NASA/TM-20240005092

FRAAME Version 1.0 User Manual

Austin J. Schemmel
Glenn Research Center, Cleveland, Ohio

July 2024

NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION.
  Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM.
  Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT.
  Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONTRACTOR REPORT.
  Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION.
  Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION.
  Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION.
  English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

NASA/TM-20240005092

FRAAME Version 1.0 User Manual

Austin J. Schemmel
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

July 2024

Acknowledgments

This work was completed with the support of the Advanced Air Transport Technology Project and the Transformational Tools and Technologies Project. The author greatly appreciates the support of both projects, as well as that of Rula Coroneos for testing the code and providing valuable feedback.

This work was sponsored by the Advanced Air Vehicles Program at the NASA Glenn Research Center.

This work was sponsored by the Transformative Aeronautics Concepts Program.

Level of Review: This material has been technically reviewed by technical management.

This report is available in electronic form at https://www.sti.nasa.gov/ and https://ntrs.nasa.gov/

NASA STI Program/Mail Stop 050
NASA Langley Research Center
Hampton, VA 23681-2199

Contents

FRAAME Version 1.0 User Manual

Austin J. Schemmel
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Summary

The code for Forced Response Aeromechanics Analysis in a MATLAB®-based (The MathWorks, Inc.) Environment (FRAAME Version 1.0) was developed for internal use in aeromechanics efforts undertaken at the NASA Glenn Research Center for computing turbomachine component forced response and Goodman diagrams via modal summation method. The main working script (`FRAAMEv1.m`) allows users to input case-specific manual inputs while the triple-nested loop invokes functions to compute forced response per blade, per nodal diameter, and per mode. Secondly, forced response values are applied to modal stresses to compute complex Von Mises stress values and generate Goodman diagrams per blade, per nodal diameter, and per mode using a linear modal summation method. Currently, this code functions in the Windows® (Microsoft Corporation) operating system using MATLAB® Version R2023a, but it can be adapted for use in the Linux® (Linus Torvalds) operating system by changing the appropriate file path structure in the main script, as well as functions that call external results files.

Introduction

FRAAME code accepts external information, including unsteady blade surface pressure time history information from NASA Glenn Research Center's computational fluid dynamics (CFD) code, TURBO (Refs. 1 and 2), as well as modal structural solutions from ANSYS, a commercial finite element analysis (FEA) solver. Version 1.0 of FRAAME code is intended to be used for validation against simulation results found in References 3 and 4 applied to the R24 fan geometry from the Boundary Layer Ingesting Inlet/Distortion-Tolerant Fan (BLI2DTF) project. FRAAME requires TURBO and Ansys solver output files as input. The code can compute forced response for arbitrary geometries given appropriate TURBO/Ansys solutions are provided as input to FRAAME. NASA TM-20240000075 (Ref. 5) describes code development efforts and validation of forced response and high cycle fatigue analyses applied to the R24 fan geometry. This report is intended to be used by new users of FRAAME Version 1.0 to recreate analysis results from Reference 5 or to compute forced response and high cycle fatigue of differing geometry provided appropriate external inputs.

Required Inputs From External Sources

Several external inputs, in no particular order, are required for this tool to function properly. The minimum required inputs for a simple case (e.g., single blade unsteady pressure, single mode shape, single nodal diameter) are as follows:

`prhist.#.hs_stream`—Unsteady blade surface pressure time history generated by TURBO that has been converted from default unformatted binary to binary stream format. The stream format removes header information from the unformatted binary output and the # represents the blade number of interest.

`Ansys nodal coordinate list`—List of all nodes and corresponding Cartesian coordinates in the FEA model generated by ANSYS.

`Ansys modal displacements`—List of Real Value blade surface nodes and corresponding modal displacements for cyclic symmetry or single blade sector solutions generated by ANSYS. This file should contain phase angle and natural frequency information.

`Ansys modal stresses`–List of Real Value blade surface nodes and corresponding modal stresses for cyclic symmetry or single blade sector solutions generated by ANSYS. This file should contain phase angle and natural frequency information.

`Ansys static only mean stresses`—List of Real Value blade surface nodes and corresponding static stresses for single blade sector solutions generated by ANSYS.

The default scheme for file management includes resolving all relevant files into one directory, then using the working directory pointer in the main script based on the user's file system path (e.g., `C:\Users\username\...`). The functions associated with the main script also contain file system naming conventions that can be modified. The naming scheme for the R24 test case can be found in the section titled "Instructions for R24 Analysis."

Case-Specific Manual Inputs

User-specified inputs listed in order are required for this tool to function properly. These inputs with information on expected values are as follows:

`specnatfreq = #`—True/False input to specify natural frequency in `arn` functions.

1: Natural frequency is specified as a manual input and not inferred from ANSYS modal input.

0: Natural frequency is not specified as a manual input and is inferred from ANSYS modal input.

Use: Certain conditions that may lead to resonant responses (i.e., when the forcing frequency approaches the natural vibration frequency at each mode) can be set with this input such that the calculated forced response value corresponds to the peak resonance. This is considered a worst-case approach to account for the largest dynamic response at mode resonance.

`wn = #`—Real Value input to specify natural frequency in Hz (partner to `specnatfreq`).

Real Value excluding 0: blade natural frequency from ANSYS modal input can be overridden and specified here.

Blank: If `specnatfreq = 0`, this value can be set to an arbitrary Real Value, zero, or commented out.

`ND = [# # … #]`—Zero and positive integer value for nodal diameter specification (case dependent) separated by spaces. For example, `ND = [1 2 3 4]` provides the main loop with the first four nodal diameters to analyze.

Use: Nodal diameters, which are functions of blade count symmetry, provide the main computation loop with indices such that forced response is computed for each nodal diameter requested. Note: Nodal diameters can be defined from index 0 to the total number of blades divided by 2.

`mode = [# # … #]`—Positive integer value for mode specification (case dependent) separated by spaces.

Use: Similar to nodal diameter, modes are defined as positive integers from 1 to an arbitrary value, depending on the availability of information. This input provides the main computation loop with indices such that forced response is computed for each mode requested.

`blade = [# # … #]`—Positive integer value for blade specification (case dependent) separated by spaces.

Use: Much like nodal diameter and mode, blades are defined as positive integers from 1 to an arbitrary value, depending on the availability of information. This input provides the main computation loop with indices such that forced response is computed for each blade requested. Blade information is obtained from TURBO results.

`casepcs = #`—Real Value to specify percent operational speed.

Use: This input is required to specify forcing frequency for the forced response computation (mode dependent).

`workdir = ['…:\Users\...\',num2str(casepcs)',…\']`—String to specify working directory syntax (case percent speed dependent).

Use: This input uses syntax for separating cases depending on operating conditions, where it is assumed that relevant files required for computations are located. It is required to create directories for various operating conditions as percent speeds (e.g., 80pc, 100pc, etc.). For example, `workdir = ['C:\Users\username\Desktop\FRAAMEv1\100pc\case1\']` specifies that the user is working in a Desktop folder named `case1`, which must contain relevant TURBO `prhist`, ANSYS model nodal coordinate `masterlist`, ANSYS modal displacement, ANSYS modal stress, ANSYS static stress, and ANSYS mean stress files. This input uses `casepcs` input for directory definition.

`Ansys_masterlist = 'filename'`—String to specify the file name of the ANSYS model nodal coordinate masterlist.

Use: FEA nodal coordinates are required to extract model information for the interpolation routine in which modal results are mapped onto the CFD mesh for forced response computations. This input allows flexibility with file naming conventions.

`dref = #`—Real Value input to specify blade length scale for dimensionalization of nondimensional TURBO grid.

Note: This information can be found in files `TURBO.#.out` where `#` is the blade of interest in forced response analysis.

`pref = #`—Real Value input to specify sea level reference pressure in psi.

Note: This information can be found in files `TURBO.#.out` where `#` is the blade of interest in forced response analysis.

`gamma = #`—Real Value to specify air-specific heat parameter.

Note: This information can be found in files TURBO.#.out where # is the blade of interest in forced response analysis.

`lref = #`—Real Value to specify unit conversion for dimensionalization of TURBO grid.

Note: This information can be found in files `TURBO.#.out` where `#` is the blade of interest in forced response analysis. Typically, this value is 0.0254 for a meter-to-inch conversion if using EE units.

`numrev = #`—Specific integers to specify TURBO revolution to be analyzed.

0: all revs in the TURBO `prhist` output.

1: first rev in the TURBO `prhist` output.

-1: final rev in the TURBO `prhist` output.

Use: Typically, in unsteady CFD analysis, a solution that shows flow field periodicity between subsequent revolutions is preferred. This input allows users to specify which revolution is of interest to analyze forced response. Results of unsteady CFD analysis typically imply that the final or final few revolutions will show periodic solutions, so `numrev = -1` is typically used.

`tpr = #`—Positive integer value to specify timesteps per revolution used in the unsteady CFD analysis.

Use: This input is required to successfully group information from TURBO `prhist` files. If this input does not match the timesteps per revolution specified in TURBO inputs, grouped data will be incorrect and an error will occur in parsing information for forced response computations.

`hl = 8`—Specify byte count for header length when reading `prhist` binary stream files from TURBO.

Use: TURBO `prhist` files by default are written to binary files from Fortran syntax. Currently, FRAAME accepts binary stream format files, which can either be output directly from TURBO with TURBO input settings or with a separate postprocessing Fortran routine that converts the original binary file into a stream format. Typically, with standard binary files written by TURBO, header and footer information is written and is interpreted as byte count information for the data required for forced response computations. It is assumed that `prhist` files will contain eight integers describing the number of blade rows, grid indices, and timestep information. Following information converted to the stream format will be double precision for simpler import to this code.

`nbd = #`—Positive integer to specify number of blades in the full annulus configuration.

Use: Number of blades considered for analysis is required if ANSYS cyclic symmetry models are included in forced response computations. This value determines a scaling factor that directly influences forced response calculations.

`zeta = #`—Real Value input to specify structural damping.

Use: Structural (mechanical) damping is either determined from separate analysis or assumed as a small value that directly influences forced response calculations.

Note: This parameter is not influenced by aerodamping. It is purely a structural damping parameter.

`rpm = #`—Real Value input to specify revolutions per minute at full operation speed (100 percent).

Use: `rpm` must be known to affect percent speed conditions. If specified as 100 percent operational speed, '`casepcs`' input will tune down rpm to percent speed conditions to isolate specific conditions in which large dynamic responses may occur.

`freqmargin = #`—Real Value between 0 and 1 to specify frequency margin percentage for resonance consideration.

0: No margin, calculated forcing frequency is used exclusively.

0<#<1: Percentage margin is used and forcing frequency is set equal to natural frequency.

1: All frequencies considered resonant, forcing frequency is set equal to natural frequency.

Use: As a forcing frequency approaches a mode's natural frequency, large dynamic resonant responses may occur during turbomachine operation. This input establishes a percentage tolerance that will set the forcing and natural frequencies equal such that the worst-case dynamic loading may occur for a conservative estimate of forced response.

`dispmod = #`—Integer input to specify ANSYS solution technique (single blade sector [unexpanded] or cyclic symmetry [expanded]).

1: Handles nodal diameters from 1 to (nbd/2)-1; expanded ANSYS model (cyclic symmetry).

-1: Handles nodal diameters from 1 to (nbd/2)-1; unexpanded ANSYS model (single blade).

Use: ANSYS solution technique must be known while setting up a case in this code. These techniques are usually either the expanded (cyclic symmetry) model or unexpanded (single blade sector) model. The major difference between the two solution types includes a scaling factor applied to the expanded model but neglected in the unexpanded model. This calculated ANSYS Scaling Factor, called 'asf,' is a function of 'nbd' and 'ND'. Specifically, the functions called are named `arne.m` (ANSYS Reader Normal Expanded) and `arnu.m` (ANSYS Reader Normal Unexpanded).

Future work: Resolve various reader functions into a single function. Include 0ND and (nbd/2)ND for mode shapes only (currently WIP).

`specgageloc = #`—Integer input to specify strain gage information.

Use: Specify strain gage location (Cartesian coordinates) or gage node number (corresponding to structural model). This input is used in the main loop to determine if strain gage Von Mises (VM) stresses are to be calculated as a part of the routine. It invokes the function `gages.m`. Currently, the code allows only one of the two gage information specifications.

0: Strain gage node number is specified.

1: Strain gage location in Cartesian is specified.

-1: Toggles the function off.

Future work: Prior efforts indicate that the search routine for the R24 case is not very robust in searching through nodes in the model when provided gage locations (variation in structural models between simulation and experiment). Specifying gage node number should always have an exact match in the structural model, which leads to a simpler path. Identifying candidate gages with location specified should be investigated further.

`gageloc = [# # #; # # #; … # # #]`—Real Value Cartesian coordinate strain gage general location inputs in groups of three followed by a semicolon. Currently, this specification allows for up to four gages to be specified for analysis.

Use: This input allows users to specify general locations of expected strain gages. If selected as the primary source of gage information, the `gages.m` routine will prioritize searching structural model coordinates in this order: y, z, x. This is done by a trial-and-error process for this R24 case due to variations in models from simulation and experiment. This method is functional but not robust enough to be considered autonomous in this analysis.

Future work: This input invokes `gages.m`, which contains various tolerance values through a range-searching routine. If the tolerance is too wide, many gages will be located as candidates. If the tolerance is too narrow, no gages will be located as candidates. This input may cause issues in properly identifying the node in which a gage belongs. For example, early tests with too wide of a tolerance showed a particular gage was located on the suction side of the blade when, in reality, it was located on the pressure side. This specification is not recommended for new users.

`gagenode = [#; #; … #]`—Real integer value input to specify gage node corresponding to the structural model. Unlike `gageloc`, this input does not require tolerance values as it directly searches through the structural model to find the gage number and location. Currently, this specification can compute VM stresses for any and all node numbers in the model.

Use: This input allows users to specify a strain gage node number that is consistent with the structural model. If selected as the primary source of gage information, the `gages.m` routine will prioritize searching through the model node and coordinate masterlist to compute VM stresses. This method is considered the preferred method if the strain gage node number is known.

`asciichars = cellstr(char(#,#,…#))`—Real integer value input to specify ASCII characters.

Use: This input is used to identify characters in an ASCII file (ANSYS standard output files) that are considered nonimportant for this analysis. ANSYS output files are read into this code as ASCII text files and they will most likely contain information that is not of interest, such as headers. This input allows users to specify common alphanumeric values that should be removed from the text files such that only relevant information is preserved. For example, this code specifies that the asterisk and all capital and lower-case English letters, with the exception of E for scientific notation, are considered irrelevant and should be removed from the ANSYS output files. This input is the primary driver for parsing information after specific strings are extracted from the text files.

`plotting = #`—True/False input for generating plots (blade tip displacement, Goodman diagrams, contours, gages).

0: Plots off.

1: Plots on.

Use: This input is used to toggle plots generated after code completion. Default plots that are generated are as follows:

1. Blade tip displacement [mils]
2. Modal displacement contours
3. Modal stress contours
4. Goodman diagram via modal summation [ksi]
5. Strain gage VM stress amplitude versus number of modes summed [ksi]*.

   *Note: This plot is only generated if `specgageloc` is turned on.

Plots 1 through 3 are embedded in various functions in order to plot results for each blade, each engine order, and each mode requested in the calculation. They can be commented out in the functions or this input can turn all plots off if desired.

`dataout = #`—True/False input for requesting data output to text file.

0: Output off.

1: Output on.

Use: This input is used to request default output as a text file. Default output structure is as follows:

1. Blade number
2. Engine order
3. Mode
4. Forced response amplitude (complex)
5. Blade surface CFD mesh indices
6. Blade surface mesh coordinates interpolated onto CFD mesh [in.]
7. Blade mode shape(s) interpolated onto CFD mesh
8. Blade VM modal stress(es) interpolated onto CFD mesh
9. FEA blade surface nodes
10. Mean stress
11. Calculated VM alternating stress including summation of all mode contributions [ksi]

Note: Specifically, the output follows a block structure that depends on the number of modes in the analysis. Expected output in blocks:

1. Block 1: Mode shape(s). Forced response, CFD mesh indices, CFD mesh coordinates, mode shape.
2. Block 2: Mode stress(es). CFD mesh indices, CFD mesh coordinates, modal stress.
3. Block 3: Goodman information. FEA surface node number, mean stress, alternating stress.

Functions

FRAAME uses functions to perform computations in the background in order to declutter the MATLAB® (The MathWorks, Inc.) workspace. Individual functions allow users to specify inputs and outputs so the intermediate calculations are not necessarily preserved in memory. If intermediate calculations are desired, users must simply request the outputs from the function in the main script. The flowchart in Figure 1 describes the order of operations for the code, including required manual inputs, required inputs from external sources, and expected results. It should be noted at this stage that effectively all functions rely on information from the specified manual inputs defined at the beginning of the script. Descriptions of functions follow the flowchart.

`masterlist.m`—This preloop function imports and parses the `ANSYS nodal coordinate list` required input. The function will import the appropriate ANSYS ASCII text file, then remove unnecessary ASCII text from the file. Built-in MATLAB® commands (`char, str2num`) will attempt to convert the remaining information to a character array, then to a numeric array. If these commands can parse the information successfully, the nodes and corresponding coordinates are written to separate arrays and the code can proceed. If there is a parsing error (e.g., if column widths are not consistent or there is

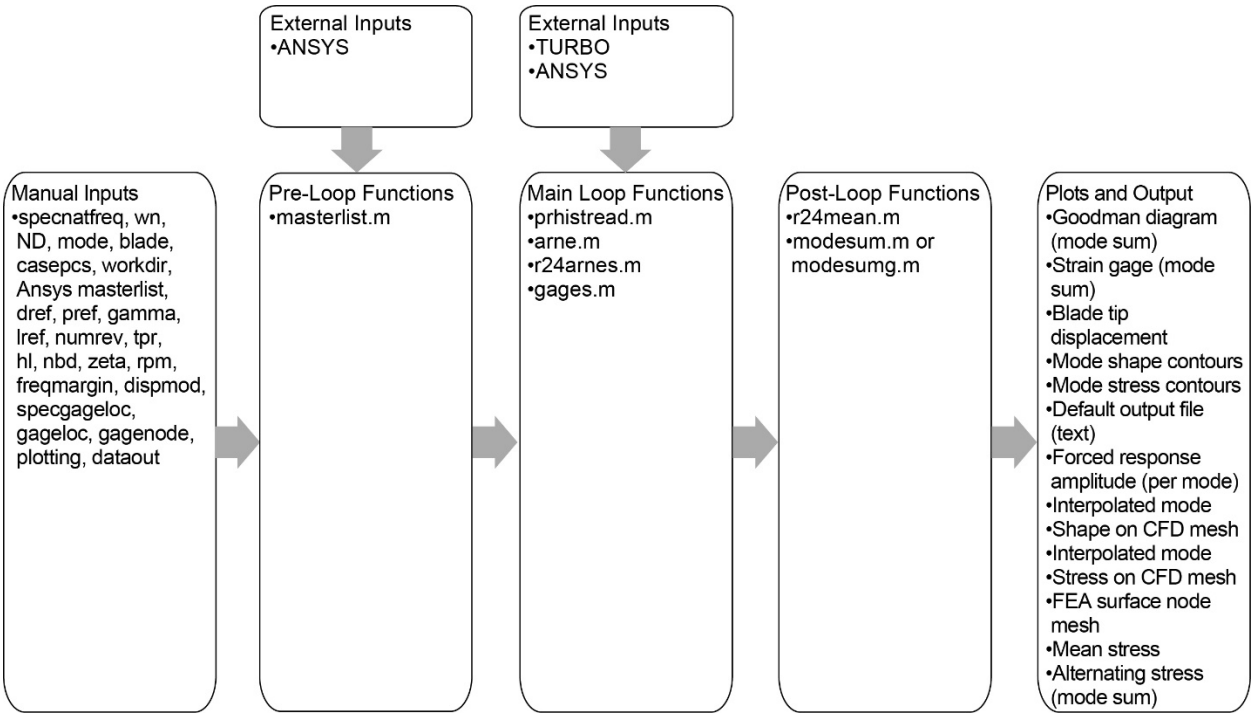

Figure 1.—FRAAME order of operations.

no white space between successive column entries), the information is converted to a matrix array and information is parsed using a fixed-width delimiter approach. This preliminary function exists before the main loop and needs to be used only once in the analysis.

Note: This step is considered one of the first checking points in this analysis. Users should not assume that the fixed-width columns will be consistent between cases. A good way to check this stage is to use a debug run and plot the output coordinates to verify mesh dimensions and spot potentially rogue nodes that might not be parsed properly. Users can either use trial and error to change the fixed-width values or request the first row of the array and count character spaces for input into the fixed-width values.

Note: ANSYS output typically includes node number, three primary coordinates, and three theta components. The theta information is not considered in this analysis.

`prhistread.m`—This main loop function is the first function in the main loop that imports and parses the `prhist.#.hs_stream` required input using a binary import MATLAB® command (`fread`). The function will import the appropriate TURBO-enerated blade surface pressure time history file, which must be a binary stream format. It is assumed that the binary information is written as little-endian and is converted to big-endian byte order for proper file reading. The beginning of this file will include header information, such as grid indices and timestep information, which are 32-bit integer values. The information that follows includes grid nodes and cell areas, which are 64-bit floating point values. The rest of the file information is the bulk of the file information that is structured [ncyct, time, pressure(timesteps)]. These blocks of information are separated as a function of timesteps and number of revolutions in which the data is reshaped in order to remove the ncyct and time information because they are considered unnecessary for computations to proceed. A feature included in this function is the selection of a revolution of interest specified in the `numrev` input in the main script. It is generally a good practice in unsteady CFD to select a numerically converged, periodically stable solution for analysis. A separate TURBO output (not included in this analysis) includes a time history of mass flow rate. Assuming the user has access to this information, one can view the mass flow time history for successive revolutions and determine if the flow field appears stable and if successive revolutions appear periodic. Although this criteria is not explicitly necessary for this forced response framework, it is generally assumed that the last revolution in the unsteady CFD time history will be the best to use for forced response analysis. Currently, this function allows users to specify all revolutions, first revolution, and final revolution for analysis. Once the revolution of interest is selected, the corresponding pressures are extracted and scaled such that computations can proceed.

Note: included at the end of this function includes a checking step that is not considered necessary. The scaled pressure time history is converted via FFT to frequency components if the harmonic pressure force and phase angle are of interest. This step can be used to verify external calculations.

`arn?.m`—These main loop functions are a series of functions separated by `if` statements invoking the `dispmod` manual input. The R24 version of this code has two functions with differing names with descriptions as follow:

`arne`—“ANSYS Reader Normal Expanded”
`arnu`—“ANSYS Reader Normal Unexpanded”

This function is considered the workhorse of this tool; it imports and parses the `ANSYS modal displacements` required input. Initially, this function searches for text strings to determine the mode natural frequency, phase angle, and various strings in order to determine the block size of the file. It is

assumed that this file will contain blocks of information equaling the number of blades in the model that is consistent with cyclic symmetry analysis. After blocking all sectors in the model, unnecessary ASCII text is removed and the blade surface-only nodes and modal displacement values are parsed using a fixed-width delimiter. It is assumed nodes in this file correspond directly to the nodes in the full model node and coordinate list from a `masterlist.m`. A search routine filters the blade surface-only nodal coordinates from the full model list.

The following step uses a built-in MATLAB® Delaunay triangulation interpolation algorithm (`scatteredinterpolant`) using a natural neighbor search method. Interpolants for each displacement's primary Cartesian direction and magnitude are created using the blade surface-only coordinates and modal displacements. These interpolants are applied to the scaled CFD structured mesh at each node. If the cyclic symmetry solution is used, a scaling factor is calculated and applied to the modal displacements. The displacements, still as mode shapes, are interpolated onto CFD mesh nodes, but the CFD pressure information is calculated on cell faces.

The next step includes averaging the modal displacements at four corner nodes to represent a general displacement applied to each cell face in the CFD mesh.

The next step includes calculating the modal force time history, which is a representative forcing on the blade. This modal force is converted to harmonic components using an FFT routine, which results in modal force components in the frequency domain with a corresponding phase. Because these components directly influence the forced response calculation, they are saved in an array that is a function of the three primary indices representing blade, mode, and nodal diameter.

The next step is not considered necessary, but it is used for the R24 case for a validation effort pertaining to a resonant response case. An `if` statement uses the `specnatfreq` input to determine if the natural frequency is specified as an input. If the natural frequency is manually defined, it can be set equal to the forcing frequency, which represents a resonant condition that will yield the most conservative estimate for blade response due to the aerodynamic loading. For off-resonant conditions, the natural frequency is inferred from the modal displacement file and the forcing frequency is calculated as a function of percent speed and mode being analyzed. A following `if` statement uses the `freqmargin` input to determine a percentage tolerance for declaring the iteration as resonant. If the calculated forcing frequency is within a certain percent margin of the natural frequency (10 percent by default), the two frequencies will be set equal to each other to calculate the conservative resonant response.

The next step includes calculating the forced response amplitude as complex values and a magnitude. This scalar parameter represents the scaling factor to be applied to the modal analysis results in order to convert them from modal to physical. In other words, at this stage, the response amplitude attributes the modal analysis with units (ksi by default).

The final steps of this function include plotting tools that generate z-component blade tip deformation in units of mils, as well as mode shape contour plots.

Note: At this stage, it is good practice to verify that the blade surface-only FEA grid matches up very closely with the scaled CFD mesh in terms of orientation. This can be accomplished by overlaying plots of each using MATLAB® plotting commands. It is assumed that the first sector in the FEA model will match up directly with the CFD mesh. If this is not the case, the blocking of information is available and a sector definition within this function can be modified to select the proper sector.

Note: The primary source of information used in this analysis assumed that the exclusive solution technique included cyclic symmetry (expanded). The single difference between cyclic symmetry (expanded) and single blade sector (unexpanded) includes a scaling factor that is a function of nodal diameter and number of blades in the model. Future work includes resolving both of these functions into a single function to simplify the main loop in the code.

r24arnes.m—This main loop function follows a procedure similar to `arne.m`, in which a stress file is imported, keywords are defined, and the file is searched for blocks of information based on keywords. A following checking step compares frequency and phase information from the stress file to the displacement file to verify that the datasets are consistent. Unnecessary text is removed from the imported file and relevant information is parsed using a fixed-width delimiter. This process is completed twice for varying material types in the blade and disk sector. Both datasets are searched via node number to identify duplicate node number entries, which are referred to as "shared nodes." The information at these shared nodes, if they exist, are then averaged in order to continue the computation. Complex forced response results are applied and a VM stress calculation is performed to represent alternating stresses for each mode's contribution to blade vibration. Following this, the modal stresses for the full model are filtered out to include the blade surface nodes only. These surface stresses are interpolated onto the CFD mesh and modal stress contour plots are generated.

Note: The R24 geometry considered for all analyses with this code version is assumed to include two different materials for the blade and disk sector. Although material properties are not considered in this code, it is possible that the junction between the blade and disk sector contains nodes that share the same node number. For the purposes of continuing the computation, these shared nodes are identified and averaged to represent a single stress value at each node in the model.

Gages.m—This main loop function, which is the final function embedded into the main loop, serves as an extra step for code validation against strain gage calculations, but it is not necessary to complete forced response analysis and Goodman diagram generation. The options that exist in this function include which type of strain gage information is available. It is assumed that either the node number that corresponds to the model or the physical strain gage location specified as coordinates are known.

The simpler case includes model node number. If gage node number corresponding to the model is specified, the model is searched directly to extract coordinate and stress information. Complex forced response calculation is applied and VM stress is calculated for each gage location.

The more complicated case includes physical strain gage coordinate locations. If gage locations are specified, a search routine based on small tolerance values (range of 0.001 to 0.15) determines candidate nodes in the full model based on coordinates in this order: *y, x,* and *z*. The tolerance values determine the range in which candidate nodes are identified. Too wide of a tolerance will produce many results and too narrow of a tolerance will produce few to no results, so this specification is not recommended in its current state. The tolerances currently in `gages.m` were obtained through trial and error. The search order of Cartesian components reflects relative dimensions and assumed availability of information. The *y* coordinate corresponds to the spanwise location of a strain gage, which is assumed to possess the largest variation due to blade proportions. The *x* coordinate corresponds to the axial location of a strain gage, which follows as the most likely second largest source of variation. The *z* coordinate is searched last. Once candidate nodes are identified, the full model is searched to extract coordinate and stress information. Complex forced response is applied and VM stress is calculated for each gage location.

Note: Currently, `gages.m` requires exactly four gage coordinate locations to function properly if strain gage coordinate locations are specified. Due to the lack of an intelligent search algorithm (future work), errors will occur if more or fewer gage locations are specified and the computation cannot proceed. If tolerances are not modified and include too many or too few gage location candidates, errors will occur if more or fewer gage locations are specified and the computation cannot proceed. Again, this function is not recommended for new users.

r24mean.m—This postloop function follows a procedure similar to `r24arnes.m,` in which a mean stress file is imported and parsed assuming various material types for the blade and disk. The file is imported, keywords are defined and searched, unnecessary text is removed, and mean stress for each

material type is parsed using an array splitting approach. Shared nodes are identified, values averaged, and VM mean stress is calculated. This function is the first post-main-loop function that only needs to be completed once.

Note: The array splitting approach assumes that successive entries contain white space separation. If an error occurs here due to parsing, fixed-width delimiting is necessary for the computation to proceed.

`modesum.m`—This postloop function imports calculated modal stresses from the main loop and performs a summation of each mode contribution to the alternating stress. The input to this function is output from other functions in the main loop that is tabulated for each blade, engine order, and mode in the analysis. Complex modal stress components are imported, parsed, and summed individually. The resulting magnitude of the complex contributions is calculated. VM stress is then calculated. The results of this function serve to populate Goodman diagram generation as the final stage of this framework.

`modesumg.m`—This postloop function is mostly identical to `modesum.m` except it includes the same procedure for calculating VM modal summation stress contributions for identified strain gages in the model. This function is only invoked if the `specgageloc` is turned on. Otherwise, `modesum.m` is used in the analysis.

Instructions for R24 Analysis

This code will produce forced response calculations and Goodman diagrams given arbitrary inputs with the exception of strain gages if gage locations are input. This code is compatible with Windows OS running MATLAB® R2023a. To run FRAAME using earlier versions of MATLAB®, refer to the "Known Issues" section. This version of the code is intended to be used for validation against the results published in References 3 and 4. Validation efforts are broken down into four cases:

- **Case 1**—85 percent speed, 4EO, mode 2, resonant response. No figure for this case appears in Reference 3, but information for comparison can be found in Section V, page 14.
- **Case 2**—100 percent speed, 1EO, mode 1, Goodman. See Figure 21 in Reference 4 on page 17 for this case.
- **Case 3**—100 percent speed, 1EO, modes 1 to 3, Goodman. For this case, Figure 21 in Reference 4 on page 17 also compares well due to first mode dominated responses.
- **Case 4**—Strain gage prediction. Same conditions as Cases 2 and 3, Reference 4, Figure 22, page 18.

The following steps allow a new user to either perform analyses for the aforementioned test cases with the R24 geometry or set up new cases with arbitrary geometry:

1. Set up a run directory folder if starting a new case and not using the example cases (85pc or 100pc). If using example cases, skip to Step 4. As shown in Figure 2, establish a folder as the working directory (e.g., `C:\Users\...\FRAAMEv1\CodeAndFunctions`) in Windows.

Figure 2.—Working directory folder.

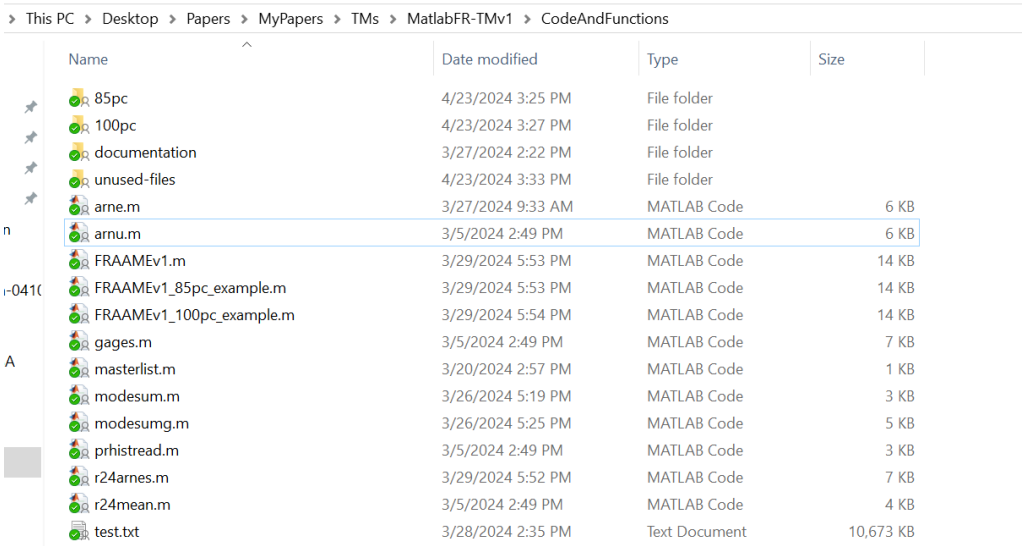

Figure 3.—Script files in the run directory.

2. Verify the following script files exist in the run directory from Step 1 as shown in Figure 3: `FRAAMEv1.m`, `arne.m`, `arnu.m`, `gages.m`, `masterlist.m`, `modesum.m`, `prhistread.m`, `r24arnes.m`, and `r24mean.m`.

   Note: These are the scripts for the base code (`FRAAMEv1.m`) and functions. `FRAAMEv1.m` is the primary script that should be run/modified. All other functions will be called by running `FRAAMEv1.m`. Running the functions individually will not produce forced response analysis calculations. They will most likely produce errors if run on their own.
3. Create folders in the run directory based on run condition, if they do not exist, for example, 85pc, 100pc, etc. The example directory shown in Figure 4 already contains 85pc and 100pc directories with relevant files. If using the example cases (85pc or 100pc), start at Step 5. The naming scheme for directories is specified by the `casepcs` input in `FRAAMEv1.m`. For example, if a 70 percent speed case is desired, create a directory named `70pc`, drop relevant TURBO/ANSYS results files into the created directory, and modify the `casepcs` and `workdir` inputs in `FRAAMEv1.m` to properly identify the path of the newly created directory. All run directories must be named `##pc` where `##` represents percent speed. If this number in the directory name and the `casepcs` input do not match, errors will occur due to failing to identify the directory path.
4. Drop relevant files into each run condition folder or verify the correct files are in the example directories. For the `85pc` example (Figure 5), those files would include the following:
   - `prhist.1.hs_stream`
   - `R24_85pc_static_only_meanstress-component-stress-list`
   - `R24_Feb4_2015_nodal_coordinate_list`
   - `R24-85pt-ND4-M2-Modal-Comp-Stress-List`
   - `R24-85pt-ND4-M2-Modal-Disp-List`

For the `100pc` example (Figure 6), those files should include the following:

- `prhist.1.hs_stream`
- `R24_100pc_static_only_meanstress-component-stress-list`
- `R24_Feb4_2015_nodal_coordinate_list`
- `R24-100pt-ND1-M1-Modal-Comp-Stress-List`
- `R24-100pt-ND1-M2-Modal-Comp-Stress-List`
- `R24-100pt-ND1-M3-Modal-Comp-Stress-List`
- `R24-100pt-ND1-M1-Modal-Disp-List`
- `R24-100pt-ND1-M2-Modal-Disp-List`
- `R24-100pt-ND1-M3-Modal-Disp-List`
- `Rotor_R24_Feb4_2015-static-only-MAT1516-Component-Stress-List`

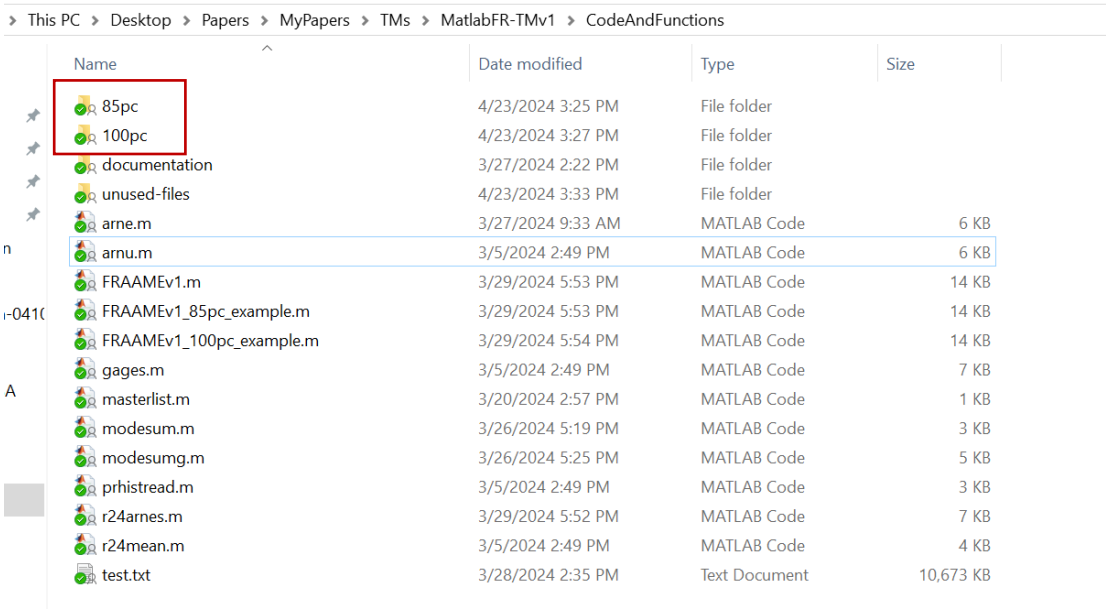

Figure 4.—Example directory with 85pc and 100pc directories.

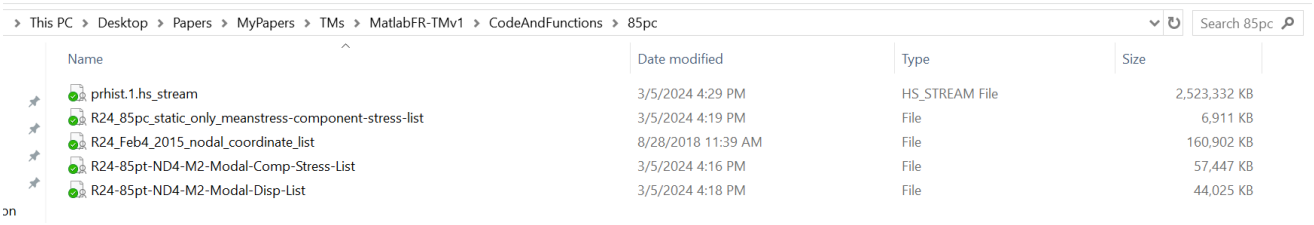

Figure 5.—Files in 85pc folder in example directory.

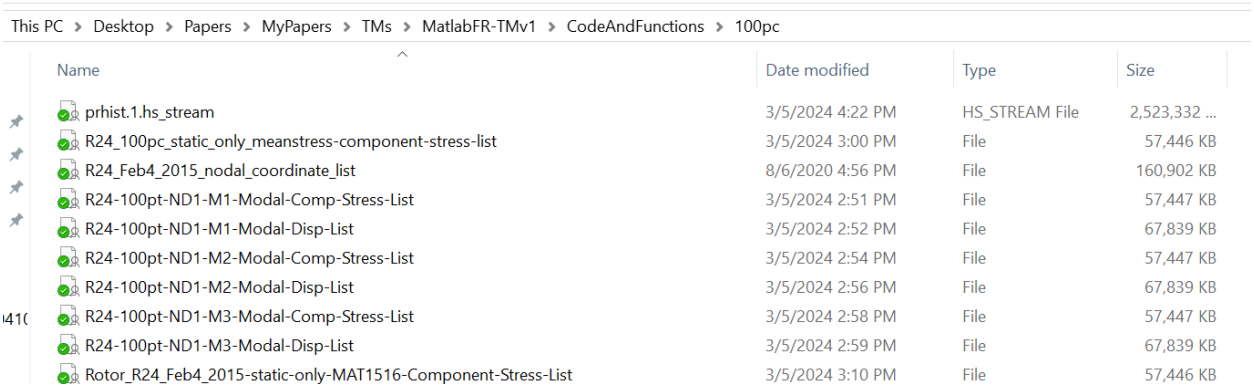

Figure 6.—Files in 100pc folder in example directory.

5. Open `FRAAMEv1.m` in the MATLAB® environment (Version R2023a preferred).

   NOTE: `FRAAMEv1.m` exists as the main script to be run/modified (Figure 7). If there are directories established in previous steps, the inputs at the beginning of `FRAAMEv1.m` can be modified to point to the proper directories. `FRAAMEv1.m` is the general script that can be modified to run other cases (varying percent speed, modes, engine orders/nodal diameters/blades). The nearly identical scripts `FRAAMEv1-85pc-example.m` and `FRAAMEv1-100pc-example.m` should contain proper inputs for each example and can be run instead of `FRAAMEv1.m`.

   NOTE: Strain gage information has been randomized in all `FRAAMEv1.m` versions in the example directory due to export-controlled information. These versions of the code have the strain gage calculations toggled off by default. If this information is desired, refer to "Strain-Gage-Information-EXPORT.docx" in the "documentation" directory for gage locations/node numbers. See the `specgageloc` input information in the section titled "Case-Specific Manual Inputs" to toggle on strain gage calculations.
6. As shown in Figure 8, modify `workdir` input in `FRAAMEv1.m` to reflect the run directory path from Step 1.

Verify manual inputs at the beginning of the script for each case.

- For the `85pc` example:
  - `specnatfreq = 1;`
  - `wn = 679.5;`
  - `ND = [4];`
  - `mode = [2];`
  - `blade = [1];`
  - `casepcs = 85;`
  - `zeta = 0.01;`
- For the `100pc` example:
  - `specnatfreq = 0;`
  - `ND = [1];`
  - `mode = [1 2 3];`
  - `blade = [1];`
  - `casepcs = 100;`
  - `zeta = 0.000003;`

7. Run code using the green arrow in the Editor tab (Figure 9).

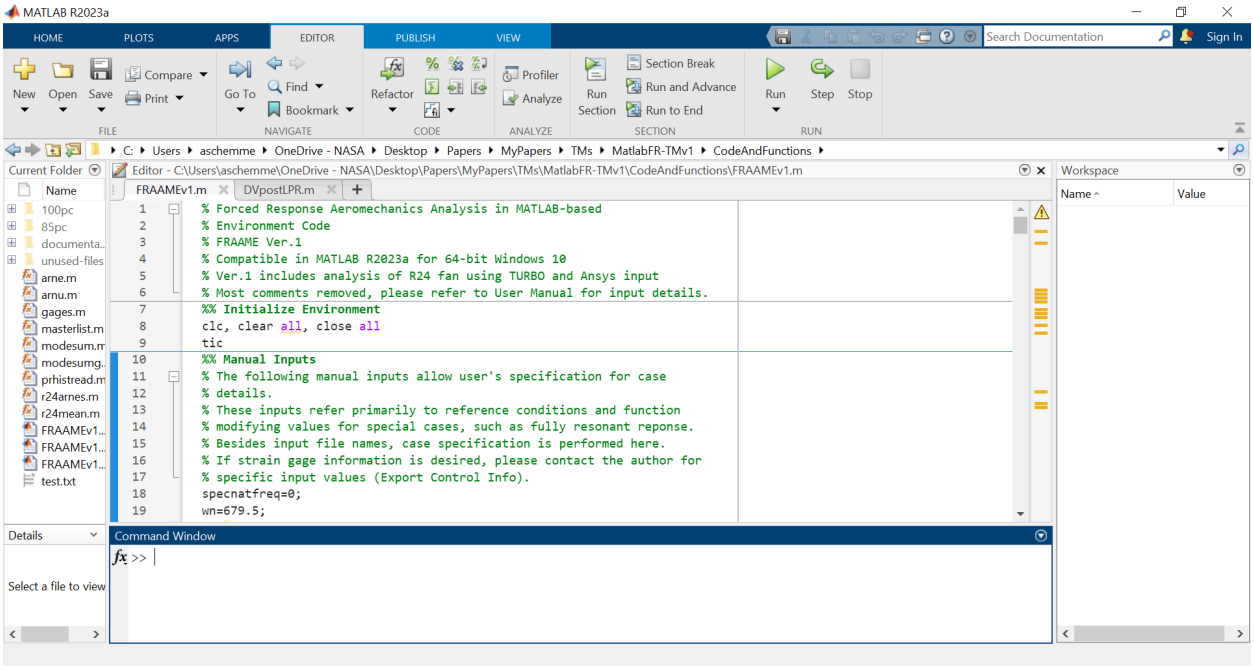

Figure 7.—Example MATLAB® user interface with `FRAAMEv1.m` script to be modified..

```
23    casepcs=100;
24    workdir=['C:\Users\aschemme\OneDrive - NASA\Desktop\Papers\MyPapers\'...
25        'TMs\MatlabFR-TMv1\CodeAndFunctions\',num2str(casepcs),'pc\'];
26    % cd(workdir)
```

Figure 8.—Modifying `workdir` input.

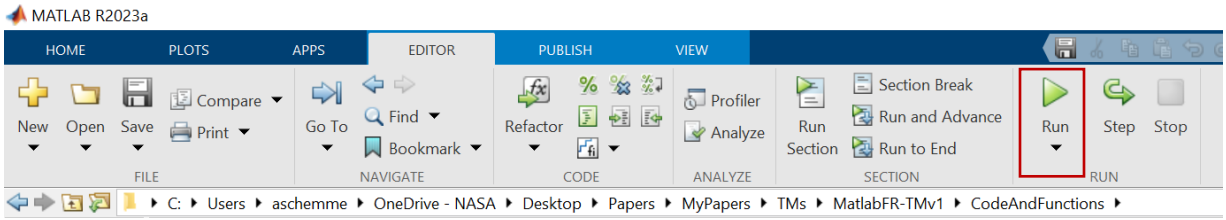

Figure 9.—Green run arrow.

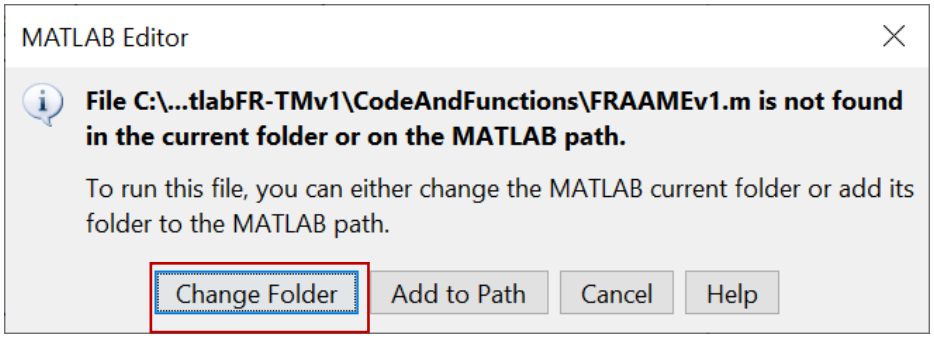

Figure 10.—MATLAB® Editor popup.

If a popup appears, select the “Change Folder” option instead of “Add to Path” (Figure 10).

Calculated variables can be found in the workspace (Figure 11).
Run status can be found in the command window (Figure 12).

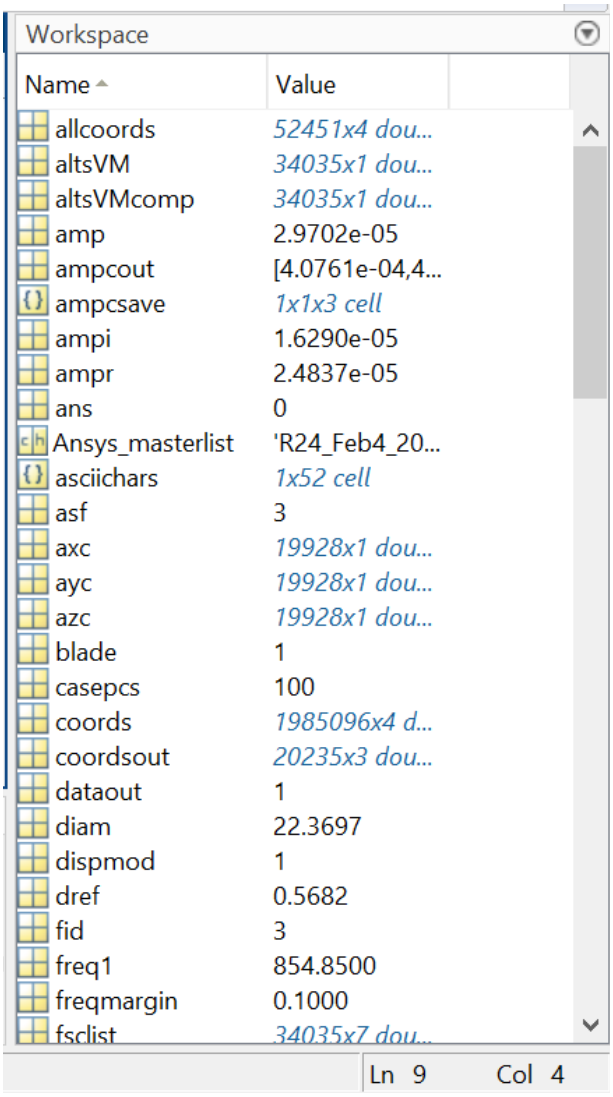

Figure 11.—MATLAB® workspace with FRAAME variables.

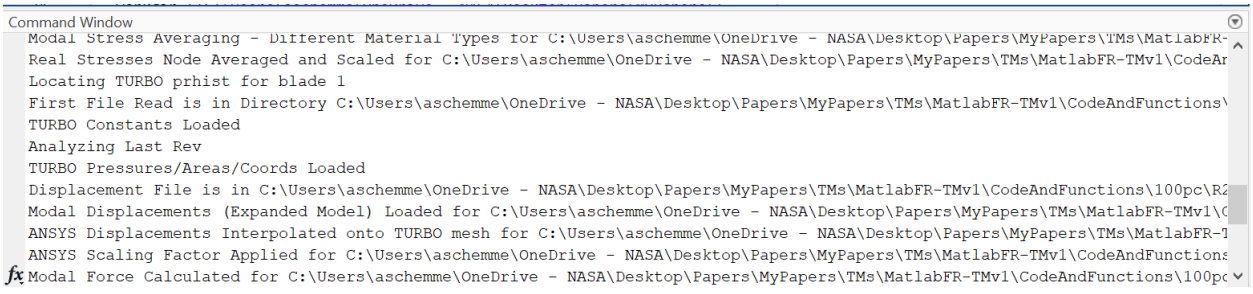

Figure 12.—MATLAB® command window with run status notifications.

Figure 13 through Figure 17 illustrate the plotting options available.

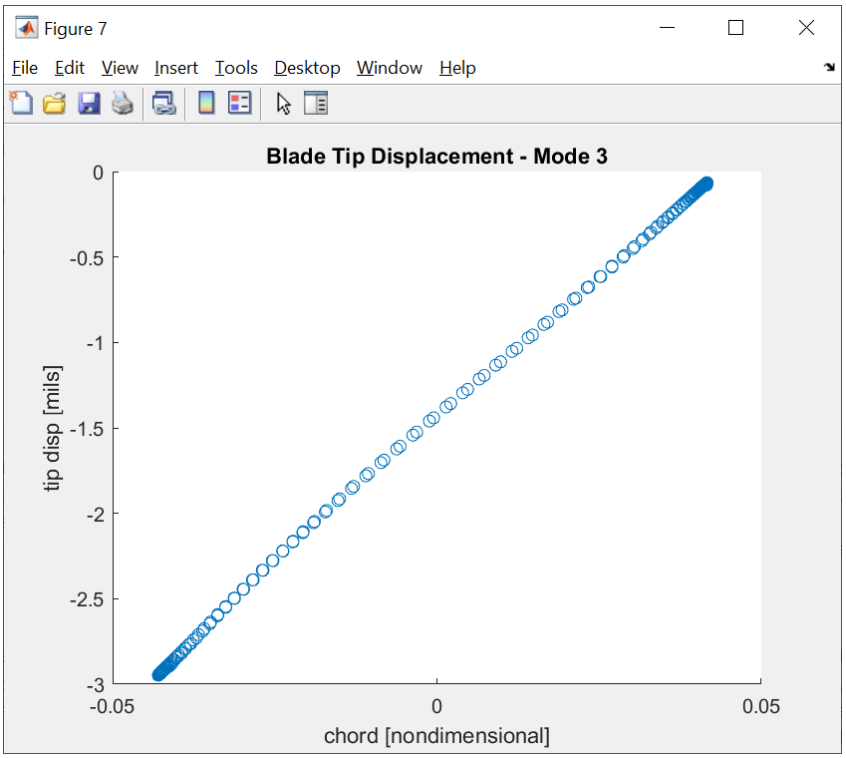

Figure 13.—Blade tip displacement, mils.

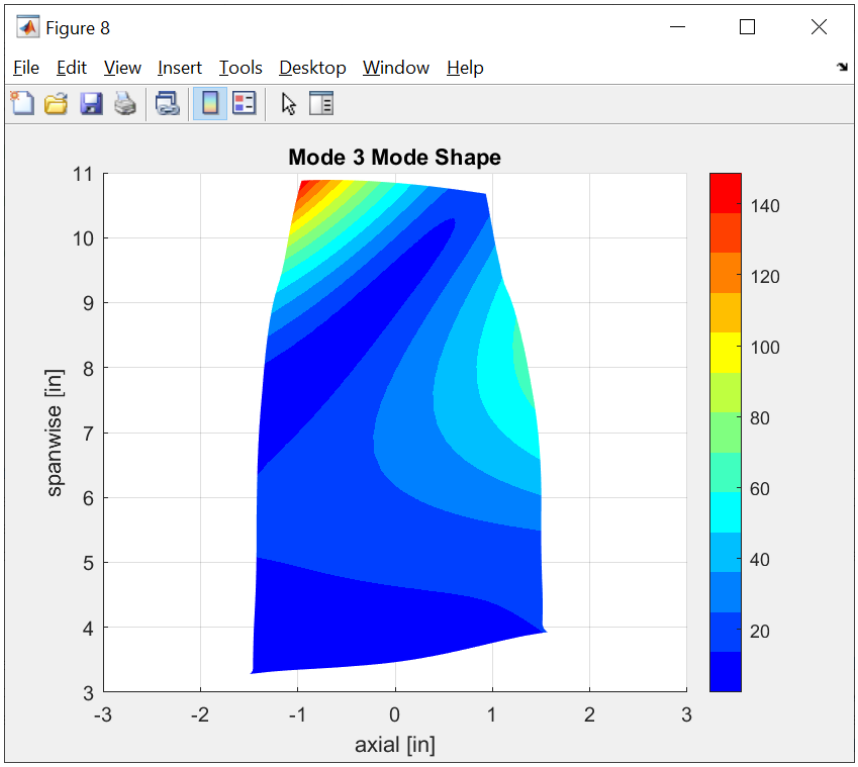

Figure 14.—Modal displacement contour plot.

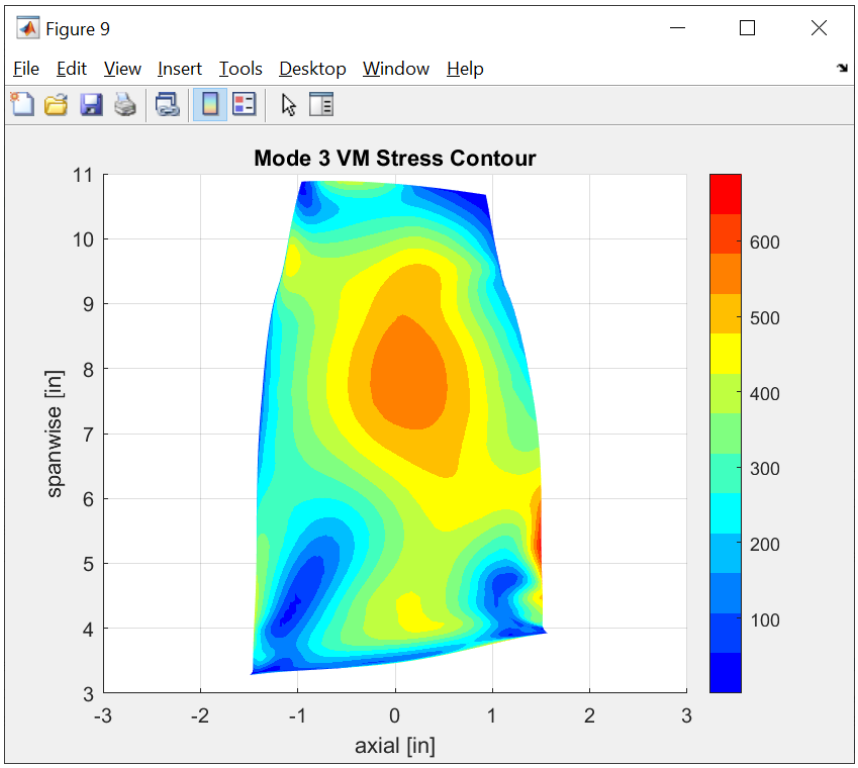

Figure 15.—Modal stress contours.

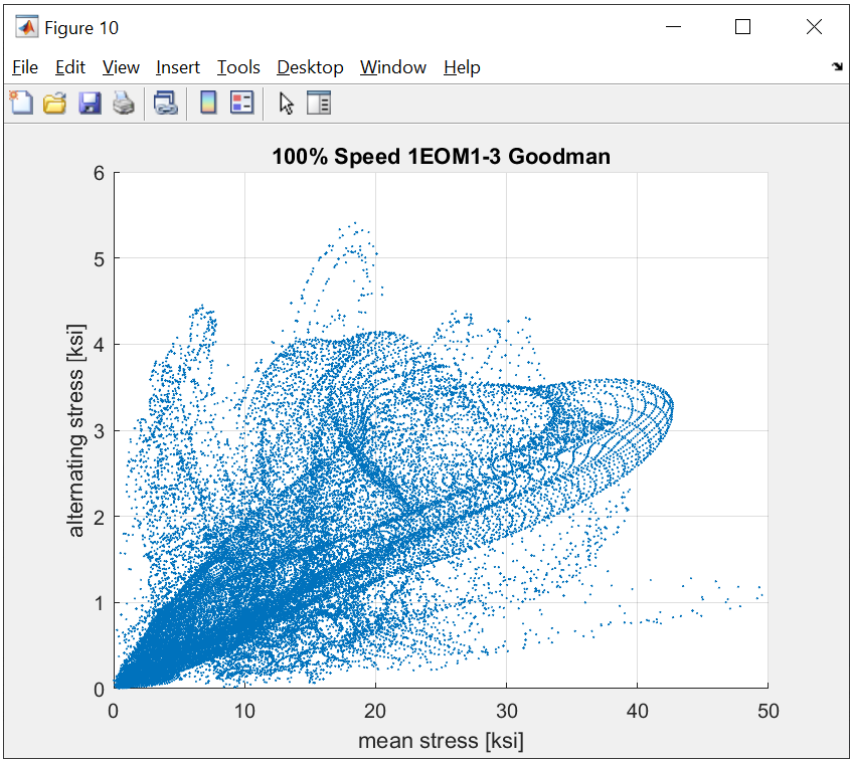

Figure 16.—Goodman diagram (modal summation).

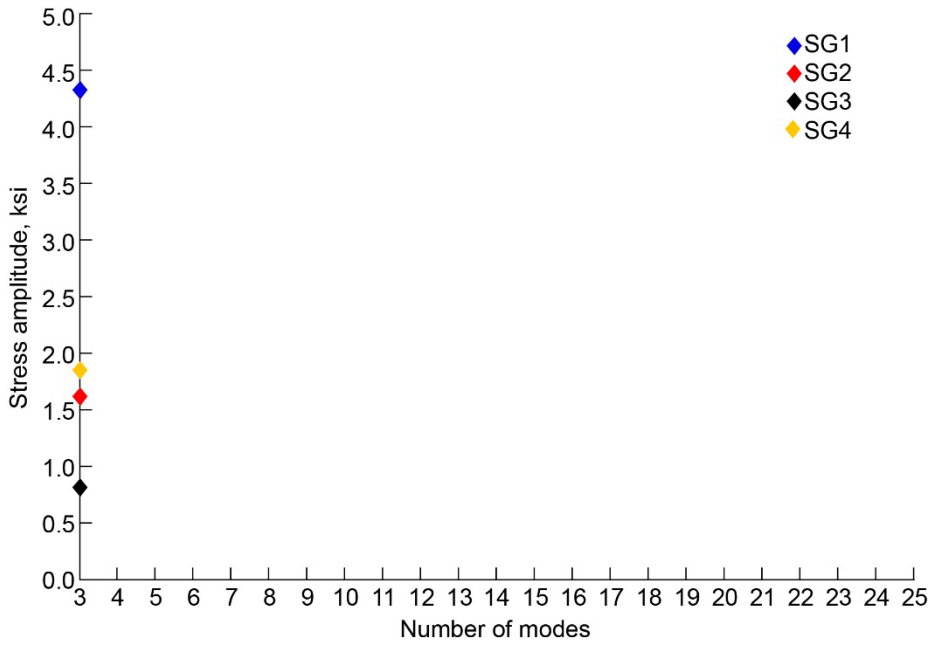

Figure 17.—Strain gage VM summation by group modes. Stress amplitude versus number of modes summed. Toggled off by default for examples.

Code Default Output

The output options for the code are at the user's discretion. A user can add or remove lines of code based on the desired output from this analysis. Currently, the default output of the code reflects the *plotting* and *dataout* variables. Various plotting commands are embedded in functions contained in the triple-nested loop for forced response calculations in order to dynamically plot results for various blades, engine orders, and modes in the analysis. Default plots and command locations are as follows:

- `arne.m`:
  - Blade tip displacements in units of mils along nondimensional chord coordinates.
  - Axial versus spanwise view of mode shape contours interpolated onto the CFD mesh in units of inches.
- `r24arnes.m`:
  - Axial versus spanwise view of modal stress VMcontours interpolated onto the CFD mesh in units of inches.
- `FRAAMEv1.m`:
  - Alternating stress versus mean stress Goodman plot, including summation of all mode contributions in units of ksi.
  - VM stress calculations at specified strain gage locations, including summation of all mode contributions in units of ksi.**

    **Note: Output only available if strain gage information is requested via the '`specgageloc`' variable.

Bulk data output for computations performed by this code is found at the end of `FRAAMEv1.m`. Results are written to a text file called `test.txt` with comma delimiters, although the name of the output file can be changed. The general format includes printouts of forced response calculations at each

mode, mode shape with mesh information for each mode, modal stress with mesh information for each mode, and Goodman diagram information with mesh information. The generated text file will be written to the main working directory. Successive runs of the code will overwrite the generated text file if it is not moved to a different folder/directory. Expected output is as follows:

1. Block 1: mode shape(s)
   – Header information [blade number (code input), engine order (code input), mode (code input)]
   – Forced response amplitude (complex)
   – Mode shape(s): [Blade surface CFD mesh indices, Blade surface mesh coordinates interpolated onto CFD mesh (in.), Blade mode shape(s) interpolated onto CFD mesh (three Cartesian components)]
2. Block 2: mode stress(es)
   – Mode stress: [Blade surface CFD mesh indices, Blade surface mesh coordinates interpolated onto CFD mesh (in.), Blade VM modal stress(es) interpolated onto CFD mesh].
3. Block 3: Goodman information
   – Goodman: [FEA blade surface node number, Mean stress, Calculated VM alternating stress including summation of all mode contributions (ksi.).]

Expected output is dependent on blade, engine order, and mode input. For the 100pc example, expected output includes the following associated with the aforementioned block structure:

– Block 1: 1 blade, 1 engine order, 3 modes: 3 sets of header information, 3 forced response amplitudes, 3 sets of mode shape data.
– Block 2: 1 blade, 1 engine order, 3 modes: 3 sets of mode stress data.
– Block 3: FRAAME produces Goodman diagram information using a modal summation method, therefore this block will always include a single block of data according to the FEA surface nodes.

For the 85pc example, expected output includes the following associated with aforementioned block structure:

– Block 1: 1 blade, 1 engine order, 1 mode: 1 line of header information, 1 forced response amplitude, 1 mode shape.
– Block 2: 1 blade, 1 engine order, 1 mode: 1 mode stress.
– Block 3: 1 set of Goodman information.

Known Issues

The code will produce forced response results for arbitrary inputs. One current issue includes the selection of strain gages for analysis. Two options are available for calculating VM stresses at strain gage locations in which the gage location or gage node number is specified.

1. The search routine for the gage location option requires trial and error to select candidate gages using a tolerance input in `gages.m`. This is not recommended because there may not be a candidate node in the search range based on location. This will produce an error. If gage locations are requested, there must be exactly four gage Cartesian coordinates (12 values) for the

calculations to proceed. The search routine expects exactly four gages for the current version of this code.

2. If the gage node number is specified, it must exist as a surface node or an error will be encountered. The gage locations and node numbers are considered export control, so they are provided externally to the code. Users may enter these values to reproduce the figure for comparison with Reference [2], Figure 22. Otherwise, the gage routine can be toggled off or arbitrary information can be input for the code to complete its job.

For versions of MATLAB® earlier than 2023a, another known issue exists dealing with data import in the `masterlist.m` function. It appears that the '`importdata`' MATLAB® command was changed in an earlier update. Previously, the imported array `allcoords` was converted into a character array before parsing. MATLAB® Version 2023a does not require this step, so it was removed from `masterlist.m`. If running an earlier version of MATLAB®, an error might be produced that looks similar to the error shown in Figure 18.

To fix this error for an earlier version of MATLAB®, input the following line of code after line 8 in `masterlist.m`:

```
allcoords=char(allcoords);
```

Otherwise, update MATLAB® to Version 2023a or later and do not add this extra line of code.

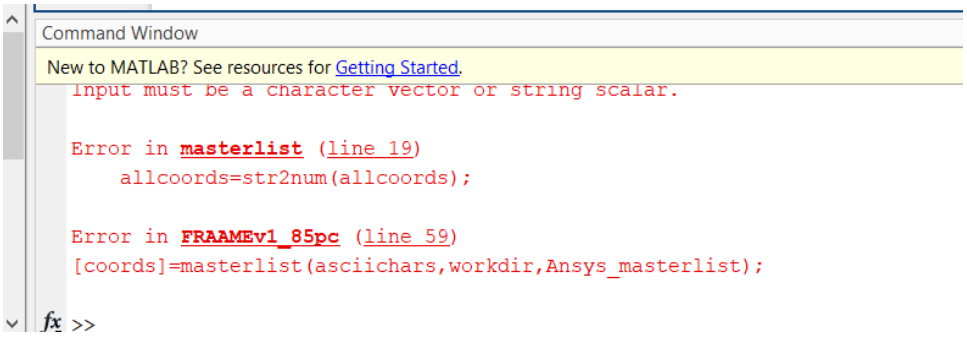

Figure 18.—MATLAB® error message.

Appendix

Source code for **FRAAMEv1.m**:

```
% Forced Response Aeromechanics Analysis in MATLAB-based
% Environment Code
% FRAAME Ver.1
% Compatible in MATLAB R2023a for 64-bit Windows 10
% Ver.1 includes analysis of R24 fan using TURBO and Ansys input
% Most comments removed, please refer to User Manual for input details.
%% Initialize Environment
clc, clear all, close all
tic
   %% Manual Inputs
   % The following manual inputs allow user's specification for case
   % details.
   % These inputs refer primarily to reference conditions and function
   % modifying values for special cases, such as fully resonant response.
   % Besides input file names, case specification is performed here.
   % If strain gage information is desired, please contact the author for
   % specific input values (Export Control Info).
   specnatfreq=0;
   wn=679.5;
   ND=[1];
   mode=[1 2 3];
   blade=[1];
   casepcs=100;
   workdir=['C:\Users\aschemme\OneDrive -
   NASA\Desktop\Papers\MyPapers\'...
       'TMs\MatlabFR-TMv1\CodeAndFunctions\',num2str(casepcs),'pc\'];
   % cd(workdir)
   Ansys_masterlist='R24_Feb4_2015_nodal_coordinate_list';
   dref=0.568190802764893;
   pref=14.69596336;
   gamma=1.40129;
   lref=0.0254;
   numrev=-1;
   tpr=1800;
   diam=dref/lref;
   hl=8;
   nbd=18;
   % zeta=0.01167;     %experimental
   % zeta=0.01;        %85pc
   zeta=0.000003;      %100pc
   rpm=11864;
   speed=casepcs/100;
   pcs=rpm*speed;
   freqmargin=0.1;
   dispmod=1;
   % Refer to "Strain-Gage-Information-EXPORT.docx" under "documentation"
```

```
46.   % directory if strain gage caluclations are desired
47.   specgageloc=-1; %specgageloc=1 NOT RECOMMENDED
48.   gageloc=[-2.1591 4.5555 0.9878; ...
49.       0.646 1.435 1.8821; ...
50.       1.1101 7.7777 0.7611; ...
51.       1.8813 11.578 0.0012]; %random locations
52.   gagenode=[26755; 49550; 68761; 82663]; %random nodes
53.   asciichars=cellstr(char(42,65,66,67,68,70,71,72,73,74,75,...
54.       76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,97,98,...
55.       99,100,101,102,103,104,105,106,107,108,109,110,111,...
56.       112,113,114,115,116,117,118,119,120,121,122))';
57.   plotting=1;
58.   dataout=1;
59.   disp('Manual Constants Loaded');
60.   %% FEA Full Model Import
61.   % This function imports the Ansys FEA full model list of nodes and
62.   % Cartesian coordinates, removes unnecessary text, and returns a sorted
63.   % list of all nodal information for the full model.
64.   [coords]=masterlist(asciichars,workdir,Ansys_masterlist);
65.   %% Triple Nested Forced Response Loop
66.   %
67.   for p=blade(1):blade(end)
68.       for i=ND(1):ND(end)
69.           for j=mode(1):mode(end)
70.               % call prhistread.m for TURBO information
71.               [iek,jek,ij,ijm,n,xc,yc,zc,axc,ayc,azc,xt,yt,...
72.                   zt,pressure,prhist]=prhistread(numrev,...
73.                   tpr,diam,gamma,pref,hl,workdir);
74.               if dispmod==0 %0ND (TCT mode shape)
75.                   [x,y,z,mshape{p,i,j},usumall{p,i,j}]=arze(p,i,j,...
76.                       asciichars,coords,xt,yt,zt,xc,yc,zc,diam,nbd);
77.               elseif dispmod==1
78.                   % call arne.m for FR calculation
79.
   [x,y,z,mfmag{p,i,j},amp,mshape{p,i,j},usumall{p,i,j},...
80.                       tipdisp,wn,w,ampr,ampi,freq1,phase1,asf,...
81.                       allcoords,usumsave{p,i,j},...
82.                       ampcsave{p,i,j}]=arne(p,i,j,asciichars,coords,...
83.                       xt,yt,zt,xc,yc,zc,diam,nbd,ij,ijm,iek,jek,...
84.                       pressure,axc,ayc,azc,n,pcs,prhist,zeta,...
85.
   freqmargin,casepcs,workdir,wn,specnatfreq,plotting);
86.                   % call r24arnes.m for Goodman calculation
87.                   [altsVM,fsclist,altsVMcomp,...
88.                       stresscontoursreal{p,i,j},...
89.                       stresscontoursimag{p,i,j},stressrealout{p,i,j},...
90.                       stressimagout{p,i,j},...
91.
   smagsave{p,i,j},matstressallu]=r24arnes(p,i,j,freq1,...
92.                       phase1,asciichars,amp,ampr,ampi,asf,workdir,...
93.                       casepcs,coords,xt,yt,zt,diam,plotting);
94.                   % call gages.m for strain gage calculations
```

```
                    % (if necessary)
                            if specgageloc==-1
                                continue
                            else

   [IDnodes,SGinfo{p,i,j}]=gages(specgageloc...
                                    ,gageloc,gagenode,matstressallu,...
                                    allcoords,ampr,ampi,asf,mode,p,i,j);
                            end
                elseif dispmod==2 % (TCT 8ND only for mode shape)
                     [x,y,z,mshape{p,i,j},usumall{p,i,j}]=arle(p,i,j,...
                         asciichars,coords,xt,yt,zt,xc,yc,zc,diam,nbd);
                elseif dispmod==-1
                    % call arnu.m for FR calculation
                     [x,y,z,mfmag{p,i,j},amp{p,i,j},mshape{p,i,j},...
                         tipdisp,wn,w,ampr,ampi,freq1,...
                         phase1,allcoords,usumsave{p,i,j},...
                         ampcsave{p,i,j}]=arnu(p,i,j,asciichars,coords,...
                         xt,yt,zt,xc,yc,zc,diam,nbd,ij,ijm,iek,jek, ...
                         pressure,axc,ayc,azc,n,pcs,prhist,zeta, ...
                         freqmargin,casepcs,workdir,wn,specnatfreq);
                     % call r24arnes.m for Goodman calculation
                     [altsVM,fsclist,altsVMcomp,...
                         stresscontoursreal{p,i,j},...
                         stresscontoursimag{p,i,j},stressrealout{p,i,j},...
                         stressimagout{p,i,j},...
                         smagsave{p,i,j},matstressallu]=r24arnes(p,...
                         i,j,freq1,phase1,asciichars,amp,ampr,ampi,...
                         asf,workdir,casepcs,coords,xt,yt,zt,diam,plotting);
                     % call gages.m for strain gage calculations
                     % (if necessary)
                            if specgageloc==-1
                                continue
                            else

   [IDnodes,SGinfo{p,i,j}]=gages(specgageloc...
                                    ,gageloc,gagenode,matstressallu,...
                                    allcoords,ampr,ampi,asf,mode,p,i,j);
                            end
                 end
             end
         end
  end
  %% Mean Stress Calculation
  % call r24mean.m to calculate mean stresses
  [meanstress]=r24mean(asciichars,...
      workdir,casepcs,fsclist);
  %% Modal Summation (Goodman)
  % call modesum.m to calculate Goodman diagram information by summing
  % selected modal contributions to alternating stress.
  % call modesumg.m if gages are specified
  if specgageloc==-1
```

```
145.      if size(mode,2)>1
146.          [msumVM]=modesum(p,i,j,stressrealout,...
147.              stressimagout,blade,ND,mode);
148.      else
149.          msumVM=altsVMcomp;
150.      end
151.  else
152.      if size(mode,2)>1
153.          [msumVM,msumVMgages]=modesumg(p,i,j,...
154.          stressrealout,stressimagout,SGinfo,blade,ND,mode);
155.      else
156.          msumVM=altsVMcomp;
157.          sxxgage=sqrt(IDnodes(:,5).^2+IDnodes(:,12).^2);
158.          syygage=sqrt(IDnodes(:,6).^2+IDnodes(:,13).^2);
159.          szzgage=sqrt(IDnodes(:,7).^2+IDnodes(:,14).^2);
160.          sxygage=sqrt(IDnodes(:,8).^2+IDnodes(:,15).^2);
161.          syzgage=sqrt(IDnodes(:,9).^2+IDnodes(:,16).^2);
162.          sxzgage=sqrt(IDnodes(:,10).^2+IDnodes(:,17).^2);
163.          msumVMgages=sqrt((1/2)*((sxxgage-syygage).^2+...
164.              (syygage-szzgage).^2+(szzgage-sxxgage).^2+...
165.              6*(sxygage.^2+syzgage.^2+sxzgage.^2)));
166.      end
167.  end
168.  %Plots
169.  if plotting==1
170.      figure
171.      hold on
172.      grid on
173.      scatter(meanstress(:,2)/1000,msumVM/1000,0.5);
174.      % axis([0 160 0 50]);
175.      % xticks([0:10:160]);
176.      % title('100% Speed 1EOM1-3 Goodman')
177.      if size(mode,2)>1
178.          title([num2str(casepcs),'% Speed ',num2str(ND),...
179.              'EOM',num2str(mode(1)),'-',num2str(mode(end)),' Goodman'])
180.      else
181.          title([num2str(casepcs),'% Speed ',num2str(ND),...
182.              'EOM',num2str(mode),' Goodman'])
183.      end
184.      xlabel('mean stress [ksi]')
185.      ylabel('alternating stress [ksi]')
186.      if specgageloc==-1
187.          disp('Strain Gage Plot Disabled by user')
188.      else
189.      figure
190.      hold on
191.      grid on
192.      scatter(3,msumVMgages(1)/1000,'blue','d','filled');
193.      scatter(3,msumVMgages(2)/1000,'red','d','filled');
194.      scatter(3,msumVMgages(3)/1000,'black','d','filled');
195.      scatter(3,msumVMgages(4)/1000,'yellow','d','filled');
196.      axis([3 25 0 5]);
```

```
197.      xticks([3:1:25])
198.      yticks([0:0.5:5]);
199.      title('SG VM Summation By Group Modes')
200.      xlabel('number of modes');
201.      ylabel('stress amplitude (ksi)')
202.      legend('SG1','SG2','SG3','SG4')
203.      end
204.  else
205.      disp('Goodman Plots Disabled by user')
206.  end
207.  %Data Output
208.  if dataout==1
209.      outfile='test.txt';
210.      for k=1:size(mode,2)
211.          ampcout{k}=ampcsave{blade,ND,mode(k)}{blade,ND,mode(k)};
212.          coordsout=mshape{blade,ND,mode(k)}{blade,ND,mode(k)}(:,1:3);
213.          mshapeout{k}=mshape{blade,ND,mode(k)}...
214.          {blade,ND,mode(k)}(:,4:end);
215.          mstressout{k}=smagsave{blade,ND,mode(k)}{blade,ND,mode(k)}(:);
216.      end
217.      ampcout=cell2mat(ampcout.');
218.      mshapeout=cell2mat(mshapeout);
219.      mstressout=cell2mat(mstressout);
220.      ieksave=repmat((1:iek).',1,jek);
221.      ieksave=ieksave(:);
222.      jeksave=repelem((1:jek),iek,1);
223.      jeksave=jeksave(:);
224.      goodmanout=[meanstress msumVM];
225.      if size(mode,2)>1
226.          writecell({'FRAAME Output'},outfile);
227.      writecell({'FRAAME Output'},outfile);
228.      writecell({' '},outfile,'WriteMode','append');
229.          for m=1:size(mode,2)
230.              modeshapeout=[ieksave jeksave coordsout ...
231.                  mshapeout(:,size(mode,2)*m-2:size(mode,2)*m)];
232.              writecell({'blade',' EO',' mode'},...
233.                  outfile,'WriteMode','append');
234.              writecell({blade, ND, mode(m)},outfile,...
235.                  'Delimiter',',','WriteMode','append');
236.              writecell({' '},outfile,'WriteMode','append');
237.              writecell({'Forced Response Amplitude Mode ' ...
238.                  num2str(m)},outfile,'WriteMode','append');
239.              writecell({'FR amp real',' FR amp imag'},outfile,...
240.                  'WriteMode','append');
241.              writecell({ampcout(m,:)},outfile,'Delimiter',',',...
242.                  'WriteMode','append');
243.              writecell({' '},outfile,'WriteMode','append');
244.              writecell({'CFD Mesh [inches] and Mode Shape for Mode '...
245.                  num2str(m)},outfile,'WriteMode','append');
246.              writecell({'iek',' jek',' x',' y',' z',' ux',' uy',...
247.                  ' uz'},outfile,'WriteMode','append');
248.              fid=fopen(outfile,'a');
```

```
            fprintf(fid,'%i, %i, %e, %e, %e, %e, %e, %e,\n',...
                modeshapeout(:,:).');
            fclose(fid);
            writecell({' '},outfile,'WriteMode','append');
        end
        for m=1:size(mode,2)
            modestressout=[ieksave jeksave coordsout mstressout(:,m)];
            writecell({['CFD Mesh [inches] and Von Mises Modal ' ...
                'Stress for Mode '] num2str(m)},outfile,...
                'WriteMode','append');
            writecell({'iek',' jek',' x',' y',' z',' sigmaVM'},...
                outfile,'WriteMode','append');
            fid=fopen(outfile,'a');
            fprintf(fid,'%i, %i, %e, %e, %e, %e,\n',...
                modestressout(:,:).');
            fclose(fid);
            writecell({' '},outfile,'WriteMode','append');
        end
    writecell({['Goodman Diagram (Modal Summation) ' ...
        'Information for Modes '] num2str(mode(1)) '-' ...
        num2str(mode(end))},outfile,'WriteMode','append');
    writecell({'FEA Surface Node Number',' Mean Stress', ...
        ' Alternating Stress'},outfile,'WriteMode','append');
    fid=fopen(outfile,'a');
    fprintf(fid,'%i, %e, %e,\n',goodmanout(:,:).');
    fclose(fid);
    writecell({' '},outfile,'WriteMode','append');
    writecell({'End of File'},outfile,'WriteMode','append');
else
    writecell({'FRAAME Output'},outfile);
    writecell({' '},outfile,'WriteMode','append');
    modeshapeout=[ieksave jeksave coordsout mshapeout];
    writecell({'blade',' EO',' mode'},...
        outfile,'WriteMode','append');
    writecell({blade, ND, mode(1)},outfile,...
        'Delimiter',',','WriteMode','append');
    writecell({' '},outfile,'WriteMode','append');
    writecell({'Forced Response Amplitude Mode ' ...
        num2str(mode(1))},outfile,'WriteMode','append');
    writecell({'FR amp real',' FR amp imag'},outfile,...
        'WriteMode','append');
    writecell({ampcout(1,:)},outfile,'Delimiter',',',...
        'WriteMode','append');
    writecell({' '},outfile,'WriteMode','append');
    writecell({'CFD Mesh [inches] and Mode Shape for Mode '...
        num2str(mode(1))},outfile,'WriteMode','append');
    writecell({'iek',' jek',' x',' y',' z',' ux',' uy',...
        ' uz'},outfile,'WriteMode','append');
    fid=fopen(outfile,'a');
    fprintf(fid,'%i, %i, %e, %e, %e, %e, %e, %e,\n',...
        modeshapeout(:,:).');
    fclose(fid);
```

```
301.      writecell({' '},outfile,'WriteMode','append');
302.      modestressout=[ieksave jeksave coordsout mstressout(:,1)];
303.      writecell({['CFD Mesh [inches] and Von Mises Modal ' ...
304.          'Stress for Mode '] num2str(mode(1))},outfile,...
305.          'WriteMode','append');
306.     writecell({'iek',' jek',' x',' y',' z',' sigmaVM'},...
307.           outfile,'WriteMode','append');
308.     fid=fopen(outfile,'a');
309.     fprintf(fid,'%i, %i, %e, %e, %e, %e,\n',...
310.           modestressout(:,:).');
311.     fclose(fid);
312.     writecell({' '},outfile,'WriteMode','append');
313.     writecell({['Goodman Diagram (Modal Summation) ' ...
314.          'Information for Mode ' num2str(mode(1))]},...
315.          outfile,'WriteMode','append');
316.      writecell({'FEA Surface Node Number',' Mean Stress', ...
317.          ' Alternating Stress'},outfile,'WriteMode','append');
318.      fid=fopen(outfile,'a');
319.      fprintf(fid,'%i, %e, %e,\n',goodmanout(:,:).');
320.      fclose(fid);
321.      writecell({' '},outfile,'WriteMode','append');
322.      writecell({'End of File'},outfile,'WriteMode','append');
323.      end
324.  else
325.      disp('Data Output Disabled by User')
326.  end
327.  toc
```

Source code for masterlist.m:

```
1. function [coords] = masterlist(asciichars,workdir,Ansys_masterlist)
2. % Read Ansys Master List
3. % Import all Ansys nodes and coordinates (surface and internal)
4. % Fixed width delimit for chance of no white space between columns,
   str2num
5. % or split non-char to avoid char
6. disp('Importing ANSYS Coordinate Master List');
7. allcoords=importdata([workdir,Ansys_masterlist],'r');
8. allcoords(contains(allcoords,asciichars))=[];
9. if isempty(str2num(char(allcoords)))==1
10.       disp('Parsing Error: Non-Uniform Ansys Coordinate Entries')
11.       mat=cell2mat(allcoords);
12.       nodelist=str2num(mat(:,1:9));
13.       xlist=str2num(mat(:,10:23));
14.       ylist=str2num(mat(:,24:38));
15.       zlist=str2num(mat(:,39:51));
16.       coords=[nodelist xlist ylist zlist];
17.   else
18.       allcoords=str2num(allcoords);
19.       nodelist=allcoords(:,1);
20.       xlist=allcoords(:,2);
21.       ylist=allcoords(:,3);
```

```
22.       zlist=allcoords(:,4);
23.       coords=[nodelist xlist ylist zlist];
24.   end
25.   disp('ANSYS Coordinates Loaded. Begin Master Loop.');
26.   end
```

Source code for prhistread.m:

```
1. function [iek,jek,ij,ijm,n,xc,yc,zc,axc,ayc,azc,xt,yt,...
2.     zt,pressure,prhist] = prhistread(numrev,tpr,diam,...
3.     gamma,pref,hl,workdir)
4. % TURBO prhist reader for old prhist format (binary) SINGLE BLADE (WIP)
5. % converted to stream format (ncyct converted to real*8).
6. % Reads binary info (header, non-dimensional coords, areas,
7. % pressures, ncyct, time). Calculates pforce components and
8. % magnitudes for specified rev (usually last for periodicity).
9. % Input: numrev,tpr,diam,gamma,pref,hl
10.   % Output:iek,jek,timesteps,ij,ijm,n,xc,yc,zc,axc,ayc,azc,xt,yt,
11.   % zt,axt,ayt,azt,pressure
12.   disp('Locating TURBO prhist for blade 1');
13.   floc=[workdir,'prhist.'];
14.   fext='.hs_stream';
15.   prhist=[floc,num2str(1),fext];
16.   fid=fopen(prhist,'r');
17.   fprintf('First File Read is in Directory %s\n',prhist);
18.   header=fread(fid,hl,'int32','ieee-be');
19.   nbr=header(1);
20.   ibk=header(2);
21.   iek=header(3);
22.   jbk=header(4);
23.   jek=header(5);
24.   isuct=header(6);
25.   nprt=header(7);
26.   ntc=header(8);
27.   timesteps=nprt*ntc;
28.   ij=iek*jek;
29.   ijm=(iek-1)*(jek-1);
30.   intbc=4;
31.   realbc=8;
32.   hbc=hl*intbc;
33.   cabc=(3*ij+3*ijm)*realbc;
34.   fseek(fid,hbc,'bof');
35.   ca=fread(fid,cabc/realbc,'real*8','ieee-be');
36.   fseek(fid,hbc+cabc,'bof');
37.   rest=fread(fid,'real*8','ieee-be');
38.   fclose(fid);
39.   disp('TURBO Constants Loaded');
40.   % Rev Selection
41.   if numrev==0
42.       ntm1=1;
43.       ntm2=timesteps;
44.       fprintf('Analyzing All Revs \n');
```

```
45.   elseif numrev>0
46.       ntm1=(numrev-1)*tpr+1;
47.       ntm2=numrev*tpr;
48.       fprintf('Analyzing First Rev \n');
49.   elseif numrev<0
50.       ntm1=timesteps+(numrev)*tpr+1;
51.       ntm2=timesteps+(numrev+1)*tpr;
52.       fprintf('Analyzing Last Rev \n');
53.   end
54.   n=ntm2-ntm1+1;
55.   % Coordinates, Areas, Unsteady Pressures
56.   xc=ca(ibk:ij);
57.   yc=ca(ij+1:2*ij);
58.   zc=ca(2*ij+1:3*ij);
59.   axc=ca(3*ij+1:3*ij+ijm);
60.   ayc=ca(3*ij+ijm+1:3*ij+2*ijm);
61.   azc=ca(3*ij+2*ijm+1:3*ij+3*ijm);
62.   xt=reshape(xc,[iek jek]);
63.   yt=reshape(yc,[iek jek]);
64.   zt=reshape(zc,[iek jek]);
65.   axt=reshape(axc,[iek-1 jek-1]);
66.   ayt=reshape(ayc,[iek-1 jek-1]);
67.   azt=reshape(azc,[iek-1 jek-1]);
68.   % Block Structure and Pressures
69.   lastblock=reshape(rest,[ijm+2 timesteps]);
70.   ncyct=lastblock(1,:)';
71.   time=lastblock(2,:)';
72.   lastblock([1 2],:)=[];
73.   allpressure=lastblock*gamma*pref;
74.   pressure=allpressure(:,ntm1:ntm2);
75.   pfx=sum(diam*diam*axc.*pressure)';
76.   pfy=sum(diam*diam*ayc.*pressure)';
77.   pfz=sum(diam*diam*azc.*pressure)';
78.   pfmag=sqrt((pfx.^2)+(pfy.^2)+(pfz.^2));
79.   pforce=[pfx pfy pfz pfmag];
80.   phist=fft(pfmag);
81.   re=real(phist);
82.   im=imag(phist);
83.   freal=2*(re/n);
84.   fimag=-2*(im/n);
85.   fmag=sqrt((freal).^2+(fimag).^2);
86.   fphase=atan(fimag./freal)*(180/pi);
87.   disp('TURBO Pressures/Areas/Coords Loaded');
88.   end
```

Source code for arne.m:

```
1. function [x,y,z,mfmag,amp,mshape,usumall,tipdisp,wn,w,ampr,ampi,freq1,...
2.     phase1,asf,allcoords,usumsave,ampcsave] = arne(p,...
3.     i,j,asciichars,coords,xt,yt,zt,xc,yc,zc,...
4.     diam,nbd,ij,ijm,iek,jek,pressure,axc,ayc,azc,n,pcs,prhist,zeta,...
5.     freqmargin,casepcs,workdir,wn,specnatfreq,plotting)
```

```
% Ansys Reader Normal Expanded for ND not 0 or nbd/2
% master function file for importing, sorting, and computing
% modal information (forces, displacements, stresses) for TCT 7-28-3
% Included in master loop: establish directory (in Windows),
   % read file info (keywords, frequency, phase), fixed delimit
   % modal info (ux, uy, uz, usum), filter coordinates from master list,
   % remove most disk nodes, interpolate displacements onto CFD mesh,
   % apply cyclic scaling factor (EXPANDED ONLY), average displacements
  over
   % cells, calculate modal force and forced response amplitude, save mode
   % shapes for output to text file.
   % Input: {p,i,j} or {blade,ND,mode}; asciichars for filtering; coords
  from
   % master list, (xt1,yt1,zt1) uniform CFD mesh coords; diam for blade
   % dimensionalization; nbd number of blades; ij total CFD nodes;
   % ijm total minus one; (iek,jek)
   % nodes minus one for areas; pressure1 full pressure data;
  (axc1,ayc1,azc1)
   % all CFD areas in column format; n for fft sample size; pcs speed;
  prhist1
   % to reference which prhist info is taken from; zeta damping
   % Output: All variables used in computation for debugging
   file=[workdir,'R24-',num2str(casepcs),'pt-ND',num2str(i),...
       '-M',num2str(j),'-Modal-Disp-List'];
   fprintf('Displacement File is in %s\n',file);
   filelog{p,i,j}=file;
   alldisp=importdata(file,'r');
   idx=find(contains(alldisp,'NODE       UX'));
   idx2=find(contains(alldisp,'*****'));
   idx3=find(contains(alldisp,'MAX'));
   idxf=find(contains(alldisp,'FREQ='));
   idxp=find(contains(alldisp,'PHASE ANGLE ='));
   freqcell=char(alldisp(idxf(1)));
   freq1=str2num(freqcell(10:22));
   phasecell=char(alldisp(idxp(1)));
   phase1=str2num(phasecell(59:71));
   for q=1:nbd
       allblocks{q}=alldisp((idx2((size(idx2,1)/nbd)*(q-1)+1)):idx3(q));
   end
   sector=1;
   ublock=allblocks{sector};
   ublock(contains(ublock,asciichars))=[];
   ublock=deblank(ublock);
   ublock(cellfun('isempty',ublock))=[];
   ublock=char(ublock);
   allnodes=str2num(ublock(:,1:8));
   allux=str2num(ublock(:,9:21));
   alluy=str2num(ublock(:,22:33));
   alluz=str2num(ublock(:,34:45));
   allusum=str2num(ublock(:,46:end));
   fprintf('Modal Displacements (Expanded Model) Loaded for %s\n',file);
   % Filter Coordinates
```

```
54.   dn=find(ismember(coords(:,1),allnodes,'rows'));
55.   allcoords=coords(dn,:);
56.   x=allcoords(:,2);
57.   y=allcoords(:,3);
58.   z=allcoords(:,4);
59.   % Interpolation (ANSYS ---> TURBO)
60.   uxnat=scatteredInterpolant(x,y,z,allux,'natural');
61.   uynat=scatteredInterpolant(x,y,z,alluy,'natural');
62.   uznat=scatteredInterpolant(x,y,z,alluz,'natural');
63.   usumnat=scatteredInterpolant(x,y,z,allusum,'natural');
64.   ux=uxnat(xt*diam,yt*diam,zt*diam);
65.   uy=uynat(xt*diam,yt*diam,zt*diam);
66.   uz=uznat(xt*diam,yt*diam,zt*diam);
67.   usum=usumnat(xt*diam,yt*diam,zt*diam);
68.   fprintf('ANSYS Displacements Interpolated onto TURBO mesh for
   %s\n',file);
69.   % ANSYS Scaling Factor [FOR EXPANDED FILE(S) ONLY]
70.   ND=i;
71.   if ND==0 || ND==nbd/2
72.       m=1;
73.       asf=sqrt(nbd/m);
74.       ux=ux*asf;
75.       uy=uy*asf;
76.       uz=uz*asf;
77.       usum=usum*asf;
78.   else
79.       m=2;
80.       asf=sqrt(nbd/m);
81.       ux=ux*asf;
82.       uy=uy*asf;
83.       uz=uz*asf;
84.       usum=usum*asf;
85.   end
86.   fprintf('ANSYS Scaling Factor Applied for %s\n',file);
87.   % Cell Averaged Displacements
88.   uxc=reshape(ux,[ij,1]);
89.   uyc=reshape(uy,[ij,1]);
90.   uzc=reshape(uz,[ij,1]);
91.   usumc=reshape(usum,[ij,1]);
92.   usumsave{p,i,j}=usum;
93.   usumall{p,i,j}=usumc;
94.       for k=1:iek-1
95.           for l=1:jek-1
96.               uxa(k,l)=(ux(k,l)+ux(k+1,l)+ux(k,l+1)+ux(k+1,l+1))/4;
97.               uya(k,l)=(uy(k,l)+uy(k+1,l)+uy(k,l+1)+uy(k+1,l+1))/4;
98.               uza(k,l)=(uz(k,l)+uz(k+1,l)+uz(k,l+1)+uz(k+1,l+1))/4;
99.               usuma(k,l)=sqrt(uxa(k,l).^2+uya(k,l).^2+uza(k,l).^2);
100.          end
101.      end
102.  uxac=reshape(uxa,[ijm 1]);
103.  uyac=reshape(uya,[ijm 1]);
104.  uzac=reshape(uza,[ijm 1]);
```

```
% Save Mode Shapes
% dx{p,i,j}=uxc;
% dy{p,i,j}=uyc;
% dz{p,i,j}=uzc;
% Modal Force Time History
mf=sum(diam*diam*pressure.*((axc.*uxac)+(ayc.*uyac)+(azc.*uzac)))';
mfhist=fft(mf);
mre=real(mfhist);
mim=imag(mfhist);
mreal=2.*(mre/n);
mimag=-2.*(mim/n);
mfmag=sqrt((mreal).^2+(mimag).^2);
mfphase=atan2d(mimag,mreal);
fprintf('Modal Force Calculated for %s\n',prhist);
mfmagall{p,i,j}=mfmag;
mfall{p,i,j}=mf;
mfphasec{p,i,j}=mfphase;
mfrsave{p,i,j}=mreal;
mfisave{p,i,j}=mimag;
% Modal Stress
% Freqency Filter
if specnatfreq==1
    wn;
elseif specnatfreq==0
    wn=freq1;
end
% wn=freq1;
% wn=679.5;
w=(pcs/60)*(i);
wall{p,i,j}=w;
wnall{p,i,j}=wn;
            if w>=wn-(wn*freqmargin) && w<=wn+(wn*freqmargin)
                w=wn;
                disp('Resonant Case Within 10% Margin');
            else
                w;
                disp('Non-Resonant Case');
            end
k=(2*pi*wn)^2;
amp=(mfmag(i+1)/k)/(sqrt(((1-(w^2)/(wn^2))^2)+(2*zeta*(w/wn)).^2));
ampsave{p,i,j}=amp;
% Complex Notation
ampr=(mreal(i+1)/k)/(sqrt(((1-(w^2)/(wn^2))^2)+(2*zeta*(w/wn)).^2));
ampi=(mimag(i+1)/k)/(sqrt(((1-(w^2)/(wn^2))^2)+(2*zeta*(w/wn)).^2));
ampcsave{p,i,j}=[ampr ampi];
mshape{p,i,j}=[xc*diam yc*diam zc*diam uxc uyc uzc];
dlaser=amp.*uzc*1000;
tipdisp{p,i,j}=dlaser;
% Plotting (blade tip displacement and mode shape contours)
% Comment out lines if plots not desired
if plotting==1
    figure
```

```
157.      scatter(xc(ij-iek:end),dlaser(ij-iek:end));
158.      title(['Blade Tip Displacement - Mode ',num2str(j)])
159.      xlabel('chord [nondimensional]')
160.      ylabel('tip disp [mils]')
161.      figure
162.      surf(xt*diam,yt*diam,zt*diam,usum);
163.      title(['Mode ',num2str(j),' Mode Shape'])
164.      view([0 90])
165.      xlim([-3 3])
166.      xlabel('axial [in]')
167.      ylabel('spanwise [in]')
168.      zlabel('z')
169.      shading interp
170.      colormap jet(13)
171.      colorbar
172.  else
173.      disp('Tip Displacement and Mode Shape Plots Disabled by User')
174.  end
175.  end
```

Source code for arnu.m:

```
1. function [x,y,z,mfmag,amp,mshape,tipdisp,wn,w,ampr,ampi,freq1,...
2.     phase1,allcoords,usumsave,ampcsave] = arnu(p,...
3.     i,j,asciichars,coords,xt,yt,zt,xc,yc,zc,diam,nbd,...
4.     ij,ijm,iek,jek,pressure,axc,ayc,azc,n,pcs,prhist,zeta,freqmargin,...
5.     casepcs,workdir,wn,specnatfreq,plotting)
6. % Ansys Reader Normal Unexpanded for ND not 0 or nbd/2
7. % master function file for importing, sorting, and computing
8. % modal information (forces, displacements, stresses) for TCT 7-28-3
9. % Included in master loop: establish directory (in Windows),
10.   % read file info (keywords, frequency, phase), fixed delimit
11.   % modal info (ux, uy, uz, usum), filter coordinates from master list,
12.   % remove most disk nodes, interpolate displacements onto CFD mesh,
13.   % apply cyclic scaling factor (EXPANDED ONLY), average displacements
   over
14.   % cells, calculate modal force and forced response amplitude, save mode
15.   % shapes for output to text file.
16.   % Input: {p,i,j} or {blade,ND,mode}; asciichars for filtering; coords
   from
17.   % master list, (xt1,yt1,zt1) uniform CFD mesh coords; diam for blade
18.   % dimensionalization; nbd number of blades; ij total CFD nodes;
19.   % ijm total minus one; (iek,jek)
20.   % nodes minus one for areas; pressure1 full pressure data;
   (axc1,ayc1,azc1)
21.   % all CFD areas in column format; n for fft sample size; pcs speed;
   prhist1
22.   % to reference which prhist info is taken from; zeta damping
23.   % Output: All variables used in computation for debugging
24.   file=[workdir,'R24-',num2str(casepcs),'pt-ND',num2str(i),...
25.       '-M',num2str(j),'-Modal-Disp-List-Unexp'];
26.   fprintf('Displacement File is in %s\n',file);
```

```
27.   filelog{p,i,j}=file;
28.   alldisp=importdata(file,'r');
29.   idx=find(contains(alldisp,'NODE       UX'));
30.   idx2=find(contains(alldisp,'*****'));
31.   idx3=find(contains(alldisp,'MAX'));
32.   idxf=find(contains(alldisp,'FREQ='));
33.   idxp=find(contains(alldisp,'PHASE ANGLE ='));
34.   freqcell=char(alldisp(idxf(1)));
35.   freq1=str2num(freqcell(10:22));
36.   phasecell=char(alldisp(idxp(1)));
37.   phase1=str2num(phasecell(59:71));
38.   for q=1:nbd
39.       allblocks{q}=alldisp((idx2((size(idx2,1)/nbd)*(q-1)+1)):idx3(q));
40.   end
41.   sector=1;
42.   ublock=allblocks{sector};
43.   ublock(contains(ublock,asciichars))=[];
44.   ublock=deblank(ublock);
45.   ublock(cellfun('isempty',ublock))=[];
46.   ublock=char(ublock);
47.   allnodes=str2num(ublock(:,1:8));
48.   allux=str2num(ublock(:,9:22));
49.   alluy=str2num(ublock(:,23:35));
50.   alluz=str2num(ublock(:,36:48));
51.   allusum=str2num(ublock(:,49:end));
52.   fprintf('Modal Displacements (Unexpanded Model) Loaded for %s\n',file);
53.   % Filter Coordinates
54.   dn=find(ismember(coords(:,1),allnodes,'rows'));
55.   allcoords=coords(dn,:);
56.   x=allcoords(:,2);
57.   y=allcoords(:,3);
58.   z=allcoords(:,4);
59.   % Interpolation (ANSYS ---> TURBO)
60.   uxnat=scatteredInterpolant(x,y,z,allux,'natural');
61.   uynat=scatteredInterpolant(x,y,z,alluy,'natural');
62.   uznat=scatteredInterpolant(x,y,z,alluz,'natural');
63.   usumnat=scatteredInterpolant(x,y,z,allusum,'natural');
64.   ux=uxnat(xt*diam,yt*diam,zt*diam);
65.   uy=uynat(xt*diam,yt*diam,zt*diam);
66.   uz=uznat(xt*diam,yt*diam,zt*diam);
67.   usum=usumnat(xt*diam,yt*diam,zt*diam);
68.   fprintf('ANSYS Displacements Interpolated onto TURBO mesh for
   %s\n',file);
69.   % Cell Averaged Displacements
70.   uxc=reshape(ux,[ij,1]);
71.   uyc=reshape(uy,[ij,1]);
72.   uzc=reshape(uz,[ij,1]);
73.   usumc=reshape(usum,[ij,1]);
74.   usumsave{p,i,j}=usum;
75.   usumall{p,i,j}=usumc;
76.       for k=1:iek-1
77.           for l=1:jek-1
```

```
78.                 uxa(k,l)=(ux(k,l)+ux(k+1,l)+ux(k,l+1)+ux(k+1,l+1))/4;
79.                 uya(k,l)=(uy(k,l)+uy(k+1,l)+uy(k,l+1)+uy(k+1,l+1))/4;
80.                 uza(k,l)=(uz(k,l)+uz(k+1,l)+uz(k,l+1)+uz(k+1,l+1))/4;
81.                 usuma(k,l)=sqrt(uxa(k,l).^2+uya(k,l).^2+uza(k,l).^2);
82.             end
83.         end
84.     uxac=reshape(uxa,[ijm 1]);
85.     uyac=reshape(uya,[ijm 1]);
86.     uzac=reshape(uza,[ijm 1]);
87.     % % Save Mode Shapes
88.     % dx{p,i,j}=uxc;
89.     % dy{p,i,j}=uyc;
90.     % dz{p,i,j}=uzc;
91.     % Modal Force Time History
92.     mf=sum(diam*diam*pressure.*((axc.*uxac)+(ayc.*uyac)+(azc.*uzac)))';
93.     mfhist=fft(mf);
94.     mre=real(mfhist);
95.     mim=imag(mfhist);
96.     mreal=2.*(mre/n);
97.     mimag=-2.*(mim/n);
98.     mfmag=sqrt((mreal).^2+(mimag).^2);
99.     mfphase=atan2d(mimag,mreal);
100.    fprintf('Modal Force Calculated for %s\n',prhist);
101.    mfmagall{p,i,j}=mfmag;
102.    mfall{p,i,j}=mf;
103.    mfphasec{p,i,j}=mfphase;
104.    mfrsave{p,i,j}=mreal;
105.    mfisave{p,i,j}=mimag;
106.    % Modal Stress
107.    % Freqency Filter
108.    if specnatfreq==1
109.        wn;
110.    elseif specnatfreq==0
111.        wn=freq1;
112.    end
113.    w=(pcs/60)*(i);
114.    wall{p,i,j}=w;
115.    wnall{p,i,j}=wn;
116.                if w>=wn-(wn*freqmargin) && w<=wn+(wn*freqmargin)
117.                    w=wn;
118.                    disp('Resonant Case Within 10% Margin');
119.                else
120.                    w;
121.                    disp('Non-Resonant Case');
122.                end
123.    k=(2*pi*wn)^2;
124.    amp=(mfmag(i+1)/k)/(sqrt(((1-(w^2)/(wn^2))^2)+(2*zeta*(w/wn)).^2));
125.    ampsave{p,i,j}=amp;
126.    % Complex Notation
127.    ampr=(mreal(i+1)/k)/(sqrt(((1-(w^2)/(wn^2))^2)+(2*zeta*(w/wn)).^2));
128.    ampi=(mimag(i+1)/k)/(sqrt(((1-(w^2)/(wn^2))^2)+(2*zeta*(w/wn)).^2));
129.    ampcsave{p,i,j}=[ampr ampi];
```

```
130.  mshape{p,i,j}=[xc*diam yc*diam zc*diam uxc uyc uzc];
131.  dlaser=amp.*uzc*1000;
132.  tipdisp{p,i,j}=dlaser;
133.  % Plotting (blade tip displacement and mode shape contours)
134.  % Comment out lines if plots not desired
135.  if plotting==1
136.      figure
137.      scatter(xc(ij-iek:end),dlaser(ij-iek:end));
138.      xlabel('chord [nondimensional]')
139.      ylabel('tip disp [mils]')
140.      figure
141.      surf(xt*diam,yt*diam,zt*diam,usum);
142.      title(['Mode ',num2str(j),' Mode Shape'])
143.      view([0 90])
144.      xlim([-3 3])
145.      xlabel('axial [in]')
146.      ylabel('spanwise [in]')
147.      zlabel('z')
148.      shading interp
149.      colormap jet(13)
150.      colorbar
151.  else
152.      disp('Plotting Disabled by user')
153.  end
154.  end
```

Source code for r24arnes.m:

```
1. function [altsVM,fsclist,altsVMcomp,...
2.     stresscontoursreal,stresscontoursimag,...
3.     stressrealout,stressimagout,smagsave,...
4.     matstressallu] = r24arnes(p,i,j,freq1,phase1,...
5.     asciichars,amp,ampr,ampi,asf,workdir,casepcs,coords,xt,yt,zt,...
6.     diam,plotting)
7. % R24 Ansys Reader Normal (ND not 0 or 8) Expanded Stresses
8. % Search main directory for expanded stress file, compare
9. % stress/displacement phase and frequency to match data sets, (R24 ONLY)
10.   % read material types (blade steel? disk titanium) and format, average
11.   % stress values at shared nodes between blade/disk, scale info by
   cyclic
12.   % factor and forced response amplitude (magnitude notation for
13.   % Goodman/complex notation for strain gages), output Von Mises
14.   % (alternating) stresses for Goodman diagram for each mode input,
   select
15.   % coordinates and selected nodes for strain gage measurements from
   stress
16.   % info
17.   file=[workdir,'R24-',num2str(casepcs),'pt-ND',num2str(i),...
18.       '-M',num2str(j),'-Modal-Comp-Stress-List'];
19.   fprintf('Stress File is in %s\n',file);
20.   sfilelog{p,i,j}=file;
21.   allstress=importdata(file,'r');
```

```
22.   idx=find(contains(allstress,'MATERIAL  15'));
23.   idx2=find(contains(allstress,'MATERIAL  16'));
24.   idx3=find(contains(allstress,'MINIMUM'));
25.   idx4=find(contains(allstress,'FREQ='));
26.   idx5=find(contains(allstress,'PHASE ANGLE ='));
27.   freqcell2=char(allstress(idx4(1)));
28.   freq2=str2num(freqcell2(10:22));
29.   frequency2{p,i,j}=freq2;
30.   disp('Frequency Check');
31.   if freq1==freq1
32.       freq=freq1;
33.       disp('Frequency Match');
34.   else
35.       Error('Frequency Mismatch: Check Displacement and Stress File
   Names');
36.   end
37.   phasecell2=char(allstress(idx5(1)));
38.   phase2=str2num(phasecell2(59:71));
39.   phaseangle2{p,i,j}=phase2;
40.   disp('Phase Check');
41.   if phase1==phase2
42.   phaseangle=phase1;
43.   disp('Phase Match');
44.   else
45.       Error('Phase Mismatch: Check Displacement and Stress File Names');
46.   end
47.   fprintf('Modal Stress Averaging - Different Material Types for
   %s\n',file);
48.   mat15=allstress(idx(1):idx2(1));
49.   mat15(contains(mat15,asciichars))=[];
50.   mat15=deblank(mat15);
51.   mat15(cellfun('isempty',mat15))=[];
52.   mat15=char(mat15);
53.   sn15=str2num(mat15(:,1:8));
54.   sx15=str2num(mat15(:,9:21));
55.   sy15=str2num(mat15(:,22:33));
56.   sz15=str2num(mat15(:,34:45));
57.   sxy15=str2num(mat15(:,46:57));
58.   syz15=str2num(mat15(:,58:69));
59.   sxz15=str2num(mat15(:,70:end));
60.   sall15=[sn15 sx15 sy15 sz15 sxy15 syz15 sxz15];
61.   mat16=allstress(idx2(1):idx3(1));
62.   mat16(contains(mat16,asciichars))=[];
63.   mat16=deblank(mat16);
64.   mat16(cellfun('isempty',mat16))=[];
65.   mat16=char(mat16);
66.   sn16=str2num(mat16(:,1:8));
67.   sx16=str2num(mat16(:,9:21));
68.   sy16=str2num(mat16(:,22:33));
69.   sz16=str2num(mat16(:,34:45));
70.   sxy16=str2num(mat16(:,46:57));
71.   syz16=str2num(mat16(:,58:69));
```

```
72.   sxz16=str2num(mat16(:,70:end));
73.   sall16=[sn16 sx16 sy16 sz16 sxy16 syz16 sxz16];
74.   shared15=find(ismember(sall15(:,1),sall16(:,1),'rows'));
75.   shared16=find(ismember(sall16(:,1),sall15(:,1),'rows'));
76.   s15fil=sall15(shared15,:);
77.   s16fil=sall16(shared16,:);
78.   ssxavg=mean([s15fil(:,2) s16fil(:,2)],2);
79.   ssyavg=mean([s15fil(:,3) s16fil(:,3)],2);
80.   sszavg=mean([s15fil(:,4) s16fil(:,4)],2);
81.   ssxyavg=mean([s15fil(:,5) s16fil(:,5)],2);
82.   ssyzavg=mean([s15fil(:,6) s16fil(:,6)],2);
83.   ssxzavg=mean([s15fil(:,7) s16fil(:,7)],2);
84.   ssall=[s15fil(:,1) ssxavg ssyavg sszavg ssxyavg ssyzavg ssxzavg];
85.   s15ri=find(ismember(sall15(:,1),ssall(:,1),'rows'));
86.   s16ri=find(ismember(sall16(:,1),ssall(:,1),'rows'));
87.   sall15(s15ri,:)=[];
88.   sall16(s16ri,:)=[];
89.   stressnodes=[sall15(:,1);sall16(:,1);ssall(:,1)];
90.   stressscaled=amp.*asf.*[sall15(:,2:end);sall16(:,2:end);ssall(:,2:end)]
   ;
91.   matstressall=[stressnodes stressscaled];
92.   matstressallu=[sall15; sall16; ssall];
93.   fprintf('Real Stresses Node Averaged and Scaled for %s\n',file);
94.   fsclist=sortrows(matstressall,'ascend');
95.   altsVM=sqrt((1/2)*((fsclist(:,2)-fsclist(:,3)).^2+(fsclist(:,3)...
96.       -fsclist(:,4)).^2+(fsclist(:,4)-fsclist(:,2)).^2+...
97.       6*(fsclist(:,5).^2+fsclist(:,6).^2+fsclist(:,7).^2)));
98.   stressreal=ampr.*asf.*[sall15(:,2:end);sall16(:,2:end);ssall(:,2:end)];
99.   stressimag=ampi.*asf.*[sall15(:,2:end);sall16(:,2:end);ssall(:,2:end)];
100.  matstressallreal=[stressnodes stressreal];
101.  matstressallimag=[stressnodes stressimag];
102.  fsclistreal=sortrows(matstressallreal,'ascend');
103.  fsclistimag=sortrows(matstressallimag,'ascend');
104.  fsclistmag=[fsclistreal(:,1) ...
105.      sqrt(fsclistreal(:,2).^2+fsclistimag(:,2).^2) ...
106.      sqrt(fsclistreal(:,3).^2+fsclistimag(:,3).^2) ...
107.      sqrt(fsclistreal(:,4).^2+fsclistimag(:,4).^2) ...
108.      sqrt(fsclistreal(:,5).^2+fsclistimag(:,5).^2) ...
109.      sqrt(fsclistreal(:,6).^2+fsclistimag(:,6).^2) ...
110.      sqrt(fsclistreal(:,7).^2+fsclistimag(:,7).^2)];
111.  altsVMcomp=sqrt((1/2)*((fsclistmag(:,2)-fsclistmag(:,3)).^2+...
112.      (fsclistmag(:,3)-fsclistmag(:,4)).^2+(fsclistmag(:,4)...
113.      -fsclistmag(:,2)).^2+6*(fsclistmag(:,5).^2+fsclistmag(:,6).^2+...
114.      fsclistmag(:,7).^2)));
115.  stressrealout{p,i,j}=fsclistreal;
116.  stressimagout{p,i,j}=fsclistimag;
117.  % Complex Stress Interp
118.  dn=find(ismember(coords(:,1),fsclistreal(:,1),'rows'));
119.  all=coords(dn,:);
120.  x=all(:,2);
121.  y=all(:,3);
122.  z=all(:,4);
```

```
sxxnatr=scatteredInterpolant(x,y,z,fsclistreal(:,2));
syynatr=scatteredInterpolant(x,y,z,fsclistreal(:,3));
szznatr=scatteredInterpolant(x,y,z,fsclistreal(:,4));
sxynatr=scatteredInterpolant(x,y,z,fsclistreal(:,5));
syznatr=scatteredInterpolant(x,y,z,fsclistreal(:,6));
sxznatr=scatteredInterpolant(x,y,z,fsclistreal(:,7));
sxxnati=scatteredInterpolant(x,y,z,fsclistimag(:,2));
syynati=scatteredInterpolant(x,y,z,fsclistimag(:,3));
szznati=scatteredInterpolant(x,y,z,fsclistimag(:,4));
sxynati=scatteredInterpolant(x,y,z,fsclistimag(:,5));
syznati=scatteredInterpolant(x,y,z,fsclistimag(:,6));
sxznati=scatteredInterpolant(x,y,z,fsclistimag(:,7));
sxxr=sxxnatr(xt*diam,yt*diam,zt*diam);
syyr=syynatr(xt*diam,yt*diam,zt*diam);
szzr=szznatr(xt*diam,yt*diam,zt*diam);
sxyr=sxynatr(xt*diam,yt*diam,zt*diam);
syzr=syznatr(xt*diam,yt*diam,zt*diam);
sxzr=sxznatr(xt*diam,yt*diam,zt*diam);
sxxi=sxxnati(xt*diam,yt*diam,zt*diam);
syyi=syynati(xt*diam,yt*diam,zt*diam);
szzi=szznati(xt*diam,yt*diam,zt*diam);
sxyi=sxynati(xt*diam,yt*diam,zt*diam);
syzi=syznati(xt*diam,yt*diam,zt*diam);
sxzi=sxznati(xt*diam,yt*diam,zt*diam);
stresscontoursreal{p,i,j}=[{sxxr} {syyr} {szzr} {sxyr} {syzr} {sxzr}];
stresscontoursimag{p,i,j}=[{sxxi} {syyi} {szzi} {sxyi} {syzi} {sxzi}];
sxxmag=sqrt(sxxr.^2+sxxi.^2);
syymag=sqrt(syyr.^2+syyi.^2);
szzmag=sqrt(szzr.^2+szzi.^2);
sxymag=sqrt(sxyr.^2+sxyi.^2);
syzmag=sqrt(syzr.^2+syzi.^2);
sxzmag=sqrt(sxzr.^2+sxzi.^2);
smagVM=sqrt((1/2)*((sxxmag-syymag).^2+(syymag-szzmag).^2+...
    (szzmag-sxxmag).^2+6*(sxymag.^2+syzmag.^2+sxzmag.^2)));
smagsave{p,i,j}=smagVM;
% Contour plotting
% Comment out lines if plots not desired
if plotting==1
    figure
    surf(xt*diam,yt*diam,zt*diam,smagVM);
    title(['Mode ',num2str(j),' VM Stress Contour'])
    view([0 90])
    xlim([-3 3])
    xlabel('axial [in]')
    ylabel('spanwise [in]')
    zlabel('z')
    shading interp
    colormap jet(13)
    colorbar
else
    disp('Modal Stress Plots Disabled by user')
end
```

```
175.  end
```

Source code for gages.m:

```
1. function [IDnodes,SGinfo] = gages(specgageloc,gageloc,gagenode,...
2.     matstressallu,allcoords,ampr,ampi,asf,mode,p,i,j)
3. %need to consolidate gage locations/orientations into a looped script so
4. %a few lines at beginning of main script serve to manipulate gage
   positions
5. %and orientations. start with case of 4 gages with UTRC data and compare
6. %with James' paper
7. % USE COMPLEX NOTATION FOR MODAL SUMMATION
8. if specgageloc==1
9.     % Establish tolerances for each gage, assuming 4 gages for this
10.       % analysis, play with parameters to capture more/less nodes
11.       % WIP: Extend logic for more robust search routine
12.       tols=[0.001 0.01 0.01; ...
13.           0.001 0.01 0.01; ...
14.           0.01 0.001 0.01; ...
15.           0.15 0.001 0.01];
16.       ycon1=allcoords(find(allcoords(:,3)>=gageloc(1,2)-tols(1,2) ...
17.           & allcoords(:,3)<=gageloc(1,2)+tols(1,2)),:);
18.       xcon1=ycon1(find(ycon1(:,2)>=gageloc(1,1)-tols(1,1) ...
19.           & ycon1(:,2)<=gageloc(1,1)+tols(1,1)),:);
20.       zcon1=xcon1(find(xcon1(:,4)>=gageloc(1,3)-tols(1,3) ...
21.           & xcon1(:,4)<=gageloc(1,3)+tols(1,3)),:);
22.       sg1node=zcon1(1,1);
23.       sg1i=find(matstressallu(:,1)==sg1node);
24.       sg1stressr=matstressallu(sg1i,:)*ampr*asf;
25.       sg1stressi=matstressallu(sg1i,:)*ampi*asf;
26.       sigmaVM1real=sqrt((1/2)*((sg1stressr(2)-sg1stressr(3)).^2+...
27.           (sg1stressr(3)-sg1stressr(4)).^2+(sg1stressr(4)...
28.           -sg1stressr(2)).^2+6*(sg1stressr(5).^2+sg1stressr(6).^2+...
29.           sg1stressr(7).^2)));
30.       sigmaVM1imag=sqrt((1/2)*((sg1stressi(2)-sg1stressi(3)).^2+...
31.           (sg1stressi(3)-sg1stressi(4)).^2+(sg1stressi(4)...
32.           -sg1stressi(2)).^2+6*(sg1stressi(5).^2+sg1stressi(6).^2+...
33.           sg1stressi(7).^2)));
34.
35.       ycon2=allcoords(find(allcoords(:,3)>=gageloc(2,2)-tols(2,2) ...
36.           & allcoords(:,3)<=gageloc(2,2)+tols(2,2)),:);
37.       xcon2=ycon2(find(ycon2(:,2)>=gageloc(2,1)-tols(2,1) ...
38.           & ycon2(:,2)<=gageloc(2,1)+tols(2,1)),:);
39.       zcon2=xcon2(find(xcon2(:,4)>=gageloc(2,3)-tols(2,3) ...
40.           & xcon2(:,4)<=gageloc(2,3)+tols(2,3)),:);
41.       sg2node=zcon2(1,1);
42.       sg2i=find(matstressallu(:,1)==sg2node);
43.       sg2stressr=matstressallu(sg2i,:)*ampr*asf;
44.       sg2stressi=matstressallu(sg2i,:)*ampi*asf;
45.       sigmaVM2real=sqrt((1/2)*((sg2stressr(2)-sg2stressr(3)).^2+...
46.           (sg2stressr(3)-sg2stressr(4)).^2+(sg2stressr(4)...
47.           -sg2stressr(2)).^2+6*(sg2stressr(5).^2+...
```

```
48.           sg2stressr(6).^2+sg2stressr(7).^2)));
49.       sigmaVM2imag=sqrt((1/2)*((sg2stressi(2)-sg2stressi(3)).^2+...
50.           (sg2stressi(3)-sg2stressi(4)).^2+(sg2stressi(4)-...
51.           sg2stressi(2)).^2+6*(sg2stressi(5).^2+sg2stressi(6).^2+...
52.           sg2stressi(7).^2)));
53.
54.       ycon3=allcoords(find(allcoords(:,3)>=gageloc(3,2)-tols(3,2)...
55.           & allcoords(:,3)<=gageloc(3,2)+tols(3,2)),:);
56.       xcon3=ycon3(find(ycon3(:,2)>=gageloc(3,1)-tols(3,1)...
57.           & ycon3(:,2)<=gageloc(3,1)+tols(3,1)),:);
58.       % zcon3=xcon3(find(xcon3(:,4)>=gageloc(3,3)-tols(3,3)...
59.       %     & xcon3(:,4)<=gageloc(3,3)+tols(3,3)),:);
60.       sg3node=xcon3(1,1);
61.       sg3i=find(matstressallu(:,1)==sg3node);
62.       sg3stressr=matstressallu(sg3i,:)*ampr*asf;
63.       sg3stressi=matstressallu(sg3i,:)*ampi*asf;
64.       sigmaVM3real=sqrt((1/2)*((sg3stressr(2)-sg3stressr(3)).^2+...
65.           (sg3stressr(3)-sg3stressr(4)).^2+(sg3stressr(4)-...
66.           sg3stressr(2)).^2+6*(sg3stressr(5).^2+sg3stressr(6).^2+...
67.           sg3stressr(7).^2)));
68.       sigmaVM3imag=sqrt((1/2)*((sg3stressi(2)-sg3stressi(3)).^2+...
69.           (sg3stressi(3)-sg3stressi(4)).^2+(sg3stressi(4)-...
70.           sg3stressi(2)).^2+6*(sg3stressi(5).^2+sg3stressi(6).^2+...
71.           sg3stressi(7).^2)));
72.
73.       ycon4=allcoords(find(allcoords(:,3)>=gageloc(4,2)-tols(4,2)...
74.           & allcoords(:,3)<=gageloc(4,2)+tols(4,2)),:);
75.       xcon4=ycon4(find(ycon4(:,2)>=gageloc(4,1)-tols(4,1)...
76.           & ycon4(:,2)<=gageloc(4,1)+tols(4,1)),:);
77.       % zcon1=xcon1(find(xcon1(:,4)>=gageloc(1,3)-tols(1,3)...
78.       %     & xcon1(:,4)<=gageloc(1,3)+tols(1,3)),:);
79.       sg4node=xcon4(1,1);
80.       sg4i=find(matstressallu(:,1)==sg4node);
81.       sg4stressr=matstressallu(sg4i,:)*ampr*asf;
82.       sg4stressi=matstressallu(sg4i,:)*ampi*asf;
83.       sigmaVM4real=sqrt((1/2)*((sg4stressr(2)-sg4stressr(3)).^2+...
84.           (sg4stressr(3)-sg4stressr(4)).^2+(sg4stressr(4)-...
85.           sg4stressr(2)).^2+6*(sg4stressr(5).^2+sg4stressr(6).^2+...
86.           sg4stressr(7).^2)));
87.       sigmaVM4imag=sqrt((1/2)*((sg4stressi(2)-sg4stressi(3)).^2+...
88.           (sg4stressi(3)-sg4stressi(4)).^2+(sg4stressi(4)-...
89.           sg4stressi(2)).^2+6*(sg4stressi(5).^2+sg4stressi(6).^2+...
90.           sg4stressi(7).^2)));
91.       IDnodes=[allcoords(find(allcoords(:,1)==sg1node),:)...
92.           sg1stressr(:,2:end) sigmaVM1real sg1stressi(:,2:end)...
93.           sigmaVM1imag;allcoords(find(allcoords(:,1)==sg2node),:)...
94.           sg2stressr(:,2:end) sigmaVM2real sg2stressi(:,2:end)...
95.           sigmaVM2imag;allcoords(find(allcoords(:,1)==sg3node),:)...
96.           sg3stressr(:,2:end) sigmaVM3real sg3stressi(:,2:end)...
97.           sigmaVM3imag;allcoords(find(allcoords(:,1)==sg4node),:)...
98.           sg4stressr(:,2:end) sigmaVM4real sg4stressi(:,2:end)
   sigmaVM4imag];
```

```
99.       SGinfo{p,i,j}=IDnodes;
100.  elseif specgageloc==0   %gage nodes specified
101.      for k=1:size(gagenode,1)
102.      sgnode=allcoords(find(allcoords(:,1)==gagenode(k)),:);
103.      sgnodecoords{k}=sgnode;
104.
   sgstressr=matstressallu(find(matstressallu(:,1)==gagenode(k)),:)*...
105.          ampr*asf;
106.
   sgstressi=matstressallu(find(matstressallu(:,1)==gagenode(k)),:)*...
107.          ampi*asf;
108.      sgstressreal{k}=sgstressr;
109.      sgstressimag{k}=sgstressi;
110.      sigmaVMr=sqrt((1/2)*((sgstressr(2)-sgstressr(3)).^2+...
111.          (sgstressr(3)-sgstressr(4)).^2+(sgstressr(4)-...
112.          sgstressr(2)).^2+6*(sgstressr(5).^2+sgstressr(6).^2+...
113.          sgstressr(7).^2)));
114.      sigmaVMi=sqrt((1/2)*((sgstressi(2)-sgstressi(3)).^2+...
115.          (sgstressi(3)-sgstressi(4)).^2+(sgstressi(4)-...
116.          sgstressi(2)).^2+6*(sgstressi(5).^2+sgstressi(6).^2+...
117.          sgstressi(7).^2)));
118.      sigmaVMreal{k}=sigmaVMr;
119.      sigmaVMimag{k}=sigmaVMi;
120.      IDnodes{k}=[sgnode sgstressr(:,2:end) sigmaVMr...
121.          sgstressi(:,2:end) sigmaVMi];
122.      end
123.      IDnodes=IDnodes.';
124.      IDnodes=cell2mat(IDnodes);
125.      SGinfo{p,i,j}=IDnodes;
126.  end
127.  end
```

Source code for r24mean.m:

```
1. function [meanstress] = r24mean(asciichars,...
2.     workdir,casepcs,fsclist)
3. file=[workdir,'R24_',num2str(casepcs),...
4.     'pc_static_only_meanstress-component-stress-list'];
5. fprintf('Stress File is in %s\n',file);
6. meanstress=importdata(file,'r');
7. idx=find(contains(meanstress,'MATERIAL  15'));
8. idx2=find(contains(meanstress,'MATERIAL  16'));
9. idx3=find(contains(meanstress,'MINIMUM'));
10.   mat15=meanstress(idx(1):idx2(1));
11.   mat15(contains(mat15,asciichars))=[];
12.   mat15=deblank(mat15);
13.   mat15(cellfun('isempty',mat15))=[];
14.   % %Fixed Width Delmit Approach (may change each configuration)
15.   % mat15=char(mat15);
16.   % sn15=str2num(mat15(:,1:8));
17.   % sx15=str2num(mat15(:,9:21));
18.   % sy15=str2num(mat15(:,22:33));
```

```
% sz15=str2num(mat15(:,34:45));
% sxy15=str2num(mat15(:,46:57));
% syz15=str2num(mat15(:,58:69));
% sxz15=str2num(mat15(:,70:end));
%Array Split Approach (assumes white space)
mat15=split(mat15);
mat15(:,1)=[];
sn15=str2double(mat15(:,1));
sx15=str2double(mat15(:,2));
sy15=str2double(mat15(:,3));
sz15=str2double(mat15(:,4));
sxy15=str2double(mat15(:,5));
syz15=str2double(mat15(:,6));
sxz15=str2double(mat15(:,7));
sall15=[sn15 sx15 sy15 sz15 sxy15 syz15 sxz15];
mat16=meanstress(idx2(1):idx3(1));
mat16(contains(mat16,asciichars))=[];
mat16=deblank(mat16);
mat16(cellfun('isempty',mat16))=[];
% % %Fixed Width Delmit Approach (may change each configuration)
% mat16=char(mat16);
% sn16=str2num(mat16(:,1:8));
% sx16=str2num(mat16(:,9:21));
% sy16=str2num(mat16(:,22:33));
% sz16=str2num(mat16(:,34:45));
% sxy16=str2num(mat16(:,46:57));
% syz16=str2num(mat16(:,58:69));
% sxz16=str2num(mat16(:,70:end));
%Array Split Approach (assumes white space)
mat16=split(mat16);
mat16(:,1)=[];
sn16=str2double(mat16(:,1));
sx16=str2double(mat16(:,2));
sy16=str2double(mat16(:,3));
sz16=str2double(mat16(:,4));
sxy16=str2double(mat16(:,5));
syz16=str2double(mat16(:,6));
sxz16=str2double(mat16(:,7));
sall16=[sn16 sx16 sy16 sz16 sxy16 syz16 sxz16];
shared15=find(ismember(sall15(:,1),sall16(:,1),'rows'));
shared16=find(ismember(sall16(:,1),sall15(:,1),'rows'));
s15fil=sall15(shared15,:);
s16fil=sall16(shared16,:);
ssxavg=mean([s15fil(:,2) s16fil(:,2)],2);
ssyavg=mean([s15fil(:,3) s16fil(:,3)],2);
sszavg=mean([s15fil(:,4) s16fil(:,4)],2);
ssxyavg=mean([s15fil(:,5) s16fil(:,5)],2);
ssyzavg=mean([s15fil(:,6) s16fil(:,6)],2);
ssxzavg=mean([s15fil(:,7) s16fil(:,7)],2);
ssall=[s15fil(:,1) ssxavg ssyavg sszavg ssxyavg ssyzavg ssxzavg];
s15ri=find(ismember(sall15(:,1),ssall(:,1),'rows'));
s16ri=find(ismember(sall16(:,1),ssall(:,1),'rows'));
```

```
71.   sall15(s15ri,:)=[];
72.   sall16(s16ri,:)=[];
73.   meanstressallu=[sall15; sall16; ssall];
74.   VMmean=sqrt((1/2)*((meanstressallu(:,2)-meanstressallu(:,3)).^2+...
75.       (meanstressallu(:,3)-meanstressallu(:,4)).^2+...
76.       (meanstressallu(:,4)-meanstressallu(:,2)).^2+...
77.       6.*(meanstressallu(:,5).^2+meanstressallu(:,6).^2+...
78.       meanstressallu(:,7).^2)));
79.   meanu=[meanstressallu(:,1) VMmean];
80.   meanstress=sortrows(meanu,'ascend');
81.   altmeansort=find(ismember(fsclist(:,1),meanstress(:,1),'rows'));
82.   meanstress=meanstress(altmeansort,:);
83.   end
```

Source code for modesum.m:

```
1. function [msumVM] = modesum(p,i,j,...
2.     stressrealout,stressimagout,blade,ND,mode)
3. % Function to include various modal stress contributions to Goodman
4. % Previously performed with commands
5. % Import complex stresses from main loop, extract each set, sum complex
6. % contributions, VM calc out for Goodman, full surface model no gages
7. for pn=blade(1):p
8.     for in=ND(1):i
9.         for jn=mode(1):j
10.              modestressreal{jn}=stressrealout{pn,in,jn}{jn};
11.              modestressimag{jn}=stressimagout{pn,in,jn}{jn};
12.          end
13.       end
14.   end
15.   for k=mode(1):jn
16.       sxxrealall{k}=modestressreal{k}(:,2);
17.       syyrealall{k}=modestressreal{k}(:,3);
18.       szzrealall{k}=modestressreal{k}(:,4);
19.       sxyrealall{k}=modestressreal{k}(:,5);
20.       syzrealall{k}=modestressreal{k}(:,6);
21.       sxzrealall{k}=modestressreal{k}(:,7);
22.       sxximagall{k}=modestressimag{k}(:,2);
23.       syyimagall{k}=modestressimag{k}(:,3);
24.       szzimagall{k}=modestressimag{k}(:,4);
25.       sxyimagall{k}=modestressimag{k}(:,5);
26.       syzimagall{k}=modestressimag{k}(:,6);
27.       sxzimagall{k}=modestressimag{k}(:,7);
28.   end
29.   sxxreal=sum(cell2mat(sxxrealall),2);
30.   syyreal=sum(cell2mat(syyrealall),2);
31.   szzreal=sum(cell2mat(szzrealall),2);
32.   sxyreal=sum(cell2mat(sxyrealall),2);
33.   syzreal=sum(cell2mat(syzrealall),2);
34.   sxzreal=sum(cell2mat(sxzrealall),2);
35.   sxximag=sum(cell2mat(sxximagall),2);
36.   syyimag=sum(cell2mat(syyimagall),2);
```

```
   szzimag=sum(cell2mat(szzimagall),2);
   sxyimag=sum(cell2mat(sxyimagall),2);
   syzimag=sum(cell2mat(syzimagall),2);
   sxzimag=sum(cell2mat(sxzimagall),2);
   msumreal=...

  [stressrealout{blade(1),ND(1),mode(1)}{blade(1),ND(1),mode(1)}(:,1)...
       sxxreal syyreal szzreal sxyreal syzreal sxzreal];
   msumimag=...

  [stressimagout{blade(1),ND(1),mode(1)}{blade(1),ND(1),mode(1)}(:,1)...
       sxximag syyimag szzimag sxyimag syzimag sxzimag];
   msummag=[sqrt(msumreal(:,2).^2+msumimag(:,2).^2) ...
       sqrt(msumreal(:,3).^2+msumimag(:,3).^2) ...
       sqrt(msumreal(:,4).^2+msumimag(:,4).^2) ...
       sqrt(msumreal(:,5).^2+msumimag(:,5).^2) ...
       sqrt(msumreal(:,6).^2+msumimag(:,6).^2) ...
       sqrt(msumreal(:,7).^2+msumimag(:,7).^2)];
   msumVM=sqrt((1/2)*((msummag(:,1)-msummag(:,2)).^2+(msummag(:,2)-...
       msummag(:,3)).^2+(msummag(:,3)-msummag(:,1)).^2+...
       6*(msummag(:,4).^2+msummag(:,5).^2+msummag(:,6).^2)));
   end
```

Source code for modesumg.m:

```
function [msumVM,msumVMgages] = modesumg(p,i,j,...
    stressrealout,stressimagout,SGinfo,blade,ND,mode)
% Function to include various modal stress contributions to Goodman
% Previously performed with commands
% Import complex stresses from main loop, extract each set, sum complex
% contributions, VM calc out for Goodman, full surface model and gages
for pn=blade(1):p
    for in=ND(1):i
        for jn=mode(1):j
              modestressreal{jn}=stressrealout{pn,in,jn}{jn};
              modestressimag{jn}=stressimagout{pn,in,jn}{jn};
              gagestressall{jn}=SGinfo{pn,in,jn}{jn};
          end
       end
   end
   for k=mode(1):jn
       sxxrealall{k}=modestressreal{k}(:,2);
       syyrealall{k}=modestressreal{k}(:,3);
       szzrealall{k}=modestressreal{k}(:,4);
       sxyrealall{k}=modestressreal{k}(:,5);
       syzrealall{k}=modestressreal{k}(:,6);
       sxzrealall{k}=modestressreal{k}(:,7);
       sxximagall{k}=modestressimag{k}(:,2);
       syyimagall{k}=modestressimag{k}(:,3);
       szzimagall{k}=modestressimag{k}(:,4);
       sxyimagall{k}=modestressimag{k}(:,5);
       syzimagall{k}=modestressimag{k}(:,6);
```

```
28.       sxzimagall{k}=modestressimag{k}(:,7);
29.       sxxrealgages{k}=gagestressall{k}(:,5);
30.       syyrealgages{k}=gagestressall{k}(:,6);
31.       szzrealgages{k}=gagestressall{k}(:,7);
32.       sxyrealgages{k}=gagestressall{k}(:,8);
33.       syzrealgages{k}=gagestressall{k}(:,9);
34.       sxzrealgages{k}=gagestressall{k}(:,10);
35.       sxximaggages{k}=gagestressall{k}(:,12);
36.       syyimaggages{k}=gagestressall{k}(:,13);
37.       szzimaggages{k}=gagestressall{k}(:,14);
38.       sxyimaggages{k}=gagestressall{k}(:,15);
39.       syzimaggages{k}=gagestressall{k}(:,16);
40.       sxzimaggages{k}=gagestressall{k}(:,17);
41.   end
42.   sxxreal=sum(cell2mat(sxxrealall),2);
43.   syyreal=sum(cell2mat(syyrealall),2);
44.   szzreal=sum(cell2mat(szzrealall),2);
45.   sxyreal=sum(cell2mat(sxyrealall),2);
46.   syzreal=sum(cell2mat(syzrealall),2);
47.   sxzreal=sum(cell2mat(sxzrealall),2);
48.   sxximag=sum(cell2mat(sxximagall),2);
49.   syyimag=sum(cell2mat(syyimagall),2);
50.   szzimag=sum(cell2mat(szzimagall),2);
51.   sxyimag=sum(cell2mat(sxyimagall),2);
52.   syzimag=sum(cell2mat(syzimagall),2);
53.   sxzimag=sum(cell2mat(sxzimagall),2);
54.   msumreal=...
55.
   [stressrealout{blade(1),ND(1),mode(1)}{blade(1),ND(1),mode(1)}(:,1)...
56.       sxxreal syyreal szzreal sxyreal syzreal sxzreal];
57.   msumimag=...
58.
   [stressimagout{blade(1),ND(1),mode(1)}{blade(1),ND(1),mode(1)}(:,1)...
59.       sxximag syyimag szzimag sxyimag syzimag sxzimag];
60.   msummag=[sqrt(msumreal(:,2).^2+msumimag(:,2).^2) ...
61.       sqrt(msumreal(:,3).^2+msumimag(:,3).^2) ...
62.       sqrt(msumreal(:,4).^2+msumimag(:,4).^2) ...
63.       sqrt(msumreal(:,5).^2+msumimag(:,5).^2) ...
64.       sqrt(msumreal(:,6).^2+msumimag(:,6).^2) ...
65.       sqrt(msumreal(:,7).^2+msumimag(:,7).^2)];
66.   msumVM=sqrt((1/2)*((msummag(:,1)-msummag(:,2)).^2+...
67.       (msummag(:,2)-msummag(:,3)).^2+(msummag(:,3)-msummag(:,1)).^2+...
68.       6*(msummag(:,4).^2+msummag(:,5).^2+msummag(:,6).^2)));
69.   sxxrealgage=sum(cell2mat(sxxrealgages),2);
70.   syyrealgage=sum(cell2mat(syyrealgages),2);
71.   szzrealgage=sum(cell2mat(szzrealgages),2);
72.   sxyrealgage=sum(cell2mat(sxyrealgages),2);
73.   syzrealgage=sum(cell2mat(syzrealgages),2);
74.   sxzrealgage=sum(cell2mat(sxzrealgages),2);
75.   sxximaggage=sum(cell2mat(sxximaggages),2);
76.   syyimaggage=sum(cell2mat(syyimaggages),2);
77.   szzimaggage=sum(cell2mat(szzimaggages),2);
```

```
78.   sxyimaggage=sum(cell2mat(sxyimaggages),2);
79.   syzimaggage=sum(cell2mat(syzimaggages),2);
80.   sxzimaggage=sum(cell2mat(sxzimaggages),2);
81.   msumrealgages=...
82.       [SGinfo{blade(1),ND(1),mode(1)}{blade(1),ND(1),mode(1)}(:,1)...
83.       sxxrealgage syyrealgage szzrealgage sxyrealgage syzrealgage...
84.       sxzrealgage];
85.   msumimaggages=...
86.       [SGinfo{blade(1),ND(1),mode(1)}{blade(1),ND(1),mode(1)}(:,1)...
87.       sxximaggage syyimaggage szzimaggage sxyimaggage syzimaggage...
88.       sxzimaggage];
89.   msummaggages=[sqrt(msumrealgages(:,2).^2+msumimaggages(:,2).^2) ...
90.       sqrt(msumrealgages(:,3).^2+msumimaggages(:,3).^2) ...
91.       sqrt(msumrealgages(:,4).^2+msumimaggages(:,4).^2) ...
92.       sqrt(msumrealgages(:,5).^2+msumimaggages(:,5).^2) ...
93.       sqrt(msumrealgages(:,6).^2+msumimaggages(:,6).^2) ...
94.       sqrt(msumrealgages(:,7).^2+msumimaggages(:,7).^2)];
95.   msumVMgages=sqrt((1/2)*((msummaggages(:,1)-msummaggages(:,2)).^2+...
96.       (msummaggages(:,2)-msummaggages(:,3)).^2+(msummaggages(:,3)-...
97.       msummaggages(:,1)).^2+6*(msummaggages(:,4).^2+...
98.       msummaggages(:,5).^2+msummaggages(:,6).^2)));
99.   end
```

References

1. Chen, Jen Ping; and Whitfield, David L.: Navier-Stokes Calculations for the Unsteady Flowfield of Turbomachinery. AIAA 1993–0676, 1993. https://arc.aiaa.org/doi/epdf/10.2514/6.1993-676 Accessed May 1, 2024.
2. Chen, J.-P.; and Briley, W.R.: A Parallel Flow Solver for Unsteady Multiple Blade Row Turbomachinery Simulations. ASME GT2001–0348, 2001. https://doi.org/10.1115/2001-GT-0348 Accessed May 1, 2024.
3. Bakhle, Milind A., et al.: Aeromechanics Analysis of a Distortion-Tolerant Fan With Boundary Layer Ingestion. AIAA SciTech Forum, Kissimmee, FL, 2018. https://arc.aiaa.org/doi/epdf/10.2514/6.2018-1891 Accessed May 1, 2024.
4. Min, J.B., et al.: Cyclic Symmetry Finite Element Forced Response Analysis of a Distortion-Tolerant Fan With Boundary Layer Ingestion. AIAA SciTech Forum, Kissimmee, FL, 2018. https://arc.aiaa.org/doi/epdf/10.2514/6.2018-1890 Accessed May 1, 2024.
5. Schemmel, Austin J.: Forced Response Aeromechanics Analysis in MATLAB®-Based Environment Code with Application to Distortion-Tolerant Fan R24 Blade Geometry. NASA/TM—20240000075, 2024. http://ntrs.nasa.gov