal software, in order to take advantage of greatly improved post-processing graphics and analysis capabilities.

3. Results

This section first shows our use of the polynomial method, found in the STAR7 Premier Modal software 3410–9710 [11,12], to identify three flapping modes and one torsional mode of vibration from the experimental *.FRF, followed by our analysis of those four modes of vibration with the Mecway analytical finite element package, first with fixed then elastic support of the base of the tail rotor blade. Then we provide a complete description of the GNU Octave open source used to integrate the mode shapes from the STAR7 experimental modal analysis and Mecway analytical finite element packages, in order to generate the desired Cross-Application Modal Assurance Criterion between these two applications.

3.1. Identifying modes of vibration using STAR7 polynomial method

Figure 2 shows exemplary 1Z–16Z Frequency Response Function (blue line) displaying four potential modes of vibration over the frequency range of $0 \leq \omega \leq 400$ Hz. 1Z–16Z in the upper left of **Figure 2** meant that this was the response at node 1Z with the reference at 16Z, the location of the accelerometer. Four bands (marked with red lines) were defined in **Figure 2** for the purpose of vibration mode identification by the polynomial method. The specific low and high frequencies associated with each cursor band are listed in **Table 1**.

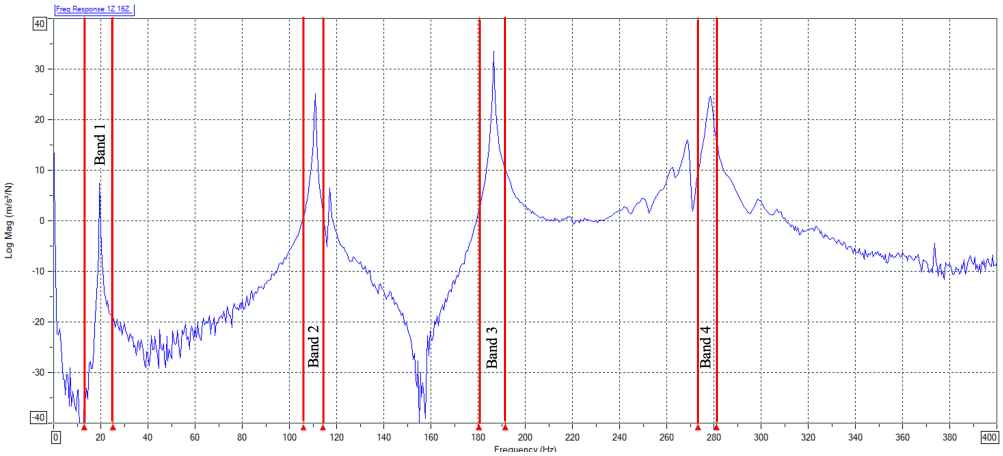

Figure 2. Defining four bands to identify four modes of vibration.

Table 1. Mode identification bands used in **Figure 2** for STAR7 polynomial method.

Cursor band number	Center Hz	Low cursor Hz	High cursor Hz	Low mode	High mode
1	19.19 Hz	13.11 Hz	25.28 Hz	1	1
2	110.15 Hz	106.02 Hz	114.29 Hz	2	2
3	186.02 Hz	180.43 Hz	191.61 Hz	3	3
4	277.22 Hz	273.32 Hz	281.12 Hz	4	4

The center frequency, low (left) boundary frequency, and the high (right) boundary frequency of each of the four bands are listed in **Table 1**. Although it was possible to identify more than one mode of vibration in a single band, we identified each desired mode individually, for clarity and simplicity.

Although this polynomial solution technique was different from the Advanced Curve Fitter (ACF) technique which we previously published [10], the four modes of vibration and associated frequencies were essentially the same as identified before. **Table 2** shows the modal frequencies for the ACF and the Polynomial methods. The difference between the previous ACF method and the polynomial method varied between 0.08 and 0.59 Hz.

Table 2. STAR7 modal frequencies for ACF (previous) and polynomial methods.

STAR7 mode number	STAR7 mode type	ACF method frequency	Polynomial method frequency	Difference in frequency
1	Flapping	19.31 Hz	19.621 Hz	0.31 Hz
2	Flapping	110.35 Hz	110.94 Hz	0.59 Hz
3	Torsion	186.43 Hz	186.61 Hz	0.18 Hz
4	Flapping	278.40 Hz	278.32 Hz	0.08 Hz

3.2. Mecway finite element analysis—Fixed support

We then downloaded version 30 of the Mecway finite element package [13], because the mode shapes were easily accessible for post-processing by our GNU Octave open source algorithms. Mecway Limited, the developer of the Mecway finite element analysis software, is headquartered in Wellington, New Zealand.

We picked Mecway because of ease of accessibility to its mode shape vectors, the focus of our study. Our previous finite element package stored the desired mode shape vectors in a binary *.rst file, which prevented us from easily accessing them. Mecway is a comprehensive, user friendly finite element analysis package for Windows, with a focus on mechanical and thermal simulation such as stress analysis, vibration, and heat flow. It is inexpensive and has an intuitive graphical interface for each mesh creation and display of solutions [14].

Figure 3 shows that we first created a finite element model [15] with 60 nodes and 14 hexahedral elements. We then employed Mesh Tools > Refine > x2 to successively subdivide the hexahedral elements by a factor of two. We stopped this grid refinement with a 1425 node model. Aluminum was the material selected. Mecway made extensive use of the XML-based *.LIML file type for data, which stood for Library Information Markup Language [25].

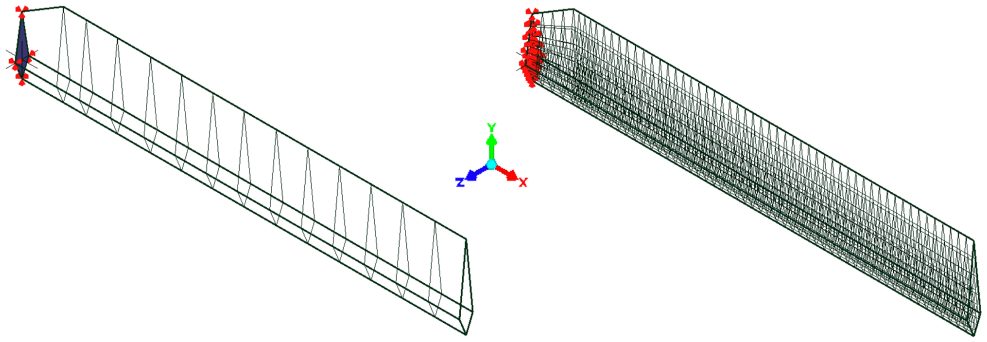

Figure 3. Mecway finite element 60 node model (left) and 1425 node model (right).

The first 45 Mecway nodes were identical to the 45 STAR7 nodes defined in our original publication [10]. This matching of the first 45 nodes was done deliberately, to facilitate creating a CrossMAC between the experimental modal analysis of STAR7 and the Mecway analytical finite element analysis. The remaining 15 nodes, to create the 60 node Mecway model, are listed in **Table 3**.

Table 3. 15 additional Mecway FEM nodes completing the 60 node tail rotor blade.

Mecway node	X (m)	Y (m)	Z (m)	Rotor blade component
46	1.070	0	−0.016	Max Thick
47	0.995	0	−0.016	Max Thick
48	0.917	0	−0.016	Max Thick
49	0.841	0	−0.016	Max Thick
50	0.765	0	−0.016	Max Thick
51	0.689	0	−0.016	Max Thick
52	0.613	0	−0.016	Max Thick
53	0.535	0	−0.016	Max Thick
54	0.460	0	−0.016	Max Thick
55	0.385	0	−0.016	Max Thick
56	0.310	0	−0.016	Max Thick
57	0.230	0	−0.016	Max Thick
58	0.155	0	−0.016	Max Thick
59	0.100	0	−0.016	Max Thick
60	0	0	−0.016	Max Thick

Table 4 lists the number of nodes used in each Mecway model, followed by the number of elements in each model, the element size, and the element type. In both models, the H8 element was used, which stands for hexahedral element with eight vertices, and one finite element node was associated with each vertex. For the initial grid refinement, each of the 14 coarse H8 hexahedral elements in the 60 node model was subdivided by a factor of two in each of the X, Y, and Z directions, giving 2^3 or 8 H8 elements for each original element and creating a 261 node model. Multiplying the number of original elements, 14, by 8 which each was subdivided into, gave 112 (8 × 14) H8 intermediate elements in the 261 node model, as shown in **Table 4**. A second grid refinement was then done, where each of the 112 intermediate H8 hexahedral elements in the 261 node model was further subdivided by a factor of two in each of the X, Y, and Z directions, giving 2^3 or 8 H8 fine elements for each intermediate element and creating a 1425 node model. Multiplying the number of intermediate elements, 112, by the 8 which each was subdivided into, gave 896 (8 × 112) H8 fine elements in what

became our final 1425 node model, as shown in **Table 4**.

Table 4. Mecway nodes, elements, and element type.

Mecway nodes	Mecway elements	Element size	Element type
60	14	Coarse	H8
261	112 (8 × 14)	Intermediate	H8
1425	896 (8 × 112)	Fine	H8

The solver we used within Mecway was CalculiX (CCX) which is a free and open source finite element analysis application.

Table 5 lists the modal frequencies for the STAR7 polynomial method, the Mecway 60 node finite element model, and the Mecway 1425 node model, for the case of fixed support at the base of the tail rotor blade. The grid refinement of the 1425 node model gave frequencies much closer to the STAR7 polynomial method, especially for the torsion mode and the third flapping mode. In addition to being closer in frequency as noted, the set of frequencies converged for the 1425 node model to a stable set of values at this level of refinement and no further refinement was necessary.

Table 5. STAR7 vs. Mecway modal frequencies—Fixed support

Mode number	Mode type	STAR7 polynomial	Mecway 60 nodes	Mecway 1425 nodes
1	Flapping	19.621 Hz	19.348 Hz	18.579 Hz
2	Flapping	110.94 Hz	122.16 Hz	116.27 Hz
3	Torsion	186.61 Hz	306.99 Hz	244.72 Hz
4	Flapping	278.32 Hz	350.31 Hz	324.80 Hz

Figures 4 and **5** show the Mecway 1425 node mode shapes in order of increasing frequency for the fixed support condition. **Figure 4** shows the first and second flapping modes. **Figure 5** then shows the torsion mode followed by the third flapping mode. The color coding in **Figures 4** and **5** followed the colors of the rainbow, where red represented the maximum deflection, followed by orange, yellow, green, and finally dark blue for minimum deflection. The tail rotor blade in **Figures 3–5** is shown in its operational orientation, where it rotates about the horizontal Z-axis, as shown in **Figure 1**.

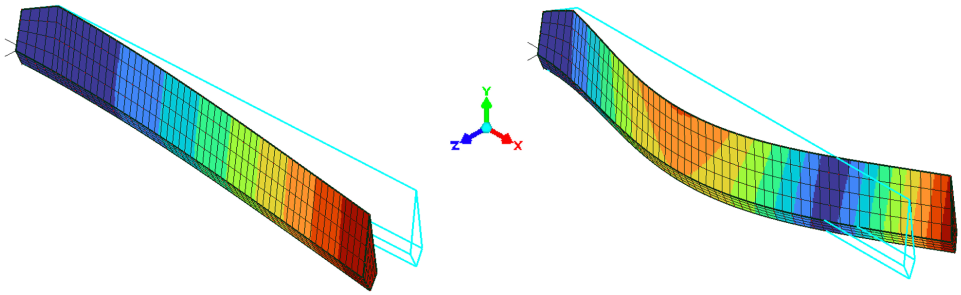

Figure 4. First flapping (left) and second flapping (right) modes—Fixed support.

Vibrational nodes are defined as stationary points in a standing wave where the rotor blade experiences zero displacement (blue color) while the points in between (antinodes) move with maximum amplitude (red color). In **Figure 4**, one vibrational node can be seen in the second flapping mode. In **Figure 5**, two vibrational node can be seen in the third

flapping mode. None of these vibrational nodes coincided with the fixed location of our B&K 4393 accelerometer at 16Z, validating our choice of that location. Red (maximum amplitude) can be seen on both sides of these blue vibrational nodes.

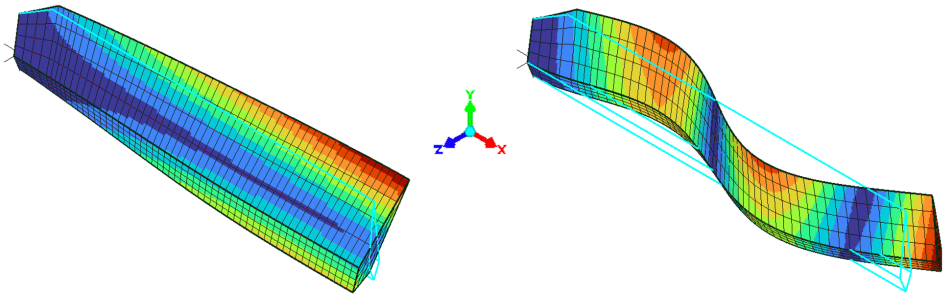

Figure 5. First torsion (left) and third flapping (right) modes—Fixed Support.

Next, we needed to set up the table which will export a *.csv (comma separated variable) file each time the Mecway solver is run. It can be disabled by right click > Suppress. To make it active, simply right click > Unsuppress. In Mecway, click on the Solution item in the left hand tree to make the Table-icon active. Fill in this table to create a nine column data file as shown in **Figure 6**. Note that mode 2 within Mecway was not selected because the finite element model generated an X-Y in-plane mode which we did not experimentally measure in our modal analysis because our accelerometer was perpendicular to the X-Y plane. This table was then populated by pressing the Update Button on the right-hand side of the table. The path at the bottom was the full path to the GNU Octave programming, including the file name. This *.csv file was needed by our GNU-Octave mwtable2modal3.m subroutine.

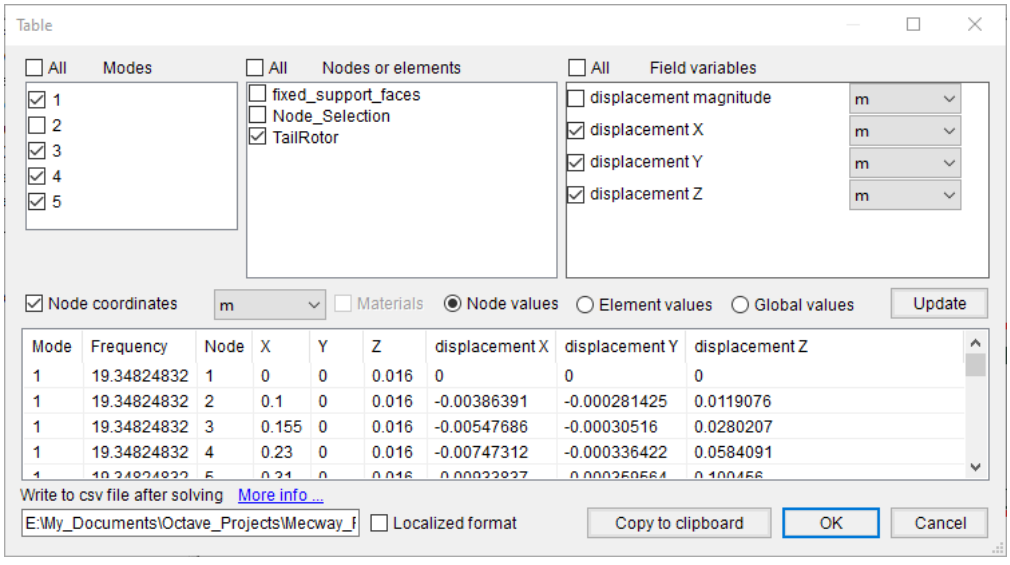

Figure 6. Set up the table to export a Mecway *.csv file.

3.3. Mecway finite element analysis—Elastic support

Figure 7 shows the Mecway 1425 node FEA model with the former fixed support now replaced by an elastic support at the base of the helicopter tail rotor blade. The left side of **Figure 7** shows the Mecway tree, which lists the tail rotor blade being modeled with 896 elements, **Table 4**, and that the tail rotor blade material is aluminum. The elastic

support is the spring rate (N/m) distributed over the contact area (square meters), with a value of 6,000 GPa/m. The tree then shows that CCX was the solver used.

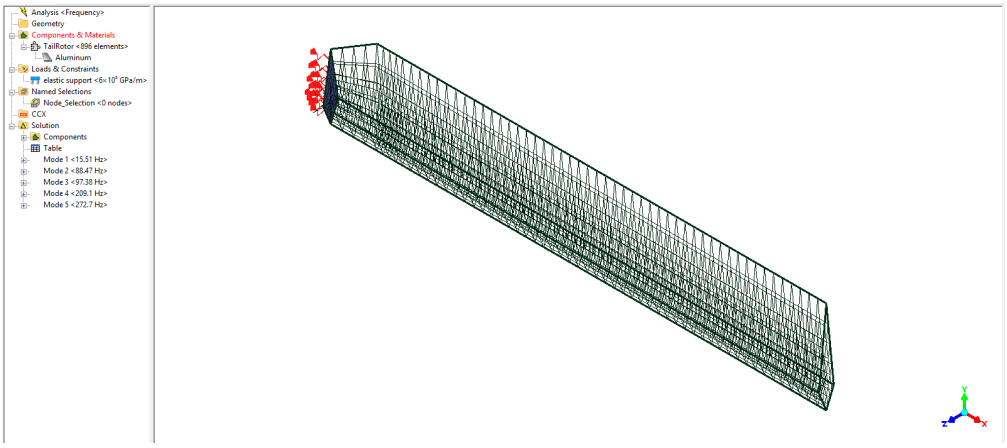

Figure 7. Mecway 1425 node FEA with elastic support.

Figures 8 and **9** show the Mecway 1425 node mode shapes in order of increasing frequency for the elastic support condition. **Figure 8** shows the first and second flapping modes. **Figure 9** then shows the torsion mode followed by the third flapping mode. The color coding in **Figures 8** and **9**, as also in **Figures 4** and **5**, followed the colors of the rainbow, where red represented the maximum deflection, followed by orange, yellow, green, and finally dark blue for minimum deflection. The tail rotor blade in **Figures 7–9** is shown in its operational orientation, where it rotates about the horizontal Z-axis, as shown in **Figure 1**.

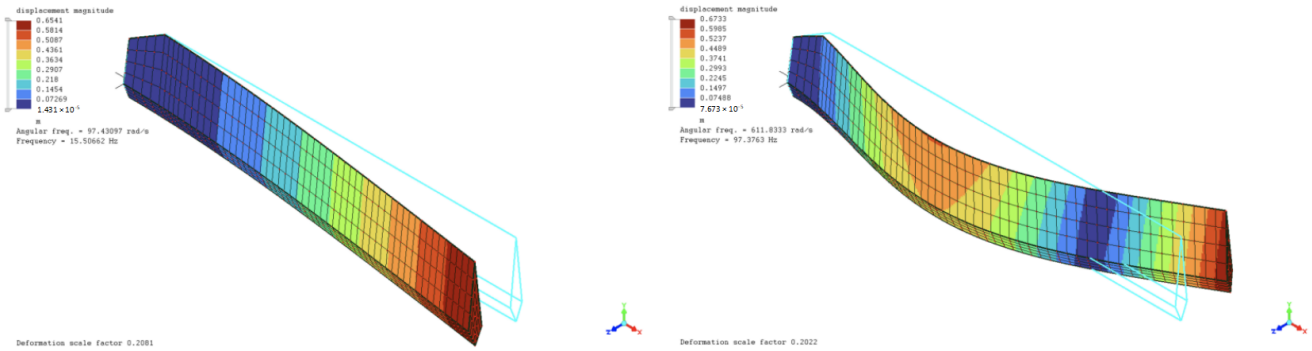

Figure 8. First flapping (left) and second flapping (right) modes—Elastic support.

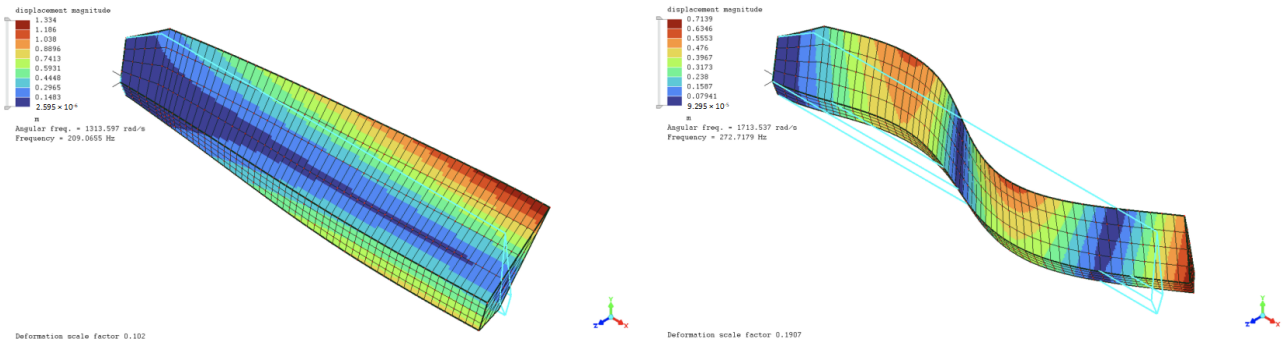

Figure 9. First torsion (left) and third flapping (right) modes—Elastic support.

3.4. GNU Octave open source programming

GNU Octave [26] open source programming was used to integrate STAR7 and Mecway. Octave was originally conceived in the late 1980s to be companion software for an undergraduate-level textbook on chemical reactor design being written by James B. Rawlings of the University of Wisconsin-Madison and John G. Ekerdt of the University of Texas [27]. Seen as a replacement for the older and somewhat cumbersome FORTRAN programming language, the first formal release of Octave was in 1994. Octave was named for one of the author's former professors, Octave Levenspiel, who wrote a famous textbook on chemical reaction engineering. Octave is licensed under the GNU General Public License version 3 [28].

GNU Octave-10.3.0-w64-installer.exe was downloaded [29] and installed. The GNU Octave online manual [30] was very helpful. The main-subroutine architecture of the flowchart shown in **Figure 10** details the main GNU Octave program ROTOR2_MainScript and the seven subroutines subsequently invoked.

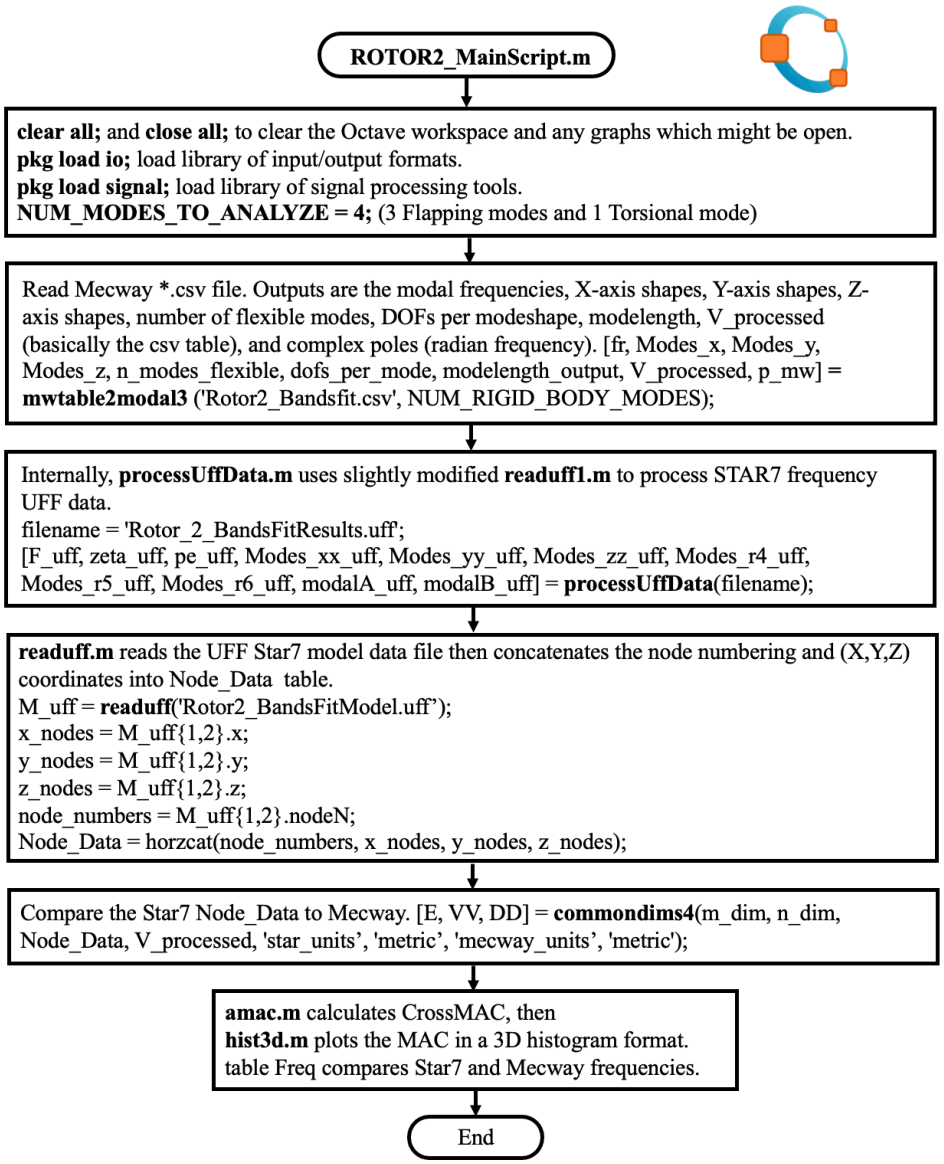

Figure 10. Flowchart of main program and the subroutines it invokes.

This flowchart begins by the main program ROTOR2_MainScript.m first performing a clear all, followed by a close all, to clear the GNU Octave workspace and any graphs that might be open from previous analyses. Then the number of modes to be analyzed was set to 4, as there were 3 flapping modes and 1 torsional mode. ROTOR2_MainScript.m was programmed by Marc Lamparelli and is listed in **Algorithm A1** in its entirety. Helpful comment statements were delineated by the % symbol. In all, there are 67 lines of code and comments in this main program, which calls upon the following seven subroutines.

The next step in the flowchart was to read and process the Rotor2_Bandsfit.csv file generated by the Mecway analytical finite element model using the mwtable2modal3.m subroutine. The mwtable2modal3.m subroutine was programmed by Marc Lamparelli and is listed in **Algorithm A2**. There are 95 lines of code and comments in this subroutine.

Then the Rotor_2_BandsFitResults.uff file was analyzed by the processUffData.m subroutine. The processUffData.m subroutine was also programmed by Marc Lamparelli and is listed in **Algorithm A3**. There are 76 lines of code and comments in this subroutine.

Following this step, readuff.m read the UFF STAR7 model data file and concatenated the node numbering and (X,Y,Z) coordinates into Node_Data table. The subroutine readuff.m, version 1.3.0, file shown in **Figure 10** and **Table 6** is open source software programmed by Dr. Primoz Čermelj, Faculty of Mechanical Engineering, University of Ljubljana, Slovenia. Readuff1.m is our slightly edited version of the readuff.m, programmed by Primoz Čermelj, where we needed to account for the specific format of the STAR7 frequency UFF data. Specifically, line 1,100 of readuff.m was changed from if six ~= 6, to if six ~= 4 to create readuff1.m. Readuff.m and readuff1.m are each over 1500 lines of code and comments, making these two the largest open source packages used by us.

Table 6. GNU Octave packages, descriptions, and primary programmer.

GNU Octave package	Description	Programmer
Rotor2_MainScript.m	Main Program. See **Algorithm A1**. Calls the following subroutines.	Marc Lamparelli
mwtable2modal3.m	Reads Mecway *.csv file. See **Algorithm A2**.	Marc Lamparelli
processUffData.m	Load and parse STAR7 UFF data. See **Algorithm A3**.	Marc Lamparelli
readuff1.m	Processes STAR7 frequency UFF data for processUffData.m	Primoz Čermelj
commondims4.m	Compare STAR7 Node_Data to Mecway. See **Algorithm A4**.	Marc Lamparelli
readuff.m	Reads STAR7 geometry for commonsdim4	Primoz Čermelj
amac.m	Calculate CrossMAC between STAR7 and Mecway	Anders Brandt
hist3d.m	Plots the 3-D MAC histograms. See **Algorithm A5**.	Marc Lamparelli

Commonsdim4.m compared the STAR7 Node_Data table with the information from the Mecway finite element analysis package. The commonsdim4.m subroutine was programmed by Marc Lamparelli and is listed in **Algorithm A4**. At 215 lines of code and comments, this subroutine is the largest of the open source packages that we programmed.

The amac.m subroutine then calculated the desired CrossMAC matrix. The amac.m, version 1.0, subroutine shown in **Figure 10** and **Table 6** was written by

Anders Brandt [31]. This subroutine, with 29 lines of code and comments, is part of ABRAVIBE [32–34] which is a popular, free MATLAB and Octave toolbox, as well as an educational resource for teaching and learning vibration analysis, which was also developed by Anders Brandt. It offers functions for signal processing, modal analysis, rotating machinery analysis, and simulations. Dr. Brandt is currently professor and head of the Department of Mechanical and Production Engineering, Aarhus University, Denmark.

When exporting shapes via UFF from STAR7, the scaling is always Unity Modal Mass (UMM). The scaling from the Mecway *.csv file is also UMM when the S.I. (metric) coordinates are used, and we did use S.I. units. While MAC and CrossMAC calculations are not dependent on similarly scaled mode shapes, other processes such as FRF creation require a specific scaling (Modal A) which is easily created from UMM shapes. The following equation was described by Pastor et al. [3] and is used by the amac.m subroutine as the scalar or dot product of two sets of mode shape vectors, that of the experimental modal analysis, ψ_{EMA}, and that of the analytical finite element analysis, ψ_{FEA}.

$$CrossMAC(i,j) = \frac{\left|\{\psi_{EMA}\}_i^H \quad \{\psi_{FEA}\}_j\right|^2}{(\{\psi_{EMA}\}_i^H \quad \{\psi_{EMA}\}_i) \quad (\{\psi_{FEA}\}_j^H \quad \{\psi_{FEA}\}_j)}$$

In the above equation, ψ represents a mode shape eigenvector, which is either from the Mecway analytical finite element analysis or the STAR7 experimental modal analysis. The superscripts H denote the Hermitian transpose of the mode shape eigenvector, which may also be called the conjugate transpose or the Hermitian conjugate. Mode numbers i and j vary from one to four. Our Cross-Application Modal Assurance Criterion is not symmetric, because we are comparing the Mecway analytical finite element analysis FEA vs. STAR7 experimental modal analysis EMA. This CrossMAC takes the form of the coherence function, which varies between zero and one. CrossMAC values near unity are preferred along the main diagonal of the CrossMAC matrix, $i = j$, to show consistency between identical modes of vibration. CrossMAC values near zero are preferred for the off-diagonal elements, $i \neq j$, to indicate the orthogonality of different modes. In the amac.m subroutine, the above equation is written as a single line of code as follows, where V1 denotes ψ_{EMA}, and V2 denotes ψ_{FMA}. The apostrophe superscript denotes transpose, the asterisk superscript denotes the complex conjugate, and together the apostrophe and asterisk superscript pair denotes the Hermitian transpose. The caret symbol (^) is used to declare an exponent, indicating that the number (2) immediately following the ^ symbol is the power to which the preceding quantity should be raised. In this case ^2 indicates the square of that quantity.

$$\mathrm{M(m1,m2)} = \\ \mathrm{(abs(V1(:,m1)'\times V2(:,m2))\text{^}2/(V1(:,m1)'\times V1(:,m1))/(V2(:,m2)'\times V2(:,m2)));}$$

The final subroutine which we used was hist3d.m, which was also programmed by Marc Lamparelli. This is the shortest subroutine, consisting of only 15 lines of code.

Listed in **Algorithm A5**, this subroutine plots the 3-D MAC histograms.

This makes for a total of 468 lines of open source code and comments programmed by co-author Marc Lamparelli in **Appendix A Algorithms A1–A5**.

The files shown in **Figure 10** and **Table 6** have a *.m file type, which is compatible with both MATLAB and GNU Octave. **Table 6** gives the name of each GNU Octave package, a brief description, and the primary programmer.

4. Discussion

This section discusses the differences between the fixed support and the elastic support conditions used in the Mecway FEA of the helicopter tail rotor.

4.1. STAR7 EMA vs. Mecway FEA—Fixed support

The calculated 2-D CrossMAC values of STAR7 empirical modal analysis vs. the 60 node Mecway analytical finite element analysis with fixed support are listed in **Table 7**. The corresponding 2-D CrossMAC values of STAR7 vs. the 1425 node Mecway model with fixed support are listed in **Table 8**.

Table 7. 2-D CrossMAC of STAR7 and 60 node Mecway mode shapes—Fixed support.

Mode	1-Mecway	2-Mecway	3-Mecway	4-Mecway
1-STAR7	0.98112	0.02787	0.0024496	0.014647
2-STAR7	0.048112	0.84815	0.0044217	0.0037818
3-STAR7	0.0023942	0.0013579	0.95671	0.12332
4-STAR7	0.023499	0.0097819	0.010514	0.63307

Table 8. 2-D CrossMAC of STAR7 and 1425 node Mecway mode shapes—Fixed support.

Mode	1-Mecway	2-Mecway	3-Mecway	4-Mecway
1-STAR7	0.98111	0.025842	0.0022563	0.013558
2-STAR7	0.047936	0.85254	0.003345	0.0038932
3-STAR7	0.0024519	0.0063803	0.97122	0.026708
4-STAR7	0.02419	0.011067	0.002537	0.71684

Pastor et al.'s [3] discussion was particularly helpful in interpreting each of our Cross-Application Modal Assurance Criterion matrices. The off-diagonal elements of the Modal Assurance Criterion can take on values near zero for the following four reasons which we dispel: (a) our system was not non-stationary resulting from changes in the mass, stiffness and damping properties during testing, (b) our system was not non-linear, (c) there was no noise in our mode shapes, and (d) our parameter extraction technique was not invalid for the measured data set. Thus, we concluded that our $i \neq j$ mode shapes were linearly independent. The main diagonal elements of the Modal Assurance Criterion can take on values near unity for the following three reasons which we dispel: (a) the number of response degrees of freedom were not insufficient to distinguish between independent mode shapes, (b) the mode shapes were not a result of unmeasured forces to the system, and (c) the mode shapes were not primarily coherent noise. Thus, we concluded that the $i = j$ mode shapes represent the same motion generally differing only by a scalar and our Cross-Application Modal Assurance Criterion can be used to indicate consistency in our mode shapes.

Tables 7 and **8** are shown in color-coded histogram form in **Figure 11**.

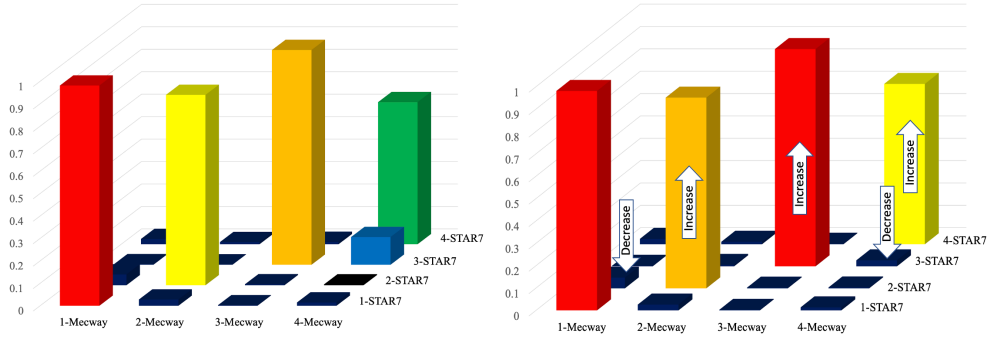

Figure 11. CrossMAC of STAR7 vs. Mecway 60 (left) and 1425 (right) nodes—Fixed support.

In the CrossMAC histograms shown in **Figure 11**, the color coding followed the colors of the rainbow, from red representing the highest values to violet representing the lowest values. Red represented a value between 0.96 (96%) and 1.00 (100%), as seen for the first flapping modes (1,1) and the torsion modes (3,3) of the 1425 node model. Orange represented a value between 0.85 (85%) and 0.96 (96%), as seen for the second flapping modes (2,2) for the 1425 node model and the torsion modes (3,3) of the 60 node model. Yellow represented a value between 0.70 (70%) and 0.85 (85%), as seen for the second flapping modes (2,2) for the 60 node model and the third flapping modes (4,4) for the 1425 node model. Green represented a value between 0.50 (50%) and 0.70 (70%), as seen for the third flapping mode (4,4) for the 60 node model. Blue represented a value between 0.06 (6%) and 0.50 (50%), as seen for one off-diagonal value for the 60 node model (3-STAR7, 4-Mecway). Finally, violet represented a value between 0.0 (0%) and 0.06 (6%), as seen for all other off-diagonal values.

In **Tables 7** and **8** as well as **Figure 11**, it can be seen that the grid refinement from 60 to 1425 nodes resulted in substantial improvements of our Cross-Application Modal Assurance Criterion between the STAR7 experimental model analysis and the analytical Mecway finite element analysis. For the Mecway grid refinement, the comparison between the third flapping modes (4,4) increased from 63.307% (green) to 71.684% (yellow) and the comparison between the second flapping modes (2,2) increased from 84.815% (yellow) to 85.254% (orange). The comparisons between the torsion modes (3,3) also favorably increased from 95.671% (orange) to 97.122% (red). Additionally the highest off-diagonal value in significantly decreased from 12.332% (blue) to 2.6708% (violet) (3-STAR7, 4-Mecway). A small decrease was also noted for the second highest off-diagonal value in the 60 node model (2-STAR7, 1-Mecway). Thus the grid refinement improved the similarity of mode shapes between the STAR7 experimental model analysis and the analytical Mecway finite element analysis by raising the lowest main-diagonal elements and lowering the highest off-diagonal elements in the CrossMAC, confirming the importance of mesh density for accurately capturing higher order tail rotor blade dynamics.

4.2. STAR7 EMA vs. Mecway FEA—Elastic support

Table 9 lists the modal frequencies for the STAR7 polynomial method, the Mecway 1425 node model, for the case of fixed support at the base of the tail rotor

blade, and the Mecway 1425 node model, for the case of elastic support at the base of the tail rotor blade. The change from fixed support to elastic support of the 1425 node model gave frequencies much closer to the STAR7 polynomial method.

Table 9. STAR7 vs. Mecway modal frequencies—Fixed and elastic support.

Mode Number	Mode type	STAR7 polynomial method	Mecway 1425 nodes fixed support	Mecway 1425 nodes elastic support
1	Flapping	19.621 Hz	18.579 Hz	15.507 Hz
2	Flapping	110.94 Hz	116.27 Hz	97.376 Hz
3	Torsion	186.61 Hz	244.72 Hz	209.07 Hz
4	Flapping	278.32 Hz	324.80 Hz	272.72 Hz

The 2-D CrossMAC values of STAR7 vs. the 1425 node Mecway model with elastic support are listed in **Table 10**. **Table 10** (Mecway elastic support) is nearly identical to **Table 8** (Mecway fixed support). The largest change between **Tables 8** and **10** is that the comparison between the third flapping modes (4,4) improved favorably from 71.684% (fixed support) to 72.897% (elastic support).

Table 10. 2-D CrossMAC of STAR7 and 1425 node Mecway mode shapes—Elastic support.

Mode	1-Mecway	2-Mecway	3-Mecway	4-Mecway
1-STAR7	0.98092	0.027948	0.0022504	0.013017
2-STAR7	0.052086	0.85277	0.0033563	0.0044589
3-STAR7	0.0024356	0.0056265	0.97013	0.027813
4-STAR7	0.022906	0.016312	0.002577	0.72897

The CrossMAC of the STAR7 and 1425 Node Mecway mode shapes, with elastic support, listed in **Table 10** is portrayed as a histogram in the left side of **Figure 12**. The right side of **Figure 12** graphs the modal frequencies of the EMA performed by STAR7 vs. the Mecway FEA with 1425 nodes and elastic support from **Table 9**. The Excel =PEARSON statistical correlation of the right side of **Figure 12** was 99.109%, indicating a very high cross correlation between these experimental and the analytically calculated modal frequencies when grid refinement and an elastic support were employed. The uncertainty in the abscissa frequencies in the frequency correlation scatter plot was very small, with the first empirical mode having a frequency spread of ±0.01 Hz, the second empirical mode having a frequency spread of ±0.47 Hz, the third empirical mode having a frequency spread of ±0.04 Hz, and the fourth empirical mode having a frequency spread of ±0.28 Hz.

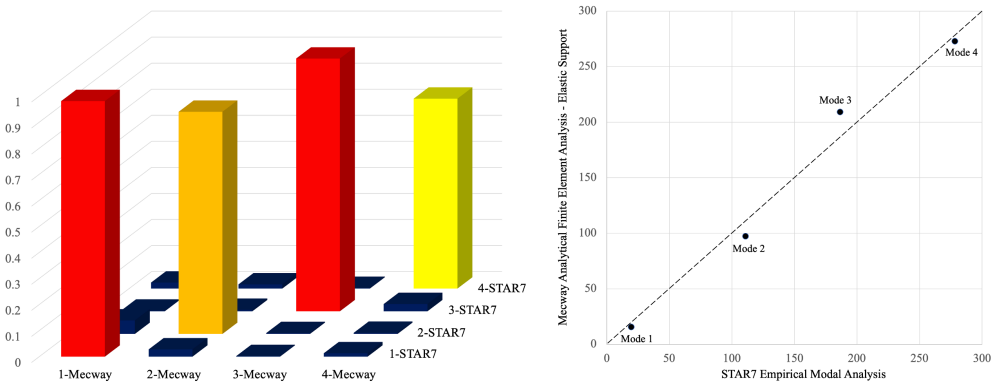

Figure 12. CrossMAC (left) of mode shapes and frequency correlation scatter plot (right).

5. Conclusion

By the use of GNU Octave open source algorithms, we were successful in integrating the modal outputs of STAR7 EMA and Mecway FEA. This enabled calculating the Cross-Application Modal Assurance Criterion between STAR7 experimental modal analysis and Mecway analytical finite element analysis for a stationary tail rotor blade of a Huey UH-1H helicopter. The CrossMACs created indicated high correlation values for main-diagonal elements (mode-i compared to mode-i), as well as favorably low correlation values for off-diagonal elements (mode-i compared to mode-j) which indicated the orthogonality of these modes, as desired. The grid refinement from a 60 node to a 1425 node Mecway finite element model successfully improved the similarity of mode shape comparisons between the STAR7 experimental model analysis and the analytical Mecway finite element analysis by favorably raising the lowest main-diagonal elements and lowering the two highest off-diagonal element in the CrossMAC. A Modal Assurance Criterion near 100% indicated a high correlation between mode shape vectors, while a value near 0% meant that the mode shape vectors were not correlated. Our fixed-support same-mode results indicated that the CrossMAC between EMA and analytical FEA achieved a cross-correlation of over 98.11% for the first flapping mode (1,1). Grid refinement increased the cross-correlation from 95.671% to 97.122% for the torsion mode (3,3), from 84.815% to 85.254% for the second flapping mode (2,2), and from 63.307% to 71.684% for the third flapping mode (4,4). With grid refinement, the maximum mixed mode cross-correlation fell from 12.332% to only 4.7936%, indicating a high degree of independent mode shapes.

Changing from a fixed support to an elastic support, within Mecway, of the helicopter tail rotor blade gave us a Pearson statistical correlation coefficient of 99.1% between the STAR7 experimental modal frequencies and Mecway analytical finite element modal frequencies, where 100% is the maximum value and represents a perfect correlation. CrossMAC comparison between the third flapping modes (4,4) improved favorably from 71.684% (fixed support) to 72.897% (elastic support).

The most significant contribution of our work was the development of GNU Octave based open source algorithms that enabled direct Cross-Application Modal Assurance Criterion computation between STAR7 empirical mode shapes and Mecway analytical finite element mode shapes. Our free open source algorithms will allow future studies to similarly compare and quantify complex experimentally measured and analytically modeled vibrational modes. Beyond the UH-1H helicopter, our open source code is broadly applicable to rotorcraft model updating, health monitoring, and vibration reduction studies, as well as it offers a valuable template for integrating experimental and computational analyses in aerospace structural dynamics.

Limitations of this study included the tail rotor blade being stationary and the lack of empirical investigation under rotational conditions. Future work could include wireless instrumentation of the tail rotor blades and studying mode shapes and frequencies during helicopter pitch, yaw, and roll maneuvers.

Author contributions: Conceptualization, MDL and DJW; methodology, MDL and

DJW; validation, MDL, DJW and TYW; formal analysis, MDL, DJW and TYW; investigation, MDL, DJW and TYW; writing—original draft preparation, MDL and DJW; writing—review and editing, MDL, DJW and TYW; visualization, MDL and DJW; supervision, DJW; project administration, DJW. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional review board statement: Not applicable.

Informed consent statement: Not applicable.

Data availability statement: The *.m open source code listed in **Table 6** is stored on ResearchGate.

Acknowledgment: Donna Robinson Winarski is acknowledged for her support of this research.

Conflict of interest: The authors declare no conflict of interest.

Abbreviations

Abbreviation	Full name
*.CSV	Comma Separated Variables plain text, tabulated data format
*.FRF	Frequency Response Function file. 45 *.FRF used in this study
*.LIML	Library Interface Markup Language, Mecway file type, XML-based
*.M	MATLAB code file type, an ASCII text file also used by GNU Octave
*.RST	Re-Structured Text file containing mode shape data in binary format
*.UFF	Universal File Format
*.XML	Extensible Markup Language file type, which is a plain text file
=PEARSON	Excel statistical function—correlation between two variables
B&K	Brüel & Kjær, as of 2019 now called HBK (Hottinger Brüel & Kjær)
EMA	Experimental Modal Analysis
FEA	Finite Element Analysis
GNU	GNU project, launched in 1983 by Richard Stallman. Massive free software collaboration
H	Complex conjugate transpose of a vector
MAC	Modal Assurance Criterion, either a 3-D histogram or a 2-D table
CrossMAC	Cross-Application MAC between STAR7 EMA and Mecway FEA
S.I.	Système International d'Unités (International System of Units)
SHP	Shaft Horse Power
UFF	Universal File Format
UMM	Unity Modal Mass
Ψ	Mode Shape Eigenvector used in CrossMAC equation
ψ^H	Hermitian Transpose of Mode Shape Eigenvector
Ω	Frequency, which ranged over $0 \leq \omega \leq 400$ Hz

References

1. Brown DL, Allemang RJ. The modern era of experimental modal analysis. Sound and Vibration. 2007; 41(1): 16–33. Available online: http://www.sandv.com/downloads/0701alle.pdf
2. Allemang RJ. The modal assurance criterion—Twenty years of use and abuse. Sound and Vibration. 2003; 37(8): 14–23. Available online: http://www.sandv.com/downloads/0308alle.pdf
3. Pastor M, Binda M, Harčarik T. Modal assurance criterion. Procedia Engineering. 2012; 48: 543–548.
4. Rezaifar O, Kabir MZ, Taribakhsh M, et al. Dynamic behaviour of 3D-panel single-storey system using shaking table testing. Engineering Structures. 2008; 30(2): 318–337.
5. Santos FLM, Peeters B, Van der Auweraer H, et al. Experimental damage detection of a helicopter main rotor blade based on modal properties. In: Proceedings of the International Conference on Engineering Structural Dynamics ICEDyn-2013; 17–19 June 2013; Sesimbra, Portugal.

6. Grappasonni C, Ameri N, Coppotelli G, et al. Dynamic identification of helicopter structures using operational modal analysis methods in the presence of harmonic loading. In: Proceedings of the ISMA2012-USD2012; 17–19 September 2012; Leuven, Belgium.
7. Safari Tarbozagh A, Khanahmadi M, Kontoni DPN, et al. A Vibration-Driven Computational Method for Localizing Interfacial Debonding in Concrete-Filled Steel Tubes. Journal of Vibration Engineering and Technologies. 2026; 14(3): 102.
8. Khanahmadi M, Gholhaki M, Rezaifar O, et al. Signal processing methodology for detection and localization of damages in columns under the effect of axial load. Measurement. 2023; 211: 112595.
9. Dai G, Ji G. Impacts of the weight coefficient and modal assurance criterion of large structures on observation station selection and optimization. Journal of Vibroengineering. 2018; 20(1): 503–518. doi: 10.21595/jve.2017.18206
10. Winarski D, Lamparelli M, Landry K, et al. Creating a vibrational digital-twin of a Bell UH-1H helicopter tail-rotor blade for use in simulating centrifugal stiffening. Sound and Vibration. 2025; 59(5). doi: 10.59400/sv3662
11. Spectral Dynamics, Inc. STAR7 Operating Manual, Rev. 1b. Spectral Dynamics, Inc; 2025.
12. CATS: Computer Aided Suite Test. Available online: https://www.scribd.com/document/394925642/665498-MATERIJALI-1-Skripta-Listopad-2013 (accessed on 10 June 2025).
13. Mecway FEA Download. Available online: https://www.mecway.com/download/ (accessed on 24 December 2025).
14. Mecway Finite Element Analysis, Version 33, 2026. Available online: https://www.mecway.com/manual.pdf (accessed on 3 January 2026).
15. Mecway FEA. Available online: https://www.mecway.com (accessed on 21 December 2025).
16. Bell UH-1 Iroquois. Available online: https://en.wikipedia.org/wiki/Bell_UH-1_Iroquois (accessed on 17 July 2025).
17. File: UH1 Huey - Fly Navy 2017 (26938005897).jpg. Available online: https://commons.wikimedia.org/w/index.php?curid=69215127 (accessed on 19 July 2025).
18. Product Data: Piezoelectric Charge Accelerometer Types 4393 and 4393-V. Available online: https://www.bksv.com/media/doc/bp2043.pdf (accessed on 30 July 2025).
19. Blevins RD. Formulas for Natural Frequency and Mode Shape. Krieger Publishing; 2001. p. 108.
20. Lima MACF. Rotating Cantilever Beams: Finite Element Modeling and Vibration Analysis [Master's Thesis]. Universidade do Porto; 2012.
21. Yoo HH, Shin SH. Vibration analysis of rotating cantilever beams. Journal of Sound and Vibration. 1998; 212(5): 807–828. doi: 10.1006/jsvi.1997.1469
22. Wright AD, Smith CE, Thresher RW, et al. Vibration modes of centrifugally stiffened beams. Journal of Applied Mechanics. 1982; 49(1): 197–202.
23. Technical Documentation: Impact Hammer Type 8202. Available online: https://media.hbkworld.com/m/7a8ee3f9bea9db2b/original/Impact-Hammer-Type-8202.pdf (accessed on 30 July 2025).
24. Serridge M, Licht T. Piezoelectric Accelerometers and Vibration Preamplifiers: Theory and Application Handbook. Brüel and Kjær; 1987.
25. Son M, Shin D, Shin D. An XML Based User Context Language for Personalized Service in Ubiquitous Digital Library. In: Proceedings of the 2008 International Conference on Advanced Language Processing and Web Information Technology; 23–25 July 2008; Liaoning, China. pp. 367–372.
26. GNU Octave. Available online: https://en.wikipedia.org/wiki/GNU_Octave (accessed on 28 December 2025).
27. GNU Octave. About. Available online: https://octave.org/about (accessed on 1 January 2026).
28. GNU General Public License, Version 3. Available online: https://www.gnu.org/licenses/gpl-3.0.html (accessed on 2 January 2026).
29. GNU Octave. GNU Octave-11.1.0-w64-installer.exe for Microsoft Windows. Available online: https://octave.org/download (accessed on 3 January 2026).
30. GNU Octave Online Manual. Available online: https://docs.octave.org/latest/ (accessed on 28 December 2025).
31. Curriculum Vitae—Anders Brandt. Available online: https://abravibe.com/wp-content/uploads/2022/07/AndersBrandtCV.pdf (accessed on 3 January 2026).
32. Brandt A. ABRAVIBE—A MATLAB toolbox for noise and vibration analysis and teaching. Available online: https://lnu.diva-portal.org/smash/record.jsf?pid=diva2%3A1033066&dswid=-9794 (accessed on 31 December 2025).
33. Brandt A. The ABRAVIBE Toolbox for Teaching Vibration Analysis and Structural Dynamics. In: Special Topics in Structural Dynamics. Springer; 2013. doi: 10.1007/978-1-4614-6546-1_13
34. ABRAVIBE—A Toolbox for Teaching and Learning Vibration Analysis. Available online: http://www.sandv.com/downloads/1311bran.pdf (accessed on 31 December 2025).

Appendix A

Algorithm A1 GNU Octave open source main program ROTOR2_MainScript.m

```
clear all;
close all;
pkg load io;
pkg load signal;
% --- Configuration and Data Loading ---
% Define constants at the top for easy modification
NUM_RIGID_BODY_MODES = 0; % Number of rigid body modes to remove
NUM_MODES_TO_ANALYZE = 4; % Number of modes for scaling and MAC analysis
% Extract modal information from MW displacement data
[fr, Modes_x, Modes_y, Modes_z, n_modes_flexible, dofs_per_mode, modelength_output, V_processed, p_mw] =
mwtable2modal3('Rotor2_Bandsfit.csv', NUM_RIGID_BODY_MODES);
% p_mw is the poles vector.
% modelength is the length of each mode.
% --- External (UFF) Data Processing ---
% Load and parse UFF file data
filename = 'Rotor_2_BandsFitResults.uff';
[F_uff, zeta_uff, pe_uff, Modes_xx_uff, Modes_yy_uff, Modes_zz_uff, ...
        Modes_r4_uff, Modes_r5_uff, Modes_r6_uff, modalA_uff, modalB_uff] = processUffData(filename);
% --- MAC Calculations ---
% Load model data for node coordinates
try
   M_uff = readuff('Rotor2_BandsFitModel.uff');
   x_nodes = M_uff{1,2}.x;
   y_nodes = M_uff{1,2}.y;
   z_nodes = M_uff{1,2}.z;
   node_numbers = M_uff{1,2}.nodeN;
   Node_Data = horzcat(node_numbers, x_nodes, y_nodes, z_nodes);
catch
   error('Error loading BEAMmodel.uff. Make sure the file exists and is accessible.');
end
% --- Find Common Dimensions of STAR and Mecway
% Adjust dimensions based on your specific DOF mapping logic
% Assuming 'm' and 'n' refer to dimensions related to your model or DOFs
m_dim = 60; % Original FE model total DOFs or nodes
n_dim = 45;  % Number of DOFs being considered for analysis (e.g., z-DOFs)
DOF = (1:modelength_output)';
% Ensure that Vmw is processed consistently with the DOF mapping
% commondims should ideally align the DOFs between your models.
% The current usage `V = Vmw(DOF,:);` seems to extract all DOFs.
% This needs to be consistent with how `mwtable2modal` extracted `Modes_x,y,z`.
[E, VV, DD] = commondims4(m_dim, n_dim, Node_Data, V_processed,'star_units','metric','mecway_units','metric');
DOFmw = E(:,2); % Assuming this selects the common DOFs
% Ensure that Modes_x, Modes_y, Modes_z from mwtable2modal are already aligned or correctly selected
% Modes_z, Modes_x, Modes_y need to be filtered by DOFmw if mwtable2modal didn't do it.
% Assuming Modes_z (and x,y) are already filtered by DOFz for the FRF calc.
% For MAC, ensure they are compatible.
Modes_x_mac = Modes_x(DOFmw, 1:NUM_MODES_TO_ANALYZE);
Modes_y_mac = Modes_y(DOFmw, 1:NUM_MODES_TO_ANALYZE);
Modes_z_mac = Modes_z(DOFmw, 1:NUM_MODES_TO_ANALYZE);
Modes_xyz_mw = vertcat(Modes_x_mac, Modes_y_mac, Modes_z_mac);
% Modes from UFF are already xx, yy, zz (presumably at the same DOFs as Mecway output)
Modes_xxyyzz_uff = vertcat(Modes_xx_uff, Modes_yy_uff, Modes_zz_uff);
% Perform MAC calculations
AutoMAC_ema_z = amac(Modes_zz_uff);
CrossMAC_z = amac(Modes_zz_uff, Modes_z_mac);
AutoMAC_ema_xyz = amac(Modes_xxyyzz_uff);
AutoMAC_xyz_mw = amac(Modes_xyz_mw);
CrossMACxyz = amac(Modes_xxyyzz_uff, Modes_xyz_mw);
% --- Plotting MAC Results ---
%figure;
hist3d(AutoMAC_xyz_mw);
title('AutoMAC (Mecway XYZ)');
hist3d(CrossMAC_z);
```

```
title('CrossMAC (UFF Z vs Mecway Z)');
hist3d(CrossMACxyz);
title('CrossMAC (UFF XYZ vs Mecway XYZ)');
% Compare Frequencies EMA/FEA
Freq = [F_uff fr];
```

Algorithm A2 GNU Octave open source subroutine mwtable2modal3.m

```
function [fr, Modes_x, Modes_y, Modes_z, n, m, modelength, V_processed, p_mw] = mwtable2modal3(filename, NUM_RIGID_BODY_MODES)
%MWTABLE2MODAL2 Processes modal analysis data from a Mecway CSV file.
%
%       [fr, Modes_x, Modes_y, Modes_z, n, m, modelength, V_processed, p_mw] = MWTABLE2MODAL2(filename, NUM_RIGID_BODY_MODES)
%
%  Inputs:
%     filename - A string specifying the path to the CSV data file.
%                The file must contain columns: Mode ID, Frequency (Hz),
%                Node ID, Node X, Node Y, Node Z, X-Disp, Y-Disp, Z-Disp.
%     NUM_RIGID_BODY_MODES - The number of rigid body modes to remove from the data (0 in this case).
%
%  Outputs:
%     fr      - A vector of flexible natural frequencies (Hz).
%     Modes_x  - A matrix where each column represents the x-components
%                of a mode shape.
%     Modes_y  - A matrix where each column represents the y-components
%                of a mode shape.
%     Modes_z  - A matrix where each column represents the z-components
%                of a mode shape.
%     n       - The number of remaining flexible modes.
%     m       - The length of each mode vector (number of DOFs per mode), same as modelength.
%     modelength - The length of each mode vector (number of DOFs per mode).
%     V_processed - The filtered raw data matrix (flexible modes only).
%     p_mw    - Poles from Mecway frequencies (complex values).
  % Load the MW displacement file
  try
    % Read the CSV data (assuming no header row in the file)
    V = csvread(filename);
    % Remove the 1st row (assuming it's a header or contains non-data)
    % Note: If your CSV is saved without headers, remove this line.
    % For safety, we keep it as it was in the original script.
    V(1,:) = [];
  catch
    error('Error loading %s. Make sure the file exists and is accessible.', filename);
  end
  % --- CRITICAL FIX: Use unique modes to calculate modelength ---
  % Get the list of unique mode identifiers
  unique_modes = unique(V(:,1));
  % Calculate the total number of modes in the file (including rigid body modes)
  n_total = length(unique_modes);
  % Calculate the length of each mode vector (DOFs per mode).
  % This is robust against non-consecutive mode numbers.
  modelength = size(V, 1) / n_total;
  % Check if the division resulted in an integer mode length
  if mod(size(V, 1), n_total) ~= 0
    error('Mode length calculation failed. Total rows (%d) is not divisible by total unique modes (%d). Check CSV file structure.', size(V, 1), n_total);
  end
  % --- Extract Frequencies ---
  % Preallocate fr_all for efficiency
  fr_all = zeros(n_total, 1);
  for i = 1:n_total
    % Get the current unique mode ID
    current_mode_id = unique_modes(i);
    % Find all rows corresponding to this mode
    mode_rows = V(V(:,1) == current_mode_id, :);
```

```
    % Frequencies are in the second column, taken from the first row of that mode block
    % NOTE: Assuming frequency is constant for all rows of a mode block.
    fr_all(i) = mode_rows(1, 2);
end
% Remove rigid body modes from frequencies
% The frequencies are now ordered by the unique_modes list, so we can filter directly
fr = fr_all(NUM_RIGID_BODY_MODES + 1 : end);
% Calculate p_mw (Poles from Mecway frequencies)
p_mw = complex(0, -2 * pi * fr);
% --- Process Mode Shapes ---
% Filter the unique modes to only include flexible modes
flexible_mode_ids = unique_modes(NUM_RIGID_BODY_MODES + 1 : end);
n = length(flexible_mode_ids); % 'n' is the number of flexible modes (4 in your case)
m = modelength;   % 'm' is the length of each mode vector (15 in your case)
% Keep only the data rows corresponding to the flexible modes
V_processed = [];
for i = 1:n
    current_mode_id = flexible_mode_ids(i);
    % Append the block of rows for the current flexible mode
    V_processed = [V_processed; V(V(:,1) == current_mode_id, :)];
end
% Check: Total rows in V_processed must be n * modelength
if size(V_processed, 1) ~= n * modelength
    error('Internal processing error: The size of V_processed does not match the expected size (n * modelength).');
end
% Initialize mode shape matrices (now correctly sized: 15 rows x 4 columns)
Modes_x = zeros(modelength, n);
Modes_y = zeros(modelength, n);
Modes_z = zeros(modelength, n);
% Extract mode shapes for x, y, and z components from the contiguous V_processed
for i = 1:n
    % V_processed is now contiguous (Mode 1 block, then Mode 3 block, etc.)
    start_idx = (i - 1) * modelength + 1;
    end_idx = i * modelength;
    % The columns for x, y, z displacements are 7, 8, 9 respectively.
    Modes_x(:,i) = V_processed(start_idx:end_idx, 7);
    Modes_y(:,i) = V_processed(start_idx:end_idx, 8);
    Modes_z(:,i) = V_processed(start_idx:end_idx, 9);
end
end
```

Algorithm A3 GNU Octave open source subroutine processUFFData.m

```
function [F_uff, zeta_uff, pe_uff, Modes_xx_uff, Modes_yy_uff, Modes_zz_uff, ...
        Modes_r4_uff, Modes_r5_uff, Modes_r6_uff, modalA_uff, modalB_uff] = processUffData(filename)
% processUffData: Loads, parses, and processes experimental UFF file data.
%
%  Inputs:
%     filename - A string specifying the path to the UFF file (e.g., 'BEAMresults.uff').
%
%  Outputs:
%     F_uff       - Frequencies extracted from the UFF data.
%     zeta_uff    - Damping ratios extracted from the UFF data.
%     pe_uff      - Poles calculated from frequencies and damping ratios.
%     Modes_xx_uff - Mode shapes (r1 components) from UFF.
%     Modes_yy_uff - Mode shapes (r2 components) from UFF.
%     Modes_zz_uff - Mode shapes (r3 components) from UFF.
%     Modes_r4_uff - Mode shapes (r4 components) from UFF.
%     Modes_r5_uff - Mode shapes (r5 components) from UFF.
%     Modes_r6_uff - Mode shapes (r6 components) from UFF.
%     modalA_uff   - Modal A matrix components from UFF.
%     modalB_uff   - Modal B matrix components from UFF.
   % --- External (UFF) Data Processing ---
   % Load and parse UFF file data
   try
      Ve = readuff1(filename);
```

```
    catch
        error(['Error loading ', filename, '. Make sure the file exists and is accessible.']);
    end
    num_uff_modes = length(Ve);
    % Check if any modes were loaded
    if num_uff_modes == 0
        warning('No modes found in the UFF file.');
        % Return empty arrays for all outputs
        F_uff = []; zeta_uff = []; pe_uff = [];
        Modes_xx_uff = []; Modes_yy_uff = []; Modes_zz_uff = [];
        Modes_r4_uff = []; Modes_r5_uff = []; Modes_r6_uff = [];
        modalA_uff = []; modalB_uff = [];
        return;
    end
    % Preallocate arrays for efficiency
    % Determine dimensions from the first mode, assuming consistency
    first_mode_r1_size = size(Ve{1}.r1, 1);
    first_mode_r2_size = size(Ve{1}.r2, 1);
    first_mode_r3_size = size(Ve{1}.r3, 1);
    first_mode_r4_size = size(Ve{1}.r4, 1);
    first_mode_r5_size = size(Ve{1}.r5, 1);
    first_mode_r6_size = size(Ve{1}.r6, 1);
    first_mode_modalA_size = size(Ve{1}.modalA, 1);
    first_mode_modalB_size = size(Ve{1}.modalB, 1);
    eigVal_uff = zeros(1, num_uff_modes);
    Modes_xx_uff = zeros(first_mode_r1_size, num_uff_modes);
    Modes_yy_uff = zeros(first_mode_r2_size, num_uff_modes);
    Modes_zz_uff = zeros(first_mode_r3_size, num_uff_modes);
    Modes_r4_uff = zeros(first_mode_r4_size, num_uff_modes);
    Modes_r5_uff = zeros(first_mode_r5_size, num_uff_modes);
    Modes_r6_uff = zeros(first_mode_r6_size, num_uff_modes);
    modalA_uff = zeros(first_mode_modalA_size, num_uff_modes);
    modalB_uff = zeros(first_mode_modalB_size, num_uff_modes);
    for i = 1:num_uff_modes
        eigVal_uff(i) = Ve{1,i}.eigVal;
        Modes_xx_uff(:,i) = Ve{1,i}.r1;
        Modes_yy_uff(:,i) = Ve{1,i}.r2;
        Modes_zz_uff(:,i) = Ve{1,i}.r3;
        Modes_r4_uff(:,i) = Ve{1,i}.r4;
        Modes_r5_uff(:,i) = Ve{1,i}.r5;
        Modes_r6_uff(:,i) = Ve{1,i}.r6;
        modalA_uff(:,i) = Ve{1,i}.modalA;
        modalB_uff(:,i) = Ve{1,i}.modalB;
    end;
    F_uff = imag(eigVal_uff)'; % Frequencies from UFF
    sigma_uff = real(eigVal_uff)'; % Damping from UFF
    % Handle potential division by zero if F_uff contains zeros
    zeta_uff = zeros(size(sigma_uff));
    non_zero_freq_idx = F_uff ~= 0;
    zeta_uff(non_zero_freq_idx) = sigma_uff(non_zero_freq_idx) ./ F_uff(non_zero_freq_idx);
    % Create poles from UFF data
    pe_uff = fz2poles(F_uff, zeta_uff);
end
```

Algorithm A4 GNU Octave open source subroutine commondims4.m

```
function [E, VV, DD] = commondims4(m, n, D, V, varargin)
% COMMONDIMS3 - Find common dimensions between STAR modal and Mecway data
%               Includes flexible axis transformation and prefers English units.
%
% INPUTS:
%   m - Number of Mecway DOFs/nodes to consider
%   n - Number of STAR DOFs/nodes to consider
%   D - STAR node data [node_id, x, y, z]
%       (Units are determined by the 'star_units' optional argument) % MODIFIED DOC
%   V - Mecway node data [node_id, ?, ?, node_id, x, y, z]
%       (Units are determined by the 'mecway_units' optional argument)
```

```
%   varargin:
%       'star_units' (string, optional): Units of the STAR data (D).
%                                   Accepted values: 'metric' or 'english' (default: 'english'). % NEW DOC
%       'mecway_units' (string, optional): Units of the Mecway data (V).
%                                   Accepted values: 'metric' or 'english' (default).
%       'mecway_axis_swap' (string, optional): Specifies axis swap for Mecway data.
%                                       Accepted values: 'yz_swap', 'none' (default).
%       'mecway_x_invert' (boolean, optional): If true, inverts the X-axis for Mecway data. Default: false.
%       'mecway_y_invert' (boolean, optional): If true, inverts the Y-axis for Mecway data. Default: false.
%       'mecway_z_invert' (boolean, optional): If true, inverts the Z-axis for Mecway data. Default: false.
%
% OUTPUTS:
%   E - Common dimensions matrix [STAR_node_id, Mecway_node_id, x_common, y_common, z_common]
%       Note: x,y,z in E will be in the *English* unit system for consistency.
%   VV - Processed Mecway data (first m rows, in original units)
%   DD - Original STAR data (first n rows, in original units)
    % Input validation
    if nargin < 4
        error('commondims3:InvalidInput', 'Four required input arguments: m, n, D, V.');
    end
    % Default units for data and axis transformation
    star_units = 'english'; % NEW: Default to English units for STAR data
    mecway_units = 'english'; % Default to English units for Mecway data
    mecway_axis_swap = 'none'; % Default axis swap: no swap
    mecway_x_invert = false; % Default: No X-axis inversion
    mecway_y_invert = false; % Default: No Y-axis inversion
    mecway_z_invert = false; % Default: No Z-axis inversion
    % Parse varargin for optional arguments
    if nargin > 4
        idx = 1;
        while idx <= length(varargin)
            if ischar(varargin{idx})
                switch lower(varargin{idx})
                    case 'star_units' % NEW CASE
                        if idx + 1 <= length(varargin)
                            unit_str = lower(varargin{idx+1});
                            if strcmp(unit_str, 'metric') || strcmp(unit_str, 'english')
                                star_units = unit_str;
                            else
                                error('commondims3:InvalidUnits', 'Invalid value for "star_units". Must be "metric" or "english".');
                            end
                            idx = idx + 1;
                        else
                            error('commondims3:MissingValue', 'Missing value for "star_units" argument.');
                        end
                    case 'mecway_units'
                        if idx + 1 <= length(varargin)
                            unit_str = lower(varargin{idx+1});
                            if strcmp(unit_str, 'metric') || strcmp(unit_str, 'english')
                                mecway_units = unit_str;
                            else
                                error('commondims3:InvalidUnits', 'Invalid value for "mecway_units". Must be "metric" or "english".');
                            end
                            idx = idx + 1; % Increment to skip the value
                        else
                            error('commondims3:MissingValue', 'Missing value for "mecway_units" argument.');
                        end
                    case 'mecway_axis_swap'
                        if idx + 1 <= length(varargin)
                            swap_str = lower(varargin{idx+1});
                            if strcmp(swap_str, 'yz_swap') || strcmp(swap_str, 'none')
                                mecway_axis_swap = swap_str;
                            else
                                warning('commondims3:InvalidAxisSwap', 'Invalid value for "mecway_axis_swap". Accepted: "yz_swap" or "none". Using default "none".');
                            end
                            idx = idx + 1;
```

```
                else
                    error('commondims3:MissingValue', 'Missing value for "mecway_axis_swap" argument.');
                end
            case 'mecway_x_invert'
                if idx + 1 <= length(varargin)
                    invert_val = varargin{idx+1};
                    if islogical(invert_val) || (isnumeric(invert_val) && (invert_val == 0 || invert_val == 1))
                        mecway_x_invert = logical(invert_val);
                    else
                        warning('commondims3:InvalidXInvert', 'Invalid value for "mecway_x_invert". Must be boolean (true/false) or 0/1. Using default false.');
                    end
                    idx = idx + 1;
                else
                    error('commondims3:MissingValue', 'Missing value for "mecway_x_invert" argument.');
                end
            case 'mecway_y_invert'
                if idx + 1 <= length(varargin)
                    invert_val = varargin{idx+1};
                    if islogical(invert_val) || (isnumeric(invert_val) && (invert_val == 0 || invert_val == 1))
                        mecway_y_invert = logical(invert_val);
                    else
                        warning('commondims3:InvalidYInvert', 'Invalid value for "mecway_y_invert". Must be boolean (true/false) or 0/1. Using default false.');
                    end
                    idx = idx + 1;
                else
                    error('commondims3:MissingValue', 'Missing value for "mecway_y_invert" argument.');
                end
            case 'mecway_z_invert'
                if idx + 1 <= length(varargin)
                    invert_val = varargin{idx+1};
                    if islogical(invert_val) || (isnumeric(invert_val) && (invert_val == 0 || invert_val == 1))
                        mecway_z_invert = logical(invert_val);
                    else
                        warning('commondims3:InvalidZInvert', 'Invalid value for "mecway_z_invert". Must be boolean (true/false) or 0/1. Using default false.');
                    end
                    idx = idx + 1;
                else
                    error('commondims3:MissingValue', 'Missing value for "mecway_z_invert" argument.');
                end
            otherwise
                warning('commondims3:UnknownArgument', 'Unknown optional argument: %s', varargin{idx});
        end
    else
        warning('commondims3:UnexpectedArgumentType', 'Unexpected argument type. Skipping: %s', class(varargin{idx}));
    end
    idx = idx + 1; % Increment to next argument
  end
end
% Process input data to limit rows
VV = V(1:min(m, size(V,1)), :);
DD = D(1:min(n, size(D,1)), :);
% Define conversion factors
% 1 meter = 1 / 0.0254 inches = 39.3701 inches
% 1 inch = 0.0254 meters
conversion_factor_m_to_in = 1 / 0.0254;
% --- Prepare STAR data for comparison (TARGET UNIT IS ENGLISH) --- % MODIFIED BLOCK
DD_for_comparison = DD; % Start with the raw STAR data
star_coords_original = DD_for_comparison(:, 2:4); % Extract original x, y, z
if strcmp(star_units, 'metric')
    fprintf('STAR data detected as Metric. Converting to English for comparison.\n');
    star_coords_converted = star_coords_original * conversion_factor_m_to_in;
else % 'english' or default
    fprintf('STAR data detected as English. No unit conversion needed for comparison.\n');
    star_coords_converted = star_coords_original;
```

```
    end
    % Update DD_for_comparison with the converted/unconverted coordinates
    DD_for_comparison(:, 2:4) = star_coords_converted;
    fprintf('STAR comparison data is in English units.\n');
    % --- Prepare Mecway data for comparison (TARGET UNIT IS ENGLISH) ---
    VV_for_comparison = VV; % Start with the raw Mecway data coordinates
    mecway_coords_original = VV_for_comparison(:, 4:6); % Extract original x, y, z
    % Step 1: Convert Mecway data to English if it's currently Metric
    if strcmp(mecway_units, 'metric')
       fprintf('Mecway data detected as Metric. Converting to English for comparison.\n');
       mecway_coords_converted = mecway_coords_original * conversion_factor_m_to_in;
    else % 'english' or default
       fprintf('Mecway data detected as English. No unit conversion needed for comparison.\n');
       mecway_coords_converted = mecway_coords_original;
    end
    % Step 2: Apply axis transformation (swap) to Mecway coordinates
    transformed_mecway_coords = mecway_coords_converted; % Start with converted coords
    if strcmp(mecway_axis_swap, 'yz_swap')
       fprintf('Applying Y-Z axis swap to Mecway coordinates (Y -> Z, Z -> Y).\n');
       temp_y = transformed_mecway_coords(:, 2); % Store original Y
       transformed_mecway_coords(:, 2) = transformed_mecway_coords(:, 3); % New Y is old Z
       transformed_mecway_coords(:, 3) = temp_y; % New Z is old Y
    end
    % Step 3: Apply individual axis inversions
    if mecway_x_invert
       fprintf('Inverting X-axis for Mecway coordinates.\n');
       transformed_mecway_coords(:, 1) = -transformed_mecway_coords(:, 1);
    end
    if mecway_y_invert
       fprintf('Inverting Y-axis for Mecway coordinates.\n');
       transformed_mecway_coords(:, 2) = -transformed_mecway_coords(:, 2);
    end
    if mecway_z_invert
       fprintf('Inverting Z-axis for Mecway coordinates.\n');
       transformed_mecway_coords(:, 3) = -transformed_mecway_coords(:, 3);
    end
    % Update VV_for_comparison with the transformed coordinates
    VV_for_comparison(:, 4:6) = transformed_mecway_coords;
    fprintf('Mecway comparison data is in English units.\n');
    % Tolerance for coordinate matching (in English units, e.g., inches)
    coord_tol = 0.04; % 0.04 inches tolerance. Adjust if your coordinate precision requires.
    fprintf('Searching for common nodes between STAR (English, %d nodes) and Mecway (English, %d nodes)...\n', size(DD_for_comparison,1),
size(VV_for_comparison,1));
    % Pre-allocate for efficiency
    E_temp = zeros(min(size(DD_for_comparison,1), size(VV_for_comparison,1)), 5); % Max possible matches
    match_count = 0;
    % Find matching coordinates
    for i = 1:size(DD_for_comparison,1) % Iterate through STAR data (English)
       star_coords = DD_for_comparison(i, 2:4); % x, y, z coordinates from STAR (English)
       for j = 1:size(VV_for_comparison,1)
          mecway_coords = VV_for_comparison(j, 4:6); % x, y, z coordinates from Mecway (English, transformed)
          % Check if coordinates match within tolerance
          if all(abs(star_coords - mecway_coords) < coord_tol)
             match_count = match_count + 1;
             % Store: [Original STAR node ID, Original Mecway node ID, Common X, Common Y, Common Z]
             % Output E will now be in English units.
             E_temp(match_count, :) = [DD(i,1), VV(j,3), star_coords];
             break; % Found match, move to next STAR node
          end
       end
    end
    % Trim E_temp to actual matches
    E = E_temp(1:match_count, :);
    % Sort results by STAR node ID for consistency
    if ~isempty(E)
       E = sortrows(E, 1);
       fprintf('Found %d common nodes between STAR and Mecway datasets.\n', size(E,1));
```

```
    else
        fprintf('Warning: No common nodes found between STAR and Mecway datasets.\n');
    end
    % Optional: Display statistics
    if size(E,1) > 0
        fprintf('Common nodes (STAR IDs) range: %d to %d.\n', min(E(:,1)), max(E(:,1)));
    end
end
```

Algorithm A5 GNU Octave open source subroutine hist3d.m

```
function hist3d(var);
figure
bin_values=var; %some random data
bin_edges_x=[0:size(bin_values,2)];
x=kron(bin_edges_x,ones(1,5));
x=x(4:end-2);
bin_edges_y=[0:size(bin_values,1)];
y=kron(bin_edges_y,ones(1,5));
y=y(4:end-2);
mask_z=[0,0,0,0,0;0,1,1,0,0;0,1,1,0,0;0,0,0,0,0;0,0,0,0,0];
mask_c=ones(5);
z=kron(bin_values,mask_z);
c=kron(bin_values,mask_c);
surf(x,y,z,c)
end
```