IMAT

Interface between MATLAB, Analysis, and Test

IMAT™, the Interface between MATLAB, Analysis, and Test, is a toolbox that allows MATLAB to exchange data with a variety of software including I-deas, Nastran, FEMAP, and Abaqus, as well as other software packages. It provides a powerful means for manipulating test and analysis data in MATLAB while preserving data attributes. IMAT's capabilities are extended even further through several extensions that are grouped according to the type of functionality they contain. All of the capabilities and their documentation are included in the IMAT package, but the extended functionality requires an additional license. Please contact us for more information.

IMAT+FEA allows the user to interface with several different analysis packages including Nastran, FEMAP, and Abaqus. In most cases bi-directional support is provided. IMAT+FEA functions are marked with a **`+FEA`** next to the documentation. You can read more about IMAT+FEA here.

IMAT+Modal provides functionality geared toward the modal test engineer. IMAT+Modal provides capabilities for generating Test Analysis Models (TAMs) in Nastran, optimal accelerometer selection using a genetic algorithm, and many routines for processing modal test data. IMAT+Modal functions are marked with a **`+Modal`** next to the documentation. You can read more about IMAT+Modal here.

IMAT+Signal extends IMAT by providing functionality that facilitates signal processing, both with stationary and rotating signals. In addition to command-line functions, SPFRF is a GUI that provide an easy-to-use interface for processing stationary signals. RTK is a GUI that encapsulates the process of obtaining response characteristics and finding insights into the mechanics of rotating events. It applies the Vold-Kalman filter technique to order-tracking of rotational events. IMAT+Signal functions are marked with a **`+Signal`** next to the documentation. You can read more about IMAT+Signal here.

Sources of Help for the IMAT Toolbox

In MATLAB, you can type "`help *function*`" on any IMAT function to get a description of the syntax and usage of that function. If you type "`help imat`", you will get a list of most available functions.

Please review the Compatibility Considerations for any compatibility changes between releases.

Installation Guide

If you have not yet installed the IMAT toolbox, start here for detailed installation instructions.

User's Guide

This document gives an overview of the IMAT toolbox, including IMAT+FEA, IMAT+Modal, and IMAT+Signal.

Reference Guide

All IMAT toolbox functions are described in more detail in this reference. You can look for functions either by subject groupings or alphabetically.

Data Attribute Reference

If you need information on the data attributes of functions, mode shapes, results, or FEM data, look in these references.

- Data attributes of functions
- Data attributes of mode shapes
- Data attributes of results
- Data structure for FEM data

Bug Reports and Enhancement Requests

Please visit our issue tracking database at tier.ata-e.com. You can search through the existing issue database to see if your issue has been previously reported, and if a solution is available. This is the best way to communicate to us any problems you find in the software.

Support Hotline

You can call us to get phone support. Please visit our website for more information.

IMAT Information

The revision history lists the enhancements and bug fixes for each IMAT release to date.

The FAQ lists some commonly asked questions regarding the IMAT toolbox.

For more information on IMAT, and to check for the latest available version, please visit the IMAT web site at www.ata-e.com/software/imat. You may also email us at imat AT ata-e DOT com.

Known Limitations of the IMAT Toolbox

Please see the README file in the IMAT distribution for platform-specific issues.

IMAT Revision History

The following lists provide a history of changes to IMAT.

v7.0.0

New Features/Enhancements

- Designed to work with MATLAB R2017b. Also works with R2015a, R2015b, R2016a, R2016b, and R2017a.
- femap_invoke now supports FEMAP 11.4.x. It also supports FEMAP 11.3.x.
- Document the FCN data type.
- *examples* directory additions
  - Added function calc_energy_fractions that calculates strain and/or kinetic energy fractions for supplied groups
  - Added function create_default_fn to create default functions of select types, setting the required properties for each of the supported types
  - Added function writerandps to write Nastran bulk data RANDPS and corresponding TABRND1 cards from supplied PSDs and CSDs
  - Added function writetabled1 to write functions to Nastran bulk data TABLED1 cards
  - Modify writempc to allow the user to specify the MPC ID
- readnas enhancements
  - Add support for `'entityids'` argument with SORT2 PCH results
  - Add support for OEFIIP and OEFIIS datablocks
  - Change OEFIT output to be Data At Nodes On Elements rather than Data On Elements
  - New function parseop2table_raw2nas to convert readnas `'asraw'` results format to a usable Nastran-centric format
  - Add support for OEE datablocks (ONRGY1 and ONRGY2) to create_op2table
  - Modify PCH file output for non-nodal results to be closer to the PCH file contents: real/imaginary (or magnitude/phase) item codes are not combined into a single complex item, but are left in the original format. Please see the Compatibility Considerations for more details.
  - Add `.datachar` field to PCH file structure output for non-nodal results, containing a cell array of strings of the string-based items. Please see the Compatibility Considerations for more details.
- writenas enhancements
  - Modify TAPELABL contents so that Nastran will read OP2 files generated by writenas
- imat_fn enhancements
  - Add optional input argument to csd and psd to allow you to specify the overlap percentage
  - Add optional input argument `'maximax'` to psd to compute peak-hold, performing an Nth octave reduction before enveloping
  - Add ability to set the legend interpreter with imat_fnplot/legend and in the context menu
  - writecsv can now optionally append to a file
- imat_shp enhancements
  - writecsv can now optionally append to a file
- imat_vtkplot enhancements
  - labelnodes and labelelements can now set the label size using optional input arguments
- writefemap enhancements
  - Add explicit support for exporting CBAR and CROD forces, stresses, and strains

Bug Fixes

- Minor documentation and bug fixes
- Fix bug with `'cycleshapes'` input argument to imat_shp/plot

- Fix complex animation in imat_shp/vtkplot
- Fix TIER reports 1471, 1993, 2103, 2264, 2273, 2323, 2426, 2468, 2495, 2506, 2509, 2545, 2581, 2592, 2593, 2594, and 2596

v6.3.0

New Features/Enhancements

- Designed to work with MATLAB R2017a. Also works with R2015a, R2015b, R2016a, and R2016b.
- imat_getfile and imat_putfile now support additional optional arguments to allow more of the capabilities of `uigetfile` and `uiputfile`
- imat_ctrace enhancements
  - vtkplot now accepts a vector of colors as input
- imat_shp enhancements
  - parse_shape has been renamed to partition to be consistent with other data types
  - list now has an optional output argument that returns the shape coefficients listed
- imat_result enhancements
  - list now has an optional output argument that returns the result components and data listed
- imat_vtkplot enhancements
  - export now allows you to specify an AVI filename, and will export an animation, just like the `Export Animation` menu
  - `Export All` in the `Export` menu now also supports AVI file export
- readnas enhancements
  - Add support for NX Nastran 10 ply stress/strain results including OESPSD2C, OESRMS1C, and OESN01C.
- readodb enhancements
  - Support Abaqus 2016 on Windows
- uiplot enhancements
  - Improved performance of initial startup
  - Add support for UNV export

Bug Fixes

- Minor documentation and bug fixes
- Fix bug where the sign on some centroidal force results imported from OP2 were flipped when using the `opt.global.femap` option.
- Fix bug in imat_result/partition with an imat_group where the partitioning for Data At Nodes On Elements and Data On Elements At Nodes results were not being processed correctly. The behavior has been modified so that if the group contains both nodes and elements, the result is partitioned to the components whose node and element matches the group entities.
- Fix TIER reports 1813, 2271, 2312, 2373, 2377, and 2445

v6.2.1

New Features/Enhancements

- imat_fnplot enhancements
  - saveas now support SVG format
- readnas enhancements
  - DMIG matrices marked as symmetric that have all-zero (therefore missing) columns will be expanded to be square

Bug Fixes

- Minor documentation and bug fixes
- Fix bug in readnas where OESVM and any SORT2 results from PCH files written in magnitude/phase format were imported incorrectly
- Fix bug in readneu where it failed to read elements (dataset 404) from post-FEMAP 11.2-versioned Neutral files
- Fix TIER reports 2260, 2274, and 2275

v6.2.0

New Features/Enhancements

- Designed to work with MATLAB R2016b. Also works with R2014b, R2015a, R2015b, and R2016a.
- This version of IMAT requires Sentinel RMS license server version 9.1 or newer.
- femap_invoke now supports FEMAP 11.3.x. It is not compatible with earlier versions of FEMAP.
- imat_ctrace enhancements
  - Improve performance of intersect, ismember, setdiff, setxor, union, and unique
- imat_filt enhancements
  - Expand imat_filt to also work with imat_shp
- imat_fn enhancements
  - imat_fnplot/legend now lets you set the legend font attributes
- readnas enhancements
  - Add support for OESVM1* and OESXRMS* datablocks
  - Add support for `'entityids'` flag to OP2 SORT2 results
- readodb enhancements
  - Import heat transfer analysis fields CFL11, CFL12, CFL13, CFL14, CFL15, HFL, HFLA, HTL, HTLA, NT11, NT12, NT13, NT14, and NT15
  - Import fields U_ANTIALIASED, UT_ANTIALIASED, V_ANTIALIASED, VT_ANTIALIASED, A_ANTIALIASED, AT_ANTIALIASED
- readneu enhancements
  - Add support for importing tracelines

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 2095, 2101, 2107, 2118, 2125, 2134, 2139, 2144, 2165, 2171, 2203, 2205, and 2222 and 2235

v6.1.0

New Features/Enhancements

- Designed to work with MATLAB R2016a. Also works with R2012b, R2013a, R2013b, R2014a, R2014b, R2015a, and R2015b.
- **IMAT no longer supports 32-bit Windows platforms because Mathworks has dropped support for Win32 starting with MATLAB R2016a.**
- imat_fem enhancements
  - New method get_id_by_nid to return element or traceline IDs associated with the supplied node IDs
- imat_fn enhancements
  - plot now allows you to set the font in the legend from the context menu

- imat_result enhancements
    - New partition methods to partition a result to a specified component list
    - You can now set the `data.component` matrix directly, rather than having to use setComponents
- New example function writeblk to export an imat_fem to Nastran BLK format
- New example function writedmig to export a matrix to Nastran DMIG format
- New example functions writempc and write_nas_field to export a matrix to NASTRAN MPC cards

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 1893, 1894, 1919, 1930, 1964, 1992, 1995, 2036, 2041, 2072, and 2095

v6.0.1

New Features/Enhancements

- Minor update to startimat

Bug Fixes

- Fix errors plotting coordinate traces with imat_ctrace/plot and imat_ctrace/vtkplot
- Fix TIER report 1948

v6.0.0

New Features/Enhancements

- Designed to work with MATLAB R2015b. Also works with R2012b, R2013a, R2013b, R2014a, R2014b, and R2015a.
- **The global Cartesian, or basic, coordinate system ID has been changed from 1 to 0 to be consistent with Nastran. This has many implications, so please read Compatibility Considerations carefully for a full description of the changes and their implications.**
- **The IMAT element types have been changed from I-deas-centric numbering to Nastran-centric numbering.**
- New helper function startimat to help new users learn some of the basics of IMAT
- imat_fn enhancements
    - writecsv writes more digits of precision for GPS data
- readnas enhancements
    - Element types are now returned using Nastran-centric numbering
    - New input argument `'entityids'` followed by a vector of entity ids causes readnas to partition OP2 and PCH results to the IDs specified in the vector
    - Add support for CQUADX, CPLSTN3, CPLSTN4, CPLSTN6, and CPLSTN8 elements
    - Add support for PLPLANE and PPLANE physical properties
    - Add stress/strain processing for CTRAX3, CQUADX4, CTRAX6, and CQUADX8 linear analysis
    - Add force processing for CDAMP1, CDAMP2, CDAMP3, and CDAMP4 elements
    - Removed obsolete options `'records'` and `'PCH_FCN'`
    - When transforming results to the basic coordinate system on import, return stress resultants in the basic coordinate system instead of the material coordinate system to be consistent with stress, strain, and force results.

- imat_result enhancements
    - New method group_by to group results by the values of a specified attribute
- Add capability to specify the center of rotation for the rotational rigid body modes created by rbmodes
- Removed obsolete functions plotfem, fem_cat, fem_labelnodes, fem_part, fem_valid, imat_shp/list_shape

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 743, 1701, 1722, 1753, 1777, 1796, 1811, and 1853

v5.1.0

New Features/Enhancements

- Designed to work with MATLAB R2015a. Also works with R2012b, R2013a, R2013b, R2014a, and R2014b.
- New functions get_octave_bands and count_freqs_in_bands to work with 1/N octave-reduced data
- New functionality in the IMAT examples to create and/or modify FEM connectivity via Microsoft Excel spreadsheets
- New function readneu to read FEMAP Neutral files
- imat_fn enhancements
    - Performance improvements in validate
- imat_shp enhancements
    - Performance improvements in validate
- readunv enhancements
    - Major performance improvements when importing large numbers of dataset 58, and general performance improvements overall
- readnas enhancements
    - Add support for MPC cards
- readodb enhancements
    - Support Abaqus 6.14-1 on Windows

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 1455, 1563, 1567, 1572, 1575, 1587, 1588, 1596, 1597, 1662, and 1690.

v5.0.0

New Features/Enhancements

- Designed to work with MATLAB R2014b. Also works with R2011b, R2012a, R2012b, R2013a, R2013b, and R2014a.
- Supports the new handle graphics engine that is the new default in MATLAB R2014b.
- imat_fn enhancements
    - diag returns the diagonal of a matrix or generates a matrix from the diagonal
- readnas enhancements
    - Significant performance improvement when storing results into imat_result for large numbers of results
    - Return the parameter value table PVT from Output2 or bulk data in the field `.pvt`.

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 1440, 1454, 1506, 1514, 1511, 1522

v4.6.1

New Features/Enhancements

- femap_invoke now supports the FEMAP 11.x Results class
- readnas enhancements
    - Add support for CREEP

Bug Fixes

- Minor documentation and bug fixes
- Fix MATLAB hang when using VTKPLOT in MATLAB versions prior to R2013b (works around a bug in VTK)
- Fix VTKPLOT bug that caused tracelines to not be displayed
- Fix TIER reports 1477, 1478

v4.6.0

New Features/Enhancements

- Designed to work with MATLAB R2014a. Should also work with R2011b, R2012a, R2012b, R2013a, and R2013b.
- readodb now supports Abaqus 6.13-2.
- femap_invoke now supports FEMAP 11.1.x. It should also work with FEMAP 11.0.x.
- SMAC now supports the new CMIF reference tracking capability.
- Removed `result` and `ResultDataLocation` classes.
- readnas enhancements
    - Add support for CTRAX3, CTRAX6, CQUADX4, and CQUADX8
    - Add support for SPCs
    - Return properties for all cross-section stations on PBEAM cards
    - Return the property name from bulk data for all material and physical property cards (beam property cards return the fore end cross-section). Property names are extracted from the NX-formatted comment card located above the property card.
- imat_fn enhancements
    - Renamed `Reaction Force` DataType value to `Force`
    - cumrms now allows you to specify the interpolation type directly
- imat_shp enhancements
    - Renamed `Reaction Force` DataType value to `Force`
- imat_result enhancements
    - New method sort to sort result components
    - Math operations now check the component lists to make sure they match, and correctly handle the case where they match if they are sorted
- imat_fnplot enhancements
    - New method axis to modify axis properties

- imat_vtkplot enhancements
    - New method export to export the current VTK figure to a file
    - New method interactionstyle to set the "flavor" of user interaction (pan, zoom, rotate) with the graphics window. This functionality is also available through the command line and the View menu.
    - Significant new capabilities with regards to controlling the model display. Please see the help for interactionstyle for more details. The help is also available from a new Help menu on a VTKPLOT figure.

Bug Fixes

- Minor documentation and bug fixes
- Beam elements are now colored correctly in VTKPLOT when element borders are on
- readnas now reads all model geometry when reading superelements written to a PARAM,POST,-1 OP2 file (works around a bug in NX Nastran)
- readnas performance has been improved when reading in lots of temperature data from OP2
- Residual effects are now applied in SMAC
- VTKPLOT animation export has been fixed for multiple monitor displays
- readunv handles importing binary dataset 58 when the additional attributes are not written to the header line
- Fix TIER reports 961, 1010, 1356, 1369, 1385, 1396, 1410, 1412, 1414, 1416, 1417, 1418, 1421, 1422, 1426, 1439, 1441, 1443, 1445, 1450, 1452, 1460, 1464

v4.5.1

New Features/Enhancements

- imat_vtkplot enhancements
    - vtkplot now creates a log file if VTK was unable to initialize, to assist in troubleshooting
    - vtkplot supports both MPEG-4 and AVI (JPEG) output for animations
- imat_shp enhancements
    - Rename list_shape to list.

Bug Fixes

- imat_result enhancements
    - For complex modes, the Frequency attribute now returns the undamped natural frequency (in the previous release it returned the damped natural frequency), and ViscousDamping returns the correct viscous damping fraction
- Minor documentation and bug fixes
- Fix issue with VTK on some Windows systems being unable to initialize, even if the paths were set. This was due to Visual Studio 2012 redistributables not being installed on the target system.

v4.5.0

New Features/Enhancements

- Designed to work with MATLAB R2013b. Should also work with R2009b, R2010a, 2010b, R2011a, R2011b, R2012a, R2012b, and R2013a.
- femap_invoke now supports FEMAP 11.0.x.
- readnas enhancements
    - Support complex eigenvalue tables (CLAMA, CLAMAD, FLAMA, ULAMA)
    - Add support for SLOAD (GEOM3) from Output2 files

- imat_fn enhancements
    - Add support for NX ADF data types: `Gravitational Acceleration`, `Signal`, `Unitless Scalar`, `Unitless Real`, `Unitless Integer`, `Voltage`, `Electric Current`. Please note that IMAT does not support or handle Volts vs milliVolt or Amp vs milliAmp units.
- imat_shp enhancements
    - Add support for NX ADF data types: `Gravitational Acceleration`, `Signal`, `Unitless Scalar`, `Unitless Real`, `Unitless Integer`, `Voltage`, `Electric Current`. Please note that IMAT does not support or handle Volts vs milliVolt or Amp vs milliAmp units.
- imat_result enhancements
    - New invariant method that calculates invariants
    - New invariant components `'MagT'` and `'MagR'`
    - New methods mtimes and mrdivide that support multiplication and division (respectively) by a numeric scalar only
- imat_vtkplot enhancements
    - Add resultlocation method to set the result location for contour Data At Nodes On Elements results.

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 1262, 1339, 1353

v4.4.0

New Features/Enhancements

- Designed to work with MATLAB R2013a. Should also work with R2009b, R2010a, 2010b, R2011a, R2011b, R2012a, and R2012b.
- imat_ctrace enhancements
    - nd2ctrace now accepts a string or cell array of strings as input for the direction
    - Added `'legacy'` option for intersect, ismember, setdiff, setxor, union, and unique, to allow pre-R2013a behavior.
- imat_fn enhancements
    - plot now has a Window Previous/Next option to move through the user's windowing history (unlimited undo/redo)
    - New function writecsv to write functions to a CSV file
- imat_shp enhancements
    - New function writecsv to write shapes to a CSV file
    - build_shape, as well as subscripting with {} and imat_ctrace, treats blank directions (i.e. Nastran SPOINT) as X+.
- imat_fnplot enhancements
    - New method savefig to save a plot to a file, similar to the native MATLAB `.fig` file functionality
    - New method loadfig to load a plot from a file, similar to the native MATLAB `.fig` file functionality
- imat_elem enhancements
    - New method convertDegenerateToLinear to convert parabolic elements with fewer nodes than required for the full parabolic definition to their linear equivalent
- readodb enhancements
    - Support Abaqus 6.12-2 on Windows
- readnas enhancements
    - Return LOAD, FORCE, FORCE1, FORCE2, MOMENT, MOMENT1, MOMENT2, PLOAD, PLOAD1, PLOAD2, PLOAD4, TEMP and TEMPD loads in the `.loads` field of the output
    - Add support for SORT1 OPG datablocks (nodal applied force)

    - Changed the output of columns 5 and 6 of the Output2 directory .blocknames field
    - The `'records'` specifier is obsolete and has been replaced with `'modes'` and `'subcases'`
- imat_vtkplot enhancements
    - interactiveselector has a new input argument 'off' to turn off the selector externally
    - New method text to place arbitrary 2-D text on the plot. It returns a vtk_text object, which can be used to control the text properties.
    - view now supports setting arbitrary views by setting the azimuth and elevation

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 1156, 1184, 1213, 1224, 1225, 1246, 1255, 1263, 1273

v4.3.0

New Features/Enhancements

- Designed to work with MATLAB R2012b. Should also work with R2009b, R2010a, 2010b, R2011a, R2011b, and R2012a.
- Removed obsolete functions and methods: ideas_ctrace, ideas_filt, ideas_fn, ideas_shp, ideas_dir2num, ideas_getfile, ideas_num2dir, ideas_progress, ideas_putfile
- result has been deprecated and has been replaced by imat_result.
- Added generalized mode indicator function utility gmif.m to the *examples* directory.
- Add support for parabolic 2-D and 3-D elements to writefemap
- imat_fem enhancements
    - imat_elem node connectivity ordering for parabolic elements is now completely converted to using Nastran order, which specifies corner nodes first, followed by midside nodes. Now readunv returns parabolic elements in this connectivity order, and writeunv expects this order when exporting. This also affects plot, which in previous versions expected the I-deas ordering in which corner and midside nodes were interleaved.
    - plot and vtkplot now support plotting pyramid elements
    - plot supports a new input argument `'unitslabel'` to specify the units label to use with the axis label
    - New method imat_elem/elemtype_i2n to return Nastran element type(s) for corresponding IMAT element types
    - New method imat_elem/elemtype_n2i to return IMAT element type(s) for corresponding Nastran element types
- imat_vtkplot enhancements
    - colorrange now lets you set the colorbar visibility
    - New method setcontourcomponent that allows you to set the contour result display component on the plot.
    - New method text3 that allows you to place arbitrary text in 3D space.
    - New method measure to measure the distance between two nodes selected on the plot.
    - New method interactiveselector to select nodes or elements interactively and optionally call a callback function on each selection.
- imat_fn enhancements
    - fn2shp supports additional `'byval'` options
- imat_fnplot enhancements
    - New methods xscale and yscale to change the scales of the X and Y axes respectively
    - New method tag_style to set the plot tag style
- imat_shp enhancements
    - plot supports a new input argument `'unitslabel'` to specify the units label to use with the axis label

- imat_result enhancements
    - Add typecasting between result data locations. For example, it is now possible to convert Data On Elements results to Data At Nodes.
    - Add capability to subscript into the individual data locations using numeric and in some instances object-based subscripts, e.g. `r(1).data('1x')`
- license_commute now lets you specify the number of days for which to commute the license, and displays the number of days a commuted license has left until it expires
- readodb enhancements
    - Add option to merge instances into a single imat_result, rather than creating a separate result for each instance. The instance number is stored as the superelement ID in the result.
- readnas enhancements
    - Add support for reading case control (currently only SETs) from bulk data and Output2.
    - Read NX-formatted comment strings from case control and apply set names to sets imported from bulk data
    - Read NX-formatted comment strings for physical properties and return the contents to the `.name` field of each property's data
    - Add support for CPYRAM elements
    - Add support for MAT11 materials
    - Matrices imported using the `'asraw'` option will now import in a raw format compatible with writenas
- writenas enhancements
    - Export matrices that are in raw format

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 1165, 1166, 1169, 1173, 1181, 1182, 1192, 1193, 1194, 1197, 1202, 1206, 1210, 1212, 1214, 1219

v4.2.0

New Features/Enhancements

- Designed to work with MATLAB R2012a. Should also work with R2009b, R2010a, 2010b, R2011a, and R2011b.
- Discontinued support for 32-bit Linux platform
- New experimental imat_result data type, which will replace the result data type in the future. Read more about it here.
- New function writeop4 to export matrices to Nastran Output4 format
- imat_ctrace enhancements
    - Allow digits in the coordinate direction string
- imat_fn enhancements
    - min and max now have a second output containing the index of the minimum or maximum value, respectively
    - Enhance findpeaks to handle data with bitnoise, where there are regsions of constant-valued data (0 slope)
- imat_fnplot enhancements
    - legend now accepts %RMS as input. This will get converted to the RMS value and corresponding units string.
    - Added Pan capability to the context menu

- imat_vtkplot enhancements
  - New method getlabels that returns the IDs of the labeled entities (nodes or elements) from the deformed or undeformed VTK plot
  - New method backgroundcolor to set the background color of the plot
  - New method textcolor to set the color of text on the plot
- imat_shp enhancements
  - New attribute `.CSType` to store the coordinate system type in which the shape coefficients are defined (`'Displacement'`, `'Basic'`, or `'Other'`)
  - vtkplot now supports complex shape display and animation
  - plot now accepts the argument `'backgroundcolor'` to set the background color, and adds a menu item to do the same
  - plot now accepts the argument `'textcolor'` to set the title text and axis color, and adds a menu item to do the same
  - plot now accepts the argument `'noaxes'` to turn off the axis display, and adds a menu item to do the same
  - xform now uses the `.CSType` attribute to determine the From CS if it is not explicitly supplied, rather than looking in the `.cs` property of the FEM nodes
- imat_fem
  - plot now accepts the argument `'backgroundcolor'` to set the background color, and adds a menu item to do the same
  - plot now accepts the argument `'textcolor'` to set the title text and axis color, and adds a menu item to do the same
  - plot now accepts the argument `'noaxes'` to turn off the axis display, and adds a menu item to do the same
  - partition now accepts an imat_group as input
  - Curly-brace subscripting now calls partition when animat_group is supplied, and you can use multi-level subscripting, e.g.

    ```
    fem{group}.node(1:3).id
    ```
- readnas enhancements
  - **The default option for results is now to NOT transform to basic on import**
  - Add support for MATHP, MATS1, MATT1, MATT2, MATT3, MATT4, MATT5, MATT8, MATT9, and MATTC
  - Support OP2 files created with PARAM,OMACHPR,YES
  - Add support for TABLED1-4, TABLEM1-4, TABLEST, TABLES1, TABDMP1, and TABRND1 from both BLK and OP2. Support DIT and DITS datablocks in OP2. This support slightly changed the output structure for tables.
  - Import CONM1 and CONM2 mass and inertia properties as physical property data
  - If Nastran has marked a matrix as symmetric, return the fully populated matrix
  - If more than 20% of the terms in the matrix are non-zero, return a full matrix instead of a sparse matrix
  - Sort matrix DOF so that they are returned in increasing order
- readodb enhancements
  - For History output, set the ResponseNode to the node/element number where appropriate

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 915, 963, 1069, 1075, 1077, 1078, 1079, 1080, 1081, 1082, 1085, 1089, 1090, 1092, 1094, 1104, 1105, 1106, 1107, 1108, 1111, 1112, 1114, 1118, 1120, 1121, 1123, 1124, 1125, 1135, 1137, 1138, 1141, 1142, 1143, 1144, 1146, 1147, 1149, 1153, 1154, 1157

v4.1.0

New Features/Enhancements

- Designed to work with MATLAB R2011b. Should also work with R2008a, R2008b, R2009a, R2009b, R2010a, 2010b, and R2011a.
- This is the final release that will support 32-bit Linux, as Mathworks is phasing out support for this platform
- Created PDF version of the documentation
- create_op2table now also creates a LAMA table automatically under certain circumstances
- New example function format_ppt (located in the examples directory) that will modify an imat_fn/plot font and line formats for better pasting into PowerPoint
- readnas enhancements
    - No longer returns midside node stress/strain result for parabolic solids. These results are not available in the Output2 file, and were previously calculated by averaging the nodal results.
    - Added support for dataset OESNLXD: nonlinear CGAP force/deflection and nonlinear CROD stress/strain.
    - Added support for nonlinear CBUSH force/stress/strain dataset OESNLXR
    - Support for reading RBE2, RBE3, and RROD elements with ALPHA coefficients
    - The option OPT.PCH.SORT1TOSORT2 will be removed in a future release of IMAT
- readodb enhancements
    - Supports Abaqus 6.11-1 and older ODBs on Windows. On Linux, supports Abaqus 6.9-EF1 due to incompatibilities between the C++ compiler used by Abaqus and the compilers supported by MATLAB.
- imat_ctrace enhancements
    - New method nd2ctrace for building an imat_ctrace from a list of nodes and directions
    - New method vtkplot to display coordinate traces on a FEM using VTK
- imat_fn enhancements
    - plot now accepts an axis handle as input
    - Added a context menu item to plot that allows you to select a subplot on a figure and copy it to a new figure
    - New function ployy to plot two sets of functions on a double Y axis.
- imat_shp enhancements
    - plot now accepts an axis handle as input
- result enhancements
    - plot now accepts an axis handle as input
- imat_fem enhancements
    - New get method
    - New renumber method
    - New method vtkplotcs to display coordinate systems on the FEM. This display can also be enabled from vtkplot.
    - plot reverse axis capability now reverses two axes at a time to keep the triad consistent, and only allows one axis reversal setting at a time
    - Many enhancements to subscripting
    - Many performance improvements
- imat_vtkplot enhancements
    - New method csstyle for changing the coordinate system display style
- writefemap enhancements
    - Now supports exporting tracelines

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 462, 913, 954, 957, 962, 964, 969, 971, 972, 973, 974, 975, 985, 989, 990, 991, 992, 997, 999, 1005, 1010, 1011, 1013, 1016, 1017, 1020, 1021, 1022, 1023, 1027, 1028, 1029, 1035, 1037, 1039, 1047, 1049, 1050, 1058, 1059, 1063, 1065, 1068, 1072, 1074

v4.0.0

New Features/Enhancements

- Designed to work with MATLAB R2011a. Should also work with R2008a, R2008b, R2009a, R2009b, R2010a, and R2010b
- readnas enhancements
  - Support the Output4 (OP4) file format
  - Reverse the sign of the CELAS force output to match I-deas dataset 2414 and punch output
  - Changed PCH file blocknames to reflect their datatype (i.e. OUG1 for SORT1 OUG results)
  - Supports reading selected SORT1 records from the OP2 file using the `'records'` optional input
  - Handle item codes from random analysis
  - Add support for combining superelement results from PCH files
  - Support reading applied loads and SPC forces from GPFB datablocks
  - Add support for NX Response Simulation results file (*.rs2)
- readunv enhancements
  - Directly support group datasets (2452, 2467, 2477)
- writeunv enhancements
  - Support group export (2477)
- imat_fn enhancements
  - cumrms now has an optonal `'reverse'` input argument to compute the cumulative RMS in reverse order
  - uiplot has the ability to change the plot display units on the fly
  - octaven handles Auto Spectrum function types and more AmplitudeUnits/Normalization types
  - New IMAT+Signal method spl to compute sound pressure level.
  - New IMAT+Signal method acoustic_weighting to apply A or C weighting.
  - New function octaven2nb to convert octave-reduced functions to narrowband.
  - plot now returns an IMAT_FNPLOT object, which gives you significant control over how the plot is displayed.
  - The uiplot interface has been overhauled.
  - New option `'resetstart'` for truncate, which sets the abscissa minimum to 0.
  - Deprecated set_fnplot_style
  - Changed default AbscissaMin to 0.0 from 1.0.
- imat_shp enhancements
  - New method get_units_labels to return the units string based on the ordinate attributes
  - New experimental method vtkplot to display and animate shapes
- result enhancements
  - New method get_units_labels to return the units string based on the ResultType
  - New experimental method vtkplot to display and animate results
  - New method criterion to create criterion data
- New class imat_group for group entities
- New class imat_vtkplot for controlling displays generated by vtkplot. It is considered beta in this release of IMAT, and is described here.

- FEM enhancements
    - Overhaul FEM entities to replace structures with new classes: IMAT_FEM, IMAT_CS, IMAT_NODE, IMAT_ELEM, and IMAT_TL.
    - Renamed fem_cat to cat, fem_part to partition, fem_valid to validate, and plotfem to plot.
    - fem_create is now imat_fem.
    - New method imat_fem/plotcs to display coordinate systems.
    - New methods imat_cs/add, imat_node/add, imat_elem/add, imat_tl/add to add entities to the associated object
    - New methods imat_cs/keep, imat_node/keep, imat_elem/keep, imat_tl/keep to keep the specified entities inthe associated object
    - New methods imat_cs/remove, imat_node/remove, imat_elem/remove, imat_tl/remove to remove the specified entities from the associated object
    - New method imat_tl/to_2node to convert tracelines to 2-node tracelines
    - New experimental method imat_fem/vtkplot to display FEM entities
    - New method get_nid_from_id to return node IDs from the elements specified by the supplied IDs

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 782, 832, 842, 847, 852, 853, 855, 859, 865, 875, 876, 877, 883, 869, 897, 899, 909, 919, 938, 939, 941, 948, 952, 955, 965, 970, 978, 982

v3.2.0

New Features/Enhancements

- Designed to work with MATLAB R2010b. Should also work with R2008a, R2008b, R2009a, R2009b, and R2010a
- ideas_fn, ideas_shp, ideas_ctrace, and ideas_filt now automatically return the corresponding imat_* classes; the ideas_* classes have now been fully deprecated
- Licenses can now be commuted (checked out and taken off network) using the license_commute utility
- readnas enhancements
    - Output2 results in SORT2 format are now supported, and are returned in the `.functions` field as imat_fn.
    - Reduced import memory usage for element force, shell stress resultant, stress, strain, heat flux, nodal force, and grid point force balance results by approximately 1/3.
    - Add support for many more Output2 datablocks
    - Add support for MSC.Nastran 2008 and NX Nastran 7.1
    - Add toggle to combine or not combine superelement results (default is to combine). Also works with SOL200 results--combining returns just the last iteration, and not combining returns all iterations.
    - Supports importing bulk data directly from PCH file import (previously had to read bulk data from the PCH file with the `'isblk'` option)
    - Modified `'blocks'` input format for PCH files. Removed the requirement that the blocks specifier must be a 1x2 cell array starting with `'PCH'`. The `'PCH'` is no longer required, but the old format is still supported. See Compatibility Considerations for more details.
    - Allow the user to specify PCH file blocks to read by block name as well as numeric indices
    - Add support for SRS records from PCH file (tagged as PCHTABLED1). Better handling of PSD attributes.
- imat_fn enhancements
    - Add back the previously removed `LoadCase` attribute
    - power now accepts NUM.^IMAT_FN

    - rms now sums the ordinate values for evenly spaced frequency domain functions, instead of using the trapezoidal rule
- imat_shp enhancements
    - New method interp to perform shape interpolation
    - plot can now cycle through shapes in an animation. This is useful for animating transient data.
    - plot adds options to reverse the X-, Y-, or Z-axis when displaying.

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 720, 721, 782, 786, 789, 795, 802, 803, 805, 812, 813, 814, 816, 817, 820, 825, 826, 829, 834, 836, 838, 840, 846, 848, 849, and 851

v3.1.0

New Features/Enhancements

- Designed to work with MATLAB R2010a. Should also work with R2008a, R2008b, R2009a, and R2009b
- Created a new Compatibility Considerations page
- ideas_filt has been deprecated and replaced with imat_filt, and will be removed in a future release of IMAT. The new class is completely compatible in implementation with the old class. Please see Compatibility Considerations for more details.
- ideas_ctrace has been deprecated and replaced with imat_ctrace, and will be removed in a future release of IMAT. The new class is completely compatible in implementation with the old class. Please see Compatibility Considerations for more details.
- ideas_shp has been deprecated and replaced with imat_shp, and will be removed in a future release of IMAT. The behavior of the new class has changed slightly. Please see Compatibility Considerations for more details.
- ideas_fn has been deprecated and replaced with imat_fn, and will be removed in a future release of IMAT. Some ideas_fn attributes have been removed in the new class, and the behavior has changed slightly. Please see Compatibility Considerations for more details.
- ideas_dir2num and ideas_num2dir have been deprecated and will be removed in a future release of IMAT. They have been.replaced with imat_dir2num and imat_num2dir.
- Add `plot4views` in the *examples* folder to display mode shapes in a 4-view display and optionally generate output picture files
- readnas enhancements
    - Significant memory savings when processing OP2 files with many results that have the same components
    - You no longer need geometry to process shell stress resultants as long as you don't request actual node numbers and don't want to transform results to the basic coordinate system
    - SPOINTS are imported as nodes at coordinates 0,0,0.
- imat_fn enhancements
    - plot now accepts MATLAB-style plot formatting strings
    - rms allows you to select interpolation types other than linear
- imat_ctrace enhancements
    - unique now returns an optional 3rd output argument
- imat_shp enhancements
    - plot adds options to color the display based on the shape amplitude
- result enhancements
    - Added Max Shear (MaxS) invariant
    - DataCharacteristic can be modified to Scalar if data is already present

- readodb enhancements
    - Supports Abaqus 6.9-EF1
    - Remove support for 32-bit Linux since Simulia has removed support for that platform

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 542, 717, 719, 725, 726, 727, 728, 732, 738, 750, 757, 764, 776, 778, and 781

v3.0.0

New Features/Enhancements

- Designed to work with MATLAB R2009b. Should also work with R2008a, R2008b, and R2009a
- Reorganize product structure to IMAT, IMAT+FEA, IMAT+Modal, and IMAT+Signal
- ATAServer licensing support has been removed
- New experimental features have been added, and are described here. The primary new features added with this release are the imat_fn and imat_ctrace classes, which will replace the imat_fn and ideas_ctrace classes in the future.
- readnas enhancements
    - Support TABLED1 cards from BLK
    - Add some PARSEOP2_* functions to demonstrate how to process raw OP2 tables
    - You no longer need geometry in the OP2 when processing stress/strain and 1D element forces as long as you don't request actual node numbers and don't want to transform results to the basic coordinate system
    - You can specify the input options structure to readnas using a file in the current working directory called `readnas_options.mat`
    - Added a global option to specify I-deas format (`opt.global.ideas`). The default is `false`.
    - Changed the default to return centroidal results. Turning off centroidal result output will no longer return any centroidal results (previously shell stresses written to the Output2 file without the CORNER option were always returned).
    - Support arbitrary number of continuation lines with bulk data (necessary for reading large DMIG and TABLE* cards, for example)
- New function create_op2table to convert ideas_shp and result formats to the structure format expected by writenas
- New function femap_invoke that allows you to call FEMAP API methods via COM directly from MATLAB
- Added a detailed section in the FEM section of the User's Guide describing how coordinate systems are handled in IMAT
- readnmat has been deprecated and will be removed in a future release of IMAT. Please use readnas instead.
- Significant performance improvements in xform, ideas_shp/xform, and imat_fn/xform
- imat_fn enhancements
    - plot 'Tag' specifier now allows you to specify a format string for the X and Y display values
- uiplot can now import SORT2 and XYPUNCH records from a Nastran PCH file
- writeunv now exports nodes to dataset 2411 by default. It is still possible to export to dataset 15 with a command-line option.

Bug Fixes

- Minor documentation and bug fixes
- Fig bug in imat_fn/octaven that caused even Nth octave center frequencies to be incorrect (some were correct, but it produced too many)

- ◦ Fix indexing bug in ideas_shp/xform that caused incorrect cylindrical and spherical transformations in some cases where the FEM had cartesian systems as well
- ◦ Fix TIER reports 597, 598, 604, 606, 610, 618, 621,627, 628, 629, 630, 636, 640, 643, 646, 647, 648, 654, 658, 660, 680, 681, 682, 683, 684, 686, 687, 688, 697, 698, 699, 700, 704, 705, 711, 713, and 714

v2.60

New Features/Enhancements

- Designed to work with MATLAB R2009a. Should also work with R2007b, R2008a and R2008b.
- New writenas command to write tables to an OP2 file
- imat_fn enhancements
  - New function psd2trans to generate a transient from a Power Spectral Density
  - New methods mpower, power, and sqrt
  - psd and csd performance significantly improved, and they no longer require the Signal Processing toolbox
  - New methods db, db2eu, log10, log2, and exp
- ideas_shp enhancements
  - plot can now animate complex modes as complex
- readnas enhancements
  - Support MSC Nastran v2007
  - New option `'asraw'` to return any table as a raw table rather than processing it
  - PCH file processing has been overhauled to return SORT1 records either as structures or ideas_shp or result, or as functions (the previous behavior)
  - No longer require geometry to be present in the OP2 file when reading nodal and energy results, as long as you are not transforming to the basic coordinate system or requesting actual node numbers in the result components
  - The `.result` field now always contains a cell array. Previously if it only contained 1 result the field would contain that result directly.
  - Support many previously unsupported result datasets
  - Return DMIG matrix DOF in original (unsorted) order
- readunv now handles blank lines at the end of datasets 58, 1858, 164, 1802, and 252.
- writeneu has been renamed to writefemap. It now supports both export to Neutral file and direct write to a FEMAP session using COM.
- ideas_progress has been renamed imat_progress, and has new capabilities. The old function will produce a warning and will be removed in a future release of IMAT.
- ideas_getfile and ideas_putfile have been renamed to imat_getfile and imat_putfile, respectively. The old functions will produce a warning and will be removed in a future release of IMAT.

Bug Fixes

- ◦ Minor documentation and bug fixes
- ◦ Fix TIER reports 451, 454, 456, 484, 500, 520, 533, 535, 536, 541, 550, 574, 592, 593, and 602

v2.50

New Features/Enhancements

- Designed to work with MATLAB R2008b. Should also work with R2008a, R2007b, and 2007a.
- Introduce new licensing mechanism that uses Sentinel RMS from SafeNet. ATAServer will be phased out over the next few releases.

- imat_fn enhancements
    - truncate now supports explicit abscissa values as input, and allows specification by index and value
    - psd adds a peak hold option
    - csd and frf have been modified to use Gyx instead of Gxy. The documentation has been updated to clarify the meaning of the subscripts
    - The imat_fn constructor function now converts results with Data At Nodes On Elements and Data On Elements DataLocations.
    - New datatype 'Voltage'. Note that this datatype is unique to IMAT and is not supported by other software packages.
- ideas_shp enhancements
    - plot now has the capability of plotting both line and shaded image styles
    - plot performance improvements
- readnas enhancements
    - Adds analysis program masks for I-deas (the default) and FEMAP, to tailor result output to be consistent with these software programs
    - Adds support for reading BSET, CSET, QSET, and USETs in addition to ASET
    - Add support for importing centroidal shell stress resultants
    - Add support for CBUSH1D and PBUSH1D element cards
    - Add support for CBUSH and CBUSH1D forces and CBUSH stresses (see readnas detailed notes for more information)
    - Remove support for obsolete CWEDGE element
    - Add support for reading ***any*** unrecognized table from the OP2 file as generic INT32 output. New helper functions TYPECAST_OP2 and PARSEOP2TABLE_DMIG.
    - Add support for ONRGY1 (strain energy) and ONRGY2 (kinetic energy) datablocks when reading OP2
    - Add support for PCOMP MSC Nastran bulk data format that has 1 ply defined per continuation line
    - Add support for PARAM,POST,-1 geometry datablocks (GEOM*S, GEOM*VU, etc.)
- readodb enhancements
    - Supports Abaqus 6.8-1.
    - Recognize more field types such as SENER and related.
    - Add support for data stored on element faces.
- Support for FEMAP (IMAT+)
    - writeneu exports functions, shapes, results, groups, and FEM geometry to a FEMAP Neutral file

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 371, 378, 408, 446, 450, 455, 485, 495, 497, 498, 499, 501, 502, 503, 511, and 523

v2.41

New Features/Enhancements

- `uiselect` for imat_fn, ideas_shp, and result now have resizable columns and are not limited to 3 attributes
- `setdisplay` for imat_fn, ideas_shp, and result allow an arbitrary number of columns to be displayed
- readnas enhancements
    - Support large file sizes (>2 Gb)
- readodb enhancements
    - Add support for more field datatypes such as COPEN, CSLIP1, and CSLIP2

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 401, 436, and 440

v2.40

New Features/Enhancements

- Designed to work with MATLAB R2008a (7.6). Should also work with R2006b,R2007a, and R2007b.
- New demo files, updated help, and helper function `imathelp` to launch IMAT documentation in a browser window
- New function mat2subst to convert from IMAT matrix structure format to substructure format
- readnas enhancements
  - Read matrix datasets from the PCH file (i.e. MEFFMASS)
  - Support material and physical properties
  - Return material ID and beam cross section IDs (where applicable) when reading elements
- New `imat_fn` capabilities
  - New function cumrms to calculate cumulative RMS from frequency domain functions
  - New function srs2trans (IMAT+Testkit) to calculate a transient from a shock response spectrum
  - New function findpeaks to locate peaks in a function
  - uiplot can now stack and cycle plot functions grouped by response coordinate
- SMAC (Synthesize Modes and Correlate) modal curve fitting utility, generously provided by Sandia National Laboratories, is included in the `examples` directory

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 355, 362, 367, 370, 382, 385, 387, 388, 396, 412, 416, 420, 432, and 433

v2.30

New Features/Enhancements

- Designed to work with MATLAB R2007b (7.5). Will also work with R2006b and R2007a, but not earlier versions
- readadf now supports ADFs generated by UGS NX that are written in non-SI units
- writeadf can now overwrite records in the ADF
- Significant performance improvements for plotfem and ideas_shp/plot.
- New `imat_fn` capabilities
  - New function frf to compute Frequency Response Functions from CSD/PSD functions
  - New function filteri to IIR filter time histories
- New `ideas_shp` capabilities
  - plot now uses and accepts uipanel handles, so you can embed plots in an existing figure with other elements in it
- New FEM structure capabilitites
  - New function fem_create to create a default FEM structure
  - plotfem now uses and accepts uipanel handles, so you can embed plots in an existing figure with other elements in it
- readnas enhancements.
  - Read ASET from OP2 and BLK files
  - Read eigenvalue table directly (OLB, LAMA, and XLAMA) from OP2 and return as a structure

    - Read PCH files (SORT1, SORT2, XYPUNCH)
    - Optionally return stress, strain, and element force at centroids
- readodb enhancements
    - Supports Abaqus 6.7 ODBs
    - Now available on 64-bit Windows and Linux platforms

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 308, 316, 322, 332, 338, 344, and 348

v2.20

New Features/Enhancements

- Designed to work with MATLAB R2007a (7.4). Will also work with R2006b (7.3), but not earlier versions
- Initial release of IMAT+RTK
- New `result` capabilities and enhancements
    - New function list to list result components and data in tabular form
    - Significant reduction in temporary memory used by the `result` object
- New `imat_fn` capabilities
    - octaven can now use either ISO/ANSI standard frequencies or exact frequencies
    - New function srs to compute shock response spectra from time histories (part of IMAT+Testkit)
    - New function sweep to create a time history sweep function (part of IMAT+Testkit)
- Multiple readnas enhancements.
    - Handles both big- and little-endian files.
    - Add support for OGPWG table.
    - Option to return actual node numbers for Data At Nodes On Elements (i.e. stress) results
    - Option to not transform results to the basic coordinate system. See the documentation for more details.
- Many other enhancements

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 253, 270, and 273

v2.11

New Features/Enhancements

- Designed to work with MATLAB R2006b (7.3)
- New readunv extension examples for reading group datasets 2452, 2467, and 2477. These are located in the examples directory.
- Support for reading matrices from OP2 files with readnas
- Many other enhancements
- Miscellaneous IMAT+Testkit bug fixes and enhancements

Bug Fixes

- Minor documentation and bug fixes
- Fix TIER reports 220, 241, and 242

v2.10

New Features/Enhancements

- Designed to work with MATLAB R2006b (7.3)
- Initial release of IMAT+Testkit, which contains even more functionality related to pretest analysis and modal test data manipulation.
- Support for many new datablocks in readnas such as stress, strain, force, heat flux, and energy
- New result object functionality
    - Significant performance improvements
- New `imat_fn` capabilities
    - New function csd that computes cross spectral densities from time histories.
    - New function window that creates a window function for use with csd and psd.
    - New function diff that performs differentiation
    - New function integ that performs integration
    - New function interp that performs interpolation
    - New function decimate that performs decimation
    - New function envelope that calculates an envelope of the supplied functions
    - New function movavg that calculates a moving average
    - New function octaven that calculates a point-per-octave reduction
    - New function polyfit that computes a polynomial regression of the supplied functions
    - Many new statistical functions, including min, max, mean, std, var, skew, kurt, and rms
- Many other enhancements

Bug Fixes

- Minor documentation and bug fixes
- Performance improvements
- Fix TIER reports 122, 134, 184, 190, and 225.

v2.01

Bug Fixes

- Minor documentation and bug fixes. These fix TIER reports 138, 145, 147, and 171.

v2.0

New Features/Enhancements

- Designed to work with MATLAB R2006a (7.2)
- Added a directory examples/Demo_Files containing a series of demo files for testing IMAT functionality
- New IMAT+ extended functionality to IMAT
    - readnas imports Nastran OP2 and bulk data files.
    - readnmat reads Nastran DMI and DMIG-formatted files. Its companion function expandDMI expands a DMI matrix.
    - readodb reads results from Abaqus ODB files.
    - New `imat_fn` methods
        - uiplot provides a full-featured GUI for plotting functions. Capabilities include multiple plot types, templates, and batch mode plot file creation.

        - edit_attributes provides a convenient GUI for editing function attributes.
    - New `ideas_shp` methods
        - edit_attributes provides a convenient GUI for editing function attributes.
- Accelerations can now be treated as G units rather than engineering units using a modifier to the units system specifier in setunits
- New result object to handle results
    - Added complete documentation describing its use and attributes
    - Added ability to import results in readunv and export them with writeunv
- New `imat_fn` capabilities
    - New function filterf that FIR filters the supplied functions
    - New function fft that computes the discrete Fourier transform
    - New function psdthat computes PSDs from the supplied time histories
    - New function truncate to truncate functions to the specified abscissa range
    - New function get_units_labels that returns units label strings
    - New capabilities with plot that allow you to create axes inside a given figure or supplied axis handle. You can now also specify tags from the function call. The Options menu is also available as a context menu over the axes. Units are now displayed on the X and Y axis labels.
    - New function plot3 that displays functions on a 3D axis.
    - New function set_fnplot_style to modify `imat_fn` line styles on a plot.
    - Virtual (read-only) attributes for time exponents
- Add support for dataset 18 (coordinate systems) in readunv
- New `ideas_shp` capabilities
    - Virtual (read-only) attributes for time exponents
- New `ideas_ctrace` capabilities
    - plot now has options to display the DOF direction labels and node labels of the supplied coordinate trace.
- New readadf options
- New writeadf options

Bug Fixes

- Minor documentation and bug fixes
- Several performance improvements

v1.45

New Features/Enhancements

- Update licensing to handle tokens
- Checked against MATLAB 7.0.1, 7.0.4, 7.1, and 7.2
- Miscellaneous bugfixes and updates

v1.41

New Features/Enhancements

- Designed to work with MATLAB 7.0.x
- New `FEM` capabilities
    - Added new function fem_labelnodes
- New `imat_fn` capabilities

    - Added support for IRIG Time format, introduced in I-deas 10m1
    - Added support for Octave Average Type, Octave Overall RMS, and Octave Weighted RMS attributes
    - New Nyquist plot option in plot
- New `ideas_shp` capabilities
    - sort to sort an `ideas_shp` by frequency
    - Enhancements and additional command-line arguments for plot and plotfem.
    - Removed obsolete function `plotshapes`
    - Added menu item to toggle node labels for undeformed geometry in plot and plotfem
- New `ideas_ctrace` capabilities
    - intersect to determine what coordinates are common between two `ideas_ctrace` variables
    - ismember to determine what coordinates in one `ideas_ctrace` are in another
    - setdiff to determine what coordinates in one `ideas_ctrace` that are not in the other
    - setxor to determine what coordinates in one `ideas_ctrace` are not in the intersection of both
    - union to return a sorted set of `ideas_ctrace` variables with no repetitions
- Changes to examples directory
    - Added `mmif.m`, computes the multivariate mode indicator function
    - Removed obsolete function `fixunv.m`
- Overhauled HTML documentation to a new format
- Enhanced ideas_datestr to generate IRIG-formatted strings
- Completely rewrote readadf and writeadf to work around filename length limits and support more platforms, and eliminate the need for adflib.dll on Windows
- `adf2mat` and `mat2adf` have been removed as they are no longer necessary
- Cleaned up `uiselect` forms for `imat_fn`, `ideas_shp`, `ideas_ctrace`, and `ideas_filt`
- readunv now handles more badly formatted UNV files seamlessly, eliminating the need for `fixunv.m` in the `examples` directory

Bug Fixes

- Minor documentation and bug fixes
- Fixed occasional problem displaying complex modes in `ideas_shp` plot

v1.31

New Features/Enhancements

- Designed to work with MATLAB 6.5
- Utilizes new client/server licensing scheme
- Added new functions to `examples` directory
    - `fixunv.m`, `fixunv.pl` fix improperly formatted UNV files generated by some 3rd party software vendors
    - `ortho.m` computes modal assurance criteria (MAC) or orthogonality
    - `nmif.m` computes the normal mode indicator function
- New `FEM` capabilities
    - New function fem_valid validates FEM structures.
- New `ideas_ctrace` capabilities
    - uiselect for `ideas_ctrace` now allows you to select coordinates with a graphical interface
- New `ideas_filt` capabilities
    - uiselect for `ideas_filt` now allows you to modify existing filter criteria
- New `imat_fn` capabilities
    - New function fn2shp to generate `ideas_shp` from a series of `imat_fn` (useful for generating Operating Deflection Shapes)

  - plot now allows command-line options to set the plot X and Y axis scaling, grid options, complex options, and X and Y axis windows
  - uiselect for `imat_fn` lets you pass in an `ideas_filt`. The record filter will be turned on when the form appears.
- New `ideas_shp` capabilities
  - New function shp2fn to generate `imat_fn` from a series of `ideas_shp`
  - Completely rewrote plot. It combines the functionality of the original plot and plotshapes, and provides a pull-down menu allowing you to change the complex mode display option and shape scaling, edit the mode shape ID lines, and toggle the title display. In addition, mode shape animation is supported. You can also create an AVI file of the animation.
  - plotshapes is no longer used. It will be removed in the next release.
- readadf now supports selective importing of functions and shapes
- readadf now supports reading only a range of abscissa values (ATI, AFU)
- writeunv now exports FEM geometry
- The readunv API has been exposed, allowing you to create Universal dataset plugins to read datasets not natively supported by readunv.
- Made several changes to plotfem. It is now much faster for large models. View and Display pulldown menus were also added to allow you to select from several pre-defined view orientations, and to change the node markers and element and traceline linestyles.
- Extended the functionality of uiselect for `imat_fn` and `ideas_shp` to allow specifying a pre-select list and to optionally return an index vector of functions selected. It also uses the attributes specified by setdisplay as the default attributes shown in the attribute columns.
- setdisplay for imat_fn and ideas_shp can now specify the maximum number of entities to display before switching to showing just the object's dimensions
- readunv now supports datasets 611/612 (substructure) and dataset 2453 (substructure)
- plot for `imat_fn` now returns an optional figure handle. Windowing and grid options are also now much faster and more efficient.

Bug Fixes

- Minor documentation and bug fixes
- Fixed a bug in uiselect for `imat_fn` where selecting functions from a filtered list would return the wrong `imat_fn` as output

v1.21

- Support for MATLAB 6. Functionally equivalent to v1.20.

v1.20

New Features/Enhancements

- Support for FEM coordinate system, node, element, and traceline entities
  - readunv now reads in FEM geometry
  - Partition a FEM structure to supplied DOF list (fem_part)
  - Concatenate FEM structures (fem_cat)
  - Perform coordinate transformations between coordinate systems (xform)
- Expanded readunv functionality to allow selective dataset reading/skipping

- Added new function to imat_fn
    - Perform coordinate transformations on functions (xform)
- Added new functions to ideas_shp
    - List shape coefficients (list_shape)
    - Parse shape to supplied DOF or node list (parse_shape)
    - Plot individual `ideas_shp` (plot)
    - Cycle through and plot multiple shapes (plotshapes)
    - Graphical `ideas_shp` selection interface (uiselect)
    - Perform coordinate transformations on shape coefficients (xform)
- Added new functions to `ideas_ctrace`
    - Plot `ideas_ctrace` on FEM geometry (plot)
    - Return index of last element (end)
- Added new general function ideas_colormap
- Added new attributes to ideas_ctrace ('name' and 'description')
- New `ideas_ctrace` helper functions ideas_ctrace/get and ideas_ctrace/set
- Various documentation updates

Bug Fixes

- Fixed bug where adf2mat could occasionally crash with a floating point exception on NT
- Fixed bug where writeadf and mat2adf would crash when trying to write out an ideas_shp with 250 or more nodes
- When creating a shape using build_shape and the shape coefficients are complex, the shape type is now correctly set to Complex
- Corrected ideas_ctrace function behavior when passing in empty ideas_ctrace variables as arguments to return empty variables
- Corrected ideas_ctrace/eq and ideas_ctrace/ne for scalar/vector comparisons to return a logical vector
- imat_fn/abs and ideas_shp/abs now tag the output datasets as real instead of leaving them tagged as complex
- Minor documentation and bug fixes

v1.10

New Features/Enhancements

- IMAT support for MATLAB 5.3
- ADFs are now treated as platform-independent due to implementation of I-DEAS Master Series 7 functionality enhancements
- Changed license scheme slightly
- Added math functions to ideas_shp
    - Addition (plus)
    - Subtraction (minus)
    - Termwise multiplication (times)
    - Array multiplication (mtimes)
    - Termwise division (ldivide)
    - Termwise division (rdivide)
    - Array division (mldivide)
    - Array division (mrdivide)
    - Absolute value (or modulus) (abs)
    - Phase angle in radians (phase)
    - Phase angle in degrees (phased)
- Improved error checking for imat_fn and ideas_shp math operations

- When trying to set an ambiguous attribute, now only the attributes that match what was entered are shown, instead of showing all of the available attributes
- Various documentation updates

Bug Fixes

- Fixed bug in imat_fn/mrdivide that prevented it from working correctly for true matrix division cases
- Created workaround for serious performance issues on NT for MATLAB 5.2 and 5.3 built-in sscanf function when processing floating point numbers. This affected readunv, and resulted in potentially significantly longer Universal file read times. A new mex-function (imat_sscanf) is provided, which may be used outside of IMAT. It follows the same usage convention as the built-in sscanf.
- Fixed problem where readadf, writeadf, adf2mat, and mat2adf could potentially crash or give MATLAB a segmentation violation with very long filenames
- Fixed miscellaneous minor logic errors and error message display bugs

v1.01

- Initial release for MATLAB 5.2 (functionally equivalent to v1.00)

v1.00

- Initial release for MATLAB 5.1 (functionally equivalent to v1.01)

IMAT Compatibility Considerations

The following lists provide a history of changes to IMAT that may impact forward compatibility.

v7.0.0

readnas

The output of readnas when importing PCH files is now different for complex results that are not nodal quantities (e.g. stress). In the PCH file, the real and imaginary (or magnitude and phase) components of a complex result are stored as separate item codes, which are separate columns in the `.data` matrix in the structure returned by readnas. Previously, readnas combined the two columns into a single column containing complex-valued data, converting magnitude and phase output into complex numbers if the results were written this way. This resulted in data that was processed from the original PCH file contents, and also created an inconsistency between the `.item` vector (which contained the list of original item codes stored in the `.data` matrix) and the number of columns in the data matrix (which corresponded to the number of item codes minus the number of imaginary/phase item codes).Starting in IMAT v7.0.0, the data is returned in the original format found in the PCH file, and the `.item` vector is now always the same length as the number of columns in `.data`.

The reasons for this change is to keep results even closer to the original Nastran format, and also to be more consistent with the output from parseop2table_raw2nas.

This may be better demonstrated by an example. The example below shows the output from a SOL111, modal frequency response analysis. The result shown below is strain output for CBEAM elements. The output below is from IMAT 7.0.0. The `.item` field contains 110 item codes, and the `.data` matrix has 110 columns, the contents of which correspond to the item codes in `.item`.

```
>> data.records{1}

ans =

  struct with fields:
         title: ' CANTILEVER BEAM EXAMPLE'
      subtitle: ' FREQ RESP SOLUTION'
         label: ' X SINE SWEEP 0.1 - 20 HZ'
            id: [2×1 double]
          item: [1×110 double]
    entitytype: 2
          seid: 0
      datatype: 3
       subcase: 1
       modenum: []
        zvalue: 0.6000
    eigenvalue: []
         acode: 54
          data: [2×110 double]
      datachar: []
```

In contrast, below is the same output as it was imported by previous versions of IMAT. The `.item` field contains 110 item codes, but the `.data` matrix only contains 66 columns. All of the item codes corresponding to the imaginary or phase component of a result were combined with the item code corresponding with the real or magnitude component.

```
>> data.records{1}

ans =

  struct with fields:
         title: ' CANTILEVER BEAM EXAMPLE'
      subtitle: ' FREQ RESP SOLUTION'
         label: ' X SINE SWEEP 0.1 - 20 HZ'
            id: [2×1 double]
          item: [1×110 double]
    entitytype: 2
          seid: 0
      datatype: 3
       subcase: 1
       modenum: []
        zvalue: 0.6000
    eigenvalue: []
         acode: 54
          data: [2×66 double]
```

One other notable change to PCH file output is the addition of the `.datachar` field of non-nodal output. Some results contain character strings among the items stored. Previously, readnas returned these as double values, which would then need to be typecast to strings to be used. Starting in IMAT 7.0.0, these are returned in the `.datachar` field of the output structure. If the result contains character-based items, they are returned in the appropriate column(s) of the cell array of strings contained in `.datachar`, and the corresponding column in `.data` is assigned `Nan` values. If the data type did not contain any character-based items, the `.datachar` field is set to an empty matrix.

The OEFIT datablock now reads in as Data At Nodes On Elements, rather than Data On Elements. This was done to be consistent with support for the failure index datablocks OEFIIP and OEFIIS, which can contain centroidal and nodal results. OEFIT failure indices are labeled as centroidal (node 0), starting at layer 1.

v6.3.0

There are no known compatibility considerations between this release of IMAT and the previous release.

v6.2.0

There are no known compatibility considerations between this release of IMAT and the previous release.

imat_filt

The imat_filt data type has been enhanced so that it can work with imat_shp in addition to imat_fn. There are no compatibility issues with existing code, as it defaults to an imat_fn-compatible filter, but there is a new calling sequence to use it with imat_shp. When an imat_filt is created, you must specify (either explicitly or implicitly) what data type it will work with. You cannot change the target data type once the filter has been created.

To create an imat_filt for use with imat_shp, pass in an imat_shp as the first input argument to imat_filt. You can either pass in imat_fn or pass in just the filter criteria to create an imat_fn-based filter. See the code snippet below for examples.

```
>> zs = imat_filt(imat_shp,'ShapeType','=','Real')              % Create an imat_filt spe-
cifically for imat_shp
zs =
...for objects of type 'imat_shp'
```

```
ShapeType == 'Real'


>> zf = imat_filt(imat_fn,'FunctionType','=','Time Response')     % Create an imat_filt spe-
cifically for imat_f
zf =
...for objects of type 'imat_fn'

FunctionType == 'Time Response'

>> zf = imat_filt('FunctionType','=','Time Response')              % Old calling format, func-
tionally equivalent to the above
zf =
...for objects of type 'imat_fn'

FunctionType == 'Time Response'
```

v6.1.0

There are no known compatibility considerations between this release of IMAT and the previous release.

v6.0.0

Coordinate Systems

The global Cartesian, or basic, coordinate system ID has been renumbered from 1 to 0 to be consistent with Nastran nomenclature. This affects import and export operations in IMAT as well as plotting and coordinate transformations.

Several IMAT functions have been modified, and the changes are described below.

imat_cs and imat_node

FEM entities will automatically be updated when loading from a MAT file. Coordinate systems with an ID of 1 whose transformation matrix is the identity matrix and whose origin is at (0,0,0) will be renumbered to 0. Nodes that reference a coordinate system of 1 will change the reference to 0.

readnas

readnas import defaults have changed. Not the OPT structure field `OPT.global.renumbercsys` has a default of FALSE. In addition, this field is modified as appropriate if you select the `OPT.global.ideas` or `OPT.global.femap` options.

readunv

readunv modifies any references to CS1 in the node and coordinate system datasets to 0 on import.

writeunv

writeunv modifies coordinate system IDs of 0 in node and coordinate system datasets being exportd to 1. If a coordinate system ID of 1 already exists, it will be renumbered to the maximum ID present + 1.

xform

imat_fem/xform, imat_shp/xform, and imat_fn/xform now display a warning if you call it with a TO coordinate system ID of 1, to alert you that if you are attempting to transform to the global Cartesian system, the calling arguments have changed. Please see the section below on migrating code to IMAT 6.0.0 for more information.

Migrating code to IMAT v6.0.0

To migrate your code to the new coordinate system nomenclature, it is likely that all you will need to do is search your code for instances of using XFORM to transform coordinate systems to global Cartesian (basic), and replace 1 with 0, as in

```
xform(...,1);
```

changes to

```
xform(...,0);
```

Code that references coordinate system IDs in both imat_cs and imat_node may also need to be modified.

Element Type Numbering

IMAT's element type numbering has been changed from I-deas-centric numbering to Nastran-centric numbering. The primary reason for this change is so FEM elements in IMAT use the same element types as the results returned by readnas. A secondary reason is that I-deas usage is declining, and its users are migrating to Siemens NX and FEMAP, both of which are more Nastran-centric.

Several IMAT functions have been modified, and the changes are described below.

Migrating code to IMAT v6.0.0

Only users who have developed scripts and functions that use the element type number will be affected by this change. One migration path is to modify your code to use the Nastran-centric numbering, which is documented here. Another path is to use the imat_elem method elemtype_n2i. to convert the new numbering to the old. This method has the disadvantage that the element types do not always map cleanly.

v5.1.0

There are no known compatibility considerations between this release of IMAT and the previous release.

v5.0.0

There are no known compatibility considerations between this release of IMAT and the previous release. However, the graphics engine in MATLAB R2014b is significantly different from previous releases, so graphical user interfaces may look and behave slightly differently.

v4.6.0

imat_fn

The `Reaction Force` data type value for `AbscissaDataType`, `OrdNumDataType`, `OrdDenDataType`, and `ZAxisDataType` has been renamed to `Force`. If you attempt to set any of these to `Reaction Force`, a warning message will issue but the attribute will be set to `Force`.

imat_shp

The `Reaction Force` data type value for `OrdNumDataType` and `OrdDenDataType` has been renamed to `Force`. If you attempt to set any of these to `Reaction Force`, a warning message will issue but the attribute will be set to `Force`.

v4.5.1

imat_result

Starting in IMAT v4.4.0, imat_result's Frequency attribute returned the damped natural frequency for complex modes. In this release, the Frequency attribute now returns the undamped natural frequency, and the ViscousDamping attribute returns the correct viscous damping value.

imat_shp

The method list_shape has been renamed to list. list_shape will be removed in a future release of IMAT.

v4.5.0

There are no known compatibility considerations from this release to the previous release.

v4.4.0

readnas

The `'records'` input argument for readnas is now obsolete. To read a subset of results from an Output2 file, previously you would pass in the string `'records'` followed by a list of record numbers to read. For example:

```
f = readnas('modes_run.op2','blocks',{'BOUGV1'},'records',[5 7:10 15])
```

This functionality has changed act more like a filter. Currently you can filter by mode number and/or subcase. Typically dynamic results (e.g. SOL103) will use the mode number filter, and static results will use the subcase filter. For example, to read select modes from a file (for all subcases):

```
f = readnas('modes_run.op2','blocks',{'BOUGV1'},'modes',[5 7:10 15])
```

To filter by subcase, use the `'subcases'` argument. For example, to read all of the displacement results for subcase 2,

```
f = readnas('modes_run.op2','blocks',{'BOUGV1'},'subcases',2)
```

You can also filter by both subcase and mode number simultaneously. To read modes 2 and 4 for subcase 3, call readnas like this:

```
f = readnas('modes_run.op2','blocks',{'BOUGV1'},'modes',[2 4],'subcases',3)
```

v4.3.0

imat_result

The previous IMAT release introduced the imat_result data type. In this release, the old `result` data type has been deprecated and replaced by the imat_result data type. While mostly compatible, there are some key differences between the two data types. Please read the user guide page on how to use the imat_result data type, even if you are already familiar with the old `result` data type. This section only highlights the key differences between the two.

Compatibility with `result` class

The imat_result class is mostly compatible with the `result` class. The descriptive attributes are the same, but accessing the components and data is different. This section highlights the differences. The table below shows the different ways to get and set components and data for `result` and imat_result objects.

Get Attributes	
result	**imat_result**
`r.NumericComponents`	`r.data.component`
<< No equivalent >>	`r.data.node`, `r.data.element`, `r.data.comp`, `r.data.layer`, etc.
`r.Data`	`r.data.data`
`r.Component`	<< No equivalent >>
<< No equivalent >>	`getComponents()`
Set Attributes	
result	**imat_result**
`.NumericComponents`	`setComponents()`
<< No equivalent >>	`r.data.node`, `r.data.element`, `r.data.comp`, `r.data.layer`, etc.
`r.Data`	`r.data.data`

Setting and Changing DataLocation

The primary difference between the old and new data types is how data and components are accessed. With the result object, specifying the DataLocation involved setting the DataLocation attribute, as shown in the examples below.

```
>> r=result(1,'DataLocation','Data at nodes on elements');

>> r=result(1);
>> r.DataLocation = 'Data at nodes on elements';
```

With imat_result, the components and data are objects stored in the `.data` property of the imat_result. As such, the DataLocation is specified simply by passing in the appropriate object.

```
>> r=imat_result(1,imat_result_danoe);

>> r=imat_result(1);
>> r.data = imat_result_danoe;
```

DataLocations can no longer be modified after data and components have been set.

Data Components

The imat_result DataLocation classes introduce components that were not available in the `result` object. The component lists for each DataLocation class are shown in the table below. New components are highlighted in blue.

DataLocation class	Component	Description
imat_result_dan	node	Node ID
	dir	Direction
	seid	Superelement ID
imat_result_doe	element	Element ID
	layer	Layer ID
	seid	Superelement ID
	eltype	Element type
imat_result_dap	point	Point ID
imat_result_danoe	element	Element ID
	node	Node ID
	comp	Component
	layer	Layer ID
	seid	Superelement ID
	eltype	Element type
imat_result_doean	node	Element ID
	element	Node ID

	comp	Component
	eltype	Element type

Accessing and Changing Components and Data

The result object provided two separate attributes for accessing components. The `.Component` attribute returned a cell array containing string representations of the components. The `.NumericComponents` attribute returned a numeric matrix containing the attributes. The columns in this matrix depended on the DataLocation.

The imat_result object only has the equivalent of the `.NumericComponents` attribute. The .component property is available from the appropriate DataLocation object, and is accessed through the .data property of the imat_result, as shown below.

```
>> r(1).data.component

ans =
     1     1    11     1     0     0
     1     1    12     1     0     0
     1     1    22     1     0     0
     1     1    13     1     0     0
     1     1    23     1     0     0
     1     1    33     1     0     0
     1     1    11     2     0     0
     1     1    12     2     0     0
     1     1    22     2     0     0
     1     1    11     2     0     0
```

Components can be accessed by retrieving the individual columns directly, as shown in the example below. Each DataLocation object defines different columns.

```
>> [r(1).data.element(1:4) r(1).data.comp(1:4)]

ans =
     1     11
     1     12
     1     22
     1     13
```

Setting components can be done by setting the individual components directly. However, the preferred way is to use the setComponents method, as shown below.

```
>> r = imat_result(1;)
>> r.data = imat_result_dan;
>> r.data= setComponent(r(1).data,[1 1 1],[1 2 3],[0 0 0],false,11:13);
>> r.data

ans =

Data At Nodes:  3DOF global translation vector
================================================
      Node        Dir       SEID              Data
------------------------------------------------
         1          1          0                11
```

```
          1         2         0                 12
          1         3         0                 13
------------------------------------------------
Number of Values: 3     Number of Nodes: 1
================================================
```

v4.2.0

imat_shp

This release of IMAT has introduced a new attribute, `.CSType`. This stores the type of coordinate system in which the shape coefficients are stored: `'Displacement'`, `'Basic'`, or `'Other'`.

When importing shapes from readnas, `.CSType` will be set to `'Displacement'` unless the datablock name starts with `'B'` or the *TransformToBasic* option was turned on. In this case, `.CSType` will be set to `'Basic'`.

When importing from Universal file with readunv, `.CSType` is always set to `'Basic'`, since this attribute is not defined in the Universal shape datasets (55 and 2414). I-deas stores shape coefficients in the global coordinate system by default in Universal files. Also, since there is no place to store this attribute in the Universal formats, it will be lost when round-tripping shapes through a Universal file.

ADFs by default store their shape coefficients in the displacement coordinate system(s). The ASH format does not have an official location to store the `.CSType` attribute in the shape header blocks. However, there are unused locations in the header, so ATA has appropriated word 39 in the header for this attribute. Since ATA does not control the ADF format, it is possible that this information could be lost in the future. However, ATA does not believe this is too likely. ASH files created by Siemens NX or I-deas software (or any other 3rd party software) will not have this attribute set, so when IMAT imports these, the `.CSType` attribute will be set to `'Displacement'`. ASH files written by IMAT will write the attribute to the file appropriately. This attribute should survive access by other software, but ATA cannot make any guarantees.

imat_shp/xform

The behavior of shape transformation is slightly different with the introduction of the `.CSType` attribute. Previously, if CSFROM was not supplied, xform would assume that the "from" coordinates were the displacement coordinates found in column 2 of the `fem.node.cs` attribute. Now, xform uses the `.CSType` setting. It also sets this attribute accordingly after transforming the shape coefficients. Note that if the `.CSType` attribute is set to `'Other'`, the CSFROM argument must be supplied.

imat_ctrace

In previous versions of IMAT, coordinate trace directions could not contain digits. This limitation has been lifted starting in this release of IMAT. The only known compatibility issue is that I-deas is not compatible with these coordinate directions, so do not use them when exporting files that will be used by I-deas.

v4.1.0

There are no known compatibility issues between v4.0.0 and v4.1.0.

v4.0.0

FEM structures

New with this release, the FEM structures have turned into classes (objects). There are 5 new classes: imat_fem, imat_cs, imat_node, imat_elem, and imat_tl. These classes operate more or less like the structures they replaced, so the transition should be fairly seamless. In most cases your existing code should continue working as-is, though you may see some warning messages about obsolete functions. The function names beginning with `fem_` have been renamed.

The new classes do consistency checking in the property setting. This ensures that your FEM data is coherent. However, if you are used to appending FEM data by simply expanding the structure field contents, this will no longer work. To work around this, you have one of two options. The first is to convert your objects into structures, append to the fields, and then convert it back to the object. The other solution is to use the appropriate ADD method. The following code block demonstrates both methods for an IMAT_NODE object.

```
>> % OLD STRUCTURE-BASED METHOD

>> ns

ns =

id: 1
cs: [1 1 1]
color: 11
coord: [0 0 0]

>> ns.id(end+1)=2;
>> ns.cs(end+1,:)=[1 1 1];
>> ns.color(end+1)=11;
>> ns.coord(end+1,:)=[1 1 1];
>> ns

ns =

id: [1 2]
cs: [2x3 double]
color: [11 11]
coord: [2x3 double]

>> % FIRST NEW APPROACH

>> n

n =

IMAT_NODE - Nodes
      ID     Def CS     Disp CS    Store CS              X              Y              Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
       1          1          1          1              0              0              0       11

>> ns=struct(n);
>> ns.id(end+1)=2;
>> ns.cs(end+1,:)=[1 1 1];
>> ns.color(end+1)=11;
```

```
>>ns.coord(end+1,:)=[1 1 1];
>> n=imat_node(ns)

n =

IMAT_NODE - Nodes
      ID     Def CS    Disp CS   Store CS            X            Y            Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
       1          1          1          1            0            0            0       11
       2          1          1          1            1            1            1       11

>> n=imat_node(ns)


>> % SECOND NEW APPROACH

>> n

n =

IMAT_NODE - Nodes
      ID     Def CS    Disp CS   Store CS            X            Y            Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
       1          1          1          1            0            0            0       11

>> n=n.add(2,[1 1 1],11,[1 1 1])

n =

IMAT_NODE - Nodes
      ID     Def CS    Disp CS   Store CS            X            Y            Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
       1          1          1          1            0            0            0       11
       2          1          1          1            1            1            1       11
```

To see the list of available properties in the imat_fem object or any of its component objects, use the `properties()` method.

```
>> fem = imat_fem;

>> properties(imat_fem)
Properties for class imat_fem:
    cs
    node
    elem
    tl

>> properties(fem.node)
Properties for class imat_node:
    id
    cs
    color
    coord
```

groups

Prior to this release, groups were loosely supported from Universal files by three sample readunv plugins. In this release, a new IMAT_GROUP object has been introduced. Its format and contents largely follow the group structure imported previously, except that the 2 columns of data are stored in reverse order from what was returned from the plugins. If you are using groups stored in the old group structure, be sure to swap the data columns when creating the IMAT_GROUP object.

imat_fn plotting

The function plotting architecture has been completely overhauled. Instead of PLOT returning a uipanel handle as it did previously, it now returns an IMAT_FNPLOT object. This object gives you significant control over the plot display, allowing you to modify the plot after it has been generated. As a result, set_fnplot_style has been deprecated, because the imat_fplot object provides all of the functionality this function used to provide, and more.

readnas

PCH file result blocknames have been renamed to reflect theirSORT type, and to make the naming consistent with Output2. For example, OUG has been renamed to OUGV1 for SORT1 format. Smilarly, OEF has been renamed to OEF1. The implications of this are that if you read in results using blocknames using the`'blocks'` specifier, you will need to rename the blocks you are specifying.

The sign on CELAS force output has been reversed, to be compatible with Ideas dataset 2414 and PCH.

v3.2.0

imat_fn

The imat_fn class adds the LoadCase attribute back. There are no compatibility issues with this change.

readnas

The readnas `'blocks'` specifier for PCH file import has been modified. Previously the specifier had to be a 1x2 cell array, where the first cell contained `'PCH'` and the second cell contained the numeric indices of the records to read. The string `'PCH'` is no longer required. The old format is still supported, but a warning will issue.

For example, the old format for reading records 1, 3, and 5 from the PCH file was

```
f=readnas('file.pch','blocks',{'PCH' [1 3 5]})
```

The new calling method is

```
f=readnas('file.pch','blocks',{[1 3 5]})
```

Another enhancement to this specifier is that now you can also use block names instead of numeric indices. For example,

```
f=readnas('file.pch','blocks',{{'OUGV1' 'PCHBULK'}})
```

v3.1.0

The ideas_fn, ideas_shp,ideas_filt and ideas_ctrace classes have been obsoleted and replaced with imat_fn, imat_shp, imat_filt and imat_ctrace. The new classes come with a lot of benefits, including the ability to easily subclass. If you don't know what this means, don't worry. The net change for users is simply that the old class names have been replaced with new class names. The old classes will continue to work for the time being, although they will be removed in a future release of IMAT.

Please see below for specific changes to each class.

imat_ctrace

There is no functional difference between ideas_ctrace and imat_ctrace. Simply replace all references of ideas_ctrace with imat_ctrace.

To convert between an `ideas_ctrace` and an `imat_ctrace`, simply recast one to the other. An example is shown below.

```
>> t = ideas_ctrace('1x','2z','3qq')

t =
    '1X+'
    '2Z+'
    '3QQ'

>> t2 = imat_ctrace(t)

t2 =
    '1X+'
    '2Z+'
    '3QQ'

>> t2.name = 'test'

test
t2 =
    '1X+'
    '2Z+'
    '3QQ'

>>
```

When using `imatctrace` in a `struct` call, you will need to wrap the `imat_ctrace` in curly braces. Previously with `ideas_ctrace` this was not necessary. For example, the following code

```
>> s = struct('field1',ideas_ctrace('1x'),'field2','text')

s =
     field1: [1x1 ideas_ctrace]
     field2: 'text'

>>
```

will no longer work. You will see an error message like this:

```
>> s = struct('field1',imat_ctrace('1x'),'field2','text')
??? Error using ==> imat_ctrace.struct
Too many input arguments.
```

Instead, do this:

```
>> s = struct('field1',{imat_ctrace('1x')},'field2','text')

s =
     field1: [1x1 imat_ctrace]
     field2: 'text'

>>
```

imat_filt

There is no functional difference between ideas_filt and imat_filt. Simply replace all references of ideas_filt with imat_filt.

To convert between an `ideas_filt` and an `imat_filt`, simply recast one to the other, as shown in the example above.

imat_shp

Generally speaking, there are very few functional differences between ideas_shp and imat_shp. Simply replace all references of ideas_shp with imat_shp. Internally the order of the numeric and string attributes have changed, but if you are not accessing these attributes directly by converting the shape into a structure, you will not be impacted by this change.

One difference in behavior is the outcome of calling the constructor with no input arguments. The `ideas_shp` class returns an empty object, but `imat_shp` returns a scalar object. To create an empty `imat_shp`, pass in an empty matrix, as shown below.

```
>> ideas_shp

ans =
I-DEAS Shape (empty)

>> imat_shp([])           % Equivalent to calling ideas_shp with no input arguments

ans =
IMAT Shape (empty)

>>
```

To convert between an `ideas_shp` and an `imat_shp`, simply recast one to the other, as shown in the example below.

```
>> s=ideas_shp(3)

s =
3x1 I-DEAS Shape with the following attributes:
Row Frequency            Damping              NumberNodes
--- -------------------- -------------------- --------------------
1   1                    0                    0
1   1                    0                    0
```

```
1   1                   0                   0

>> t=imat_shp(s)

s =
3x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1                   0                   0
1   1                   0                   0
1   1                   0                   0

>>
```

Another important change involves how attributes are returned from the get and set methods. The `ideas_shp` class had some inconsistencies with what it returned from get, between scalar and array outputs for attributes stored internally as numeric and those stored as strings. The former were returned as strings for scalar output, while the latter were returned as a single cell. `imat_shp` returns all of these attributes consistently, where scalar output is returned as a string, and array output is returned as a cell array of strings.

Finally, when using `imat_shp` in a `struct` call, you will need to wrap the `imat_shp` in curly braces. Previously with `ideas_shp` this was not necessary. For example, the following code

```
>> s = struct('field1',ideas_shp(1),'field2','text')

s =
     field1: [1x1 ideas_shp]
     field2: 'text'

>>
```

will no longer work. You will see an error message like this:

```
>> s = struct('field1',imat_shp(1),'field2','text')
??? Error using ==> imat_shp.struct
Too many input arguments.
```

Instead, do this:

```
>> s = struct('field1',{imat_shp(1)},'field2','text')

s =
     field1: [1x1 imat_shp]
     field2: 'text'

>>
```

imat_fn

Generally speaking, there are a few functional differences between imat_fn and imat_fn. The differences that do exist are highlighted below.

Several attributes that no longer serve a useful purpose have already been removed.

`ideas_fn` attribute	**`imat_fn` attribute**
`AbscissaOffset`	<removed>
`MaxOrdValReal`	<removed>
`MaxOrdValImag`	<removed>
`MinOrdValReal`	<removed>
`MinOrdValImag`	<removed>
`OrdOffsetReal`	<removed>
`OrdOffsetImag`	<removed>
`OrdScaleReal`	<removed>
`OrdScaleImag`	<removed>
`OctaveOverallRMS`	<removed>
`OctaveWeightedRMS`	<removed>
`LoadCase`	<removed>
`CoordSys`	<removed>
`ResponseEntity`	<removed>
`ReferenceEntity`	<removed>

Another difference in behavior is the outcome of calling the constructor with no input arguments. The `imat_fn` class returns an empty object, but `imat_fn` returns a scalar object. To create an empty `imat_fn`, pass in an empty matrix, as shown below.

```
>> ideas_fn

ans =
I-DEAS Function (empty)

>> imat_fn([])          % Equivalent to calling imat_fn with no input arguments

ans =
IMAT Function (empty)

>>
```

To convert between an `ideas_fn` and an `imat_fn`, simply recast one to the other. Some examples are shown below.

```
>> f=ideas_fn(3)

f =
3x1 I-DEAS Function with the following attributes:
```

```
Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Time Response    Even             0
2_(1X+,1X+)                  Time Response    Even             0
3_(1X+,1X+)                  Time Response    Even             0

>> g=imat_fn(f)

g =
3x1 IMAT Function with the following attributes:
Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Time Response    Even             0
2_(1X+,1X+)                  Time Response    Even             0
3_(1X+,1X+)                  Time Response    Even             0

>> g.responsenode=1:3

g =
3x1 IMAT Function with the following attributes:
Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Time Response    Even             0
2_(1X+,2X+)                  Time Response    Even             0
3_(1X+,3X+)                  Time Response    Even             0

>> h=imat_fn(g)

h =
3x1 IMAT Function with the following attributes:
Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Time Response    Even             0
2_(1X+,2X+)                  Time Response    Even             0
3_(1X+,3X+)                  Time Response    Even             0

>>
```

Another important change involves how attributes are returned from the get and set methods. The `ideas_fn` class had some inconsistencies with what it returned from get, between scalar and array outputs for attributes stored internally as numeric and those stored as strings. The former were returned as strings for scalar output, while the latter were returned as a single cell. `imat_fn` returns all of these attributes consistently, where scalar output is returned as a string, and array output is returned as a cell array of strings.

Finally, when using `imat_fn` in a `struct` call, you will need to wrap the `imat_fn` in curly braces. Previously with `ideas_fn` this was not necessary. For example, the following code

```
>> s = struct('field1',ideas_fn(1),'field2','text')

s =
     field1: [1x1 ideas_fn]
     field2: 'text'

>>
```

will no longer work. You will see an error message like this:

```
>> s = struct('field1',imat_fn(1),'field2','text')
??? Error using ==> imat_fn.struct
Too many input arguments.
```

Instead, do this:

```
>> s = struct('field1',{imat_fn(1)},'field2','text')

s =
     field1: [1x1 imat_fn]
     field2: 'text'

>>
```

v3.0.0

readnmat has been deprecated and will be removed in a future release of IMAT. The readnmat functionality is available in readnas. The current readnmat functionality actually calls readnas internally. Since the calling arguments and outputs are slightly different between the two functions, the source code for the readnmat wrapper is provided.

v2.60

Three functions have been renamed. In all cases the function name is different, but the calling sequence is identical. You can simply search/replace in your code to change the function names. The old function names are ideas_progress, ideas_getfile, and ideas_putfile, and the corresponding new function names are imat_progress, imat_getfile, and imat_putfile.

IMAT Installation Guide

Introduction

IMAT is fairly easy to install. There are two components: the license server (Sentinel RMS), and the IMAT toolbox itself.

You can download Sentinel RMS and IMAT at www.ata-e.com/software/ata-software.

Sentinel RMS Installation

Sentinel RMS is a robust, commercial client-server based licensing system from SafeNet that can serve multiple licenses for multiple software products simultaneously. The server typically resides on a central computer. The client software such as IMAT will reside on computers that will be utilizing the software. The client computer may or may not be the same as the license server. It can also be on a different software platform than the client.

Sentinel RMS installation is straight forward. Detailed installation instructions are shipped with the Sentinel RMS package available from ATA's website. This document highlights the basics of the installation process.

Installing Sentinel RMS

On Windows the Sentinel RMS package is an InstallShield application. As such, it must be installed by someone with administrator privileges. ATA recommends that the default selections be used during the installation.

On Unix/Linux the Sentinel RMSpackage is a gzipped tar file. Extracting this file will create an `ata_rms` directory with several files. Sentinel RMS does not need to be installed as root, but to install it so it automatically starts when the server boots, root privileges are needed. The Sentinel RMS documentation provides details about the setup process.

Environment Variable

All Sentinel RMS clients need a way to determine where the server is running. By default, the client will scan the subnet your client system is connected to and will identify any Sentinel RMS servers running on your subnet. It will then contact each until the license request is satisfied or it has run out of servers. You can control the order in which the servers are contacted through an environment variable called `LSHOST`. Set the environment variable to the hostname of the server, separated by a tilde (~). The server name must be prepended by the name '`no-net`'. For example, set it to

```
no-net~server
```

where `server` is the name of the license server.

If you only have one server and wish to bypass the network scan by the client, you can use the environment variable `LSFORCEHOST`. You can only specify a single server with this environment variable. For example, you can set it to

```
server
```

On Windows the environment variable can be set in ***Control Panel***, typically in ***System***. On Unix/Linux you can set the environment variable with `setenv` or by using `export`, depending on your shell. Please ask your system administrator or refer to your operating system documentation for more details on how to set environment variables.

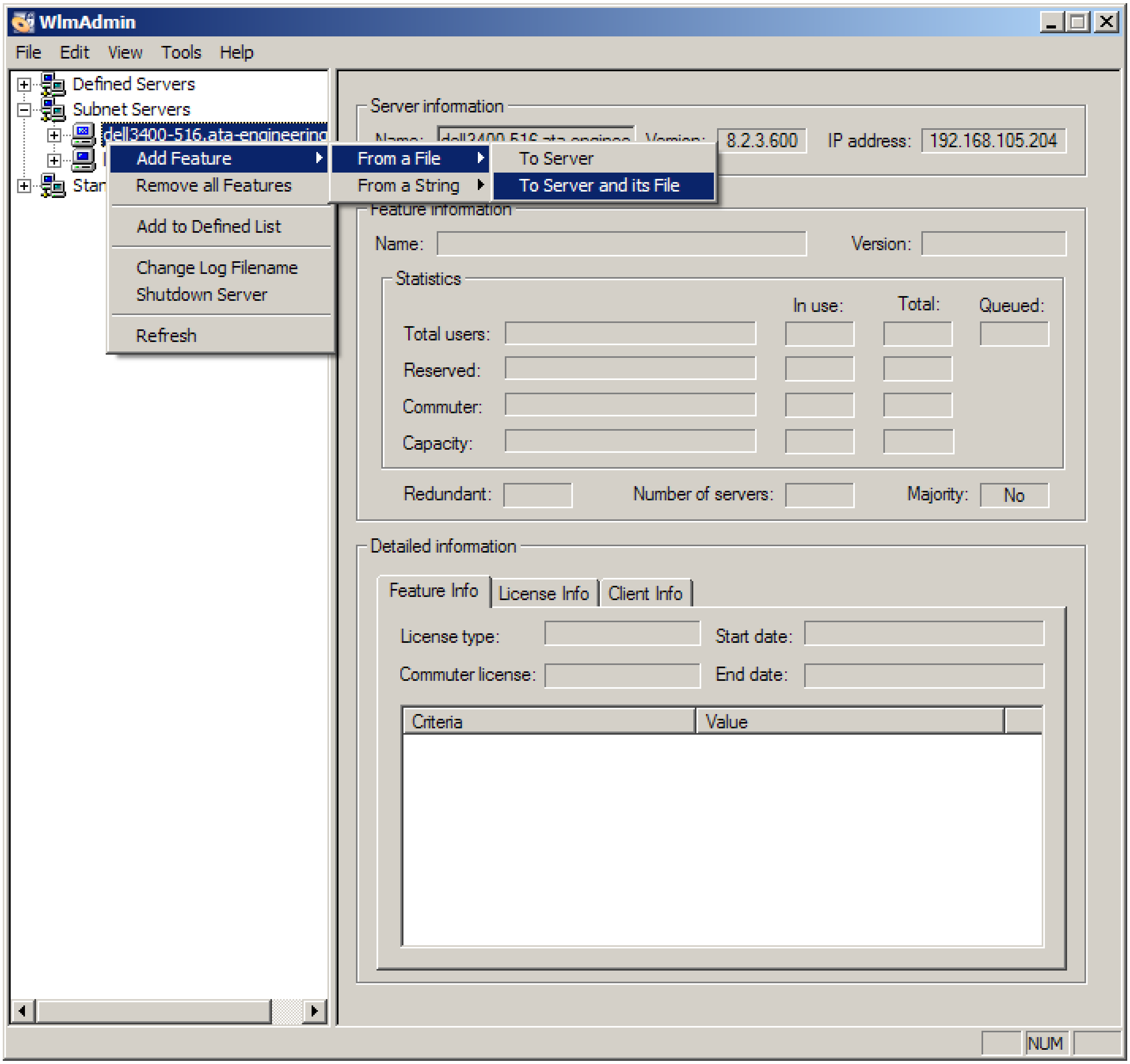
Checking Your License Status

You can use the `WlmAdmin` application included in the top-level IMAT directory to check the status of the licenses. When the GUI opens, open up Subnet Servers in the tree on the left. Under each license server, the individual licenses will be listed. Clicking on a license will display statistics about that license, including who is using it and how many are available.

Adding Or Updating Your License File

For a complete set of instructions on performing this task, please refer to the Sentinel RMS user guide. The instructions here simply demonstrate the

From Windows, open the RMS License Administration application (WlmAdmin) located in the IMAT installation. It is located in Start | Programs | ATA Engineering | Sentinel RMS Licensing | RMS License Administration. Once the application starts, install the license by right-clicking on the appropriate license server and following the menus as shown in the figure below. The license will be immediately available.



Version: 8.2.3.600
IP address: 192.168.105.204
Majority: No

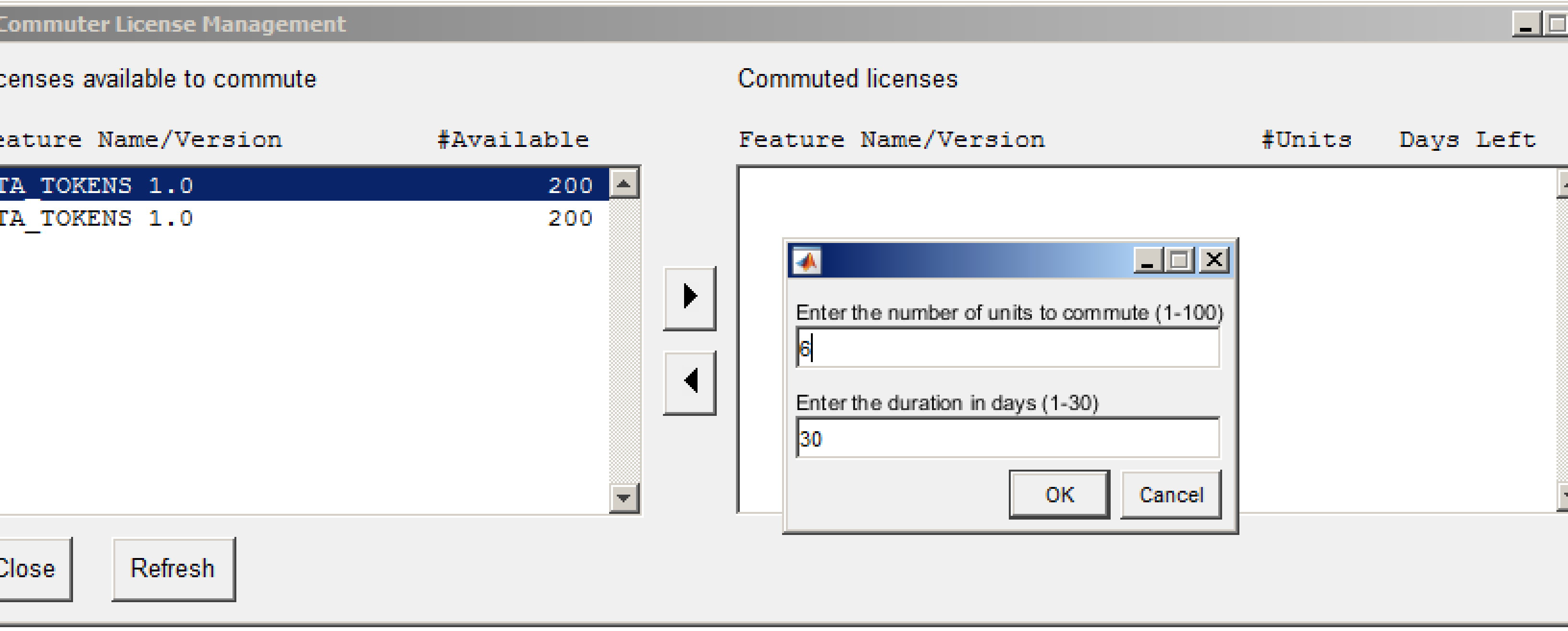
On Linux/Unix, your system administrator can install the license file by adding the license text to the `lservrc` file in the license server installation directory.

Commuting A License

Sentinel RMS provides the capability to commute a license. This capability allows you to check out a license from the server, disconnect the client machine from the network, and continue to use that license off the network for a specified period of time. When you reconnect to the network, you can check the commuted license back in.

IMAT has made this capability available through a utility called `license_commute`. Called with no arguments, it displays a graphical interface that shows the licenses available for checkout, as well as the licenses already commuted. You can check out additional licenses or return commuted licenses through this interface. Please note that you can only have one set of licenses commuted for a given Feature Name and Feature Version combination. It is also up to you to commute the correct number of units that you need to run the IMAT toolboxes you need. This is important if you are using tokens, because the different IMAT toolboxes use different numbers of tokens.

When you commute a license with `license_commute` (shown in the figure below) you can specify the duration of the commuted license, up to the maximum duration allowed by the license server. will check out the license for the maximum duration allowed. The maximum duration is specified in the license file on the server. After the allotted time period, if you have not checked the license(s) back in, they will expire on the client and will be available for use again on the server. You can also specify the number of licenses to commute. This may be necessary if you are using tokens, as different products use different numbers of tokens.



eature Name/Version	#Available
TA TOKENS 1.0	200
TA_TOKENS 1.0	200

Feature Name/Version	#Units	Days Left

Enter the number of units to commute (1-100)
6

Enter the duration in days (1-30)
30

On Windows, the `WlmAdmin` utility provided with IMAT also displays the commuted licenses. Look under ***Standalone* | *Commuter***.

Please note that when you commute a license and remove your machine from the network, you may need to reset your `LSHOST` environment variable to only specify `no-net` (or use `LSFORCEHOST`), otherwise the license client may timeout with an error about being unable to locate the your license server. The following steps guide you through the process of commuting, using, and then checking in a license.

1. Set the `LSFORCEHOST` environment variable to the name of the license server from where you want to commute a license. You can set the environment variable on the operating system, but it is probably simpler to set it in a fresh MATLAB session with the command `setenv('LSFORCEHOST','server')` where `'server'` is the name or IP address of your license server.
2. Run `license_commute` and commute the license(s). Exit MATLAB.
3. To use the commuted license, make sure your `LSHOST` environment variable starts with `'no-net'`, or set `LSFORCEHOST=no-net`. You can set this on the operating system level, or at the beginning of your MATLAB sessions as described in step 1.
4. When ready to check in a commuted license, make sure LSFORCEHOST is unset, or set to the license server from which the license(s) were commuted, or LSHOST is set appropriate for your network environment.
5. Run `license_commute` and check the license(s) in.

IMAT Installation

IMAT installation consists of extracting the package contents, setting up your MATLAB path, and setting up your system path. On Windows IMAT is distributed as an InstallShield application. It will automatically install itself in the MATLAB/toolbox directory by default. You must still edit your MATLAB path.

Extracting the IMAT package

The IMAT toolbox can reside anywhere on your system, as long as MATLAB can find it. However, ATA recommends that IMAT be installed the `imat` subdirectory (which you must create) in the `toolbox` directory of your MATLAB installation. On Windows the toolbox comes as a self-extracting archive. On Unix/Linux and Mac the toolbox comes as a gzipped tar file.

The IMAT Toolbox files are located in the following directories (where `%IMAT_INSTL%` is the top level directory of the IMAT Toolbox):

`%IMAT_INSTL%\README.txt`	The readme file
`%IMAT_INSTL%\imat`	Toolbox files for IMAT (also contains IMAT+FEA and other toolbox functionality)
`%IMAT_INSTL%\afpoly`	Toolbox files for AFPoly modal curvefitter (IMAT+Modal)
`%IMAT_INSTL%\spfrf`	Toolbox files for Signal Processing GUI (IMAT+Signal)
`%IMAT_INSTL%\ga`	Toolbox files for Genetic Algorithm (IMAT+Modal)
`%IMAT_INSTL%\mtk`	Toolbox files for Modal Toolkit (IMAT+Modal)
`%IMAT_INSTL%\tamkit`	Toolbox files for TAMKIT (IMAT+Modal)
`%IMAT_INSTL%\rtk`	Toolbox files for Rotational Toolkit(IMAT+Signal)
`%IMAT_INSTL%\doc`	Documentation (HTML format)
`%IMAT_INSTL%\examples`	Useful examples using the IMATtoolbox
`%IMAT_INSTL%\gethostid.exe`	Graphical utility that generates `hostid.txt` for license server credentials
`%IMAT_INSTL%\echoid.cmd`	Text-based utility that generates `echoid.txt` for license server credentials
`%IMAT_INSTL%\WlmAdmin.exe`	Check license status

Setting MATLAB Path

In order to use the IMAT Toolbox, you must arrange for the `%IMAT_INSTL%\imat` directory to be in your MATLAB search path. This can be done by an individual user, or it can be set up by the system administrator for all MATLAB users. Two methods for setting up the path are given below, and experienced MATLAB users will know of other ways. (In the following instructions, `%MATLAB%` refers to your top level MATLAB directory, and `%IMAT_INSTL%` refers to your top level directory where IMAT is installed.)

Option 1 (does not require sysadmin):

Create a file called `startup.m` (or edit an existing one). Put the following command in this file:

```
addpath %IMAT_INSTL%\imat -end
addpath %IMAT_INSTL%\afpoly -end
addpath %IMAT_INSTL%\spfrf -end
addpath %IMAT_INSTL%\ga -end
addpath %IMAT_INSTL%\mtk -end
addpath %IMAT_INSTL%\tamkit -end
addpath %IMAT_INSTL%\rtk -end
addpath %IMAT_INSTL%\examples -end
```

(but use the actual path in place of `%IMAT_INSTL%`.) This will add the IMAT toolbox to the end of your MATLAB path. The second command is necessary to use the example programs provided with the IMAT toolbox.

Option 2 (requires sysadmin):

Edit the file `%MATLAB%\toolbox\local\matlabrc.m` and insert the following commands in this file (after the default path is set):

```
addpath %IMAT_INSTL%\imat -end
addpath %IMAT_INSTL%\afpoly -end
addpath %IMAT_INSTL%\spfrf -end
addpath %IMAT_INSTL%\ga -end
addpath %IMAT_INSTL%\mtk -end
addpath %IMAT_INSTL%\tamkit -end
addpath %IMAT_INSTL%\rtk -end
addpath %IMAT_INSTL%\examples -end
```

(but use the actual path in place of `%IMAT_INSTL%`.) This will add the IMAT toolbox to the end of your MATLAB path. This change will need to be repeated if you update or reinstall MATLAB.

Option 3 (recommended, requires sysadmin):

Edit the file `%MATLAB%\toolbox\local\pathdef.m` and insert a line for the `%IMAT_INSTL%\imat` directory. The new file will look something like this:

```
function p = pathdef
<lines omitted...>
p = [...
%-- BEGIN ENTRIES --%
matlabroot,'\toolbox\matlab\general;',...

<lines omitted...>

matlabroot,'\toolbox\local;',...
'%IMAT_INSTL%/imat;',...
'%IMAT_INSTL%/afpoly;',...
'%IMAT_INSTL%/spfrf;',...
'%IMAT_INSTL%/ga;',...
'%IMAT_INSTL%/mtk;',...
'%IMAT_INSTL%/tamkit;',...
'%IMAT_INSTL%/rtk;',...
'%IMAT_INSTL%/examples;',...
%-- END ENTRIES --%
...
];
p = [userpath,p];
```

Use the actual path name (e.g., '`C:\apps\matlab\toolbox`') for `%IMAT_INSTL%`. A disadvantage of this method is that any MATLAB upgrade or new installation will overwrite this file, so the change will need to be repeated.

Setting System Path

The readodb function in IMAT uses shared object libraries to access the ODB. These shared object libraries must be installed somewhere in your system path, so MATLAB can find them. Note that this is NOT the same as your MATLAB path. Please see your system administrator or operating system documentation for details if you are not sure how to modify your system path. If you are on Windows and used the InstallShield installer, the system path should automatically be set up.

The necessary ABAQUS libraries are included in the IMAT distribution in the `imat\odblib` directory. You may either add this directory to your path, or move the shared object libraries to a location that is already in your system path. These libraries are also distributed with ABAQUS. If you have the version of ABAQUS supported by this version of IMAT (see the revision history for this information) installed on your system, you have the option of adding these directories to your system path:

```
%ABAQUS%\cae\exec\lbr
%ABAQUS%\cae\external
```

Updating the Toolbox Path Cache

If you install IMAT into MATLAB's toolbox directory, and you are using the Toolbox Path Cache, you will need to update it. You can do this in MATLAB in File, Preferences. Under the General section at the top of the form there is a section titled "Toolbox path caching". Simply click on the button that says "Update Toolbox Path Cache". Alternately, you can also disable the toolbox path cache by unchecking the box labeled "Enable toolbox path cache".

Configuring the Java Paths

IMAT uses Java for several features. One is our implementation of UITABLE, which uses `uitable_renderer.jar`. The second is VTKPLOT, a full-featured FEM and result visualization capability. The VTKPLOT functionality uses two JAR files, `vtk.jar` and `femplotclass.jar`. All of these JAR files must be in MATLAB's Java classpath (`classpath.txt`; see javaclasspath in MATLAB). If the JAR files are not on the Java classpath, IMAT will attempt to add them to the dynamic path at runtime. For better performance, the JAR files should be on the static classpath. See below for instructions on how to add these JAR files to MATLAB's static Java classpath.

In addition, VTKPLOT requires that the shared object files that it uses be located on the Java JNI library path (`librarypath.txt`). This path cannot be added dynamically at runtime; it must be configured in `librarypath.txt` prior to starting MATLAB.

On Windows, the installer will automatically update the `classpath.txt` and `librarypath.txt` files in the MATLAB installation, so no further action is required. On Linux and Mac, you will need to manually modify the `librarypath.txt` file. If you modify your MATLAB installation after installing IMAT, MATLAB will overwrite the `classpath.txt` file, and you will need to modify it manually.

If you do not have administrator permissions to your MATLAB installation, you can place local copies of `classpath.txt` and `librarypath.txt` into your MATLAB startup directory. Please note that starting in MATLAB 2012b (8.0), the local Java classpath files must be named `javaclasspath.txt` and `javalibrarypath.txt`.

To add the JAR files to the Java static classpath, add these lines to the bottom of `classpath.txt`, if you are editing the file located in the MATLAB installation. If you are creating a local file in your MATLAB startup directory, it need only contain these lines. You will need to replace `%IMAT_INSTL%` with the actual path to your IMAT installation, and `%PLATFORM%` to the actual platform directory. You can verify the correct `%PLATFORM%` to use by navigating your IMAT installation directory to see what is installed.

```
%IMAT_INSTL%\imat\vtklib\%PLATFORM%\vtk.jar
%IMAT_INSTL%\imat\vtklib\%PLATFORM%\femplotClass.jar
%IMAT_INSTL%\imat\uitable_renderer.jar
```

To add the shared object directory to the Java static JNI library classpath, add this line to the bottom of `librarypath.txt`.

```
%IMAT_INSTL%\imat\vtklib\%PLATFORM%\
```

IMAT Frequently Asked Questions

This list is intended to address the following commonly asked questions about IMAT. If your question is not answered here, please contact IMAT support.

1. How can I tell what version of IMAT I'm using?
2. How can I tell what version of MATLAB is supported by my version of IMAT?
3. Where can I find more information on how to use IMAT? Where can I submit a bug report or enhancement request?

How can I tell what version of IMAT I'm using?

Simply type `imat_ver` in your MATLAB window. Type "`help imat_ver`" for more information.

How can I tell what version of MATLAB is supported by my version of IMAT?

The following table shows what version of MATLAB was used to compile each IMAT release.

IMAT Version	MATLAB Versions supported
pre-1.0	MATLAB 5.1
1.0	MATLAB 5.1
1.01	MATLAB 5.2
1.10	MATLAB 5.3.x
1.20	MATLAB 5.3.x
1.21	MATLAB 6.0, 6.1
1.31, 1.31lite	MATLAB 6.5.x
1.41-1.42, 1.41lite-1.42lite	MATLAB 7.0.1
1.45, 1.45lite	MATLAB 7.0.1, 7.0.4, 7.1, 7.2
2.0	MATLAB R2006a (7.2)
2.10, 2.11	MATLAB R2006b (7.3), 7.2
2.20	MATLAB R2007a (7.4), 7.3
2.30	MATLAB R2007b (7.5), 7.4, 7.3
2.40	MATLAB R2008a (7.6), 7.5, 7.4, 7.3
2.50	MATLAB R2008b (7.7), 7.6, 7.5, 7.4, 7.3
2.60	MATLAB R2009a (7.8), 7.7, 7.6
3.0.0	MATLAB R2009b (7.9), 7.8, 7.7, 7.6
3.1.0	MATLAB R2010a (7.10), 7.9, 7.8, 7.7, 7.6
3.2.0	MATLAB R2010b (7.11), 7.10, 7.9, 7.8, 7.7, 7.6

4.0.0	MATLAB R2011a (7.12), 7.11, 7.10, 7.9, 7.8, 7.7, 7.6
4.1.0	MATLAB R2011b (7.13), 7.12, 7.11, 7.10, 7.9, 7.8, 7.7, 7.6
4.2.0	MATLAB R2012a (7.14), 7.13, 7.12, 7.11, 7.10, 7.9
4.3.0	MATLAB R2012b (8.0), 7.14, 7.13, 7.12, 7.11, 7.10, 7.9
4.4.0	MATLAB R2013a (8.1), 8.0, 7.14, 7.13, 7.12, 7.11, 7.10, 7.9
4.5.0, 4.5.1	MATLAB R2013b (8.2), 8.1, 8.0, 7.14, 7.13, 7.12, 7.11, 7.10, 7.9
4.6.0, 4.6.1	MATLAB R2014a (8.3), 8.2, 8.1, 8.0, 7.14, 7.13
5.0.0	MATLAB R2014b (8.4), 8.3, 8.2, 8.1, 8.0, 7.14, 7.13
5.1.0	MATLAB R2015a (8.5), 8.4, 8.3, 8.2, 8.1. 8.0
6.0.0, 6.0.1	MATLAB R2015b (8.6), 8.5, 8.4, 8.3, 8.2, 8.1. 8.0
6.1.0	MATLAB R2016a (9.0), 8.6, 8.5, 8.4, 8.3, 8.2, 8.1. 8.0
6.2.0, 6.2.1	MATLAB R2016b (9.1), 9.0 8.6, 8.5, 8.4
6.3.0	MATLAB R2017a (9.2), 9.1, 9.0, 8.6, 8.5
7.0.0	MATLAB R2017b (9.3), 9.2, 9.1, 9.0, 8.6, 8.5

Where can I find more information on how to use IMAT? Where can I submit a bug report or enhancement request?

There are several resources available to you. IMAT is a very functional toolbox. It can seem overwhelming at first. Utilizing the various resources presented here should allow you to get up to speed in using the software and make you more productive.

If you have technical questions regarding IMAT, please contact us at imat AT ata-e DOT com. ATA has developed training materials for IMAT. Please contact us for more information.

If you would like to submit a bug report or an enhancement request, please visit our issue tracking database at tier.ata-e.com. You can search through the existing issue database to see if your issue has been previously reported, and if a solution is available. This is the best way to communicate to us any problems you find in the software.

IMAT User's Guide

Introduction

IMAT is a toolbox that allows you to access function, time history, shape, result, and finite element model (FEM) data from I-DEAS Test, FEMAP, Nastran, Abaqus, or other software packages that support the Universal file format, operate on this data in the MATLAB environment, and write data back to I-DEAS Test or other software. For example, you can create functions or time histories in MATLAB and view them in I-DEAS, taking advantage of I-DEAS' graphing capabilities. Or you could acquire test data in I-DEAS, and operate on that data using the extensive matrix and mathematical functions available in MATLAB. Using the FEM data combined with the shape data, you can plot mode shapes in MATLAB.

IMAT+FEA provides the capability to read data from Nastran (DMI, DMIG, OP2, and bulk data) and Abaqus (ODB). You can also export data to FEMAP. In addition, IMAT+FEA provides useful graphing capabilities for imat_fn objects.

An important feature of the toolbox is that you have access to all of the I-DEAS data attributes from MATLAB. These data attributes include data type information (acceleration, displacement, sound pressure, etc.), as well as function types, descriptive information, coordinate labels, etc. All of this information is carried with each function when you import it, and you can both examine and modify the attributes.

This user guide describes the MATLAB data structures and operations which allow you to manipulate functions and other entities.

Summary of Features

Here is a summary of the features of the IMAT toolbox:

- Two different ways to transfer data between I-DEAS and MATLAB. This includes the ability to read Universal files from any software that supports this format.
- Store multiple function or shape records in a single variable. For example, you will frequently read all function records from an external file into an IMAT Function variable. Then you can use subscripting to access an individual function, a group of functions, or the entire set of functions.
- Read and write data attributes using descriptive text terms. For example, when you type

  ```
  f.functiontype='Order Function'
  ```

  the toolbox automatically substitutes the proper numeric value to represent an order function.

- Extensive selection and filtering capabilities. Select function or time history records based on coordinate traces, or on attribute filters of arbitrary complexity. Some examples:

  ```
  >> t=imat_ctrace('1001x','1002y','1003z-');
  >> f_top = f{t};      % Selection using a coordinate trace
  >> z=imat_filt('responsecoord', '=', '100?z*');
  >> f_up = f{z};       % Selection using a filter
  ```

- Much more capability is available through the upgrade to the IMAT toolbox. Please visit , or for more information.

Here is a summary of the additional features provided by IMAT+. These functions are installed with IMAT, but require additional licensing to use:

- Two different ways to import data from Nastran. Supported formats include Output2 (OP2), bulk data, and DMI and DMIG formats.
- Import data from Abaqus (ODB).

- Export data to FEMAP (NEU).
- Robust, functional plot manager GUI that can handle stacked plots and sequential plots, automatic graphic file generation, and attribute editing.

Data Transfer Options

IMAT gives you several different ways to transfer data between different software packages and MATLAB.

Direct ADF access

The most direct and efficient method is to directly read and write I-DEAS ADFs (associated data files) from within MATLAB, using the readadf and writeadf functions.

Direct ADF access works best when MATLAB and I-DEAS run on the same workstation or share access to a common file server.

Universal files

Toolbox functions are provided for reading and writing Universal data files from within MATLAB. The Universal files can be used to transport data to and from I-DEAS and any other software package that supports the format. The readunvand writeunv functions operate on universal files containing functions, time histories, mode shapes, coordinate traces, degree of freedom sets, and finite element geometry, including coordinate systems, nodes, elements, and tracelines. The readunv and writesubst functions operate on universal files containing substructure matrices (mass, stiffness, back expansion) for I-DEAS Test/Correlation. You can also extend the import capabilities of readunv by writing your own dataset plugins. The Extending Readunv guide steps you through this process.

Nastran

IMAT+FEA supports importing Nastran data from Output2, bulk data, and DMI and DMIG-formatted files. The readnas function can import OP2 and bulk data files. Not all data types within these files is supported. The documentation for these functions specifies what data types they can read.

Abaqus

IMAT+FEA supports importing Abaqus data from ODB files. The readodb function supports xyData, and history and field objects on a step. This data is imported as imat_fn, imat_shp, and result objects as appropriate.

FEMAP

IMAT+FEA supports exporting data to FEMAP through the Neutral file or directly to a FEMAP session through the COM interface. The writefemap function supports imat_fn, imat_shp, result, imat_fem, and imat_group.

Systems of Units

One of the powerful features of I-DEAS is its ability to change unit systems "on the fly". This is handled internally by storing all information in SI units, and presenting results to the user in whatever unit system is currently selected.

A different scheme has been implemented for handling units in MATLAB with the IMAT toolbox. With IMAT, you must preselect the unit system you will be working in (using the setunits function). You can select one of the nine built-in unit systems, or you can define your own units. When you read data from I-DEAS, the numeric data will be stored in your selected unit system. The units also come into effect when you write data to I-DEAS. The getunits function tells you what unit system you are working in. IMAT assumes that data imported from Nastran and Abaqus is stored in the current units system in MATLAB.

You can call setunits at any time during your MATLAB session to change your working unit system. Unlike I-deas, however, this will not automatically change the numeric values of existing data. It is the user's responsibility to maintain a consistent unit system while working in MATLAB. (If you always work in the same unit system, you may find it convenient to put a call to setunits in your `startup.m` file.)

I-DEAS does not have a G units system for acceleration data. However, IMAT has implemented this capability through a modifier to the units system specified with setunits. If this modifier is applied, all data that is tagged as acceleration will be treated as G's rather than engineering units. The modifier is as simple as appending a `'g'` to the units specifier or toggling the acceleration units treatment on the setunits form.

Functions and Time Histories

A *function* is a series of data pairs (*x*,*y*). A function can be visualized on an XY plot. The *x* values are referred to as the *abscissa*, and the *y* values are the *ordinate*. The abscissa values must always be real-valued and non-decreasing, whereas the ordinate can take arbitrary values, real or complex.

Data attributes are stored with each function to capture additional information (coordinate labels, data types, descriptive text labels, etc.). In I-DEAS, these data attributes can be accessed in the function selection form via the "Data Attributes" button. When you acquire test data in I-DEAS, many data attributes are set for you from the channel table and the setup of the data acquisition. Using the IMAT toolbox, you will be able to access the data attributes while in MATLAB.

First a word about the distinction between functions and time histories. I-deas treats functions and time histories as distinct data types. Functions are stored in function ADFs (extension `.afu`), while time histories are stored in time history ADFs (extension `.ati`). Actually, for most purposes a time history is just a special class of functions for which the abscissa is time, and the ordinate is real. Time histories can be stored in function ADFs, but most types of functions cannot be stored in time history ADFs. Siemens NX supports function ADFs (`.afu`).

Because of the similarities, the IMAT toolbox treats functions and time histories as a common data type. The distinction is necessary only when reading from or writing to an ADF, when you must use the proper file extension. The rest of this document will refer generically to functions, with the understanding that time histories are included as a special case.

IMAT+FEA utilizes the imat_fn data type for function and time history data imported from Nastran and Abaqus files. In these cases not all of the attributes available in the imat_fn object are applicable, but using this data type offers consistency within MATLAB.

Data Format for Functions

In MATLAB, the IMAT toolbox stores both the function data (abscissa and ordinate values) and the data attributes in a single MATLAB variable. This variable is marked as an `imat_fn` object, which tells MATLAB to use special methods to operate on it. The main advantage of this combined storage format is that it keeps you from losing the association between a function's data and its associated attributes.

To get started, you can see the full list of function attributes with the `set` function (described in more detail later):

```
>> set(imat_fn)

FunctionType: [ General | Time Response | Auto Spectrum
| Cross Spectrum | Frequency Response Function | Transmissibility
| Coherence | Auto Correlation | Cross Correlation
| Power Spectral Density | Energy Spectral Density
| Probability Density Function | Spectrum
| Cumulative Frequency Distribution | Peaks Valley | Stress/Cycles
| Strain/Cycles | Orbit | Mode Indicator Function | Force Pattern
| Partial Power | Partial Coherence | Eigenvalue | Eigenvector
| Shock Response Spectrum | Finite Impulse Response Filter
| Multiple Coherence | Order Function | Phase Compensation
| Harmonic Function | Octave | Temperature | Stress vs Strain
| Life | Campbell Diagram ]OwnerName: [ char string length 16 ]
Version: [ integer value ]
SetRecord: [ integer value ]
ResponseCoord: [ coordinate string ]
ResponseNode: [ integer value ]
ResponseDir: [ char string length 4 ]
ReferenceCoord: [ coordinate string ]
...
OctaveAvgType: [ None | Fast | Slow | Impulse | Linear ]
ExpDampingFact: [ real value ]
PulsesPerRev: [ real value ]
MeasurementRun: [ real value ]
IRIGTime: [ char string length 19 format DDD:HH:MM:SS.SSSSSS ]
```

The specific meaning of each data attribute can be found in the reference section. In general, you will have access to the abscissa, the ordinate, and three types of attributes:

- *Numeric attributes*, which take on numeric values (e.g., *AbscissaMin*);
- *String attributes*, which are set to character string values (e.g., *IDLine1*); and
- *List attributes*, which require you to select one value from a limited list (e.g., *FunctionType*).

You will see later how to examine and modify the function values and data attributes. Before we do, you will need to understand how functions are grouped together.

Arrays of Functions

IMAT allows you to combine multiple functions in arrays of functions.

An easy way to get such an array of functions is to read data from an ADF or universal file. If there are multiple records in the file, they will be stacked in a MATLAB column vector, each element of which is an individual IMAT Function. The following example reads functions from a universal file:

```
>> f=readunv('testdata.unv')
Read 6x1 imat_fn
Will read data in SI

f =

6x1 IMAT Function with the following attributes:
Record Name       FunctionType     AbscissaSpacing  NumberElements
----------------- ---------------- ---------------- --------------
1_(100Z+,1Z+)     Frequency Respon Even             801
2_(1Z+,1Z+)       Auto Spectrum    Even             801
3_(100Z+,2Z+)     Frequency Respon Even             801
4_(2Z+,2Z+)       Auto Spectrum    Even             801
5_(100Z+,3Z+)     Frequency Respon Even             801
6_(3Z+,3Z+)       Auto Spectrum    Even             801

>>
```

In this example, the MATLAB variable `f` is actually a 6x1 array of functions. Each element in `f` is itself a distinct `imat_fn`, and there is no requirement that the functions have the same size or compatible data attributes. Each element of `f` has its own unique function values and data attributes.

You can access individual functions in an array of functions using the same index options you use for numeric arrays in MATLAB. For the example function above, `f(3)` refers to the third record (reference 100Z+ and response 2Z+), and `f(4:6)` is an array containing the last three records of `f`. Or you can manipulate subsets of function arrays like matrix elements:

```
>> g=f;
>> g(3:4)=f(5:6)

g =

6x1 IMAT Function with the following attributes:
Record Name       FunctionType     AbscissaSpacing  NumberElements
----------------- ---------------- ---------------- --------------
1_(100Z+,1Z+)     Frequency Respon Even             801
2_(1Z+,1Z+)       Auto Spectrum    Even             801
3_(100Z+,3Z+)     Frequency Respon Even             801
4_(3Z+,3Z+)       Auto Spectrum    Even             801
5_(100Z+,3Z+)     Frequency Respon Even             801
6_(3Z+,3Z+)       Auto Spectrum    Even             801

>>
```

You are not restricted to just column vectors of functions, though this is the most common arrangement. An `imat_fn` can be two dimensional (rows and columns) like a matrix of functions, or could even have more than two dimensions.

Creating a Function in MATLAB

To create an `imat_fn` from scratch, you use the imat_fn constructor as in the following examples:

```
>> f=imat_fn(3);    % Creates 3x1 column vector of functions
>> f=imat_fn(2,3);  % Creates 2x3 matrix of functions
```

The functions are created with no abscissa/ordinate values, and with default data attributes. After you create the function, you will want to set its attributes and data values. Click here to see how to do this.

Importing Functions

There are three ways to get function (or time history) data from I-deas or other software package into MATLAB.

Direct ADF access with readadf

You can directly read the contents of a function ADF (filename `*.afu`) or a time history ADF (filename `*.ati`) using the readadf function. This function accepts an optional filename argument of the file to read. If the filename is omitted or contains wildcards, then the user will be prompted for the file to read. Here is an example:

```
>> f=readadf('testdata.ati')
ADF Name: /users/imatdemo/testdata.ati
Units   : <7> IN
Begin processing...
  5 records processed...
End processing...

f =
5x1 IMAT Function with the following attributes:
Record Name       FunctionType     AbscissaSpacing  NumberElements
----------------- ---------------- ---------------- --------------
1_(,101X+)        Time Response    Even             801
2_(,101Y-)        Time Response    Even             801
3_(,98)           Time Response    Even             801
4_(,12X-)         Time Response    Even             801
5_(,12Y+)         Time Response    Even             801
>>
```

Universal file import with readunv

The IMAT toolbox allows you to read a wide variety of *universal files*. Universal files are generally ASCII text files that can be read and written by a number of software packages. There are also binary universal file formats available for functions and time histories.

To read a universal file containing datasets 58 into MATLAB, use the readunv function in the IMAT toolbox. The calling sequence is the same as for readadf. You can provide a filename argument, or call readunv with a wildcard or no argument to interactively select the file to read.

Universal files provide a way to move function data from one platform to another. Use any text file transfer utility (such as FTP) to copy the file. Binary universal files (dataset 58b) can also be transported across platforms, but you should be sure to use a binary file transfer protocol.

Abaqus ODB file import with readodb

The IMAT+FEA extended functionality allows you to read xyData and histories from steps in an Abaqus ODB file using readodb. These functions will be imported as `imat_fn`. The Abaqus xyData and historyObject attributes will be mapped as best as possible into the `imat_fn` attributes.

Exporting Functions

Similar to importing, you have two options for transmitting function data to other software package.

Direct ADF access with writeadf

You can directly write one or more `imat_fn` variables into a function ADF or a time history ADF (if appropriate) using the writeadf function. This function takes a filename argument and one or more `imat_fn` arguments. If the filename contains wildcards, then the user will be prompted for the file to write. Here is an example:

```
>> writeadf('testdata.afu',f)
>>
ADF Name: /users/imatdemo/testdata.afu
Units   : <7> IN
  Writing 'f' (34 records)...
```

If the specified ADF already exists, the new records will be appended after any existing records in the file.

Universal file export with writeunv

You can create either ASCII or binary universal files (dataset 58 or 58b) using the writeunv function in the IMAT toolbox. The arguments to writeunv are the same as for writeadf. The first argument is a filename, which can be a wildcard. The remaining arguments are one or more `imat_fn` variables to be written to the universal file.

To write a binary universal file, you must provide a first argument of `'binary'`. This argument is followed by the filename and `imat_fn` arguments.

If you copy the universal file to a different platform, you should be sure to use an ASCII file transfer protocol so that carriage returns get converted properly. Binary universal files should be copied with a binary file transfer protocol.

Accessing Function Data and Attributes

IMAT offers two methods for you to examine or modify the contents of an `imat_fn`.

Reading attributes and data

The simplest way to get information about a function is with the syntax *`f.attrib`*, where *`f`* is the name of the variable, and *`attrib`* is the name of the attribute you want. The possible attribute names are listed in the reference section of this document, or you can get a listing with the set function. Here are some examples:

```
>> f.abscissamin

ans =

     0
     0
     0
     0
     0
     0

>> f.abscissaspacing

ans =

    'Even'
    'Even'
    'Even'
    'Even'
    'Even'
    'Even'
```

These examples illustrate that numeric attributes like *AbscissaMin* are returned in a numeric array the same size as `f`. String-valued attributes like *AbscissaSpacing* as well as list attributes are returned either as a character string (if `f` is a single function) or in a *cell array of strings* if `f` is a vector or array of functions. (For more information on cell arrays and cell arrays of strings, consult the MATLAB documentation.)

Data attributes return a single numeric or string value for each element of a function array. The *abscissa* and *ordinate* "attributes", on the other hand, each return an array of values for each element of the function array. When you ask for the ordinate you will get a multidimensional array whose first dimension corresponds to the ordinate elements, and whose remaining dimensions match the size of the `imat_fn` array. For the example we've been following above, the statement

```
x=f.ordinate
```

will set `x` to an 801x6 numeric array, since all of the elements of `f` have 801 ordinate elements. The first column of `x` is the ordinate for `f(1)`, and so on. If `f` had been a 5x3 array of functions, then `x` would have been an 801x5x3 multidimensional numeric array.

The same thing happens when you access the abscissa: an extra dimension is added as the first dimension of the result. Note: abscissa values are returned even if the function is evenly spaced. Evenly spaced functions do not actually store the abscissa, but the abscissa values are constructed when requested.

What do you get from `f.abscissa` or `f.ordinate` if the functions in `f` are not equal in length? In this case, IMAT sets the first dimension of the result (abscissa or ordinate) to the largest of any of the functions in the array. For those functions with less data than the maximum, the extra abscissa and ordinate values are filled with NaN (i.e., "not a number") values.

An alternative way to access function information is with get. In general, the expression `get(f,'attrib')` is equivalent to the expression `f.attrib` as discussed above. However, the get function also allows you to extract multiple attributes in one call. Here's an example:

```
>> r=get(f,'abscissamin','abscissainc','functiontype')

r =
   AbscissaMin: [6x1 double]
  FunctionType: {6x1 cell}
   AbscissaInc: [6x1 double]

>>
```

When you ask for multiple attributes, get returns a structure with field names equal to the attributes you selected. The value in each field is the value of the corresponding attribute, as discussed above. If you do not specify any attributes at all, then get will return all of the data attributes. After a call to get, you can refer to the individual returned values by name, provided you use the proper combination of upper and lower case to refer to the fields:

```
>> r.AbscissaMin

ans =
     0
     0
     0
     0
     0
     0
>>
```

Changing attributes and data

To change an attribute of a function, you use the syntax `f.attrib=value`. Numeric attributes should be set to a numeric array with dimension matching `f`. String-valued attributes and list attributes should be set to a cell array of strings, and the cell array should be the same dimension as `f`. The ordinate and abscissa should have an extra first dimension as discussed above.

A special case is made for times when you want to set all functions in an `imat_fn` array to have the same attribute. In this case, you can set the `value` to be a scalar numeric value or a simple character string. The same value will then be applied to all of the functions in the array. If you set the abscissa or ordinate to a single column vector, then the same vector is used for all elements of `f`.

Here are some examples of attribute manipulations:

```
>> f=imat_fn(3);                          % Make f a 3x1 imat_fn
>> f.ordinate=rand(5,3);                  % Set random ordinate values
>> f(2:3).referencecoord={'2x';'3x'}; % Set coords with cell array
>> f.ordnumdatatype='acceleration';   % Set all data types to acceleration
>> f.responsenode=[1001;1002;1003];   % Set node labels to 1001-1003
```

The set function is an alternative way to change attributes or function data. The syntax is `g=set(f,'attrib',value,...)`. You can set one or many attributes. The result *g* is a duplicate of *f*, with the exception of the attributes you specify. set also accepts a structure, like the one that get returns. For example,

```
>> r=get(f(1),'functiontype','referencecoord','responsecoord')

r =

    FunctionType: 'Frequency Response Function'
   ResponseCoord: {1x1  cell}
  ReferenceCoord: {1x1  cell}

>> f=set(f,r)

f =

6x1 IMAT Function with the following attributes:
Record Name       FunctionType     AbscissaSpacing  NumberElements
----------------- ---------------- ---------------- --------------
1_(100Z+,1Z+)     Frequency Respon Even             801
2_(100Z+,1Z+)     Frequency Respon Even             801
3_(100Z+,1Z+)     Frequency Respon Even             801
4_(100Z+,1Z+)     Frequency Respon Even             801
5_(100Z+,1Z+)     Frequency Respon Even             801
6_(100Z+,1Z+)     Frequency Respon Even             801
>>
```

The function type, reference, and response coordinate attributes of all six functions in `f` have been changed to match the corresponding attributes of `f(1)`.

Function Selection

It is often necessary to select a subset of functions from a function array. For example, you may want to select frequency response functions from a particular reference coordinate. Or you may want to look only at time histories from acceleration channels. IMAT offers two convenient techniques for function selection: coordinate trace selection and filter selection.

Selecting functions by coordinate trace

Each function record in I-DEAS (or each element of a function array in MATLAB) is identified by its reference and response coordinates. (In some cases the reference coordinate may be null.) You can extract elements of a function array based on the response coordinate identifiers. This is done using the syntax `f{trace}`, where *f* is the `imat_fn` array and *trace* is either a coordinate trace variable or a list of coordinate strings. More information on coordinate traces in IMAT can be found here.

Here are some representative examples of function selection using coordinate traces, assuming `testdata` is an `imat_fn` array:

```
>> x_accel=testdata{'101x-'};
>> strain_trace=imat_ctrace('11x','12x','13x','14x');
>> strains = testdata{strain_trace};
```

When you select functions by coordinate trace, you will obtain exactly one function for each element of the coordinate trace (and the functions in the result will be in the same order as the coordinate trace). If more than one function in the array has a given response coordinate, only the first match will be returned. If any coordinate in the coordinate trace cannot be found in the function, an error will result.

It is also possible to select functions based on reference/response pairs. The syntax for this option is `f{ref_trace,res_trace}`, where `f` is the `imat_fn` array, and `ref_trace` and `res_trace` are `imat_ctrace` variables. The result of this syntax is a two-dimensional function array with rows matching the reference coordinates in `ref_trace`, and columns matching the response coordinates in `res_trace`. Here is an example:

```
>> f=readadf('testdata.afu')
ADF Name: /users/imatdemo/testdata.afu
Units   : <7> IN
Begin processing...
75 records processed...
End processing...

f =

75x1 IMAT Function with the following attributes:
Record Name       FunctionType     AbscissaSpacing  NumberElements
----------------- ---------------- ---------------- --------------
>> ref=imat_ctrace('1x','2y-');
1_(1X+,1X-)       Frequency Respon Even             1601
2_(1X+,2X-)       Frequency Respon Even             1601
3_(1X+,3X-)       Frequency Respon Even             1601
4_(1X+,101Y+)     Frequency Respon Even             1601
5_(1X+,101Z-)     Frequency Respon Even             1601
6_(1X+,101X-)     Frequency Respon Even             1601
7_(1X+,401Z-)     Frequency Respon Even             1601
8_(1X+,401Y-)     Frequency Respon Even             1601
...
71_(2Y-,2X-)      Frequency Respon Even             1601
72_(2Y-,3X-)      Frequency Respon Even             1601
73_(3X+,1X-)      Frequency Respon Even             1601
74_(3X+,2X-)      Frequency Respon Even             1601
75_(3X+,3X-)      Frequency Respon Even             1601

>> res=imat_ctrace('101x','101y','101z-');
>> frf = f{ref,res}

frf =

2x3 IMAT Function with the following attributes:
Row Col Record Name      FunctionType     AbscissaSpacing  NumberElements
--- --- ---------------- ---------------- ---------------- --------------
1   1   1_(1X+,101X+)    Frequency Respon Even             1601
2   1   2_(2Y-,101X+)    Frequency Respon Even             1601
1   2   3_(1X+,101Y+)    Frequency Respon Even             1601
2   2   4_(2Y-,101Y+)    Frequency Respon Even             1601
1   3   5_(1X+,101Z-)    Frequency Respon Even             1601
2   3   6_(2Y-,101Z-)    Frequency Respon Even             1601
>>
```

This type of function array is very useful for some types of modal analysis. In the above example, `frf.ordinate` would return a 1601x2x3 array of FRF values vs. frequency, reference, and response.

Note that for the reference/response selection to work, the coordinate traces in the braces must be actual imat_ctrace variables. For example, `f{'1x','2x'}` will look for response coordinates equal to `'1x'` and `'2x'`, rather than returning a single function with reference coordinate `'1x'` and response coordinate `'2x'`. Also, you will get an error if any of the reference/response pairs are not present in the function array.

Selection by filter

Another powerful selection method makes use of "filters". A filter (in this context) refers to a set of selection criteria. For a more complete discussion of filters, click here.

To use a filter to select functions, you use the syntax *`f{filter}`*. The filter in braces can either be an imat_filt variable (which allows for fairly complicated filter criteria), or can be a simple filter expression using 3 inputs. Here are some examples of both types of expressions:

```
>> g = f{'functiontype', '=', 'frequency response function'};
>> g = f{'responsenode', '~=', 1000};
>> z1 = imat_filt('functiontype', '=', 'time response');
>> z2 = imat_filt(imat_fn,'responsecoord', '=', '101*');
>> g = f{z1&z2};
```

The first example selects all frequency response functions. The second example selects all functions except those with a response node of 1000. The third example selects time histories whose response coordinate starts with 101.

You can write a filter criterion for reference or response coordinates being members of a coordinate trace. Consider this example:

```
>> t = imat_ctrace('101x','102y');
>> g = f{'responsecoord', '=', t};
```

The function `g` will have all elements of `f` whose response coordinate is either 101X or 102Y. This is slightly different than using coordinate trace selection. In this case, all elements of `f` that match the criterion will be returned, and the records will be in their original order.

Math Operations

You can perform any MATLAB operation on an `imat_fn` simply by extracting its ordinate into a numeric array, performing the desired operation, and creating a new `imat_fn` with the resulting ordinate. However, some operations can be performed directly on `imat_fn` objects, using methods built into IMAT. Here is a summary of the available operations:

Unary Operations	
+f	Unary plus (leaves `f` unchanged)
-f	Unary minus (negate the ordinate of `f`)
real(f)	Take real part of the ordinate of `f`
imag(f)	Take imaginary part of the ordinate of `f`
conj(f)	Take complex conjugate of the ordinate of `f`
abs(f)	Take absolute value of the ordinate of `f` (or modulus if `f` is complex)
phase(f)	Return phase angle of complex ordinate in radians
phased(f)	Return phase angle of complex ordinate in degrees

Binary Operations	
f+scalar, f-scalar	Add, subtract scalar value to all ordinate values
f.*scalar, f./scalar, f.\scalar	Multiply or divide function ordinate by scalar value
f+g, f-g	Add or subtract two functions. Operation is performed on ordinates of matching array elements of `f` and `g` (i.e., `f(1)-g(1)`, `f(2)-g(2)`, etc.)
f.*g, f.\g, f./g	Termwise multiply or divide two functions. Operation is performed termwise on ordinates of matching array elements of `f` and `g`.
A*f, f/A, A\f	Multiply (or divide) matrix `A` times function `f`. The matrix operation is performed at all abscissa elements, with `f` replaced by its ordinate values at that element. The number of rows in the function array `f` should equal the number of columns of `A`, and all functions in `f` should have equal size.

The matrix multiply operations are the most confusing, because of the extra dimension on the ordinate. Matrix multiplication operates on the vector of functions, and is repeated at each ordinate value. Suppose `f` is a 2x1 `imat_fn`, and `A` is 3x2. Then the statement `g=A*f` creates a 3x1 `imat_fn` whose attributes match `f(1)`, and whose ordinate is

```
g(1).ordinate = A(1,1)*f(1).ordinate + A(1,2)*f(2).ordinate
g(3).ordinate = A(3,1)*f(1).ordinate + A(3,2)*f(2).ordinate
g(2).ordinate = A(2,1)*f(1).ordinate + A(2,2)*f(2).ordinate
```

The multiplication by `A` is done for each ordinate value. This type of operation is very useful for cases where one set of measurements can be constructed as a linear combination of another set of measurements. The combination is done at every time step (for a time history) or at every frequency (for a frequency response function). Note that you may need to set the ordinate data type and other data attributes on the result of such an operation.

Function Sets

A *function set* is a group of functions with all data attributes in common except for ordinate values and Z-axis attributes. These are handy for grouping functions together for a waterfall plot, for example.

When you read an ADF containing function sets into MATLAB, the function sets will not be immediately obvious. Each individual function record will show up in the `imat_fn` variable. However, it is easy to reconstruct individual function sets, because each function set has a unique value of the *Set* data attribute. For example, to extract function set 2 from an ADF, you can use the following commands:

```
>> f=readadf('filename.afu');     % Read all records into f
>> f2=f{'set','=',2};             % Select only those records in set 2
```

Now you could produce a waterfall plot in MATLAB with the commands:

```
>> freq=f2(1).abscissa;
>> time=f2.ztimevalue;              % Get time values (could also get RPM, etc)
>> mesh(time,freq,f2.ordinate);     % Frequency axis is the same for all
```

Alternatively, you can create a function set in MATLAB, making sure that all elements of the function have consistent data attributes and abscissa values. Give each function the same *Set* data attribute (e.g., 1). Then set the Z-axis values for all of the functions. You can then export this function set to I-DEAS Test (using writeadf or writeunv) and view the waterfall in I-DEAS.

Mode Shapes

Mode shapes describe the amplitude of motion at various locations in a structure. In conjunction with modal frequencies, damping, and scale factor (such as modal mass), mode shapes allow the structural dynamic response to be described by a modal model.

There is a special ADF type for mode shapes. It has the extension `.ash`. Similar to functions and time histories, the shape records contain more than just the raw shape data. Additional data attributes are stored with the shapes, including modal parameters. The data attributes describe the data type of the shape data (for unit conversion), the frequency, damping, and modal mass of the mode, as well as node numbers associated with the shape coefficients.

IMAT+FEA utilizes the imat_shp data type for nodal result data imported from Nastran and Abaqus files. In these cases not all of the attributes available in the imat_shp object are applicable, but using this data type offers consistency and convenience within MATLAB.

Data Format for Shapes

The IMAT toolbox includes a data class (`imat_shp`) for mode shapes and other nodal result types. Assuming you have already worked with functions in IMAT, you will find that shapes are treated in a similar way. An `imat_shp` variable contains both the shape data (i.e., the shape coefficients) and various numeric and string attributes.

An important feature of an `imat_shp` is that it includes shape coefficients for all degrees of freedom for each node. In case the shape coefficient was not calculated or is not defined (e.g., for a uniaxial or biaxial measurement), a zero value is filled in for that degree of freedom. The shape record retains no memory of which degrees of freedom were measured and which were filled in. Also, all coordinate signs are positive in a shape record (the sign correction is made when the shape is created.)

To see the true contents of a shape variable, you can use the set function:

```
>> set(imat_shp)
           IDLine1: [ char string length 80 ]
           IDLine2: [ char string length 80 ]
           IDLine3: [ char string length 80 ]
           IDLine4: [ char string length 80 ]
           IDLine5: [ char string length 80 ]
   AbscissaAxisLab: [ char string length 20 ]
  AbscissaUnitsLab: [ char string length 20 ]
   OrdinateAxisLab: [ char string length 20 ]
  OrdinateUnitsLab: [ char string length 20 ]
        CreateDate: [ char string length 20 ]
        ModifyDate: [ char string length 20 ]
         OwnerName: [ char string length 16 ]
         Frequency: [ real value ]
           Damping: [ real value ]
     ModalMassReal: [ real value ]
     ModalMassImag: [ real value ]
...
    OrdDenTypeQual: [ Translation | Rotation ]
   OrdDenExpLength: [ integer value ]
    OrdDenExpForce: [ integer value ]
     OrdDenExpTemp: [ integer value ]
```

```
          OrdDenExpTime: [ integer value ]

>>
```

The specific meaning of each attribute can be found in the reference section. In general, you will have access to the shape coefficients, the node labels, and three types of attributes:

- *Numeric attributes*, which take on numeric values (e.g., *Frequency*);
- *String attributes*, which are set to character string values (e.g., *IDLine1*); and
- *List attributes*, which require you to select one value from a limited list (e.g., *ShapeType*).

You will see later how to examine and modify the function values and data attributes.

Arrays of Shapes

Just as for functions, multiple shapes can be arrayed in a single `imat_shp` variable. Most often, you will work with a "column vector" of shapes (number of modes by 1), corresponding to multiple modes of the same structure (either analytically generated or extracted from test data). The various elements of a shape array need not be compatible, but most operations work most effectively when they are.

If you can think of a use for multidimensional arrays of shapes, IMAT will happily work with them.

Creating a Shape in MATLAB

To create an `imat_shp` from scratch, you use the imat_shp constructor:

```
>> s=imat_shp(3);    % Creates 3x1 column vector of shapes
>> s=imat_shp(2,3);  % Creates 2x3 matrix of shapes
```

The shapes created in this way will have the default attributes, which normally means they will have zero frequency and damping, and no shape coefficients defined. After you create a shape, you will want to set its attributes and data values. Click here to see how to do this.

It is common to have a matrix of shape coefficients associated with a list of coordinates. For example, you may use a set of frequency response functions measured at various response coordinates to determine mode shape coefficients at those coordinates. A convenient way to build a shape with those coefficients is the build_shape function. This function essentially does the same operation that I-DEAS does when it converts shape information at coordinates into a shape record.

Importing Shapes

There are three ways to get shape data from other software packges into MATLAB.

Direct ADF access with readadf

You can directly read the contents of a shape ADF (filename `*.ash`) using the readadf function. This function accepts an optional filename argument of the file to read. If the filename is omitted or contains wildcards, then the user will be prompted for the file to read. Here is an example:

```
>> s=readadf('testdata.ash')
ADF Name: /users/imatdemo/testdata.ash
Units    : <7> IN
Begin processing...
5 records processed...
End processing...

s =
5x1 IMAT Shape with the following attributes:
Row Frequency            Damping              NumberNodes
--- -------------------- -------------------- --------------------
1   5.92815              0.0229441            10
2   9.56484              0.00663818           10
3   11.3931              0.0271432            10
4   83.2349              0.0362356            10
5   106.677              0.0130822            10
>>
```

Universal file import with readunv

The IMAT toolbox allows you to read a wide variety of *universal files*. Universal files are generally ASCII text files that can be read and written by a number of software packages.

It is important to realize that by convention, shape coefficients in a universal file are stored in the global (basic) coordinate system. If any of the nodes of your currently active finite element model has a local displacement coordinate system, *the shape coefficients will be rotated to align with the global coordinate system*. Conversely, shape coefficients in an ADF are by convention stored in the displacement (local) coordinate system for that node.

To read a test or analytical shape universal file into MATLAB, use the readunv function in the IMAT toolbox. The calling sequence is the same as for readadf. You can provide a filename argument, or call readunv with a wildcard or no argument to interactively select the file to read.

When transferring universal files between platforms, be sure to use an ASCII (text) file transfer protocol so that carriage returns get converted properly.

Abaqus ODB file import with readodb

The IMAT+FEA extended functionality allows you to read nodal field data from steps in an Abaqus ODB file using readodb. These functions will be imported as `imat_shp`. The Abaqus field attributes will be mapped as best as possible into the `imat_shp` attributes.

Exporting Shapes

Similar to importing, you have two options for exporting shapes.

Direct ADF access with writeadf

You can directly write one or more `imat_shp` variables into a shape ADF using the writeadf function. This function takes a filename argument and one or more `imat_shp` arguments. If the filename contains wildcards, then the user will be prompted for the file to write. Here is an example:

```
>> writeadf('testdata.ash',s)
ADF Name: /users/imatdemo/testdata.ash
Units   : <7> IN
Begin processing...
s processed (5 records)...
End processing...

>>
```

If the specified ADF already exists, the new records will be appended after any existing records in the file.

Universal file export with writeunv

You can create a shape universal file (dataset 55) using the writeunv function in the IMAT toolbox. The arguments to writeunv are the same as for writeadf. The first argument is a filename, which can be a wildcard. The remaining arguments are one or more `imat_shp` variables to be written to the universal file.

It is important to realize that by convention shape coefficients in a universal file are assumed to be in the global (basic) coordinate system. IMAT will not perform coordinate transformations on export, so if the shape coefficients in your `imat_shp` are not in the global coordinate system, you will need to transform them first with xform.

If you copy the universal file between platforms, you should be sure to use an ASCII file transfer protocol so that carriage returns get converted properly.

Accessing Shape Data and Attributes

IMAT offers two methods for you to examine or modify the contents of an `imat_shp`.

Reading attributes and data

The simplest way to get information about a shape is with the syntax *`s.attrib`*, where *`s`* is the name of the variable, and *`attrib`* is the name of the attribute you want. The possible attribute names are listed in the reference section of this document, or you can get a listing with the set function. Here are some examples:

```
>> s.frequency

ans =

    5.9281
    9.5648
    11.3931
    83.2349
    106.6768

>> s.referencecoord

ans =

    '1X+'
    '3Z-'
    '1X+'
    '3Z-'
    '1X+'
>>
```

These examples illustrate that numeric attributes like *Frequency* are returned in a numeric array the same size as `s`. String-valued attributes like *ReferenceCoord* as well as list attributes are returned either as a character string (if `s` is a single shape) or in a *cell array of strings* if `s` is a vector or array of shapes. (For more information on cell arrays and cell arrays of strings, consult the MATLAB documentation.)

Data attributes return a single numeric or string value for each element of a shape array. The *node* and *shape* "attributes", on the other hand, each return an array of values for each element of the shape array. When you ask for the *node* attribute (node labels of the shape coefficients), you will get a multidimensional array whose first dimension corresponds to the number of nodes, and whose remaining dimensions match the size of the `imat_shp` array. For the example we've been following above, the statement

```
n=s.node
```

will set `n` to a 10x5 numeric array, since all of the elements of `s` have 10 nodes. The first column of `n` is the list of node labels for `s(1)`, and so on. If `s` had been a 5x3 array of shapes, then `n` would have been a 10x5x3 multidimensional numeric array.

The same thing happens when you access the *shape* attribute (the actual shape coefficients): an extra dimension is added as the first dimension of the result. The row dimension of the shape coefficient matrix is equal to the number of nodes times the number of degrees of freedom per node (either 3 or 6). The *DOFType* attribute determines whether the shape has 3 or 6 degrees of freedom per node. If a shape is 3DOF, then the first three rows of the shape are the X, Y, and Z coefficients for the first node, followed by 3 values for the second node, and so on. For 6DOF shapes, there are six values per node.

If not all of the shapes have the same number of nodes or degrees of freedom, IMAT sets the first dimension of the result (*node* or *shape*) to the largest of any of the shapes in the array. For those shapes with less data than the maximum, the extra elements are filled with NaN (i.e., "not a number") values.

An alternative way to access shape information is with get. In general, the expression `get(s,'attrib')` is equivalent to the expression `s.attrib` as discussed above. However, the get function also allows you to extract multiple attributes in one call. Here's an example:

```
>> r=get(s,'frequency','damping','referencecoord')

r =

     Frequency: [5x1 double]
ReferenceCoord: {5x1 cell}
       Damping: [5x1 double]

>>
```

When you ask for multiple attributes, get returns a structure with field names equal to the attributes you selected. The value in each field is the value of the corresponding attribute, as discussed above. If you do not specify any attributes at all, then get will return all of the data attributes. After a call to get, you can refer to the individual returned values by name, provided you use the proper combination of upper and lower case to refer to the fields:

```
>> r.ReferenceCoord

ans =

    '1X+'
    '3Z-'
    '1X+'
    '3Z-'
    '1X+'

>>
```

Changing attributes and data

To change an attribute of a shape, you use the syntax `s.attrib=value`. Numeric attributes should be set to a numeric array with dimension matching `s`. String-valued attributes and list attributes should be set to a cell array of strings, and the cell array should be the same dimension as `s`. The *node* and *shape* attributes should have an extra first dimension as discussed above.

A special case is made for times when you want to set all shapes in an `imat_shp` array to have the same attribute. In this case, you can set the *`value`* to be a scalar numeric value or a simple character string. The same value will then be applied to all of the shapes in the array. If you set the *node* or *shape* attribute to a single column vector, then the same vector is used for all elements of *`s`*.

Here are some examples of attribute manipulations:

```
>> t=imat_ctrace('1x','2x','3x-')        % Measurement coordinates

t =

    '1X+'
    '2X+'
    '3X-'

>> x=[1.15 -3.21 ; -0.75 -1.18 ; -2.03 5.03 ];        % Shape coefficients for 2 modes

>> s=build_shape(t,x)             % Build an imat_shp w/ 2 modes

s =

2x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1                   0                   3
2   1                   0                   3

>> s.node               % Here are the node values

ans =
    1    1
    2    2
    3    3

>> s.shape               % Here are the shape values

ans =
    1.1500    -3.2100
         0          0
         0          0
   -0.7500    -1.1800
         0          0
         0          0
   -2.0300     5.0300
         0          0
         0          0

>> s.frequency=[6.83 ; 12.05]; s.modalmassreal=1; s.damping=[0.014 0.008]      % Now set other
attributes

s = 2x1 IMAT Shape with the following attributes:

2x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   6.83                0.014               3
```

```
     2   12.05               0.008               3
```

The set function is an alternative way to change attributes or function data. The syntax is `s2=set(s,'attrib',value,...)`. You can set one or many attributes. The result `s2` is a duplicate of `s`, with the exception of the attributes you specify. set also accepts a structure, like the one that get returns.

Partitioning Mode Shapes

There are times when it is necessary to extract mode shape coefficients at a subset of the nodes in a model. For example, you may have computed mode shapes for a full model, but you only need to display the mode shapes in a subset of the full model, often referred to as a "display model". To accomplish this partitioning operation, you create a numeric vector `n` which contains the nodes you want to keep. The statement

```
s_part=s{n}
```

creates an `imat_shp` variable `s_part` which contains mode shape information only for the nodes listed in `n`. For examples, if `s` contains shape coefficients for nodes 101, 103, 105, and 107, then

```
s{ [101 105] }
```

is an `imat_shp` variable with shape coefficients only for nodes 101 and 105. The partitioning operation applies to all modes contained in `s`.

The partition method also partitions mode shapes.

Shape Coefficient Selection

IMAT provides a very useful syntax for accessing shape coefficients based on coordinates. If `s` is an `imat_shp` object and `t` is an `imat_ctrace` object, then the expression

```
s{t}
```

creates a matrix of shape coefficients, with as many rows as there are coordinates in `t`, and as many columns as there are modes in `s`. Alternatively, a list of coordinates can be entered in the braces. For example, if `s` contained 5 modes, the expression

```
s{'105x','101y-','102z'}
```

creates a 3x5 numeric array. The first row contains the five shape coefficients for coordinate 105X, the second row contains the (negative) shape coefficients for coordinate 101Y, and the third row contains the shape coefficients for coordinate 102Z. Note that an error results if any of the specified coordinates is not present in any of the modes in `s`.

You may also modify shape coefficients using this syntax. The command

```
s{t}=x
```

sets the shape coefficients specified by the coordinate trace `t` to the values in the numeric matrix `x`. The matrix `x` should have as many rows as there are coordinates in `t`, and as many columns as there are shapes in `s`. Note that this syntax can only be used to modify existing coordinates in a mode shape. If the specified coordinates do not already exist in the shape variable, then an error will result.

The build_shape function provides a convenient way to create an `imat_shp` object from a matrix of shape coefficients at given coordinates.

Selection by filter

Another powerful selection method makes use of "filters".A filter (in this context) refers to a set of selection criteria referencing the shape data attributes. For a more complete discussion of filters, click here.

To use a filter to select shapes, you use the syntax *s{filter}*. The filter in braces can either be an imat_filt variable (which allows for fairly complicated filter criteria), or can be a simple filter expression using 3 inputs. Here are some examples of both types of expressions:

```
>> t = s{'shapetype', '=', 'real'};
>> z1 = imat_filt(imat_shp,'shapetye', '=', 'complex');
>> z2 = imat_filt(imat_shp,'doftype', '=', '3dof');
>> t = s{z1&z2};
```

The first example selects all shapes that have real normal shape coefficients. The second example selects shapes with complex shape coefficients that are 3 DOF per node.

Math Operations

You can perform any MATLAB operation on an `imat_shp` simply by extracting its shape coefficients into a numeric array, performing the desired operation, and creating a new `imat_shp` with the resulting coefficients. However, some operations can be performed directly on `imat_shp` objects, using methods built into IMAT. Here is a summary of the available operations:

	Unary Operations
+s	Unary plus (leaves `s` unchanged)
-s	Unary minus (negate the shape coefficients of `s`)
real(s)	Take real part of the shape coefficients of `s`
imag(s)	Take imaginary part of the shape coefficients of `s`
conj(s)	Take complex conjugate of the shape coefficients of `s`
abs(s)	Take absolute value of the shape coefficients of `s` (or modulus if `s` is complex)
phase(s)	Return phase angle of complex shape coefficients in radians
phased(s)	Return phase angle of complex shape coefficients in degrees
	Binary Operations
s+scalar, s-scalar	Add, subtract scalar value to all shape coefficients
s.*scalar, s./scalar, s.\scalar	Multiply or divide shape coefficients by scalar value
s+t, s-t	Add or subtract two shapes. Operation is performed on shape coefficients only if node numbers match between `s` and `t`.
s.*t, s./t, s.\t	Termwise multiply or divide two shapes. Operation is performed termwise only if nodes of `s` and `t` match.
A*s, A\s,	Multiply (or divide) matrix `A` times shape `s`. The matrix operation is performed at all shape coefficients. The num-

s/A	ber of rows in the shape `s` should equal the number of columns of `A`, and all functions in `s` should have the same number of nodes and DOF.

The matrix multiply operations are the most confusing, because of the extra dimension on the shape coefficients. Matrix multiplication operates on the vector of shapes, and is repeated at each shape coefficient. Suppose `s` is a 2x1 `imat_shp`, and `A` is 3x2. Then the statement `t=A*s` creates a 3x1 `imat_shp` whose attributes match s`(1)`, and whose shape coefficients are

```
s(1).shape = A(1,1)*s(1).shape + A(1,2)*s(2).shape
s(2).shape = A(2,1)*s(1).shape + A(2,2)*s(2).shape
s(3).shape = A(3,1)*s(1).shape + A(3,2)*s(2).shape
```

The multiplication by `A` is done for each shape coefficient. This type of operation is very useful for cases where one set of shapes can be constructed as a linear combination of another set of shapes. Note that you may need to set the ordinate data type and other data attributes on the result of such an operation.

Coordinate Traces

A *coordinate trace* is simply an ordered list of coordinates. Each individual coordinate in the trace consists of a node number followed by a four-character *direction string*. Examples of coordinates include 101X+, 1234RY-, and 12ABC. In most cases, the direction string will be X, Y, Z, RX, RY, or RZ, followed by a plus or minus sign. These directions refer to the local coordinates in which the nodal displacements are resolved. (There are cases such as acoustic measurements when a more general direction string is used.)

Coordinate traces are used in IMAT to perform selection operations on functions and shapes. This section of the user's guide shows how to manipulate coordinate traces.

Data Format for Coordinate Traces

The IMAT Toolbox includes a data class (`imat_ctrace`) for coordinate traces. These objects are much simpler than either `imat_fn` or `imat_shp` objects, as there are only a few attributes associated with a coordinate trace. You should think of an `imat_ctrace` object as an Nx1 array of coordinate strings.

You can create a coordinate trace by passing coordinate strings to the imat_ctrace constructor function:

```
>> t=imat_ctrace('1001x','1001y-','1001z')

t =
    '1001X+'
    '1001Y-'
    '1001Z+'

>>
```

Note that positive signs are assumed if you do not enter them. There is no such thing as a multidimensional coordinate trace; each coordinate trace variable is a single list of coordinates.

The attributes currently associated with an `imat_ctrace` object is its name, a description of its contents, and an internal structure version number. The name and description are both strings. The version number is for internal use, so you should not need to use it. You can assign or extract the name by using the 'dot' nomenclature, as shown in the following example, or by using the get and set functions. If the `imat_ctrace` object has a name assigned to it, the name will be displayed when the coordinate trace is displayed. If it is empty, it will not be displayed (as in the example above).

```
>> t=imat_ctrace('1001x','1001y-','1001z');
>> t.name='Coordinate trace'

Coordinate trace
t =

    '1001X+'
    '1001Y-'
    '1001Z+'

>> t.name

ans =
Coordinate trace

>> get(t,'name')

ans =

Coordinate trace

>>
```

The imat_ctrace function reference describes other ways of creating a coordinate trace.

Manipulating Coordinate Traces

You can access individual coordinates in a coordinate trace, or a subset of a coordinate trace, using normal indexing. For example, `t(2)` refers to the second coordinate in the list. You can also modify individual elements of a coordinate trace:

```
>> t(1)=t(3);
>> t(2)='1002rx';
```

Concatenating coordinate traces works the same as for vectors. For example, `[t1;t2]` creates a coordinate trace with the coordinates in `t1` followed by the coordinate in `t2`. (Actually, `[t1 t2]` would create the same result, since coordinate traces can have only one column.)

There are other ways to combine multiple coordinate traces:

- An or operation (`t1|t2`) merges two coordinate traces, eliminating any duplicates. Note that the original order of the coordinate traces is lost in such an operation. The resulting coordinate trace will be sorted by node and direction. The same operation occurs if you add two coordinate traces (`t1+t2`).
- An and operation (`t1&t2`) returns the intersection of two coordinate traces. The resulting coordinate trace will be sorted by node and direction.
- A minus operation (`t1-t2`) removes selected coordinates from a coordinate trace. The resulting coordinate trace will be sorted by node and direction.

You can also sort coordinate traces and keep only unique coordinates (i.e., eliminate duplicates).

For more complicated manipulations, it may be useful to convert a coordinate trace into a numeric or string format. The following options are available:

- The cellstr function converts a coordinate trace to a cell array of coordinate strings;
- The char function converts a coordinate trace to a string array;
- The double function converts a coordinate trace to an Nx2 numeric array, with node numbers in the first column and numeric direction codes.

Any of these forms can be manipulated in MATLAB, and the results passed back to the imat_ctrace constructor function to create a new coordinate trace. (The numeric direction codes created by the double function can be converted back to strings using the imat_num2dir function.) Strings passed into the imat_ctrace constructor should be converted to a cell array first. For example:

```
>> t=imat_ctrace('1001x','1002rz-','101z')

t =

'1001X+'
'1002RZ-'
'101Z+'

>> d=double(t)

d =

1001           1
1002          -6
101            3

>> t2=imat_ctrace(d)

t2 =

    '1001X+'
    '1002RZ-'
    '101Z+'

>> c=char(t)

c =

1001X+
1002RZ-
101Z+

>> t2=imat_ctrace(cellstr(c))      % Turn character array into cell array first

t2 =

    '1001X+'
    '1002RZ-'
    '101Z+'

>> whos

Name      Size          Bytes  Class
c         3x7              42  char array
d         3x2              48  double array
```

```
t          3x1            244  imat_ctrace object
t2         3x1            244  imat_ctrace object
Grand total is 59 elements using 578 bytes

>>
```

Other conversion functions include the following:

- node(t) returns an Nx1 column of node numbers;
- dir(t) returns an Nx1 cell array of strings containing direction strings;
- id(t) returns an Nx5 numeric array whose first column contains the node numbers, with columns 2 through 5 containing the ASCII codes of the direction strings.

Finally, you can extract the signs of the coordinate directions using the sign function. To make all coordinates positive, the abs function works on coordinate traces.

Attribute Filters

In many cases when working with objects that have a descriptive attributes, such as it is helpful to be able to filter that variable to a subset based on some of those attributes. IMAT has a selection filter data type called imat_filt that allows you to do this. An imat_filt is simply a set of criteria which can be used to select a subset of indices of a given variable. A filter can have a single criterion, or multiple criteria can be combined to create more complicated filters. You use function filters to select functions from an imat_fn or imat_shp array, using the syntax `f{filt}`.

Data Format for Filters

Each imat_filt variable is a separate filter (though it can contain multiple criteria). You may not use indexing operations on filter variables.

An imat_filt can only be used with a single data type. For example, a filter generated for use with imat_fn cannot be used with an imat_shp, as the attributes are different. You also cannot combine filters created for more than one type together with logical expressions. The default data type (if not specified when creating the filter) is imat_fn.

Each imat_filt variable consists of one or more criteria, combined by logical operators. An individual criterion has three parts:

attribute, *relation*, *value*

The *attribute* of a criterion can be the name of any valid data attribute for an imat_fn or imat_shp, with the exception of *Abscissa* and *Ordinate* for imat_fn and *Node* and *Shape* for imat_shp. The *relation* of a criterion can be any of the following:

Relation	*Criterion is satisfied if...*
`'=='` or `'='`	*attribute* is equal to the specified *value*
`'~='` or `'!='`	*attribute* is not equal to the specified *value*
`'>'`	*attribute* is greater than the specified *value*
`'>='`	*attribute* is greater than the specified *value*
`'<'`	*attribute* is less than the specified *value*
`'<='`	*attribute* is less than or equal to the specified *value*

The last four relations are applicable only to numeric data attributes.

The *value* of a criterion should be a scalar (for numeric data attributes) or a string (for string attributes) suitable for the data attribute. String matching is case insensitive. Two special features should be noted:

- For the *ReferenceCoord* and *ResponseCoord* data attributes, you may specify a coordinate trace as the value of the criterion. In this case, equality is satisfied if the reference (or response) coordinate of a function is included in the coordinate trace.
- For string properties, you may include wildcard characters: a question mark matches any single character, and an asterisk matches zero or more characters. (For example, `'ab?d*'` matches the strings `'AbCd'` and `'abxdefg'`, but not `'abccd'`.) Wildcards cannot be used for list attributes such as *FunctionType*, which have a restricted list of possible values.

You create a function filter by specifying the attribute, relation, and value to the imat_filt constructor function. Using this creation syntax will by default create an imat_filt for use with imat_fn. Here are some examples:

```
>> z=imat_filt('functiontype', '=', 'frequency response function');
>> z=imat_filt('abscissainc', '<', 0.005);
>> t=imat_ctrace('1x','2z-');
>> z=imat_filt('responsecoord', '=', t);
```

It is also possible to specify multiple criteria to the imat_filt constructor (using an Nx3 cell array), in which case the criteria must all be satisfied:

```
>> z=imat_filt(  { 'functiontype', '=', 'auto spectrum' ;
                   'idline1',     '=', 'Test*'          } )
z =
...for objects of type 'imat_fn'

  ( ( FunctionType == 'Auto Spectrum' ) & ( IDLine1 == 'Test*' ) )

>>
```

To create an imat_filt for use with another data type, or to explicitly specify the default data type when creating the filter (which could be useful for code self-documentation), pass in the valid data type as the first input argument, followed by the criteria. Here are a few examples:

```
>> zs = imat_filt(imat_shp,'ShapeType','=','Real')                % Create an imat_filt spe-
cifically for imat_shp
zs =
...for objects of type 'imat_shp'

ShapeType == 'Real'


>> zf = imat_filt(imat_fn,'FunctionType','=','Time Response')     % Create an imat_filt spe-
cifically for imat_f
zf =
...for objects of type 'imat_fn'

FunctionType == 'Time Response'

>> zf = imat_filt('FunctionType','=','Time Response')             % Default calling format, func-
tionally equivalent to the above
zf =
...for objects of type 'imat_fn'

FunctionType == 'Time Response'
```

Filters have a few attributes accessible either through dot "." notation, or through the get and set methods, and are described below.

The "Name" attribute contains a string. This is useful for giving a description to the created filter. If the Name property has been defined, it will appear when displaying the filter contains, as shown below.

```
>> z=imat_filt(imat_fn,'functiontype', '=', 'frequency response function');
>> z.name='FRF'

Name: FRF
z =
...for objects of type 'imat_fn'

FunctionType == 'Frequency Response Function'

>> get(z,'name')
  FRF

>>
```

The "criteria" attribute contains a cell array of strings of the criteria for this filter. Its output is equivalent to the output from the criteria method. The corresponding "equation" property returns a string that specifies the equation showing how the criteria relate to each other.

```
>> z=imat_filt(imat_fn,{'functiontype', '=', 'frequency response function';
'IDLine1','=','ABC*'});

>> z.criteria
ans =
    'FunctionType'    '=='    'Frequency Response…'
    'IDLine1'         '=='    'ABC*'

>> z.equation
ans =
( %s & %s )

>>
```

Manipulating Function Filters

Other than applying a filter to an imat_fn or imat_shp array, the only operations you can perform on function filters are logical combinations:

- Combining two filters with an and operation (`z1&z2`) creates a new filter that is satisfied only if *both* of the filters are satisfied;
- Combining two filters with an or operation (`z1|z2`) creates a new filter that is satisfied if *either* of the filters is satisfied;
- Logically negating a filter (`~z`) creates a new filter that is satisfied only if the original filter is *not* satisfied.

These logical operations can be nested to create arbitrarily complex filters.

Results

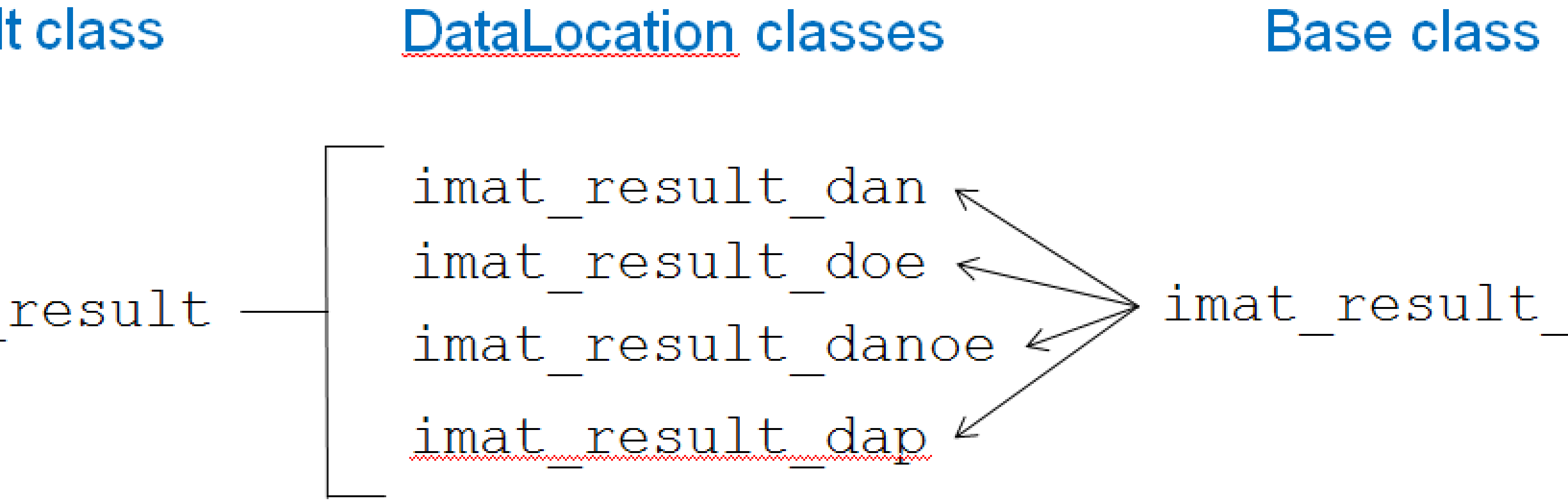
Results in IMAT refer to data that is output from a process. This could be output from a finite element solution from I-deas, or it could be results from a modal test. One specific subset of results can be stored in IMAT as an imat_shp. In addition to the actual data values, results also have many attributes that help describe the nature of this data. Attributes include ID Lines, analysis source, result type (acceleration, stress, etc.), and many others. IMAT keeps all of these attributes and the data together in a convenient result object data type.

IMAT+FEA utilizes the imat_result data type for result data imported from Nastran and Abaqus files. In these cases not all of the attributes available in the imat_result object are applicable, but using this data type offers consistency within MATLAB.

Class Organization

Before discussing how to use the imat_result class (data type), it may be helpful to describe how it is organized.

The imat_result class actually consists of a hierarchy of classes. Each individual result is defined in a class specific to the type of data of which it consists. All of the individual results are gathered in the top-level imat_result class, which contains most of the descriptive attributes. The class hierarchy is shown in the figure below.



One powerful feature of this class organization is that it is possible for the user to create their own result data location class and have it automatically included in the imat_result hierarchy. It only needs to be a subclass of `imat_result_dbase`, and define the methods and properties that the other result location classes define.

The table below describes all of the imat_result classes and their relationships.

`imat_result_dbase`	This is the base class that describes the properties and methods (including abstract) for each of the result location classes.
`imat_result_dan`	Defines the properties and methods for a single Data At Nodes result. Subclass of `imat_result_dbase`.

imat_result_doe	Defines the properties and methods for a single Data On Elements result. Subclass of `imat_result_dbase`.
imat_result_dap	Defines the properties and methods for a single Data At Points result. Subclass of `imat_result_dbase`.
imat_result_danoe	Defines the properties and methods for a single Data At Nodes On Elements result. Subclass of `imat_result_dbase`.
imat_result	Defines the properties and methods for the overall results. It defines the descriptive attributes common to all of the results. It can be a vector or matrix of results. Each individual result is one of the `imat_result_dbase`-derived classes above.

Data Format for Results

Assuming you have already worked with shapes in IMAT, you will find that results are treated in a similar way. An imat_result variable contains both the result data and various numeric and string attributes.

Results can be generated for various locations. Results can be data at a specific node, data on each of the nodes of an element (data at nodes on elements), data on an element, and others. Corresponding to the data locations and the data are data components which specify where the data values are located. For data at nodes, the component has three parts: the node number and the degree of freedom (component) on that node, such as X+, and the superelement ID. For data at nodes on elements, the component includes the element number, component (such as XX), and the node number on that element, the layer ID, superelement ID, and element type.

The imat_result object has properties for the attributes, the data (split into the various types of data), and other supporting attributes. To see the available properties and their expected data types or valid values, you can use the set function:

```
>> set(imat_result)
               Name: [ String ]
            IDLine1: [ String ]
            IDLine2: [ String ]
            IDLine3: [ String ]
            IDLine4: [ String ]
            IDLine5: [ String ]
          ModelType: [ Unknown | Structural | Heat transfer | Fluid flow ]
       AnalysisType: [ Unknown | Static | Normal mode | Complex eigenvalue first order
                       | Transient | Frequency response | Buckling
                       | Complex eigenvalue second order | Static non-linear  Craig-Bampton con-
straint modes
                       | Equivalent attachment modes | Effective mass modes | Effective mass mat-
rix 1
                       | Effective mass matrix 2 | Distributed Load Load Distribution | Dis-
tributed Load Attachment modes ]

         ResultType: [ Stress | Strain | Element Force | Temperature | Heat Flux
                       | Strain Energy | Displacement | Reaction Force | Kinetic Energy | Velocity
          ExpLength: [ Numeric value ]
           ExpForce: [ Numeric value ]
     ExpTemperature: [ Numeric value ]
            ExpTime: [ Numeric value ]
...
               Time: [ Numeric value ]
```

```
              Frequency: [ Numeric value ]
             Eigenvalue: [ Numeric value ]
         EigenvalueReal: [ Numeric value ]
         EigenvalueImag: [ Numeric value ]
              ModalMass: [ Numeric value ]
          ModalMassReal: [ Numeric value ]
          ModalMassImag: [ Numeric value ]
         ViscousDamping: [ Numeric value ]
      HystereticDamping: [ Numeric value ]
                 ModalA: [ Numeric value ]
             ModalAReal: [ Numeric value ]
             ModalAImag: [ Numeric value ]
                 ModalB: [ Numeric value ]
             ModalBReal: [ Numeric value ]
             ModalBImag: [ Numeric value ]
              Stiffness: [ Numeric value ]
          StiffnessReal: [ Numeric value ]
          StiffnessImag: [ Numeric value ]
          EffectiveMass: [ 6x1 numeric vector ]
    ParticipationFactor: [ 6x1 numeric vector ]
                   data: [ IMAT_RESULT_* object]

>>
```

The specific meaning of each attribute can be found in the reference section. In general, you will have access to the data coefficients, the component labels, and three types of attributes:

- *Numeric attributes*, which take on numeric values (e.g., *Frequency*);
- *String attributes*, which are set to character string values (e.g., *IDLine1*); and
- *List attributes*, which require you to select one value from a limited list (e.g., *ResultType*).

Each of the individual result DataLocation classes also has its own set of attributes, which are described in the reference section. You can see the list of attributes for each by using MATLAB's properties method, as shown below.

```
>> properties(imat_result_dan)        % Data At Nodes

Properties for class imat_result_dan:

    DataLocation
    NumberNodes
    NumberValues
    DataCharacteristic
    component
    node
    dir
    seid
    data
    numDisplay
```

You will see later how to examine and modify the data values and data attributes.

Arrays of Results

Just as for functions and shapes, multiple results can be arrayed in a single imat_result variable. Most often, you will work with a "column vector" of results (number of results by 1). The various elements of a result array need not be compatible, but most operations work most effectively when they are.

If you can think of a use for multidimensional arrays of results, IMAT will happily work with them.

Creating a Result in MATLAB

To create an imat_result from scratch, you use the imat_result constructor. To set the type of data stored in an individual result index, simply set the .data attribute to an object of the datatype you want to use. You can also pass this object into the constructor, as shown in the examples below.

```
>> r = imat_result                                      % Create a scalar imat_result

r =

1x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
1          Data At Nodes  Unknown    User defined

>> r = imat_result(3)                                   % Create a 3x1 imat_result

r =

3x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
1          Data At Nodes  Unknown    User defined
2          Data At Nodes  Unknown    User defined
3          Data At Nodes  Unknown    User defined

>> r = imat_result('ResultType','stress')               % Change the ResultType

r =

1x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
1          Data At Nodes  Unknown    Stress

>> r.data = imat_result_danoe                           % Change the DataLocation (different from
result object)

r =

1x1 IMAT_RESULT
Row  Name  DataLocation                ModelType  ResultType
---  ----  --------------------------  ---------  ------------
1          Data At Nodes On Elements  Unknown    Stress

>> r=imat_result('ResultType','stress',imat_result_danoe)     % Set the DataLocation directly in
the constructor (different from result object)
```

```
r =

1x1 IMAT_RESULT
Row  Name  DataLocation               ModelType  ResultType
---  ----  -------------------------  ---------  ------------
1          Data At Nodes On Elements  Unknown    Stress
```

After you create a result, you will want to set its attributes and data values. Click here to see how to do this.

It is also possible to convert an `imat_fn` or `imat_shp` into an `imat_result`, and vice versa. Simply pass the variable into the result constructor function as shown below.

```
>> f=imat_fn(3,'responsenode',1:3,'ordinate',reshape(1:18,6,3));    % Creates 3x1 imat_fn
>> r=imat_result(f);  % Creates 6x1 imat_result (one for each abscissa value in f)
```

Importing Results

There are several ways to get results from I-deas and other software packages into MATLAB. Results are transferred through Universal files in dataset 2414. They are also returned from readnas and readodb for the appropriate data types.

Universal file import with readunv

The IMAT toolbox allows you to read a wide variety of *universal files*. Universal files are generally ASCII text files that can be read and written by I-deas and other software packages. You can create a universal file containing results by accessing the Export menu in I-deas.

You create a result universal file in I-deas using the File/Export menu selection. Select the option "Simulation Universal File" option, and be sure to highlight the results.

To read a results universal file into MATLAB, use the readunv function in the IMAT toolbox. You can provide a filename argument, or call readunv with a wildcard or no argument to interactively select the file to read. Note that readunvwill by default read mode shape results in as an imat_shp. To read them in as a result, use the `'asresult'` optional input argument.

When transferring universal files between platforms, be sure to use an ASCII (text) file transfer protocol so that carriage returns get converted properly.

Exporting Results

You have one option for transmitting results to I-deas and other software packages.

Universal file export with writeunv

You can create a result universal file (dataset 2414) using the writeunv function in the IMAT toolbox. The first argument is a file-name, which can be a wildcard. The remaining arguments are one or more imat_result variables to be written to the universal file. You can also combine these variables with the other IMAT objects.

You read the shape into I-deas using the File/Import menu selection using the "Test Universal File" or the "Simulation Universal File" option, depending on what application you are in.

If you copy the universal file between platforms, you should be sure to use an ASCII file transfer protocol so that carriage returns get converted properly.

Accessing Result Data and Attributes

IMAT offers two methods for you to examine or modify the contents of an imat_result.

Accessing attributes

The simplest way to get information about a result is with the syntax *`r.attrib`*, where *`r`* is the name of the variable, and *`attrib`* is the name of the attribute you want. The possible attribute names are listed in the reference section of this document, or you can get a listing with the set function. Here are some examples:

```
>> r.frequency

ans =

         1.0
         2.1
       3.224
       27.56

>> r.name

ans =

    ' B.C. 2,DISPLACEMENT_11,LOAD SET 1'
    ' B.C. 2,REACTION FORCE_12,LOAD SET 1'
    ' B.C. 2,STRESS_13,LOAD SET 1'
    ' B.C. 2,STRAIN_14,LOAD SET 1'

>>
```

An alternative way to access shape information is with get. In general, the expression `get(`*`r`*`,'`*`attrib`*`')` is equivalent to the expression *`r.attrib`* as discussed above. However, the get function also allows you to extract multiple attributes in one call. Here's an example:

```
>> g=get(r,'idline1','loadset','datalocation','resulttype')

g =

         IDLine1: {4x1 cell}
         LoadSet: [4x1 double]
    DataLocation: {4x1 cell}
      ResultType: {4x1 cell}

>>
```

When you ask for multiple attributes, get returns a structure with field names equal to the attributes you selected. The value in each field is the value of the corresponding attribute, as discussed above. If you do not specify any attributes at all, then get will return all of the data attributes. After a call to get, you can refer to the individual returned values by name, provided you use the proper combination of upper and lower case to refer to the fields:

```
>> r.ResultType

ans =
    'Displacement'
    'Reaction Force'
    'Stress'
    'Strain'

>>
```

These examples illustrate that numeric attributes like *Frequency* are returned in a numeric array the same size as `r`. String-valued attributes like *Name* as well as list attributes are returned either as a character string (if `r` is a single result or in a *cell array of strings* if `r` is a vector or array of results. (For more information on cell arrays and cell arrays of strings, consult the MATLAB documentation.)

Accessing components and data

You can access the components and data of a given imat_result through the `.data` property, which contains the result DataLocation object for a given imat_result. There are several ways to access the components. The imat_result class does allow you to access some of the underlying DataLocation result properties, but for complete access you will need to access the underlying objects directly.

To retrieve the entire component matrix directly, use the `.component` property, as shown below. However, you cannot set the component matrix this way.

```
>> r

r =

2x1 IMAT_RESULT
Row  Name                               DataLocation               ModelType   ResultType
---  ---------------------------------  -------------------------  ----------  ------------
1    SUBCASE=1, LOAD ID=1, TYPE=STRESS  Data At Nodes On Elements  Structural  Stress
2    SUBCASE=2, LOAD ID=2, TYPE=STRESS  Data At Nodes On Elements  Structural  Stress

>> r(1).data.component(1:10,:)

ans =
     1     1     11     1     0     0
     1     1     12     1     0     0
     1     1     22     1     0     0
     1     1     13     1     0     0
     1     1     23     1     0     0
     1     1     33     1     0     0
     1     1     11     2     0     0
     1     1     12     2     0     0
     1     1     22     2     0     0
     1     1     11     2     0     0

>> r(1).data.data(1:3)

ans =
     1.23435
     3.65766
           0
```

It can be more useful to access the components by their individual columns. Each result DataLocation object has a different set of property names for the component columns, since each one has a different set of columns. Below are some examples, based on the result shown above. You can both get and set the component list this way. When setting these attributes, the object will automatically extend or truncate all of the component columns and the data if you change the number of values.

```
>> r(1).data.element(1:4)

ans =
     1
     1
     1
     1

>> r(1).data.comp(1:4)

ans =
     11
     12
     22
     13
```

Using the individual data component columns can be a very handy way of extracting a subset of results. For example, to extract node numbers 1 and 2 and direction X+ from a Data At Nodes result, you can use the following code snippet.

```
>> ind = ismember(r(1).data.node,[1 2]) & r(1).data.dir == 1;         % Build a logical index vec-
tor that matches nodes 1 and 2, direction 1


>> r(1).data.component(ind,:)                                         % Extract the component sub-
matrix

ans =
           1           1           0
           2           1           0

>> r(1).data.data(ind)                                                % Extract the corresponding
data

ans =
         1.1
         1.2
```

Data attributes of an imat_result return a single numeric or string value for each element of a result array. The *Component* and *Data* "properties", on the other hand, each return an array of values for each element of the result array. When you ask for the *.data.data* attribute (data values for each of the results), you will get a multidimensional array whose first dimension corresponds to the number of data values, and whose remaining dimensions match the size of the imat_result array.

The same thing happens when you access the *.data.component* attribute (the data components): an extra dimension is added as the first dimension of the result.

If not all of the results have the same number of data values or components, IMAT sets the first dimension of the result to the largest of any of the results in the array. For those results with less data than the maximum, the extra elements are filled with NaN (i.e., "not a number") values.

Examples of this are shown below.

```
>> r

r =

2x1 IMAT_RESULT
Row  Name                              DataLocation               ModelType   ResultType
---  --------------------------------  -------------------------  ----------  ------------
1    DERIVED RESULT                    Data On Elements At Nodes  Structural  User defined
2    SUBCASE=2, LOAD ID=2, TYPE=STRESS  Data At Nodes On Elements  Structural  Stress

>> r.data.component

ans(:,:,1) =

      1     1     1     0   NaN   NaN
      1     1     2     0   NaN   NaN
      1     1     3     0   NaN   NaN

ans(:,:,2) =

      1     1    11     0     0     0
      1     1    12     0     0     0
      1     1    22     0     0     0

>> r.data.data

ans =
     1.23435    6.53175
     3.65766   -3.65766
           0          0
```

Partitioning results

Sometimes it is useful to partition results to a subset of components. One way to achieve this is to index into the component matrixas shown in the examples above, and then use `setComponents` to build a new result. However, that can be cumbersome. IMAT provides partition methods to wrap this capability into an easy-to-use method. In addition, it allows you to use imat_ctrace, imat_group, and numeric submatrices, depending on the result DataLocation. Not all data types support all of these input types, and the way some of these partitioning inputs are interpreted vary with the result DataLocation.

In the following example, we will start with a scalar imat_result containing Data At Nodes data. We will then partition the `imat_result_dan` object in the `.data` property using an imat_ctrace and a numeric submatrix. Following that, we will show that you can also partition the result using the partition method on the imat_result itself. Please note that if your result contains multiple DataLocations, this method may fail because not all partitioning input types are recognized by each of the different DataLocations.

```
>> r
r =
1x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
  1        Data At Nodes  Unknown    User defined

>> dan=r.data                                   % Extract the Data At Nodes object
dan =
Data At Nodes:  3DOF global translation vector
================================================
      Node      Dir     SEID               Data
------------------------------------------------
         1        1        0                   1
         1        2        0                   2
         1        3        0                   3
        11        1        0                   4
        11        2        0                   5
        11        3        0                   6
------------------------------------------------
Number of Values: 6   Number of Nodes: 2
================================================

>> ct=imat_ctrace('1y','11z')                    % Create an imat_ctrace to use for partitioning
>> dan.partition(ct)                             % Partition at the imat_result.data level
ans =
Data At Nodes:  Unknown
================================================
      Node      Dir     SEID               Data
------------------------------------------------
         1        2        0                   2
        11        3        0                   6
------------------------------------------------
Number of Values: 2   Number of Nodes: 2
================================================

>> dan.partition({[1 2; 11 3]})                  % Look for matches in the first 2 columns; notice
that the submatrix is passed in as a cell containing the submatrix
ans =
Data At Nodes:  Unknown
================================================
      Node      Dir     SEID               Data
------------------------------------------------
         1        2        0                   2
        11        3        0                   6
------------------------------------------------
Number of Values: 2   Number of Nodes: 2
================================================

>> r2=r.partition(ct)                            % You can also partition at the imat_result
level, but this only works if all of the DataLocations support the supplied partitioning input
r2 =
1x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  --------------  ---------  ------------
  1        Data At Nodes  Unknown    User defined

>> r2.data
ans =
Data At Nodes:  Unknown
================================================
      Node      Dir     SEID               Data
------------------------------------------------
         1        2        0                   2
        11        3        0                   6
------------------------------------------------
Number of Values: 2   Number of Nodes: 2
================================================
```

The numeric submatrix method of partitioning has the additional advantage of being able to specify wildcards by specifying `inf` for a given component value. In the example below, we are specifying two different wildcard matches. In the first row, we are asking for all components whose direction is 2. In the second row, we are asking for all rows with a node number of 11. Notice that the two matches overlap, but only the unique match is returned since the component rows must be unique in a given DataLocation object.

```
>> dan.partition({[inf 2; 11 inf]})                 % Look for matches in the first 2 columns
using wildcards
ans =
Data At Nodes:  Unknown
================================================
      Node       Dir      SEID                Data
------------------------------------------------
         1         2         0                   2
        11         1         0                   4
        11         2         0                   5
        11         3         0                   6
------------------------------------------------
Number of Values: 6   Number of Nodes: 2
================================================
```

Subscripting into data locations

Another way to partition results is to use subscripting. You can subscript into the individual `imat_result_*` objects using logical numeric subscripting, as well as the other input types supported by the partition methods described above. In fact, IMAT uses the partition method directly when you apply subscripting.

Generally, using numeric indices extracts the *i*th row(s) from the result, as shown in the example below. Currently this subscripting only operates on the underlying data location objects, so to use it with an imat_result you must first extract the data location object.

```
>> r
r =
1x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
  1        Data At Nodes  Unknown    User defined

>> dan=r.data                                    % Extract the Data At Nodes object
dan =
Data At Nodes:  3DOF global translation vector
================================================
      Node       Dir      SEID                Data
------------------------------------------------
         1         1         0                   1
         1         2         0                   2
         1         3         0                   3
------------------------------------------------
Number of Values: 3   Number of Nodes: 1
================================================

>> dan(2)                                        % Extract the 2nd row using numeric subscripting
```

```
ans =
Data At Nodes:  Unknown
================================================
      Node      Dir     SEID                Data
------------------------------------------------
         1        2        0                   2
------------------------------------------------
Number of Values: 1   Number of Nodes: 1
================================================

>> dan(dan.dir==3)                              % Extract the Z direction rows using logical sub-
scripting
ans =
Data At Nodes:  Unknown
================================================
      Node      Dir     SEID                Data
------------------------------------------------
         1        3        0                   3
------------------------------------------------
Number of Values: 1   Number of Nodes: 1
================================================

>> dan('1x')                                   % Extract the 1X result (only valid with Data At
Nodes)
ans =
Data At Nodes:  Unknown
================================================
      Node      Dir     SEID                Data
------------------------------------------------
         1        1        0                   1
------------------------------------------------
Number of Values: 1   Number of Nodes: 1
================================================

>> dan(imat_ctrace('1x','1z'))                 % Extract the 1X and 1Z results using an imat_
ctrace
ans =
Data At Nodes:  Unknown
================================================
      Node      Dir     SEID                Data
------------------------------------------------
         1        1        0                   1
         1        3        0                   3
------------------------------------------------
Number of Values: 2   Number of Nodes: 1
================================================
```

Changing attributes

To change an attribute of a result, you use the syntax `r.attrib=value`. Numeric attributes should be set to a numeric array with dimension matching `r`. String-valued attributes and list attributes should be set to a cell array of strings, and the cell array should be the same dimension as `r`. There are two special numeric attributes that should have a first extra dimension of 6. These are

EffectiveMass and *ParticipationFactor*.If you set the *EffectiveMass*, or *Participation* attribute to a single column vector, then the same vector is used for all elements of *r*.

A special case is made for times when you want to set all results in an imat_result array to have the same attribute. In this case, you can set the *value* to be a scalar numeric value or a simple character string. The same value will then be applied to all of the results in the array.

Changing some attributes will affect others. For example, if you change *Frequency* or *ViscousDamping*, the *Eigenvalue* attribute will change.

The set function is an alternative way to change attributes or result data. The syntax is `r2=set(r,'attrib',value,...)`. You can set one or many attributes. The result *r2* is a duplicate of *r*, with the exception of the attributes you specify. set also accepts a structure, like the one that get returns.

Changing components and data

Setting the data attribute is straight-forward. Simply set the `.data.data` property with the new data values, as shown in the example below. The *Data* attribute should have an extra first dimension as discussed above. If the number of values changes for a given result, the component list will change accordingly.

```
>> r

r =

2x1 IMAT_RESULT
Row  Name                               DataLocation               ModelType   ResultType
---  ---------------------------------  -------------------------  ----------  ------------
1    DERIVED RESULT                     Data On Elements At Nodes  Structural  User defined
2    SUBCASE=2, LOAD ID=2, TYPE=STRESS  Data At Nodes On Elements  Structural  Stress

>> r(1).data.data = [1 2 3 4]';
```

Setting components can be done one of two ways. The most natural way is to set the `.data.component` property directly. The second way involves using the `setComponents` method for each individual *DataLocation* object. There are reasons why you might choose one method over the other. One reason to set `.data.component` directly is its ease of use. One potential disadvantage is that when you set the components this way, the data values are set to 0. This is because the component matrix may or may not have the same size as what was there previously, and it may or may not have the same components. Using `setComponents` allows you to set the data values at the same time as the components. It also lets you specify the components in the internal compressed format, which is discussed in more detail here. Note that internally, IMAT utilizes `setComponents` when you set the `.data.component` property directly.

Setting the `.data.component` matrix directly can be done both at the imat_result level, and individually at the *DataLocation* object level. In the example below, we will set the data components at the imat_result level by first extracting the components, and then setting them as a subset of the original components. The component matrix will be a multi-dimensional matrix. For example, if your imat_result is a 2x1 vector of Data At Nodes results, and each result has 12 component rows, your component matrix will be 10x3x2. If your imat_result is 1x2, then your component matrix will be 10x3x1x2. If your result contains mixed result types (as shown in the example below), the first 2 dimensions of the component matrix will be the largest of the dimensions of each individual results, and unused rows and columns must contain Nan values.

```
>> r

r =

2x1 IMAT_RESULT
Row  Name                               DataLocation               ModelType   ResultType
```

```
---  ---------------------------------  -------------------------  ----------  ------------
1    DERIVED RESULT                     Data On Elements At Nodes  Structural  User defined
2    SUBCASE=2, LOAD ID=2, TYPE=STRESS  Data At Nodes On Elements  Structural  Stress

>> cc = r.data.component                          % Extract the component matrix, which is 3x6x2
because r is 2x1

cc(:,:,1) =

      1     1     1     0   NaN   NaN
      1     1     2     0   NaN   NaN
      1     1     3     0   NaN   NaN

cc(:,:,2) =

      1     1    11     0     0     0
      1     1    12     0     0     0
      1     1    22     0     0     0

>> r.data.data                                    % Each column contains the data values for that
result

ans =
     1.23435    6.53175
     3.65766   -3.65766
           0          0

>> r.data.component = cc(1:2,:,:);                % Set the component matrix using the first 2
rows of each
>> r.data.component

cc(:,:,1) =

      1     1     1     0   NaN   NaN
      1     1     2     0   NaN   NaN

cc(:,:,2) =

      1     1    11     0     0     0
      1     1    12     0     0     0

>> r.data.data                                    % Notice that the data values are now 0

ans =
           0          0
           0          0


>> cc2 = cc(:,1:4,1)                              % Extract the component submatrix for the first
result

cc2 =

      1     1     1     0
      1     1     2     0
      1     1     3     0
```

```
>> r(1).data.component = cc2;                     % Set the component matrix for the first result
```

Using the `setComponents` method allows you to set the components and data at the same time. The `setComponents` method accepts different input arguments for each different *DataLocation* object type, so please see the help for each.

The *DataLocation* objects all store the component lists internally in a compressed format if at all possible, as described below. `setComponents` will automatically attempt to compress component lists that are supplied in an uncompressed form. To store the component lists in compressed form, additional internal attributes are needed. These show up as additional input arguments to the `setComponents` method for some of the DataLocation types. If you wish to supply components in compressed form to `setComponents` directly, the `getComponents` method is a very useful way to see how the individual component columns and related attributes are stored internally. The output of `getComponents` can be passed directly back in to `setComponents`.

The example below shows how to create a new Data At Nodes object and populate its components and data. The inputs are supplied in compressed form. The result components consist of directions 1-6 repeated for nodes 1-100.

```
>> d = imat_result_dan;                             % Create a new Data At Nodes object
>> d = setComponents(d,1:100,1:6,0,true,1:600);     % Supply the components and data in compressed
form

>> r(3).data = d;                                   % Place the data into the imat_result object
```

Editing specific components

You can use the subscripting capabilities of the imat_result_* data location objects for individual component and data assignments. Some examples are shown below.

```
>> r
r =
1x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
  1        Data At Nodes  Unknown    User defined

>> dan=r.data
dan =
Data At Nodes:  3DOF global translation vector
================================================
      Node      Dir     SEID                Data
------------------------------------------------
         1        1        0                   1
         1        2        0                   2
         1        3        0                   3
------------------------------------------------
Number of Values: 3   Number of Nodes: 1
================================================

>> dan(2).data = 22
ans =
Data At Nodes:  Unknown
================================================
      Node      Dir     SEID                Data
------------------------------------------------
         1        1        0                   1
         1        2        0                  22
         1        3        0                   3
------------------------------------------------
Number of Values: 3   Number of Nodes: 1
================================================

>> dan(dan.dir==3).seid = 1
ans =
Data At Nodes:  Unknown
================================================
      Node      Dir     SEID                Data
------------------------------------------------
         1        1        0                   1
         1        2        0                  22
         1        3        1                   3
------------------------------------------------
Number of Values: 3   Number of Nodes: 1
================================================


>> dan(imat_ctrace('1x','1z')).node = 2
ans =
Data At Nodes:  Unknown
================================================
      Node      Dir     SEID                Data
------------------------------------------------
         2        1        0                   1
         1        2        0                  22
         2        3        1                   3
------------------------------------------------
Number of Values: 2   Number of Nodes: 2
================================================
```

Converting between DataLocations

In some cases it is possible to convert results from one DataLocation to another. This process is called typecasting, because you are converting one object type to another.

```
>> dan = imat_result_dan;                        % Create an empty Data At Nodes object
>> dan.data=1:12                                 % Put some data into it

Data At Nodes:  3DOF global translation vector
================================================
      Node       Dir      SEID              Data
------------------------------------------------
         1         1         0                 1
         1         2         0                 2
         1         3         0                 3
         2         1         0                 4
         2         2         0                 5
         2         3         0                 6
         3         1         0                 7
         3         2         0                 8
         3         3         0                 9
         4         1         0                10
         4         2         0                11
         4         3         0                12
```

```
------------------------------------------------
Number of Values: 12     Number of Nodes: 4
================================================


>> r = imat_result(dan)                          % Place the data into the imat_result object

r =

1x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
  1        Data At Nodes  Unknown    User defined

>> r.data = imat_result_doe(r.data)              % Convert the Data At Nodes result into Data On
Elements

r =

1x1 IMAT_RESULT
Row  Name  DataLocation      ModelType  ResultType
---  ----  ----------------  ---------  ------------
  1        Data On Elements  Unknown    User defined

>> r.data                                        % View the typecasted data

Data On Elements
================================================
   Element     Layer      SEID              Data
------------------------------------------------
         1         1         0                 1
         1         2         0                 2
         1         3         0                 3
         2         1         0                 4
         2         2         0                 5
         2         3         0                 6
         3         1         0                 7
         3         2         0                 8
         3         3         0                 9
         4         1         0                10
         4         2         0                11
         4         3         0                12
------------------------------------------------
Number of Values: 12     Number of Elements: 4
================================================
```

Internal Storage

Since the component list can be extremely large for a given result, where possible the imat_result class will store the component matrix in a compressed form. If it cannot, it will store them in uncompressed form. When the component list is set or modified, it will always attempt to compress the component list. In addition, the component list is stored internally in separate columns as INT32.

To access the compressed component definition, use `getComponents`, as shown below. This component definition can be passed in to `setComponents` directly.

```
>> c = getComponents(r(1).data)

ans =

      [208x1 int32]    [816x1 int32]     [208x1 double]   [6x1 int32]   [1616x1 int32]     [816x1
int32]     [0]     [0]     [1]

>> r(1).data = setComponents(r(1).data,c{:});
```

Math Operations

You can perform any MATLAB operation on an imat_result simply by extracting its data values into a numeric array, performing the desired operation, and creating a new imat_result with the resulting data. However, some operations can be performed directly on imat_result objects, using methods built into IMAT. Here is a summary of the available operations:

Unary Operations	
+r	Unary plus (leaves `r` unchanged)
-r	Unary minus (negate the data values of `r`)
real(r)	Take real part of the data values of `r`
imag(r)	Take imaginary part of the data values of `r`
conj(r)	Take complex conjugate of the data values of `r`
abs(r)	Take absolute value of the data values of `r` (or modulus if `r` is complex)
Binary Operations	
r+scalar, r-scalar	Add, subtract scalar value to all data values
r.*scalar, r./scalar	Multiply or divide data values by scalar value
r+t, r-t	Add or subtract two results. Operation is performed on data values only if data locations and number of values match between `r` and `t`.
r.*t, r./t	Termwise multiply or divide two shapes. Operation is performed termwise only if nodes of `r` and `t` match.

Finite Element (FEM) Geometry

Finite element model geometry data is partially supported by IMAT. The primary purpose for FEM support with IMAT is visualization, both of the FEM geometry and of mode shapes and contour results. Current support is limited to FEM entities consisting of coordinate systems, nodes, elements, and tracelines. Nodes are used to define point locations in 3D space, and elements and tracelines are used to define connectivity between the node for visualization purposes. Coordinate systems allow for definition of nodes in a local orientation.

IMAT stores FEM geometry as individual classes (data types) for each of the FEM entity types, with an overall FEM class that contains each of the individual entity objects. All of the data found in Universal files for these data types is stored in these entities.

Data Format for FEM Geometry

The primary class (imat_fem) containing the coordinate system, node, element, and traceline data has four properties (`cs`, `node`, `elem`, and `tl`). Each of these properties, in turn,contains an object that contains the data specific to that data type. For example, the `.node` property of the imat_fem contains an IMAT_NODE object that contains all of the nodes and their associated properties and methods. For a complete list of the various classes and descriptions of their contents, please refer to the FEM Data Format Reference.

Below is an example of the imat_fem object and its contents. These data classes operate very much like MATLAB structures.

```
>> fem                                                       % Each type of FEM entity is stored
as an object in the imat_fem

fem =
IMAT_FEM Finite Element Model
        76 coordinate systems
      5379 nodes
      5611 elements
         0 tracelines

>> fem.cs                                                    % Extract and view the imat_cs object
containing the coordinate systems

ans =

IMAT_CS - Coordinate systems
      ID Name                                     Type Color
-------- ---------------------------------------- ---- -----
       4 CS4                                         0     8
       5 CS5                                         0     8
       6 CS6                                         0     8
 . . .
      76 CS76                                        0     9

>> fem.node                                                  % Extract and view the imat_node
object containing the nodes

ans =

IMAT_NODE - Nodes
      ID     Def CS    Disp CS   Store CS            X            Y            Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
       1          4          4          4         -4.5         -7.5          -14       11
       2          4          4          4         -2.5         -7.5          -14       11
       3          4          4          4         -4.5         -7.5        -11.5       11
       4          4          4          4         -2.5         -7.5          -14       11
 . . .
    5379         76         76         76            0          -27          -12       11

>> fem.tl                                                   % Extract and view the imat_tl object
containing the tracelines
IMAT_TL - Tracelines
 --- Empty ---
```

```
>> fem.elem

IMAT_ELEM - Elements
      ID       Type      Color    # Nodes
-------- ---------- ---------- ----------
       1         33          1          4
       2         33          1          4
       3         33          1          4
       4         33          1          4
 . . .
    5611        902          4         33

>>
```

Importing FEM Data

Currently, FEM data can be read through Universal files and Nastran OP2 and bulk data files with IMAT+FEA. Universal files provide a way to move function data from one platform to another.

The IMAT toolbox allows you to read a wide variety of Universal files. Universal files are generally ASCII text files that can be read and written by I-deas and other software packages. You can create a Universal file containing FEM geometry by accessing the File/Export menu in I-deas. Select the option "Test Universal File" (or "Simulation Universal File" from the Simulation task). You may export more than one FEM to the same Universal file. You then must choose the universal filename and select the FEMs you want to export.

To read a universal file containing FEM geometry into MATLAB, use the readunv function in the IMAT toolbox. You can provide a filename argument, or call readunv with a wildcard or no argument to interactively select the file to read. Generally, readunv supports both the old and new dataset formats for geometry.

To read FEM geometry from Nastran files, use the readnas function.

FEM Methods

IMAT provides several methods to operate on the FEM data. Some of the methods are implemented on the imat_fem class as well as the individual imat_cs, imat_node, imat_elem, and imat_tl classes. To determine what methods are available for each object, use Matlab's methods function. For example, to see the available imat_fem methods, type `methods(imat_fem)` in the Matlab command window. Generally, methods that operate on the FEM as a whole are methods of the imat_fem class. For example, the xform method transforms node coordinates between coordinate systems. It is an imat_fem method since it uses both nodes and coordinate systems. The partition method partitions a FEM down to the entities that are associated with the supplied partitioning entity such as an imat_ctrace. It partitions each of the individual FEM entities stored in the imat_fem.The cat method concatenates two or more FEMs into a single FEM.

By contrast, some methods and properties are specific to that entity type. For example, to determine the number of nodes in your FEM, use `length(fem.node)`.

For example, let's say that you read in a large analysis FEM from a Universal file, but the mode shapes that you want to plot only contain a subset of the geometry. You may want to partition your FEM down to the nodes corresponding to your test degrees of freedom. The following MATLAB session illustrates how to do this.

```
>> whos
Name       Size          Bytes  Class
ct       975x1           39582  imat_ctrace
fem        1x1         2700458  imat_fem
```

```
>> fem                          % Here is the original FEM read in from a Universal file

fem =
IMAT_FEM Finite Element Model
        76 coordinate systems
      5379 nodes
      5611 elements
         0 tracelines

>> numel(unique(node(ct)))        % The partitioning coordinate trace has 275 nodes

ans =
     275

>> fem2=partition(fem,ct)        % Partition the FEM. The elements are also partitioned to those
fully defined by the remaining nodes.
Partitioning nodes
Partitioning elements
Partitioning tracelines

fem =
IMAT_FEM Finite Element Model
        76 coordinate systems
       275 nodes
       361 elements
         0 tracelines

>>
```

FEM Subscripting

FEM objects, including the individual coordinate system, node, element, and traceline objects, are scalar objects. However, the individual objects can obviously contain many entities. For example, a single IMAT_NODE object can contain many nodes.

There are several ways of accessing the information in the FEM and FEM entity objects. The most straight-forward way to access the properties using "dot" (structure) notation. You can also access methods this way, following the Matlab convention. The following code snippet shows several examples of this.

```
>> fem

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 3tracelines

>> fem.tl            % Access the scalar traceline object, which contains 3 tracelines

ans =

IMAT_TL - Tracelines
      ID Description                                   Color    # Nodes
-------- ---------------------------------------- ---------- ----------
       1                                                   8         10
       2                                                  8         31
       3                                                  8         20

>> fem.tl.color      % Extract the traceline colors, which are a property of the traceline object

ans =

         8
         8
         8

>> n=imat_node;      % Create an empty node object
>> n=n.add(1);       % Call the ADD method to add a node
>> n=add(n,1);       % Another way to call the ADD method
>>
```

You can use parenthesis "()" subscripting with the individual FEM entity objects. These subscripts return the entities referenced by those indices. To extract entities by their ID, use curly brace "{}" notation instead. This is equivalent to using the KEEP method. See the following code snippet for examples of this.

```
>> node = fem.node;
>> setdisplay(node,3)

 ans =

IMAT_NODE - Nodes
   30 nodes

>> n([3 5])         % Extract the 3rd and 5th nodes -- note the parenthesis notation

ans =

IMAT_NODE - Nodes
      ID     Def CS    Disp CS   Store CS            X            Y            Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
     103          0          0          0      -0.2032      0.41487            0       11
     105          0          0          0      -0.1016      0.43603            0       11

>> n{[103 105]}     % Extract node IDs 103 and 105 -- note the curly brace notation

ans =

IMAT_NODE - Nodes
      ID     Def CS    Disp CS   Store CS            X            Y            Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
     103          0          0          0      -0.2032      0.41487            0       11
     105          0          0          0      -0.1016      0.43603            0       11

>>
```

Curly brace "{}" notation has a special meaning with an imat_fem. If you use this notation and pass in an imat_group object, the FEM will be partitioned to the entities in the group. Groups are covered in the next section.

```
>> fem

fem =
IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 tracelines

>> group=group(1)

group =

IMAT_GROUP - Groups
GROUP(1): 1 - EVERYTHING
      ID     Type Name
-------- -------- ---------------
       1        1 coordinate system
       1        7 node
       2        7 node
       3        7 node
       4        7 node
       1        8 element
-------- -------- ---------------
       1 coordinate system
       4 nodes
       1 element

>> fem{group}

ans =

IMAT_FEM Finite Element Model
         1 coordinate system
         4 nodes
         1 element
         0 tracelines

>>
```

Adding and removing FEM entities

You can add or remove FEM entities using one of three methods: add, keep, or remove. These methods exist for the IMAT_CS, IMAT_NODE, IMAT_ELEM, and IMAT_TL classes.

The following examples show how to add and remove FEM entities.

```
>> fem

fem =
IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 tracelines

>> fem.node = fem.node.add(100)           % Add node 100, accepting defaults for the node prop-
erties
```

```
fem =

IMAT_FEM Finite Element Model
         1 coordinate system
        31 nodes
        10 elements
         3 tracelines

>> fem.node{100}                          % List node 100

ans =

IMAT_NODE - Nodes
      ID     Def CS    Disp CS   Store CS            X            Y            Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
     100          0          0          0            0            0            0       11

>> fem.node = fem.node.remove(100)        % Remove node 100

fem =
IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 tracelines

>>
```

Understanding Coordinate Systems In IMAT

Coordinate systems can be confusing, especially since different analysis programs use different naming conventions, and different file formats store their data in different ways. The purpose of this section is to hopefully reduce some of the confusion by defining the naming conventions that IMAT uses. It will also describe how some of the IMAT-supported file formats store their data. Finally, it will provide some simple examples of how one might want to use the coordinate transformation capabilities provided by IMAT.

Naming Conventions

Different solvers use different naming conventions for the different classes of coordinate systems. Unfortunately, the same terminology can have different meanings in different contexts. The following table outlines different coordinate system labels and shows their equivalence across different platforms.This documentation uses the IMAT naming convention when describing coordinate systems and results.

IMAT	Nastran	I-deas	Description
global	basic	global or Part	The global cartesian system, or in other words the reference coordinate system.
local	global	local	A local coordinate system that defines the coordinate directions for a specific location. This is always referenced back to the global cartesian system. A transformation matrix and offset defines the orientation of this system relative to the global.
definition	definition	definition	A local coordinate system that specifies the coordinate system in which the node's coordinates are defined in software such as Nastran. When creating a node in IMAT, the coordinates of the node are entered in the stor-

			age CS not the definition CS. See more below.
displacement	displacement	displacement	A local coordinate system that specifies the coordinate system in which displacement output is generated.
storage	-	-	The coordinate system in which the X, Y, and Z coordinates of the node are displayed in IMAT. This is also the coordinate system in which the node coordinates are supplied when creating or adding nodes.

This is described in the FEM Data Format Reference, but is worth highlighting here. The `node.cs` property is an nx3 matrix of coordinate system identifiers. The first column specifies the definition coordinate system for each node. Column 2 specifies the displacement coordinate system. Column 3 is the storage coordinate system, and is specific to IMAT. It specifies the coordinate system in which the node coordinates (found in the `.coord` property) are actually stored, created, and displayed. The storage coordinate system does not have to be the same as the displacement or definition system for that node, though in many cases it will be the same as the definition system.

The primary reasons the storage coordinate system exists in IMAT is to offer flexibility when transforming coordinate systems, and to preserve information. For example, let's say your nodes coordinates are stored in local coordinates and you transform them to global Cartesian. The storage coordinate system for each node will be set to 1, but the definition and displacement coordinate systems would stay what they were, thus preserving this information for future tasks such as exporting to a node format where you want the coordinates specified in the definition system. If IMAT did not have a storage coordinate system, the definition system would have to change on transform to the coordinate system(s) you are transforming to.

File Formats

Different file formats assume different coordinate system storage conventions. It is very important that you know what coordinate system(s) your data is specified in when working with this data in IMAT.

Universal files for the most part store their information in the global coordinate system. This includes FEM nodes as well as shape coefficients. Thus when you import either entity from a Universal file, the coordinate system that defines this data is 1. This is why the 3rd column of the `node.cs` matrix (storage CS) is 1 for FEM nodes imported from a Universal file.

Shape Associated Data Files (ASH) store their shape coefficients in the local coordinate system. This is because these shape files are typically generated from a curve-fit of Frequency Response Functions. These FRF of course measure the response of a structure in the direction in which the accelerometer was installed, which is by definition a local coordinate system. In some cases tests are designed so that the accelerometers are all lined up with a previously defined global coordinate system.

Nastran stores shape coefficients in either local or global coordinates, depending on the file format. PCH files by default will store displacement shape coefficients in local coordinates. It is possible to write them in different systems using the appropriate PARAM. OP2 files generally store shape coefficients in the OUGV1, BOUGV1, or BOPHIG datablock name. The easy way to remember what coordinate system the shape coefficients are stored in is that if the datablock name starts with 'B' ('basic'), the shape coefficients are stored in global coordinates. Thus OUGV1 stores its coefficients in local coordinates. However, it is important to note that the default option for readnas is to transform displacement coefficients to the global (basic) coordinate system. This can be controlled either through the Options button on the OP2 directory form in readnas, or through an input argument to readnas. Please note that if the coefficients are already in global coordinates in the OP2 file (datablock begins with 'B'), they will be imported into IMAT in global coordinates regardless of the options setting.

Transforming Between Different Coordinate Systems

Please see the documentation for the most up-to-date specific documentation on how to use the coordinate system transformation functions.

Because the data you import into IMAT may not be in the coordinate systems with which you wish to use the data, IMAT provides transformation methods for FEM (nodes), imat_shp, and imat_fn. These transformation methods offer great flexibility in what coordinate systems you transform from and to, but their basic use is fairly simple. In this section we will show a few examples of actions you might want to perform.

In the first example, we will import a FEM from a Universal file and shapes from an ADF, and we will transform the shape coefficients to the global cartesian coordinate system.

```
>> fem=readunv
Universal file written in IN units
Read 1 FEMs

fem =
IMAT_FEM Finite Element Model
         8 coordinate systems
       371 nodes
       528 elements
         4 tracelines

>> shp=readadf('shapes.ash');
ADF Name: shapes.ash
Units : <7> IN
Begin processing...
19 records processed...
End processing...

>> shpg=xform(shp,fe,1);
Shape 1: Transforming LOCAL->GLOBAL

>>
```

Note that IMAT treats shapes and functions defined in cylindrical and spherical coordinate systems specially. It assumes that the actual coefficients are really cartesian values in a cartesian system that aligns with the cylindrical or spherical coordinate system at that particular node location, and thus transforms the data accordingly.

To better understand how coordinate systems work in IMAT, consider the following 2-node example. First, a default imat_cs is created. This object contains the global Cartesian coordinate system. Next, a local coordinate system is created such that it is oriented the same way as the global Cartesian system but translated one unit in the X direction. Node 1 is created and stored at the global Cartesian origin since its storage CS is 1. Node 2 is created at the origin of the local coordinate system, since its storage CS is 2.

```
>> cs = imat_cs();                                   % creates imat_cs with the global Cartesian sys-
tem
>> cs = cs.add(2,'Translated',1,1,[eye(3);1,0,0])
cs =

IMAT_CS - Coordinate systems
ID       Name                                     Type Color
-------- ---------------------------------------- ---- -----
       0 Global cartesian                            0    12
       2 Translated                                  1     1
2 coordinate systems

>> fem = imat_fem(cs);                                % create an imat_fem that uses the newly
```

```
created imat_cs
>> fem.node = fem.node.add(1,[2 0 0],1,[0 0 0]);
>> fem.node = fem.node.add(2,[2 0 2],1,[0 0 0]);
>> fem.node

ans =

IMAT_NODE - Nodes
ID           Def CS    Disp CS   Store CS             X            Y            Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
       1          2          0          0            0            0            0        1
       2          2          0          2            0            0            0        1
2 nodes
```

When the nodes are created, the coordinates are entered in the storage system, NOT the definition system. For node 1 the storage system is global Cartesian (basic), and for node 2 the storage system is the local system. The node coordinates are listed in their storage coordinate systems.

Next let us transform the FEM to the global cartesian system.

```
>> femg = xform(fem,0);     % equivalent to xform(fem); since the storage CS is local for node 2,
xform assumes TO is global cartesian
Nodes: Transforming LOCAL->GLOBAL
>> femg.node

ans =

IMAT_NODE - Nodes
ID           Def CS    Disp CS   Store CS             X            Y            Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
       1          2          0          0            0            0            0        1
       2          2          0          0            1            0            0        1
2 nodes
```

In `femg` the node coordinates are stored and displayed in the global Cartesian coordinate system. To transform the FEM geometry to local (definition) coordinates, simply do the following. You do not need to specify the TO coordinate system because the storage coordinate system is global Cartesian, so xform assumes that you want to transform to the definition system. Note that even though xform assumes your intentions, it is probably helpful to be explicit about which coordinate system(s) you are transforming from and to.

```
>> feml=xform(femg);
Nodes: Transforming GLOBAL->LOCAL

>> feml.node

ans =

IMAT_NODE - Nodes
ID           Def CS    Disp CS   Store CS             X            Y            Z    Color
-------- ---------- ---------- ---------- ------------ ------------ ------------ --------
       1          2          0          2           -1            0            0        1
       2          2          0          2            0            0            0        1
2 nodes
```

In this case XFORM knows that the node coordinate are stored in the global cartesian system, and it defaults to translating the coordinate to the node definition coordinate system. It is also possible to transform the node coordinates to an arbitrary coordinate system, as seen below.

```
>> feml=xform(fem,3);
Nodes: Transforming GLOBAL->LOCAL

>>
```

In this example the node coordinates are all converted to coordinate system 3, which does not have to be the definition or displacement coordinate system for these nodes. After performing this transformation, the storage coordinate system is 3.

It needs to be stressed that the FEM contains all of the coordinate system transformation information for both shapes and functions. This is why it is a required input argument to the shape and function transformation functions. If FROM and TO coordinate systems are not specified, the FEM is used to decide what these should be. In the above example we saw that XFORM transformed from the global to the definition coordinate system for nodes. For shapes and functions, the default transformation is between global and the displacement coordinate system. We can use this to our advantage to transform shape coefficients from the global to the displacement coordinate system, as seen in the example below.

```
>> shpl=xform(shpg,fem);
Shape 1: Transforming GLOBAL->LOCAL

>>
```

In the above example, XFORM uses column 3 of the FEM `node.cs` matrix to determine the FROM coordinate system (which in this case is 0, or global cartesian). It uses column 2, which is the displacement coordinate system, to determine the TO coordinate system.

Groups

Groups are a way of defining a list of entities such as nodes and elements. IMAT supplies an imat_group object, which contains lists of entities. Entities are stored as ID/TYPE pairs. Multiple entity types can be stored in a single group. Entity IDs must be unique within a given entity type, but do not need to be unique across entity types, even within the same group.

Groups are marginally useful on their own, but become much more useful when used in conjunction with imat_fem objects. The primary purpose of groups in this context is defining subsets of the FEM.

Data Format for groups

The IMAT_GROUP class is a fairly simple class, with three public properties (`id`, `name`, and `data`), and a special property, (`type`). The `.id` property contains the group ID. This is generally not used by IMAT, but is there for complete round-tripping through Universal files. The `.name` property is a string up to 80 characters that contains the group name. The .data property contains the entities in the group. This property is an Nx2 matrix, where the first column contains the entity IDs and the second column contains the entity types. The combination of entity ID and type must be unique within a group. The type property is a special dependent property that returns enumerations for the supported entity types. We will show how to use this later in this section.

For a complete definition of the various classes and descriptions of their contents, please refer to the Group Data Format Reference. Below is an example of the IMAT_GROUP object and its contents. This class operates very much like MATLAB structures.

```
>> g=readunv('../data/groups.unv',2477);
Read 4 imat_group objects

>> g

g =

4x1 IMAT_GROUP - Groups
GROUP(1): 1 - EVERYTHING
      ID     Type Name
-------- -------- ---------------
       1        1 coordinate system
       1        7 node
       2        7 node
       3        7 node
       4        7 node
       1        8 element
-------- -------- ---------------
       1 coordinate system
       4 nodes
       1 element

GROUP(2): 2 - TWO NODES
      ID     Type Name
-------- -------- ---------------
       3        7 node
       4        7 node
-------- -------- ---------------
       2 nodes

GROUP(3): 3 - ONE ELEMENT
      ID     Type Name
-------- -------- ---------------
       1        8 element
-------- -------- ---------------
       1 element

GROUP(4): 4 - ONE CS ONE ELEM TWO NODES
      ID     Type Name
-------- -------- ---------------
       1        1 coordinate system
       2        7 node
       3        7 node
       1        8 element
-------- -------- ---------------
       1 coordinate system
       2 nodes
       1 element

>> g2 = g(1)

g2 =
IMAT_GROUP - Groups
GROUP(1): 1 - EVERYTHING
      ID     Type Name
-------- -------- ---------------
       1        1 coordinate system
       1        7 node
       2        7 node
       3        7 node
       4        7 node
       1        8 element
-------- -------- ---------------
       1 coordinate system
       4 nodes
       1 element


>> g.type

ans =
     gUnknown: 0
        gCSys: 1
        gNode: 7
     gElement: 8
   gTraceline: 27

>>
```

In the example above, note that the group object can contain an array of groups. Each index into the group variable `g` contains a different group. Access to these groups follows the familiar MATLAB parenthesis "()" notation.

Group Methods

IMAT provides several methods to operate on groups. The methods are primarily boolean methods for combining groups, and are shown in the code snippet below. You can find a complete list of IMAT_GROUP methods here.

```
>> g(2) & g(3)                      % Boolean AND will produce an empty group

ans =

GROUP(2): 2 - TWO NODES
 --- Empty ---


>> g(2) | g(3)                      % Boolean OR merges without sorting

ans =

GROUP(2): 2 - TWO NODES
      ID     Type Name
-------- -------- ---------------
       3        7 node
       4        7 node
       1        8 element
-------- -------- ---------------
       2 nodes
       1 element

>> union(g(2),g(3))               % Union merges and sorts

ans =

GROUP(2): 2 - TWO NODES
      ID     Type Name
-------- -------- ---------------
       1        8 element
       3        7 node
       4        7 node
-------- -------- ---------------
       2 nodes
       1 element

>> g(2).extract(g.type.gNode)     % Extract the node IDs

ans =

     3
     4

>>
```

Group Subscripting

Group objects can be arrays of groups. Group properties can be accessed using the "dot" (structure) notation. Please see the following code snippet for examples of IMAT_GROUP subscripting.

```
>> g2 = g(3)              % Subscript notation accesses individual groups in the group object

g2 =

GROUP(1): 3 - ONE ELEMENT
      ID     Type Name
-------- -------- ---------------
       1        8 element
-------- -------- ---------------
       1 element

>> g2.id = 1234           % "dot" notation accesses individual properties

g2 =

GROUP(1): 1234 - ONE ELEMENT
      ID     Type Name
-------- -------- ---------------
       1        8 element
-------- -------- ---------------
       1 element

>> g2.name = 'new name'

g2 =
GROUP(1): 1234 - newname
      ID     Type Name
-------- -------- ---------------
       1        8 element
-------- -------- ---------------
       1 element


>> g.id                  % "dot" notation also works on arrays of groups

ans =
    [1]
    [2]
    [3]
    [4]

>>
```

Additional Data Types

IMAT has defined additional data types beyond the ones presented so far. A subset of these is presented here.

FCN

The fcn data type is a function data type designed specifically for Vibrata, ATA's Response Simulation product. It is a subclass of imat_fn, which means that it inherits all of the properties and methods of imat_fn. This means that all of the imat_fn properties are also properties of fcn, and all of the imat_fn methods that are not specifically overloaded for fcn will also work. This also means that the Function Reference and Data Attribute Reference in this User Guide for imat_fn also apply to fcn.

fcn extends imat_fn to add several properties. These properties are described here.

To see what methods are available for fcn, use this command:

```
>> methods(fcn)
```

To see the help for a specific fcn method, type

```
>> help fcn/<method_name>           % or use 'doc', and put in the actual method name for
<method_name>
```

Some IMAT functionality has specific understanding of fcn objects, and will tailor their functionality to fcn -specific properties. For example, uiplot recognizes fcn and displays functions in the list by Name rather than by reference/response coordinate and IDLine1, which is what it displays for imat_fn. It can also import and export *.fcn* files directly through its interface.

The fcn constructor accepts the same input arguments as the imat_fn constructor. For example, to create a 4x1 fcn and set its FunctionType to 'Frequency Response Function', use the following command.

```
>> f = fcn(4,'FunctionType','Frequency Response Function')

f =
(4x1) FCN with the following attributes:
Row  Name        FunctionType                 NumberElements  IDLine1
---  ----------  ---------------------------  --------------  -------
  1  Function 1  Frequency Response Function  0
  2  Function 2  Frequency Response Function  0
  3  Function 3  Frequency Response Function  0
  4  Function 4  Frequency Response Function  0
```

To convert between imat_fn and fcn, simply pass your variable through the appropriate constructor. See below for an example. When you convert an fcn to an imat_fn, you will lose the FCN-specific properties. When you convert an imat_fn to an fcn, the FCN-specific properties will be given default values.

```
>> f_i = imat_fn(3)            % Create a 3x1 imat_fn

f_i =

3x1 IMAT Function with the following attributes:
Record Name                   FunctionType      AbscissaSpacing   NumberElements
----------------------------  ----------------  ----------------  ----------------
1_(1X+,1X+)                   Time Response     Even              0
2_(1X+,1X+)                   Time Response     Even              0
3_(1X+,1X+)                   Time Response     Even              0

>> f_i.ResponseNode = 1:3      % Set an imat_fn property (which is therefor also a FCN-specific
property)

f_i =

3x1 IMAT Function with the following attributes:
Record Name                   FunctionType      AbscissaSpacing   NumberElements
----------------------------  ----------------  ----------------  ----------------
1_(1X+,1X+)                   Time Response     Even              0
2_(1X+,2X+)                   Time Response     Even              0
3_(1X+,3X+)                   Time Response     Even              0

>> f_f = fcn(f_i)              % Convert the imat_fn to an FCN--default values are set for the FCN-
specific properties

f_f =

(3x1) FCN with the following attributes:
Row  Name           FunctionType   NumberElements  IDLine1
---  -------------  -------------  --------------  -------
  1  1_(1X+,1X+)_1  Time Response  0
  2  2_(1X+,2X+)_1  Time Response  0
  3  3_(1X+,3X+)_1  Time Response  0

>> f_f.Name = 'My function'    % Set a FCN-specific property

f_f =

(3x1) FCN with the following attributes:
Row  Name         FunctionType   NumberElements  IDLine1
---  -----------  -------------  --------------  -------
  1  My function  Time Response  0
  2  My function  Time Response  0
  3  My function  Time Response  0

>> f_i2 = imat_fn(f_f);      % Create a 3x1 imat_fn

f_i2 =

3x1 IMAT Function with the following attributes:
Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Time Response    Even             0
2_(1X+,2X+)                  Time Response    Even             0
3_(1X+,3X+)                  Time Response    Even             0
```

Loading and Saving FCN

The fcn object defines several properties in addition to the imat_fn properties. These properties cannot be stored in a Universal file (UNV) or Associated Data File (ADF), so we utilized a different file format so that round-tripping fcn objects would not lose any information. The *.fcn* file is a MATLAB *.mat* file that contains specific information related to the fcn and how to process it. This includes units information, so that importing a *.fcn* into MATLAB when IMAT has been set to use a different units system from the one used to create the *.fcn* will properly convert the fcn data to the current units system.

To load fcn objects from a *.fcn* file, use the load method. This is an fcn method, so for MATLAB to recognize that you want to use this method, and not MATLAB's built-in `load` function, you must pass in an fcn object as the first input argument. If you do not pass in a filename, you will be prompted for one. Some examples of load usage are provided below.

```
>> f = load(fcn);              % You will be prompted for the .fcn file to load

>> f = load(fcn,'file.fcn');

>>
```

To save an fcn to a *.fcn* file, use the save method. Since you are passing in an fcn object as one of the input arguments, MATLAB understands that you want to use the FCN's save method instead of its built-in `save` function.

```
>> save(fcn);                  % You will be prompted for the .fcn file to write to

>> save(fcn,'file.fcn');

>>
```

IMAT+FEA Extended Functionality

IMAT+FEA offers significant additional functionality that extends IMAT's capabilities into Simulation. With IMAT+FEA you can work with a much broader range of data from different analytical solvers and utilize the significant strength and capabilities provided by IMAT. It provides the ability to interface with Nastran, Abaqus, and FEMAP, in addition to I-deas. Details of what IMAT+FEA supports for each analysis package is described in more detail below. The imported data is placed into the IMAT data types, so it is easy to use in MATLAB, and all of the existing IMAT functionality is readily available.

All of the IMAT+FEA functionality ships with the IMAT toolbox, but requires a separate license to access. You can use either tokens or a perpetual license to access these useful features. Please contact us for more information.

Nastran

IMAT+FEA can import data from Nastran bulk data (icluding DMI and DMIG), Punch, and Output2 files, using the readnas command. When importing from PCH or OP2 formats, readnas will prompt you with a GUI of the directory of the file's contents, allowing you to select the entities you with to import. You can also supply input arguments to readnas to use it in a batch mode that does not prompt you with a GUI.

While readnas supports a large amount of data natively, it also has the ability to read datablocks from the OP2 file in their "raw" (unprocessed) format. This is useful for cases where IMAT does not natively support the given dataset, or when you wish to perform low-level modifications to the OP2 file datablocks. IMAT+FEA provides several helper functions (`typecast_op2` and several functions that begin with `parseop2table`) to assist in processing "raw" datablocks. The `parseop2table` functions are provided as m-files so you can have some useful examples of how to go about processing raw datablocks.

The companion function expandDMI will expand DMI-formatted matrices into its full double representation. Once these matrices are in MATLAB, you can use MATLAB's powerful matrix-handling capabilities. The companion function mat2subst converts substructure matrices imported from OP2 into the structure format used by IMAT+Modal.

You can write directly to a Nastran OP2 file using writenas. Note that writenas only accepts the "raw" structure format returns by readnas. The companion function create_op2table converts IMAT datatypes to this raw format. Please see the help for create_op2table for a list of supported datatypes.

Abaqus

Abaqus results can be imported using readodb. This function reads user xyData and both history and field data from any steps in the ODB. xyData and history data imports into MATLAB as `imat_fn` objects. Nodal field data imports into `imat_shp` objects, and other field data is imported into `result` objects. This function also provides a GUI interface to allow you to select which entities to read from the ODB. readodb will prompt you with a GUI, from which you can select the results you wish to import. You can also view a directory of the contents of the xyData and step history and field data, from which you can select individual records to read. Alternately, you can supply readodb with command-line arguments that allow you to read the data in batch mode without being prompted with the GUI.

FEMAP

IMAT+FEA provides two primary means of communicating with FEMAP. The first is through the Neutral file, and the second is directly through COM, also known as ActiveX. The latter interface necessarily requires you to be on the Windows platform.

readneu imports selected datasets from a FEMAP Neutral file.

writefemap can export data either to a FEMAP Neutral file, or directly to a FEMAP session (Windows only) through COM (ActiveX). Nearly all of the native IMAT objects are supported.

MATLAB has buillt-in functions to interface with other Windows COM-aware programs. The functions `actxserver` and `actxGetRunningServer` initiate access to another COM process. The functions `get` and `set` access COM object properties, and `invoke` executes COM object methods. Due to the nature of how FEMAP makes itself available through COM and the way MATLAB works, MATLAB's native COM interface has limitations in the FEMAP methods it can access through its interface. Any of the methods that have output arguments will not work through the native MATLAB interface. Because of this, IMAT+FEA provides a companion function called femap_invoke. This function supports a subset of the methods that do not work through the MATLAB `invoke` function. If you attempt to use a method that femap_invoke is aware of, it will use its interface to invoke that method. Otherwise it will attempt to use MATLAB's native `invoke` function. You should use femap_invoke for all FEMAP method calls.

VTKPLOT

To use the VTK plotting capabilities, use the vtkplot function. This function will take an imat_fem object, along with an optional imat_shp or result object, and create a visualization of the FEM in a new MATLAB figure. An imat_vtkplot handle object will be returned, which can be used to manipulate the figure from the command line. The figure itself contains a number of menus, buttons, and toolbar items that will further modify the display. For complete information on these items, browse through the help for vtkplot and its associated methods. A brief example of vtkplot’s usage is included below.

A VTK plot is generated by using vtkplot.

```
>> vtkplot(fem);                % fem is an imat_fem
```

Displacement shapes can be added to the figure by passing them in when you call vtkplot. Shapes that will be used for deformed plots must be an imat_shp.Including them will enable all of the animation-specific features, such as the ability to animate and save an animation to an AVI file.

```
>> vtkplot(fem,shp)             % shp is an imat_shp
```

Alternatively, contour data can be displayed by passing in a result object when the plot is created. Data at Nodes and Data on Elements data formats are currently supported. Data at Nodes on Elements data can be used, if it is first converted into Data on Elements data by using the criterion method. Passing in result data will enable the Contour menu. From here, the different components of the result can be viewed individually.

```
>> vtkplot(fem,res)             % res is a result
```

Contour data can be included with displacement data, as well. The order of the inputs is not important.

```
>> vtkplot(fem,res,shp)
```

An array of groups can also be added to the plot by passing it in on the command line. The groups must be an imat_group. Passing in groups allows you to use them to display subsets of the FEM.

```
>> vtkplot(fem,group);          % group is an imat_group
```

In addition to controlling the display through vtkplot input arguments, you can make modifications after generating the plot by using the imat_vtkplot object returned by vtkplot. Some examples are provided below.

There are two ways to set the plot view besides through the figure menus.

```
>> vtkplot(fem,'view','xz')

>> obj = vtkplot(fem)           % obj is an imat_vtkplot object
>> obj.view('xz');
```

You can change the deformation scale of the displacement shapes. An equivalent imat_vtkplot method is not available.

```
>> vtkplot(fem,'scale',5.2)
```

You can change the figure title. An equivalent imat_vtkplot method is not available.

```
>> vtkplot(fem,'title','FEM Figure Title');
```

You can also control the visibility of the deformed and/or undeformed display.

```
>> vtkplot(fem,shp,'undeformed')

>> obj = vtkplot(fem,shp)
>> obj.visible('undeformed',1);      % Turn on the undeformed display
>> obj.visible('deformed',0);        % Turn off the deformed display
```

This example forces the undeformed geometry to be shaded.

```
>> vtkplot(fem,shp,'undeformed','shaded')

>> obj = vtkplot(fem,shp)
>> obj.displaystyle('undeformed','shaded');
```

You can animate the mode shapes by cycling through the shapes. This means that each frame is a subsequent shape. This is useful for animating transient output such as an Operating Deflection Shape.

```
>> vtkplot(fem,shp,'cycleshapes')
```

Create the figure without the button(s) at the bottom using the argument shown below.

```
>> vtkplot(fem,'nogui')
```

IMAT+Modal Extended Functionality

IMAT+Modal offers significant additional functionality that extends IMAT's capabilities for the modal test engineer. Several major areas of functionality are included, and are described below. These are useful for pre-test analysis to select optimal accelerometer locations for the modal test, to curve-fitting routines for extracting shapes from the test data.

All of the IMAT+Modal functionality ships with the IMAT toolbox, but requires a separate license to access. You can use either tokens or a perpetual license to access these useful features. Please contact us for more information.

Genetic Algorithm

The Genetic Algorithm (GA) for Accelerometer Placement will optimally select accelerometer locations for a modal vibration test. This iterative process replicates the way a new gene sequence is produced through direct reproduction, crossover, and mutation from an existing population. By this process a set of accelerometer locations can be efficiently selected to maximize the linear independence of test-measured mode shapes. The GA can also accommodate multiple FEM configurations, simultaneously selecting the best accelerometer locations for multi-configuration modal tests to minimize the test setup time. GA is directly compatible with Nastran; DMIG matrices are read in and final accelerometer DOF can be written to ASET cards.

The GA documentation is located in a separate document.

selectASET

selectASET is another accelerometer selection capability available in IMAT. It uses a brute-force method to iteratively removed the least important DOF (or triax) to achieve a user-defined target error or number of DOF. Multiple FEM configurations can be processed simultaneously. The selectASET functionality is available both from a GUI and from the command line. The GUI helps guide the user through the process of setting up the relevant parameters and run options and automates the process of post processing the results, including the ability to select intermediate TAM results for visualization and post processing. Results summaries can be exported to Excel-compatible XML format and the final degrees of freedom can be exported to Nastran ASET cards.

The selectASET GUI documentation is located in a separate document.

TAMKIT

TAMKIT provides procedures for selecting the instrumented DOF and for reducing the FEM matrices to these DOF. It is implemented as a set of Nastran DMAP alters, with some MATLAB functions used to read and interpret the Nastran data. It also includes procedures for comparing two similar models, and for comparing test and analysis modes on completion of the modal test. The documentation includes extensive discussion on the pros and cons of each DOF selection and matrix reduction method.

The TAMKIT documentation is located in a separate document.

Modal Test Kit (MTK)

Modal Test Kit, or MTK, contains routines that are useful when performing modal survey tests. These routines include methods for sensor and exciter placement, extracting modes from test data, verifying shape extractions through FRF synthesis comparisons, providing shape independence and completeness checks, and other useful routines. This documentation also covers steps for generating a Test Analysis Model (TAM) in Nastran without the use of a DMAP alter.

The MTK documentation is located in a separate document.

AFPoly™

Included in IMAT+Modal is AFPoly™, the Alias-Free Polyreference modal parameter estimation utility. This is a dramatic improvement over the classic polyreference technique. AFPoly is protected in Korea under patent number 10-1194238 and in Japan under patent number 4994448.

The AFPoly documentation is located in a separate document.

SDOFit™

SDOFit™ is a single degree of freedom (SDOF) modal parameter estimation (or "curve-fitting") application. The algorithm is a single-input, single-output, frequency domain, rational fraction polynomial model with generalized residuals. It will estimate the frequency and damping of modes in a narrow frequency band from a single frequency response function (FRF). The residuals are included in the model to account for the out-of-band modes. Although it is primarily intended as an SDOF method, it will allow more than one mode to be included in the model, as there are occasions that this approach will yield better results.

The SDOFit documentation is located in a separate document.

IMAT+Signal Extended Functionality

IMAT+Signal extends IMAT's functionality into signal processing, which in general encompasses processing time domain data into the frequency domain. In addition to command-line functions for tasks such as computing PSD and CSD from time histories, computing FRF, and time domain filtering, IMAT+Signal also provides two GUIs that greatly simplify the process. SPFRF provides a fully functional GUI for converting time histories to FRF, and RTK provides a GUI that facilitates the analysis of rotational events.

The command-line functions available are marked in the IMAT documentation with a `(+Signal)` by their name in the function list.

All of the IMAT+Signal functionality ships with the IMAT toolbox, but requires a separate license to access. You can use either tokens or a perpetual license to access these useful features. Please contact us for more information.

Example

The following is an example of how to use the command-line functionality available in IMAT+Signal to process time histories into Frequency Response Functions (FRF).

The first step in the process is to import your time histories into MATLAB. In this example, the time histories are stored in an Associated Data File (.ati).

```
>> t=readadf('test_data.ati')

t =
4x1 IMAT Function with the following attributes:
Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(,101X+)                   Time Response    Even             159681
2_(,103X+)                   Time Response    Even             159681
3_(,101Y+)                   Time Response    Even             159681
4_(,103Y+)                   Time Response    Even             159681
```

Once the time histories are in MATLAB, the next step is to compute Power Spectral Densities (PSD) of the response channels, and Cross Spectral (CSD) of of the responses and references. In this example we have two reference channels, which are the first two time histories. We want to include the reference channels as responses so we can evaluate the cross-responses between the two reference channels. We will apply a Hanning window using a blocksize of 4096.

```
>> p = psd(t,window(t,'hanning',4096))
PSD Processing Parameters
-------------------------
Number of points:  155648
Block Size:        4096 (window vector supplied)
Averages:          38
Overlap:           0 samples (0.00%)
Delta F:           1.0000 Hz
Spectral lines:    2049
-------------------------
...Processing time histories

p =
4x1 IMAT Function with the following attributes:
Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(101X+,101X+)              Power Spectral D Even             2049
2_(103X+,103X+)              Power Spectral D Even             2049
3_(101Y+,101Y+)              Power Spectral D Even             2049
4_(103Y+,103Y+)              Power Spectral D Even             2049

>> c = csd(t(1:2),t,window(t,'hanning',4096))
CSD Processing Parameters
-------------------------
Number of points:  155648
Block Size:        4096 (window vector supplied)
Averages:          38
Overlap:           0 samples (0.00%)
Delta F:           1.0000 Hz
```

```
Spectral lines:    2049
-------------------------
...Processing time histories

c =
4x2 IMAT Function with the following attributes:
Row Col Record Name          FunctionType     AbscissaSpacing  NumberElements
--- --- -------------------- ---------------- ---------------- ----------------
1   1   1_(101X+,101X+)      Auto Spectrum    Even             2049
2   1   2_(101X+,103X+)      Cross Spectrum   Even             2049
3   1   3_(101X+,101Y+)      Cross Spectrum   Even             2049
4   1   4_(101X+,103Y+)      Cross Spectrum   Even             2049
1   2   5_(103X+,101X+)      Auto Spectrum    Even             2049
2   2   6_(103X+,103X+)      Cross Spectrum   Even             2049
3   2   7_(103X+,101Y+)      Cross Spectrum   Even             2049
4   2   8_(103X+,103Y+)      Cross Spectrum   Even             2049
```

Once we have PSDs and CSDs, we can compute the FRF. Though not shown here, it is also a very good idea to compute the coherence functions, and review them to evaluate the quality of your FRF.

```
>> f = frf(c,p)

f =
4x2 IMAT Function with the following attributes:
Row Col Record Name          FunctionType     AbscissaSpacing  NumberElements
--- --- -------------------- ---------------- ---------------- ----------------
1   1   1_(101X+,101X+)      Frequency Respon Even             2049
2   1   2_(101X+,103X+)      Frequency Respon Even             2049
3   1   3_(101X+,101Y+)      Frequency Respon Even             2049
4   1   4_(101X+,103Y+)      Frequency Respon Even             2049
1   2   5_(103X+,101X+)      Frequency Respon Even             2049
2   2   6_(103X+,103X+)      Frequency Respon Even             2049
3   2   7_(103X+,101Y+)      Frequency Respon Even             2049
4   2   8_(103X+,103Y+)      Frequency Respon Even             2049
```

spFRF

spFRF™ stands for "Signal Processing for Frequency Response Functions". The primary purpose of this application is to process time response data into spectral functions such as frequency response, coherence, auto-spectra and cross-spectra. spFRF can read time response functions from ADF files, Universal files, or from the MATLAB workspace, and can write time response and spectral functions to the same. spFRF can process continuous measurements, as well as triggered measurements with the ability to preview, and accept or reject each frame. Although the name implies that spFRF produces frequency response functions, it can also be used to generate the auto-spectra of response-only datasets.

The spFRF documentation is located in a separate document.

spVIEW

spVIEW™ is an application for plotting multiple datasets of measurements. It was originally intended for viewing the spectral function results from spFRF™. However, spVIEW's scope has expanded to viewing time response, mode indicator functions (MIF), and sound pressure level (SPL). While spVIEW's primary purpose is plotting, it also has some signal processing and computational capabilities, such as filtering, resampling, decimation, and octave band reduction.

spVIEW can read time response and spectral functions from ADF files, Universal files, or from the MATLAB workspace, and can write sieved and truncated datasets of functions to the same. spVIEW has four modules (Time Response, Spectra, MIFs, and SPL) and some data can be sent between the modules.

The spVIEW documentation is located in a separate document.

RTK

RTK is the Rotational Toolkit, which is a GUI that facilitates the analysis and order-tracking of rotational events, using the Vold-Kalman filtering technique. The application uses time-history data files from simulations or real-time data collections to process and visualize the spectral properties of rotating events. This tool facilitates the process of obtaining response characteristics and finding insights into the mechanics of rotating events.

The RTK documentation is located in a separate document. This documentation also covers the command-line usage of the Vold-Kalman filtering capability.

Examples

IMAT and its related toolboxes provide you with significant functionality. However, it may not seem obvious how to use IMAT's capabilities, even after reading the documentation. Because of this, we have provided several useful examples that demonstrate how to use IMAT. All of the example files can be found in the `examples` subdirectory of your IMAT installation. We highly recommend that you include this directory in your MATLAB path if you have not already done so.

The following sections describe what you can find in the `examples` directory.

Example Functions

IMAT provides a number of useful utilities. These utilities are provided as MATLAB m-files, so the source code is readily visible. The following table summarizes the example functions provided.

Function	**Description**
`calc_energy_fractions.m`	Calculate and report kinetic and strain energy fractions for groups
`cmif.m`	Compute Complex Mode Indicator Functions
`create_default_fn`	Create a default imat_fn of a specified type
`format_ppt.m`	Format an imat_fn plot for pasting into Microsoft Powerpoint
`frfsyn.m`	Synthesize Frequency Response Functions from real or complex mode shapes
`gmif.m`	Compute the various mode indicator functions
`mmif.m`	Compute Multivariate Mode Indicator Functions
`ortho.m`	Compute Modal Assurance Criterion or Orthogonality for a set of mode shapes
`plot4views.m`	Create a four-paneled mode shape plot
`psmif.m`	Compute Power Spectrum Mode Indicator Functions
`qmif.m`	Compute Quadrature Mode Indicator Functions (CMIF using the imaginary component of the FRF)
`rbmodes.m`	Generate rigid body modes from FEM geometry

`readunv_180.m_example.m`	Example for using the readunv API to import dataset 180
`tdm_xlsread.m`	Read Test Display Model from Excel worksheets
`tdm_xlsread_example_1.xlsx`	TDM example Excel worksheet
`tdm_xlsread_example_2.xlsx`	TDM example Excel worksheet
`tdm_xlsread_template.xlsx`	TDM Excel worksheet template
`tdm_xlswrite.m`	Write Test Display Model to Excel worksheets
`writeblk.m`	Write to a Nastran BLK file
`writedmig.m`	Write matrix in Nastran DMIG format
`writempc.m`	Write a row- or column-oriented matrix as NASTRAN MPC cards
`writerandps.m`	Write RANDPS and TABRND1 cards to a Nastran bulk data file
`writetabled1.m`	Write imat_fn or fcn as Nastran TABLED1 cards
`write_nas_field.m`	Pack real numbers into 16-character fields in Nastran format
SMAC subdirectory	
`smac.m`	Synthesize Modes and Correlate modal parameter extraction utility
Anapost subdirectory	
`aelempost.m`	Summarize solid element stress results from Nastran Output2/Punch files
`max_von_mises.m`	Compute the highest von Mises stress for a load of unknown direction

The examples we provide also happen to be useful. We provide functions to compute Modal Assurance Criteria (MAC) and orthogonality, generate rigid body mode shapes from supplied finite element model geometry, generate a modal substructure from mode shapes, compute normal, complex, and multivariate mode indicator functions, and synthesize frequency response functions (FRF) from real normal modes.

Another very useful set of example functions allows you to easily create and/or modify FEM connectivity using Microsoft Excel. This capability is particularly useful for test engineers who might use a test display model (TDM) for animating mode shapes. The examples directory contains several functions and spreadsheets whose filenames start with `tdm_xls*`. You can develop FEM connectivity including local coordinate systems, nodes, elements, and tracelines in Excel, and then invoke `tdm_xlsread.m` to import the TDM into an imat_fem in MATLAB for display. You can export an imat_fem from MATLAB using `tdm_xlswrite.m` to an Excel spreadsheet for modification and then subsequent import back into MATLAB. Also included are a blank template Excel file called `tdm_xlsread_template.xlsx` and two example TDM files.

In addition to these useful functions, we provide several readunv extension examples that show you how read Universal file datasets that are not natively handled by readunv. The three examples provided read datasets 2452, 2467, and 2477, which contain group information. If the `examples` directory is in your MATLAB path, readunv will automatically detect and use these extension functions.

The best way to familiarize yourself with the functions we provide is to browse the examples directory and read the help for these functions.

In addition to the sample functions found in this directory, the main IMAT source directory contains several MATLAB scripts that start with `demo_*`. These files contain code and comments that show you how to use IMAT by going through some simple examples. You can access these scripts by running them (type `demo_` at the MATLAB prompt, followed by a <TAB> to see the available choices).

Demo Data Files

In the `Demo_Files` subdirectory you will find a number of data files. All of these files are related to a fictional spacecraft (the General Purpose Spacecraft, or GPSC). You can use these data files to exercise the various IMAT import functions, and also to load IMAT datatypes so you can exercise the functionality available to them.

This directory contains function and shape ADFs, Nastran bulk data, Output2, and Punch files, and an Abaqus Input file and ODB.

SMAC

SMAC, or Synthesize Modes and Correlate, is not so much an example as it is a fully functional modal curve-fitting utility. It utilizes a different approach from most modal curve-fitting algorithms. SMAC is based on modal filter theory. You can find an introduction to SMAC along with helpful hints on how best to use it here.

To use SMAC, simply add the `examples/SMAC` directory to your MATLAB path, and invoke it by typing `'smac'` at the MATLAB prompt.

SMAC is generously provided by Sandia National Laboratories through the United States government, and is released under Mozilla Public License version 1.1. The text of this license can be found here.

Analysis Post-Processing Utilities

The `anapost` directory contains a collection of useful functions for post-processing Nastran results. These include `aelempost`, a utility that reads solid element stress results from Nastran OP2 or PCH and post-processes them. `max_von_mises` calculates the worst direction von Mises stress on a per-element basis, given stress inputs in 3 orthogonal directions. For more details on the supplied utilities, please review the contents of this directory in your installation. README.txt, located in this directory, gives more details about the available utilities.

To use any of these utilities, simply add the `examples/anapost` directory to your MATLAB path.

Experimental New Features

The following experimental feature is included in this version of IMAT. It is considered beta in this release, so it may not work properly. Use at your own risk. Please provide feedback to ATA for this feature, so we can make improvements and meet your needs.

There are no new experimental features to introduce for this release.

Reference Guide

IMAT Methods and Functions for `imat_fn` objects

---

imat_fn	Create an `imat_fn` object
imat_fn/get	Get one or more attributes of an `imat_fn` object
imat_fn/set	Set one or more attributes of an `imat_fn` object
imat_fn/validate	Validate the internal consistency of an `imat_fn`
imat_fn/setdef	Set default attributes for function creation
imat_fn/setdisplay	Set attributes to show for function displays
imat_fn/edit_attributes	Convenient GUI for editing `imat_fn` data attributes
imat_fn/allref	Get a coordinate trace of all reference coordinates
imat_fn/allres	Get a coordinate trace of all response coordinates
imat_fn/build_ctrace	Generate coordinate trace of all records
imat_fn/fn2shp	Create an `imat_shp` from the supplied `imat_fn`
imat_fn/get_units_labels	Get units labels for the supplied `imat_fn`
imat_fn/truncate	Truncate an `imat_fn` to a specified abscissa range
imat_fn/chgunits	Convert an `imat_fn` to a different unit system
imat_fn/plot	Plot an `imat_fn` in a special figure window
imat_fnplot class	Description of the object properties and methods returned by imat_fn/plot
imat_fn/plotyy	Double Y-axis plot

imat_fn/plot3	Plot an `imat_fn` on a 3D axis in a special figure window
imat_fn/uiselect	Select functions using a graphical interface
imat_fn/uplus	Unary plus (`+f`)
imat_fn/uminus	Unary minus (`-f`)
imat_fn/plus	Add ordinates (`f1+f2`)
imat_fn/minus	Subtract ordinates (`f1-f2`)
imat_fn/times	Termwise multiply ordinates (`f1.*f2`)
imat_fn/mtimes	Matrix multiply ordinates (`A*f`)
imat_fn/rdivide	Termwise divide ordinates (`f1./f2`)
imat_fn/ldivide	Termwise divide ordinates (`f1.\f2`)
imat_fn/mrdivide	Matrix divide ordinates (`f/A`)
imat_fn/mldivide	Matrix divide ordinates (`A\f`)
imat_fn/mpower	Matrix power (`f^a`)
imat_fn/power	Array power (`f.^a`)
imat_fn/sqrt	Square root
imat_fn/db	Convert to decibels
imat_fn/db2eu	Convert decibels to engineering units
imat_fn/exp	Exponential
imat_fn/log2	Base 2 logarithm and dissect floating point number
imat_fn/log10	Common (base 10) logarithm
imat_fn/real	Take real part of ordinate

imat_fn/imag	Take imaginary part of ordinate
imat_fn/conj	Take complex conjugate of ordinate
imat_fn/abs	Take absolute value (or modulus) of ordinate
imat_fn/diag	Diagonal `imat_fn` matrices and diagonal of a matrix
imat_fn/phase	Replace ordinate with its phase angle in radians
imat_fn/phased	Replace ordinate with its phase angle in degrees
imat_fn/cbred	Compute constant band reduction of the supplied `imat_fn`
imat_fn/fft	Compute PSDs from the supplied time history `imat_fn`
imat_fn/filterf	FIR Filter the supplied `imat_fn` time history
imat_fn/filteri	IIR Filter the supplied `imat_fn` time history
imat_fn/findpeaks	Locate peaks in an `imat_fn` using a tolerance
imat_fn/csd **(+Signal)**	Compute CSDs from the supplied time history `imat_fn`
imat_fn/psd **(+Signal)**	Compute PSDs from the supplied time history `imat_fn`
imat_fn/psd2trans **(+Signal)**	Generate transient from Power Spectral Density
imat_fn/frf **(+Signal)**	Compute FRFs from the supplied CSD/PSD functions
imat_fn/spl **(+Signal)**	Convert spectra to sound pressure level
imat_fn/acoustic_weighting **(+Signal)**	Apply acoustic weighting
imat_fn/window	Create a window function
imat_fn/diff	Differentiate the supplied `imat_fn`
imat_fn/integ	Integrate the supplied `imat_fn`
imat_fn/interp	Interpolate the supplied `imat_fn`

imat_fn/decimate	Decimate the supplied `imat_fn`
imat_fn/envelope	Create an envelope function of the supplied `imat_fn`
imat_fn/movavg	Moving average of the supplied `imat_fn`
imat_fn/octaven	Reduce PSD and FRF `imat_fn` to 1/N octaves
imat_fn/octaven2nb	Convert octave-reduced functions to narrowband
imat_fn/polyfit	Create an polynomial regression of the supplied `imat_fn`
imat_fn/srs **(+Signal)**	Compute shock response spectra from the supplied `imat_fn`
imat_fn/srs2trans **(+Signal)**	Generate transient from Shock Response Spectrum
imat_fn/sweep **(+Signal)**	Generate a sweep function
imat_fn/cumrms	Calculate cumulative RMS on the supplied frequency domain `imat_fn`
imat_fn/rms	Root mean square
imat_fn/Statistical Functions	Calculate various statistics on the supplied `imat_fn`, including min, max, mean, standard deviation, variance, skewness, and kurtosis
imat_fn/uiplot	Plot an `imat_fn` in a special GUI that provides extended functionality
imat_fn/writecsv	Writes an `imat_fn` to a CSV file
imat_fn/xform	Coordinate transform functions and time histories

imat_fn/imat_fn

Purpose

Create an `imat_fn` object.

Syntax

```
f=imat_fn
f=imat_fn(m,n,p,...)
f=imat_fn(m,n,p,...,'Attr',value,...)
f=imat_fn(m,n,p,...,v)
f=imat_fn(v)
f=imat_fn(imat_result)
```

Description

You call `imat_fn` to construct an `imat_fn` object. An `imat_fn` object contains function or time history abscissa/ordinate values as well as data attributes such as function type, data type, coordinate labels, etc.

Calling `imat_fn` with no arguments creates an empty `imat_fn` object.

If the first one or more arguments to `imat_fn` are integers, then an `m`x`n`x`p`x... function is created. Each element of this object is a single function. (Alternatively, the dimension of the function object can be specified by a row vector of dimensions, as in `imat_fn ([2 2])`.) The output function will have the default attributes, which can be changed by setdef.

To override default attributes at time of object creation, you can specify one or more attribute names and values in the call to `imat_fn`. This is equivalent to creating the object with default attributes, and then setting the specified attributes in sequential order.

Instead of a list of attribute names and values, you may supply a structure variable `v` with field names equal to attribute names, and field values set to the desired attribute values. (Such a structure can be obtained from the `get` function with multiple attribute requests.)

If the supplied input is an imat_result object, it will be converted to an `imat_fn`. Results with DataLocations of Data At Nodes, Data At Nodes On Elements, and Data On Elements are converted, and results with no data are skipped. The results are first grouped by DataLocation, then by ResultType. Within each matching result type, the Time or Frequency fields must be unique. These generate the abscissa values for the `imat_fn`. A new function is created for each component in the result. If the time field contains unique values, the output `imat_fn` will be a Time Response. Otherwise it will be a Spectrum. Resulting functions with a given reference node were created from the same set of results. The reference node is set to the index of the first result in the supplied `imat_result` object used to generate the functions.

When Data At Nodes results are converted, the node number and direction are placed in the response coordinate, and the reference coordinate is blank. For Data On Elements results, the reference coordinate is blanked and the response node is set to the element number and the response direction is 'ELEM'. For Data At Nodes On Elements results, the reference node is set to the element number, reference direction is 'ELEM', response node is node number, response direction is component, and the layer is placed in UserValue3.

Examples

```
>> f=imat_fn(3,'functiontype','frequency response function');
```

See Also

imat_fn/setdef, imat_fn/set, imat_fn/get, result

User's Guide

imat_fn/get

Purpose

Get one or more attributes of an `imat_fn` object.

Syntax

```
v=get(f,'Attrib')
v=get(f)
```

Description

When applied to an `imat_fn` object `f`, the GET function returns the values of selected attributes of `f`. The available attributes are listed here. Attribute names are not case sensitive.

If only a single attribute is requested, then the value of that attribute is returned, or printed if no output argument is supplied. Numeric attributes are returned in a numeric array with the same dimensions as `f`. String-valued attributes (including list attributes) are returned in a string variable (if `f` is a single function) or a cell array of strings with the same dimensions as `f` (if `f` contains more than one function). The *Abscissa* attribute returns a numeric array with an extra initial dimension equal to the largest number of abscissa values in `f`. (For example, if `f` is 2x3 and all functions have 100 or fewer abscissa values, then the returned value is a 100x2x3 array.) The *Ordinate* attribute similarly returns a numeric array with an extra dimension equal to the largest number of abscissa values. If any of the functions in `f` have a smaller number of abscissa values, that column will be padded with NaN values. Note that an *Abscissa* array is returned even for evenly spaced functions. In such cases, the abscissa values are not actually stored in the function, but are created by the `get` function.

If more than one attribute is specified, then the output argument will be a scalar structure variable with field names equal to the requested attributes. The value of each field will be as described above. If no output argument is supplied, the resulting structure will be printed on the standard output.

If no attributes are specified, then the values of *all* attributes will be returned as described above.

An alternative to the `get` function is the syntax `f.`*`attrib`*.

Examples

```
>> setunits('in');
Units set to IN
>> f=readunv('/ms5/examples/tda/air_2ref.unv');
Universal file written in MM units, converting to IN
Read 14x1 imat_fn

f =
14x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(,2Z+)             Time Response    Even             20480
2_(,14Z+)            Time Response    Even             20480
3_(,2X+)             Time Response    Even             20480
4_(,2Y+)             Time Response    Even             20480
5_(,2Z+)             Time Response    Even             20480
6_(,14X+)            Time Response    Even             20480
7_(,14Y+)            Time Response    Even             20480
8_(,14Z+)            Time Response    Even             20480
9_(,30X+)            Time Response    Even             20480
10_(,30Y+)           Time Response    Even             20480
11_(,30Z+)           Time Response    Even             20480
12_(,24X+)           Time Response    Even             20480
13_(,24Y+)           Time Response    Even             20480
14_(,24Z+)           Time Response    Even             20480

>> x=get(f,'ordinate');
>> size(x)

ans =
       20480          14

>> get(f,'responsecoord','abscissamin')
ResponseCoord: {14x1 cell}
  AbscissaMin: 0

>>
```

See Also

imat_fn/set

---

imat_fn/set

Purpose

Set one or more attributes of an `imat_fn` object.

Syntax

```
set(f)
set(f,'attrib1',value1,'attrib2',value2,...)
f(1:2)=set(f(1:2),'attrib1',value1,...)
g=set(f,v)
```

Description

The SET function allows you to change any attribute of an `imat_fn` variable.

If SET is called with no attribute arguments, then all attribute names and their possible values are printed to standard output. This functionality acts as a built-in help mechanism for data attributes. (Note that in this case, the argument `f` is not referenced, but simply causes MATLAB to call the SET function associated with `imat_fn` objects.).

If the reference to the input variable `f` includes subscripts, you must assign the output from SET as in the 3rd example above, otherwise MATLAB will issue an error.

To change attribute values, you must either supply pairs of attribute names and values (syntax 2), or you must supply a scalar structure variable (like the one returned by get) whose field names are the attribute names to be changed, and whose field values are the desired values (syntax 3).

The attribute names can be any valid attribute for the `imat_fn` object. Attribute values can either be a single value (in which case the value is used to set all elements of `f`), or an array of values dimensioned the same size as `f`. A cell array of strings should be used to set string or list attributes. The *Abscissa* and *Ordinate* attributes are special cases.

These can be set either to Nx1 arrays (in which case the same values will be used for all elements of `f`), or they can be dimensioned the same size as `f` with an extra first dimension equal to the number of nodes or shape coefficients. This is consistent with the values returned by the get function. (Also, NaN values at the end of a column will be discarded when setting that element.)

The attributes are set in the order listed. Note that setting some attributes will cause side effects:

- Setting the *Abscissa* causes the *AbscissaSpacing* attribute to be set to `'Uneven'`. To set up an evenly spaced abscissa, you should set the *AbscissaSpacing* to `'Even'`, and set the *AbscissaMin* and *AbscissaInc* numeric attributes.
- If you increase the number of abscissa values by setting the *Abscissa* attribute, then zeros will be added to the *Ordinate* attribute to keep the size consistent. If you decrease the number of abscissa values, then the ordinate will be truncated. The *NumberElements* attribute will also be adjusted.
- If you increase the number of ordinate values by setting the *Ordinate* attribute, then additional abscissa values will be added to the *Abscissa* attribute if the *AbscissaSpacing* is `'Uneven'`. The additional abscissa values will be repetitions of the maximum abscissa value. If you decrease the number of ordinate values, then the *Abscissa* will be truncated. The *NumberElements* attribute will also be adjusted in either case.
- Changing the *NumberElements* attribute will affect the *Abscissa* and *Ordinate* attributes, either by truncation (if decreasing) or by adding abscissa and ordinate values (if increasing).
- Setting the *Ordinate* attribute will automatically adjust the *OrdinateType* attribute (real or complex).
- If you change the *AbscissaSpacing* attribute from `'Even'` to `'Uneven'`, then an abscissa vector will be created based on the current *AbscissaMin* and *AbscissaInc*. If you change the *AbscissaSpacing* attribute from '`Uneven'` to `'Even'`, then the *AbscissaMin* attribute will be set to the minimum of the *Abscissa* values, and the *AbscissaInc* attribute will be set to the best-fit slope through the *Abscissa* values.
- Changing the *AbscissaDataType*, *OrdNumDataType*, *OrdDenDataType*, or *ZAxisDataType* adjusts the corresponding exponents to match. Changing the exponents can cause the data type to change to `'General'`.

An alternative way to set attributes is with the syntax `f.attrib=value`.

Examples

```
>> f=imat_fn(3);
>> f=set(f, ...
'functiontype', 'frequency response function', ...
'abscissamin', 0, ...
'abscissainc', 0.05, ...
'abscissadatatype', 'frequency', ...
'ordnumdatatype', 'acceleration', ...
'orddendatatype', 'excitation force', ...
'ordinate', randn(500,3), ...
'referencecoord', '101x', ...
'responsecoord', {'1x';'2x';'3x'} )

f =
3x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(101X+,1X+)        Frequency Respon Even             500
2_(101X+,2X+)        Frequency Respon Even             500
3_(101X+,3X+)        Frequency Respon Even             500

>>
```

See Also

imat_fn/get, imat_fn/setdef

---

imat_fn/validate

Purpose

Validate the internal consistency of an `imat_fn`.

Syntax

```
g=validate(f)
```

Description

VALIDATE returns a "validated" version of an `imat_fn` object. The validation process includes checking the internal data structure. This function is a useful check when reading back suspect data objects.

---

imat_fn/setdef

Purpose

Set default attributes for function creation.

Syntax

```
setdef(f,'Attrib1',Value1,...)
setdef(f,0)
```

Description

SETDEF is used to change the default attributes of IMAT Functions (i.e., the attributes set by the `imat_fn` constructor function). The first argument to SETDEF must be an `imat_fn` variable (to cause MATLAB to call the correct version of SETDEF), but its value does not matter. For example, you can use the syntax `setdef(imat_fn,...)`.

In the first form, you can provide any arguments that would be acceptable to the `set` function for a single function. These attributes will be used for future `imat_fn` variables created by the `imat_fn` constructor.

The second syntax `setdef(f,0)` restores all attributes to their original values. The original default attributes are listed in the attribute reference section.

Examples

```
>> setdef(imat_fn, 'abscissamin', 0, 'abscissainc', 0.1);
```

See Also

imat_fn/get, imat_fn/set

---

imat_fn/setdisplay

Purpose

Set attributes to show for function displays.

Syntax

```
setdisplay(f,'Attrib1','Attrib2','Attrib3')
att=setdisplay(f,{'Attrib1','Attrib2'})
setdisplay(imat_fn,999)
```

Description

When an `imat_fn` is displayed on the screen (such as when a semicolon is omitted at the end of a statement), a summary of the variable is printed. By default, the *FunctionType*, *AbscissaSpacing*, and *NumberElements* attributes are displayed.

Input attribute names can be specified either as a series of strings, or a cell array of strings. The optional output ATT is a cell array of strings containing the current display attributes.

Use the SETDISPLAY function to change the displayed attributes. At least one attribute names must be specified. If no attributes are specified, the default attributes will be assumed. Any attributes other than *Abscissa* or *Ordinate* may be selected.

The third syntax shown above allows you to specify the maximum number of functions to display before switching to the truncated display form of just showing the dimensions of the `imat_fn`. Setting this to 0 means that all of the functions should always be displayed.

Examples

```
>> setdisplay(imat_fn,'functiontype','idline1','idline4')
```

See Also

imat_fn/set

---

imat_fn/edit_attributes

Purpose

Convenient GUI for editing `imat_fn` data attributes.

Syntax

```
g=edit_attributes(f)
```

Description

EDIT_ATTRIUBTES is a convenience Graphical User Interface for editing `imat_fn` attributes. Most attributes are editable. The GUI groups related attributes for easy editing.

The input argument to EDIT_ATTRIBUTES is an `imat_fn`. It can be an array of functions. The output G is an `imat_fn` containing the modified `imat_fn`. If the user clicks on 'Cancel', then `g=-1` is returned.

See Also

imat_shp/edit_attributes, result/edit_attributes

---

imat_fn/allref

Purpose

Get a trace of all reference coordinates.

Syntax

```
t=allref(f)
```

Description

ALLREF returns a coordinate trace `t` containing all reference coordinates in the `imat_fn` `f`. The coordinate trace will be in the order of the functions in `f(:)`.

To create a coordinate trace of all unique reference coordinates in `f`, use `unique(allref(f))`.

Examples

```
>> f=readunv('/ms5/examples/tda/air_func.unv')
Universal file written in SI units, converting to IN
Read 211x1 imat_fn

f =
211x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(30X+,1Z+)         Frequency Respon Even             512
2_(1Z+,1Z+)          Frequency Respon Even             512
3_(16Z+,1Z+)         Frequency Respon Even             512
4_(30X+,2Z+)         Frequency Respon Even             512
5_(1Z+,2Z+)          Frequency Respon Even             512
6_(16Z+,2Z+)         Frequency Respon Even             512
7_(30X+,3Z+)         Frequency Respon Even             512
8_(1Z+,3Z+)          Frequency Respon Even             512
9_(16Z+,3Z+)         Frequency Respon Even             512
10_(30X+,4Z+)        Frequency Respon Even             512
11_(1Z+,4Z+)         Frequency Respon Even             512
12_(16Z+,4Z+)        Frequency Respon Even             512
...
197_(16Z+,25Y+)      Frequency Respon Even             512
198_(30X+,30Y-)      Frequency Respon Even             512
199_(1Z+,30Y-)       Frequency Respon Even             512
200_(16Z+,30Y-)      Frequency Respon Even             512
201_(16Z+,1Z+)       Frequency Respon Even             512
202_(16Z+,15Y-)      Frequency Respon Even             512
203_(16Z+,16Y+)      Frequency Respon Even             512
204_(16Z+,17Y-)      Frequency Respon Even             512
205_(16Z+,18Y+)      Frequency Respon Even             512
206_(16Z+,21Y-)      Frequency Respon Even             512
207_(16Z+,22Y+)      Frequency Respon Even             512
208_(16Z+,23Y-)      Frequency Respon Even             512
209_(16Z+,24Y+)      Frequency Respon Even             512
210_(16Z+,25Y+)      Frequency Respon Even             512
211_(16Z+,30Y-)      Frequency Respon Even             512

>> t=allref(f);
>> length(t)

ans =
   211

>> t(1:5)

ans =
    '30X+'
    '1Z+'
    '16Z+'
    '30X+'
    '1Z+'

>> unique(t)
```

```
ans =
    '1Z+'
    '16Z+'
    '30X+'

>>
```

See Also

imat_fn/allres

---

imat_fn/allres

Purpose

Get a trace of all response coordinates.

Syntax

```
t=allres(f)
```

Description

ALLRES returns a coordinate trace `t` containing all response coordinates in the `imat_fn f`. The coordinate trace will be in the order of the functions in `f(:)`.

To create a coordinate trace of all unique response coordinates in `f`, use `unique(allres(f))`.

Examples

```
>> f=readunv('/ms5/examples/tda/air_func.unv')
Universal file written in SI units, converting to IN
Read 211x1 imat_fn

f =
211x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(30X+,1Z+)         Frequency Respon Even             512
2_(1Z+,1Z+)          Frequency Respon Even             512
3_(16Z+,1Z+)         Frequency Respon Even             512
4_(30X+,2Z+)         Frequency Respon Even             512
5_(1Z+,2Z+)          Frequency Respon Even             512
6_(16Z+,2Z+)         Frequency Respon Even             512
7_(30X+,3Z+)         Frequency Respon Even             512
8_(1Z+,3Z+)          Frequency Respon Even             512
9_(16Z+,3Z+)         Frequency Respon Even             512
10_(30X+,4Z+)        Frequency Respon Even             512
11_(1Z+,4Z+)         Frequency Respon Even             512
12_(16Z+,4Z+)        Frequency Respon Even             512
...
197_(16Z+,25Y+)      Frequency Respon Even             512
198_(30X+,30Y-)      Frequency Respon Even             512
199_(1Z+,30Y-)       Frequency Respon Even             512
200_(16Z+,30Y-)      Frequency Respon Even             512
201_(16Z+,1Z+)       Frequency Respon Even             512
202_(16Z+,15Y-)      Frequency Respon Even             512
203_(16Z+,16Y+)      Frequency Respon Even             512
204_(16Z+,17Y-)      Frequency Respon Even             512
205_(16Z+,18Y+)      Frequency Respon Even             512
206_(16Z+,21Y-)      Frequency Respon Even             512
207_(16Z+,22Y+)      Frequency Respon Even             512
208_(16Z+,23Y-)      Frequency Respon Even             512
209_(16Z+,24Y+)      Frequency Respon Even             512
210_(16Z+,25Y+)      Frequency Respon Even             512
211_(16Z+,30Y-)      Frequency Respon Even             512

>> t=allres(f);
>> length(t)

ans =
   211

>> t(1:5)

ans =
    '1Z+'
    '1Z+'
    '1Z+'
    '2Z+'
    '2Z+'

>> unique(t)
```

```
ans =
    '1X+'
    '1Y-'
    '1Z+'
    '2X+'
    '2Y+'
    '2Z+'
    '3Y-'
    '3Z+'
...
    '26X+'
    '26Z+'
    '27X+'
    '27Z+'
    '28X+'
    '28Z+'
    '29X+'
    '29Z+'
    '30X+'
    '30Y-'
    '30Z+'

>>
```

See Also

imat_fn/allref

---

imat_fn/build_ctrace

Purpose

Return an `imat_ctrace` of all reference or response coordinates in the supplied `imat_fn`.

Syntax

```
t=build_ctrace(f,'ref')
t=build_ctrace(f,'res')
```

Description

Given the supplied `imat_fn`, `f`, BUILD_CTRACE will return an `imat_ctrace` of all of the reference or response coordinates. Passing in `'ref'` as the second argument will return the reference coordinates, and passing in `'res'` will return the response coordinates.

Examples

```
>> f=imat_fn(4);
>> f.responsenode=1:4

f =
4x1 IMAT Function with the following attributes:
Record Name                FunctionType     AbscissaSpacing  NumberElements
-------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                Time Response    Even             0
2_(1X+,2X+)                Time Response    Even             0
3_(1X+,3X+)                Time Response    Even             0
4_(1X+,4X+)                Time Response    Even             0

>> t=build_ctrace(f,'res')

t =
    '1X+'
    '2X+'
    '3X+'
    '4X+'
```

---

imat_fn/fn2shp

Purpose

Create an `imat_shp` from the supplied `imat_fn`.

Syntax

```
s=fn2shp(f)
s=fn2shp(f,index)
s=fn2shp(f,'byval',[800 820])
```

Description

FN2SHP generates an `imat_shp` object from the supplied `imat_fn` object. After checking attributes for consistency, it will assign the ordinate value of each coordinate to the corresponding `imat_shp` for each abscissa point specified. The `imat_shp` frequency will be set to the corresponding abscissa value. The output `imat_shp` object will be an nx1 shape, where n is the number of abscissa values in the `imat_fn`.

An optional input vector, `index`, can be used to specify the abscissa indices to use in creating the shape.

Another way of indexing involves passing in the string `'byval'`, followed by a list of frequencies or frequency ranges. If VALLIST is an Nx1 vector, FN2SHP will find the closest abscissa value to the values in VALLIST. If VALLIST is an Nx2 matrix, FN2SHP will treat each row as a start and end abscissa value, and will create a shape for each abscissa value within the specified range.

Examples

```
>> f=imat_fn(3,'ordinate',reshape(1:15,5,3));
>> f.responsenode=(1:3)'

f =
3x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)          Time Response    Even             5
2_(1X+,2X+)          Time Response    Even             5
3_(1X+,3X+)          Time Response    Even             5

>> [f(1).abscissa f.ordinate]                % f is 3x1 with 5 abscissa values

ans =
     1     1     6    11
     2     2     7    12
     3     3     8    13
     4     4     9    14
     5     5    10    15

>> shp=fn2shp(f)

shp =
5x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1                   0                   3
2   2                   0                   3
3   3                   0                   3
4   4                   0                   3
5   5                   0                   3

>> shp.shape

ans =
     1     2     3     4     5
     0     0     0     0     0
     0     0     0     0     0
     6     7     8     9    10
     0     0     0     0     0
     0     0     0     0     0
     11    12    13    14   15
     0     0     0     0     0
     0     0     0     0     0

>> shp=fn2shp(f,3);
>> shp.shape

ans =
     3
     0
     0
     8
```

```
           0
           0
          13
           0
           0

>>
```

See Also

imat_shp/shp2fn

---

imat_fn/get_units_labels

Purpose

Get units labels for the supplied `imat_fn`.

Syntax

```
str=get_units_labels(f)
str=get_units_labels(f,type)
str=get_units_labels(f,type,'full')
```

Description

GET_UNITS_LABELS returns a cell array of strings the size of the input `imat_fn` F which contains the units label string for each of the functions. TYPE specifies which axis to return the units labels for. If an empty string is supplied, the default type will be used. The optional input string `'full'` specifies whether to use the units abbreviations or their full names. Leaving this argument off uses abbreviations.

TYPE can be one of the following:

'abscissa'	Abscissa
'ordinate'	(default) Ordinate (numerator/denominator)
'ordnum'	Ordinate numerator
'ordden'	Ordinate denominator
'zaxis'	Z Axis

Examples

```
>> f=imat_fn(1,'ordnumdatatype','acceleration');
>> setunits('in');
>> get_units_labels(f)

ans =
    'in/s^2'
```

```
>> get_units_labels(f,'ordinate','full')

ans =
    'inch/second^2'

>>
```

See Also

imat_shp/get_units_labels, imat_result/get_units_labels

---

imat_fn/truncate

Purpose

Truncate an `imat_fn` ito the specified abscissa range.

Syntax

```
g=truncate(f,[1.5 2.5])
g=truncate(f,[1.5 2.5],'keep')
g=truncate(f,[2 5],'remove')
g=truncate(f,[1 3 6 7],'keep','index','resetstart')
```

Description

TRUNCATE will truncate the supplied `imat_fn` based on the specified abscissa range. F is an `imat_fn`. The second input is a vector containing the absicssa range or indices to keep or remove. If it is a 1x2 vector, then TRUNCATE treats the values as a range. Otherwise it assumes they are discrete values. If they are discrete abscissa values, TRUNCATE will select the closest actual abscissa value to the specified value.

Two optional input arguments are supported. The first specifies what to do with the supplied abscissa values. If set to `'keep'` (the default), TRUNCATE will keep all data values that lie between the two abscissa values specified in the range. If set to `'remove'`, TRUNCATE will remove all abscissa values that fall within this range (not including the data points specified in the input argument). The optional string `'index'` specifies that TRUNCATE should treat the supplied values as indices rather than abscissa values.

If you pass in the string `'resetstart'`, TRUNCATE will set the minimum abscissa value to `0`.

Examples

```
>> f=imat_fn(3);
>> f.ordinate=1:5;
>> f(2).abscissainc=0.5;
>> f(3).abscissa=[1 2 4 5.5 6];
>> f.abscissa

ans =
    1.0000    1.0000    1.0000
    2.0000    1.5000    2.0000
    3.0000    2.0000    4.0000
    4.0000    2.5000    5.5000
    5.0000    3.0000    6.0000

>> g=truncate(f,[1.5 2.5])

g =
3x1 IMAT Function with the following attributes:
Record Name                 FunctionType     AbscissaSpacing  NumberElement
--------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                 Time Response    Even             1
2_(1X+,1X+)                 Time Response    Even             3
3_(1X+,1X+)                 Time Response    Uneven           1

>> g.abscissa

ans =
    2.0000    1.5000    2.0000
       NaN    2.0000       NaN
       NaN    2.5000       NaN

>> g=truncate(f,[1.5 2.5],'remove')

g =
3x1 IMAT Function with the following attributes:
Record Name                 FunctionType     AbscissaSpacing  NumberElements
--------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                 Time Response    Uneven           4
2_(1X+,1X+)                 Time Response    Uneven           2
3_(1X+,1X+)                 Time Response    Uneven           4

>> g.abscissa

ans =
    1.0000    1.0000    1.0000
    3.0000    3.0000    4.0000
    4.0000       NaN    5.5000
    5.0000       NaN    6.0000
```

---

imat_fn/chgunits

Purpose

Convert an `imat_fn` to a different unit system.

Syntax

```
g=chgunits(f,from,to)
```

Description

CHGUNITS converts an `imat_fn` to a different system of units. This potentially affects the abscissa values (*Abscissa*, *AbscissaMin*, and *AbscissaInc* attributes), the ordinate values (*Ordinate*, *OrdOffsetReal*, *OrdOffsetImag*, *OrdScaleReal*, and *OrdScaleImag* attributes), and Z-axis values (*ZGeneralValue* attribute).

The `from` argument is the unit system that `f` is expressed in currently. The `to` argument is the desired unit system (if `to` is omitted, the current unit system is assumed). The unit system arguments should either be unit strings (such as `'SI'`) or should be 5x1 vectors of unit conversion factors as returned by the getunits function.

If you follow the recommended practice of setting your units at the start of a session and maintaining consistency, you should not need to use this function.

Examples

```
>> f=chgunits(f,'mm','in');      % Change f from MM to IN
```

See Also

getunits, setunits

---

imat_fn/plot

Purpose

Plot an `imat_fn` in a special figure window.

Syntax

```
plot(f)
plot(f1,f2,...)
h=plot(f)
h=plot(f,handle)
plot(f,'xscale','lin','yscale','log','grid','none','complex','mp','xwin',[100 200],'ywin',[1e-1
1e4],'tag',tagstruct,'allphase')
```

Description

When you PLOT an `imat_fn`, the IMAT toolbox brings up the XY Plot window and plots all functions in `f`. Plot options (log/linear, etc.) are controlled through a graphical user interface.

If a figure handle is supplied, PLOT will display the plot(s) inside the existing figure rather than creating a new one. If a uipanel handle is supplied, the plot will be displayed inside the panel. If HI is an axis handle, PLOT will create a uipanel over it using its OuterPosition for the size and location, and will plot the functions in this uipanel.

You can control the plot axis scaling, grid, complex options, and window either through the GUI interface or through command-line arguments. The command-line arguments come in pairs. The first value of the pair is a string specifying the attribute, and the second is a value containing the value of the attribute. The following table summarizes the supported strings and their possible values.

By default PLOT will display a "cleaned" phase plot. Points where the phase differs by more than 270 degrees from the previous point are not displayed. This behavior can be controlled through the Options menu or through a supplied format string.

<table>
<tr><td><code>'legend'</code></td><td>Set the legend type (<code>'idline1'</code>,<code>'idline4'</code>,<code>'ref/res'</code>).</td></tr>
<tr><td><code>'xscale'</code></td><td>Specify <code>'lin'</code> for linear axis or <code>'log'</code> for logarithmic axis.</td></tr>
<tr><td><code>'yscale'</code></td><td>Specify <code>'lin'</code> for linear axis or <code>'log'</code> for logarithmic axis.</td></tr>
<tr><td><code>'grid'</code></td><td>Specify <code>'x'</code> for x-only grid, <code>'y'</code> for y-only, <code>'xy'</code> for x and y grid, and <code>'none'</code> for no grid</td></tr>
<tr><td><code>'complex'</code></td><td>Specify <code>'mp'</code> for modulus + phase, <code>'m'</code> for modulus only, <code>'p'</code> for phase only, <code>'r'</code> for real only, <code>'i'</code> for imaginary only, <code>'ri'</code> for real + imaginary, and <code>'nyq'</code> for Nyquist</td></tr>
<tr><td><code>'xwin'</code></td><td>Specify a 1x2 numeric vector containing the minimum and maximum X axis window values. The string <code>'tight'</code> is also valid.</td></tr>
<tr><td><code>'ywin'</code></td><td>Specify a 1x2 numeric vector containing the minimum and maximum Y axis window values. The string <code>'tight'</code> is also valid.</td></tr>
<tr><td><code>'tag'</code></td><td>nx1 structure containing the following fields:
<table>
<tr><td><code>coord</code></td><td>If doing tag on data, set to the X value. If doing tag on grid, set to [x_value y_value].</td></tr>
<tr><td><code>axis</code></td><td>Specifies the axis number for the tag (either 1 or 2). Axis 2 is the upper axis.</td></tr>
<tr><td><code>func</code></td><td>If doing tag on data, set to the function number to tag. Otherwise set to [].</td></tr>
<tr><td><code>type</code></td><td>Specifies tag type: 'x', 'y', or 'xy'.</td></tr>
<tr><td><code>formatx</code></td><td>[Optional]. If supplied, specifies the format string for X values. The format specifier should set the format for a single numeric value. The default is <code>'   %g'</code>.</td></tr>
<tr><td><code>formaty</code></td><td>[Optional]. If supplied, specifies the format string for Y values. The format specifier should set the format for a single numeric value. The default is <code>'   %g'</code>.</td></tr>
</table>
</td></tr>
<tr><td><code>'allphase'</code></td><td>Display all of the phase values on phase plots. The default is <code>false</code>.</td></tr>
<tr><td><code>'template'</code></td><td>Specify a template file to apply to the plot (follow with path to valid template file).</td></tr>
</table>

You can pass in a string that contains valid MATLAB characters for line color, linestyle, and marker. See the help for PLOT for a list of valid characters. This will cause all of the functions to use those attributes for the line display.

`H` is an optional output that contains the IMAT_FNPLOT object defining the plot. You can access the methods and properties of this object to modify the plot. The IMAT_FNPLOT class is documented here.

See Also

imat_fn/plot3, imat_fnplot

IMAT_FNPLOT CLASS

Description

The IMAT_FNPLOT class is a class that consists of a number of methods and properties that provide convenient control of an `imat_fn` plot. When a plot is created, its definition is stored in an IMAT_FNPLOT object, which allows you to change the plot display after the plot has been generated. The axes associated with the plot are associated with a UIPANEL. This allows the panel to be embedded in other GUIs and manipulated accordingly (see UIPLOT). This class also offers a lot of convenience for plotting. For example, you can link different plots together, so that changing an attribute of one (for example the window) will automatically change the axes on the other linked plots.

A contextual menu is created on all IMAT_FNPLOT plots. The menu provides convenient access to the majority of the IMAT_FNPLOT methods. For a complete description of each menu item, see the contextual menu section in the imat_fn/plot documentation.

Templates are a functionality that was once exclusive to UIPLOT. They have now been added to the basic function plotting capabilities. See the imat_fn/plot documentation for more information.

An important thing to understand about an IMAT_FNPLOT is that it consists of three axes that never get destroyed. The first axes of the three is always either the real or magnitude plot. The second axis is always the imaginary or phase plot. The third axis is always the Nyquist plot. Only the axes that are relevant to the currently displayed Complex Option are visible.

Reference Guide

The IMAT_FNPLOT has both properties and methods available to you.

Properties

The top level properties of the IMAT_FNPLOT object are the most basic to its function. The top level properties are series of handles to the different components of the plot, as well as a structure that contains the complete information about its status and format. These properties are described in the table below.

`hpanel`	Handle to uipanel that contains the entire plot.
`ha`	Handles (1x3) to the three axes that comprise the plot.

`hf`	Handle to the figure that contains the plot.
`hl`	Cell array (1x3) containing the handles to the lines on each of the three axes that comprise the plot.
`f`	`imat_fn` object containing the functions that are plotted.
`s`	Structure containing information about the plot settings.

S Structure Fields

The fields in the `.s` field of the IMAT_FNPLOT object contain a complete description of the current status/formatting of the IMAT_FNPLOT display. It should be considered 'read-only', as any changes will not affect the plot itself. Only the object methods functions called from the contextual menu affect the contents of this field.

Attributes with size 1x3 correspond to the three classifications of plots. That is: `'modulus + phase'`, `'real + imaginary'`, and `'nyquist'`. Items such as labeling, windowing, and scaling are stored on an individual basis for the three different types of plots.

`title`	String containing the plot title text. If it is `'default '`, the title is auto-generated based on the type of data shown on the plot. If it is anything else, it overrides the default setting.
`xlabel`	Cell array (1x3) of strings that defines the X axis label for each of the three axes. If it is `'default '`, the labels are auto-generated based on the type of data shown as well as the type of plot. For instance, in a magnitude/phase plot the x-label would appear only on the magnitude axis, and not the phase axis.
`ylabel`	Cell array (1x3) of cell arrays. They contain the Y axis labels for each axis of the three plot classifications. The first two cells are size 1x2 since they are multi-axis plots.
`xscale`	Cell array (1x3) of strings containing the X axis scale for each of the three plot types.
`yscale`	Cell array (1x3) of strings containing the X axis scale for each of the three plot types.
`complex`	String which defines the type of plot. The available types are `'modulus + phase'`,`'modulus'`, `'phase'`, `'real + imaginary'`, `'real'`, `'imaginary'`, and `'nyquist'`.
`window`	Cell array (1x3) of cell arrays storing the window information for each of the three plot classifications. The window information specifies the X and Y limits of the axes. The first two classifications are stored as {xwin ywin1 ywin2}. Nyquist is stored as {xwin ywin}. X-windowing is always the same for multi-axis plots.
`windowhistory`	1x3 structure containing the windowing history for each axis type. The field `.val` contains the windowing information, and the field .ind specifies the current index into .val.
`gridopt`	String which defines which grid lines are shown on the plot. The available options are `'X only'`, `'Y only'`, `'X and Y'`, and `'none'`.
`linkplots`	Logical value which defines whether this plot listens when a notification is sent out indicating another plot has changed.
`linkaxes`	Logical value which defines whether the three axes that make up a plot listen to each other. It is advisable not to turn this off. If this is turned off, axes that belong together (such as magnitude and phase) will not act together.
`cleanphase`	Logical value defining whether the phase has been cleaned.
`tag`	Structure that completely defines any tags that have been applied to the plot.
`style`	Structure that completely defines the formatting of the lines on the plot. This field will be empty until the STYLE method is used. Each style that has been set will then appear as a field of a structure in `.style`. This structure will be an array of the same length as the number of functions currently plotted.

`axis`	Structure that defines the formatting of the axes on the plot. This field will be empty until the AXIS method is used. Each axis property that has been set will then appear as a field of the structure in `.axis`.
`border`	Structure that defines the plot borders in pixels.
`font`	Structure that completely defines the formatting of the various fonts on the plot. It is described in more detail below.
`legend`	Structure that completely defines the legend on the plot. It is described in more detail below.
`xtick`	Structure that completely defines the formatting of the X axis ticks.
`ytick`	Structure that completely defines the formatting of the Y axis ticks.

Tag Fields

`type`	String defining whether the X or Y data (or both) is displayed on the plot when tagging. This can be `'x'`, `'y'`, or `'xy'`.
`cursoron`	Logical value specifying whether cursor is currently on.
`loc`	String defining whether tagging happens on data or grid. This can be `'data'` or `'grid'`.
`cursorh`	Handle to the cursor line if cursoring is currently on.
`title`	String storing the original title of the plot before tagging began.
`tags`	Structure containing information on each of the tags, such as which function the tag is one, what the type of the tag was, etc.
`h`	Handles to the text and marker for each tag.
`print`	Logical value defining whether the tag coordinate will be printed to the command window.

Font Fields

`title`	Structure containing the title font attributes and their current settings.
`xlabel`	Structure containing the X axis label font attributes and their current settings.
`ylabel`	Structure containing the Y axis label font attributes and their current settings.
`axis`	Structure containing the axis font attributes and their current settings.
`default`	Structure containing the default font attributes and their current settings. When fonts are set back to their defaults, they will use these settings.

Legend Fields

`show`	This is a logical flag defining whether the legend is shown.
`type`	This is the type of legend that is shown. There are several types: `'default '`, `'idline1'`, `'idline4'`, `'ref/res'`, and `'custom'`.
`handle`	This is the handle to the legend.
`n`	This is the maximum number of legend items to display when generating the legend. This is not used when

	the user manually enters a custom legend.
`custom`	This is a cell array of strings that define the entries for the custom legend. This is only populated if the type is set to `'custom'`.
`location`	This is the location of the legend. The available options are the same as MATLAB's LEGEND function.
`orientation`	This is the orientation of the legend It is either `'Horizontal'` or `'Vertical'`.
`font`	Structure containing the legend font settings: `FontName` — Font name (e.g. `'Arial'`). Must be a valid MATLAB-supported value. `FontWeight` — Font weight (e.g. `'bold'`). Must be a valid MATLAB-supported value. `FontAngle` — Font angle (e.g. `'italic'`). Must be a valid MATLAB-supported value. `FontSize` — Font size in points. Must be a valid MATLAB-supported value.
`interpreter`	This specifies the interpreter to use for legend labels. The available options are the same as MATLAB's LEGEND function.

Methods

A number of methods are available for the IMAT_FNPLOT class. These methods allow you to manipulate the plot after it has been generated. In the documentation below, the IMAT_FNPLOT object is denoted by HPLOT.

HPLOT.applytemplate([tmpl][fname])

This method applies a template stored in the template structure TMPL or in the file FNAME to the plot. If neither is input, a file dialog will be shown where the template file can be selected.

HPLOT.axis(attribute1,val1,attribute2,val2,...,)

This method changes the axis properties of the axes that are displayed on an IMAT_FNPLOT. This method directly controls the style of the lines on an IMAT_FNPLOT. ATTRIBUTE1, ATTRIBUTE2, etc are attributes such as `'TickDir'` and `'Box'`. Attributes must be followed by their corresponding values. They are applied as attribute pairs as they would on any set() command. All of the axes on the plot will use these settings.

If other IMAT_FNPLOTs on the same figure are linked (they are by default), then the axes of those plots will also be changed by using this method.

HPLOT.complex(option)

This method changes the complex option of the IMAT_FNPLOT. The allowed options are: `'magnitude + phase'`, `'magnitude'`, `'phase'`, `'real + imaginary'`, `'real'`, `'imaginary'`, and `'nyquist'`.

If other IMAT_FNPLOTs on the same figure are linked (they are by default), then the complex option of those plots will also be changed by using this method.

HPLOT.font(['title','xlabel','ylabel','axis'][,'name',name][,'weight',weight]['angle',angle][,'size',size]['color',color])

This method changes the font of the title, xlabel, ylabel, or axes. The first argument defines which of these are being defined. The following arguments define the name, weight, and angle, size, and color of the font. All don't need to be defined. If one isn't defined, the current font will be used as the starting point.

The name can be any valid font name on the system. The weight is either `'bold'` or `'normal'`. The angle is either `'italic'` or `'normal'`. The size is any real number greater than one.

HPLOT.grid(option)

This method changes the grid lines that are displayed on an IMAT_FNPLOT. The available options are:

`'on'`	Turn on both x and y grid lines
`'off'`	Turn off all grid lines
`'x and y'`	Turn on both x and y grid lines (same as `'on'`)
`'x only'`	Turn on only the x grid lines
`'y only'`	Turn on only the y grid lines
`'none'`	Turn off all grid lines (same as `'off'`)

If other IMAT_FNPLOT on the same figure are linked (they are by default), then the grid lines of those plots will also be changed by using this method.

HPLOT.legend([string1,string2,string3,...][,'Location',location][,'Orientation',orientation][,'Type',type]['Interpreter',interp][,'Visible',visible]['FontName',FN][,'FontWeight',FW][,'FontAngle',FA][,'FontSize',FS][,numlegend])

This method functions much like the standard MATLAB LEGEND function. This one is just used specifically with IMAT_FNPLOT to enable advanced features.

The legend strings may be supplied as a series of string arguments, or as a cell array of strings. Strings specified that do not otherwise match a valid option are assumed to be the labels for the lines on the plot. You can use `%ResponseCoord`, `%ReferenceCoord`, `%Name`, `%IDLine1`, `%IDLine2`, `%IDLine3`, `%IDLine4`, `%FunctionType`, and `%RMS` as part of the legend and they will get replaced with the corresponding function values. The `'Location'`, `'Visible'`, `'Orientation'`, `'Interpreter'`, `'FontName'`, `'FontWeight'`, `'FontAngle'`, and `'FontSize'` arguments act the same a sMATLAB's legend function.

The `'Type'` argument is an added feature beyond the normal legend function. It determines how the legend labels are automatically generated. The allowed options are `'default '`, `'idline1'`, `'idline4'`, and `'ref/res'`. If custom string labels are entered, this is automatically set to `'custom'` and should not be entered.

Passing in a positive numeric scalar NUMLEGEND specifies the number of legend entries to display on the plot.

HPLOT.link(true/false)

This method sets the linking of the plot object. If TRUE, then whenever another IMAT_FNPLOT on the same figure sends its notification that it has changed, it will apply the same change. If FALSE, it simply ignores the notification. It is important to note, that all imat_fnplots on the same figure are linked by default.

HPLOT = HPLOT.loadfig([filename])

This method loads the figure data from a special figure file and recreates the figure from the information stored in the file. This functionality is the IMAT_FNPLOT equivalent of loading from a .fig file.

FILENAME is an optional string or Nx2 cell array of strings. If it is a cell array of strings or it contains wildcards, the user will be prompted for the filename with a graphical file dialog.

HPLOT.print()

This method prints the current plot to the printer, using MATLAB's print dialog. If you select *File* -> *Print* from the figure, it will bring up the print dialog. Otherwise, it will bring up the print preview dialog.

HPLOT.remove()

The REMOVE method removes the plot from the figure. It does all of the clean up that is necessary for a plot's deletion not to impact the stablity of the rest of the figure.

HPLOT.replace(G)

The replace method replaces the functions that are plotted on the IMAT_FNPLOT with those in the `imat_fn`, G, keeping the format as similar as possible. If the number of functions in G is less than or equal to the number currently plotted, the formats will be applied in order. If the number of functions in G exceeds the number of formats currently plotted, the existing formatting will be applied for all possible, with the extras having the default formatting.

HPLOT.saveas([filename])

This method saves the IMAT_FNPLOT to a file. It is called when you select *File* -> *Save As* from the figure menu. If FILENAME is not supplied, or is a cell array of strings, or contains wildcards, the user will be prompted with a graphical file dialog.

HPLOT.savefig([filename])

This method saves the IMAT_FNPLOT to a special figure file that can then be loaded to recreate the figure. This functionality is the IMAT_FNPLOT equivalent of saving to a `.fig` file.

FILENAME is an optional string or Nx2 cell array of strings. If it is a cell array of strings or it contains wildcards, the user will be prompted for the filename with a graphical file dialog.

HPLOT.savetemplate([filename])

This method saves the formatting of the current plot to a template file. A filename to save the template to, FILENAME, can be optionally passed in as well.

HPLOT.style(attribute1,val1,attribute2,val2,...,[,index])

This method directly controls the style of the lines on an IMAT_FNPLOT. ATTRIBUTE1, ATTRIBUTE2, etc are properties such as `'LineWidth'`, `'Marker'`, or `'LineStyle'`. You can set any of the Primitive Line Properties available in MATLAB. Please refer to the MATLAB documentation for more details.

Properties must be followed by their corresponding values. They are applied as property/value pairs as they would on any set() command. The INDEX argument controls which lines the properties will apply to. The index directly corresponds to the order of the functions as they were initially plotted (and as stored in the function as they were passed in). If the index is not supplied, then the style properties will be applied to all of the lines on the plot.

HPLOT.tag_style([index,]attribute1,val1,attribute2,val2,...)

This method directly controls the style of the tags on an IMAT_FNPLOT. ATTRIBUTE1, ATTRIBUTE2, etc are attributes such as 'FontSize', `'FontName'`, or 'Color'. They are applied as attribute pairs as they would on any set() command.The INDEX argument controls which tags the attributes will apply to. The index directly corresponds to the order of the tags as they were initially displayed. If the index is not supplied, then the style commands will apply to all of the tags on the plot.

HPLOT.title(title[,'Property1',value1,'Property2',value2,...])

This method adds a title to an imat_fnplot. If the complex option of the plot is changed, the title will always appear over the correct axis.

You can use `%ResponseCoord`, `%ReferenceCoord`, `%Name`, `%IDLine1`, `%IDLine2`, `%IDLine3`, `%IDLine4`, and `%FunctionType` as part of the label and they will get replaced with the value from the first function plotted.

In addition to the title, you can also specify Property/Value pairs. Valid pairs are any that MATLAB's TITLE function accepts. These properties and their values are stored as fields in `hplot.s.font.title`.

If other imat_fnplots on the same figure are linked (they are by default), then the xlabel of those plots will also be changed by using this method (assuming they are the same complex option).

HPLOT.uistyle()

This method opens a window where the line color, style and width can be edited on a line by line basis.

To edit the line style, select the line from the table and change the entries in the pulldowns in the table. Once the style values are set, press **Apply** to set the changes to the plot. When completely done, press **Done** to set the changes and close the window.

HPLOT.xlabel(label[,'Property1',value1,'Property2',value2,...])

This method changes the xlabel of an imat_fnplot. It works slightly differently than the typical MATLAB xlabel. Here, LABEL is the desired label for the plot. If it is entered without any associated INDEX, then the label is applied to all of the axes of the plot. If an index is specified, that label is applied to that specific axis only. INDEX can range from 1 to 3. An index of 1 corresponds to the `'magnitude'` or `'real'` axis. An index of 2 corresponds to the `'phase'` or `'imaginary'` axis. An index of 3 corresponds to the `'nyquist'` axis.

LABEL can be entered as a cell array of strings as long as INDEX is a vector of integers of the same size.

You can use `%ResponseCoord`, `%ReferenceCoord`, `%Name`, `%IDLine1`, `%IDLine2`, `%IDLine3`, `%IDLine4`, and `%FunctionType` as part of the label and they will get replaced with the value from the first function plotted.

In addition to the xlabel, you can also specify Property/Value pairs. Valid pairs are any that MATLAB's XLABEL function accepts. These properties and their values are stored as fields in `hplot.s.font.xlabel`.

If other imat_fnplots on the same figure are linked (they are by default), then the xlabel of those plots will also be changed by using this method (assuming they are the same complex option).

HPLOT.xlim(xlim[,index])

This method changes the x limits of an imat_fnplot. XLIM is a 1x2 vector defining the new limits. If XLIM is entered without any associated INDEX, then the limit is applied to the current complex option. If an index is specified, the limit is applied to that specific axis only. INDEX can range from 1 to 3. An index of 1 corresponds to the 'magnitude' or 'real' axis. An index of 2 corresponds to the 'phase' or 'imaginary' axis. An index of 3 corresponds to the `'nyquist'` axis.

Passing INF in for XLIM will fit the X limits to the data plotted, which is the same as the `'tight'` option. Passing in `'auto'` or [] for XLIM sets the limit to `'auto'`.

If other imat_fnplots on the same figure are linked (they are by default), then the limits of those plots will also be changed by using this method (even if they are not the same complex option).

HPLOT.xscale(scale)

This method changes the X axis scale of the currently visible plot.

SCALE is a string containing `'lin'`, `'log'`, or `'default'`.

If other IMAT_FNPLOTs on the same figure are linked (they are by default), then the X axis scale of those plots will also be changed by using this method (assuming they are the same complex option).

HPLOT.xtick([,'value',value][,'label',label][,'valuemode',valuemode][,'labelmode',labelmode])

This method sets the formatting of the xticks on an IMAT_FNPLOT. The xtick values and labels can be set manually using the `'value'` and `'label'` arguments, or they can be set so MATLAB sets them automatically using the `'valuemode'` and/or `'labelmode'` options.

The VALUE argument must be a numeric vector. This is the location where tick marks will be placed. The LABEL argument is a manual override of how the ticks are labeled. This must be a cell array of strings. If the size of the cell array is less than the number of ticks, then the cell array will be repeated so that all ticks have a label. If the size of the cell array is more than the number of ticks, only the labels up to the number of ticks will be used, starting from the beginning of the cell array.

VALUEMODE and LABELMODE can be set to `'manual'` or `'automatic'`. If values or labels are set, then these modes are set to `'manual'` automatically.

It is important to note that the xticks will automatically be applied to all axes associated with the current complex option.

HPLOT.ylabel(label[,index][,'Property1',value1,'Property2',value2,...])

This method changes the ylabel of an imat_fnplot. It works slightly differently than the typical MATLAB ylabel. Here, LABEL is the desired label for the plot. If it is entered without any associated INDEX, then the label is applied to all of the axes of the plot. If an index is specified, that label is applied to that specific axis only. INDEX can range from 1 to 3. An index of 1 corresponds to the `'magnitude'` or `'real'` axis. An index of 2 corresponds to the `'phase'` or `'imaginary'` axis. An index of 3 corresponds to the `'nyquist'` axis.

LABEL can be entered as a cell array of strings as long as INDEX is a vector of integers of the same size.

You can use `%ResponseCoord`, `%ReferenceCoord`, `%Name`, `%IDLine1`, `%IDLine2`, `%IDLine3`, `%IDLine4`, and `%FunctionType` as part of the label and they will get replaced with the value from the first function plotted.

In addition to the ylabel, you can also specify Property/Value pairs. Valid pairs are any that MATLAB's YLABEL function accepts. These properties and their values are stored as fields in `hplot.s.font.ylabel`.

If other imat_fnplots on the same figure are linked (they are by default), then the xlabel of those plots will also be changed by using this method (assuming they are the same complex option).

HELOT.ylim(ylim[,index])

This method changes the y limits of an IMAT_FNPLOT. Here, YLIM is the desired y limits for the plot. If it is entered without any associated INDEX, then the limits will be applied as appropriately as possible. That is, they will be applied to all axes shown, except in the case of a magnitude/phase plot, where only the magnitude will be windowed.

If an index is specified, then the y limits are applied to that specific axis only. INDEX can range from 1 to 3. An index of 1 corresponds to the `'magnitude'` or `'real'` axis. An index of 2 corresponds to the `'phase'` or `'imaginary'` axis. An index of 3 corresponds to the `'nyquist'` axis.

YLIM can be entered as a cell array of strings as long as INDEX is a vector of integers of the same size.

Passing INF in for YLIM will fit the Y limits to the data plotted, which is the same as the `'tight'` option. Passing in `'auto'` or [] for YLIM sets the limit to `'auto'`.

If other imat_fnplots on the same figure are linked (they are by default), then the y limits of those plots will also be changed by using this method (assuming they are the same complex option).

HPLOT.yscale(scale,axind)

This method changes the Y axis scale of the currently visible plot. Note that you cannot change the scale of the phase axis on a magnitude/phase plot.

SCALE is a string containing `'lin'`, `'log'`, or `'default'`.

AXIND is the index of the axis to change. For multi-axis plot (e.g. magnitude/phase), 1 is the bottom axis and 2 is the top axis.

If other IMAT_FNPLOTs on the same figure are linked (they are by default), then the X axis scale of those plots will also be changed by using this method (assuming they are the same complex option).

HPLOT.ytick(index[,'value',value][,'label',label][,'valuemode',valuemode][,'labelmode',labelmode])

This method sets the formatting of the yticks on an IMAT_FNPLOT. The ytick values and labels can be set manually using the `'value'` and `'label'` arguments, or they can be set so MATLAB sets them automatically using the `'valuemode'` and/or `'labelmode'` options.

INDEX is a required argument that defines which axis the yticks will be applied to. It will only be stored for that axis in the current complex option. An index of 1 corresponds to the `'magnitude'` or `'real'` axis. An index of 2 corresponds to the `'phase'` or `'imaginary'` axis. An index of 3 corresponds to the `'nyquist'` axis.

The VALUE argument must be a numeric vector. This is the location where tick marks will be placed. The LABEL argument is a manual override of how the ticks are labeled. This must be a cell array of strings. If the size of the cell array is less than the number of ticks, then the cell array will be repeated so that all ticks have a label. If the size of the cell array is more than the number of ticks, only the labels up to the number of ticks will be used, starting from the beginning of the cell array.

VALUEMODE and LABELMODE can be set to `'manual'` or `'automatic'`. If values or labels are set, then these modes are set to `'manual'` automatically.

It is important to note that the xticks will automatically be applied to all axes associated with the current complex option.

See Also

imat_fn/plot

---

imat_fn/plotyy

Purpose

Double Y-axis plot.

Syntax

```
[hf,hg] = plotyy(f,g)
```

Description

PLOTYY displays the `imat_fn`s in F and G on a double Y-axis plot. The Y axis label for F appears on the left, and the Y axis label for G appears on the right.

HF and HG are IMAT_FNPLOT objects for the plots of F and G, respectively.

See Also

imat_fn/plot, imat_fnplot

---

imat_fn/plot3

Purpose

Plot an `imat_fn` on a 3D axis in a special figure window.

Syntax

```
plot3(f)
plot3(f1,f2,...)
h=plot3(f)
[h,ha]=plot3(f,handle)
plot3(f,'xscale','linear','yscale','log','grid','','complex','m','xwin',[100 200],'ywin',[1e-1
1e4],'tags',[100 1; 200 2],'silent')
```

Description

PLOT3(F, ...) plots all the functions in `F` as a sequence of adjacent lines in 3 dimensions.`H` is an optional output containing the figure handle. `HA` is an optional output containing the axis handle.

If an axis or figure handle is supplied, PLOT3 will draw the new plot into the existing axis or figure.

You can control the plot formatting by passing in the following string or strings followed by the appropriate value. To save typing, you can use only a prefix of the option name, as long as that that prefix is unique.

`'style'`	Choose the plot style (`'wires'`, `'plates'`, `'surface'`). Wires plots each function as a single line. Plates displays functions as parallel filled plates. Surface covers the functions with a faceted mesh.

`'complex'`	Set the complex option. Valid choices are `'m'` Modulus (amplitude). `'p'` Phase in degrees. Append `'c'` to enable Cleaned Phase (i.e. `'pc'`). `'r'` Real portion. `'i'` Imaginary portion.
`'tags'`	Draw tags into the plot window. To tag on data, provide an Mx2 matrix with [x y] values in each row. To tag in space, provide an Mx3 matrix with rows containing [x y z]. You may also tag on data using an Mx3 with rows containing [x y Inf].
`'taglbls'`	Choose which dimensions (`'x'`,`'y'`,`'z'`) to display in the text label for each tag. (i.e. `'xyz'`,`'zyx'`,`'xz'`). This setting applies to all tags which come after it, but not before.
`'xwindow'`	Set the X axis window range. Follow with a 1x2 vector containing the minimum and maximum window values. Use -Inf and/or Inf to enable automatic tight fitting in that direction. The default is [-Inf,Inf] for all dimensions.
`'ywindow'`	Set the Y axis window range. Follow with a 1x2 vector containing the minimum and maximum window values. Use -Inf and/or Inf to enable automatic tight fitting in that direction. The default is [-Inf,Inf] for all dimensions.
`'zwindow'`	Set the Z axis window range. Follow with a 1x2 vector containing the minimum and maximum window values. Use -Inf and/or Inf to enable automatic tight fitting in that direction. The default is [-Inf,Inf] for all dimensions.
`'xscale'`	Set the X axis type (`'linear'` or `'log'`)
`'yscale'`	Set the Y axis type (`'linear'` or `'log'`)
`'zscale'`	Set the Z axis type (`'linear'` or `'log'`)
`'grid'`	Set the grid option. Use any combination of `'x'`,`'y'`, and `'z'`. (i.e. `''`, `'xy'`, `'xz'`, or `'xyz'`)
`'renderer'`	Select a graphics renderer, either `'opengl'` or `'zbuffer'`. OpenGL is typically faster, but it does not currently support Log scales. Each renderer has its own quirks, so you may want to try both, depending on your needs.
`'silent'`	Suppress message output.

All plot options (except tag in space) are also available through the GUI's context menus.

Examples

```
>> plot3(f,'style','plates','complex','p','grid','xyz')     % Plate plot of phase with X,Y, and Z gridlines
```

See Also

imat_fn/plot

---

imat_fn/uiselect

Purpose

Select functions using a graphical interface.

Syntax

```
g=uiselect(f)
g=uiselect(f,'Title')
g=uiselect(f,[presel])
g=uiselect(f,'adfsel')
g=uiselect(f,{'Attrib1','Attrib2','Attrib3'})
[g,ind]=uiselect(f)
[g,ind]=uiselect(f,'Title',[presel])
[g,ind]=uiselect(f,'Title',imat_filt('Functiontype','=','Time response'))
```

Description

The UISELECT function brings up a function selection form similar to the IMAT Function selection form, listing all elements of the `imat_fn`. The user can select elements of `f`, then click on 'OK'. The selected elements are returned in the output argument `g`. If a title string is provided, it is used to name the function selection window. If a numeric vector is provided, the functions corresponding to the indices in the vector will be preselected when the form is displayed. A third optional argument, an `imat_filt`, can also be supplied. If supplied, the filter will be turned on, and the initial function list will be filtered. The optional second output argument `ind` will contain the indices into `f` of the functions selected and returned in `g`. The attributes listed in the attribute columns on the form will default to the attributes set by SETDISPLAY. Alternately, you can specify them by passing in a cell array of strings of attributes to display.

The optional input string `'adfsel'` will enable and display the ADF Selection button on the form. It is not displayed by default.

If the user clicks on 'Cancel', then `g=-1` is returned.

The function selection window also provides buttons named "ADF Selection" and "Filter". The "ADF Selection" button brings up a file dialog, and adds records from the selected ADF to the function selection form. The "Filter" button allows you to build a filter which controls which functions are displayed.

See Also

imat_shp/uiselect, imat_shp/setdisplay

---

imat_fn/uplus

Purpose

Unary plus (`+f`).

Syntax

```
+f
```

Description

The UPLUS operation does not change an `imat_fn`. It is included in the toolbox for completeness.

See Also

imat_fn/uminus

---

imat_fn/uminus

Purpose

Unary minus (`-f`).

Syntax

```
-f
```

Description

The minus of an `imat_fn` is obtained by negating all ordinate values.

Examples

```
>> f(5)=-f(5);    % Flip sign of the 5th function
```

See Also

imat_fn/uplus

---

imat_fn/plus

Purpose

Add ordinates (`f1+f2`).

Syntax

```
f1+f2
f+x
x+f
```

Description

The following addition operations are possible with `imat_fn` objects:

- Adding two `imat_fn` objects causes their ordinate values to be added. If one of the operands is a single function (1x1), then its ordinate will be added to all elements in the other operand. If both operands are function arrays, they must have the same dimensions. The data attributes of the result of an addition operation will be taken from the left operand. No checking is performed to assure that the ordinate attributes of the two operands are consistent.

- Adding a numeric array or scalar to an `imat_fn` causes the numeric values to be added to the ordinate values of the `imat_fn`.

Examples

```
>> f=imat_fn(3,'ordinate',reshape(1:15,5,3))

f =
3x1 IMAT Function with the following attributes:
Record Name          FunctionType     OrdNumDataType   NumberElements
-------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)          Time Response    Unknown          5
2_(1X+,1X+)          Time Response    Unknown          5
3_(1X+,1X+)          Time Response    Unknown          5

>> f.ordinate

ans =
     1     6    11
     2     7    12
     3     8    13
     4     9    14
     5    10    15

>> g=f+f; g.ordinate

ans =
     2    12    22
     4    14    24
     6    16    26
     8    18    28
    10    20    30

>> g=g+g(1); g.ordinate

ans =
     4    14    24
     8    18    28
    12    22    32
    16    26    36
    20    30    40

>> g=g+1; g.ordinate

ans =
     5    15    25
     9    19    29
    13    23    33
    17    27    37
    21    31    41

>>
```

See Also

imat_fn/minus

---

imat_fn/minus

Purpose

Subtract ordinates (`f1-f2`).

Syntax

```
f1+f2
f-x
x-f
```

Description

The following subtraction operations are possible with `imat_fn` objects:

- Subtracting two `imat_fn` objects causes their ordinate values to be subtracted. If one of the operands is a single function (1x1), then its ordinate will be used with all elements in the other operand. If both operands are function arrays, they must have the same dimensions. The data attributes of the result of a subtraction operation will be taken from the left operand. No checking is performed to assure that the ordinate attributes of the two operands are consistent.
- Subtracting a numeric array or scalar from an `imat_fn` causes the numeric values to be subtracted from the ordinate values of the `imat_fn`.
- Subtracting an `imat_fn` from a numeric array or scalar causes the ordinate values of the `imat_fn` to be subtracted from the numeric values.

Examples

```
>> f=imat_fn(3,'ordinate',reshape(1:15,5,3))

f =
3x1 IMAT Function with the following attributes:
Record Name          FunctionType     OrdNumDataType   NumberElements
-------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)          Time Response    Unknown          5
2_(1X+,1X+)          Time Response    Unknown          5
3_(1X+,1X+)          Time Response    Unknown          5

>> f.ordinate

ans =
     1     6    11
     2     7    12
     3     8    13
     4     9    14
     5    10    15

>> g=f-3; g.ordinate

ans =
    -2     3     8
    -1     4     9
     0     5    10
     1     6    11
     2     7    12

>> g=g-[-1 0 3]; g.ordinate

ans =
    -1     3     5
     0     4     6
     1     5     7
     2     6     8
     3     7     9

>>
```

See Also

imat_fn/plus

---

imat_fn/times

Purpose

Termwise multiply ordinates (`f1.*f2`).

Syntax

```
f1.*f2
f.*x
```

Description

The following termwise multiplication operations are possible with `imat_fn` objects:

- Multiplying two `imat_fn` objects causes their ordinates to be multiplied termwise. If one of the operands is a single function (1x1), then its ordinate will be multiplied by all elements in the other operand. If both operands are function arrays, they must have the same dimensions. The data attributes of the result of a termwise multiplication will be taken from the left operand, except that ordinate data types (i.e., units exponents) will correctly reflect the combination of the two operands.
- Multiplying a numeric array or scalar times an `imat_fn` causes the numeric values to be termwise multiplied into the ordinate values of the `imat_fn`.

Examples

```
>> f=imat_fn(3,'ordinate',reshape(1:15,5,3))

f =
3x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)          Time Response    Even             5
2_(1X+,1X+)          Time Response    Even             5
3_(1X+,1X+)          Time Response    Even             5

>> f.ordinate

ans =
     1     6    11
     2     7    12
     3     8    13
     4     9    14
     5    10    15

>> g=f.*f; g.ordinate

ans =
     1    36   121
     4    49   144
     9    64   169
    16    81   196
    25   100   225

>> g=g.*g(1); g.ordinate

ans =
           1          36         121
          16         196         576
          81         576        1521
         256        1296        3136
         625        2500        5625

>>
```

See Also

imat_fn/plus, imat_fn/minus, imat_fn/ldivide, imat_fn/rdivide

---

imat_fn/mtimes

Purpose

Matrix multiply ordinates (`A*f`).

Syntax

```
g=A*f
g=f*A
```

Description

Matrix multiplication between a numeric array `A` and an `imat_fn` object `f` is performed repeatedly at all abscissa/ordinate values. This type of operation requires that all elements of `f` have the same abscissa values, and that the column dimension of the left operand match the row dimension of the right operand. At each abscissa value, a matrix or vector the same dimension as `f` is formed, and the matrix product is computed. The resulting matrix or vector is placed into the ordinate of the result, with the same abscissa values as `f`. The data attributes of the result are taken from `f(1)`. In particular, the units exponents will be set assuming that `A` is a unitless quantity.

Note that `A*f` does not give an ordinate equal to `A*(f.ordinate)`. In the first expression, `A` must have column dimension equal to the row dimension of `f`. In the second expression, `A` must have column dimension equal to the number of abscissa values of `f`.

Examples

```
>> f=imat_fn(3,'ordinate',reshape(1:15,5,3));
>> f.responsenode=(1:3)'

f =

3x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)          Time Response    Even             5
2_(1X+,2X+)          Time Response    Even             5
3_(1X+,3X+)          Time Response    Even             5

>> f.ordinate                % f is 3x1 with 5 abscissa values

ans =
     1     6    11
     2     7    12
     3     8    13
     4     9    14
     5    10    15

>> x=[1 -1 0 ; -1 0 1]       % x is 2x3 (ncols = nrows of f)

x =
     1    -1     0
    -1     0     1

>> g=x*f; g.ordinate         % g is 2x1 with 5 abscissa values

ans =
    -5    10
    -5    10
    -5    10
    -5    10
    -5    10

>> g                         % Note all outputs have attributes of f(1)

g =
2x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)          Time Response    Even             5
2_(1X+,1X+)          Time Response    Even             5

>>
```

See Also

imat_fn/mldivide, imat_fn/mrdivide

imat_fn/rdivide

Purpose

Termwise divide ordinates (`f1./f2`).

Syntax

```
f1./f2
f./x
x./f
```

Description

The following termwise division operations are possible with `imat_fn` objects:

- Dividing two `imat_fn` objects causes their ordinates to be divided termwise. If one of the operands is a single function (1x1), then its ordinate will be used with all elements in the other operand. If both operands are function arrays, they must have the same dimensions. The data attributes of the result of a termwise division will be taken from the left operand, except that ordinate data types (i.e., units exponents) will correctly reflect the combination of the two operands.
- Dividing a numeric array or scalar and an `imat_fn` causes the numeric values to be termwise divided into or divided by the ordinate values of the `imat_fn`.

Examples

```
>> f=imat_fn(3,'ordinate',reshape(1:15,5,3))

f =
3x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)          Time Response    Even             5
2_(1X+,1X+)          Time Response    Even             5
3_(1X+,1X+)          Time Response    Even             5

>> f.ordinate

ans =
     1     6    11
     2     7    12
     3     8    13
     4     9    14
     5    10    15

>> g=f+1; g.ordinate

ans =
     2     7    12
     3     8    13
     4     9    14
     5    10    15
     6    11    16

>> h=g./f; h.ordinate

ans =
    2.0000    1.1667    1.0909
    1.5000    1.1429    1.0833
    1.3333    1.1250    1.0769
    1.2500    1.1111    1.0714
    1.2000    1.1000    1.0667

>>
```

See Also

imat_fn/mtimes, imat_fn/ldivide

---

imat_fn/ldivide

Purpose

Termwise divide ordinates (`f1.\f2`).

Syntax

```
f1.\f2
f.\x
x.\f
```

Description

The following termwise multiplication operations are possible with `imat_fn` objects:

- Dividing two `imat_fn` objects causes their ordinates to be divided termwise. If one of the operands is a single function (1x1), then its ordinate will be used with all elements in the other operand. If both operands are function arrays, they must have the same dimensions. The data attributes of the result of a termwise division will be taken from the left operand, except that ordinate data types (i.e., units exponents) will correctly reflect the combination of the two operands.
- Dividing a numeric array or scalar and an `imat_fn` causes the numeric values to be termwise divided into or divided by the ordinate values of the `imat_fn`.

A termwise left division operation in MATLAB is the same as a right division with the operands switched.

Examples

```
>> f=imat_fn(3,'ordinate',reshape(1:15,5,3))

f =
3x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)          Time Response    Even             5
2_(1X+,1X+)          Time Response    Even             5
3_(1X+,1X+)          Time Response    Even             5

>> f.ordinate

ans =
     1     6    11
     2     7    12
     3     8    13
     4     9    14
     5    10    15

>> g=f+1; g.ordinate

ans =
     2     7    12
     3     8    13
     4     9    14
     5    10    15
     6    11    16

>> h=g.\f; h.ordinate

ans =
    0.5000    0.8571    0.9167
    0.6667    0.8750    0.9231
    0.7500    0.8889    0.9286
    0.8000    0.9000    0.9333
    0.8333    0.9091    0.9375

>>
```

See Also

imat_fn/mtimes, imat_fn/rdivide

---

imat_fn/mrdivide

Purpose

Matrix divide ordinates (`f/A`).

Syntax

```
g=f/A
```

Description

Matrix division between an `imat_fn` object `f` and a numeric array `A` is performed repeatedly at all abscissa/ordinate values. This type of operation requires that all elements of `f` have the same abscissa values, and that the column dimension of `f` match the column dimension of `A`. At each abscissa value, a matrix or vector the same dimension as `f` is formed, and the matrix division is computed. The resulting matrix or vector is placed into the ordinate of the result, with the same abscissa values as `f`. The data attributes of the result are taken from `f(1)`. In particular, the units exponents will be set assuming that `A` is a unitless quantity.

Note that `f/A` does not give an ordinate equal to `(f.ordinate)/A`. In the first expression, `A` must have a column dimension equal to the column dimension of `f`. In the second expression, `A` must have a column dimension equal to the number of abscissa values of `f`.

Examples

```
>> f=imat_fn(3,'ordinate',reshape(1:15,5,3));
>> f.responsenode=(1:3)'

f =
3x1 IMAT Function with the following attributes:
Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)          Time Response    Even             5
2_(1X+,2X+)          Time Response    Even             5
3_(1X+,3X+)          Time Response    Even             5

>> f.ordinate                % f is 3x1 with 5 abscissa values

ans =
     1     6    11
     2     7    12
     3     8    13
     4     9    14
     5    10    15

>> A=(1:3)'                  % A is 3x1 (ncols = ncols of f)

A =
     1
     2
     3

>> g=f/A; g.ordinate         % g is 3x3 with 5 ordinate values

ans(:,:,1) =
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0

ans(:,:,2) =
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0

ans(:,:,3) =
    0.3333    2.0000    3.6667
    0.6667    2.3333    4.0000
    1.0000    2.6667    4.3333
    1.3333    3.0000    4.6667
    1.6667    3.3333    5.0000

>> g                         % Note all outputs have attributes of f(1)

g =
3x3 IMAT Function with the following attributes:
Row Col Record Name          FunctionType     AbscissaSpacing  NumberElements
--- --- -------------------- ---------------- ---------------- ----------------
1   1   1_(1X+,1X+)          Time Response    Even             5
2   1   2_(1X+,1X+)          Time Response    Even             5
3   1   3_(1X+,1X+)          Time Response    Even             5
1   2   4_(1X+,1X+)          Time Response    Even             5
2   2   5_(1X+,1X+)          Time Response    Even             5
3   2   6_(1X+,1X+)          Time Response    Even             5
1   3   7_(1X+,1X+)          Time Response    Even             5
2   3   8_(1X+,1X+)          Time Response    Even             5
3   3   9_(1X+,1X+)          Time Response    Even             5

>>
```

See Also

imat_fn/mldivide, imat_fn/mtimes

---

imat_fn/mpower

Purpose

Matrix power.

Syntax

```
g=mpower(f,num)
g=mpower(f,2)
g=f^2
```

Description

MPOWER takes the `imat_fn` to the power of NUM. NUM must be a scalar.

See Also

imat_fn/power

---

imat_fn/mpower

Purpose

Matrix power.

Syntax

```
g=mpower(f,num)
g=mpower(f,2)
g=f^2
```

Description

MPOWER takes the `imat_fn` to the power of NUM. NUM must be a scalar.

See Also

imat_fn/power

---

imat_fn/power

Purpose

Array power.

Syntax

```
g=power(f,num)
g=power(f,2)
g=f.^2
g=2.^f
```

Description

IMAT_FN.^NUM takes the `imat_fn` to the power of NUM. NUM must be a scalar.

NUM.^IMAT_FN takes NUM to the power of the ordinate values in the `imat_fn`. The output is an IMAT_FN the same size as the input, where all of the ordinate values are NUM to the power of the ordinate values of the input IMAT_FN.

See Also

imat_fn/mpower

---

imat_fn/sqrt

Purpose

Square root.

Syntax

```
g=sqrt(f)
```

Description

SQRT takes the square root of the `imat_fn`M. NUM must be a scalar. Complex results are produced if F is not positive.

---

imat_fn/real

Purpose

Take real part of ordinate.

Syntax

```
g=real(f)
```

Description

This function replaces all ordinate values of `f` with their real parts.

See Also

imat_fn/imag, imat_fn/conj

---

imat_fn/imag

Purpose

Take imaginary part of ordinate.

Syntax

```
g=imag(f)
```

Description

This function replaces all ordinate values of `f` with their imaginary parts.

See Also

imat_fn/real, imat_fn/conj

---

imat_fn/conj

Purpose

Take complex conjugate of ordinate.

Syntax

```
g=conj(f)
```

Description

This function replaces all ordinate values of `f` with their complex conjugate values.

See Also

imat_fn/real, imat_fn/imag

---

imat_fn/abs

Purpose

Take absolute value (or modulus) of ordinate.

Syntax

```
g=abs(f)
```

Description

This function replaces all ordinate values of `f` with their absolute value (if `f` is real-valued) or modulus (if `f` is complex-valued). If `f` is complex-valued, its OrdinateType attribute will be tagged as real.

See Also

imat_fn/real, imat_fn/imag, imat_fn/phase, imat_fn/phased

---

imat_fn/diag

Purpose

Diagonal IMAT_FN matrices and diagonal of a matrix.

Syntax

```
g=diag(f)
g=diag(f,k)
```

Description

DIAG implements MATLAB's DIAG function for IMAT_FN.

diag(F,K) when F is an IMAT_FN vector with N components is a square matrix of order N+ABS(K) with the elements of V on the K-th diagonal. K = 0 is the main diagonal, K > 0 is above the main diagonal and K < 0 is below the main diagonal.

diag(V) is the same as diag(V,0) and puts V on the main diagonal.

diag(X,K) when X is an IMAT_FN matrix is an IMAT_FN column vector formed from the elements of the K-th diagonal of X.

diag(X) is the main diagonal of X. diag(diag(X)) is a diagonal matrix.

---

imat_fn/phase

Purpose

Replace ordinate with its phase angle in radians.

Syntax

```
g=phase(f)
```

Description

This function replaces all ordinate values of `f` with the complex phase angle in radians. The phase angle will be in the range from -2*pi to 0.

See Also

imat_fn/abs, imat_fn/phased

---

imat_fn/phased

Purpose

Replace ordinate with its phase angle in degrees.

Syntax

```
g=phased(f)
```

Description

This function replaces all ordinate values of `f` with the complex phase angle in degrees. The phase angle will be in the range from -360 to 0.

See Also

imat_fn/abs, imat_fn/phase

---

imat_fn/cbred

Purpose

Compute constant band reduction of the supplied `imat_fn`.

Syntax

```
g=cbred(f)
g=cbred(f,1.1)
g=cbred(f,1.1,'silent','min')
g=cbred(f,1.1,'silent','min','nearest')
```

Description

CBRED computes a constant band reduction of the supplied `imat_fn`. If all of the input functions are the same size and abscissa increment, CBRED works much more quickly. You may pass in an optional scalar specifying the size (in abscissa units) for each constant band block. If the block size is not supplied, CBRED will attempt to determine it by using the maximum amplitude of an FFT of the function. Note that this only works for a time history `imat_fn`. Passing in an optional string `'nearest'` tells CBRED to use the closest integer multiple of dT in the reduction. This allows CBRED to use the faster reduction method.

Passing in a string containing `'max'`, `'min'`, or `'mean'` specifies the reduction method to use. The default reduction method is `'max'`. Passing in the string `'silent'` will suppress output during the calculations.

Examples

```
>> f=imat_fn(1);
>> f.abscissamin=0;
>> f.abscissainc=1/100;
>> f.ordinate=sin(2*pi*(0:999)/10);

>> fr=cbred(f)
Automatically computing constant band using fft
Computed a band frequency of 10.000000 Hz (dT=0.100000 sec)
Computing MAX reduction
Samples per band = 10

fr =

IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Time Response    Even             100

>> fr=cbred(f,1.2,'nearest','mean')
Computing MEAN reduction
Samples per band = 120

fr =

IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Time Response    Even             9
```

See Also

imat_fn/octaven

---

imat_fn/fft

Purpose

Compute PSDs from the supplied time history `imat_fn`.

Syntax

```
g=fft(f)
g=psd(f,AMPUNITS)
```

Description

FFT converts between time and frequency-domain functions. It automatically determines the FFT conversion direction based on the function type. Time-domain functions are converted to frequency domain, and vice versa.

F is an `imat_fn` of time and/or frequency domain functions. AMPUNITS is an optional string to override the amplitude units specified in the function.

Valid values for AMPUNITS are

`'half-peak'` - Default(also known as 0-peak amplitude)
`'peak'`
`'RMS'`

When converting from time to frequency domain, only the real part of the abscissa is retained, and only the "positive frequencies" of the result are returned. Thus, the FFT of a 1024-point time history will be a 513-point spectrum. By default, the spectrum will be normalized to half-peak, so that a sine wave of unit amplitude will have a spectral amplitude of 0.5 (this is consistent with I-DEAS Test). You may override this with the optional AMPUNITS argument.

When converting from frequency to time domain, a real function is returned. The AmplitudeUnits attribute of the spectrum will determine the interpretation ('Unknown' will be treated as 'Half-peak').

Examples

```
>> f=imat_fn(1);
>> f.abscissamin=0;
>> f.abscissainc=1/100;
>> f.ordinate=sin(2*pi*(0:999)/10);
>> setdisplay(imat_fn,'functiontype','amplitudeunits','numberelements');
>> g

g =

IMAT Function with the following attributes:

Record Name                 FunctionType     AmplitudeUnits   NumberElements
--------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                 Spectrum         Half-peak scale  501

>> max(abs(g.ordinate))

ans =
     0.5000

>> h=fft(g)

h =
IMAT Function with the following attributes:
Record Name                 FunctionType     AmplitudeUnits   NumberElements
--------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                 Time Response    Half-peak scale  1001

>> max(abs(h.ordinate))

ans =
     1.0010

>>
```

See Also

imat_fn/psd

---

imat_fn/csd (+Signal)

Purpose

Compute CSDs from the supplied time history `imat_fn`

Syntax

```
gyx=csd(tref,tres,5)
gyx=csd(tref,tres,5,10)
gyx=csd(TREF,TRES,NBLK,NAVG,'silent','noprogbar')
gyx=csd(TREF,TRES,NBLK,NAVG,'overlap',OVPCT)
```

Description

CSD will compute single-sided Cross Power Spectral Density functions from the supplied time histories. TREF is an `imat_fn` containing the reference time histories. TRES is an `imat_fn` containing the response time histories. All of the supplied time histories must have the same length. NBLK is a required parameter indicating the window size. If it is a scalar it indicates the block size to be used in the CSD calculation. A Hamming window will be used. If it is a vector or `imat_fn`, it should contain the window to be used. The length of the vector indicates the window size.

NAVG is an optional scalar specifying the number of averages to use. If you do not specify the number of averages or the overlap, the default for NAVG is the number of frames that can be used given the specified NBLK without overlapping, or 5, whichever is greater. If NAVG is specified, and OVPCT is not, CSD will automatically use overlap processing to achieve the requested number of averages.

`'overlap'` is an optional input that must be followed by OVPCT to specify the amount of overlap desired. OVPCT is a numeric scalar between 0 and less than 100 specifying the overlap percentage. If you specify OVPCT, but do not specify NAVG, CSD will calcalate NAVG to use as much of the time history as possible with the specified overlap.

`'silent'` is an optional input string that suppresses output during the CSD calculation.

`'noprogbar'` is an optional input string that disables the progress bar.

CSD returns an `imat_fn` of the same size as T containing the CSDs.

GYX is an `imat_fn` of size TRESxTREF containing the CSDs. The subscript Y represents the output (responses), and X represents the input (references).

Examples

```
>> t
t =
1x2 IMAT Function with the following attributes:
Row Col Record Name         FunctionType     AbscissaSpacing  NumberElements
--- --- ------------------- ---------------- ---------------- ----------------
1   1   1_(,1X+)            Time Response    Even             159681
1   2   2_(,2X+)            Time Response    Even             159681

>> c=csd(t,t,8192)
...Processing time histories
-------------------------
Number of points:   155648
      Block Size:   4096
        Averages:     38
         Overlap:      0 samples (0.00%)
         Delta F:  1.0000 Hz
  Spectral lines:   2049
-------------------------
CSD Processing Parameters

c =

2x2 IMAT Function with the following attributes:

Row Col Record Name         FunctionType     AbscissaSpacing  NumberElements
--- --- ------------------- ---------------- ---------------- ----------------
1   1   1_(1X+,1X+)         Auto Spectrum    Even             2049
2   1   2_(1X+,2X+)         Cross Spectrum   Even             2049
1   2   3_(2X+,1X+)         Cross Spectrum   Even             2049
2   2   4_(2X+,2X+)         Auto Spectrum    Even             2049

>> c=csd(t,t,window(t,8192,'flattop'),50)
CSD Processing Parameters
-------------------------
Number of points:   155648
      Block Size:   8192
        Averages:     50
         Overlap:   5100 samples (62.26%)
         Delta F:  0.5000 Hz
  Spectral lines:   4097
-------------------------
...Processing time histories

c =

2x2 IMAT Function with the following attributes:

Row Col Record Name         FunctionType     AbscissaSpacing  NumberElements
--- --- ------------------- ---------------- ---------------- ----------------
1   1   1_(1X+,1X+)         Auto Spectrum    Even             4097
2   1   2_(1X+,2X+)         Cross Spectrum   Even             4097
1   2   3_(2X+,1X+)         Cross Spectrum   Even             4097
```

```
2   2   4_(2X+,2X+)         Auto Spectrum    Even             4097

>>
```

See Also

imat_fn/fft, imat_fn/psd, imat_fn/window

---

imat_fn/psd (+Signal)

Purpose

Compute PSDs from the supplied time history `imat_fn`

Syntax

```
g=psd(f,5)
g=psd(f,5,10)
g=psd(f,5,[],'peakhold')
g=psd(T,NBLK,NAVG,'silent','noprogbar')
g=psd(T,NBLK,NAVG,'overlap',OVPCT)
g=psd(T,NBLK,'maximax',NOCT)
```

Description

PSD computes single-sided Power Spectral Density functions from the supplied time histories in the `imat_fn` T. All of the supplied time histories must have the same length. NBLK is a required parameter indicating the window size. If it is a scalar it indicates the block size to be used in the PSD calculation. A Hamming window will be used. If it is a vector or `imat_fn`, it should contain the window to be used. The length of the vector indicates the window size.

NAVG is an optional scalar specifying the number of averages to use. If you do not specify the number of averages or the overlap, the default for NAVG is the number of frames that can be used given the specified NBLK without overlapping, or 5, whichever is greater. If NAVG is specified, and OVPCT is not, PSD will automatically use overlap processing to achieve the requested number of averages.

`'overlap'` is an optional input that must be followed by OVPCT to specify the amount of overlap desired. OVPCT is a numeric scalar between 0 and less than 100 specifying the overlap percentage. If you specify OVPCT, but do not specify NAVG, PSD will calcalate NAVG to use as much of the time history as possible with the specified overlap.

`'peakhold'` is an optional input that keeps the maximum values for each spectral line rather than averaging across all of the frames.

`'maximax'`is similar to `'peakhold'` except that it does an Nth octave reduction before applying the peak-hold. It must be followed by a numeric scalar NOCT specifying the Nth octave reduction to be used. Passing in an empty value specifies the default, which is 6.

`'silent'` is an optional input string that suppresses output during the PSD calculation.

`'noprogbar'` is an optional input string that disables the progress bar.

PSD returns an `imat_fn` of the same size as T containing the PSDs.

Examples

```
>> g=psd(f,128)
PSD Processing Parameters
-------------------------
Number of points:    640
      Block Size:    128
        Averages:      5
         Overlap:      0 samples (0.00%)
         Delta F:  0.7813 Hz
  Spectral lines:     65
-------------------------
...Processing time histories

g =

3x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Power Spectral D Even             65
2_(1X+,1X+)                  Power Spectral D Even             65
3_(1X+,1X+)                  Power Spectral D Even             65


>> g=psd(f,window(f,'hanning',128),50)
PSD Processing Parameters
-------------------------
Number of points:   1000
Block Size:    128
Averages:     50
Overlap:    110 samples (85.94%)
Delta F:  0.7813 Hz
Spectral lines:     65
-------------------------
...Processing time histories

g =

3x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Power Spectral D Even             65
2_(1X+,1X+)                  Power Spectral D Even             65
3_(1X+,1X+)                  Power Spectral D Even             65

>>
```

See Also

imat_fn/fft, imat_fn/csd, imat_fn/window

---

imat_fn/psd2trans (+Signal)

Purpose

Generate transient from Power Spectral Density.

Syntax

```
outtrans=psd2trans(psdfun,deltat,duration)
[outtrans,stats]=psd2trans(psdfun,deltat,duration,'silent',interptype)
```

Description

PSD2TRANS generates a transient using a summation of sine waves with random phase. PSDFUN is an `imat_fn` containing one or more power spectral density functions. DELTAT is the time increment to be used in the equivalent transient function. DURATION is the total length of the equivalent transient function. To use the default value for DELTA or DURATION, enter [].

The default value for DELTAT is 10 points in time at the highest frequency. The default value for DURATION is 10 cycles at the lowest frequency.

If the string `'silent'` is passed in, all text output is suppressed including warnings. INTERPTYPE is an optional string argument defining the interpolation method used for the PSD. Accepted values for INTERPTYPE are: `'linlin'` or (`'lin'`), `'linlog'`, `'loglin'`, or `'loglog'`. `'loglog'` is the default.

OUTTRANS is an `imat_fn` containing the transient(s). STATS is an 8xnumFunc array containing some useful statistics of the function(s). The rows contain the following info:

row 1: mean value of equivalent transient
row 2: RMS value of input PSD
row 3: RMS value of equivalent transient
row 4: number of peaks between 1 and 2 standard deviations
row 5: number of peaks between 2 and 3 standard deviations
row 6: number of peaks between 3 and 4 standard deviations
row 7: number of peaks above 4 standard deviations
row 8: ratio of peak value to RMS value for equivalent transient

See Also

imat_fn/psd

---

imat_fn/frf (+Signal)

Purpose

Compute FRFs from the supplied CSD/PSD functions.

Syntax

```
f=frf(gyx,gyy)
[f,c,co]=frf(gyx,gyy,'h1','silent','useinv')
```

Description

FRF computes Frequency Response Functions from the supplied CSD/PSD functions. GYX is a spectral matrix of CSDs and PSDs of size NCHANxNREF where NCHAN is the number of channelsand NREF is the number of references. Each column corresponds to a different reference. The first NREF rows must contain the reference CSDs and PSDs for the references, in other words it must be Gxx. GYY is a NCHANx1 vector of PSDs of all of the reference channels in the same order as they appear in GYX. In the subscript nomenclature, X represents the inputs (references) and Y represents the responses. This function expects that Y actually contains all of the channels, references first, followed by responses.

METHOD is an optional string specifying the FRF calculation method to use. Available methods are

`'H1'` - Gyx / Gxx (default)
`'H2'` - Gyy / Gxy (single reference only)
`'H3'` - (H1 + H2) / 2 (single reference only)
`'Hv'` - Optimal scaling, used to minimize the effects of noise

In the above equations, *x* is the reference and *y* is the response.

The optional string input `'silent'` suppresses any output.

If two output arguments are requested, FRF will also calculate the coherence. For single-reference input, this is the ordinary coherence, which is defined as

```
C = H1 ./ H2
```

For multiple-reference input, this corresponds to the multiple coherence, which is defined as (taken one response at a time)

```
C = [Gyx] [Gxx]^-1 [Gyx] / Gyy
```

F is an `imat_fn` of size NCHANxNREF containing the FRF. C is an optional output of size NCHANx1 containing the coherence (ordinary for single reference, multiple for multi-reference). CO is an optional output of size NCHANxNREF containing the ordinary coherence (for multi-reference).

The example below shows how to generate FRF from a series of time histories stored in the `imat_fn` T. The first 2 time histories are the reference channels.

Examples

```
>> gyx = csd(t(1:2),t,8192);

>> gyy = psd(t,8192);

>> [f,c] = frf(gyx,gyy);
```

See Also

imat_fn/csd, imat_fn/psd

---

imat_fn/spl (+Signal)

Purpose

Convert spectra to sound pressure level.

Syntax

```
g=spl(f)
[g,oaspl]=spl(f)
```

Description

SPL takes an imat_fn F of PSD, Auto Spectrum, or Spectrum functions and and converts it to "SPL" functions. If the optional output argument OASPL is included, an array of the overall sound pressure level values for the corresponding functions in G is also returned. The overall SPL is also stored in the UserValue1 field of the functions in G.

The abscissa data type must be frequency. The ordinate numerator data type must be either pressure or sound pressure. The normalization and ordinate numerator type qualifier attributes must be consistent. That is, if the normalization is one of the units squared values, then the type qualifier must be translation squared. Units squared sec/Hz normalization is not supported. Non-octave uneven abscissa spacing with units squared/Hz normalization is not supported.

The "SPL" functions returned in G inherit the attributes from the spectra functions, except as listed below.

```
'FunctionType'      : 'Auto Spectrum'
'OrdNumDataType'    : 'Unknown'
'OrdinateAxisLab'   : 'SPL'
'OrdinateUnitsLab'  : 'dB re 20microPA'
'AmplitudeUnits'    : 'Unknown'
'Normalization'     : 'Unknown'
```

The dB reference is 20 microPa or the equivalent in the current units system. User defined units system is not supported.

See Also

imat_fn/acoustic_weighting

---

imat_fn/acoustic_weighting

Purpose

Apply acoustic weighting.

Syntax

```
g=acoustic_weighting(f,'A')
g=acoustic_weighting(f,'C')
```

Description

ACOUSTIC_WEIGHTING applies the specified acoustic weighting to the PSD, Auto Spectrum, Cross Spectrum, or Spectrum functions in the imat_fn F. WEIGHT can be either `'A'` or `'C'` weighting.

The abscissa data type must be frequency. The ordinate numerator data type must be either pressure or sound pressure. The normalization and ordinate numerator type qualifier attributes must be consistent. That is, if the normalization is one of the units squared values, then the type qualifier must be translation squared. The weighting type must be none, otherwise another weighting will not be applied.

ACOUSTIC_WEIGHTING will also add an acoustic weighting in dB to the "SPL" functions created by SPL.

See Also

imat_fn/spl

---

imat_fn/window

Purpose

Create a window function.

Syntax

```
g=window(imat_fn,'hanning',8192)
g=window(imat_fn,'exponential',1024,0.2)
g=window(F,TYPE,L,ARGS)
```

Description

WINDOW creates a window function based on the type of window specified in the string TYPE. F must be an `imat_fn`. It can be an empty `imat_fn`. If F is not empty, WINDOW will extract the AbscissaMin and AbscissaInc attributes from the first function in F, provided that it is evenly spaced. L is the length of the window in samples.

Some of the windows have additional optional or required arguments. These are described in more detail below.

Several window types are supported. Some of these are built-in, and some require the Signal Processing Toolbox. The built-in windows are described here. For a list of the windows provided by the Signal Processing toolbox, see the help for the function WINDOW provided by that toolbox.

TYPE	DESCRIPTION
`'hanning'`	Hann window, $\sin(\pi*t/T)^2$
`'flattop'`	Flattop window. Almost identical to `'flattopwin'` from the Signal Processing toolbox. Equation fromhttp://zone.ni.com/reference/en-XX/help/371361A-01/lvanlsconcepts/char_smoothing_windows/
`'exponential'`	Exponential decay window. One additional required input argument, DECAY. This specifies the amplitude fraction at the 50% point of the window. For example, to set the window amplitude to be 30% at the halfway point, set DECAY=0.3.
`'impact'`	Impact window. It is set to 1 at the beginning of the window and zero everywhere past the fraction of the window specified by the required input argument FRAC. FRAC should be between 0.0 and 1.0.

Examples

```
>> w=window(imat_fn,'hanning',8192)

w =

1x1 IMAT Function with the following attributes:

Record Name          FunctionType     AbscissaSpacing  NumberElements
-------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)          Time Response    Even 8192

>> w.idline1

ans =

'Window: 'hanning', 8192 points'

>>
```

See Also

imat_fn/csd, imat_fn/psd

---

imat_fn/filterf

Purpose

FIR Filter the supplied `imat_fn` time history.

Syntax

```
g=filterf(f,'low',5)
g=filterf(f,'band',[5.5 6.8])
g=filterf(IN,TYPE,WP,WS,'silent','nopad','noprogbar')
```

Description

FILTERF applies a Finite Impulse Response (FIR) filter to the supplied `imat_fn` IN. It uses a Kaiser window. TYPE is a string specifying the type of filter. Supported filter types are

`'high'` - High pass
`'low'` - Low pass
`'band'` - Band pass
`'stop'` - Notch

WP is a scalar (for high- and low-pass) or a 1x2 vector (for bandpass and notch) filters. It specifies the filter frequencies. WS is an optional scalar or vector that specifies the edges of the filter. For example, for a low-pass filter, you might set WP to 5 Hz and WS to 5.5 Hz. The range between 5 and 5.5 Hz is called the slopeband. If WS is not supplied, it is set to 2% around WP. The allowable ripple in the passband and stopband is 1%. FILTERF stores the requested filter and parameters in IDLine2 of the output.

The string `'silent'` suppresses output. If it is not supplied (the default), status messages will be displayed. A plot of the filter desired and actual characteristics will be displayed.

The string `'noprogbar'` suppresses the progress bar.

By default FILTERF will extend the time history if necessary so that the desired filter order can be used. The filter requires about 3 times as many time history points as the filter order. It does this by flipping the supplied time history along the X axis and taking the inverse both forward and backward until it has enough time samples. The optional string `'nopad'` bypasses this, reducing the filter order to work with the supplied time history.

FILTERF requires the Signal Processing toolbox.

Examples

```
>> g=filterf(f,'band',[5.5 6.8])
-------------------------------------
Nyquist frequency: 50.000000 Hz
Filter requested bandpass frequency:  5.500000-6.800000 Hz
Filter requested stopband frequency:  5.390000-6.936000 Hz
Using a Kaiser filter of order 2030
Filter maximum amplitude is 1.009982
Filter actual passband frequency:  5.504926-6.810345 Hz
Filter actual stopband frequency:  5.406404-6.908867 Hz
Padding time history 1 by 5132 samples to achieve sufficient length for the filter
Padding time history 2 by 5132 samples to achieve sufficient length for the filter
Padding time history 3 by 5132 samples to achieve sufficient length for the filter

g =

3x1 IMAT Function with the following attributes:

Record Name                   FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                   Time Response    Even             1000
2_(1X+,1X+)                   Time Response    Even             1000
3_(1X+,1X+)                   Time Response    Even             1000

>> g=filterf(f,'band',[5.5 6.8],'silent','nopad')

g =

3x1 IMAT Function with the following attributes:

Record Name                   FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                   Time Response    Even             1000
2_(1X+,1X+)                   Time Response    Even             1000
3_(1X+,1X+)                   Time Response    Even             1000

>>
```

See Also

imat_fn/filteri

---

imat_fn/filteri

Purpose

IIR Filter the supplied `imat_fn` time history.

Syntax

```
g=filteri(f,'low',5)
g=filteri(f,'band',[5.5 6.8])
g=filteri(IN,TYPE,WP,NPOLES,'silent','noprogbar')
```

Description

FILTERI applies an Infinite Impulse Response (IIR) filter to the supplied `imat_fn` IN. It uses a Butterworth filter. TYPE is a string specifying the type of filter. Supported filter types are

`'high'`	- High pass
`'low'`	- Low pass
`'band'`	- Band pass
`'stop'`	- Notch

WP is a scalar (for high- and low-pass) or a 1x2 vector (for bandpass and notch) filters. It specifies the filter frequencies. NPOLES is an optional scalar specifying the number of poles to use in the filter. The default is 4. FILTERI stores the requested filter and parameters in IDLine2 of the output.

The string `'silent'` suppresses output. If it is not supplied (the default), status messages will be displayed. A plot of the filter characteristics will be displayed.

The string `'noprogbar'` suppresses the progress bar.

FILTERI requires the Signal Processing toolbox.

Examples

```
>> g=filteri(f,'band',[5.5 6.8])
-------------------------------------
Nyquist frequency: 50 Hz
Using a 4-pole Butterworth filter
Filter bandpass frequency:  5.5-6.8 Hz
Filter maximum amplitude is 1


g =

3x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Time Response    Even             1000
2_(1X+,1X+)                  Time Response    Even             1000
3_(1X+,1X+)                  Time Response    Even             1000

>>
```

See Also

imat_fn/filterf

---

imat_fn/findpeaks

Purpose

Locate peaks in an `imat_fn` using a tolerance

Syntax

```
mvals=findpeaks(f)
[mvals,ind]=filterf(f,[],1e-4)
[mvals,ind,inds,inde=filterf(f,tol,thresh,type,debug)
```

Description

FINDPEAKS will find peaks in a single `imat_fn` F, and return the peak values as a double vector MVALS, with optional index vector IND containing indices into the ordinate values corresponding to the maximum values. INDS and INDE are optional vectors that give indices to the start and end of the peaks, as defined by the change in slope to either side of the peak. If F is complex, FINDPEAKS will take the absolute value to find the peaks.

FINDPEAKS allows several optional input arguments. TOL is the tolerance to use for finding peaks. The default tolerance is `0.1`. THRESH is a threshhold value below which to ignore. An empty value (the default) causes FINDPEAKS to consider all values. TYPE is a flag that determines how the tolerance is calculated. Valid arguments are `'max'` and `'rel'`. The default is `'max'`. The paragraph below outlines what each does. Sending in an empty matrix for any of these optional arguments causes the default to be used. DEBUG is an optional logical (default false) that will display a plot of candidate and selected peaks on top of the supplied function if set to true.

The general algorithm of FINDPEAKS is as follows. FINDPEAKS first determines the sign of the slope of the ordinate. Peaks occur where the left side of a given point has positive slope, and the right side of a given point has negative slope. Valleys occur at the opposite slopes. In areas of flat slope, FINDPEAKS assumes that the slope of the ordinate to the left of the center of the flat area has the same slope as the left edge of the flat area, and the slope of the ordinate to the right of the center of the flat area has the same slope as the right edge of the flat area.

If the change in amplitude between a local maximum or local minimum and the previous maximum or previous minimum, is less than the tolerance times the function maximum, the change is ignored. This is how it works when TYPE is `'max'`. When TYPE is `'rel'`, the tolerance is multiplied by the function local extrema. In other words, if the change in amplitude between two local extrema is less than the tolerance times the 2nd extremum, the change is ignored.

---

imat_fn/diff

Purpose

Differentiate the supplied `imat_fn`.

Syntax

```
g=diff(f)
g=diff(f,'freq')
```

Description

DIFF will differentiate the supplied `imat_fn` F, and return the output in G. Two types of differentiation are supported. The default type is the simple dy/dx differentiation. If TYPE is set to `'freq'`, then DIFF will perform a frequency differentiation using the i*omega operator. Note that frequency differentiation can only operate on functions with an AbscissaDataType of Frequency. Any other type will result in an error.

DIFF will automatically modify the ordinate numerator data type from Displacement to Velocity to Acceleration to General. It also handles acceleration units in G's and EU's.

See Also

imat_fn/integ

---

imat_fn/integ

Purpose

Integrate the supplied `imat_fn`.

Syntax

```
g=integ(f)
g=integ(f,'freq')
```

Description

INTEG will differentiate the supplied `imat_fn` F, and return the output in G. Two types of integration are supported. The default type is the simple "area under the curve". INTEG uses the trapezoidal rule to compute the integration, using the CUMTRAPZ function. If TYPE is set to `'freq'`, INTEG will perform a frequency integration using the 1/i*omega operator. Note that frequency integration can only operate on functions with an AbscissaDataType of Frequency. Any other type will result in an error.

INTEG will automatically modify the ordinate numerator data type from Acceleration to Velocity to Displacement to General. It also handles acceleration units in G's and EU's.

See Also

imat_fn/diff

---

imat_fn/decimate

Purpose

Decimate the supplied `imat_fn`.

Syntax

```
g=decimate(f,100)
g=decimate(f,FACTOR)
g=decimate(f,FACTOR,METHOD,ARGS)
```

Description

DECIMATE will reduce the number of points in the function F by a factor of FACTOR. F must have evenly-spaced Abscissa values. FACTOR must be a positive integer. If F is a time-domain function, then DECIMATE from the Signal Processing Toolbox will be used. For other types of functions, or if the Signal Processing toolbox is not available, a simpler averaging method will be used instead. Please note that the Signal Processing Toolbox DECIMATE function by default uses a Chebyshev Type 1 low-pass filter which could alter your results slightly. In some cases a small DC offset has been observed. Please check your decimated function against the original, and if unacceptable differences are observed, direct it to use a different filter type.

To force a specific method, set the METHOD parameter to `'mean'` or `'spt'`. `'mean'` will do an average of the data in each segment, and `'spt'` will attempt to use the Signal Processing Toolbox decimate function if it's available. Please note that the 'mean' option does not apply a low-pass filter to the data, which means that your data may end up aliased. Allowing DECIMATE to use the Signal Processing DECIMATE function, or low-pass filtering your data prior to calling DECIMATE, is much more robust.

ARGS are additional arguments that you can pass to the Signal Processing DECIMATE function (for example to change the filter order and/or type). Please see the help for DECIMATE for a description of these additional arguments.

Examples

```
>> g=decimate(f,2);
>> g=decimate(f,4,'mean');
>> g=decimate(f,4,'spt');
>> g=decimate(f,4,10);      % 10 is passed to Signal Processing DECIMATE
>>
```

See Also

imat_fn/interp

---

imat_fn/interp

Purpose

Interpolate the supplied `imat_fn`.

Syntax

```
g=interp(f,100)
g=interp(f,NPOINTS,SCALES,METHOD)
g=interp(f,'inc',INCREMENT)
g=interp(f,'inc',INCREMENT,SCALES,METHOD,EXTRAPTYPE)
```

Description

INTERP will generate linear- or log-spaced abscissa values for the provided function, then pair each point with an interpolated ordinate value. INTERP uses MATLAB's INTERP1 function to perform the interpolation. If the function is complex, INTERP will interpolate the magnitude according to the specified SCALES, and will interpolate the phase linearly.

NPOINTS is a number specifying how many abscissa values to generate, or a vector of abscissa values at which to interpolate. If the string `'values'` is supplied,then NPOINTS is always interpreted as abscissa values. This is necessary if NPOINTS is a scalar abscissa value. If the string `'inc'` is supplied, then INCREMENT is a number specifying the spacing between consecutive abscissa values.

SCALES is an optional string specifying whether the abscissa/ordinate values should use linear or logarithmic scaling for the abscissa and ordinate, respectively. Possible values are `'lin'` (default), `'linlog'`, `'loglin'`, and `'loglog'`.

METHOD is an optional string, passed to INTERP1 to control the method of interpolation. Values accepted by INTERP1 include `'nearest'`, `'linear'`, `'spline'`, `'pchip'`, `'cubic'`, and `'v5cubic'`. The default is `'linear'`. An empty string specifies the default method.

EXTRAPTYPE is an optional numeric scalar, passed to INTERP1 to control the value to be used for extrapolated data. If specifying EXTRAPTYPE, you must also specify METHOD.

INTERP will automatically set the abscissa spacing of the output function to `'Even'` if the difference between the abscissa data points is within `1000*eps`. If this tolerance is too tight, you can always force the spacing to be even by changing the AbscissaSpacing of your output function.

Examples

```
>> g=interp(f,'inc',0.05,'lin');
>> g=interp(f,100,'linlog');
>> g=interp(f,100,'loglog','cubic');
>> g=interp(f,100,'linlog','cubic',0);
>>
```

See Also

imat_fn/decimate

---

imat_fn/envelope

Purpose

Create an envelope function of the supplied `imat_fn`.

Syntax

```
g=envelope(f)
g=envelope(f,'min')
g=envelope(F,TYPE,INTTYPE,EXTRAP)
g=envelope(f,'mean','spline',false)
```

Description

ENVELOPE will create an envelope of the supplied `imat_fn`. F is the supplied input `imat_fn`. TYPE is an optional string specifying the type of envelope. Complex data is converted to amplitude before taking the envelope. Passing in an empty string ('') for TYPE tells ENVELOPE to use the default. Valid TYPEs are

TYPE	**Description**
`'min'`	Envelope the minima
`'max'`	Envelope the maximum values [default]
`'mean'`	Envelope the mean values

INTTYPE is a string containing the interpolation type. Valid types are any supported by the MATLAB function INTERP1. The default is `'linear'`. Interpolation (and extrapolation) only comes into play if not all of the supplied functions have the same abscissa values.

EXTRAP is a logical specifying whether to allow extrapolation if the function is interpolated. The default is TRUE. However, note that the extrapolation can be dangerous since it generates data values that don't exist in the data. If you get strange results, try turning off extrapolation.

ENVELOPE takes into consideration the abscissa values for each of the functions. At a given abscissa data point, envelope will only consider functions that have data at point. ENVELOPE will create one function per abscissa data type.

G is an `imat_fn` containing the output. There will be one function per data type in the input. Each output function takes on the attributes of the first function in F with that abscissa data type.

Examples

```
>> f=imat_fn(2);
>> f(1).ordinate=5:-1:1;
>> f(2).ordinate=1:5;
>> g=envelope(f,'max','linear');
>> [f.ordinate g.ordinate]

ans =

    5     1     5
    4     2     4
    3     3     3
    2     4     4
    1     5     5

>>
```

imat_fn/movavg

Purpose

Moving average of the supplied `imat_fn`.

Syntax

```
g=movavg(f,10)
g=movavg(f,10,'silent')
g=movavg(F,NPTS)
```

Description

MOVAVG performs a moving average on the supplied `imat_fn`. NPTS is an optional input argument specifying how many points to use in the moving average. If it is not supplied, it uses 1/100 of the number of points of the first function in F.

The optional input string `'silent'` suppresses any output as well as the progress bar.

---

imat_fn/octaven

Purpose

Reduce PSD and FRF `imat_fn` to 1/N octaves.

Syntax

```
g=octaven(f)
g=octaven(f,3)
g=octaven(f,3,'ansi')
g=octaven(f,1,'silent')
```

Description

OCTAVEN takes a function F with "narrow band" PSD, Spectrum, Auto Spectrum, or FRF data, and reduces it to "1/N octave" PSD or FRF data. If N is not specified, it will default to 1. The abscissa must be evenly spaced.

The optional input argument `'silent'` suppresses any output.

By default OCTAVEN does not use the "preferred" frequencies. Instead, the following formulas are used:

```
center = 1000*2^(i/N)
      where i = 0 for f0 = 1000 Hz

      upper = center * 2^(1/2/N)
      lower = center / 2^(1/2/N)
```

To use the ANSI/ISO standard frequencies defined in ISO R 266 and ANSI S1.6-1984, pass in the string `'ansi'`. If octave or third octave reduction is requested, the standard frequencies will be used. Otherwise OCTAVEN will issue a warning and use the above formulas instead.

The output function G will contain the 1/N octave functions. Values are returned for every 1/N octave in which some narrow band data was found. For the first band, the value at the beginning of the band is assumed to be equal to the 1st value in the band. For the last band, the value at the end of the band is assumed to be equal to the last value in the band.

By default the reduced value is the average across each 1/N octave. If the optional input string `'rms'` is supplied, OCTAVEN will return the result as the RMS of the reduced input. PSDs will be converted to linear spectra.

See Also

imat_fn/cbred, imat_fn/octaven2nb, get_octave_bands

---

imat_fn/octaven2nb

Purpose

Convert octave-reduced functions to narrowband.

Syntax

```
g=octaven2nb(f)
g=octaven2nb(f,20)
g=octaven2nb(f,'silent')
```

Description

OCTAVEN2NB takes a function F with octave-reduced "narrow band" PSD, Spectrum, Auto Spectrum, or FRF data, and expands it to narrow-band data. It determines the N-octave reduction by examining the OctaveFormat attribute. It will automatically determine whether the N-octave bands are ANSI or calculated from formulas by examining the abscissas.

NPTS is an optional input specifying the number of points to be used in the first N-octave band for the narrowband output. The default is 10. This value determines the frequency spacing for the output functions.

The optional input argument `'silent'` suppresses any output.

The following formulas are used for the calculated N-octave bins:

```
center = 1000*2^(i/N)
      where i = 0 for f0 = 1000 Hz

      upper = center * 2^(1/2/N)
      lower = center / 2^(1/2/N)
```

The ANSI/ISO standard frequencies used by OCTAVEN2NB are defined in ISO R 266 and ANSI S1.6-1984.

See Also

imat_fn/octaven

---

imat_fn/polyfit

Purpose

Create a polynomial fit of the supplied `imat_fn`.

Syntax

```
g=polyfit(f,2)
[g,terms]=psd(f,order)
```

Description

POLYFIT performs a polynomial regression on the supplied `imat_fn` F and returns the regressed functions in G. ORDER is a non-negative scalar specifying the polynomial order. The number of ordinate values in F should be greater than ORDER.

TERMS is an optional double matrix containing the polynomial terms. Its size is ORDER+1 x NFUNC, where NFUNC is the number of functions in F.

---

imat_fn/srs (+Signal)

Purpose

Compute shock response on the supplied input `imat_fn`.

Syntax

```
g=srs(f)
g=srs(f,[],[],imat_fn(1,'IDLine1','Template'))
g=srs(f,freq,damp,'pos','neg','maxi','silent')
```

Description

SRS computes shock response spectra from the supplied time histories in the `imat_fn` vriable F. These time histories must be evenly spaced. FREQ is an optional numeric vector specifying the frequencies at which to compute the response. If none is supplied, or FREQ is empty, `logspace(-1,4,100)` will be used. DAMP is an optional numeric vector containing a list of damping ratios to use. The damping ratios should be between 0.0 and 1.0. If none is supplied, or it is empty, a damping ratio of 0.05 is used.

The response spectra calculation assumes the absolute acceleration form. The optional string input arguments define what kind of SRS will be computed. Available choices are `'pos'`, `'neg'`, `'maxi'`, and `'all'`. The default is `'maxi'`. To specify these, you must also specify FREQ and DAMP, although you can pass in empty matrices to use the defaults.

The optional input string `'silent'` suppresses output.

OUTFN is an `imat_fn` containing the output. It is dimensioned as MxNxP, where M is the number of SRS functions (positive, negative, maximax), N is the number of damping values, and P is the number of supplied time histories supplied to SRS. IDLine3 contains a description of the data type and damping value used for the function. UserValue3 contains the Q value used for the function (1/2*damping fraction).

The core of SRS uses routines provided by David Smallwood at Sandia National Laboratories. SRS is simply a wrapper around these routines to accommodate IMAT data types.

PORTIONS COPYRIGHT 1994-1998 SANDIA NATIONAL LABORATORIES

Examples

```
>> f=imat_fn(1,'AbscissaInc',1/100,'Ordinate',ones(1,100),'OrdNumDataType','Acceleration');
>> g=srs(f)
Portions of this code provided by Dave Smallwood and are (C) Sandia National Laboratories

g =

IMAT Function with the following attributes
Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1MAXI,1X+)                Shock Response S Uneven           100

>> g=srs(f,[],.01:.01:.05,'pos','neg')
Portions of this code provided by Dave Smallwood and are (C) Sandia National Laboratories

g =

2x5 IMAT Function with the following attributes:
Row Col Record Name          FunctionType     AbscissaSpacing  NumberElements
--- --- -------------------- ---------------- ---------------- ----------------
1   1   1_(1CPOS,1X+)        Shock Response S Uneven           100
2   1   2_(1CNEG,1X+)        Shock Response S Uneven           100
1   2   3_(1CPOS,1X+)        Shock Response S Uneven           100
2   2   4_(1CNEG,1X+)        Shock Response S Uneven           100
1   3   5_(1CPOS,1X+)        Shock Response S Uneven           100
2   3   6_(1CNEG,1X+)        Shock Response S Uneven           100
1   4   7_(1CPOS,1X+)        Shock Response S Uneven           100
2   4   8_(1CNEG,1X+)        Shock Response S Uneven           100
1   5   9_(1CPOS,1X+)        Shock Response S Uneven           100
2   5   10_(1CNEG,1X+)       Shock Response S Uneven           100

>>
```

See Also

imat_fn/srs2trans

imat_fn/srs2trans (+Modal)

Purpose

Generate transient from Shock Response Spectrum.

Syntax

```
outtrans=srs2trans(specfun,damping,deltat,duration,interpfreq)
[outtrans,outsrs,outrms]=srs2trans(spec-
fun,damping,deltat,duration,interpfreq,options,'interactive')
```

Description

SRS2TRANS generates a transient using a summation of decaying sine waves. The frequency of the decaying sine waves come from the frequencies at which the SRS is specified. The amplitudes of the individual decaying sine waves are initially guessed and then the guesses are improved by trying to minimize the RMS error between the spec SRS and the SRS of the equivalent transient. SRS2TRANS will plot the transient in one graph window and the SRS spec (blue) overlayed with the transient SRS (green) in another window. It will also list the RMS error in the command prompt window. If the RMS is less than the specified RMS tolerance, SRS2TRANS will quit.

SPECFUN is a single `imat_fn` containing a shock response spectrum. DAMPING is the damping coefficient (critical damping factor) used for computing equivalent transient. If the value is less than 0.0, default damping will be used. The default value is derived from IDLine3. SRS2TRANS looks for a string of the format set by the SRS method.

If the interpolation method is not specified by INTERPFREQ, UserValue4 of SPECFUN is used. If UserValue4 of SPECFUN does not contain an integer between 0 and 3, the default interpolation will be used (linear on both axes). To compare transient functions back to the original shock response, an SRS is computed. It is useful to compare SRS functions computed in the same way. So, if SPECFUN has a REFERENCEDIR property that has one of the following values (`pos`, `ppri`, `cres`, `cpos`, `neg`, `npri`, `nres`, `cneg`, `maxi`, `mpri`, or `mres`), the SRS calculation will use that option. If REFERENCEDIR contains some other string, the default `'maxi'` will be used.

********* WARNING ********

It is crucially important that the user know what damping is being used. The SRS definition is not complete without the damping value at which it was created.

********* WARNING ********

DAMPCOEF is the critical damping factor associated with the SRS. It must be a real number between 0 and 1.

DELTAT is the time increment to be used in the equivalent transient function. DURATION is the total length of the equivalent transient function.

INTERPFREQ is a cell array. If the cell array is empty, the frequencies at which the error between the spec and the equivalent transient will be compared will be taken from SPECFUN. If the cell array is not empty, the first cell will contains an array of frequencies at which the spec and equivalent transient will be compared. If there is a second element of the array, it contains a string identifying the method for interpolating SPECFUN. If this cell does not contain a string, the interpolation type specified by SPECFUN will be used. The interpolation type is specified in the UserValue4 property. The default interpolation type is linear for both axes.

OPTIONS is an optional 2x1 array containing the maximum number of iterations (default=100), and user-specified tolerance for convergence of RMS error (default=1.0). If the string `'interactive'` is supplied, the function runs interactively, plotting candidate functions for the user's acceptance/rejection. Otherwise, the function will iterate a set number of times without providing any visual feedback.

OUTTRANS is an `imat_fn` containing the transient from the lowest RMS error iteration. OUTSRS is an `imat_fn` containing the SRS corresponding the lowest RMS error iteration. OUTRMS contains the RMS value of the lowest RMS iteration.

See Also

imat_fn/srs

---

imat_fn/sweep (+Modal)

Purpose

Generate a sweep function.

Syntax

```
g=sweep(10,.001,1000)
g=sweep([10 20],.001,'time',5.0)
g=sweep([10 20],1/1000,ftemplate,'cos','log')
g=sweep(freq,damp,npts,ftemplate,type,spacing)
```

Description

SWEEP generates a sine, cosine, or square wave sweep, based on the specified inputs. It returns the output in the `imat_fn` F.

Three input arguments are required. FREQ is a 1x2 vector or a scalar specifying the frequency range over which to sweep. DELTAT is a scalar giving the time spacing between points. NPTS is a scalar specifying the number of data points to create. An alternate specification method is to pass in the string `'time'` followed by a number containing the duration (in seconds) of the sweep. SWEEP will calculate the number of points that most closely achieves this end time based on the provided DELTAT.

FTEMPLATE is a required `imat_fn` input that provides a template function in which to place the sweep function. It is required so that MATLAB can find this function. You can use this to pre-specify attributes in the output function. If you don't have a template function, simply pass in an empty `imat_fn`.

TYPE is a string specifying the type of sweep function you wish to create. Valid options are `'sin'`, `'cos'`, and `'square'`. The default is `'sin'`.

SPACING is a string specifying the sweep type. Valid options are `'lin'` for a linear sweep and `'log'` for a logarithmic sweep. The default is `'lin'`.

NOTE: Care should be taken when specifying a different start and end frequency. You may want to sweep a slightly wider frequency range if you are concerned about the energy content in the frequency band of interest. It is a good idea to review a FFT of your sweep time history.

See Also

imat_fn/fft

---

imat_fn/cumrms

Purpose

Compute cumulative root mean square (RMS) from the supplied frequency domain `imat_fn`.

Syntax

```
g=cumrms(f)
g=cumrms(f,'reverse')
```

Description

CUMRMS calculates the cumulative root mean square value of the supplied frequency domain `imat_fn`, using the trapezoidal rule to integrate the function. CUMRMS looks at the AbscissaDataType attribute to determine if the function is in the frequency domain. It returns the output in the `imat_fn` G.

The optional input string specifying that CUMRMS should calculate the cumulative RMS in reverse order, from highest frequency to lowest.

If the function in F is a frequency domain function and is a spectrum (any type) or frequency response function, RMS will be calculated in one of several ways. The AmplitudeUnits and Normalization attributes become very important.

If the AmplitudeUnits are squared and normalized per Hz, the RMS is calculated by taking the square root area under the curve. Otherwise, the bandwidth (ΔF) is not used in the calculation.

RMS values for linear amplitude functions such as spectra and FRF are calculated by taking the square root of the sum of the absolute values of the amplitude squared.

RMS determines whether the frequency domain function is linear or squared by looking at Amplitude Units and OrdNumDataType. If either of these attributes contain the word `'squared'`, or FunctionType is Power Spectral Density, Auto Spectrum, or Cross Spectrum, the units are considered to be squared.

RMS also takes into account the AmplitudeUnits attribute. If it is `'Peak'`, the calculated RMS value from the above formulas will be multiplied by 1/sqrt(2). If it is `'Half-Peak'`, it will be multiplied by sqrt(2).

See Also

imat_fn/Statistical Functions

---

imat_fn/rms

Purpose

Root mean square.

Syntax

```
g=rms(f)
g=rms(f,'linlog')
g=rms(f,MODE)
```

Description

SQRT takes the square root of the `imat_fn`M. NUM must be a scalar. Complex results are produced if F is not positive.

RMS calculates the root mean square value of the supplied imat_fn. For evenly spaced functions the formula used is

```
rms = sqrt( 1/length(y) * sum(y.^2) );
```

Note that when F is unevenly spaced, RMS will use the formula based on the formula for a continuous function,

```
rms = sqrt(1./(x(end)-x(1)) * sum((y.^2).*[0; diff(x)]))
```

This can cause differences in the RMS value calculated versus an equivalent evenly spaced function.

If the function in F is a frequency domain function and is a spectrum (any type) or frequency response function, RMS will be calculated in one of several ways. The AmplitudeUnits and Normalization attributes become very important. Unfortunately, I-deas and NX do not set these attributes properly, so in this case if the FunctionType is Power Spectral Density and the AmplitudeUnits and Normalization are Unknown, RMS will assume `'RMS'` and `'Units squared/Hz'`.

If the AmplitudeUnits attribute is `'Peak'`, the calculated RMS value from the above formulas will be multiplied by 1/sqrt(2). If it is `'Half-Peak'`, it will be multiplied by sqrt(2).

If the Normalization is `'Units squared/Hz'`, the RMS is calculated by taking the square root area under the curve. Otherwise, the bandwidth (delta F) is not used in the calculation. For unevenly spaced functions, the trapezoidal rule is used to get the area under the curve. Otherwise, the absolute value of the ordinate values are summed.

RMS values for linear amplitude functions such as spectra and FRF are calculated by taking the square root of the sum of the absolute values of the amplitude squared.

RMS determines whether the frequency domain function is linear or squared by looking at Normalization and OrdNumTypeQual. If either of these attributes contain the word `'squared'`, or FunctionType is Power Spectral Density, Auto Spectrum, or Cross Spectrum, the units are considered to be squared.

MODE is an optional input string specifying the interpolation type to use. Valid options are `'linlin'` (default), `'linlog'`, `'loglin'`, and `'loglog'`.

See also

imat_fn/mean, imat_fn/std

---

imat_fn/ Statistical Functions

Purpose

Calculate statistics on the supplied `imat_fn`, including minimum, maximum, mean, standard deviation, variance, skewness, kurtosis, and RMS.

Syntax

```
[g,ind]=min(f)
[g,ind]=max(f)
g=mean(f)
g=std(f)
g=std(f,1)
g=var(f)
g=var(f,1)
g=skew(f)
g=kurt(f)
g=rms(f,mode)
```

Description

The following statistical functions are supported.

TYPE	Description
`min`	Minimum value.
`max`	Maximum value.
`mean`	Mean value.
`std`	Standard deviation.
`var`	Variance.
`skew`	Skewness.
`kurt`	Kurtosis.

In each case, the function will return a matrix of the same size as F that contains the value being calculated. Some of the functions support multiple input arguments. These are described in more detail below.

Both MIN and MAX support multiple output arguments. The second output argument IND is an index into each function for the ordinate value representing the minimum or maximum value, respectively.

STD

STD normalizes F by (N-1), where N is the number of elements in each function. This is the square root of an unbiased estimator of the variance of the population from which function values are drawn, as long as these values consist of independent, identically distributed samples.

F = STD(A,1) normalizes by N and produces the square root of the second moment of the sample about its mean. STD(A,0) is the same as STD(A).

VAR

VAR normalizes F elements by N-1 if N>1, where N is the sample size in the function. This is the square root of an unbiased estimator of the variance of the population from which function values are drawn, as long as these values consist of independent, identically distributed samples. For N=1, Y is normalized by N.

F = VAR(A,1) normalizes by N and produces the second moment of the sample about its mean. VAR(A,0) is the same as VAR(A).

F = VAR(A,W) computes the variance using the weight vector W. The length of W must equal the number of elements of each function in F, and its elements must be nonnegative. VAR normalizes W to sum to one.

The variance is the square of the standard deviation (STD).

SKEW

Skewness is defined as the third standardized moment, and is calculated by dividing the 3rd moment of the samples by the cube of the standard deviation, which is the variance raised to the power of 3/2.

```
       μ3            1 / N * Σ (x-mean(x).^3)
g3 = ---     =   ------------------------------------
       σ3         ( 1 / N * Σ (x-mean(x).^2) ) ^ (3/2)
```

where X are the samples and N is the number of samples.

KURT

Kurtosis is defined as the third standardized moment, and is calculated by dividing the 4th moment of the samples by the cube of the standard deviation, which is the variance raised to the power of 3/2.

```
       μ4          1 / N * Σ ( (x-mean(x).^4) / N
g4 = ---    =   ------------------------------------
       σ4        ( 1 / N * Σ (x-mean(x).^2) ) ^ (4/2)
```

where X are the samples and N is the number of samples.

The kurtosis for a standard normal distribution is 3. Thus excess kurtosis is defined as KURT(A)-3. The standard normal distribution has an excess kurtosis of zero. Positive excess kurtosis indicates a "peaked" distribution and negative excess kurtosis indicates a "flat" distribution.

See Also

imat_fn/interp, imat_fn/rms

---

imat_fn/uiplot

Purpose

Plot an `imat_fn` in a special GUI that provides extended functionality.

Syntax

```
g=uiplot(f)
uiplot(f,g,...)
g=uiplot(f,z)
```

Description

UIPLOT provides a convenient graphical user interface for plotting `imat_fn`. You can plot functions individually, overlaid, or stacked. Plots are overlaid according to ordinate data type. You can control the plot style through the "Mode" pulldown in the Plot panel at the bottom left of the UIPLOT window. A context menu (right-click over the plot) allows you to window, apply tags,and change the overall plot appearance.

F is an `imat_fn` (or `fcn`). You can pass in more than one `imat_fn` variable. Z is an optional input containing an `imat_filt`, or a structure with the fields `.name` and `.index`. The `.index` field must contain valid numeric indices into F. The `.name` field must contain a string. This is the filter name that will be displayed in the UIPLOT GUI. These optional input arguments are not available when running standalone UIPLOT.

UIPLOT also provides the ability to create and apply templates to plots. Once a plot is configured as desired, all subsequent plots can have the same appearance. The settings can be saved and loaded for use during a different session.

All plots can be exported to image files or shown as a slideshow. A command window allows standard MATLAB commands to be used to format the plots.

If G is requested, UIPLOT blocks input on the command window. By default, however, UIPLOT is non-blocking, meaning control is returned immediately to the MATLAB prompt.

Starting UIPLOT

In MATLAB, you can start UIPLOT by typing `uiplot` at the MATLAB command prompt. If you are running the standalone executable, you can start UIPLOT through Windows *Start* -> *Programs* menus. This will bring up the main interface, shown in Figure 1.

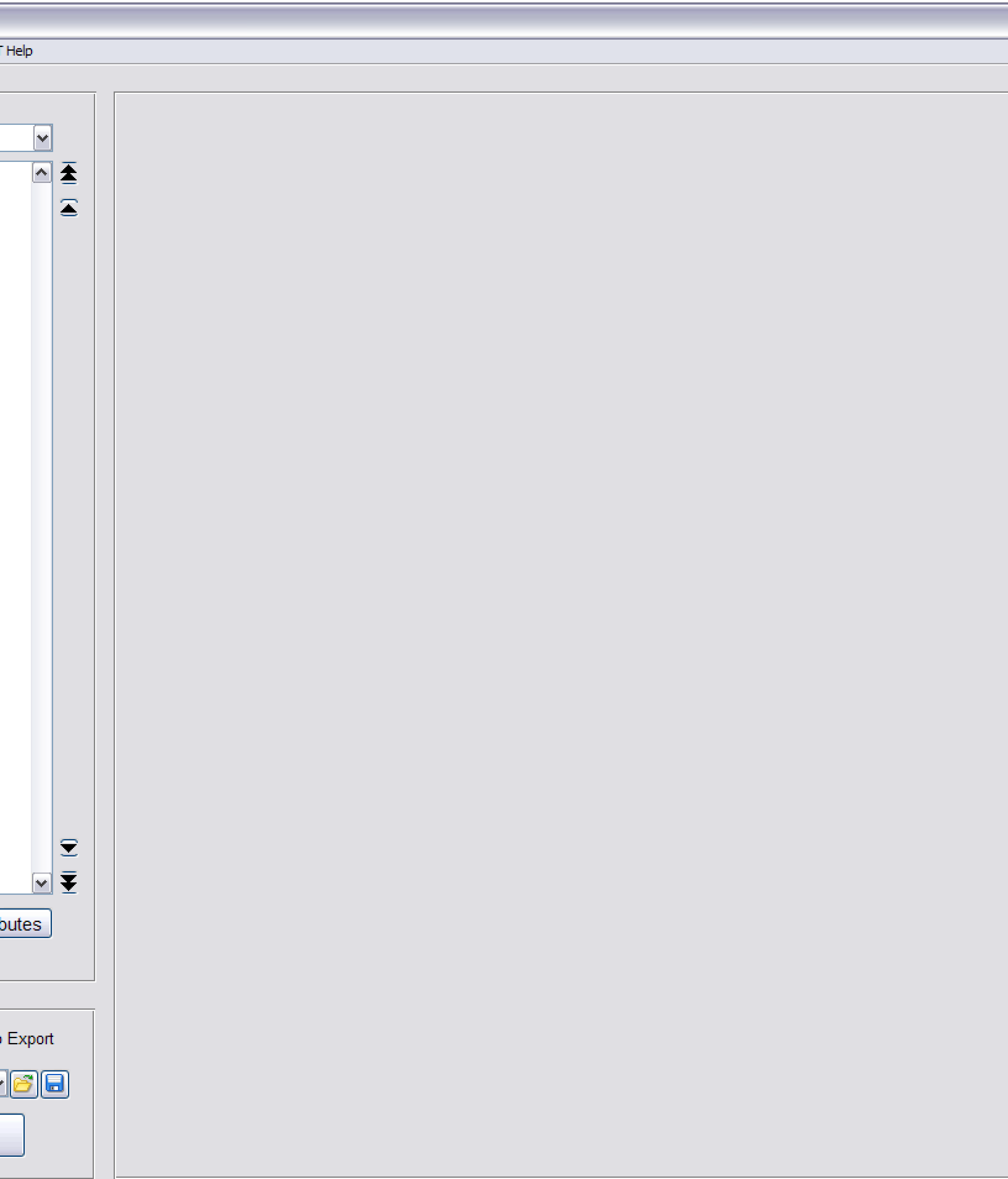

Help
butes
Export

Figure 1: UIPLOT Main Interface

Main Interface

The main interface of UIPLOT has four distinct functional areas, which are highlighted in Figure 2.

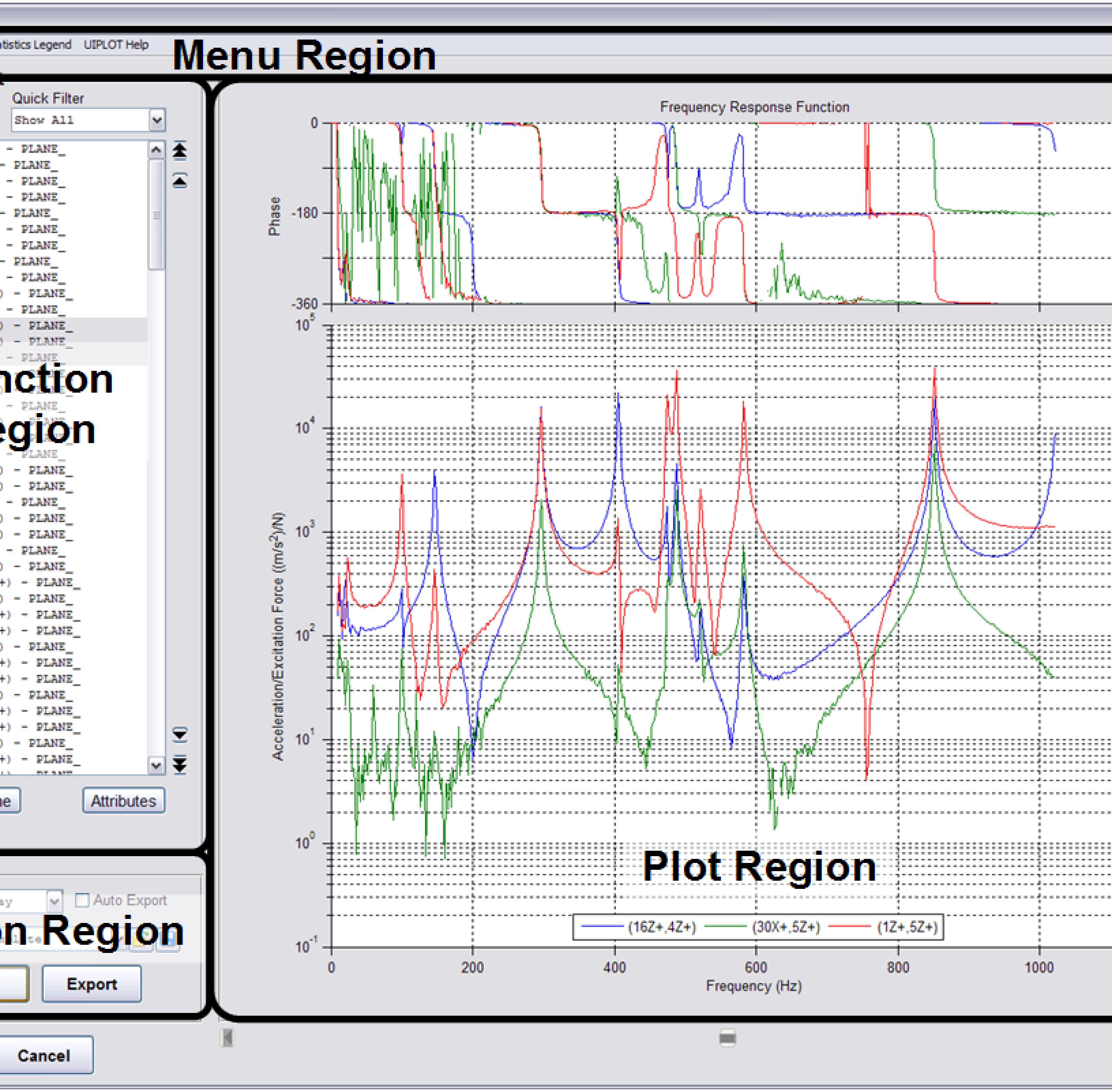


Figure 2: UIPLOT interface regions.

The **Plot Region** contains the axes where all plots will be created. The set of plots currently displayed is considered a slide. As mentioned previously, if the requested slide will contain more axes than can be displayed, they will overflow onto addition slides and can be navigated using the buttons at the bottom of the Plot Region. The units pulldown allows you to change the plot display units. This only affects the plot; it will not modify the data returned by UIPLOT.

The **File** menu in the **Menu Region** controls the most basic functions of UIPLOT.

Load File	Functions are loaded by selecting the **Load File** menu item and subsequently selecting an appropriate file from the *Open File* dialog box. The loaded functions will appear in the function list in the Function Region at the left side of the interface. Supported file formats available to load are FCN, AFU, ATI, PCH (Nastran) and `.vra_xyout` (Vibrata).
Load Workspace	The **Load Workspace** menu item allows you to select variables from the base workspace to load into UIPLOT. The loaded functions will appear in the function list in the Function Region at the left side of the interface. Supported function types are IMAT_FN and FCN.
Plot	The **Plot** menu item will plot the selected functions in the Plot Region. The plot preferences are set using the controls in the **Action Region**. After any change has been made to the plot options, the **Plot** button must be pressed again to plot the functions with the new options. Plots can be overlaid by type, stacked, or cycled. This is controlled by the *Mode* pulldown. The *Overlay by Type* option will plot the functions grouped by ordinate data type. The *Stack* option will plot each function individually, with the total number on each slide controlled by the *Plots Per Page* option in the *Plot Options* menu. The *Cycle* option will plot one function on a slide in the Plot Region. In all cases, if the plots do not fit onto one slide, the navigation buttons at the bottom of the plot will be activated and can be used to navigate the slides. The current slide and total number of slides will appear at the lower left of the Plot Region. By default, each plot also will contain a legend describing the lines of the plot. This option can be toggled on or off from the *Plot Action* menu.
Save File	The **Save File** menu item will save any selected functions in the Function Region to a new file. The supported file format for output is I-deas Associated Data Files (AFU or ATI).
Save Workspace	The **Save Workspace** menu item will save any selected functions in the Function Region to the base workspace.
Export	The **Export** menu item will save the current slide to an image file. The available formats are BMP, EMF, PNG, PS, or TIFF. The format for output can be changed by selecting the format on the dialog box.

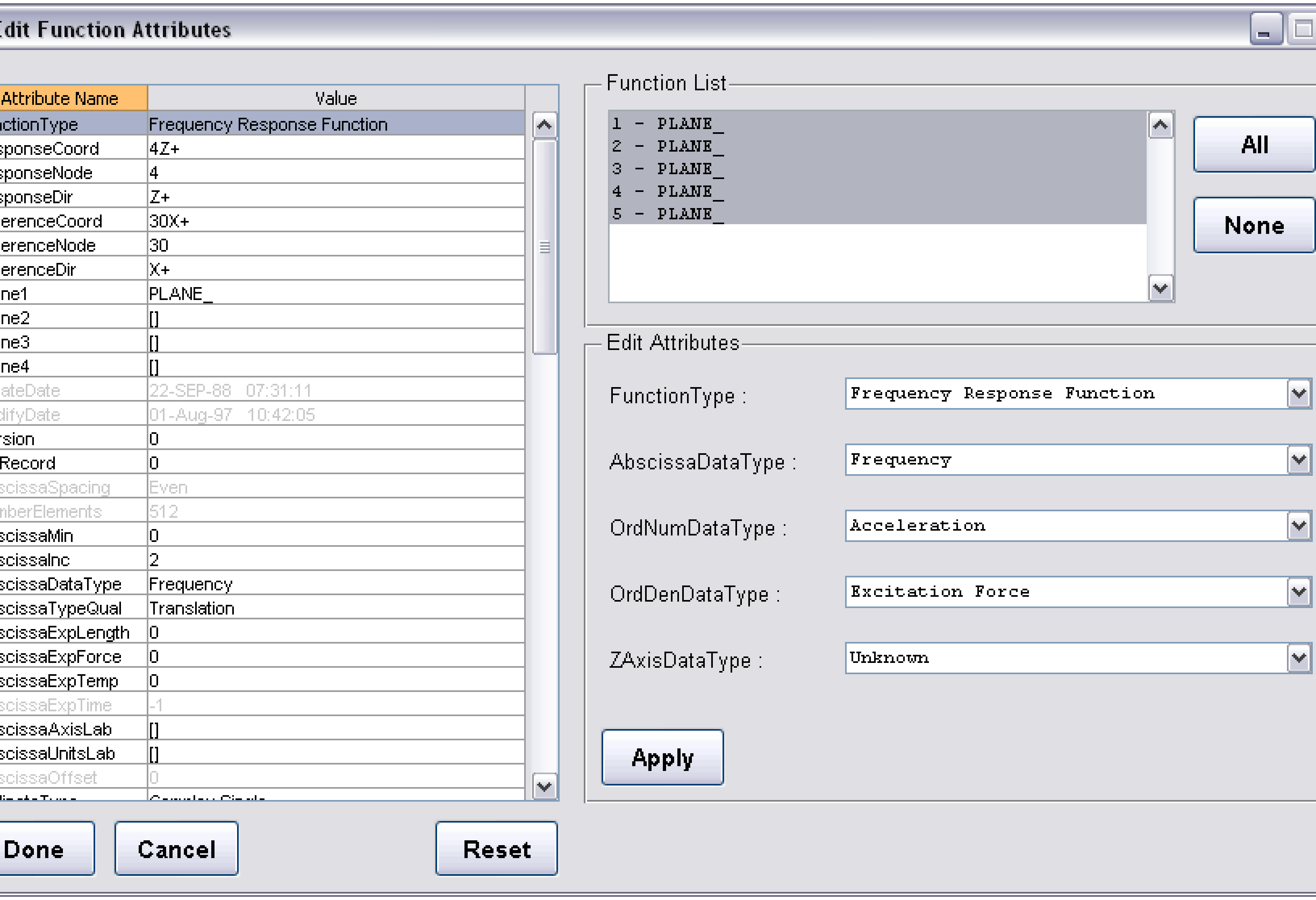
The **Action Region** controls how UIPLOT creates each slide.

Plot	The **Plot** menu item will plot the selected functions in the **Plot Region**. The plot preferences are set using the controls in the **Action Region**. After any change has been made to the plot options, the **Plot** button must be pressed again to plot the functions with the new options. Plots can be overlaid by type, stacked, or cycled. This is controlled by the *Mode* pulldown. The *Overlay by Type* option will plot the functions grouped by ordinate data type. The *Stack* option will plot each function individually, with the total number on each slide controlled by the *Plots Per Page* option in the **Plot Options** menu. The *Cycle* option will plot one function on a slide in the **Plot Region**. In all cases, if the plots do not fit onto one slide, the navigation buttons at the bottom of the plot will be activated and can be used to navigate the slides. The current slide and total number of slides will appear at the lower left of the **Plot Region**. By default, each plot also will contain a legend describing the lines of the plot. This option can be toggled on or off from the **Plot Action** menu.
Mode	The **Mode** pulldown controls whether the plot will be overlaid by type, stacked or cycled. See the text for **Plot** for more information.
Auto Export	If enabled, all plots subsequently generated by the **Plot** button will be automatically exported to a file. A dialog box will first appear however, requesting a base name for the outputted files. The format is controlled by the **Save As Type** pulldown on the dialog box.

Template	All loaded templates will appear in the **Template** pulldown. To use a template, select it from the list and press **Plot**. To plot with out a template, select *No Template* as the template.
Open Template	The **Open Template** button has a folder icon on it. This button opens an existing template file and adds it to the list of available templates.
Save Template	The **Save Template** button has a disk icon on it. This button saves the settings of the current slide to a file. A file dialog will appear where the name and location of the file can be entered.

The **Function Region** displays the functions that have been loaded into UIPLOT and provides methods for selecting specific functions. The single arrow buttons to the right of the list will move the current selection in the function list up or down. The double arrow buttons will jump the current selection to the very top or bottom of the list.

All	The **All** button selects all of the functions in the function list box.
None	The **None** button deselects all of the functions in the function list box.
Select...	The **Select...** button is used to create a custom selection of functions, and selecting it will open the Filter Functions window, seen in Figure 4. This window displays all loaded functions, and provides pulldown menus for displaying function attributes. A custom filter can be created by selecting the **Filter...** button at the top-right of the window. This will bring up the form seen in Figure 5. The filter generated by **Select...** will then be applied to the functions listed on the main UIPLOT window. To select functions using a custom filter, combine the attribute, attribute value, and operator at the bottom of the form as desired, and press the **Add** button. This will add the filter to the list. Multiple filters can be combined to form a single filter. Once the desired combination of filters has been created, press the **OK** button to return to the Filter Functions window. The function list in this window should now display the list of functions after the new filter has been applied. Select **OK** to return to the main interface.
Quick Filter	The **Quick Filter** pulldown is used to quickly apply filters to the full function list. Standard filters are Drive Point and Reciprocity. The <<Select Button>> item is used to return to the selection that was made using **Select...**
Attributes	The **Attributes** button will launch the *Edit Function Attributes* window seen in Figure 3, with the selected functions from the **Function Region** preloaded into the interface. From this window, any editable function attribute can be changed. The functions will appear in the Function List section of the window, with all the attributes listed in the table on the left side of the window. Selecting an attribute from the table will refresh the Edit Attributes section, where the attributes can be edited. To set the changes, press the **Set** button and the changes will be reflected in the table. The **Done** button on this form will save the changes made, and return to UIPLOT. The **Cancel** button will also close the window and return to UIPLOT without saving anychanges that were made to the attributes. The **Reset** button will reset the functions to their state when the Edit Function Attributes window was opened, but will keep the window open so new changes can be made.

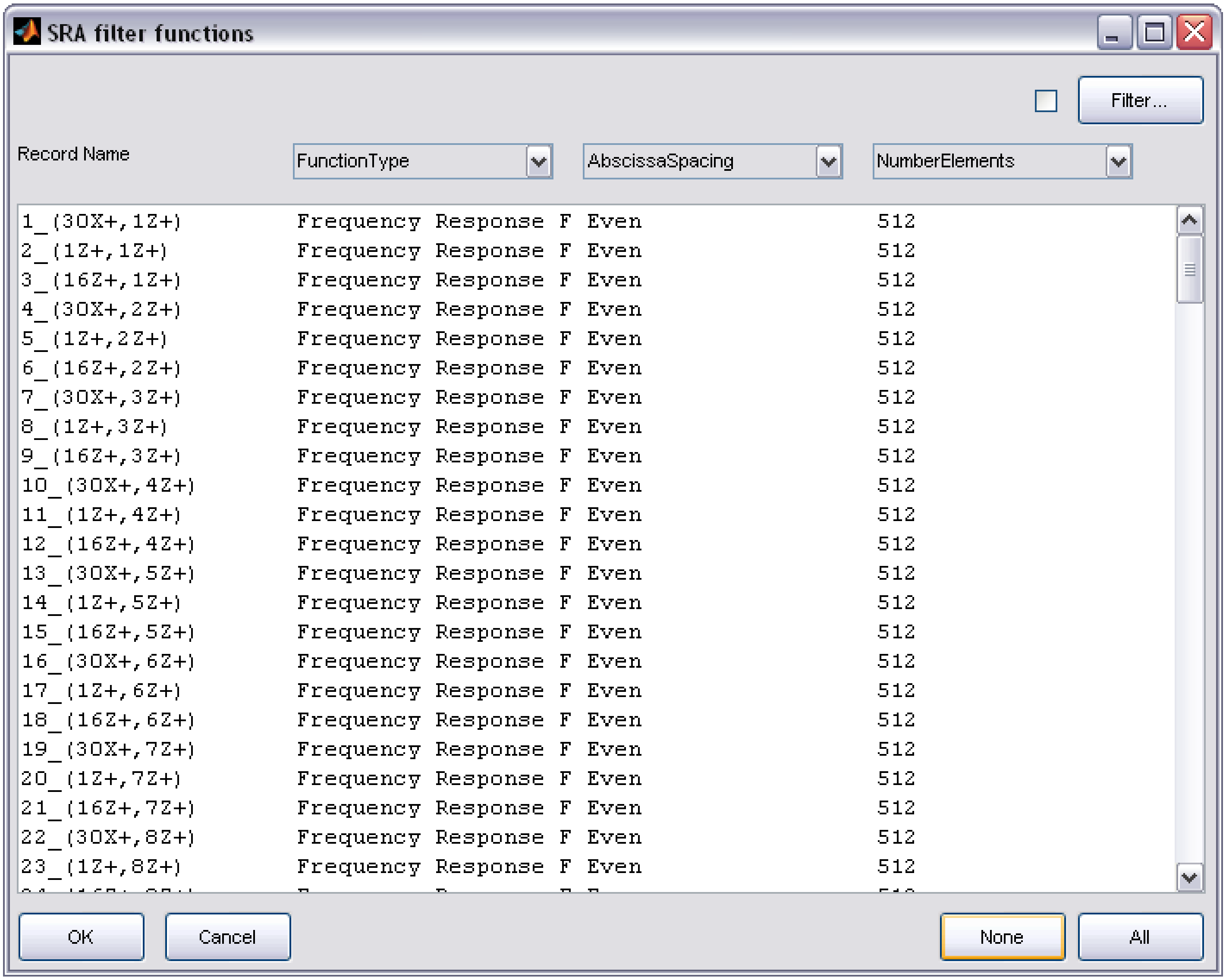


Attribute Name	Value
ctionType	Frequency Response Function
sponseCoord	4Z+
sponseNode	4
sponseDir	Z+
erenceCoord	30X+
erenceNode	30
erenceDir	X+
ne1	PLANE_
ne2	[]
ne3	[]
ne4	[]
ateDate	22-SEP-88 07:31:11
difyDate	01-Aug-97 10:42:05
rsion	0
Record	0
scissaSpacing	Even
nberElements	512
scissaMin	0
scissaInc	2
scissaDataType	Frequency
scissaTypeQual	Translation
scissaExpLength	0
scissaExpForce	0
scissaExpTemp	0
scissaExpTime	-1
scissaAxisLab	[]
scissaUnitsLab	[]
scissaOffset	0

1 - PLANE_
2 - PLANE_
3 - PLANE_
4 - PLANE_
5 - PLANE_

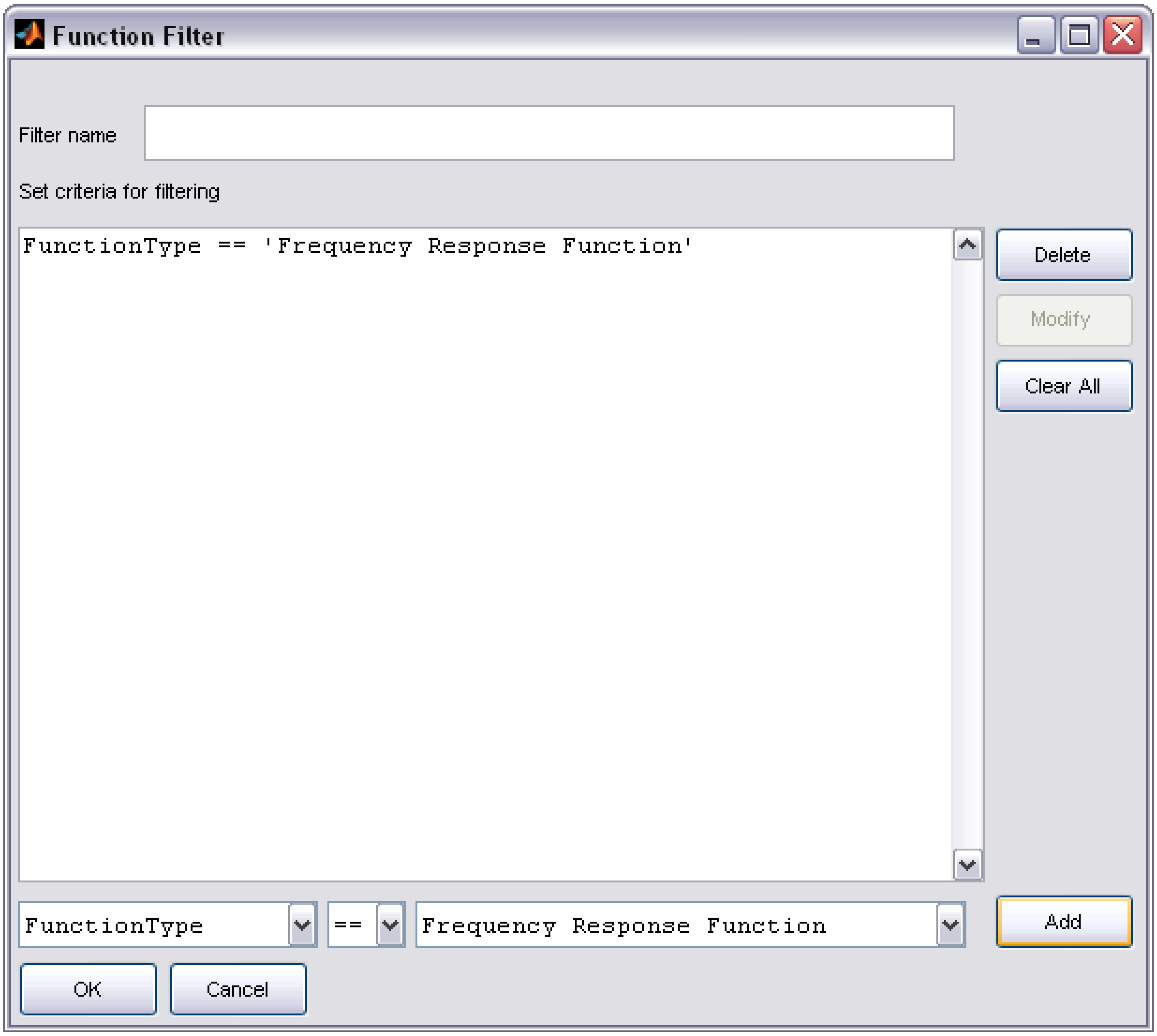
FunctionType : Frequency Response Function
AbscissaDataType : Frequency
OrdNumDataType : Acceleration
OrdDenDataType : Excitation Force
ZAxisDataType : Unknown

Figure 3: Edit Function Attributes Window



Record Name	FunctionType	AbscissaSpacing	NumberElements
1_(30X+,1Z+)	Frequency Response F	Even	512
2_(1Z+,1Z+)	Frequency Response F	Even	512
3_(16Z+,1Z+)	Frequency Response F	Even	512
4_(30X+,2Z+)	Frequency Response F	Even	512
5_(1Z+,2Z+)	Frequency Response F	Even	512
6_(16Z+,2Z+)	Frequency Response F	Even	512
7_(30X+,3Z+)	Frequency Response F	Even	512
8_(1Z+,3Z+)	Frequency Response F	Even	512
9_(16Z+,3Z+)	Frequency Response F	Even	512
10_(30X+,4Z+)	Frequency Response F	Even	512
11_(1Z+,4Z+)	Frequency Response F	Even	512
12_(16Z+,4Z+)	Frequency Response F	Even	512
13_(30X+,5Z+)	Frequency Response F	Even	512
14_(1Z+,5Z+)	Frequency Response F	Even	512
15_(16Z+,5Z+)	Frequency Response F	Even	512
16_(30X+,6Z+)	Frequency Response F	Even	512
17_(1Z+,6Z+)	Frequency Response F	Even	512
18_(16Z+,6Z+)	Frequency Response F	Even	512
19_(30X+,7Z+)	Frequency Response F	Even	512
20_(1Z+,7Z+)	Frequency Response F	Even	512
21_(16Z+,7Z+)	Frequency Response F	Even	512
22_(30X+,8Z+)	Frequency Response F	Even	512
23_(1Z+,8Z+)	Frequency Response F	Even	512

Figure 4: Filter Functions Window



```
FunctionType == 'Frequency Response Function'
```

FunctionType == Frequency Response Function

Figure 5: New Filter Window

Contextual Menus

The contextual menu, which controls many aspects of the plot appearance, can be activated by right-clicking on any plot in the Plot Region, as seen in Figure 6. The following options are available:

Window	None	When selected, axis limits will be controlled by MATLAB.
	Tight	When selected, axis limits will be adjusted to exactly envelope the data plotted.
	Pick X	When selected, X limits will be defined by manually picking them from the plot.

	Pick Y	When selected, Y limits will be defined by manually picking them from the plot.
	Pick XY	When selected, both X and Y limits will be defined by manually picking them from the plot.
	Key in X	When selected, X limits will be defined by manually typing them in.
	Key in Y	When selected, Y limits will be defined by manually typing them in.

XScale	Default	When selected, the X axis will be set to its default, which varies by Function Type.
	Linear	When selected, the X axis will be displayed on a linear scale.
	Log	When selected, the X axis will be displayed on a logarithmic (base 10) scale.

Y Scale	Default	When selected, the Y axis will be set to its default, which varies by Function Type.
	Linear	When selected, the Y axis will be displayed on a linear scale.
	Log	When selected, the Y axis will be displayed on a logarithmic (base 10) scale.

Complex Option	Default	When selected, the plot format will be set to its default.
	Modulus + Phase	When selected, the plot will be displayed in two axes. The lower axis displays the modulus (magnitude), and the upper axis displays the phase.
	Modulus	When selected, the plot will show just the modulus (magnitude).
	Phase	When selected, the plot will show just the phase.
	Real + Imaginary	When selected, the plot will be displayed in two axes. The lower axis displays the real part of the functions, and the upper axis displays the imaginary part.
	Real	When selected, the plot will display the real portion of the functions.
	Imaginary	When selected, the plot will display the imaginary portion of the functions.
	Nyquist	When selected, the plot will converted into a Nyquist plot, with the real part of the function plotted on the X axis, and the imaginary part plotted on the Y axis.

Grid	X and Y	When selected, grid lines will be applied in both the X and Y directions.
	X only	When selected, grid lines will only be applied in the X direction.
	Y only	When selected, grid lines will only be applied in the Y direction.
	None	When selected, no grid lines will be applied to the plot.

Tag	Tag on Grid	When selected, the *Tag on Grid* option will activate crosshairs to tag any grid point on a plot.
	Tag on Data	When selected, the *Tag on Data* option will activate crosshairs to tag any data point on a plot. Any click on the plot will tag the nearest data point.
	Tag X	When tags are applied, only the X coordinate will be tagged.
	Tag Y	When tags are applied, only the Y coordinate will be tagged.
	Tag X and Y	When tags are applied, both the X and Y coordinate will be tagged.

	Delete Tags	When selected, all tags on the plot will be deleted.
	Cursor	The cursor toggle provides a convenient vertical line to assist in tagging.

Labels	X Label	Changes the X Label of the plot. If the plot is two axes, the X Label always appears on the lower plot.
	Y Label	Changes the Y Label of the axes clicked.
	Title	Changes the Title of the plot. The title always appears on the upper-most axes.
	Reset	Resets the X Label, Y Label, and Title to their default values. The default values are determined by what type of data is plotted.

Links	Link Axes	This toggle links the axes of a multi-axes plot together. By default, this is turned on. This controls things like linking the X Windowing of a multi-axis plot together.
	Link Plots	This toggle links multiple plots on a single figure together. If selected, the selected plot will be “listening” when other plots change. For instance, if the complex option of another plot on the same figure changes, all plots who have this toggle selected will change their complex option as well.

Legend	Show Legend	Toggles whether the legend is shown on the figure.
	Default	Sets the legend to label the plotted lines with the default labels.
	IDLine1	Sets the legend to label the plotted lines with their IDLine1 value.
	IDLine2	Sets the legend to label the plotted lines with their IDLine2 value.
	IDLine3	Sets the legend to label the plotted lines with their IDLine3 value.
	IDLine4	Sets the legend to label the plotted lines with their IDLine4 value.
	Ref/Res	Sets the legend to label the plotted lines with their ReferenceCoord and ResponseCoord values.
	Font	Sets the legend font properties.
	Location	This contains a submenu that controls the location of the legend.
	Orientation	This contains a submenu that controls the orientation of the legend.
	Items Per Plot	This controls the maximum number of lines that will be included in the legend.

Plot Style		This brings up the Plot Style window where plotted functions can be formatted individually.

Clean Phase		If enabled, points where the phase differs by more than 270 degrees from the previous point will not be displayed. This option is enabled by default.

Save Template		Stores the settings of the current plot to a template file for later use.

Load Template		Loads a template from a file and applies the settings to the plot.

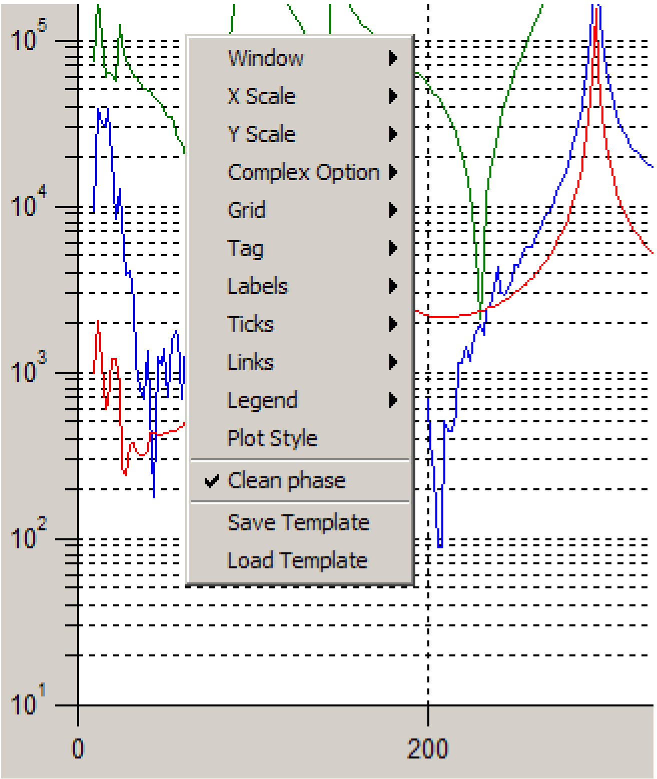


Figure 6: Contextual Menu

Menu Options

The *Options* menu at the top of UIPLOT is the location of numerous references and actions for controlling UIPLOT.

Plot Options	Legend	Opens another menu, where you can select which attribute gets displayed in the plot legend. The default is to display the reference and response coordinate. You can also specify how many legends to display per plot.
	Plots Per Page	Controls the maximum number of plots that will appear on one slide. When selected, a dialog box will appear where the new setting can be input. The menu text will be updated to reflect the current setting.
	Link All Axes	This toggle links the axes of a multi-axes plot together. By default, this is turned on for all plots. Toggling this will change the setting for all plots on the current slide.
	Link All Plots	This toggle links multiple plots on a single slide together. If toggled, whenever a plot updates, all plots on the slide will be updated identically.
	Grouping	This option controls how plots are grouped for stacked and cycled plots. By default, the plots are not grouped. Optionally, they can be grouped by reference or response coordinate.
	Slideshow	When enabled, all plots which result in multiple slides will automatically cycle. The time between the display of each slide is controlled through the *Slideshow Interval* option under the same submenu. The slide show can be looped by enabling the Loop option. A slideshow can be stopped by pushing the stop button at the bottom of the **Plot Region**.

Statistics Legend	Enable	When activated, a secondary legend will be added to the right of the plot which documents the statistics of the plot. The available statistics are Maximum, Minimum, Mean, and RMS. By default, they are all enabled, but can be toggled on or off through the same menu.

	Max	The maximum ordinate value on the plot.
	Min	The minimum ordinate value on the plot.
	Mean	The mean ordinate value on the plot.
	RMS	The root mean square (RMS) ordinate value on the plot.

Other Options

UIPLOT also offers a host of other features. The slideshow feature can be accessed through *Plot Options* -> *Slideshow*. When enabled, all plots which result in multiple slides will automatically cycle. The time between the display of each slide is controlled through the *Slideshow Interval* option under the same submenu. The slide show can be looped by enabling the *Loop* option. A slideshow can be stopped by pushing the stop button at the bottom of the Plot Region.

The statistical legend can be enabled by navigating to *Options* -> *Statistical Legend* -> *Enable*. When activated, a secondary legend will be added to the right of the plot which documents the statistics of the plot. The available statistics are Maximum, Minimum, Mean, and RMS. By default, they are all enabled, but can be toggled on or off through the same menu.

The auto export feature toggle found in the plot panel. It works exactly like the **Export** button discussed earlier; however, when activated, it will automatically export all slides generated when the **Plot** button is pressed. The user will be prompted by a file dialog to provide a base name for the image files. The extension of the file will determine the format of the output.

See Also

imat_fn/plot, imat_fn/plot3, imat_fn/edit_attributes

---

imat_fn/xform

Purpose

Coordinate transform functions and time histories.

Syntax

```
g=xform(f,fem)
g=xform(f,fem,csto)
g=xform(f,fem,csfrom,csto)
g=xform(f,fem,'silent')
```

Description

XFORM will transform functions and time histories in the supplied `imat_fn` using coordinate system and node information provided in the IMAT_FEM object FEM. The `imat_fn` can be multi-dimensional and contain functions of various sizes. However, the functions for a given reference coordinate and response node must all be the same length. The coordinate system transformation will occur for the ordinate values at each abscissa data point. Coordinate system transformations can be local to global, global to local, or local to local. Transformations can occur between any two arbitrary coordinate systems.

CSFROM and CSTO are coordinate system labels to transform from and to, respectively. They can either be a vector of the same length as the number of nodes in the function, or a scalar. If neither is supplied, the coordinate system(s) to tranform from will be extracted from the third column (current node coordinate system) of the `cs` field of the `node` structure. The coordinate system(s) to transform to will be extracted from the second column (node displacement coordinate system) of the `cs` field of the `node` structure. If the nodes are currently in local coordinates, XFORM assumes that you want to transform back to the global. If only CSTO is supplied, the coordinate system to transform from will be extracted from the second column (node displacement coordinate system) of the `cs` field of the `node` structure. Coordinate system 0 is defined as the global Cartesian coordinate system.

Passing in the string `'silent'` suppresses output to the screen.

When transforming coordinates, XFORM assumes that the function values at nodes with coordinate systems defined in cylindrical and spherical coordinate systems are actually stored in Cartesian coordinates for a Cartesian system based on the node location in the cylindrical or spherical system. As a result it only applies a Cartesian transformation. The coordinate system offsets are also not applied. For cylindrical and spherical coordinate transforms, shape coefficient transformations for nodes located at a local origin may produce inaccurate results due to numerical roundoff.

XFORM will return an `imat_fn` containing transformed data. The size of G may not match the size of F. For each node and reference coordinate set present in the F data, XFORM will create either 3 or 6 new functions in which to store the transformed data.

Examples

```
>> fem = readunv('simple_fem.unv');
>> fem.node.id        % Node labels

ans =
     100
     200
     300
     400
     500

>> fem.node.cs           % Nodal coordinate system definitions

ans =
     2     2     0
     2     2     0
     2     2     0
     2     2     0
     2     2     0

>> fem.cs.matrix(:,:,2)    % Transformation matrix for CS 2

ans =
    0.5000   -0.4330    0.7500
         0    0.8660    0.5000
   -0.8660   -0.2500    0.4330
    1.0000    2.0000    3.0000

>> f = readadf('simple_functions.afu')

f =
10x1 IMAT Function with the following attributes:
Record Name                 OrdinateType     AbscissaSpacing  NumberElements
--------------------------- ---------------- ---------------- ----------------
1_(1X+,100X+)               Real Single      Even             2
2_(1X+,200X+)               Real Single      Even             2
3_(1X+,300X+)               Real Single      Even             2
4_(1X+,400X+)               Real Single      Even             2
5_(1X+,500X+)               Real Single      Even             2
6_(1X+,100Y+)               Real Single      Even             2
7_(1X+,200Y+)               Real Single      Even             2
8_(1X+,300Y+)               Real Single      Even             2
9_(1X+,400Y+)               Real Single      Even             2
10_(1X+,500Y+)              Real Single      Even             2

>> f.ordinate

ans =
     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1

>> g = xform(f,fem)
Function: Transforming GLOBAL->LOCAL
```

```
g =
15x1 IMAT Function with the following attributes:
Record Name                 OrdinateType     AbscissaSpacing  NumberElements
--------------------------- ---------------- ---------------- ----------------
1_(1X+,100X+)               Real Single      Even             2
2_(1X+,100Y+)               Real Single      Even             2
3_(1X+,100Z+)               Real Single      Even             2
4_(1X+,200X+)               Real Single      Even             2
5_(1X+,200Y+)               Real Single      Even             2
6_(1X+,200Z+)               Real Single      Even             2
7_(1X+,300X+)               Real Single      Even             2
8_(1X+,300Y+)               Real Single      Even             2
9_(1X+,300Z+)               Real Single      Even             2
10_(1X+,400X+)              Real Single      Even             2
11_(1X+,400Y+)              Real Single      Even             2
12_(1X+,400Z+)              Real Single      Even             2
13_(1X+,500X+)              Real Single      Even             2
14_(1X+,500Y+)              Real Single      Even             2
15_(1X+,500Z+)              Real Single      Even             2

>> g.ordinate

ans =
  Columns 1 through 7

    0.0670    0.8660   -1.1160    0.0670    0.8660   -1.1160    0.0670
    0.0670    0.8660   -1.1160    0.0670    0.8660   -1.1160    0.0670

  Columns 8 through 14

    0.8660   -1.1160    0.0670    0.8660   -1.1160    0.0670    0.8660
    0.8660   -1.1160    0.0670    0.8660   -1.1160    0.0670    0.8660

  Column 15

   -1.1160
   -1.1160

>> h = xform(g,fem,2,0)
Function: Transforming LOCAL->GLOBAL

h =
15x1 IMAT Function with the following attributes:
Record Name                 OrdinateType     AbscissaSpacing  NumberElements
--------------------------- ---------------- ---------------- ----------------
1_(1X+,100X+)               Real Single      Even             2
2_(1X+,100Y+)               Real Single      Even             2
3_(1X+,100Z+)               Real Single      Even             2
4_(1X+,200X+)               Real Single      Even             2
5_(1X+,200Y+)               Real Single      Even             2
6_(1X+,200Z+)               Real Single      Even             2
7_(1X+,300X+)               Real Single      Even             2
8_(1X+,300Y+)               Real Single      Even             2
9_(1X+,300Z+)               Real Single      Even             2
10_(1X+,400X+)              Real Single      Even             2
11_(1X+,400Y+)              Real Single      Even             2
12_(1X+,400Z+)              Real Single      Even             2
```

```
13_(1X+,500X+)              Real Single      Even              2
14_(1X+,500Y+)              Real Single      Even              2
15_(1X+,500Z+)              Real Single      Even              2

>> h.ordinate

ans =
  Columns 1 through 7

    1.0000    1.0000         0    1.0000    1.0000         0    1.0000
    1.0000    1.0000         0    1.0000    1.0000         0    1.0000

  Columns 8 through 14

    1.0000         0    1.0000    1.0000         0    1.0000    1.0000
    1.0000         0    1.0000    1.0000         0    1.0000    1.0000

  Column 15

         0
         0

>>
```

See Also

imat_shp/xform, imat_fem/xform

IMAT Methods and Functions for `imat_shp` objects

---

imat_shp	Create an `imat_shp` object
imat_shp/get	Get one or more attributes of an `imat_shp` object
imat_shp/set	Set one or more attributes of an `imat_shp` object
imat_shp/validate	Validate the internal consistency of an `imat_shp`
imat_shp/setdef	Set default attributes for shape creation
imat_shp/setdisplay	Set attributes to show for shape displays
imat_shp/edit_attributes	Convenient GUI for editing `imat_shp` data attributes
imat_shp/alldof	Get a coordinate trace of all degrees of freedom in a shape

imat_shp/shp2fn	Create an `imat_shp` from the supplied `imat_fn`
imat_shp/chgunits	Convert an `imat_shp` to a different unit system
imat_shp/plot	Plot mode shapes in 3-D space
imat_shp/vtkplot	Plot and animate `imat_shp`and/or `result` in 3-D space
imat_shp/uiselect	Select shapes using a graphical interface
imat_shp/uplus	Unary plus (`+s`)
imat_shp/uminus	Unary minus (`-s`)
imat_shp/plus	Add shape coefficients (`s1+s2`)
imat_shp/minus	Subtract shape coefficients (`s1-s2`)
imat_shp/times	Termwise multiply shape coefficients (`s1.*s2`)
imat_shp/mtimes	Matrix multiply shapes (`A*s`)
imat_shp/rdivide	Termwise divide shape coefficients (`s1./s2`)
imat_shp/ldivide	Termwise divide shape coefficients (`s1.\s2`)
imat_shp/mrdivide	Matrix divide shapes (`s/A`)
imat_shp/mldivide	Matrix divide shapes (`A\s`)
imat_shp/real	Take real part of shape coefficients
imat_shp/imag	Take imaginary part of shape coefficients
imat_shp/conj	Take complex conjugate of shape coefficients
imat_shp/abs	Take absolute value (or modulus) of shape coefficients
imat_shp/phase	Replace shape coefficients with their phase angle in radians
imat_shp/phased	Replace shape coefficients with their phase angle in degrees
imat_shp/sort	Sort an `imat_shp` by frequency

imat_ctrace/build_shape	Build an `imat_shp` from shape coefficients
imat_shp/list	List an `imat_shp` nodes and shape coefficients in tabular format
imat_shp/partition	Partition an `imat_shp` to the supplied coordinate trace or node list
imat_shp/writecsv	Writes an `imat_shp` to a CSV file
imat_shp/xform	Coordinate transform shape coefficients.

---

imat_shp/imat_shp

Purpose

Create an `imat_shp` object.

Syntax

```
s=imat_shp
s=imat_shp(m,n,p,...)
s=imat_shp(m,n,p,...,'Attr',value,...)
s=imat_shp('Attr',value,...)
s=imat_shp(m,n,p,...,v)
s=imat_shp(v)
s=imat_shp(imat_result)
```

Description

You call imat_shp to construct an `imat_shp` object. An `imat_shp` object contains mode shape coefficients as well as natural frequencies, damping, and node labels.

Calling imat_shp with no arguments creates an empty shape object.

If the first one or more arguments to imat_shp are integers, then an `m`x`n`x`p`x... mode shape is created. Each element of this object is a single mode shape. (Alternatively, the dimension of the mode shape can be specified by a row vector of dimensions, as in `imat_shp([2 2])`.) The output shape will have the default attributes, which can be changed by setdef.

To override default attributes at time of object creation, you can specify one or more attribute names and values in the call to `imat_shp`. This is equivalent to creating the object with default attributes, and then setting the specified attributes in sequential order.

Instead of a list of attribute names and values, you may supply a structure variable `v` with field names equal to attribute names, and field values set to the desired attribute values. (Such a structure can be obtained from the get function with multiple attribute requests.)

A convenient way to create a new mode shape object is with the build_shape function.

If the input is an imat_result object, IMAT_SHP creates an `imat_shp` from the supplied `imat_result` object. It creates one shape for each result of type Data At Nodes.

Examples

```
>> s=imat_shp(3,'frequency',[1.55;2.36;4.02],'damping',0.01)

s =
3x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1.55                0.01                0
2   2.36                0.01                0
3   4.02                0.01                0

>>
```

See Also

imat_shp/setdef, imat_shp/set, imat_shp/get , imat_ctrace/build_shape, result

User's Guide

---

imat_shp/get

Purpose

Get one or more attributes of an `imat_shp` object.

Syntax

```
v=get(s,'Attrib')
v=get(s,'Attrib1','Attrib2',...)
v=get(s)
```

Description

When applied to an `imat_shp` object `s`, the GET function returns the values of selected attributes of `s`. The available attributes are listed here. Attribute names are not case sensitive.

If only a single attribute is requested, then the value of that attribute is returned, or printed if no output argument is supplied. Numeric attributes are returned in a numeric array with the same dimensions as `s`. String-valued attributes (including list attributes) are returned in a string variable (if `s` is a single mode) or a cell array of strings with the same dimensions as `s` (if `s` contains more than one mode). The *Node* attribute returns a numeric array with an extra initial dimension equal to the largest number of nodes in `s`. (For example, if `s` is 2x3 and all modes have 10 or fewer nodes, then the returned value is a 10x2x3 array.) The *Shape* attribute similarly returns a numeric array with an extra dimension equal to the largest number of nodal DOF (number of nodes times number of degrees of freedom per node). If any of the modes has a smaller number of nodes or shape coefficients, that column will be padded with NaN values.

If more than one attribute is specified, then the output argument will be a scalar structure variable with field names equal to the requested attributes. The value of each field will be as described above. If no output argument is supplied, the resulting structure will be printed on the standard output.

If no attributes are specified, then the values of *all* attributes will be returned as described above.

An alternative to the `get` function is the syntax `s.`*`attrib`*.

Examples

```
>> setunits('in')
Units set to IN
>> s=readunv('/ms5/examples/tda/air_test_shapes.unv')
Universal file written in MM units, converting to IN
Read 10x1 imat_shp

s =
10x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   100.334             0.00777494          30
2   146.458             0.00738474          30
3   296.134             0.00306095          30
4   296.721             0.00344297          30
5   405.738             0.00250777          30
6   474.503             0.00189975          30
7   487.771             0.00192485          30
8   521.287             0.00290228          30
9   582.747             0.00147384          30
10  851.518             0.00138781          30

>> shp=get(s,'shape');
>> size(shp)

ans =
      90    10

>> get(s(1))
         IDLine1: ''
         IDLine2: ''
         IDLine3: ''
         IDLine4: ''
         IDLine5: ''
 AbscissaAxisLab: ''
AbscissaUnitsLab: ''
 OrdinateAxisLab: ''
OrdinateUnitsLab: ''
      CreateDate: '30-Dec-97   14:30:26'
      ModifyDate: ''
       OwnerName: ''
            Node: [30x1  double]
           Shape: [90x1  double]
       Frequency: 100.3340
         Damping: 0.0078
   ModalMassReal: 5.3926e+03
   ModalMassImag: 0
ModalDampingReal: 0
ModalDampingImag: 0
   ReferenceNode: 0
    ReferenceDir: ''
  ReferenceCoord: ''
    ResponseNode: 0
     ResponseDir: ''
   ResponseCoord: ''
     NumberNodes: 30
       ShapeType: 'Real'
         DOFType: '3DOF'
  SolutionMethod: 'User Defined'
  OrdNumDataType: 'Unknown'
  OrdNumTypeQual: 'Translation'
 OrdNumExpLength: 0
  OrdNumExpForce: 0
   OrdNumExpTemp: 0
   OrdNumExpTime: 0
  OrdDenDataType: 'Unknown'
  OrdDenTypeQual: 'Translation'
 OrdDenExpLength: 0
  OrdDenExpForce: 0
   OrdDenExpTemp: 0
   OrdDenExpTime: 0

>>
```

See Also

imat_shp, imat_shp/set

---

imat_shp/set

Purpose

Set one or more attributes of an `imat_shp` object.

Syntax

```
set(s)
s1=set(s,'attrib1',value1,'attrib2',value2,...)
s(1:2)=set(s(1:2),'attrib1',value1,...)
s1=set(s,v)
```

Description

The SET function allows you to change any attribute of an `imat_shp` variable.

If SET is called with no attribute arguments, then all attribute names and their possible values are printed to standard output. This functionality acts as a built-in help mechanism for data attributes. (Note that in this case, the argument `s` is not referenced, but simply causes MATLAB to call the SET function associated with `imat_shp` objects.)

If the reference to the input variable `s` includes subscripts, you must assign the output from SET as in the 3rd example above, otherwise MATLAB will issue an error.

To change attribute values, you must either supply pairs of attribute names and values (syntax 2), or you must supply a scalar structure variable (like the one returned by get) whose field names are the attribute names to be changed, and whose field values are the desired values (syntax 3).

The attribute names can be any valid attribute for the `imat_shp` object. Attribute values can either be a single value (in which case the value is used to set all elements of `s`), or an array of values dimensioned the same size as `s`. A cell array of strings should be used to set string or list attributes. The *Node* and *Shape* attributes are special cases. These can be set either to Nx1 arrays (in which case the same values will be used for all elements of `s`), or they can be dimensioned the same size as `s` with an extra first dimension equal to the number of nodes or shape coefficients. This is consistent with the values returned by the get function. (Also, NaN values at the end of a column will be discarded when setting that element.)

The attributes are set in the order listed. Note that setting some attributes will cause side effects:

- If you increase the number of nodes by setting the *Node* attribute, then zero shape coefficients will be added to the *Shape* attribute to keep the size consistent. If you decrease the number of nodes, then shape coefficients will be truncated. The *NumberNodes* attribute will also be adjusted.
- If you increase the number of shape coefficients by setting the *Shape* attribute, then additional node labels will be added to the *Node* attribute. If you decrease the number of shape coefficients, then nodes will be truncated. The *NumberNodes* attribute will also be adjusted.
- Changing the *NumberNodes* attribute will affect the *Node* and *Shape* attributes, either by truncation (if decreasing) or by adding node labels and zero shape coefficients (if increasing).
- Setting the *Shape* attribute will automatically adjust the *ShapeType* attribute (real or complex).
- If you change the *DOFType* attribute from `'3DOF'` to `'6DOF'`, then zero shape coefficients will be added for the rotational DOF. If you change the *DOFType* attribute from `'6DOF'` to `'3DOF'`, then the rotational shape coefficients will be discarded.
- Changing the *OrdNumDataType* or *OrdDenDataType* adjusts the corresponding exponents to match. Changing the exponents can cause the data type to change to `'General'`.

An alternative way to set attributes is with the syntax `s.`*`attrib`*`=`*`value`*.

Examples

```
>> s=imat_shp(3);
>> s=set(s,'frequency',[12.35 ; 22.87; 29.02]);
>> s=set(s,'damping',[0.013 ; 0.009; 0.022], 'modalmassreal', 1.0);
>> s=set(s,'node',(1:10)', 'doftype', '3dof');
>> s=set(s,'shape', rand(30,3));
>> s

s =

3x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   12.35               0.013               10
2   22.87               0.009               10
3   29.02               0.022               10

>>
```

See Also

imat_shp, imat_shp/get, imat_shp/setdef

imat_shp/validate

Purpose

Validate the internal consistency of an `imat_shp`.

Syntax

```
s1=validate(s)
```

Description

VALIDATE returns a "validated" version of an `imat_shp` object. The validation process includes checking the internal data structure. This function is a useful check when reading back suspect data objects.

---

imat_shp/setdef

Purpose

Set default attributes for shape creation.

Syntax

```
setdef(s,'Attrib1',Value1,...)
setdef(s,0)
```

Description

SETDEF is used to change the default attributes of IMAT Shapes (i.e., the attributes set by the `imat_shp` constructor function). The first argument to SETDEF must be an `imat_shp` variable (to cause MATLAB to call the correct version of SETDEF), but its value does not matter. For example, you can use the syntax `setdef(imat_shp,...)`

In the first form, you can provide any arguments that would be acceptable to the `set` function for a single mode. These attributes will be used for future `imat_shp` variables created by the `imat_shp` constructor.

The second syntax `setdef(s,0)` restores all attributes to their original values. The original default attributes are listed in the attribute reference section.

Examples

```
>> setdef(imat_shp, 'damping', 0.01);
```

See Also

imat_shp/get, imat_shp/set

---

imat_shp/setdisplay

Purpose

Set attributes to show for shape displays.

Syntax

```
setdisplay(s,'Attrib1','Attrib2','Attrib3')
att=setdisplay(s,{'Attrib1','Attrib2'})
setdisplay(imat_shp,999)
```

Description

When an `imat_shp` is displayed on the screen (such as when a semicolon is omitted at the end of a statement), a summary of the variable is printed. By default, the *Frequency*, *Damping*, and *NumberNodes* attributes are displayed.

Input attribute names can be specified either as a series of strings, or a cell array of strings. The optional output ATT is a cell array of strings containing the current display attributes.

Use the SETDISPLAY function to change the displayed attributes. From one to three attribute names may be specified. If no attributes are specified, the default attributes will be assumed. Any attributes other than *Node* or *Shape* may be selected.

The third syntax shown above allows you to specify the maximum number of shapes to display before switching to the truncated display form of just showing the dimensions of the `imat_shp`. Setting this to 0 means that all of the shapes should always be displayed.

Examples

```
>> setdisplay(imat_shp,'frequency','modalmassreal','modalmassimag')
```

See Also

imat_shp/set

---

imat_shp/edit_attributes

Purpose

Convenient GUI for editing `imat_shp` data attributes.

Syntax

```
g=edit_attributes(f)
```

Description

EDIT_ATTRIUBTES is a convenience Graphical User Interface for editing `imat_shp` attributes. Most attributes are editable. The GUI groups related attributes for easy editing.

The input argument to EDIT_ATTRIBUTES is an `imat_shp`. It can be an array of functions. The output G is an `imat_shp` containing the modified `imat_shp`. If the user clicks on 'Cancel', then `g=-1` is returned.

See Also

imat_fn/edit_attributes, result/edit_attributes

---

imat_shp/alldof

Purpose

Get a trace of all degrees of freedom in a shape.

Syntax

```
t=alldof(s)
```

Description

ALLDOF returns a coordinate trace `t` containing all degrees of freedom in the `imat_shp` `s`. The coordinate trace will be in the order of the nodes in `s`. If `s` contains more than one shape, then the coordinate trace will be taken from `s(1)`.

Examples

```
>> s=readunv('/ms5/examples/tda/air_test_shapes.unv')
Universal file written in MM units, converting to SI
Read 10x1 imat_shp

s =
10x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   100.334             0.00777494          30
2   146.458             0.00738474          30
3   296.134             0.00306095          30
4   296.721             0.00344297          30
5   405.738             0.00250777          30
6   474.503             0.00189975          30
7   487.771             0.00192485          30
8   521.287             0.00290228          30
9   582.747             0.00147384          30
10  851.518             0.00138781          30

>> s(1).node'

ans =
  Columns 1 through 12
     1     2     3     4     5     6     7     8     9    10    11    12
  Columns 13 through 24
    13    14    15    16    17    18    19    20    21    22    23    24
  Columns 25 through 30
    25    26    27    28    29    30

>> s(1).doftype

ans =
     3DOF

>> t=alldof(s)

t =
    '1X+'
    '1Y+'
    '1Z+'
    '2X+'
    '2Y+'
    '2Z+'
    '3X+'
    '3Y+'
    '3Z+'
    '4X+'
    '4Y+'
    '4Z+'
    '5X+'
    '5Y+'
    '5Z+'
    '6X+'
```

```
        '6Y+'
        '6Z+'
...
        '28X+'
        '28Y+'
        '28Z+'
        '29X+'
        '29Y+'
        '29Z+'
        '30X+'
        '30Y+'
        '30Z+'

>>
```

See Also

imat_fn/allres

---

imat_shp/shp2fn

Purpose

Create an `imat_fn` from the supplied `imat_shp`.

Syntax

```
f=shp2fn(s)
f=shp2fn(s,ct)
```

Description

SHP2FN generates an `imat_fn` object from the supplied `imat_shp` object `s`. It will extract the shape coefficients for all of the shapes and assign the shape coefficients to a function record. One record will be generated for each degree of freedom in the shape. If the optional `imat_ctrace ct` is provided, it will create a function record for each coordinate in `ct` that is found in `s`. If inconsistent attributes are found, a warning will be issued but SHP2FN will continue. The ordinate numerator and denominator attributes from the first shape in `s` will be assigned to the `imat_fn`. The abscissa values will be set to the frequencies

After checking attributes for consistency, it will assign the ordinate value of each coordinate to the corresponding `imat_shp` for each abscissa point specified. The `imat_shp` frequency will be set to the corresponding abscissa value. The output `imat_shp` object will be an nx1 shape, where n is the number of abscissa values in the `imat_fn`.

Examples

```
>> s=imat_shp(3,'shape',reshape(1:18,6,3),'frequency',100:102);
>> list(s)

Shape 1 - 100.000000 Hz, 0.000000 %Cr damping
ID Line 1:

      Node       Disp-X       Disp-Y       Disp-Z
-------------------------------------------------
         1            1            2            3
         2            4            5            6

Shape 2 - 101.000000 Hz, 0.000000 %Cr damping
ID Line 1:

      Node       Disp-X       Disp-Y       Disp-Z
-------------------------------------------------
         1            7            8            9
         2            10           11           12

Shape 3 - 102.000000 Hz, 0.000000 %Cr damping
ID Line 1:

      Node       Disp-X       Disp-Y       Disp-Z
-------------------------------------------------
         1           13           14           15
         2           16           17           18

>> f=shp2fn(s,imat_ctrace('1z'))

f =

IMAT Function with the following attributes:
Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(,1Z+)                     General          Uneven           3

>> [f(1).abscissa f.ordinate]

ans =
     100     3
     101     9
     102    15

>> g=shp2fn(s)

g =

6x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(,1X+)                     General          Uneven           3
2_(,1Y+)                     General          Uneven           3
```

```
3_(,1Z+)                    General         Uneven          3
4_(,2X+)                    General         Uneven          3
5_(,2Y+)                    General         Uneven          3
6_(,2Z+)                    General         Uneven          3

>> [g(1).abscissa g.ordinate]

ans =
   100     1     2     3     4     5     6
   101     7     8     9    10    11    12
   102    13    14    15    16    17    18

>>
```

See Also

imat_fn/fn2shp

---

imat_shp/get_units_labels

Purpose

Get units labels for the supplied `imat_shp`.

Syntax

```
str=get_units_labels(s)
str=get_units_labels(s,'full')
```

Description

GET_UNITS_LABELS returns a cell array of strings the size of the input `imat_shp` S which contains the units label string for each of the shapes. The optional input string `'full'` specifies whether to use the units abbreviations or their full names. Leaving this argument off uses abbreviations.

Examples

```
>> s=imat_shp(1,'ordnumdatatype','acceleration');
>> setunits('in');
>> get_units_labels(s)

ans =
    'in/s^2'

>> get_units_labels(s,'full')

ans =
    'inch/second^2'

>>
```

See Also

imat_fn/get_units_labels, imat_result/get_units_labels

---

imat_shp/interp

Purpose

Interpolate the supplied `imat_shp`.

Syntax

```
t=interp(s,100)
t=interp(s,NPOINTS,SCALES,METHOD)
t=interp(s,'inc',INCREMENT)
t=interp(s,'inc',INCREMENT,SCALES,METHOD,EXTRAPTYPE)
```

Description

INTERP will generate linear- or log-spaced frequency values for the provided shapes, then pair each point with an interpolated shape coefficient. INTERP uses MATLAB's INTERP1 function to perform the interpolation.

NPOINTS is a number specifying how many frequency values to generate, or a vector of frequency values at which to interpolate. If the string `'values'` is supplied,then NPOINTS is always interpreted as frequency values. This is necessary if NPOINTS is a scalar frequency. If the string `'inc'` is supplied, then INCREMENT is a number specifying the spacing between consecutive frequencies.

SCALES is an optional string specifying whether the frequency/coefficient values should use linear or logarithmic scaling for the frequencies and shape coefficients, respectively. Possible values are `'lin'` (default), `'linlog'`, `'loglin'`, and `'loglog'`.

METHOD is an optional string, passed to INTERP1 to control the method of interpolation. Values accepted by INTERP1 include `'nearest'`, `'linear'`, `'spline'`, `'pchip'`, `'cubic'`, and `'v5cubic'`. The default is `'linear'`. An empty string specifies the default method.

EXTRAPTYPE is an optional numeric scalar, passed to INTERP1 to control the value to be used for extrapolated data. If specifying EXTRAPTYPE, you must also specify METHOD.

Examples

```
>> g=interp(f,'inc',0.05,'lin');
>> g=interp(f,100,'linlog');
>> g=interp(f,100,'loglog','cubic');
>> g=interp(f,100,'linlog','cubic',0);
>>
```

---

imat_shp/chgunits

Purpose

Convert an `imat_shp` to a different unit system.

Syntax

```
s1=chgunits(s,from,to)
```

Description

CHGUNITS converts an `imat_shp` to a different system of units. This affects the shape coefficients (*Shape* attribute), the modal mass (*ModalMassReal* and *ModalMassImag* attributes), and modal damping (*ModalDampingReal* and *ModalDampingImag* attributes).

The `from` argument is the unit system that `s` is expressed in currently. The `to` argument is the desired unit system (if `to` is omitted, the current unit system is assumed). The unit system arguments should either be unit strings (such as `'SI'`) or should be 5x1 vectors of unit conversion factors as returned by the getunits function.

If you follow the recommended practice of setting your units at the start of a session and maintaining consistency, you should not need to use this function.

Examples

```
>> s=chgunits(s,'mm','in');    % Change s from MM to IN
```

See Also

getunits, setunits

---

imat_shp/plot

Purpose

Plot an `imat_shp` in a special figure window.

Syntax

```
plot(s,fem)
[h,shpout]=plot(shp,fem,handle,-
form-
at,complexdisplay,'noundeformed','title',titlestr,'scale',scalefactor,'silent','unitslabel',unitslab)
```

Description

PLOT displays the mode shapes supplied in the `imat_shp` object SHP using the supplied IMAT_FEM object FEM. PLOT expects that FEM information is either an IMAT_FEM object, or as individual IMAT_NODE, IMAT_ELEM, IMAT_CS, and/or IMAT_TL objects. If node geometry is not in the global coordinate system, the coordinate system structure must be present so PLOT can transform the node coordinates. The input shape coefficients are assumed to be in the FEM's displacement coordinate system(s).

By default, both the undeformed and the deformed model will be displayed. The undeformed model will be drawn with `'.'` node markers and dashed lines (`'--'`) for the elements and tracelines. The deformed model will be displayed with `'.'` node markers and solid lines for the elements and tracelines. Colors are determined by the FEM entity colors.

HANDLE is a figure or panel handle. If a figure handle is supplied, the shape will be displayed on the supplied figure after deleting the contents of the figure. If a panel handle is supplied, the shape will be displayed on the supplied panel after deleting the contents of the panel. If an axis handle is supplied, PLOT will create a uipanel on top of the axis, using its OuterPosition for location and sizing. It will place the shape plot in the uipanel.

FORMATSTRING is a string containing valid point and line styles. These follow MATLAB PLOT() conventions, and will be applied to the deformed geometry.

Other valid display and format arguments and their descriptions are shown in the table below.

General Options	
`'noundeformed'`	Turn off undeformed display.
`'nodeformed'`	Turn off deformed display.
`'cycleshapes'`	When animating, cycle through each shape. This is useful for animating transient results.
`'title'`	Must be followed by a string containing the figure title.
`'unitslabel'`	This must be followed by a string containing the units label that will be placed in parentheses after the axis label. Pass in a space (' ') to disable the units label. Note that if `'unitslabel'` is not supplied, and the OrdinateUnitsLab of the supplied SHP is not empty, the label on the first shape will be used instead.
`'scale'`	Scale factor by which to multiply the default deformed display amplitude.
`'jpeg'`	Use Motion JPEG AVI compression for AVI file creation (default).
`'nocompression'`	Turn off compression for AVI file creation.
`'animate'`	Animate the shape as soon as the figure is created.
`'silent'`	Do not print progress messages when displaying the shape.
`'nogui'`	Removes all buttons on plot window.
`'noidlines'`	Removes ID Line information from plot area.
`'noaxes'`	Turns off the axis display
`'names'`	This must be followed by a cell array of strings the same length as the total number of shapes to be plotted. The strings will replace the 'Mode #' text in the upper left of the plot window. If a single string is passed in, it will be used for all of the modes.
`'colormag'`	Colors elements based on shape magnitude.
`'colormask'`	1x3 boolean vector to specify whether X, Y, and Z components are included in the color calculation. To color by magnitude (default), turn on all three.
`'staticcolor'`	Do not recalculate colors during animation.
`'loop'`	Logical specifying whether the shape animation should loop.Defaults to false for transient plots. Defaults to true for all others.

`'reversex'`	Reverse the X- and Z-axis directions. You cannot use more than one reverse axis option at a time.
`'reversey'`	Reverse the X- and Y-axis directions. You cannot use more than one reverse axis option at a time.
`'reversez'`	Reverse the Y- and Z-axis directions. You cannot use more than one reverse axis option at a time.
`'backgroundcolor',COLOR`	Set the panel's background color. You can pass in a string or a 1x3 numeric RGB vector.
`'textcolor',COLOR`	Set the title text and axis color. You can pass in a string or a 1x3 numeric RGB vector.
Complex Display Options	
`'real'`	Display the real portion of the complex mode shape.
`'imag'`	Display the imaginary portion of the complex mode shape.
`'amp'`	Display the shape coefficients as a signed amplitude (default).
`'complex'`	Display the shape coefficients as a complex shape. The static display is a signed amplitude, but the animation displays as complex (i.e. each node hits its maximum amplitude at different points in time).

Formatting arguments passed in to the PLOT command only apply to the deformed shape. You have much more control over the node, element, and traceline formats through the pulldown menus available on the figure.

Several pulldown menus will appear on the PLOT figure. In addition to the pulldowns drawn with imat_fem/plot, a pulldown menu called Shape will appear. This menu allows you to toggle the title display, change the complex shape display option, change the deformed geometry scale, edit the shape's ID lines, and change the animation speed. In addition, pushbuttons for starting and stopping animation and closing the figure appear at the bottom of the figure window.

You can also create an AVI animation of the current shape through the Shape pulldown menu. You have the option of compressing the image using the *Motion JPEG AVI* CODEC (the default) or creating an uncompressed file. To create an AVI file, select your compression option, then select Create AVI on Animate. The next time you press the Animate button, you will be prompted for an AVI filename. The first cycle of animation will create the AVI file. After the file has been created, animation will continue.

Please note that if you want to retrieve the updated `imat_shp` with modified ID lines, you need to request both output arguments. In this case PLOT will retain program control until you close the figure.

If you pass in an `imat_shp` with multiple shapes, three additional buttons will appear at the bottom of the figure. These allow you to cycle to the previous or next shape, or select a shape to plot.

Please note that performance is slow for large models. For large models, consider using VTKPLOT.

Examples

```
>> plot(s(1),fem)
Plotting undeformed shape
Plotting deformed shape
>> plot(s(1),fem,'real','*-.','silent')
>> plot(s(1),fem,'real','*-.','silent','scale',2)    % Double deformations
>> h=plot(fem);
>> plot(s(1),fem,h,'noundef','*-.','title','This is the title')
Plotting deformed shape
```

See Also

imat_fem/plot, imat_fem/labelnodes, imat_shp/vtkplot

---

imat_shp/vtkplot (+FEA)

Purpose

Plot and animate `imat_shp`and/or `result` in 3-D space

Syntax

```
plot(s,fem)
[h,shpout]=plot(shp,fem,group,res,handle,'title',titlestr,'scale',scale,options)
```

Description

VTKPLOT displays the shapes supplied in the `imat_shp` object SHP and/or the results stored in the result object RES using the supplied FEM. In general, the model's displacement is determined by the shape, and its contour by the result.

VTKPLOT expects that FEM information is an IMAT_FEM object. If node geometry is not in the global coordinate system, the coordinate system data must be present so PLOT can transform the node coordinates. The input shape coefficients are assumed to be in the FEM's displacement coordinate system(s).

By default, if a shape is present, only the deformed model will be displayed. The undeformed model can be displayed by using the 'Display' menu. The color is either determined by the result, if one was passed in, or by the displacement of the shape. The contour or shape component displayed can be changed using the 'Result' menu.

The Display menu control what is shown on the plot and how. The 'Complex Display' menu controls how complex data is displayed if the displacement shape is complex. Below this, the clipping menus allow you to set up clipping planes for the model. There are four available clipping planes: three aligned with each axis and a general one that is arbitrary. Each can be enabled using the corresponding 'Enable' menu item. To keep the result of the clipping plane, but remove the plane from view, set the 'Set Invisible' option. To flip the side of the plane that is not visible, select the 'Flip Clipping Direction' option. To further customize the view, you can use the 'Element Visibility' GUI to adjust what kinds of elements are shown on the plot. The 'Baseline Model' and 'Result Model' menus contain options for changing how nodes, elements and tracelines are formatted and can be used so set the transparency of the model.

The view menu allows you to snap the view to the difference defined planes. If the toolbar is visible, each of the view options have a corresponding toolbar icon. From this menu you can also use a 'Zoom Box' to zoom into a portion of the model, or turn on the 3D mode where red/blue 3D glasses can be used to view the model.

To export an image of the model or an animation of the mode shape, use the 'Export' menu. If you want to export all of the mode shapes using the current view, use Export->All Slides.

If displacement shapes are present, the 'Animation' menu controls the animation. Here you can set the scale of the animation, and set whether it will animate as a mode shape or animate as a transient result.

Several optional input arguments are supported:

GROUP is an IMAT_GROUP containing group information. If supplied, you will be able to display subsets of your FEM based on the elements in the group(s).

HANDLE is a figure or panel handle. If a figure handle is supplied, the shape will be displayed on the supplied figure after deleting the contents of the figure. If a panel handle is supplied, the shape will be displayed on the supplied panel after deleting the content of the panel.

OPTIONS is a string containing one of the following strings. Multiple options may be specified.

General Options	
`'undeformed'`	Undeformed geometry turned on.
`'noundeformed'`	Undeformed geometry turned off.
`'deformed'`	Deformed geometry turned on.
`'nodeformed'`	Deformed geometry turned off.
`'colorbar'`	Display the color bar.
`'nocolorbar'`	Do not display the color bar.
`'notoolbar'`	Do not display the toolbar.
`'nogui'`	Removes all buttons on plot window.
`'noidlines'`	Removes ID Line information from plot area.
`'cycleshapes'`	When animating, cycle through each shape. This is useful for animating transient results.
`'view'`	Set plot view. See imat_vtkplot/view for valid options.
`'interactionstyle'`	Set the interaction style. See imat_vtkplot/interactionstyle for more details.
`'title'`	This must be followed by a string, which will be the figure title.
`'scale'`	This must be followed by a number, which will be the scale factor by which to multiply the default deformed display amplitude. For example, a scale factor of 2.0 will double the default deformation.
`'names'`	This must be followed by a cell array of strings the same length as the total number of shapes to be plotted. The strings will replace the 'Mode #' text in the upper left of the plot window. If a single string is passed in, it will be used for all of the modes.
`'loop'`	Logical specifying whether the shape animation should loop.Defaults to false for transient plots. Defaults to true for all others.
`'nanimframes'`	Must be followed by a positive integer scalar specifying the number of animation frames.

`'silent'`	Suppress output to the Matlab window.
Complex Display Options	
`'real'`	Display the real portion of the complex mode shape.
`'imag'`	Display the imaginary portion of the complex mode shape.
`'amp'`	Display the shape coefficients as a signed amplitude (default).
`'complex'`	Display the shape coefficients as a complex shape. The static display is a signed amplitude, but the animation displays as complex (i.e. each node hits its maximum amplitude at different points in time).

The output argument for VTKPLOT is an IMAT_VTKPLOT object. It contains all of the formatting and contents of the plot. If IDLines are modified during the plotting session, and they need to be retrieved, place the handle to the figure into `uiwait`, and then retrieve the modified `imat_shp` object once the user is done editing it.

Examples

```
>> vtkplot(s(1),fem)
>> h = vtkplot(s(1),fem,h,'undeformed')
>> h = vtkplot(shape,fem,h,'title','Mode Shapes','imag','scale',2.3)
```

See Also

imat_fem/vtkplot, result/vtkplot

---

imat_shp/uiselect

Purpose

Select shapes using a graphical interface.

Syntax

```
g=uiselect(f)
g=uiselect(f,'Title')
g=uiselect(f,[presel])
g=uiselect(f,'adfsel')
g=uiselect(f,{'Attrib1','Attrib2','Attrib3'})
[g,shpind]=uiselect(f)
[g,shpind]=uiselect(f,'Title',[presel])
```

Description

The UISELECT function brings up a shape selection form similar to the IMAT Function selection form, listing all elements of the `imat_shp`. The user can select elements of `f`, then click on 'OK'. The selected elements are returned in the output argument `g`. If a title string is provided, it is used to name the function selection window. If a numeric vector is provided, the shapes corresponding to the indices in the vector will be preselected when the form is displayed. The optional second output argument `shpind` will contain the indices into `f` of the shapes selected and returned in `g`. The attributes listed in the attribute columns on the form will default to the attributes set by SETDISPLAY. Alternately, you can specify them by passing in a cell array of strings of attributes to display.

The optional input string `'adfsel'` will enable and display the ADF Selection button on the form. It is not displayed by default.

If the user clicks on 'Cancel', then `g=-1` is returned.

The shape selection window also provides a button named "ADF Selection". The "ADF Selection" button brings up a file dialog, and displays the shapes from the selected ADF to the shape selection form.

See Also

imat_fn/uiselect, imat_filt/uiselect, imat_ctrace/uiselect, imat_fn/setdisplay

---

imat_shp/uplus

Purpose

Unary plus (`+s`).

Syntax

```
+s
```

Description

This function does not affect an `imat_shp`. It is included for completeness only.

See Also

imat_shp/uminus

---

imat_shp/uminus

Purpose

Unary minus (`-s`).

Syntax

```
-s
```

Description

Taking the negative of an `imat_shp` object causes all shape coefficients to be replaced with their negatives.

Examples

```
>> for k=1:length(s)
    if -min(s(k).shape) > max(s(k).shape), s(k)=-s(k); end
end
>>
```

See Also

imat_shp/uplus

---

imat_shp/plus

Purpose

Add shape coefficients (`s1+s2`).

Syntax

```
s1+s2
s+x
x+s
```

Description

The following addition operations are possible with `imat_shp` objects:

- Adding two `imat_shp` objects causes their shape coordinates to be added. The shape node numbers must agree between the shapes being added. If one of the operands is a single shape (1x1), then its coefficients will be added to all elements in the other operand. If both operands are shape arrays, they must have the same dimensions. The data attributes of the result of an addition operation will be taken from the left operand. A warning will be issued if the ordinate attributes of the two operands are not consistent, but the operation is allowed.
- Adding a numeric array or scalar to an `imat_shp` causes the numeric values to be added to the shape coefficient values of the `imat_shp`.

Examples

```
>> s=imat_shp(3,'shape',reshape(1:18,6,3))

s =
3x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1                   0.01                2
2   1                   0.01                2
3   1                   0.01                2

>> s.shape

ans =
     1     7    13
     2     8    14
     3     9    15
     4    10    16
     5    11    17
     6    12    18

>> t=s+s; t.shape

ans =
     2    14    26
     4    16    28
     6    18    30
     8    20    32
    10    22    34
    12    24    36

>> t=t+t(1); t.shape

ans =
     4    16    28
     8    20    32
    12    24    36
    16    28    40
    20    32    44
    24    36    48

>> t=t+1; t.shape

ans =
     5    17    29
     9    21    33
    13    25    37
    17    29    41
    21    33    45
    25    37    49

>>
```

See Also

imat_shp/minus

---

imat_shp/minus

Purpose

Subtract shape coefficients (`s1-s2`).

Syntax

```
s1-s2
s-x
x-s
```

Description

The following subtraction operations are possible with `imat_shp` objects:

- Subtracting two `imat_shp` objects causes their shape coordinates to be subtracted. The shape node numbers must agree between the shapes being subtracted. If one of the operands is a single shape (1x1), then its coefficients will be subtracted from all elements in the other operand. If both operands are shape arrays, they must have the same dimensions. The data attributes of the result of an subtraction operation will be taken from the left operand. A warning will be issued if the ordinate attributes of the two operands are not consistent, but the operation is allowed.
- Subtracting a numeric array or scalar from an `imat_shp` causes the numeric values to be subtracted from the shape coefficient values of the `imat_shp`.
- Subtracting an `imat_shp` from a numeric array or scalar causes the ordinate values of the `imat_shp` to be subtracted from the numeric values.

Examples

```
>> s=imat_shp(3,'shape',reshape(1:18,6,3))

s =
3x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1                   0.01                2
2   1                   0.01                2
3   1                   0.01                2

>> s.shape

ans =
     1     7    13
     2     8    14
     3     9    15
     4    10    16
     5    11    17
     6    12    18

>> t=s-3;  t.shape

ans =
    -2     4    10
    -1     5    11
     0     6    12
     1     7    13
     2     8    14
     3     9    15

>> t=t-t(1);  t.shape

ans =
     0     6    12
     0     6    12
     0     6    12
     0     6    12
     0     6    12
     0     6    12

>> t=t-[-1 0 3];  t.shape

ans =
    -1     6    15
    -1     6    15
    -1     6    15
    -1     6    15
    -1     6    15
    -1     6    15

>>
```

See Also

imat_shp/plus

---

imat_shp/times

Purpose

Termwise multiply shape coefficients (`s1.*s2`).

Syntax

```
s1.*s2
s.*x
```

Description

The following termwise multiplication operations are possible with `imat_shp` objects:

- Multiplying two `imat_shp` objects causes their shape coefficients to be multiplied termwise. The shape node numbers must agree between the shapes being multiplied. If one of the operands is a single shape (1x1), then its coefficients will be multiplied by all elements in the other operand. If both operands are shape arrays, they must have the same dimensions. The data attributes of the result of a termwise multiplication will be taken from the left operand, except that ordinate data types (i.e., units exponents) will correctly reflect the combination of the two operands.
- Multiplying a numeric array or scalar times an `imat_shp` causes the numeric values to be termwise multiplied into the shape coefficient values of the `imat_shp`.

Examples

```
>> s=imat_shp(3,'shape',reshape(1:18,6,3))

s =
3x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1                   0.01                2
2   1                   0.01                2
3   1                   0.01                2

>> s.shape

ans =
     1     7    13
     2     8    14
     3     9    15
     4    10    16
     5    11    17
     6    12    18

>> t=s.*s; t.shape

ans =
     1    49   169
     4    64   196
     9    81   225
    16   100   256
    25   121   289
    36   144   324

>> t=t.*t(1); t.shape

ans =
           1          49         169
          16         256         784
          81         729        2025
         256        1600        4096
         625        3025        7225
        1296        5184       11664

>>
```

See Also

imat_shp/plus, imat_shp/minus, imat_shp/ldivide, imat_shp/rdivide

---

imat_shp/mtimes

Purpose

Matrix multiply shape coefficients (`A*s`).

Syntax

```
t=A*s
t=s*A
```

Description

Matrix multiplication between a numeric array `A` and an `imat_shp` object `s` is performed repeatedly at all shape coefficients. This type of operation requires that all elements of `s` have the same node numbers and DOF, and that the column dimension of the left operand matches the row dimension of the right operand. At each shape coefficient, a matrix or vector the same dimension as `s` is formed, and the matrix product is computed. The resulting matrix or vector is placed into the shape of the result, with the same nodes and DOF as `s`. The data attributes of the result are taken from s`(1)`. In particular, the units exponents will be set assuming that `A` is a unitless quantity.

Note that `A*s` does not give a shape equal to `A*(s.shape)`. In the first expression, `A` must have column dimension equal to the row dimension of `s`. In the second expression, `A` must have column dimension equal to the number of shape coefficients in `s`.

Examples

```
>> s=imat_shp(3,'shape',reshape(1:18,6,3));
>> s.frequency=(1:3)'

s =
3x1 IMAT Shape with the following attributes:
Row Frequency            Damping              NumberNodes
--- -------------------- -------------------- --------------------
1   1                    0.01                 2
2   2                    0.01                 2
3   3                    0.01                 2

>> s.shape                     % s is 3x1 with 6 shape coefficients

ans =
     1     7    13
     2     8    14
     3     9    15
     4    10    16
     5    11    17
     6    12    18

>> x=[1 -1 0 ; -1 0 1]         % x is 2x3 (ncols = nrows of s)

x =
     1    -1     0
    -1     0     1

>> t=x*s;   t.shape            % t is 2x1 with 6 shape coefficients

ans =
    -6    12
    -6    12
    -6    12
    -6    12
    -6    12
    -6    12

>> t                           % Note all outputs have attributes of t(1)

t =
2x1 IMAT Shape with the following attributes:
Row Frequency            Damping              NumberNodes
--- -------------------- -------------------- --------------------
1   1                    0.01                 2
2   1                    0.01                 2

>>
```

See Also

imat_shp/mldivide, imat_shp/mrdivide

---

imat_shp/rdivide

Purpose

Termwise divide shape coefficients (`s1./s2`).

Syntax

```
s1./s2
s./x
x./s
```

Description

The following termwise division operations are possible with `imat_shp` objects:

- Dividing two `imat_shp` objects causes their ordinates to be divided termwise. The shape node numbers must agree between the shapes being divided. If one of the operands is a single shape (1x1), then its coefficients will be used with all elements in the other operand. If both operands are shape arrays, they must have the same dimensions. The data attributes of the result of a termwise division will be taken from the left operand, except that ordinate data types (i.e., units exponents) will correctly reflect the combination of the two operands.
- Dividing a numeric array or scalar and an `imat_shp` causes the numeric values to be termwise divided into or divided by the shape coefficient values of the `imat_shp`.

Examples

```
>> s=imat_shp(3,'shape',reshape(1:18,6,3))

s =
3x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1                   0.01                2
2   1                   0.01                2
3   1                   0.01                2

>> s.shape

ans =
      1     7    13
      2     8    14
      3     9    15
      4    10    16
      5    11    17
      6    12    18

>> t=s+1; t.shape

ans =
      2     8    14
      3     9    15
      4    10    16
      5    11    17
      6    12    18
      7    13    19

>> u=t./s;   u.shape

ans =
    2.0000    1.1429    1.0769
    1.5000    1.1250    1.0714
    1.3333    1.1111    1.0667
    1.2500    1.1000    1.0625
    1.2000    1.0909    1.0588
    1.1667    1.0833    1.0556

>>
```

See Also

imat_shp/mtimes, imat_shp/ldivide

---

imat_shp/ldivide

Purpose

Termwise divide shape coefficients (`s1.\s2`).

Syntax

```
s1.\s2
s.\x
x.\s
```

Description

The following termwise multiplication operations are possible with `imat_shp` objects

- Dividing two `imat_shp` objects causes their ordinates to be divided termwise. The shape node numbers must agree between the shapes being divided. If one of the operands is a single shape (1x1), then its coefficients will be used with all elements in the other operand. If both operands are shape arrays, they must have the same dimensions. The data attributes of the result of a termwise division will be taken from the left operand, except that ordinate data types (i.e., units exponents) will correctly reflect the combination of the two operands.
- Dividing a numeric array or scalar and an `imat_shp` causes the numeric values to be termwise divided into or divided by the shape coefficient values of the `imat_shp`.

A termwise left division operation in MATLAB is the same as a right division with the operands switched.

Examples

```
>> s=imat_shp(3,'shape',reshape(1:18,6,3))

s =
3x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1                   0.01                2
2   1                   0.01                2
3   1                   0.01                2

>> s.shape

ans =
     1     7    13
     2     8    14
     3     9    15
     4    10    16
     5    11    17
     6    12    18

>> t=s+1;   t.shape

ans =
     2     8    14
     3     9    15
     4    10    16
     5    11    17
     6    12    18
     7    13    19

>> u=t.\s;   u.shape

ans =
    0.5000    0.8750    0.9286
    0.6667    0.8889    0.9333
    0.7500    0.9000    0.9375
    0.8000    0.9091    0.9412
    0.8333    0.9167    0.9444
    0.8571    0.9231    0.9474

>>
```

See Also

imat_shp/mtimes, imat_shp/rdivide

---

imat_shp/mrdivide

Purpose

Matrix divide shapes (`s/A`).

Syntax

```
t=s/A
```

Description

Matrix division between an `imat_shp` object `s` and a numeric array `A` is performed repeatedly at all shape coefficients. This type of operation requires that all elements of `s` have the same node numbers and DOF, and that the column dimension of `s` matches the column dimension of `A`. At each shape coefficient, a matrix or vector the same dimension as `s` is formed, and the matrix division is computed. The resulting matrix or vector is placed into the shape of the result, with the same nodes and DOF as `s`. The data attributes of the result are taken from s`(1)`. In particular, the units exponents will be set assuming that `A` is a unitless quantity.

Note that `s/A` does not give an ordinate equal to `(s.shape)/A`. In the first expression, `A` must have a column dimension equal to the row dimension of `s`. In the second expression, `A` must have a column dimension equal to the number of shape coefficients of `s`.

Examples

```
>> s=imat_shp(3,'shape',reshape(1:18,6,3));
>> s.frequency=(1:3)'

s =

3x1 IMAT Shape with the following attributes:
Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1                   0.01                2
2   2                   0.01                2
3   3                   0.01                2

>> s.shape                    % s is 3x1 with 6 shape coefficients

ans =
     1     7    13
     2     8    14
     3     9    15
     4    10    16
     5    11    17
     6    12    18

>> A=(1:3)'                   % A is 3x1 (ncols = ncols of s)

A =
     1
     2
     3

>> t=s/A; t.shape             % t is 3x3 with 6 shape coefficients

ans(:,:,1) =
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0

ans(:,:,2) =
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0

ans(:,:,3) =
    0.3333    2.3333    4.3333
    0.6667    2.6667    4.6667
    1.0000    3.0000    5.0000
    1.3333    3.3333    5.3333
    1.6667    3.6667    5.6667
    2.0000    4.0000    6.0000

>> t                         % Note all outputs have attributes of t(1)

t =

3x3 IMAT Shape with the following attributes:
Row Col Frequency           Damping             NumberNodes
--- --- ------------------- ------------------- -------------------
1   1   1                   0.01                2
2   1   1                   0.01                2
3   1   1                   0.01                2
1   2   1                   0.01                2
2   2   1                   0.01                2
3   2   1                   0.01                2
1   3   1                   0.01                2
2   3   1                   0.01                2
3   3   1                   0.01                2

>>
```

See Also

imat_fn/mldivide, imat_fn/mtimes

---

imat_shp/mldivide

Purpose

Matrix divide shapes (`A\s`).

Syntax

```
t=A\s
```

Description

Matrix left division between an `imat_shp` object `s` and a numeric array `A` is performed repeatedly at all shape coefficients. This type of operation requires that all elements of `s` have the same node numbers and DOF, and that the row dimension of `s` match the row dimension of `A`. At each shape coefficient, a matrix or vector the same dimension as `s` is formed, and the matrix division is computed. The resulting matrix or vector is placed into the shape of the result, with the same nodes and DOF as `s`. The data attributes of the result are taken from s`(1)`. In particular, the units exponents will be set assuming that `A` is a unitless quantity.

Note that `A\s` does not give an ordinate equal to `A\(s.shape)`. In the first expression, `A` must have a row dimension equal to the row dimension of `s`. In the second expression, `A` must have a row dimension equal to the number of shape coefficients of `s`.

Examples

```
>> s=imat_shp(3,'shape',reshape(1:18,6,3));
>> s.frequency=(1:3)'
```

```
s =
3x1 IMAT Shape with the following attributes:
Row Frequency            Damping              NumberNodes
--- ------------------- ------------------- -------------------
1   1                    0.01                 2
2   2                    0.01                 2
3   3                    0.01                 2

>> s.shape                    % s is 3x1 with 6 shape coefficients

ans =
     1     7    13
     2     8    14
     3     9    15
     4    10    16
     5    11    17
     6    12    18

>> A=(1:3)'                   % A is 3x1 (nrows = nrows of s)

A =
     1
     2
     3

>> t=A\s; t.shape             % t is 3x3 with 6 shape coefficients

ans =
    3.8571
    4.2857
    4.7143
    5.1429
    5.5714
    6.0000

>> t                          % Note all outputs have attributes of t(1)

t =
IMAT Shape with the following attributes:
Frequency            Damping              NumberNodes
------------------- ------------------- -------------------
1                    0.01                 2

>>
```

See Also

imat_fn/mrdivide, imat_fn/mtimes

---

imat_shp/real

Purpose

Take real part of shape coefficients.

Syntax

```
s1=real(s)
```

Description

This function replaces all shape coefficients of `s` with their real parts.

See Also

imat_shp/imag, imat_shp/conj

---

imat_shp/imag

Purpose

Take imaginary part of shape coefficients.

Syntax

```
s1=imag(s)
```

Description

This function replaces all shape coefficients with their imaginary parts.

See Also

imat_shp/real, imat_shp/conj

---

imat_shp/conj

Purpose

Take complex conjugate of shape coefficients.

Syntax

```
s1=conj(s)
```

Description

This function replaces all shape coefficients with their complex conjugate values.

See Also

imat_shp/real, imat_shp/imag

---

imat_shp/abs

Purpose

Take absolute value (or modulus) of shape coefficients.

Syntax

```
t=abs(s)
```

Description

This function replaces all shape coefficients of `s` with their absolute value (if `s` is real-valued) or modulus (if `s` is complex-valued). If `s` is complex-valued, its ShapeType attribute will be tagged as real.

See Also

imat_shp/real, imat_shp/imag, imat_shp/phase, imat_shp/phased

---

imat_shp/phase

Purpose

Replace shape coefficients with their phase angle in radians.

Syntax

```
t=phase(s)
```

Description

This function replaces all shape coefficients of `s` with the complex phase angle in radians. The phase angle will be in the range from -2*pi to 0.

See Also

imat_shp/abs, imat_shp/phased

imat_shp/phased

Purpose

Replace shape coefficients with their phase angle in degrees.

Syntax

```
t=phased(f)
```

Description

This function replaces all ordinate values of `f` with the complex phase angle in degrees. The phase angle will be in the range from -360 to 0.

See Also

imat_shp/abs, imat_shp/phase

---

imat_shp/sort

Purpose

Sort an `imat_shp` by frequency.

Syntax

```
t=sort(s)
[t,ind]=sort(s)
```

Description

SORT will sort an `imat_shp` object by frequency. `ind` is an optional second output argument containing an index vector such that `t=s(ind)`.

Example

```
>> s=imat_shp(3,'shape',reshape(1:18,6,3),'frequency',[103 101 102])

s =

3x1 IMAT Shape with the following attributes:

Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   103                 0                   2
2   101                 0                   2
3   102                 0                   2

>> [t,ind]=sort(s)
t =

3x1 IMAT Shape with the following attributes:

Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   101                 0                   2
2   102                 0                   2
3   103                 0                   2


ind =
       2
       3
       1

>>
```

---

imat_ctrace/build_shape

Purpose

Create an `imat_shp` from shape coefficients.

Syntax

```
s=build_shape(t,coeff)
```

Description

This function is useful for creating an `imat_shp` object when mode shape coefficients have been determined at certain coordinates. BUILD_SHAPE takes an `imat_ctrace` coordinate trace `t` and a numeric array `coeff` which is dimensioned `length(t)` by the number of modes. It builds an `imat_shp` object `s` with the given shape coefficients. Sign corrections are made for coordinates oriented in a minus direction.

The output shape `s` will have default attributes, and node labels including all nodes in the coordinate trace `t`. `s` will be a 3DOF shape if `t` contains only translational degrees of freedom; otherwise it will be a 6DOF shape. Any degrees of freedom not contained in the coordinate trace will be filled with zero shape coefficients. Blank coordinate directions in T will be converted to X+.

After calling BUILD_SHAPE, you can set other attributes of `s` such as frequency, damping, modal mass, etc.

Examples

```
>> t      % measurement coordinates

t =

    '101Y+'
    '101Z-'
    '64X+'

>> coeff  % shape coefficients at measured coord (5 modes)

coeff =

   -0.4326    0.2877    1.1892    0.1746   -0.5883
   -1.6656   -1.1465   -0.0376   -0.1867    2.1832
    0.1253    1.1909    0.3273    0.7258   -0.1364

>> shp=build_shape(t,coeff)    % build shape object

shp =

5x1 IMATShape with the following attributes:

Row Frequency           Damping             NumberNodes
--- ------------------- ------------------- -------------------
1   1                   0.01                2
2   1                   0.01                2
3   1                   0.01                2
4   1                   0.01                2
5   1                   0.01                2

>> s.node

ans =

       64    64    64    64    64
      101   101   101   101   101

>> s.shape

ans =

    0.1253    1.1909    0.3273    0.7258   -0.1364
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
   -0.4326    0.2877    1.1892    0.1746   -0.5883
    1.6656    1.1465    0.0376    0.1867   -2.1832

>>
```

See Also

imat_shp/imat_shp

---

imat_shp/list

Purpose

List the nodes and shape coefficients of an `imat_shp` object.

Syntax

```
list(s)
list(s,format)
list(s,nodelist)
list(s,nodelist,format)
data=list(s,...)
```

Description

When applied to an `imat_shp` object `s`, the LIST function lists the nodes and shape coefficients in tabular form. It will provide the shape number, frequency, damping, and IDLine 1 before listing the shape coefficients. It will loop through all shapes provided. If the number format is long and the shape has 6 DOF per node, or for Complex shapes with 6 DOF per node, the shape coefficients will be listed on two lines. The first line contains the translational DOF and the second line contains the rotational DOF.

An optional string argument may be supplied to override the default number format setting. Valid options are `'long'` and `'short'`. A node list may also be supplied. This must be a numeric vector. The subset of nodes matching nodes provided in the node list will be printed for each shape. If no nodes match, a warning message will be printed.

DATA is an optional output cell array the same size as S containing the nodes and shape coefficients listed for each shape.

Examples

```
>> s=imat_shp(1);
>> s.shape=[1:9]*(1+1/9);
>> s.node=100:100:300

s =
IMAT Shape with the following attributes:
Frequency            Damping              NumberNodes
-------------------- -------------------- --------------------
1                    0.01                 3

>> list(s)

Shape 1 - 1.000000 Hz, 1.000000 %Cr damping
ID Line 1:
      Node       Disp-X       Disp-Y       Disp-Z
--------------------------------------------------
       100       1.1111       2.2222       3.3333
       200       4.4444       5.5556       6.6667
       300       7.7778       8.8889           10

>> list(s,[100 300])

Shape 1 - 1.000000 Hz, 1.000000 %Cr damping
ID Line 1:

      Node       Disp-X       Disp-Y       Disp-Z
--------------------------------------------------
      100        1.1111       2.2222       3.3333
      300        7.7778       8.8889           10

>> format long
>> list(s)

Shape 1 - 1.000000 Hz, 1.000000 %Cr damping
ID Line 1:

      Node                 Disp-X                 Disp-Y                 Disp-Z
----------------------------------------------------------------------------------
       100       1.11111111111111       2.22222222222222       3.33333333333333
       200       4.44444444444444       5.55555555555556       6.66666666666667
       300       7.77777777777778       8.88888888888889                     10

>> s.doftype='6dof';
>> list(s)

Shape 1 - 1.000000 Hz, 1.000000 %Cr damping
ID Line 1:

      Node                 Disp-X                 Disp-Y                 Disp-Z
                            Rot-X                  Rot-Y                  Rot-Z
----------------------------------------------------------------------------------
       100       1.11111111111111       2.22222222222222       3.33333333333333
                                0                      0                      0
       200       4.44444444444444       5.55555555555556       6.66666666666667
                                0                      0                      0
       300       7.77777777777778       8.88888888888889                     10
                               0                       0                      0

>>
```

See Also

imat_shp

---

imat_shp/partition

Purpose

Partition the shape coefficients of an `imat_shp` object to a smaller set.

Syntax

```
t=partition(s,ctrace)
t=partition(s,nodelist)
```

Description

The PARTITION functions will partition the supplied `imat_shp` object `s` to the supplied DOF list. The DOF list may either be a list of coordinates supplied as an `imat_ctrace` object, or a list of nodes supllied as a numeric vector. All shape attributes of the resulting `imat_shp` object `t` will be copied from the supplied `imat_shp`. Only nodes supplied in `nodelist` that match nodes in the shape will be present in the resulting shape object. If coordinates in the supplied `ctrace` do not match coordinates in the supplied `imat_shp`, an error will occur.

The supplied shape `s` can contain multiple shapes of different lengths and attributes, but performance will be much greater if the `imat_shp` contains shapes that all have the same number of nodes and DOF per node.

Examples

```
>> s=imat_shp(1);
>> s.shape=[1:9]*(1+1/9);
>> s.node=100:100:300

s =
IMAT Shape with the following attributes:
Frequency           Damping             NumberNodes
------------------- ------------------- -------------------
1                   0.01                3

>> list(s)

Shape 1 - 1.000000 Hz, 1.000000 %Cr damping
ID Line 1:
      Node       Disp-X       Disp-Y       Disp-Z
--------------------------------------------------
       100       1.1111       2.2222       3.3333
       200       4.4444       5.5556       6.6667
       300       7.7778       8.8889           10

>> t=partition(s,[100 300]);    list(t)

Shape 1 - 1.000000 Hz, 1.000000 %Cr damping
ID Line 1:

      Node       Disp-X       Disp-Y       Disp-Z
--------------------------------------------------
       100       1.1111       2.2222       3.3333
       300       7.7778       8.8889           10

>> t=partition(s,imat_ctrace('100x','200y','300z'));   list(t)

Shape 1 - 1.000000 Hz, 1.000000 %Cr damping
ID Line 1:
      Node       Disp-X       Disp-Y       Disp-Z
--------------------------------------------------
       100       1.1111            0            0
       200            0       5.5556            0
       300            0            0           10

>>
```

See Also

imat_shp

---

imat_shp/xform

Purpose

Coordinate transform shape coefficients.

Syntax

```
shp2=xform(shp,fem)
shp2=xform(shp,fem,csto)
shp2=xform(shp,fem,csfrom,csto)
shp2=xform(shp,fem,'silent')
```

Description

XFORM will transform shape coefficients in the supplied `imat_shp` using coordinate system and node information provided in the IMAT_FEM object FEM. The `imat_shp` can be multi-dimensional and contain shapes of various sizes. Coordinate system transformations can be local to global, global to local, or local to local. Transformations can occur between any two arbitrary coordinate systems.

CSFROM and CSTO are coordinate system labels to transform from and to, respectively. They can either be a vector of the same length as the number of nodes in the shape, or a scalar. If CSFROM is not supplied, the coordinate system(s) to transform from will be extracted from the .CSType attribute of SHAPE. If this is set to `'Other'`, an error will occur. If CSTO is not supplied, the coordinate system(s) to transform to will be extracted from the second column (node displacement coordinate system) of the `.cs` field of the node structure. If the nodes are currently in local coordinates, XFORM assumes that you want to transform back to the global. Coordinate system 0 is defined as the global Cartesian coordinate system.

Passing in the string `'silent'` suppresses output to the screen.

When transforming coordinates, XFORM assumes that shape coefficients for coordinate systems defined in cylindrical and spherical coordinate systems are actually stored in Cartesian coordinates for a Cartesian system based on the node location in the cylindrical or spherical system. As a result it only applies a Cartesian transformation. The coordinate system offsets are also not applied. For cylindrical and spherical coordinate transforms, shape coefficient transformations for nodes located at a local origin may produce inaccurate results due to numerical roundoff.

XFORM will return an `imat_shp` the same size as the input `imat_shp` containing the transformed shape coefficients.

Examples

```
>> fem = readunv('simple_fem.unv');

>> fem.node.id             % Node labels

ans =
      100
      200
      300
      400
      500

>> fem.node.cs             % Nodal coordinate system definitions

ans =
     2     2     0
     2     2     0
     2     2     0
     2     2     0
     2     2     0

>> fem.cs.matrix(:,:,2)    % Transformation matrix for CS 2

ans =
    0.5000   -0.4330    0.7500
         0    0.8660    0.5000
   -0.8660   -0.2500    0.4330
    1.0000    2.0000    3.0000

>> shp = readadf('simple_shapes.ash');
>> list(shp)

Shape 1 - 1.100000 Hz, 1.000000 %Cr damping
ID Line 1:

      Node       Disp-X       Disp-Y       Disp-Z
-------------------------------------------------
       100            1            0            0
       200            0            1            0
       300            1            0            0
       400            1            0            0
       500            1            0            1

Shape 2 - 2.200000 Hz, 1.000000 %Cr damping
ID Line 1:

      Node       Disp-X       Disp-Y       Disp-Z
-------------------------------------------------
       100            1            2            3
       300            4            5            6

>> shp2 = xform(shp,fem);
Shape 1:Transforming GLOBAL->LOCAL
>> list(shp2)
```

```
Shape 1 - 1.100000 Hz, 1.000000 %Cr damping
ID Line 1:

      Node        Disp-X        Disp-Y        Disp-Z
-------------------------------------------------
       100           0.5             0      -0.86603
       200      -0.43301       0.86603         -0.25
       300           0.5             0      -0.86603
       400           0.5             0      -0.86603
       500          1.25           0.5      -0.43301

Shape 2 - 2.200000 Hz, 1.000000 %Cr damping
ID Line 1:

      Node        Disp-X        Disp-Y        Disp-Z
-------------------------------------------------
       100         1.884        3.2321     -0.066987
       300        4.3349        7.3301        -2.116
```

See Also

imat_fn/xform, xform

IMAT Methods and Functions for `imat_ctrace` objects

imat_ctrace	Create an `imat_ctrace` object
imat_ctrace/nd2ctrace	Build an `imat_ctrace` from an ID and direction list
imat_ctrace/get	Get one or more attributes of an `imat_ctrace` object
imat_ctrace/set	Set one or more attributes of an `imat_ctrace` object
imat_ctrace/plot	Plot coordinate trace arrows in 3-D space
imat_ctrace/vtkplot	Plot coordinate trace arrows in 3-D space using VTK
imat_ctrace/uiselect	Select coordinates using a graphical interface
imat_ctrace/length	Return number of elements in coordinate trace
imat_ctrace/node	Return node numbers of all coordinates
imat_ctrace/dir	Return direction strings of all coordinates

imat_ctrace/id	Return Nx5 array of node and direction characters
imat_ctrace/ismember	Return a vector indicating coordinates from one trace in another
imat_ctrace/cellstr	Convert coordinate trace into cell array of strings
imat_ctrace/char	Convert coordinate trace into array of strings
imat_ctrace/double	Convert coordinate trace into Nx2 numeric array
imat_ctrace/uplus	Unary plus of coordinate trace (`+t`)
imat_ctrace/uminus	Unary minus of coordinate trace (`-t`)
imat_ctrace/abs	Absolute value of coordinate trace
imat_ctrace/sign	Sign (+/-1) of coordinate trace
imat_ctrace/cat	Merge multiple coordinate traces
imat_ctrace/and	Return intersection of two coordinate traces (`s1&s2`)
imat_ctrace/or	Return union of two coordinate traces (`s1|s2`)
imat_ctrace/plus	Return union of two coordinate traces (`s1+s2`)
imat_ctrace/minus	Remove coordinates from a coordinate trace (`s1-s2`)
imat_ctrace/eq	Check whether two coordinate traces are identical (`s1==s2`)
imat_ctrace/ne	Check whether two coordinate traces are identical (`s1~=s2`)
imat_ctrace/sort	Sort a coordinate trace
imat_ctrace/unique	Eliminate duplicate coordinates
imat_ctrace/in_ctrace	Check which `imat_fn` elements match coordinate trace terms
imat_ctrace/build_shape	Build an `imat_shp` from shape coefficients
imat_ctrace/size	Return the size of the coordinate trace
imat_ctrace/union	Return a sorted union of two `imat_ctrace` with no repetitions

imat_ctrace/setdiff	Return a set difference of two `imat_ctrace` with no repetitions
imat_ctrace/setxor	Return a sorted union of two `imat_ctrace` with no repetitions
imat_ctrace/intersect	Return the coordinates common to two `imat_ctrace`

---

imat_ctrace/imat_ctrace

Purpose

Create an imat_ctrace object.

Syntax

```
t=imat_ctrace
t=imat_ctrace(node,dir)
t=imat_ctrace( [node dir] )
t=imat_ctrace( {'coord1','coord2',...} )
t=imat_ctrace('coord1','coord2',...)
t=imat_ctrace(s)
```

Description

The IMAT coordinate trace constructor function, creates a coordinate trace object from different types of input. An `imat_ctrace` has several attributes. These are the coordinate list ('Id'), the name ('Name'), a description of the coordinate trace ('Description'), and the coordinate trace internal structure version number ('Version'). The version number is intended for internal use only and should not be modified. See the imat_ctrace/get function for more information about the `imat_ctrace` attributes.

The first form, calling imat_ctrace with no arguments creates an empty coordinate trace.

The second form takes an nx1 numeric array of node numbers (`node`) and an nx1 cell array of strings (`dir`) containing the coordinate direction strings. One coordinate is created for each row of `node` and `dir`.

Alternatively, `dir` may be a numeric array containing numeric direction codes from the following table:

Numeric Code	Direction	Numeric Code	Direction
1	X+	-1	X-
2	Y+	-2	Y-
3	Z+	-3	Z-
4	RX+	-4	RX-
5	RY+	-5	RY-
6	RZ+	-6	RZ-

0	<null>		

The third form takes a single nx2 numeric array argument. The first column contains node numbers, and the second column contains numeric direction codes from the above table.

The fourth form takes a cell array of strings (either row or column orientation is OK). Each string should be a valid coordinate (i.e., node number followed by a direction string).

The fifth form allows individual coordinate strings to be provided directly to the constructor function.

The final form allows you to pass in a structure. The field names should either match `imat_ctrace` attribute names, or the structure should be in the form of the `imat_ctrace` internal structure (i.e. what you get from `struct(t)`). This form is useful, for example, if you extracted a structure using imat_ctrace/get and wanted to turn it back into an `imat_ctrace`.

Examples

```
>> t=imat_ctrace('5y', '6rx', '3z', '4x-')   % Fifth syntax

t =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> x=double(t)     % Gets nx2 numeric representation

x =

     5     2
     6     4
     3     3
     4    -1

>> y=sort(x)       % Don't ask me why we would want to do this!

y =

     3    -1
     4     2
     5     3
     6     4

>> t2=imat_ctrace(y)   % Syntax #3

t2 =

    '3X-'
    '4Y+'
    '5Z+'
    '6RX+'

>>
```

See Also

imat_ctrace/get, imat_ctrace/set

User's Guide

---

imat_ctrace/nd2ctrace

Purpose

Build an `imat_ctrace` from an ID and direction list.

Syntax

```
tf=nd2ctrace(ids,dirs)
```

Description

ND2CTRACE will construct an `imat_ctrace` from a vector of IDs (IDS) and directions (DIRS). It will create a coordinate for each combination of ID and direction.

ID must contain positive integers.

DIR can either be a list of integers, or a string or cell array of strings. If it contains integers, they must be in the range of -6 to 6, where 1 means X+, -3 means Z-, 0 means no direction, and so on.

Examples

```
>> t = nd2ctrace(1:10,1:3);

>> t = nd2ctrace(1:10,[-3 1 6]);

>> t = nd2ctrace(1:5,{'DIRA' 'DIRB'})

>>
```

---

imat_ctrace/get

Purpose

Get one or more attributes of an `imat_ctrace` object.

Syntax

```
v=get(t,'Attrib')
v=get(t,'Attrib1','Attrib2',...)
v=get(t)
```

Description

When applied to an `imat_ctrace` object `t`, the GET function returns the values of selected attributes of `t`. Attribute names are not case sensitive.

If only a single attribute is requested, then the value of that attribute is returned, or printed if no output argument is supplied. If more than one attribute is specified, then the output argument will be a scalar structure variable with field names equal to the requested attributes. The value of each field will be the contents of that attribute of the `imat_ctrace` object. If no output argument is supplied, the resulting structure will be printed on the standard output.

If no attributes are specified, then the values of *all* attributes will be returned as described above.

An alternative to the GET function is the syntax `t.attrib`.

Examples

```
>> t=imat_ctrace('1x','2x','3y')

t =
    '1X+'
    '2X+'
    '3Y+'

>> get(t)
         Id: [3x5 double]
       Name: ''
Description: ''
    Version: 2

>> v=get(t,'id')

v =
     1    88    43    32    32
     2    88    43    32    32
     3    89    43    32

>> get(t,'id','desc')
         Id: [3x5 double]
Description: ''

>>
```

See Also

imat_ctrace/set

---

imat_ctrace/set

Purpose

Set one or more attributes of an `imat_ctrace` object.

Syntax

```
set(t)
v=set(t,'Attrib1',value1,'Attrib2',value2,...)
t(1:2)=set(t(1:2),'attrib1',value1,...)
u=set(t,v)
```

Description

The SET function allows you to change any attribute of an `imat_ctrace` object.

If SET is called with no attribute arguments, then all attribute names and their possible values are printed to standard output. This functionality acts as a built-in help mechanism for data attributes. (Note that in this case, the argument `t` is not referenced, but simply causes MATLAB to call the SET function associated with `imat_ctrace` objects.)

If the reference to the input variable `t` includes subscripts, you must assign the output from SET as in the 3rd example above, otherwise MATLAB will issue an error.

To change attribute values, you must either supply pairs of attribute names and values (syntax 2), or you must supply a scalar structure variable (like the one returned by get) whose field names are the attribute names to be changed, and whose field values are the desired values (syntax 3).

The attribute names can be any valid attribute for the `imat_ctrace` object. Attribute values must be a single value, since an `imat_ctrace` object is not multidimensional (the coordinates are stored as a single nx5 double matrix in the object). This is consistent with the values returned by the get function.

The attributes are set in the order listed. If the same attribute is set more than once using SET, only the last applied change will hold.

An alternative way to set attributes is with the syntax *`t.attrib=value`*.

Examples

```
>> t=imat_ctrace('1x','2x','3y')

t =

    '1X+'
    '2X+'
    '3Y+'

>> set(t)
         Id:  [ nx5 double ]
       Name:  [ string ]
Description:  [ string ]
```

```
    Version:  [ integer value ]

>> u=set(t,'name','Copied ctrace')

Copied ctrace
u =

    '1X+'
    '2X+'
    '3Y+'

>> v=get(t)

v =

             Id: [3x5 double]
           Name: ''
    Description: ''
        Version: 2

>> v.Name='New name';
>> set(t,v); t

New name
t =

    '1X+'
    '2X+'
    '3Y+'

>>
```

See Also

imat_ctrace/get

---

imat_ctrace/plot

Purpose

Plot an `imat_ctrace` in a special figure window.

Syntax

```
plot(t,fem)
[hp,hl]=plot(t,fem,handle,format,sfact,color,'title',titlestr,'silent','doflabels','nodelabels')
```

Description

When you PLOT an `imat_ctrace`, the IMAT toolbox displays the `imat_ctrace`, `t`, in a figure. The finite element model must also be provided in a structure with the fields `node`, `elem`, `tl`, and `cs`, or as individual structures for nodes, elements, coordinate systems, and/or traceline. If node geometry is not in the global coordinate system, the coord inate system structure must be present so PLOT can transform the node coordinates. Several display and format options are provided, and are specified by passing in the appropriate string as an argument.

PLOT displays the FEM in a uipanel. If a figure or uipanel handle is provided as an input argument, and the handle already contains a FEM display, PLOT will display the coordinate trace in that figure/panel without plotting the supplied FEM. Otherwise, it will display the FEM and the `imat_ctrace`, creating a new figure if necessary. PLOT returns the uipanel handle in H.

The format string is a valid formatting string following MATLAB's builtin PLOT function conventions. The coordinate trace arrows can be scaled by passing in SFACT. SFACT is either a scalar or a vector of the same length as `t`. You can also change the color of the coordinate trace arrows by passing in a string containing a valid I-DEAS color.

Other valid display and format arguments and their descriptions are shown in the table below.

General Options	
`'title'`	Must be followed by a string containing the figure title.
`'nofem'`	Do not plot the FEM geometry; just plot the coordinate trace arrows.
`'doflabels'`	Display DOF direction labels of the DOF in the supplied coordinate trace.
`'nodelabels'`	Display node labels of the nodes in the supplied coordinate trace.
`'silent'`	Do not print progress messages when displaying the shape.

Formatting only applies to the coordinate trace arrows. To modify the formatting of the FEM, first plot the undeformed geometry using imat_fem/plot, then pass in the handle returned by imat_fem/plot to PLOT. Currently rotational DOF are plotted the same way as translational DOF.

Two optional output arguments are supported. HP is the handle of the uipanel in which the plot is displayed. HL contains the handles of the lines and patches that define the arrows.

This plotting facility is intended to provide simple viewing of an `imat_ctrace`. Performance is slow for large models. For large models, consider using VTKPLOT.

Examples

```
>> plot(t,fem)
>> plot(t,fem,':','silent')
>> h=plot(fem);
>> [hp,hl]=plot(t,fem,h,'nofem','*-.','title','This is the title')
```

See Also

imat_fem/plot, imat_shp/plot, imat_ctrace/vtkplot

---

imat_ctrace/vtkplot

Purpose

Plot coordinate trace arrows in 3-D space using VTK.

Syntax

```
vtkplot(ctrace,fem)
handle = vtkplot(ctrace,fem,handle,sfact,color,'title',titlestr,'silent','doflabels','nodelabels')
```

Description

VTKPLOT displays a coordinate trace on a FEM using VTK. Translational degrees of freedom are displayed with arrowhead tips, and rotational degrees of freedom are displayed with sphere tips.

CTRACE is an IMAT_CTRACE. FEM is an IMAT_FEM object from which the locations and directions of the coordinates in the CTRACE will be extracted. If node geometry is not in the global coordinate system, the coordinate system structure must be present so VTKPLOT can transform the node coordinates. VTKPLOT returns the handle to the IMAT_VTKPLOT object that allows you control the plot display.

HANDLE is the IMAT_VTKPLOT handle containing the display. If it is provided, the coordinate trace arrows will be placed on this display, using the supplied FEM to locate the arrows.

You can change the arrow color by passing in an optional input argument COLOR. If COLOR is a scalar string, all of the arrows will be plotted in that color. You can also specify individual colors for each arrow by passing in a cell array of strings or a vector of integers of `length(CTRACE)` corresponding to the I-deas color specifications. If COLOR is an Nx3 matrix, it is interpreted as an RGB matrix (range 0-1), where N is `length(CTRACE)`. If COLOR is not specified, the color of the FEM nodes referenced in CTRACE is used.

SFACT is a scale factor for scaling the arrow size. A scale factor of 1 corresponds to the default arrow size.

Other valid display and format arguments and their descriptions are shown in the table below.

General Options	
`'title'`	Must be followed by a string containing the figure title. If a title is not specified, the coordinate trace name and description will be used if they are set.
`'doflabels'`	Display DOF direction labels of the DOF in the supplied coordinate trace.
`'nodelabels'`	Display node labels of the nodes in the supplied coordinate trace.
`'silent'`	Do not print progress messages when displaying the shape.

Formatting only applies to the coordinate trace arrows. To modify the formatting of the FEM, first plot the undeformed geometry using imat_fem/plot, then pass in the handle returned by imat_fem/plot to PLOT. Currently rotational DOF are plotted the same way as translational DOF.

Two optional output arguments are supported. HP is the handle of the uipanel in which the plot is displayed. HL contains the handles of the lines and patches that define the arrows.

Examples

```
>> vtkplot(ctrace,fem)
>> h = vtkplot(t,fem,h,'blue','title','This is the title')
```

See Also

imat_fem/vtkplot, imat_vtkplot, ideas_colormap

---

imat_ctrace/uiselect

Purpose

Select coordinates using a graphical interface.

Syntax

```
g=uiselect(f)
g=uiselect(f,'Title')
g=uiselect(f,[presel])
[g,ctind]=uiselect(f)
[g,ctind]=uiselect(f,'Title',[presel])
```

Description

The UISELECT function brings up a coordinate trace selection form, listing all coordinates of the `imat_ctrace`. The user can select elements of `f`, then click on 'OK'. The selected elements are returned in the output argument `g`. If a title string is provided, it is used to name the function selection window. If a numeric vector is provided, the coordinates corresponding to the indices in the vector will be preselected when the form is displayed. The optional second output argument `ctind` will contain the indices into `f` of the coordinates selected and returned in `g`.

If the user clicks on 'Cancel', then `g=-1` is returned.

See Also

imat_fn/uiselect, imat_shp/uiselect, imat_filt/uiselect, imat_fn/setdisplay

---

imat_ctrace/length

Purpose

Return number of elements in coordinate trace.

Syntax

```
l=length(t)
```

Description

When applied to an `imat_ctrace` object, the LENGTH function returns the length of the coordinate trace (i.e., the number of coordinates).

Examples

```
>> t=imat_ctrace('1x','2x','3x','4x')

t =

    '1X+'
    '2X+'
    '3X+'
    '4X+'

>> length(t)

ans =

     4

>>
```

See Also

imat_ctrace/size

---

imat_ctrace/node

Purpose

Return node numbers of all coordinates.

Syntax

```
x=node(t)
```

Description

The NODE function returns an nx1 numeric array of node numbers for all coordinates in the coordinate trace `t`.

Examples

```
>> t=imat_ctrace('5y', '6rx', '3z', '4x-')

t =
    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> node(t)

ans =
     5
     6
     3
     4

>>
```

See Also

imat_ctrace/dir, imat_ctrace/double, imat_ctrace/cellstr, imat_ctrace/char

---

imat_ctrace/dir

Purpose

Return direction strings of all coordinates.

Syntax

```
d=dir(t)
```

Description

The DIR function takes an `imat_ctrace` object and returns an nx1 cell array of strings containing only the direction string for each coordinate. Most often these strings will be of the form `'X+'`, `'RZ-'`, etc., but in general can be any 4-character uppercase string.

Examples

```
>> t=imat_ctrace('5y', '6rx', '3z', '4x-')

t =
    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> d=dir(t)

d =
    'Y+'
    'RX+'
    'Z+'
    'X-'

>> whos
Name      Size          Bytes  Class
d         4x1             386  cell array
t         4x1             284  imat_ctrace object
Grand total is 34 elements using 670 bytes

>>
```

See Also

imat_ctrace/imat_ctrace, imat_ctrace/node, imat_ctrace/id, imat_num2dir, imat_dir2num

---

imat_ctrace/id

Purpose

Return Nx5 array of node and direction characters.

Syntax

```
x=id(t)
```

Description

The ID utility function converts a coordinate trace `t` into an nx5 matrix of coordinate identifiers suitable for row sorting and set operations like SETDIFF, etc. The first column of the output matrix is the node number, and columns 2 through 5 are ASCII codes of the direction field.

Examples

```
>> t=imat_ctrace('5y', '6rx', '3z', '4x-')

t =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> id(t)

ans =
     5    89    43    32    32
     6    82    88    43    32
     3    90    43    32    32
     4    88    45    32    32
>>
```

See Also

imat_ctrace/node, imat_ctrace/dir, imat_ctrace/cellstr, imat_ctrace/char, imat_ctrace/double

---

imat_ctrace/ismember

Purpose

Find `imat_ctrace` coordinates in another `imat_ctrace`.

Syntax

```
tf=ismember(a,s)
[tf,loc]=ismember(a,s)
```

Description

ISMEMBER for the `imat_ctrace` `A` returns a vector `TF` of the same length as `A` containing 1 where the coordinates of `A` are in the `imat_ctrace` `S` and 0 otherwise. LOC is an optional output index array containing the highest absolute index in `S` for each element in `A` which is a member of `S` and 0 if there is no such index.

Starting in MATLAB R2013a, the behavior of ISMEMBER has changed. To preserve the behavior of previous versions of MATLAB, pass in the string `'legacy'`.

Examples

```
>> a=imat_ctrace('1x','2y','3x');
>> s=imat_ctrace('1x','1y','1z','2x','2y','2z','3x','3y');
>> [tf,loc]=ismember(a,s)

tf =
     1
     1
     1

loc =
     1
     5
     7

>> [tf,loc]=ismember(s,a)

tf =
     1
     0
     0
     0
     1
     0
     1
     0

loc =
     1
     0
     0
     0
     2
     0
     3
     0

>>
```

See Also

imat_ctrace/intersect, imat_ctrace/setdiff, imat_ctrace/setxor, imat_ctrace/union, imat_ctrace/unique

---

imat_ctrace/cellstr

Purpose

Convert coordinate trace into cell array of strings.

Syntax

```
c=cellstr(t)
```

Description

When applied to an `imat_ctrace` object, the CELLSTR function converts a coordinate trace `t` into a cell array of strings with one string entry per coordinate. A cell array of this type can be converted back into a coordinate trace by passing it to the `imat_ctrace` function.

Examples

```
>> t=imat_ctrace('5y', '6rx', '3z', '4x-')

t =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> c=cellstr(t)

c =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> whos

Name      Size          Bytes  Class
c         4x1             394  cell array
t         4x1             284  imat_ctrace object
Grand total is 38 elements using 678 bytes

>> c{2}

ans =

6RX+

>>
```

See Also

imat_ctrace/imat_ctrace, imat_ctrace/char

---

imat_ctrace/char

Purpose

Convert coordinate trace into array of strings.

Syntax

```
s=char(t)
```

Description

When applied to an `imat_ctrace` object, the CHAR function converts a coordinate trace `t` into an array of strings. Each row of the string array corresponds to one coordinate in `t`. All coordinate strings are padded with spaces to make the result a rectangular array.

Examples

```
>> t=imat_ctrace('5y', '6rx', '3z', '4x-')

t =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> s=char(t)

s =

5Y+
6RX+
3Z+
4X-

>> whos

Name      Size          Bytes  Class

s         4x4              32  char array
t         4x1             284  imat_ctrace object

Grand total is 37 elements using 316 bytes

>>
```

See Also

imat_ctrace/imat_ctrace, imat_ctrace/cellstr

---

imat_ctrace/double

Purpose

Convert coordinate trace into Nx2 numeric array.

Syntax

```
x=double(t)
```

Description

When given an `imat_ctrace` object, the DOUBLE function converts a coordinate trace `t` into an nx2 numeric array. The first column of the result contains the node number, and the second column of the result contains a numeric code representing the coordinate direction:

Numeric code	*Direction*	*Numeric code*	*Direction*
1	X+	-1	X-
2	Y+	-2	Y-
3	Z+	-3	Z-
4	RX+	-4	RX-
5	RY+	-5	RY-
6	RZ+	-6	RZ-
0	any other		

A numeric array of this type can be used to create a coordinate trace by passing it to the `imat_ctrace` function.

Examples

```
>> t=imat_ctrace('5y', '6rx', '3z', '4x-')

t =
    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> double(t)

ans =
     5     2
     6     4
     3     3
     4    -1

>>
```

See Also

imat_ctrace/imat_ctrace, imat_ctrace/node, imat_ctrace/dir, imat_dir2num, imat_num2dir

---

imat_ctrace/uplus

Purpose

Unary plus of coordinate trace (+t).

Syntax

```
+t
```

Description

This operation does not change a coordinate trace in any way; it is included only for completeness.

See Also

imat_ctrace/uminus

---

imat_ctrace/uminus

Purpose

Unary minus of coordinate trace.

Syntax

```
-t
uminus(t)
```

Description

The unary minus of a coordinate trace changes all plus coordinate directions to minus, and vice versa.

Examples

```
>> t=imat_ctrace('5y', '6rx', '3z', '4x-')

t =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> -t

ans =

    '5Y-'
    '6RX-'
    '3Z-'
    '4X+'

>>
```

See Also

imat_ctrace/abs, imat_ctrace/sign

---

imat_ctrace/abs

Purpose

Absolute value of coordinate trace.

Syntax

```
u=abs(t)
```

Description

Taking the absolute value of a coordinate trace changes all coordinates with negative directions to their positive counterparts.

Examples

```
>> t=imat_ctrace('101x-', '55z', '75y-')

t =

    '101X-'
    '55Z+'
    '75Y-'

>> abs(t)

ans =

    '101X+'
    '55Z+'
    '75Y+'
```

See Also

imat_ctrace/sign, imat_ctrace/uminus

---

imat_ctrace/sign

Purpose

Sign (+/- 1) of coordinate trace.

Syntax

```
u=sign(t)
```

Description

When applied to an `imat_ctrace` object, the SIGN function returns the directional sign (+/- 1) of the coordinate trace. The output argument is an nx1 numeric array.

Examples

```
>> t=imat_ctrace('5y', '6rx', '3z', '4x-')

t =
    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> sign(t)

ans =
     1
     1
     1
    -1

>>
```

See Also

imat_ctrace/abs, imat_ctrace/uminus

---

imat_ctrace/cat

Purpose

Merge multiple coordinate traces.

Syntax

```
u=cat(1,t1,t2,...)
```

Description

The `cat` function concatenates coordinate traces without attempting to remove duplicates. The first field is ignored, but is provided to be compatible with the builtin `cat` function. The resultant coordinate trace will return the name of the first coordinate trace passed in.

Examples

```
>> t1=imat_ctrace('5y', '6rx', '3z', '4x-')

t1 =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> t2=imat_ctrace('4x', '3z', '2z-', '5y', '8x+')

t2 =

    '4X+'
    '3Z+'
    '2Z-'
    '5Y+'
    '8X+'

>> t=cat(1,t1,t2)

t =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'
    '4X+'
    '3Z+'
    '2Z-'
    '5Y+'
    '8X+'

>>
```

See Also

imat_ctrace/and, imat_ctrace/or, imat_ctrace/unique

---

imat_ctrace/and

Purpose

Return intersection of two coordinate traces.

Syntax

```
t=t1&t2
```

```
[t,i1,i2]=and(t1,t2)
```

Description

AND returns intersection of two coordinate traces. The resulting coordinate trace will have all coordinates which are included in both arguments, in sorted order. (Note that positive and negative directions are considered distinct.) The resultant coordinate trace will retain the name of the first coordinate trace passed in.

The second format also returns index arrays `i1` and `i2` such that `t=t1(i1)=t2(i2)`. This form can be useful when the order of the coordinates in the original arrays is important.

Examples

```
>> t1=imat_ctrace('5y', '6rx', '3z', '4x-')

t1 =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> t2=imat_ctrace('4x', '3z', '2z-', '5y', '8x+')

t2 =

    '4X+'
    '3Z+'
    '2Z-'
    '5Y+'
    '8X+'

>> t1&t2

ans =

    '3Z+'
    '5Y+'

>>
```

See Also

imat_ctrace/or, imat_ctrace/unique

---

imat_ctrace/or

Purpose

Return union of two traces (t1|t2).

Syntax

```
t1 | t2
or(t1,t2)
```

Description

When operating on two coordinate traces, the OR function returns the union (i.e., all unique coordinates in sorted order). The resultant coordinate trace will retain the name of the first coordinate trace passed in.

Examples

```
>> t1=imat_ctrace('5y', '6rx', '3z', '4x-')

t1 =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> t2=imat_ctrace('4x', '3z', '2z-', '5y', '8x+')

t2 =

    '4X+'
    '3Z+'
    '2Z-'
    '5Y+'
    '8X+'

>> t=t1|t2

t =

    '2Z-'
    '3Z+'
    '4X+'
    '4X-'
    '5Y+'
    '6RX+'
    '8X+'

>>
```

See Also

imat_ctrace/and

imat_ctrace/plus

Purpose

Return union of two traces (t1+t2).

Syntax

```
t1 + t2
plus(t1,t2)
```

Description

When operating on two coordinate traces, the PLUS function returns the union (i.e., all unique coordinates in sorted order).

Examples

```
>> t1=imat_ctrace('5y', '6rx', '3z', '4x-')

t1 =
    '5Y
    '6RX+'
    '3Z+'
    '4X-'

>> t2=imat_ctrace('4x', '3z', '2z-', '5y', '8x+')

t2 =
    '4X+'
    '3Z+'
    '2Z-'
    '5Y+'
    '8X+'

>> t=t1+t2

t =
    '2Z-'
    '3Z+'
    '4X+'
    '4X-'
    '5Y+'
    '6RX+'
    '8X+'

>>
```

See Also

imat_ctrace/minus

---

imat_ctrace/minus

Purpose

Subtract coordinate traces (t1-t2).

Syntax

```
t1 - t2
[t,i]=minus(t1,t2)
```

Description

When operating on two coordinate traces, the MINUS function removes any matching coordinates of the second argument from the first. The resulting coordinate trace is sorted. If a second output argument is provided, then an index array `i` is returned such that `t=t1(i)`.

Examples

```
>> t1=imat_ctrace('5y', '6rx', '3z', '4x-')

t1 = 

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> t2=imat_ctrace('4x', '3z', '2z-', '5y', '8x+')

t2 = 

    '4X+'
    '3Z+'
    '2Z-'
    '5Y+'
    '8X+'

>> t2-t1

ans = 

    '2Z-'
    '4X+'
    '8X+'

>> [t,i]=minus(t2,t1);

>> i

i =

     3
     1
     5

>>
```

See Also

imat_ctrace/plus

---

imat_ctrace/eq

Purpose

Check whether two coordinate traces are identical.

Syntax

```
t1==t2
eq(t1,t2)
```

Description

This function returns a logical true (1) if the two coordinate traces are equal, and false (0) otherwise. Two coordinate traces are equal if all coordinates and directions (including signs) are identical. The order of coordinates in the coordinate traces is significant.

Examples

```
>> t1=imat_ctrace('5y', '6rx', '3z', '4x-')

t1 =
    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> t2=imat_ctrace('4x', '3z', '2z-', '5y', '8x+')

t2 =
    '4X+'
    '3Z+'
    '2Z-'
    '5Y+'
    '8X+'

>> if t1(1)==t2(4), disp('Equal'); end
Equal

>>
```

See Also

imat_ctrace/ne

---

imat_ctrace/ne

Purpose

Check whether two traces are different.

Syntax

```
t1~=t2
ne(t1,t2)
```

Description

This function returns a logical false (0) if the two coordinate traces are equal, and true (1) otherwise. Two coordinate traces are equal if all coordinates and directions (including signs) are identical. The order of coordinates in the coordinate traces is significant.

Examples

```
>> t1=imat_ctrace('5y', '6rx', '3z', '4x-')

t1 =

    '5Y+'
    '6RX+'
    '3Z+'
    '4X-'

>> t2=imat_ctrace('4x', '3z', '2z-', '5y', '8x+')

t2 =

    '4X+'
    '3Z+'
    '2Z-'
    '5Y+'
    '8X+'

>> if t1(1)~=t2(4), disp('Unequal') else disp('Equal'); end
Equal

>>
```

See Also

imat_ctrace/eq

---

imat_ctrace/sort

Purpose

Sort a coordinate trace.

Syntax

```
[tsort,i]=sort(t)
```

Description

When applied to an `imat_ctrace` object, the SORT function sorts a coordinate trace in order of node, then degree of freedom. The (optional) second output argument is an index array such that `tsort=t(i)`.

Rotational degrees of freedom are sorted to follow the translational degrees of freedom. Repeated coordinates are not eliminated, but pluscoordinates sort before minus coordinates if the coordinates match in other respects.

Examples

```
>> t=imat_ctrace('5y', '3rx', '3z', '4x-', '3z-')

t =

    '5Y+'
    '3RX+'
    '3Z+'
    '4X-'
    '3Z-'

>> sort(t)

ans =

    '3Z+'
    '3Z-'
    '3RX+'
    '4X-'
    '5Y+'

>> [ts,i]=sort(t);   i


i =

     3
     5
     2
     4
     1

>>
```

See Also

imat_ctrace/unique

---

imat_ctrace/unique

Purpose

Sort coordinate trace and remove repeated coordinates.

Syntax

```
[tsort,i,q]=unique(t)
```

Description

When applied to an `imat_ctrace` object, the UNIQUE function sorts a coordinate trace in order of node, then degree of freedom, and eliminates duplicate coordinates. Plus and minus coordinates are considered different.

The (optional) second output argument is an index array such that `tsort=t(i)`.

The (optional) third output argument is a logical array Q of size T containing true for the selected unique coordinates.

Rotational degrees of freedom are sorted to follow the translational degrees of freedom. Repeated coordinates are not eliminated, but plus coordinates sort before minus coordinates if the coordinates match in other respects.

Starting in MATLAB R2013a, the behavior of UNIQUE has changed. To preserve the behavior of previous versions of MATLAB, pass in the string `'legacy'`.

Examples

```
>> t=imat_ctrace('5y', '3rx', '3z', '4x-', '3z', '5y+')

t =

    '5Y+'
    '3RX+'
    '3Z+'
    '4X-'
    '3Z+'
    '5Y+'

>> unique(t)

ans =

    '3Z+'
    '3RX+'
    '4X-'
    '5Y+'

>> [ts,i]=unique(t);    i

i =

     5
     2
     4
     6

>>
```

See Also

imat_ctrace/sort

---

imat_ctrace/in_ctrace

Purpose

Check which `imat_fn` elements match trace.

Syntax

```
v=in_ctrace( t, f, 'ref' )
v=in_ctrace( t, f, 'res' )
```

Description

The IN_CTRACE utility function creates a logical array `v` which describes which function elements in `f` have reference or response coordinate labels matching elements of the coordinate trace `t`. The output array is dimensioned `[prod(size(f)) length(t)]`. (Thus, even if `f` is multidimensional it is treated as a column vector.)

If the third argument is `'ref'`, then the output array `v` is defined so that `v(m,n)` is true (1) if the reference coordinate of the m'th element of `f` is equal to the n'th element of the coordinate trace `t`. If the third argument is `'res'`, then the logical values are based on the response coordinates of `f`.

If the third argument is omitted, `'res'` is assumed.

Note that `any(in_ctrace(t,f,...))` returns a row vector which could be used to select those elements of `f` that are in the coordinate trace `t`.

Examples

```
>> f=readadf('sample.ati')

f =

3x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Time Response    Even             1601
2_(1X+,2X+)                  Time Response    Even             1601
3_(1X+,3X+)                  Time Response    Even             1601

>> t=imat_ctrace('1x','2x','3x','4x')

t =

    '1X+'
    '2X+'
    '3X+'
    '4X+'

>> in_ctrace(t,f,'res')   % Match response coordinates

ans =

      1     0     0     0
      0     1     0     0
      0     0     1     0

>> in_ctrace(t,f,'ref')   % Match reference coordinates

ans =

      1     0     0     0
      1     0     0     0
      1     0     0     0

>>
```

See Also

imat_fn/allref, imat_fn/allres

---

imat_ctrace/build_shape

Purpose

Create an `imat_shp` from shape coefficients.

Syntax

```
s=build_shape(t,coeff)
```

Description

This function is useful for creating an `imat_shp` object when mode shape coefficients have been determined at certain coordinates. BUILD_SHAPE takes an `imat_ctrace` coordinate trace `t` and a numeric array `coeff` which is dimensioned `length(t)` by the number of modes. It builds an `imat_shp` object `s` with the given shape coefficients. Sign corrections are made for coordinates oriented in a minus direction.

The output shape `s` will have default attributes, and node labels including all nodes in the coordinate trace `t`. `s` will be a 3DOF shape if `t` contains only translational degrees of freedom; otherwise it will be a 6DOF shape. Any degrees of freedom not contained in the coordinate trace will be filled with zero shape coefficients. Blank coordinate directions in T will be converted to X+.

After calling BUILD_SHAPE, you can set other attributes of `s` such as frequency, damping, modal mass, etc.

Examples

```
>> t      % measurement coordinates

t =

    '101Y+'
    '101Z-'
    '64X+'

>> coeff  % shape coefficients at measured coord (5 modes)

coeff =

   -0.4326    0.2877    1.1892    0.1746   -0.5883
   -1.6656   -1.1465   -0.0376   -0.1867    2.1832
    0.1253    1.1909    0.3273    0.7258   -0.1364

>> shp=build_shape(t,coeff)    % build shape object

shp =

5x1 IMATShape with the following attributes:

Row Frequency            Damping              NumberNodes
--- -------------------- -------------------- --------------------
1   1                    0.01                 2
2   1                    0.01                 2
3   1                    0.01                 2
4   1                    0.01                 2
5   1                    0.01                 2

>> s.node

ans =

       64    64    64    64    64
      101   101   101   101   101

>> s.shape

ans =

    0.1253    1.1909    0.3273    0.7258   -0.1364
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
   -0.4326    0.2877    1.1892    0.1746   -0.5883
    1.6656    1.1465    0.0376    0.1867   -2.1832

>>
```

See Also

imat_shp/imat_shp

---

imat_ctrace/size

Purpose

Return the size of a coordinate trace.

Syntax

```
l=size(t)
```

Description

When applied to an `imat_ctrace` object, the SIZE function returns the size of the coordinate trace (i.e., the number of coordinates).

Examples

```
>> t=imat_ctrace('1x','2x','3x','4x')

t =
    '1X+'
    '2X+'
    '3X+'
    '4X+'

>> size(t)

ans =
     4     1

>>
```

See Also

imat_ctrace/length

---

imat_ctrace/union

Purpose

Return a sorted union of two `imat_ctrace` with no repetitions.

Syntax

```
tf=union(ta,tb)
[tf,ia,ib]=union(ta,tb)
```

Description

UNION for the `imat_ctrace` `TA` and `TB` returns an `imat_ctrace` `TF` containing the sorted combination of `TA` and `TB` with no repetitions. IA and IB are optional vector outputs such that `TF` is a sorted combination of `TA(IA)` and `TB(IB)`.

Starting in MATLAB R2013a, the behavior of UNION has changed. To preserve the behavior of previous versions of MATLAB, pass in the string `'legacy'`.

Examples

```
>> ta=imat_ctrace('1x','2y','3x');
>> tb=imat_ctrace('1x','1y','1z','2x','2y','2z','3x','3y');
>> [tf,ia,ib]=union(ta,tb)

tf =

    '1X+'
    '1Y+'
    '1Z+'
    '2X+'
    '2Y+'
    '2Z+'
    '3X+'
    '3Y+'

ia =

Empty matrix: 0-by-1

ib =

    1
    2
    3
    4
    5
    6
    7
    8

>>
```

See Also

imat_ctrace/intersect, imat_ctrace/ismember, imat_ctrace/setdiff, imat_ctrace/setxor, imat_ctrace/unique

---

imat_ctrace/setdiff

Purpose

Return a set difference of two `imat_ctrace` with no repetitions.

Syntax

```
tf=setdiff(ta,tb)
[tf,ia]=setdiff(ta,tb)
```

Description

SETDIFF for the `imat_ctrace`s `TA` and `TB` returns an `imat_ctrace` `TF` containing the coordinates in `TA` that are not in `TB`. `IA` is an optional vector output such that `TF=TA(IA)`.

Starting in MATLAB R2013a, the behavior of SETDIFF has changed. To preserve the behavior of previous versions of MATLAB, pass in the string `'legacy'`.

Examples

```
>> ta=imat_ctrace('1x','2y','3x');
>> tb=imat_ctrace('1x','1y','1z','2x','2y','2z','3x','3y');
>> setdiff(ta,tb)

ans =

empty IMAT coordinate trace

>> [t,ind]=setdiff(tb,ta)

t =

'1Y+'
'1Z+'
'2X+'
'2Z+'
'3Y+'

ind =

2
3
4
6
8

>>
```

See Also

imat_ctrace/intersect, imat_ctrace/ismember, imat_ctrace/setxor, imat_ctrace/union, imat_ctrace/unique

---

imat_ctrace/setxor

Purpose

Return a sorted union of two `imat_ctrace` with no repetitions.

Syntax

```
tf=setxor(ta,tb)
[tf,ia,ib]=setxor(ta,tb)
```

Description

SETXOR for the `imat_ctrace`s `TA` and `TB` returns an `imat_ctrace` `TF` containing the coordinates that are not in the intersection of `TA` that are not in `TB`. `IA` and `IB` are index vectors such that `TF` is the sorted combination of `TA(IA)` and `TB(IB)`.

Starting in MATLAB R2013a, the behavior of SETXOR has changed. To preserve the behavior of previous versions of MATLAB, pass in the string `'legacy'`.

Examples

```
>> ta=imat_ctrace('1x','2y','3x');
>> tb=imat_ctrace('1x','1y','1z','2x','2y','2z','3x','3y');
>> [t,ia,ib]=setxor(ta,tb)

t =

    '1Y+'
    '1Z+'
    '2X+'
    '2Z+'
    '3Y+'

ia =

Empty matrix: 0-by-1

ib =

    2
    3
    4
    6
    8

>>
```

See Also

imat_ctrace/intersect, imat_ctrace/ismember, imat_ctrace/setdiff, imat_ctrace/union, imat_ctrace/unique

---

imat_ctrace/intersect

Purpose

Return the coordinates common to two `imat_ctrace`.

Syntax

```
tf=intersect(ta,tb)
[tf,ia,ib]=intersect(ta,tb)
```

Description

INTERSECT for the `imat_ctrace TA` and `TB` returns an `imat_ctrace TF` containing the coordinates common to both `TA` and `TB`. IA and IB are optional vector outputs such that `TF=TA(IA)` and `TF=TB(IB)`.

Starting in MATLAB R2013a, the behavior of INTERSECT has changed. To preserve the behavior of previous versions of MATLAB, pass in the string `'legacy'`.

Examples

```
>> ta=imat_ctrace('1x','2y','3x');
>> tb=imat_ctrace('1x','1y','1z','2x','2y','2z','3x','3y');
>> [tf,ia,ib]=intersect(ta,tb)

tf =
     '1X+'
     '2Y+'
     '3X+'

ia =
     1
     2
     3

ib =
     1
     5
     7

>>
```

See Also

imat_ctrace/ismember, imat_ctrace/setdiff, imat_ctrace/setxor, imat_ctrace/union, imat_ctrace/unique

IMAT Methods and Functions for `imat_filt` objects

imat_filt	Create an `imat_filt` object
imat_filt/get	Get filter attributes.
imat_filt/set	Set filter attributes.
imat_filt/length	Return number of criteria in filter
imat_filt/char	Convert function filter into character string
imat_filt/and	Generate a combined filter (`z1&z2`)
imat_filt/or	Generate a combined filter (`z1|z2`)
imat_filt/not	Logically negate a filter (`~z`)
imat_filt/criteria	Extract individual criteria from `imat_filt`
imat_filt/in_filter	Check which `imat_fn` elements satisfy a filter
imat_filt/uiselect	Build or edit filter using graphical interface

imat_filt/imat_filt

Purpose

Create an `imat_filt` object which can be used to select elements of an `imat_fn`.

Syntax

```
z=imat_filt( attribute, relation, value )
z=imat_filt( { attrib1, rel1, val1 ; attrib2, rel2, val2 ; ... } )
```

Description

This function, the IMAT filter constructor, creates a function filter (`imat_filt` object) from one or more criteria.

An `imat_filt` variable consists of one or more criteria, each composed of an *attribute*, a *relation*, and a *value*.

The *attribute* of a criterion can be the name of any valid data attribute for an `imat_fn`, with the exception of *Abscissa* and *Ordinate*. The *relation* of a criterion can be any any of the following:

Relation	Criterion is satisfied if...
`'=='` or `'='`	*attribute* is equal to the specified *value*
`'~='` or `'!='`	*attribute* is not equal to the specified *value*
`'>'`	*attribute* is greater than the specified *value*
`'>='`	*attribute* is greater than or equal to the specified *value*
`'<'`	*attribute* is less than the specified *value*
`'<='`	*attribute* is less than or equal to the specified *value*

The last four relations are applicable only to numeric data attributes.

The *value* of a criterion should be a scalar (for numeric data attributes) or a string (for string attributes) suitable for the data attribute. String matching is case insensitive. Two special features should be noted:

- For the *ReferenceCoord* and *ResponseCoord* data attributes, you may specify a coordinate trace as the value of the criterion. In this case, equality is satisfied if the reference (or response) coordinate of a function is included in the coordinate trace.
- For string attributes, you may include wildcard characters: a question mark matches any single character, and an asterisk matches zero or more characters. (For example, `'ab?d*'` matches the strings `'AbCd'` and `'abxdefg'`, but not `'abccd'`.) Wildcards cannot be used for list attributes such as *FunctionType*, which have a restricted list of possible values.

The first syntax creates an `imat_filt` with a single criterion. The second syntax (with a single Nx3 cell array) creates an `imat_filt` with multiple criteria, all of which must be satisfied.

To create more complicated filters, combine simple filters using and, or, and not logical operators.

Examples

```
>> imat_filt('AbscissaSpacing','==','Even');
>> imat_filt('AbscissaMin','>',0);
>> imat_filt('ResponseCoord','=',{'1x','2y-'})
```

See Also

imat_filt/and, imat_filt/or, imat_filt/not, imat_filt/in_filter

User's Guide

---

imat_filt/length

Purpose

Return number of criteria in function filter.

Syntax

```
l=length(z)
```

Description

When applied to an `imat_filt` object, the LENGTH function returns the number of criteria in the filter.

Examples

```
>> z=imat_filt('functiontype','=','frequency response');

>> z=z & {'responsenode','>',100}
z =
  (FunctionType == 'Frequency Response Function') & (ResponseNode > 100)

>> length(z)
ans =
     2
>>
```

---

imat_filt/get

Purpose

Get filter attributes.

Syntax

```
v=get(f,'Attrib')
v=get(f)
```

Description

When applied to `imat_filt` objects, the GET function returns the values of selected attributes of `f`. The only available attribute is `'Name'`, the filter name. Attribute names are not case sensitive.

If only a single attribute is requested, then the value of that attribute is returned, or printed if no output argument is supplied.

If no attributes are specified, then the values of *all* attributes will be returned as described above.

An alternative to the GET function is the syntax `f.`*`attrib`*.

Examples

```
>> f.name='FRF';

>> get(f,'Name')
FRF

>>
```

See Also

imat_filt/imat_filt, imat_filt/set

---

imat_filt/set

Purpose

Set filter attributes.

Syntax

```
f=set(f,'Attrib',value)
set(f)
set(f,'Attrib')
```

Description

When applied to `imat_filt` objects, the SET function sets the values of selected attributes of `f`. The only available attribute is 'Name', the filter name. Attribute names are not case sensitive.

If SET is called with no attribute arguments, then all attribute names and their possible values are printed to standard output. This functionality acts as a built-in help mechanism for data attributes. (Note that in this case, the argument `f` is not referenced, but simply causes MATLAB to call the SET function associated with `imat_filt` objects.).

To change attribute values, you must either supply pairs of attribute names and values.

If only a single attribute is listed with no corresponding value, then the attribute is displayed along with the type of data allowed for the corresponding value.

If no attributes are specified, then the values of *all* attributes will be returned as described above.

An alternative way to set attributes is with the syntax `f.`*`attrib`*`=value`.

Examples

```
>> f.name='FRF';
>> set(f,'Name','FRF');
>> set(f)

    Name: [ string ]

>> set(f,'Name')
Name: [ string ]
```

See Also

imat_filt/imat_filt, imat_filt/get

---

imat_filt/char

Purpose

Convert function filter into character string.

Syntax

```
s=char(z)
```

Description

When applied to an `imat_filt` object, the CHAR function converts a filter `z` into a human-readable string representation. This process is not reversible (i.e., you cannot pass the resulting string to the `imat_filt` constructor to build an `imat_filt` variable).

Note that the character representation is what gets printed when you display a filter.

Examples

```
>> z=imat_filt('functiontype','=','frequency response')

z =
  FunctionType == 'Frequency Response Function'

>> s=char(z)

s =
     FunctionType == 'Frequency Response Function'

>> whos
Name      Size         Bytes  Class
s         1x45            90  char array
z         1x1            610  imat_filt object
Grand total is 93 elements using 700 bytes
>>
```

See Also

imat_filt/imat_filt

---

imat_filt/and

Purpose

Combine two filters with an AND operation.

Syntax

```
z1 & z2
and(z1,z2)
```

Description

When applied to `imat_filt` objects, the AND function creates a new `imat_filt` object which selects only those records that satisfy both filters.

Examples

```
>> f

f =

5x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Frequency Respon Even             1601
```

```
2_(1X+,2Y+)                  Frequency Respon Even             1601
3_(1X+,2Z+)                  Frequency Respon Even             1601
4_(1X+,3X+)                  Frequency Respon Even             1601
5_(1X+,1X+)                  Auto Spectrum    Even             1601

>> z1=imat_filt('functiontype', '=', 'Frequency Response Function');
>> z2=imat_filt('responsecoord', '=', '1x*');
>> f{z1}

ans =

4x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Frequency Respon Even             1601
2_(1X+,2Y+)                  Frequency Respon Even             1601
3_(1X+,2Z+)                  Frequency Respon Even             1601
4_(1X+,3X+)                  Frequency Respon Even             1601

>> f{z2}

ans =

2x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Frequency Respon Even             1601
2_(1X+,1X+)                  Auto Spectrum    Even             1601
>> f{z1&z2}

ans =

IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Frequency Respon Even             1601

>>
```

See Also

imat_filt/imat_filt, imat_filt/or, imat_filt/not

---

imat_filt/or

Purpose

Combine two filters with an OR operation.

Syntax

```
z1 | z2
or(z1,z2)
```

Description

When applied to `imat_filt` objects, the OR function creates a new `imat_filt` object which selects records that satisfy either of the two filters.

Examples

```
>> f

f =

5x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Frequency Respon Even             1601
2_(1X+,2Y+)                  Frequency Respon Even             1601
3_(1X+,2Z+)                  Frequency Respon Even             1601
4_(1X+,3X+)                  Frequency Respon Even             1601
5_(1X+,1X+)                  Auto Spectrum    Even             1601

>> z1=imat_filt('functiontype', '=', 'Auto Spectrum');
>> z2=imat_filt('responsecoord', '=', '3x*');
>> f{z1}

ans =

IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Auto Spectrum    Even             1601

>> f{z2}

ans =

IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,3X+)                  Frequency Respon Even             1601

>> f{z1|z2}

ans =

2x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,3X+)                  Frequency Respon Even             1601
2_(1X+,1X+)                  Auto Spectrum    Even             1601

>>
```

See Also

imat_filt/imat_filt, imat_filt/and, imat_filt/not

---

imat_filt/not

Purpose

Logically negate a filter.

Syntax

```
~z
not(z)
```

Description

When applied to an `imat_filt` object, the NOT function creates a new `imat_filt` object which selects those records that were not satisfied by the original filter.

Examples

```
>> f

f =
5x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Frequency Respon Even             1601
2_(1X+,2Y+)                  Frequency Respon Even             1601
3_(1X+,2Z+)                  Frequency Respon Even             1601
4_(1X+,3X+)                  Frequency Respon Even             1601
5_(1X+,1X+)                  Auto Spectrum    Even             1601

>> z=imat_filt('functiontype', '=', 'Frequency Response Function');

>> f{z}

ans =

4x1 IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Frequency Respon Even             1601
2_(1X+,2Y+)                  Frequency Respon Even             1601
3_(1X+,2Z+)                  Frequency Respon Even             1601
4_(1X+,3X+)                  Frequency Respon Even             1601

>> f{~z}

ans =

IMAT Function with the following attributes:

Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Auto Spectrum    Even             1601

>>
```

See Also

imat_filt/imat_filt, imat_filt/and, imat_filt/or

---

imat_filt/criteria

Purpose

Extract individual criteria from `imat_filt`.

Syntax

```
x=criteria(z)
```

Description

CRITERIA returns a cell array `X` whose contents are `imat_filt` variables with the individual criteria composing the `imat_filt Z`. In this process, the desired combination (and, or, not) is lost.

For example, if `z1`, `z2`, and `z3` are single-criterion filters, and `z=(z1&(z2|z3))`, then `x=criteria(z)` returns `x` such that `x{1}=z1`, etc.

---

imat_filt/in_filter

Purpose

Check which `imat_fn` elements satisfy a filter.

Syntax

```
v=in_filter(z,f)
```

Description

The IN_FILTER utility function creates a logical array v which is true (1) for all elements of and imat_fn object f which satisfy the `imat_filt` object `z`, and false (0) elsewhere. The output array has the same dimension as `f` (even if `f` is multidimensional).

Examples

```
>> f

f =
5x1 IMAT Function with the following attributes:
Record Name                  FunctionType     AbscissaSpacing  NumberElements
---------------------------- ---------------- ---------------- ----------------
1_(1X+,1X+)                  Fquency Respon   Even             1601
2_(1X+,2Y+)                  Frequency Respon Even             1601
3_(1X+,2Z+)                  Frequency Respon Even             1601
4_(1X+,3X+)                  Frequency Respon Even             1601
5_(1X+,1X+)                  Auto Spectrum    Even             1601

>> z=imat_filt('functiontype', '=', 'Frequency Response Function');

>> in_filter(z,f)
```

```
ans =
       1
       1
       1
       1
       0

>>
```

See Also

imat_filt/imat_filt

---

imat_filt/uiselect

Purpose

Build or edit function filter using a graphical interface.

Syntax

```
z=uiselect(imat_filt)
z=uiselect(z1)
```

Description

The UISELECT function for `imat_filt` objects brings up a filter form reminiscent of the Imat filter form for function and time history ADF record selection. In the first form, an empty initial filter (built by the `imat_filt` constructor) is passed to UISELECT, and a new filter will be built from scratch. In the second form, an existing `imat_filt` object `z1` is passed as an input argument, and the criteria can be edited.

The UISELECT function only works for filters whose criteria are combined with an "and" operation. Regardless of the criteria combination of `z1`, the output filter `z` will have all criteria combined with an "and".

The graphical interface displays the filter criteria. The user can select criteria and delete them. The user can also add criteria using popup menus, as well as modify existing criteria.

If the user clicks on 'Cancel', then `z=-1` is returned.

IMAT Methods and Functions for `imat_result` objects

---

imat_result	Create an `imat_result` object
imat_result/get	Get one or more attributes of an `imat_result` object

imat_result/set	Set one or more attributes of an `imat_result` object
imat_result/edit_attributes	Convenient GUI for editing `imat_result` data attributes
imat_result/setdisplay	Set attributes to show for `imat_result` displays
imat_result/list	List `imat_result` components and data in tabular form
imat_result/sort	Sort components for `imat_result`
imat_result/chgunits	Convert an `imat_result` to a different unit system
imat_result/plot	Plot results
imat_results/vtkplot (+FEA)	Plot and animate `imat_shp`and/or `imat_result` in 3-D space
imat_result/uiselect	Select results using a graphical interface
imat_result/ctranspose	Conjugate transpose an `imat_result` object.
imat_result/transpose	Transpose the dimensions of an `imat_result` object.
imat_result/uplus	Unary plus (`+s`)
imat_result/uminus	Unary minus (`-s`)
imat_result/plus	Add results (r`1+r2`)
imat_result/minus	Subtract results (`r1-r2`)
imat_result/mtimes	Matrix multiply results (r1*r2)
imat_result/times	Termwise multiply result data values (`r1.*r2`)
imat_result/mrdivide	Matrix divide results (r1/r2)
imat_result/rdivide	Termwise divide result data values (`r1./r2`)
imat_result/real	Take real part of result data values
imat_result/imag	Take imaginary part of result data values

imat_result/conj	Take complex conjugate of result data values
imat_result/abs	Take absolute value (or modulus) of result data values
imat_result/group_by	Group results by the values of a specified attribute
imat_result/criterion	Create criterion data from Data At Nodes On Elements results.
imat_result/invariant	Compute invariants.

imat_result/partition	Partition results.
imat_result_dan/partition	Partition Data At Nodes results.
imat_result_danoe/partition	Partition Data At Nodes On Elements results.
imat_result_dap/partition	Partition Data At Points results.
imat_result_doe/partition	Partition Data On Elements results.
imat_result_doean/partition	Partition Data On Elements At Nodes results.

imat_result_dan/setComponents	Set the components for Data At Nodes result.
imat_result_doe/setComponents	Set the components for Data On Elements result.
imat_result_danoe/setComponents	Set the components for Data At Nodes On Elements result.
imat_result_doean/setComponents	Set the components for Data On Elements At Nodes result.
imat_result_dap/setComponents	Set the components for Data At Points result.

imat_result/sort

Purpose

Sort components for `imat_result`.

Syntax

```
sort(r)
sort(r,COLLIST)
[r,ind] = sort(r)
```

Description

SORT sorts the components and data in increasing ID order.

COLLIST is an optional numeric vector specifying the list of columns to use in the sort. The components will be sorted in the order specified in COLLIST. A negative column number will sort that column in descending order. The default is 1.

IND is an optional output cell array containing vectors of the sort order for each individual result. It is the same size as R.

Examples

```
>> d = imat_result_dan;
>> d = setComponents(d,[100 101],3:-1:1,0,true,1:6)

d =
Data At Nodes: Unknown
================================================
      Node       Dir      SEID              Data
------------------------------------------------
      100          3         0                 1
      100          2         0                 2
      100          1         0                 3
      101          3         0                 4
      101          2         0                 5
      101          1         0                 6
------------------------------------------------
Number of Values: 6 Number of Nodes: 2
================================================

>> sort(d)

ans =

Data At Nodes: Unknown
================================================
      Node       Dir      SEID              Data
------------------------------------------------
       100         3         0                 1
       100         2         0                 2
       100         1         0                 3
       101         3         0                 4
       101         2         0                 5
       101         1         0                 6
------------------------------------------------
Number of Values: 6 Number of Nodes: 2
================================================

>> sort(d,[1 2 3])

ans =

Data At Nodes: 3DOF global translation vector
================================================
      Node       Dir      SEID              Data
------------------------------------------------
       100         1         0                 3
       100         2         0                 2
       100         3         0                 1
       101         1         0                 6
       101         2         0                 5
       101         3         0                 4
------------------------------------------------
Number of Values: 6 Number of Nodes: 2
================================================
```

---

imat_result/imat_result

Purpose

Create an `imat_result` object.

Syntax

```
r=imat_result
r=imat_result(m,n,p,...)
r=imat_result(m,n,p,...,'Attr',value,...)
r=imat_result('Attr',value,...)
r=imat_result(m,n,p,...,v)
r=imat_result(v)
r=imat_result(imat_shp)
r=imat_result(imat_fn)
```

Description

You call IMAT_RESULT to construct an `imat_result` object. A result object contains result data and attributes that define the data source and contents such as data location, result type, and data components.

Calling IMAT_RESULT with no arguments creates a scalar `imat_result` object.

If the first one or more arguments to IMAT_RESULT are integers, then an `mxnxpx... result` is created. Each element of this object is a single result. (Alternatively, the dimension of the result can be specified by a row vector of dimensions, as in `imat_result([2 2])`.) The output result will have the default attributes.

To override default attributes at time of object creation, you can specify one or more attribute names and values in the call to `imat_result`. This is equivalent to creating the object with default attributes, and then setting the specified attributes in sequential order.

Instead of a list of attribute names and values, you may supply a structure variable v with field names equal to attribute names, and field values set to the desired attribute values. (Such a structure can be obtained from the get function with multiple attribute requests.)

If the input is an `imat_shp` object, IMAT_RESULT creates an `imat_result` from the supplied `imat_shp`. It creates one result for each shape. Squared attribute types will be converted as linear, which may produce incorrect results if the units are changed.

If the input is an `imat_fn` object, IMAT_RESULT creates a result from the supplied `imat_fn`. It only converts functions with an abscissa data type of Time or Frequency that are not empty. It builds results based on function type, abscissa data type, ordinate numerator, then ordinate denominator data types. The abscissa values go into the Time or Frequency attributes of the result. If there are multiple groups of functions, F will be a cell array, each containing an `imat_result` object based on the function group.

Examples

```
>> r=imat_result(3,imat_result_dan,'resulttype','acceleration')
r =
3x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
1          Data At Nodes  Unknown    Acceleration
2          Data At Nodes  Unknown    Acceleration
3          Data At Nodes  Unknown    Acceleration

>>
```

See Also

imat_result/set, imat_result/get

---

imat_result/get

Purpose

Get one or more attributes of an `imat_result` object.

Syntax

```
v=get(r,'Attrib')
v=get(r,'Attrib1','Attrib2',...)
v=get(r)
```

Description

When applied to an `imat_result` object `s`, the GET function returns the values of selected attributes of `r`. The available attributes are listed here. Attribute names are not case sensitive.

If only a single attribute is requested, then the value of that attribute is returned, or printed if no output argument is supplied. Numeric attributes are returned in a numeric array with the same dimensions as `r`. String-valued attributes (including list attributes) are returned in a string variable (if `s` is a single result) or a cell array of strings with the same dimensions as `r` (if `r` contains more than one result). The *Data* attribute returns a cell array containing the individual result data objects. If any of the results has a smaller number of data values, that column will be padded with NaN values.

If more than one attribute is specified, then the output argument will be a scalar structure variable with field names equal to the requested attributes. The value of each field will be as described above. If no output argument is supplied, the resulting structure will be printed on the standard output.

If no attributes are specified, then the values of *all* attributes will be returned as described above.

An alternative to the `get` function is the syntax *r.attrib*.

Examples

```
>> setunits('si')
Units set to SI

>> r=readunv(fullfile('..','data','ideas_results_modes_statics.unv'),'asresult',2414)
No units in universal file, assuming SI
Read 17x1 result
r =
17x1 IMAT_RESULT
Row  Name                                 DataLocation               ModelType   ResultType
---  -----------------------------------  -------------------------  ----------  -----------------
-------
  1  B.C. 1,NORMAL_MODE 1,DISPLACEMENT_   Data at nodes              Structural  Displacement
  2  B.C. 1,NORMAL_MODE 2,DISPLACEMENT_   Data at nodes              Structural  Displacement
  3  B.C. 1,NORMAL_MODE 3,DISPLACEMENT_   Data at nodes              Structural  Displacement
  4  B.C. 1,NORMAL_MODE 4,DISPLACEMENT_   Data at nodes              Structural  Displacement
  5  B.C. 1,NORMAL_MODE 5,DISPLACEMENT_   Data at nodes              Structural  Displacement
  6  B.C. 1,NORMAL_MODE 6,DISPLACEMENT_   Data at nodes              Structural  Displacement
  7  B.C. 1,NORMAL_MODE 7,DISPLACEMENT_   Data at nodes              Structural  Displacement
  8  B.C. 1,NORMAL_MODE 8,DISPLACEMENT_   Data at nodes              Structural  Displacement
  9  B.C. 1,NORMAL_MODE 9,DISPLACEMENT_   Data at nodes              Structural  Displacement
 10  B.C. 1,NORMAL_MODE 10,DISPLACEMEN_   Data at nodes              Structural  Displacement
 11  B.C. 2,DISPLACEMENT_11,LOAD SET 1    Data at nodes              Structural  Displacement
 12  B.C. 2,REACTION FORCE_12,LOAD SET    Data at nodes              Structural  Reaction Force
 13  B.C. 2,STRESS_13,LOAD SET 1          Data at nodes on elements  Structural  Stress
 14  B.C. 2,STRAIN_14,LOAD SET 1          Data at nodes on elements  Structural  Strain
 15  B.C. 2,EL STRESS RES_15,LOAD SET 1   Data at nodes on elements  Structural  Element Stress
Resultant
 16  B.C. 2,STRAIN ENERGY_16,LOAD SET 1   Data on elements           Structural  Strain Energy
 17  B.C. 2,ELEMENT FORCE_17,LOAD SET 1   Data at nodes on elements  Structural  Element Force

>> get(r(1))
                   Name: ' B.C. 1,NORMAL_MODE 1,DISPLACEMENT_1'
                IDLine1: ''
                IDLine2: 'MODEL_SOLUTION_SOLVE'
                IDLine3: 'Mode Shape 1  Frequency<Hz> =  13.09024'
                IDLine4: 'Error =  4.5E-12 %  Analysis time was     24-Mar-05 15:41:47'
                IDLine5: 'Solset 1 - NORMAL MODES  M orthog= 1.9E-16  K orthog= 6.5E-14'
              ModelType: 'Structural'
           AnalysisType: 'Normal mode'
             ResultType: 'Displacement'
           DataLocation: 'Data at nodes'
           NumberValues: 216
              ExpLength: 1
               ExpForce: 0
         ExpTemperature: 0
                ExpTime: 0
            DesignSetID: -10
        IterationNumber: 0
          SolutionSetID: 1
      BoundaryCondition: 1
                LoadSet: 0
             ModeNumber: 1
         TimeStepNumber: 0
```

```
          FrequencyNumber: 0
           CreationOption: 2
           NumberRetained: 0
                     Time: 0
                Frequency: 13.0902
               Eigenvalue: 82.2482
           EigenvalueReal: 82.2482
           EigenvalueImag: 0
                ModalMass: 0
            ModalMassReal: 0
            ModalMassImag: 0
           ViscousDamping: 0
       HystereticDamping: 0
                   ModalA: 0
               ModalAReal: 0
               ModalAImag: 0
                   ModalB: 0
               ModalBReal: 0
               ModalBImag: 0
                Stiffness: 0
            StiffnessReal: 0
            StiffnessImag: 0
            EffectiveMass: [6x1 double]
      ParticipationFactor: [6x1 double]
                     data: [1x1 imat_result_dan]

>>
```

See Also

imat_result, imat_result/set

---

imat_result/set

Purpose

Set one or more attributes of an `imat_result` object.

Syntax

```
set(r)
r1=set(r,'attrib1',value1,'attrib2',value2,...)
r1=set(r,v)
```

Description

The SET function allows you to change any attribute of an `imat_result` variable.

If SET is called with no attribute arguments, then all attribute names and their possible values are printed to standard output. This functionality acts as a built-in help mechanism for data attributes. (Note that in this case, the argument `r` is not referenced, but simply causes MATLAB to call the SET function associated with `imat_result` objects.)

To change attribute values, you must either supply pairs of attribute names and values (syntax 2), or you must supply a scalar structure variable (like the one returned by get) whose field names are the attribute names to be changed, and whose field values are the desired values (syntax 3). The second argument is a numeric vector of indices into `r`.

The attribute names can be any valid attribute for the `imat_result` object. Attribute values can either be a single value (in which case the value is used to set all elements of `r`), or an array of values dimensioned the same size as `r`. A cell array of strings should be used to set string or list attributes.

The attributes are set in the order listed. Note that setting some attributes will cause side effects:

- Changing the *ResultType* adjusts the corresponding exponents to match. Changing the exponents can cause the data type to change to `'User defined'`.

An alternative way to set attributes is with the syntax *r.attrib=value*.

Examples

```
>> r=imat_result(2);

>> r=set(r,'Name',{'Name 1','Name 2'},'modeltype','structural')

r =

2x1 IMAT_RESULT

Row  Name    DataLocation   ModelType   ResultType
---  ------  -------------  ----------  ------------
  1  Name 1  Data at nodes  Structural  User defined
  2  Name 2  Data at nodes  Structural  User defined

>> r(1)=set(r(1),'resulttype','displacement');

>> r=set(r,1:2,'data',ones(10,2))

r =

2x1 IMAT_RESULT

Row  Name    DataLocation   ModelType   ResultType
---  ------  -------------  ----------  ------------
  1  Name 1  Data at nodes  Structural  Displacement
  2  Name 2  Data at nodes  Structural  User defined

>>
```

See Also

imat_result, imat_result/get

imat_result/setdisplay

Purpose

Set attributes to show for `imat_result` displays.

Syntax

```
setdisplay(r,'Attrib1','Attrib2','Attrib3')
att=setdisplay(r,{'Attrib1','Attrib2'})
setdisplay(imat_result,999)
```

Description

When an `imat_result` is displayed on the screen (such as when a semicolon is omitted at the end of a statement), a summary of the variable is printed. Several attributes are shown in the display (by default, the *Name*, *DataLocation*, *ModelType*, and *ResultType* attributes). If you specify a specific result variable as the first input argument, the attributes specified will apply to that object only. If you specify the constructor function as in the second example above, the display attributes will apply to all subsequent `imat_result` objects created.

Input attribute names can be specified either as a series of strings, or a cell array of strings. The optional output ATT is a cell array of strings containing the current display attributes.

Use the SETDISPLAY function to change the displayed attributes. If no attributes are specified, the default attributes will be assumed. Any attributes other than *Data*, *EffectiveMass*, or *ParticipationFactor* may be selected.

The third syntax shown above allows you to specify the maximum number of results to display before switching to the truncated display form of just showing the dimensions of the `imat_result`. Setting this to 0 means that all of the results should always be displayed.

Examples

```
>> setdisplay(imat_result,'name,'frequency','idline1')
```

See Also

imat_result/set

---

imat_result/list

Purpose

List `imat_result` components and data in tabular form.

Syntax

```
list(rst)
list(rst,(1:10)')
list(rst(1:3),[1 1; 1 2; 1 3])
list(rst,[1 inf 3; inf inf 2]);
data=list(rst,...)
```

Description

LIST lists the components and data for the supplied `imat_result` in RST one at a time. The component columns are listed on the left, and the data column is listed on the right.

COMPONENTS is an optional vector or matrix of components. If supplied, only components and data matching COMPONENTS will be listed. If not supplied, all of the components and data will be listed. If COMPONENTS is an MxN matrix and has fewer columns than the number of component columns for a result, then all components matching the first N columns will be displayed. If you just want to match the first column, then you must pass in a column vector. Passing in an INF for any of the columns will match all values in that column.

DATA is an optional output containing a cell array the same size as RST. Each cell contains a matrix of the components and data displayed for each result.

---

imat_result/edit_attributes

Purpose

Convenient GUI for editing `imat_result` data attributes.

Syntax

```
g=edit_attributes(f)
```

Description

EDIT_ATTRIUBTES is a convenience Graphical User Interface for editing `imat_result` attributes. Most attributes are editable. The GUI groups related attributes for easy editing.

The input argument to EDIT_ATTRIBUTES is an `imat_result`. It can be an array of functions. The output G is an `imat_result` containing the modified `imat_result`. If the user clicks on 'Cancel', then `g=-1` is returned.

See Also

imat_fn/edit_attributes, imat_shp/edit_attributes

---

imat_result/chgunits

Purpose

Convert an `imat_result` to a different unit system.

Syntax

```
r1=chgunits(r,from,to)
```

Description

CHGUNITS converts an `imat_result` to a different system of units. This affects the data values (*Data* attribute), the modal mass (*ModalMass* attribute), the stiffness (*Stiffness* attribute), the modal A value (*ModalA* attribute), and the modal B value (*ModalB* attribute).

The FROM argument is the unit system that `r` is expressed in currently. The `to` argument is the desired unit system (if `to` is omitted, the current unit system is assumed). The unit system arguments should either be unit strings (such as `'SI'`) or should be 5x1 vectors of unit conversion factors as returned by the getunits function.

If you follow the recommended practice of setting your units at the start of a session and maintaining consistency, you should not need to use this function.

Examples

```
>> r=chgunits(r,'mm','in');      % Change r from MM to IN
```

See Also

getunits, setunits

---

imat_result/get_units_labels

Purpose

Get units labels for the supplied `imat_result`.

Syntax

```
str=get_units_labels(r)
str=get_units_labels(r,'full')
```

Description

GET_UNITS_LABELS returns a cell array of strings the size of the input `imat_result` R which contains the units label string for each of the results. The optional input string `'full'` specifies whether to use the units abbreviations or their full names. Leaving this argument off uses abbreviations.

Examples

```
>> r=imat_result(1,'resulttype','acceleration');
>> setunits('in');
>> get_units_labels(r)

ans =
    'in/s^2'

>> get_units_labels(r,'full')

ans =
    'inch/second^2'

>>
```

See Also

imat_fn/get_units_labels, imat_shp/get_units_labels

---

imat_result/plot

Purpose

Plot an `imat_result` in a special figure window.

Syntax

```
plot(r,fem)
[h,shpout]=plot(r,fem,handle,-
format,complexdisplay,'noundeformed','title',titlestr,'scale',scalefactor,'silent')
plot(r)
```

Description

When you PLOT an `imat_result`, the IMAT toolbox displays the data in a figure. Internally, PLOT will either convert to an `imat_shp` or `imat_fn`, and then calls the appropriate PLOT method for either of these two objects, passing along the optional input arguments to this function. See the help for either of these PLOT functions for details on their optional arguments.

If an IMAT_FEM object is passed into PLOT, it will call the `imat_shp` PLOT method. Otherwise it will call the `imat_fn` PLOT method.

For large models, consider using VTKPLOT.

See Also

imat_fem/plot, imat_shp/plot, imat_fn/plot, imat_result/vtkplot

imat_result/vtkplot (+FEA)

Purpose

Plot and animate `imat_shp`and/or `imat_result` in 3-D space

Syntax

```
plot(s,fem)
[h,shpout]=plot(shp,fem,res,handle,'title',titlestr,'scale',scale,options)
```

Description

VTKPLOT displays the shapes supplied in the `imat_shp` object SHP and/or the results stored in the `imat_result` object RES using the supplied FEM. In general, the model's displacement is determined by the shape, and its contour by the result.

VTKPLOT expects that FEM information is an IMAT_FEM object. If node geometry is not in the global coordinate system, the coordinate system data must be present so PLOT can transform the node coordinates. The input shape coefficients are assumed to be in the FEM's displacement coordinate system(s).

By default, if a shape is present, only the deformed model will be displayed. The undeformed model can be displayed by using the 'Display' menu. The color is either determined by the result, if one was passed in, or by the displacement of the shape. The contour or shape component displayed can be changed using the 'Result' menu.

The Display menu control what is shown on the plot and how. The 'Complex Display' menu controls how complex data is displayed if the displacement shape is complex. Below this, the clipping menus allow you to set up clipping planes for the model. There are four available clipping planes: three aligned with each axis and a general one that is arbitrary. Each can be enabled using the corresponding 'Enable' menu item. To keep the result of the clipping plane, but remove the plane from view, set the 'Set Invisible' option. To flip the side of the plane that is not visible, select the 'Flip Clipping Direction' option. To further customize the view, you can use the 'Element Visibility' GUI to adjust what kinds of elements are shown on the plot. The 'Baseline Model' and 'Result Model' menus contain options for changing how nodes, elements and tracelines are formatted and can be used so set the transparency of the model.

The view menu allows you to snap the view to the difference defined planes. If the toolbar is visible, each of the view options have a corresponding toolbar icon. From this menu you can also use a 'Zoom Box' to zoom into a portion of the model, or turn on the 3D mode where red/blue 3D glasses can be used to view the model.

To export an image of the model or an animation of the mode shape, use the 'Export' menu. If you want to export all of the mode shapes using the current view, use Export->All Slides.

If displacement shapes are present, the 'Animation' menu controls the animation. Here you can set the scale of the animation, and set whether it will animate as a mode shape or animate as a transient result.

Several optional input arguments are supported:

GROUP is an IMAT_GROUP containing group information. If supplied, you will be able to display subsets of your FEM based on the elements in the group(s).

HANDLE is a figure or panel handle. If a figure handle is supplied, the shape will be displayed on the supplied figure after deleting the contents of the figure. If a panel handle is supplied, the shape will be displayed on the supplied panel after deleting the content of the panel.

OPTIONS is a string containing one of the following strings. Multiple options may be specified. See imat_shp/vtkplot for a list of valid options.

The output argument for VTKPLOT is an IMAT_VTKPLOT object. It contains all of the formatting and contents of the plot. If IDLines are modified during the plotting session, and they need to be retrieved, place the handle to the figure into `uiwait`, and then retrieve the modified `imat_shp` object once the user is done editing it.

Examples

```
>> vtkplot(s(1),fem)
>> h = vtkplot(s(1),fem,h,'undeformed')
>> h = vtkplot(shape,fem,h,'title','Mode Shapes','imag','scale',2.3)
```

See Also

imat_fem/vtkplot, imat_shp/vtkplot

---

imat_result/uiselect

Purpose

Select `imat_result` using a graphical interface.

Syntax

```
g=uiselect(f)
g=uiselect(f,'Title')
g=uiselect(f,[presel])
g=uiselect(f,{'Attrib1','Attrib2','Attrib3'})
[g,ind]=uiselect(f)
[g,ind]=uiselect(f,'Title',[presel])
```

Description

The UISELECT function brings up a result selection form, listing all elements of the `imat_result`. The user can select elements of `f`, then click on 'OK'. The selected elements are returned in the output argument `g`. If a title string is provided, it is used to name the result selection window. If a numeric vector is provided, the results corresponding to the indices in the vector will be preselected when the form is displayed. The optional second output argument `ind` will contain the indices into `f` of the functions selected and returned in `g`. The attributes listed in the attribute columns on the form will default to the attributes set by SETDISPLAY. Alternately, you can specify them by passing in a cell array of strings of attributes to display.

If the user clicks on 'Cancel', then `g=-1` is returned.

The UISELECT form also contains a button that allows you to specify which attribute should be displayed in the form, and in what order.

See Also

imat_result/setdisplay

---

imat_result/ctranspose

Purpose

Conjugate transpose an `imat_result` object.

Syntax

```
r'
ctranspose(r)
```

Description

CTRANSPOSE transposes the dimensions of an `imat_result` object. Since it is a conjugate transpose, it also sets the data values to their conjugate. This only works for 2-dimensional `imat_result` objects.

---

imat_result/transpose

Purpose

Transpose the dimensions of an `imat_result` object.

Syntax

```
r.'
transpose(r)
```

Description

TRANSPOSE transposes the dimensions of a result object. It performs the non-conjugate transpose, so that only the dimensions are changed, not the data. This only works for 2-dimensional `imat_result` objects.

---

imat_result/uplus

Purpose

Unary plus (`+r`).

Syntax

```
+r
```

Description

This function does not affect an `imat_result`. It is included for completeness only.

See Also

imat_result/uminus

---

imat_result/uminus

Purpose

Unary minus (`-r`).

Syntax

```
-r
```

Description

Taking the negative of an `imat_result` object causes all data values to be replaced with their negatives.

Examples

```
>> for k=1:length(r)
    if -min(r(k).data)>max(r(k).data), r(k)=-r(k); end
end
>>
```

See Also

imat_result/uplus

---

imat_result/plus

Purpose

Add results (`r1+r2`).

Syntax

```
r1+r2
r+x
x+r
```

Description

The following addition operations are possible with `imat_result` objects:

- Adding two `imat_result` objects causes their data values to be added. If both operands are results, they must have the same dimensions. The data attributes of the result of an addition operation will be taken from the left operand.
- Adding a numeric array or scalar to an `imat_result` causes the numeric values to be added to the data values of the `imat_result`. If the numeric array is not a scalar, it can either have the dimensions of the `imat_result` or the dimensions of the *Data* attribute of the `imat_result`. If it has the same dimensions as the `imat_result`, the numeric value corresponding to that `imat_result` will be added to all of the data values of that index in the `imat_result`.

Examples

```
>> r=imat_result(3,'data');
>> r.data.data = reshape(1:18,6,3);

r =
3x1 Result with the following attributes:
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
  1        Data at nodes  Unknown    User defined
  2        Data at nodes  Unknown    User defined
  3        Data at nodes  Unknown    User defined

>> s=r+r;  s.data.data

ans =
     2    14    26
     4    16    28
     6    18    30
     8    20    32
    10    22    34
    12    24    36

>> s=r+1;  s.data.data

ans =
     2     8    14
     3     9    15
     4    10    16
     5    11    17
     6    12    18
     7    13    19

>> s=r+[1 10 100]';  s.data.data

ans =
     2    17   113
     3    18   114
     4    19   115
     5    20   116
     6    21   117
     7    22   118

>> s=r+10*ones(6,3);  s.data.data

ans =
    11    17    23
    12    18    24
    13    19    25
    14    20    26
    15    21    27
    16    22    28

>>
```

See Also

imat_result/minus

---

imat_result/minus

Purpose

Subtract results (`r1-r2`).

Syntax

```
r1+r2
r+x
x+r
```

Description

The following subtraction operations are possible with `imat_result` objects:

- Subtracting two `imat_result` objects causes their data values to be subtracted. If both operands are results, they must have the same dimensions. The data attributes of the result of a subtraction operation will be taken from the left operand.
- Adding a numeric array or scalar to an `imat_result` causes the numeric values to be subtracted from the data values of the `imat_result`. If the numeric array is not a scalar, it can either have the dimensions of the `result` or the dimensions of the *Data* attribute of the `imat_result`. If it has the same dimensions as the `imat_result`, the numeric value corresponding to that `imat_result` will be subtracted from all of the data values of that index in the `imat_result`.

Examples

```
>> r=imat_result(3);
>> r.data.data = reshape(1:18,6,3)

r =
3x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
  1        Data at nodes  Unknown    User defined
  2        Data at nodes  Unknown    User defined
  3        Data at nodes  Unknown    User defined

>> s=r-r;  s.data.data

ans =
     0     0      0
     0     0      0
     0     0      0
     0     0      0
     0     0      0
     0     0      0
```

```
>> s=r01;  s.data.data

ans =
     0     6    12
     1     7    13
     2     8    14
     3     9    15
     4    10    16
     5    11    17

>> s=r-[1 10 100]';  s.data.data

ans =
     0    -3   -87
     1    -2   -86
     2    -1   -85
     3     0   -84
     4     1   -83
     5     2   -82

>> s=r-10*ones(6,3);  s.data.data

ans =
    -9    -3     3
    -8    -2     4
    -7    -1     5
    -6     0     6
    -5     1     7
    -4     2     8

>>
```

See Also

imat_result/plus

---

imat_result/mtimes

Purpose

Matrix multiply results (`r1*r2`).

Syntax

```
r1*r2
```

Description

R1*R2 when either R1 or R2 is an IMAT_RESULT calls this method. Currently only multiplication by a numeric scalar is supported.

The output takes on the attributes of the first IMAT_RESULT input.

See Also

imat_result/times

---

imat_result/times

Purpose

Termwise multiply results (`r1.*r2`).

Syntax

```
r1.*r2
r.*x
x.*r
```

Description

The following termwise multiplication operations are possible with `imat_result` objects:

- Multiplying two `imat_result` objects causes their data values to be multiplied termwise. The *DataLocation*s must agree between the results being multiplied. If both operands are results, they must have the same dimensions. The data attributes of the result of a termwise multiplication will be taken from the left operand, except that result data types (i.e., units exponents) will correctly reflect the combination of the two operands.
- Multiplying a numeric array or scalar times an `imat_result` causes the numeric values to be termwise multiplied into the data values of the `imat_result`. If the numeric array is not a scalar, it can either have the dimensions of the `result` or the dimensions of the *Data* attribute of the `imat_result`. If it has the same dimensions as the `imat_result`, the numeric value corresponding to that index will be multiplied by all of the data values of that index in the `imat_result`.

Examples

```
>> r=imat_result(3);
>> r.data.data = reshape(1:18,6,3)

r =
3x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
  1        Data at nodes  Unknown    User defined
  2        Data at nodes  Unknown    User defined
  3        Data at nodes  Unknown    User defined

>> r.data

ans =
     1     7    13
     2     8    14
     3     9    15
     4    10    16
     5    11    17
     6    12    18

>> s=r.*r;  s.data

ans =
     1    49   169
     4    64   196
     9    81   225
    16   100   256
    25   121   289
    36   144   324

>> s=r.*[2 3 4]';  s.data

ans =
     2    21    52
     4    24    56
     6    27    60
     8    30    64
    10    33    68
    12    36    72

>> s=r.*(2*ones(6,3));  s.data

ans =
     2    14    26
     4    16    28
     6    18    30
     8    20    32
    10    22    34
    12    24    36

>>
```

See Also

imat_result/plus, imat_result/minus, imat_result/rdivide

---

imat_result/mrdivide

Purpose

Matrix divide results (`r1/r2`).

Syntax

```
r1/r2
```

Description

R1/R2 when either R1 or R2 is an IMAT_RESULT calls this method. Currently only division by a numeric scalar is supported.

The output takes on the attributes of the first IMAT_RESULT input.

See Also

imat_result/rdivide

---

imat_result/rdivide

Purpose

Termwise divide results (`r1./r2`).

Syntax

```
r1./r2
r./x
x./r
```

Description

The following termwise division operations are possible with `imat_result` objects:

- Dividing two `imat_result` objects causes their data values to be divided termwise. The *DataLocation*s must agree between the results being multiplied. If both operands are results, they must have the same dimensions. The data attributes of the result of a termwise division will be taken from the left operand, except that result data types (i.e., units exponents) will correctly reflect the combination of the two operands.
- Dividing a numeric array or scalar times an `imat_result` causes the numeric values to be termwise divided into the data values of the `imat_result`. If the numeric array is not a scalar, it can either have the dimensions of the `imat_result` or the dimensions of the *Data* attribute of the `imat_result`. If it has the same dimensions as the `imat_result`, the numeric value corresponding to that index will be divided by all of the data values of that index in the `imat_result`.

Examples

```
>> r=imat_result(3);
>> r.data.data = reshape(1:18,6,3)

r =
3x1 IMAT_RESULT
Row  Name  DataLocation   ModelType  ResultType
---  ----  -------------  ---------  ------------
  1        Data at nodes  Unknown    User defined
  2        Data at nodes  Unknown    User defined
  3        Data at nodes  Unknown    User defined

>> r.data.data
```

```
ans =
     1      7     13
     2      8     14
     3      9     15
     4     10     16
     5     11     17
     6     12     18

>> s=r./r; s.data.data

ans =
     1      1      1
     1      1      1
     1      1      1
     1      1      1
     1      1      1
     1      1      1

>> s=r./[2 3 4]';   s.data.data

ans =
    0.5000     2.3333     3.2500
    1.0000     2.6667     3.5000
    1.5000     3.0000     3.7500
    2.0000     3.3333     4.0000
    2.5000     3.6667     4.2500
    3.0000     4.0000     4.5000

>> s=r./(2*ones(6,3));   s.data.data

ans =
    0.5000     3.5000     6.5000
    1.0000     4.0000     7.0000
    1.5000     4.5000     7.5000
    2.0000     5.0000     8.0000
    2.5000     5.5000     8.5000
    3.0000     6.0000     9.0000

>>
```

See Also

imat_result/plus, imat_result/minus, imat_result/times

---

imat_result/real

Purpose

Take real part of result data values.

Syntax

```
r1=real(r)
```

Description

This function replaces all data values of `r` with their real parts.

See Also

imat_result/imag, imat_result/conj

---

imat_result/imag

Purpose

Take imaginary part of `imat_result` data values.

Syntax

```
r1=imag(r)
```

Description

This function replaces all data values of `r` with their imaginary parts.

See Also

imat_result/real, imat_result/conj

---

imat_result/conj

Purpose

Take complex conjugate of result data values.

Syntax

```
r1=conj(r)
```

Description

This function replaces all data values of `r` with their complex conjugates.

See Also

imat_result/real, imat_result/imag

---

imat_result/group_by

Purpose

Group results by the values of a specified attribute.

Syntax

```
out = group_by(r,att)
out = group_by(r,'DataLocation')
```

Description

GROUP_BY groups an `imat_result` by the values in the attribute specified, and return the output in a structure. This is useful for grouping results by their DataLocation, for example.

R is an `imat_result`.

ATT is a string containing the attribute by whose values the `imat_result` should be grouped. The specified attribute must be one whose type is a string.

OUT is a structure whose fields correspond to the names of the attributes contained in the values of the attribute ATT. Spaces are replaced by underscores (_) so that the structure field names are valid.

Examples

```
>> r = imat_result(3,'Data',imat_result_dan);
>> r(2).Data = imat_result_danoe;
>> r.name = {'1st' '2nd' '3rd'}

r =
3x1 IMAT_RESULT
Row  Name  DataLocation               ModelType  ResultType
---  ----  -------------------------  ---------  ------------
  1  1st   Data At Nodes              Unknown    User defined
  2  2nd   Data At Nodes On Elements  Unknown    User defined
  3  3rd   Data At Nodes              Unknown    User defined

>> out = group_by(r,'DataLocation');

out =
                Data_At_Nodes: [2x1 imat_result]
    Data_At_Nodes_On_Elements: [1x1 imat_result]
```

imat_result/abs

Purpose

Take absolute value of result data values.

Syntax

```
r1=abs(r)
```

Description

This function replaces all data values of `r` with their absolute value. For complex results this is equivalent to the magnitude of the complex value.

See Also

imat_result/real, imat_result/imag

---

imat_result/criterion

Purpose

Create criterion data from Data At Nodes On Elements results.

Syntax

```
s=criterion(r)
s=criterion(r,'max')
s=criterion(r,type)
```

Description

CRITERION processes Data At Nodes On Elements results (i.e. stress) and Data On Elements At Nodes in the result object R and returns the output in the `imat_result` object S. This is useful for creating the results necessary for a criterion plot.

Results other than Data At Nodes On Elements are returned unprocessed. If the result being processed contains centroidal results for `'mean'` output, these are returned. Otherwise, the results at the nodes for each unique combination of element, component, and layer are processed and returned. They are tagged as being located at the centroid ("node" 0).

TYPE is an optional string specifying the processing type. The available choices are

`'mean'`	(Default) Average of nodal results
`'min'`	Take the minimum of the results on the nodes

`'max'`	Take the maximum of the results on the nodes
`'absmax'`	Take the absolute maximum of the results on the nodes

Please note that if you are not using the `'mean'` processing type, and your results have actual node numbers, performance will be much slower.

---

imat_result/imag

Purpose

Compute invariants.

Syntax

```
r=invariant(r,type[,intersect])
r=r.invariant('vonm')
r.invariant('magt',true)
```

Description

INVARIANT computes invariants for each IMAT_RESULT. Supported invariants vary with result DataLocation. Please see the help for each individual result to see what invariants are supported.

TYPE is a string containing the invariant type. For INVARIANT to be successful, all of the results in the IMAT_RESULT must support that invariant type. Depending on the DataLocation, the following invariants are supported. The following table shows the supported invariant types for each DataLocation. The numbers in the DataLocation Columns denote the DataCharacteristic(s) required to compute that invariant.

		DataLocation				
TYPE	**Name**	**Data At Nodes**	**Data On Elements**	**Data At Points**	**Data At Nodes On Elements**	**Data On Elements At Nodes**
`'magt'`	Translational Magnitude	2, 3				
`'magr'`	Rotational Magnitude	3				
`'vonm'`	von Mises				4	4
`'minp'`	Minimum Principal				4	4
`'midp'`	Mid Principal				4	4
`'maxp'`	Maximum Principal				4	4
`'maxs'`	Maximum shear				4	4

The following table defines the DataCharacteristic numbers used in the table above.

DataCharacteristic	Description	Component numbers
2	3DOF vector	1, 2, 3
3	6DOF vector	1, 2, 3, 4, 5, 6
4	Symmetric Tensor	11, 12, 13, 22, 23, 33

INTERSECT is an optional logical specifying whether the components should be intersected. If not, and the components do not match for all of the components required to compute the invariant, an error will occur.

---

imat_result/partition

Purpose

Partition results.

Syntax

```
r=partition(r,ct)
r=d.partition(ct)
[r,v]=partition(r,ct)
[~,v]=partition(r,ct,'veconly')
```

Description

PARTITION partitions imat_result using the input partitioning data CT. Supported partitioning types vary with result DataLocation. Please see the help for each individual result to see what partitioning inputs are supported.

CT is the entity to use to partition the results. It can be a logical or numeric index vector, imat_ctrace, imat_group, or cell array containing an MxN numeric matrix. In the latter case, PARTITION will attempt to match the first N rows of the individual result's component matrix.

V is an optional output. It is a cell array of the same size as R containing logical vectors of the same length as the number of rows in the component matrix for each result containing true for rows that were kept when partitioning.

Examples

imat_result_dan/partition, imat_result_danoe/partition, imat_result_dap/partition, imat_result_doe/partition, imat_result_doe-an/partition

imat_result_dan/partition

Purpose

Partition Data At Nodes results.

Syntax

```
d=partition(d,ct)
d=d.partition(ct)
[d,v]=partition(d,ct)
[~,v]=partition(d,ct,'veconly')
```

Description

PARTITION partitions results using the input partitioning data CT. When CT is a numeric or logical vector, PARTITION will partition the results to the indices specified in CT. This is the same as subscripting the object.

When CT is an imat_ctrace, the result will be partitioned using the nodes and directions in CT. If not all of the coordinates were found, PARTITION will issue a warning.

When CT is an imat_group, the result will be partitioned to the nodes specified in the group. If not all of the nodes in the group were found, PARTITION will issue a warning.

When CT is a cell array containing numeric MxN matrix, PARTITION attempts to match the first N rows in this matrix to the first N rows in the component matrix. To match all of the entries in a column, use INF for that column. For example,

```
CT = {[1 inf 300]}
```

specifies that PARTITION should match all components with node 1, all directions, and SEID 300. CT can contain rows both with and without INF. PARTITION will not issue a warning if not all of the entries were found.

The optional input string `'veconly'` tells PARTITION to generate the output vector V only, and not create the partitioned result.

V is a logical vector of the same length as the number of rows in the component matrix containing true for rows that were kept when partitioning.

Examples

```
>> d = d.partition([1 2 7]);

>> d = d.partition(imat_ctrace('1x'));

>> [d,v] = d.partition({[1 inf 300; 2 2 0]});

>> [~,v] = d.partition([1 2 7],'veconly');
```

---

imat_result_danoe/partition

Purpose

Partition Data At Nodes On Elements results.

Syntax

```
d=partition(d,ct)
d=d.partition(ct)
[d,v]=partition(d,ct)
[~,v]=partition(d,ct,'veconly')
```

Description

PARTITION partitions results using the input partitioning data CT. When CT is a numeric or logical vector, PARTITION will partition the results to the indices specified in CT. This is the same as subscripting the object.

When CT is an imat_ctrace, the result will be partitioned using the nodes and directions (component locations) in CT. If not all of the entities in the coordinate trace were found, PARTITION will issue a warning.

When CT is an imat_group, the result will be partitioned to the nodes and elements specified in the group. If not all of the entities in the group were found, PARTITION will issue a warning.

When CT is a cell array containing numeric MxN matrix, PARTITION attempts to match the first N rows in this matrix to the first N rows in the component matrix. To match all of the entries in a column, use INF for that column. For example,

```
CT = {[1 inf 33]}
```

specifies that PARTITION should match all components with element 1, all nodes, and component 33. CT can contain rows both with and without INF. PARTITION will not issue a warning if not all of the entries were found.

The optional input string `'veconly'` tells PARTITION to generate the output vector V only, and not create the partitioned result.

V is a logical vector of the same length as the number of rows in the component matrix containing true for rows that were kept when partitioning.

Examples

```
>> d = d.partition([1 2 7]);

>> d = d.partition(imat_ctrace('1x'));

>> [d,v] = d.partition({[1 inf 33; 2 2 22]});

>> [~,v] = d.partition([1 2 7],'veconly');
```

---

imat_result_dap/partition

Purpose

Partition Data At Points results.

Syntax

```
d=partition(d,ct)
d=d.partition(ct)
[d,v]=partition(d,ct)
[~,v]=partition(d,ct,'veconly')
```

Description

PARTITION partitions results using the input partitioning data CT. When CT is a numeric or logical vector, PARTITION will partition the results to the indices specified in CT. This is the same as subscripting the object.

When CT is a cell array containing numeric MxN matrix, PARTITION attempts to match the first N rows in this matrix to the first N rows in the component matrix. To match all of the entries in a column, use INF for that column. For example,

```
CT = {[1; 2]}
```

specifies that PARTITION should match all components with points 1 and 2. CT can contain rows both with and without INF. PARTITION will not issue a warning if not all of the entries were found.

The optional input string `'veconly'` tells PARTITION to generate the output vector V only, and not create the partitioned result.

V is a logical vector of the same length as the number of rows in the component matrix containing true for rows that were kept when partitioning.

Examples

```
>> d = d.partition([1 2 7]);

>> [d,v] = d.partition({[1; 2]});

>> [~,v] = d.partition([1 2 7],'veconly');
```

---

imat_result_doe/partition

Purpose

Partition Data On Elements results.

Syntax

```
d=partition(d,ct)
d=d.partition(ct)
[d,v]=partition(d,ct)
[~,v]=partition(d,ct,'veconly')
```

Description

PARTITION partitions results using the input partitioning data CT. When CT is a numeric or logical vector, PARTITION will partition the results to the indices specified in CT. This is the same as subscripting the object.

When CT is an imat_group, the result will be partitioned to the elements specified in the group. If not all of the elements in the group were found, PARTITION will issue a warning.

When CT is a cell array containing numeric MxN matrix, PARTITION attempts to match the first N rows in this matrix to the first N rows in the component matrix. To match all of the entries in a column, use INF for that column. For example,

```
CT = {[1 inf 300]}
```

specifies that PARTITION should match all components with element 1, all layers, and SEID 300. CT can contain rows both with and without INF. PARTITION will not issue a warning if not all of the entries were found.

The optional input string `'veconly'` tells PARTITION to generate the output vector V only, and not create the partitioned result.

V is a logical vector of the same length as the number of rows in the component matrix containing true for rows that were kept when partitioning.

Examples

```
>> d = d.partition([1 2 7]);

>> [d,v] = d.partition({[1 inf 300; 2 2 0]});

>> [~,v] = d.partition([1 2 7],'veconly');
```

---

imat_result_doean/partition

Purpose

Partition Data On Elements At Nodes results.

Syntax

```
d=partition(d,ct)
d=d.partition(ct)
[d,v]=partition(d,ct)
[~,v]=partition(d,ct,'veconly')
```

Description

PARTITION partitions results using the input partitioning data CT. When CT is a numeric or logical vector, PARTITION will partition the results to the indices specified in CT. This is the same as subscripting the object.

When CT is an imat_ctrace, the result will be partitioned using the nodes and directions (component locations) in CT. If not all of the entities in the coordinate trace were found, PARTITION will issue a warning.

When CT is an imat_group, the result will be partitioned to the nodes and elements specified in the group. If not all of the entities in the group were found, PARTITION will issue a warning.

When CT is a cell array containing numeric MxN matrix, PARTITION attempts to match the first N rows in this matrix to the first N rows in the component matrix. To match all of the entries in a column, use INF for that column. For example,

```
CT = {[1 inf 3]}
```

specifies that PARTITION should match all components with node 1, all elements, and component 3. CT can contain rows both with and without INF. PARTITION will not issue a warning if not all of the entries were found.

The optional input string `'veconly'` tells PARTITION to generate the output vector V only, and not create the partitioned result.

V is a logical vector of the same length as the number of rows in the component matrix containing true for rows that were kept when partitioning.

Examples

```
>> d = d.partition([1 2 7]);

>> d = d.partition(imat_ctrace('1x'));

>> [d,v] = d.partition({[1 inf 3; 2 2 0]});

>> [~,v] = d.partition([1 2 7],'veconly');
```

---

imat_result_dan/setComponents

Purpose

Set the components for Data At Nodes result.

Syntax

```
d=setComponents(d,node,dir,seid)
d=setComponents(d,node,dir,seid,compressed,data)
```

Description

SETCOMPONENTS sets the component vectors and optionally data for the Data At Nodes result object. The individual components can be in compressed or uncompressed format. If uncompressed, SETCOMPONENTS will attempt to compress them.

NODE contains a vector of node IDs.

DIR contains a vector of component directions.

SEID contains a vector of superelement IDs.

COMPRESSED is an optional input logical specifying whether the component vectors are compressed. The default is FALSE.

DATA is an optional input vector containing the data values for this result.

Examples

```
>> d = setComponents(d,1:100,1:6,0,true,1:600);
```

See Also

imat_result_dan/getComponents

---

imat_result_doe/setComponents

Purpose

Set the components for Data On Elements result.

Syntax

```
d=setComponents(d,element,layer,seid)
d=setComponents(d,element,layer,seid,compressed,data)
```

Description

SETCOMPONENTS sets the component vectors and optionally data for the Data On Elements result object. The individual components can be in compressed or uncompressed format. If uncompressed, SETCOMPONENTS will attempt to compress them.

ELEMENT contains a vector of element IDs.

LAYER contains a vector of layer IDs.

SEID contains a vector of superelement IDs.

COMPRESSED is an optional input logical specifying whether the component vectors are compressed. The default is FALSE.

DATA is an optional input vector containing the data values for this result.

Examples

```
>> d = setComponents(d,1:100,1,0,true,1:100);
```

See Also

imat_result_doe/getComponents

---

imat_result_danoe/setComponents

Set the components for Data At Nodes On Elements result.

Syntax

```
d=setComponents(d,node,dir,seid)
d=setComponents(d,node,dir,seid,compressed,data)
```

Description

SETCOMPONENTS sets the component vectors and optionally data for the Data At Nodes result object. The individual components can be in compressed or uncompressed format. If uncompressed, SETCOMPONENTS will attempt to compress them.

NODE contains a vector of node IDs.

DIR contains a vector of component directions.

SEID contains a vector of superelement IDs.

COMPRESSED is an optional input logical specifying whether the component vectors are compressed. The default is FALSE.

DATA is an optional input vector containing the data values for this result.

Examples

```
>> d = setComponents(d,1:100,1:6,0,true,1:600);
```

See Also

imat_result_danoe/getComponents

imat_result_doean/setComponents

Purpose

Set the components for Data On Elements At Nodes result.

Syntax

```
d=setComponents(d.node,element,comp,eltype)
d=setComponents(d,node,element,comp,eltype,compressed,data)
```

Description

SETCOMPONENTS sets the component vectors and optionally data for the Data At Nodes result object. The individual components can be in compressed or uncompressed format. If uncompressed, SETCOMPONENTS will attempt to compress them.

NODE contains the node IDs, which are stored once per node if the vectors are compressed.

ELEMENT contains the element IDs which are stored explicitly, once per node (compressed).

COMP stores the components just once. This means that to be compressed, all of the results in this object must have the same component list.

ELTYPE contains the element types, and must be the same length as ELEMENT.

COMPRESSED is an optional input logical specifying whether the component vectors are compressed. The default is FALSE.

DATA is an optional input vector containing the data values for this result.

Examples

```
>> d = setComponents(d,11:12,1:2,[1 2 3 4 5 6],[34;34],true,1:12);
```

See Also

imat_result_doean/getComponents

---

imat_result_dap/setComponents

Purpose

Set the components for Data At Points result.

Syntax

```
d=setComponents(d,point,inc,nvals)
d=setComponents(d,point,inc,nvals,compressed,data)
```

Description

SETCOMPONENTS sets the component vectors and optionally data for the Data At Nodes result object. The individual components can be in compressed or uncompressed format. If uncompressed, SETCOMPONENTS will attempt to compress them.

POINT contains a vector of point IDs. If the component list is compressed, POINT contains just the first point ID.

INC contains the point ID increment if compressed, or empty if uncompressed.

NVALS contains the number of point values if compressed, or empty if uncompressed.

COMPRESSED is an optional input logical specifying whether the component vectors are compressed. The default is FALSE.

DATA is an optional input vector containing the data values for this result.

Examples

```
>> d = setComponents(d,1:100,[],[],false,1:100);
```

See Also

imat_result_dap/getComponents

IMAT Methods for `imat_fem` objects

---

imat_fem	Create a default IMAT_FEM object
imat_cs	Create an IMAT_CS coordinate system object.
imat_node	Create an IMAT_NODE node object.
imat_elem	Create an IMAT_ELEM element object.
imat_tl	Create an IMAT_TL traceline object.
imat_cs/add imat_node/add imat_elem/add imat_tl/add	Add coordinate systems.

imat_cs/keep imat_node/keep imat_elem/keep imat_tl/keep	Keep entities.
imat_cs/remove imat_node/remove imat_elem/remove imat_tl/remove	Remove entities.
imat_fem/plus imat_cs/plus imat_node/plus imat_elem/plus imat_tl/plus	Add (concatenate) entities.
imat_fem/minus imat_cs/minus imat_node/minus imat_elem/minus imat_tl/minus	Subtract one entity from another.
imat_fem/and imat_cs/and imat_node/and imat_elem/and imat_tl/and	Boolean AND.
imat_fem/or imat_cs/or imat_node/or imat_elem/or imat_tl/or	Boolean OR.
imat_elem/elemtype_f2i	Return I-deas element type for given FEMAP element type and topology.
imat_elem/elemtype_f2n	Return Nastran element type for given FEMAP element type.
imat_elem/elemtype_n2f	Return FEMAP element type for given Nastran element type.
imat_elem/elemtype_i2n	Return Nastran element type for given I-deas element type.
imat_elem/elemtype_n2i	Return I-deas element type for given Nastran element type.
imat_elem/convertDegenerateToLinear	Convert degenerate elements to linear

imat_fem/renumber	Renumber an IMAT_FEM using the supplied increments
imat_tl/to_2node	Convert traceline into 2-node tracelines
imat_fem/cat	Concatenate multiple IMAT_FEM objects into a single one
imat_fem/partition	Partition an IMAT_FEM to the supplied DOF list
imat_fem/plotcs	Display IMAT_FEM coordinate systems
imat_fem/vtkplotcs	Display IMAT_FEM coordinate systems using VTK
imat_fem/validate	Validates an IMAT_FEM object
imat_fem/labelnodes	Display node labels on FEM geometry
imat_fem/plot	Plot an IMAT_FEM in 3D space
imat_fem/xform	Coordinate transform node coordinates

imat_fem/imat_fem

Purpose

Create a default IMAT_FEM object.

Syntax

```
fem=imat_fem
fem=imat_fem([])
fem=imat_fem(cs,node,elem,tl)
fem=imat_fem(cs,node)
```

Description

The IMAT_FEM class contains FEM coordinate system, node, element, and traceline information. It provides a number of methods for working with the data.

IMAT_FEM with no arguments will create an empty IMAT_FEM object with the global cartesian coordinate system defined.

IMAT_FEM([]) will create an empty IMAT_FEM object with no coordinate systems.

IMAT_FEM(S), where S is a structure containing fields with the same name as the properties of IMAT_FEM will convert the structure into an IMAT_FEM object.

IMAT_FEM([CS][,NODE][,ELEM][,TL]) where the inputs are IMAT_CS, IMAT_NODE, IMAT_ELEM, and IMAT_TL objects, respectively, will return an IMAT_FEM object containing these inputs.

The IMAT_FEM object contains several properties that define it:

`.cs`	IMAT_CS containing coordinate system information
`.node`	IMAT_NODE containing node information
`.elem`	IMAT_ELEM containing element information
`.tl`	IMAT_TL containing traceline information

See Also

imat_cs, imat_node, imat_elem, imat_tl, imat_fem/partition, imat_fem/validate

---

imat_cs/imat_cs

Purpose

Create an IMAT_CS coordinate system object.

Syntax

```
cs=imat_cs
cs=imat_cs([])
cs=imat_cs(s)
cs=imat_cs(id,cs,name,color,type,matrix)
```

Description

The IMAT_CS class contains FEM coordinate system information. It provides a number of methods for working with the data.

IMAT_CS with no arguments will create an object with the global cartesian system defined.

IMAT_CS(0) or IMAT_CS([]) will create an empty object.

IMAT_CS(S), where S is a structure containing fields with the same name as the properties of IMAT_CS will convert the structure into an IMAT_CSobject.

IMAT_CS(ID,...) where the input argument are the same as the ADD method, will create an IMAT_CS with the node information added.

The IMAT_CS object contains several properties that define the coordinate systems:

`.part`	1x2 cell array containing an integer and a string. This is used for export to Universal files.

`.id`	Coordinate system ID (list must be unique and positive)
`.name`	Cell array of strings containing the coordinate system names
`.color`	Vector of coordinate system colors (non-negative). The colors follow the I-deas convention.
`.type`	Coordinate system type 0 = cartesian 1 = cylindrical 2 = spherical
`.matrix`	4x3xN matrix containing the transformation matrices. The upper 3x3 contains the transformation matrix, and the 4th row contains the origin of this coordinate system in global coordinates.

The transformation matrix is defined as the transformation from global

to local coordinates:

```
          COORD_LOCAL = T * (COORD_GLOBAL - OFFSET)
```

where `T` is the upper 3x3 of .matrix, and offset is the 4th row of `.matrix`.

See Also

imat_fem, imat_node, imat_elem, imat_tl, imat_fem/partition, imat_fem/validate

---

imat_node/imat_node

Purpose

Create an IMAT_NODE node object.

Syntax

```
          node=imat_node
          node=imat_node(s)
          node=imat_node(id,cs,color,coord)
          node=imat_node(id)
```

Description

The IMAT_NODE class contains FEM node information. It provides a number of methods for working with the data.

IMAT_NODE with no arguments will create an empty IMAT_NODE object.

IMAT_NODE(S), where S is a structure containing fields with the same name as the properties of IMAT_NODE will convert the structure into an IMAT_NODE object.

IMAT_NODE(ID,...) where the input argument are the same as the ADD method, will create an IMAT_NODE with the node information added.

The IMAT_NODE object contains several properties that define the nodes:

`.id`	Node IDs (list must be unique and positive)
`.cs`	Nx3 matrix of coordinate system IDs. The first column is the definition (reference) coordinate system, the second is the displacement coordinate system, and the 3rd is the IMAT_NODE storage coordinate system. This is the coordinate system in which the node coordinates in the IMAT_NODE object are stored.
`.color`	Vector of node colors (non-negative). The colors follow the I-deas convention.
`.coord`	Nx3 matrix of coordinates, which are in the coordinate system(s) defined by column 3 of the `.cs` property.

See Also

imat_fem, imat_cs, imat_elem, imat_tl, imat_fem/partition, imat_fem/validate

---

imat_elem/imat_elem

Purpose

Create an IMAT_ELEM element object.

Syntax

```
elem=imat_elem
elem=imat_elem(s)
elem=imat_elem(id,type,conn,color,prop,beamdata)
elem=imat_elem(id,type,conn)
```

Description

The IMAT_ELEM class contains FEM element information. It provides a number of methods for working with the data.

IMAT_ELEM with no arguments will create an empty IMAT_ELEM object.

IMAT_ELEM(S), where S is a structure containing fields with the same name as the properties of IMAT_ELEM will convert the structure into an IMAT_ELEM object.

IMAT_ELEM(ID,...) where the input argument are the same as the ADD method, will create an IMAT_ELEM with the element information added.

The IMAT_ELEM object contains several properties that define the elements:

`.id`	Element IDs (list must be unique and positive)
`.type`	Vector of element types (non-negative). The type follows the Nastran convention.
`.color`	Vector of node colors (non-negative). The colors follow the I-deas convention.
`.prop`	Nx2 matrix of property IDs. IMAT does not use this information; it is stored here for round-trip completeness with Universal files. The first column contains the physical property ID and the second contains the material property ID.
`.beamdata`	Nx3 matrix of beam data IDs. IMAT does not use this information; it is stored here for round-trip completeness with Universal files. For non-beam elements, the entries will be 0. For beam elements, the first column contains the beam orientation node number. The second column contains the fore end beam cross-section, and the third column contains the aft end ID.
`.conn`	Cell array of numeric vectors containing the node IDs defining the element connectivity.

See Also

imat_fem, imat_cs, imat_node, imat_tl, imat_fem/partition, imat_fem/validate

---

imat_tl/imat_tl

Purpose

Create an IMAT_TL traceline object.

Syntax

```
tl=imat_tl
tl=imat__tl(s)
tl=imat__tl(id,conn,color,desc)
tl=imat__tl(id,conn)
```

Description

The IMAT_TL class contains FEM traceline information. It provides a number of methods for working with the data.

IMAT_TL with no arguments will create an empty IMAT_TL object.

IMAT_TL(S), where S is a structure containing fields with the same name as the properties of IMAT_TL will convert the structure into an IMAT_TL object.

IMAT_TL(ID,...) where the input argument are the same as the ADD method, will create an IMAT_TL with the traceline information added.

The IMAT_TL object contains several properties that define the tracelines:

`.id`	Traceline IDs (list must be unique and positive)
`.color`	Vector of node colors (non-negative). The colors follow the I-deas convention.
`.desc`	Cell array of strings containing the traceline descriptions.
`.conn`	Cell array of numeric vectors containing the node IDs defining the traceline connectivity. This connectivity list can contain 0's, which indicate that the traceline should not draw connectivity between the previous node and the next node.

See Also

imat_fem, imat_cs, imat_node, imat_elem, imat_tl, imat_fem/partition, imat_fem/validate

---

imat_cs/add

Purpose

Add coordinate systems.

Syntax

```
cs=add(cs,id,name,color,type,matrix)
cs=cs.add(id,name)
cs=cs.add(id,name,color)
cs=cs.add(id,name,color,type)
cs=cs.add(newcs)
```

Description

ADD will add a coordinate system or coordinate systems to the IMAT_CS object.

ID is a vector containing the coordinate system IDs. They must be unique.

NAME is a cell array of strings containing the coordinate system names. If not supplied, or it is empty, the default (empty string) will be used.

COLOR is a numeric vector containing the coordinate system colors. If not supplied, or it is empty, the default color of 12 is used.

TYPE is a numeric vector containing the coordinate system types. If not supplied, or it is empty, the default (0 = cartesian) will be used.

MATRIX is a 4x3xN matrix containing the transformation matrices. If not supplied, or it is empty, the default transformation will be used.

An alternate calling sequence is to supply an IMAT_CS object NEWCS to add.

See Also

imat_cs/keep, imat_cs/remove

---

imat_node/add

Purpose

Add nodes.

Syntax

```
node=add(node,id,cs,color,coord)
node=node.add(id,cs)
node=node.add(id,cs,color)
node=node.add(newnode)
```

Description

ADD will add a node or nodes to the IMAT_NODE object.

ID is a vector containing the node IDs. They must be unique.

CS is an Nx3 matrix containing the coordinate system IDs. If not supplied, or it is empty, the default ([0 0 0]) will be used.

COLOR is a numeric vector containing the node colors. If not supplied, or it is empty, the default color of 11 is used.

COORD is an Nx3 matrix containing the node coordinates. If not supplied, or it is empty, the global origin will be used.

An alternate calling sequence is to supply an IMAT_NODE object NEWNODE to add.

See Also

imat_node/keep, imat_node/remove

---

imat_elem/add

Purpose

Add elements.

Syntax

```
elem=add(elem,id,type,conn,color,prop,beamdata)
elem=elem.add(id,type,conn)
elem=elem.add(id,type,conn,color)
elem=elem.add(id,type,conn,color,prop)
elem=elem.add(newelem)
```

Description

ADD will add an element or elements to the IMAT_ELEM object.

ID is a vector containing the element IDs. They must be unique.

TYPE is a vector containing the element type(s).

CONN is a cell array of vectors containing the element node connectivity.

COLOR is a numeric vector containing the element colors. If not supplied, or it is empty, the default color of 7 is used.

PROP is an Nx2 matrix containing the material and physical property IDs. If not supplied, they will be set to 0.

BEAMDATA is an Nx3 matrix containing the beam cross-section references. If not supplied, they will be set to 0.

An alternate calling sequence is to supply an IMAT_ELEM object NEWELEM to add.

See Also

imat_elem/keep, imat_elem/remove

---

imat_tl/add

Purpose

Add tracelines.

Syntax

```
tl=add(tl,id,conn,color,desc)
tl=tl.add(id,conn)
tl=tl.add(id,conn,color)
tl=tl.add(tlnew)
```

Description

ADD will add a traceline or tracelines to the IMAT_TL object.

ID is a vector containing the element IDs. They must be unique.

CONN is a cell array of vectors containing the traceline node connectivity.

COLOR is a numeric vector containing the traceline colors. If not supplied, or it is empty, the default color of 1 is used.

DESC is a cell array of strings containing the traceline descriptions.

An alternate calling sequence is to supply an IMAT_TL object NEWTL to add.

See Also

imat_tl/keep, imat_tl/remove

---

imat_cs/keep

Purpose

Keep coordinate systems.

Syntax

```
cs=keep(cs,1:10)
cs=cs.keep(1:10)
```

Description

KEEP will keep coordinate systems from the IMAT_CS object CS. If ID is numeric, it must contain the coordinate system ID(s) to keep. If it is logical, it contains the indices into CS of the coordinate system(s) to keep.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 tracelines

>> cs=fem.cs;
>> cs=cs.keep(1:3);
>> cs=keep(cs,node.id==3);
```

See Also

imat_cs/add, imat_cs/remove

---

imat_node/keep

Purpose

Keep nodes.

Syntax

```
node=keep(node,1:10)
node=node.keep(1:10)
```

Description

KEEP will keep nodes from the IMAT_NODE object NODE. If ID is numeric, it must contain the node ID(s) to keep. If it is logical, it contains the indices into NODE of the node(s) to keep.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
          1 coordinate system
         30 nodes
         10 elements
          3 tracelines

>> node=fem.node;
>> node=node.keep(1:3);
>> node=keep(node,node.id==3);
```

See Also

imat_node/add, imat_node/remove

---

imat_elem/keep

Purpose

Keep elements.

Syntax

```
elem=keep(elem,1:10)
elem=elem.keep(1:10)
```

Description

KEEP will keep elements from the IMAT_ELEM object ELEM. If ID is numeric, it must contain the element ID(s) to keep. If it is logical, it contains the indices into ELEM of the element(s) to keep.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 tracelines

>> elem=fem.elem;
>> elem=elem.keep(1:3);
>> elem=keep(elem,node.id==3);
```

See Also

imat_elem/add, imat_elem/remove

---

imat_tl/keep

Purpose

Keep tracelines.

Syntax

```
tl=keep(tl,1:10)
tl=tl.keep(1:10)
```

Description

KEEP will keep tracelines from the IMAT_TL object TL. If ID is numeric, it must contain the traceline ID(s) to keep. If it is logical, it contains the indices into TL of the traceline(s) to keep.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 tracelines

>> tl=fem.tl;
>> tl=tl.keep(1:2);
>> tl=keep(tl,tl.id==1);
```

See Also

imat_tl/add, imat_tl/remove

---

imat_cs/remove

Purpose

Remove coordinate systems.

Syntax

```
cs=remove(tl,1:10)
cs=cs.remove(1:10)
```

Description

REMOVE will remove coordinate systems from the IMAT_CS object CS. If ID is numeric, it must contain the coordinate system ID (s) to remove. If it is logical, it contains the indices into CS of the coordinate system(s) to remove.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 tracelines

>> cs=fem.cs;
>> cs=cs.remove(1);
>> cs=remove(cs,cs.id==1);
```

See Also

imat_cs/add, imat_cs/keep

---

imat_node/remove

Purpose

Remove nodes.

Syntax

```
node=remove(node,1:10)
node=node.remove(1:10)
```

Description

REMOVE will remove nodes from the IMAT_NODE object NODE. If ID is numeric, it must contain the node ID(s) to remove. If it is logical, it contains the indices into NODE of the node(s) to remove.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 tracelines

>> node=fem.node;
>> nodenode.remove(1:2);
>> node=remove(tl,tl.id==1);
```

See Also

imat_node/add, imat_node/keep

---

imat_elem/remove

Purpose

Remove elements.

Syntax

```
elem=remove(elem,1:10)
elem=elem.remove(1:10)
```

Description

REMOVE will remove elements from the IMAT_ELEM object ELEM. If ID is numeric, it must contain the element ID(s) to remove. If it is logical, it contains the indices into ELEM of the element(s) to remove.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 tracelines

>> elem=fem.elem;
>> elem=elem.remove(1:2);
>> elem=remove(elem,elem.id==1);
```

See Also

imat_elem/add, imat_elem/keep

---

imat_tl/remove

Purpose

Remove tracelines.

Syntax

```
tl=remove(tl,1:10)
tl=tl.remove(1:10)
```

Description

REMOVE will remove tracelines from the IMAT_TL object TL. If ID is numeric, it must contain the traceline ID(s) to remove. If it is logical, it contains the indices into TL of the traceline(s) to remove.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
        30 nodes
        10 elements
         3 tracelines

>> tl=fem.tl;
>> tl=tl.remove(1:2);
>> tl=remove(tl,tl.id==1);
```

See Also

imat_tl/add, imat_tl/keep

---

imat_fem/plus

imat_cs/plus, imat_node/plus, imat_elem/plus, imat_tl/plus

Purpose

Add (concatenate) FEM objects.

Syntax

```
fem=plus(fem1,fem2)
fem=fem1+fem2
```

Description

PLUS performs the equivalent of concatenating two IMAT_FEM objects.

See Also

imat_fem/cat

---

imat_fem/minus

imat_cs/minus, imat_node/minus, imat_elem/minus, imat_tl/minus

Purpose

Subtract FEM objects.

Syntax

```
fem=minus(fem1,fem2)
fem=fem1-fem2
```

Description

MINUS performs the equivalent of removing the FEM entities from the IMAT_FEM object B from the IMAT_FEM contained in A. It matches entities by ID only.

See Also

imat_cs/remove, imat_node/remove, imat_elem/remove, imat_tl/remove

---

imat_fem/and

imat_cs/and, imat_node/and, imat_elem/and, imat_tl/and

Purpose

Boolean AND.

Syntax

```
fem=and(fem1,fem2)
fem=fem1&fem2
```

Description

AND returns the IMAT_FEM based on the input IMAT_FEM A which has entities common to both A and B. It matches by ID only.

See Also

imat_fem/or

---

imat_fem/or

imat_cs/or, imat_node/or, imat_elem/or, imat_tl/or

Purpose

Boolean OR.

Syntax

```
fem=or(fem1,fem2)
fem=fem1|fem2
```

Description

OR returns an IMAT_FEM that contains the FEM entities in A and the FEM entities that are in B that are not in A. It matches by ID only.

See Also

imat_fem/and

---

imat_fem/cat

imat_cs/cat, imat_node/cat, imat_elem/cat, imat_tl/cat

Purpose

Concatenate multiple IMAT_FEM objects into a single one.

Syntax

```
fout=cat(fem,fem2[,fem3][,fem4][,...])
```

Description

CAT will concatenate multiple IMAT_FEM objects into a single one. It will check each FEM entity (coordinate system, node, element, and traceline) for duplicate ids (labels). If any duplicates are found, and they are not an exact match (in other words, any of the attributes of that entity are different between the ones with matching labels), an error will result and an empty IMAT_FEM will be returned. If the entities are an exact match, the duplicates will be discarded.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
       527 nodes
       675 elements
         0 tracelines

>> fem2=cat(fem,fem,fem);     % The 2nd and 3rd fems are duplicates of the 1st

fem2 =

IMAT_FEM Finite Element Model
         1 coordinate system
       527 nodes
       675 elements
         0 tracelines

>> whos

Name        Size          Bytes  Class
fem         1x1          164452  imat_fem
fem2        1x1          164452  imat_fem

>> fem2=readunv('fem_file2.unv')

fem2 =

IMAT_FEM Finite Element Model
         1 coordinate system
       192 nodes
        46 elements
         0 tracelines

>> fem3=cat(fem,fem2);     % No duplicate entities in the two models
>> fem3

fem2 =

IMAT_FEM Finite Element Model
         1 coordinate system
       719 nodes
       721 elements
         0 tracelines
```

See Also

imat_fem/partition, imat_fem/validate

---

imat_fem/partition

imat_node/partition, imat_elem/partition, imat_tl/partition

Purpose

Partition an IMAT_FEM to the supplied DOF list.

Syntax

```
fem2=partition(fem,doflist,'silent')
```

Description

PARTITION will partition an IMAT_FEM using the supplied partitioning IDs.

DOFLIST is either an IMAT_CTRACE, numeric vector containing node IDs, or an IMAT_GROUP containing at minimum node IDs.

If DOFLIST is an IMAT_CTRACE or node list, if the FEM does not contain all of the nodes specified in the DOF list, only those that do match the nodes contained in the DOF list will be returned in the partitioned FEM. Elements and tracelines will be partitioned down to only those entities that are completely defined using the partitioned node list.

If DOFLIST is an IMAT_GROUP, PARTITION will first partition the FEM to the nodes in the group, keeping only the elements and tracelines that are fully defined by those nodes. If elements, tracelines, or coordinate systems are also in the group, it will partition the FEM to those entities as well.

Passing in the string `'silent'` will suppress message displays.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
       527 nodes
       675 elements
         0 tracelines

>> fem2=fem.partition(fem.node.id(1:50))    % Partition down to the first 50 nodes
Partitioning nodes
Partitioning elements

fem2 =

IMAT_FEM Finite Element Model
         1 coordinate system
       477 nodes
       519 elements
         0 tracelines

>>
```

See Also

imat_fem/cat, imat_fem/validate

---

imat_fem/renumber

Purpose

Renumber an IMAT_FEM using the supplied increments.

Syntax

```
g=renumber(fem,ics,ind,iel,itl)
```

Description

RENUMBER will renumber a FEM based on the supplied increments. ICS is the coordinate system increment. IND is the node increment. IEL is the element increment. ITL is the traceline increment. Passing in 0 or an empty matrix for any of these will case that particular entity type to not be renumbered.

Any of the increment variables may also be a vector of the same length as the number of entities.

Examples

```
>> g=renumber(fem,1,100,[],0)     % Renumber the coordinate systems and nodes by a scalar increment

g =

IMAT_FEM Finite Element Model
         1 coordinate system
       527 nodes
       675 elements
         0 tracelines

>> g=renumber(fem,1,100)          % Same as above

fem2 =

IMAT_FEM Finite Element Model
         1 coordinate system
       527 nodes
       675 elements
         0 tracelines

>> g=renumber(fem,0,1:527)          % Renumber nodes using a vector of increasing increments

fem2 =

IMAT_FEM Finite Element Model
         1 coordinate system
       527 nodes
       675 elements
         0 tracelines
```

---

imat_tl/to_2node

Purpose

Convert traceline into 2-node tracelines.

Syntax

```
tl=to_2node(tl)
```

Description

TO_2NODE processes the supplied tracelines into a series of 2-node tracelines. This essentially converts them into Nastran PLOTEL-style line segments.

imat_elem/elemtype_f2i

Purpose

Return I-deas element type for given FEMAP element type and topology.

Syntax

```
ntype=imat_elem.elemtype_f2i(13,4)
ntype=imat_elem.elemtype_f2i(ftype,ftopo)
```

Description

ELEMTYPE_F2I returns a corresponding I-deas element type for the supplied FEMAP element type(s). This method is still available for completeness, since IMAT uses Nastran element type numbering. Please note that in many cases a FEMAP element type could map to more than one I-deas element type, and some FEMAP element types do not have a corresponding I-deas element type. To see the element type mapping, please look at the source code for this method.

FTYPE and FTOPO are a vector of FEMAP element types and their accompanying topologies, respectively. They must be the same length.

ITYPE is a corresponding vector of I-deas element types. If there is no corresponding element type, ITYPE is -1.

Note that since this is a static method of IMAT_ELEM, to call it you must prefix it with the class name, as in

```
>> imat_elem.elemtype_f2i(13,4)
```

See Also

imat_elem/elemtype_n2i

---

imat_elem/elemtype_f2n

Purpose

Return NASTRAN element type for given FEMAP element type.

Syntax

```
ntype=imat_elem.elemtype_f2n(17)
ntype=imat_elem.elemtype_f2n(ftype)
```

Description

ELEMTYPE_F2I returns corresponding IMAT element types for the supplied FEMAP element types. IMAT_ELEM uses the NASTRAN element type numbering. This function can be useful for mapping between NASTRAN and FEMAP. Please note that in many cases a

FEMAP element type could map to more than one NASTRAN element type, and some FEMAP element types do not have a corresponding NASTRAN element type. To see the element type mapping, please look at the source code for this method.

FTYPE and FTOPO are a vector of FEMAP element types and their accompanying topologies, respectively. They must be the same length.

NTYPE is a corresponding vector of NASTRAN element types, which are the IMAT element types. If there is no corresponding element type, NTYPE is -1.

Note that since this is a static method of IMAT_ELEM, to call it you must prefix it with the class name, as in

```
>> imat_elem.elemtype_f2n(94)
```

See Also

imat_elem/elemtype_n2f

---

imat_elem/elemtype_n2f

Purpose

Return FEMAP element type for given NASTRAN element type.

Syntax

```
ftype=imat_elem.elemtype_n2f(33)
ftype=imat_elem.elemtype_n2f(ntype)
```

Description

ELEMTYPE_N2F returns corresponding FEMAP element types for the supplied NASTRAN element types. IMAT_ELEM uses the NASTRAN element type numbering. This function can be useful for mapping between NASTRAN and FEMAP. Please note that in many cases a NASTRAN element type could map to more than one FEMAP element type, and some NASTRAN element types do not have a corresponding FEMAP element type. To see the element type mapping, please look at the source code for this method.

NTYPE is a vector of NASTRAN element types, which are the IMAT element types.

FTYPE is a corresponding vector of FEMAP element types. If there is no corresponding element type, FTYPE is -1.

Note that since this is a static method of IMAT_ELEM, to call it you must prefix it with the class name, as in

```
>> imat_elem.elemtype_n2f(33)
```

See Also

imat_elem/elemtype_f2n

---

imat_elem/elemtype_i2n

Purpose

Return NASTRAN element type for given I-deas element type.

Syntax

```
ntype=imat_elem.elemtype_i2n(94)
ntype=imat_elem.elemtype_i2n(itype)
```

Description

ELEMTYPE_I2N returns a corresponding NASTRAN element type(s) for the supplied I-deas element type(s). IMAT_ELEM uses NASTRAN element type numbering, and results imported from Universal files use the I-deas element type numbering. This function can be useful for mapping between the two. Please note that in many cases an I-deas element type could map to more than one NASTRAN element type, and some I-deas element types do not have a corresponding NASTRAN element type.

ITYPE is a vector of I-deas element types.

NTYPE is a corresponding vector of NASTRAN element types, which are the IMAT element types. If there is no corresponding element type, NTYPE returns -1.

Note that since this is a static method of IMAT_ELEM, to call it you must prefix it with the class name, as in

```
>> imat_elem.elemtype_i2n(94)
```

See Also

imat_elem/elemtype_n2i

---

imat_elem/elemtype_n2i

Purpose

Return I-deas element type for given NASTRAN element type.

Syntax

```
itype=imat_elem.elemtype_n2i(33)
itype=imat_elem.elemtype_n2i(ntype)
```

Description

ELEMTYPE_N2I returns a corresponding I-deas element type(s) for the supplied NASTRAN element type(s). IMAT_ELEM uses Nastran element type numbering, and results imported from Universal files use the I-deas element type numbering. This method can be useful for mapping between the two. Please note that in many cases a NASTRAN element type could map to more than one I-deas element type, and some NASTRAN element types do not have a corresponding I-deas element type.

NTYPE is a vector of NASTRAN element types, which are the IMAT element types.

ITYPE is a corresponding vector of I-deas element types. If there is no corresponding element type, ITYPE returns -1.

Note that since this is a static method of IMAT_ELEM, to call it you must prefix it with the class name, as in

```
>> imat_elem.elemtype_n2i(33)
```

See Also

imat_elem/elemtype_i2n

---

imat_fem/labelnodes

Purpose

Display node labels on FEM geometry.

Syntax

```
fem_labelnodes(fem)
h=fem_labelnodes(fem,handle,nodes,color)
```

Description

FEM_LABELNODES will display node labels on the supplied IMAT_FEM object FEM. NODES is an optional vector of node labels to plot. If it is not supplied, LABELNODES will display node labels for all of the nodes in FEM. If nodes supplied in NODES are not found in FEM, a warning will be issued and only those node labels found in FEM will be displayed. H is an optional figure or uipanel handle. If it is supplied, node labels will be displayed on the supplied figure/uipanel. Otherwise a new figure will be generated. If a handle referencing a FEM is supplied, then FEM is not a required input. COLOR is an optional argument to specify the node label color. By default, the node color will be used. COLOR may be specified either as a numeric or as a string. If it is a string containing the color name (i.e. `'blue'`), that is all that is required. To specify the color using a number, pass in a string 'Color', followed by the number. A scalar number represents a color in I-DEAS notation (see `ideas colormap`), and a 1x3 vector specifies the color in RGB notation.

HP is the handle to the uipanel on which the FEM node labels were displayed.

Examples

```
>> labelnodes(fem)
>> h=labelnodes(fem,'green');     % Draw nodes labels in green
>>
```

See Also

imat_shp/plot, imat_fem/plot

---

imat_fem/plot

Purpose

Plot finite element model geometry in a special figure window.

Syntax

```
plot(fem)
h=plot(fem,handle,-
form-
at,'shaded','title',TITLESTR,'silent','renderer',RENDERER,'elemlinewidth',WIDTH,'tracelinewidth',W-
IDTH,'backgroundcolor',COLOR,'unitslabel',UNITSLAB)
```

Description

PLOT allows you to display finite element model (FEM) nodes, elements, and tracelines in a figure window. FEM is an IMAT_FEM object. Several display and format options are provided, and are specified by passing in the appropriate string as an argument. If a figure handle is passed in, the FEM will be drawn in that figure. If an axis handle is passed in, the FEM will be drawn on those axes. If you pass in a uipanel handle, the FEM will be displayed in that panel.

The format string is a valid formatting string following MATLAB's builtin PLOT function conventions. A valid marker controls the node display, and a valid linestyle controls the element and traceline display. The default node marker is a dot ('.'), and the default element and traceline linestyle is a solid line ('-'). Node, element, and traceline colors are specified in the corresponding IMAT_FEM object properties. Element and trace line widths can be controlled with the `'elemlinewidth'` and `'tracelinewidth'` options. To turn off the display of either nodes or elements/tracelines, send in a space (' ') as the format string. To turn off element/traceline display, you must also include a valid node marker format, since only one space is permitted in the format string, and the space will default to the node marker format if one is not provided. To generate the display and then turn it off completely, you can send in the string `'off'`. After the plot is generated, View and Display pulldown menus allow for further control of the display.

Most elements are drawn using connected line segments. The `'shaded'` command line option will color the entire element. Lumped mass elements are drawn with a square marker. Node-to-ground springs are drawn with an up triangle. Node-to-ground dampers are drawn with a down triangle. Node-to-ground gaps are drawn with a diamond.

A pulldown menu called 'Display' will appear on the figure menubar that allows you to change the node marker and the element and traceline linestyles, as well as the node marker size and node labels, and the figure renderer. A pulldown menu called 'View' allows you to change the default view to several preset views.

Other valid display and format arguments and their descriptions are shown in the table below.

`'title'`	Must be followed by a string containing the figure title.
`'unitslabel'`	This must be followed by a string containing the units label that will be placed in parentheses after the axis label. Pass in a space (' ') to disable the units label.
`'silent'`	Do not print progress messages when displaying the shape.
`'shaded'`	Display the FEM as a shaded image.
`'deformed'`	Specify that the supplied geometry is deformed (creates a separate FEM entity changer pull-down). Generally this is not used.

`'off'`	Draw the geometry and then turn the display off.
`'renderer',RENDERER`	The default figure renderer is OpenGL. You can specify a different renderer either through the Display menu or by passing in the string `'renderer'` followed by the renderer to use. Valid options are `'zbuffer'` and `'opengl'`.
`'elemlinewidth',WIDTH`	Set the element line width (default 1). Must be a positive number.
`'tracelinewidth',WIDTH`	Set the traceline line width (default 1). Must be a positive number.
`'reversex'`	Reverse the X- and Z- axis directions. You cannot use this with `'reversey'` or `'reversez'`.
`'reversey'`	Reverse the X- and Y- axis directions. You cannot use this with `'reversex'` or `'reversez'`.
`'reversez'`	Reverse the Y- and Z- axis directions. You cannot use this with `'reversex'` or `'reversey'`.
`'backgroundcolor',COLOR`	Set the panel's background color. You can pass in a string or a 1x3 numeric RGB vector.
`'text',COLOR`	Set the axis color. You can pass in a string or a 1x3 numeric RGB vector.
`'noaxes'`	Turn off the axis display

This plotting facility is intended to provide simple viewing of a FEM. Performance is slow for large models. For large models, consider using VTKPLOT.

Examples

```
>> plot(fem)
Working on node display...
Working on element display...
>> h=plot(fem,'*-.','silent');
>> plot(fem,h,'title','This is the title');
Working on node display...
Working on element display...

>>
```

See Also

imat_shp/plot, fem_labelnodes, imat_fem/vtkplot

---

imat_fem/plotcs

Purpose

Display FEM coordinate systems.

Syntax

```
plotcs(fem)
h=plotcs(fem,'nolabels','showfem')
plotcs(fem,'light blue',3)
```

Description

PLOTCS displays the coordinate systems defined in the supplied IMAT_FEM object. By default it will display the coordinate systems in golden orange, and label them. PLOTCS uses imat_ctrace/plot to draw the arrows.

The following optional input strings are supported.

`'nolabels'`	Turn off the coordinate system labels and just plot the arrows.
`'showfem'`	Display the coordinate systems on the FEM. Otherwise, just the coordinate systems will be displayed.

Any other input arguments are passed along toimat_ctrace/plot. Please see the help for this function for options such as specifying the coordinate system color and scale factor.

H is the handle of the uipanel in which the plot is displayed.

For large models, consider using VTKPLOTCS.

See Also

imat_ctrace/plot, imat_fem/vtkplotcs

---

imat_fem/vtkplot (+FEA)

Purpose

Plot and animate FEM geometry in 3-D space using VTK

Syntax

```
vtkplot(fem)
[h,shpout]=plot(shp,fem,-
group,handle,format,complexdisplay,'noundeformed','title',titlestr,'scale',scalefactor,'silent')
```

Description

VTKPLOT generates a 3-D representation of an IMAT_FEM using the Visualization Toolkit libraries. This visualization is very fast and can be further manipulated using the methods on the IMAT_VTKPLOT object, as well as through the toolbar icons and menus. This includes setting the formatting of the model, as well as setting the view and adding clipping planes to better view the model. An image of the model can also be exported to a number of formats, including X3D and VRML.

VTKPLOT expects that FEM information is passed in as an IMAT_FEM object. It returns the handle to the IMAT_VTKPLOT object containing the plot. This object can be manipulated from the command line using its methods. The available methods can be viewed by using the `methods` function, or by reading the help for IMAT_VTKPLOT.

GROUP is an optional IMAT_GROUP containing group information. If supplied, you will be able to display subsets of your FEM based on the elements in the group(s).

If you pass in a valid figure handle in the VTKPLOT function, the FEM will be displayed on the supplied figure. If you pass in a uipanel handle, the FEM will be displayed in that panel.

Node-to-ground springs are drawn with an up triangle. Node-to-ground dampers are drawn with adown triangle. Node-to-ground gaps are drawn with a diamond.

To specify the figure title, pass in the string `'title'`, followed by a string containing the title. If a title is not specified, the variable name for the supplied FEM will be used, if it has a name.

Passing in the string `'silent'` will suppress message displays.

See imat_shp/vtkplot for a list of other supported command line options.

The output argument for VTKPLOT is an IMAT_VTKPLOT object. It contains all of the formatting and contents of the plot. If IDLines are modified during the plotting session, and they need to be retrieved, place the handle to the figure into `uiwait`, and then retrieve the modified `imat_shp` object once the user is done editing it.

Examples

```
>> vtkplot(fem)
>> h = vtkplot(fem,h,'title','The title')
```

See also

imat_shp/vtkplot

---

imat_fem/vtkplotcs (+FEA)

Purpose

Display FEM coordinate systems.

Syntax

```
vtkplotcs(fem)
out=vtkplotcs(fem,'nolabels','showfem')
vtkplotcs(fem,'light blue',3)
```

Description

VTKPLOTCS displays the coordinate systems defined in the supplied IMAT_FEM object.

H is a handle. If you pass in a valid figure handle, the FEM will be displayed on the supplied figure. If you pass in a uipanel handle, the FEM will be displayed in that panel.

You can specify additional input arguments as well. These will simply be passed into CSSTYLE.

The following optional input strings are supported.

`'nolabels'`	Turn off the coordinate system labels and just plot the arrows.
`'showfem'`	Has no effect. Is retained for compatibility with plotcs.

Any other input arguments are passed along toimat_ctrace/plot. Please see the help for this function for options such as specifying the coordinate system color and scale factor.

OUT is the IMAT_VTKPLOT object, which you can use to further modify the display.

See Also

imat_fem/vtkplot, imat_fem/plotcs, imat_vtkplot/csstyle

---

imat_fem/xform

imat_node/xform

Purpose

Coordinate transform node coordinates.

Syntax

```
fem2=xform(fem)
fem2=xform(fem,csto)
fem2=xform(fem,csfrom,csto)
fem2=xform(fem,'silent')
```

Description

XFORM will transform nodal coordinates in the suppliedIMAT_FEM object FEM between two sets of coordinate systems. Coordinate system transformations can be local to global, global to local, or local to local. Transformations can occur between any two arbitrary coordinate systems.

CSFROM and CSTO are coordinate system labels to transform from and to, respectively. They can either be a vector of the same length as the number of nodes, or a scalar. If neither is supplied, the coordinate system(s) to transform from will be extracted from the third column (current node coordinate system) of the `cs` property of the `node` property (IMAT_NODE). The coordinate system (s) to transform to will be extracted from the first column (node definition coordinate system) of the `cs` property of the `node` property. If the nodes are currently in local coordinates, XFORM assumes that you want to transform back to the global. If only CSTO is supplied, the coordinate system to transform from will be extracted from the `node` property. Coordinate system 0 is defined as the global Cartesian coordinate system.

Passing in the string `'silent'` suppresses output to the screen.

XFORM assumes that the angular coordinates for nodes defined in cylindrical or spherical systems are in radians. For cylindrical and spherical coordinate transforms, nodes located at a local origin may produce inaccurate results due to numerical roundoff.

XFORM will return an IMAT_FEM containing the transformed node coordinate data.

Examples

```
>> fem = readunv('simple_fem.unv');
>> fem.node.coord          % Nodal coordinates

ans =
      0     0     0
      1     0     0
      0     1     0
      0     0     1
      1     1     1

>> fem.node.cs             % Nodal coordinate system definitions

ans =
      2     2     0
      2     2     0
      2     2     0
      2     2     0
      2     2     0

>> fem.cs.matrix(:,:,2)    % Transformation matrix for CS 2

ans =
     0.5000   -0.4330    0.7500
          0    0.8660    0.5000
    -0.8660   -0.2500    0.4330
     1.0000    2.0000    3.0000

>> fem2 = xform(fem);
Nodes: Transforming GLOBAL->LOCAL

>> fem2.node.coord

ans =
     -1.8840   -3.2321    0.0670
     -1.3840   -3.2321   -0.7990
     -2.3170   -2.3660   -0.1830
     -1.1340   -2.7321    0.5000
     -1.0670   -1.8660   -0.6160

>> fem3 = xform(fem2,2,0);
Nodes: Transforming LOCAL->GLOBAL

>> fem3.node.coord

ans =
          0         0         0
     1.0000         0         0
          0    1.0000         0
          0         0    1.0000
     1.0000    1.0000    1.0000

>>
```

See Also

imat_shp/xform, imat_fn/xform

---

imat_fem/get_id_by_nid

Purpose

Return element or traceline IDs referenced by the supplied node IDs.

Syntax

```
nid=get_id_by_nid(fem)
nid=get_id_by_nid(fem,id)
nid=get_id_by_nid(fem,id,'elem')
```

Description

GET_NID_BY_ID returns a sorted list of node IDs that make up the connectivity of the IDs supplied. If ID is not supplied, or is empty, all of the entities will be used.

TYPE is an optional string specifying the entity from which to extract the associated IDs. Valid options are `'elem'` (default) and `'tl'`.

---

imat_fem/get_nid_by_id

Purpose

Return node IDs referenced by the supplied IDs.

Syntax

```
nid=get_nid_by_id(fem)
nid=get_nid_by_id(fem,id)
nid=get_nid_by_id(fem,id,'elem')
```

Description

GET_NID_BY_ID returns a sorted list of node IDs that make up the connectivity of the IDs supplied. If ID is not supplied, or is empty, all of the entities will be used.

TYPE is an optional string specifying the entity from which to extract the node IDs. Valid options are `'elem'` (default) and `'tl'`.

---

imat_fem/validate

imat_cs/validate, imat_node/validate, imat_elem/validate, imat_tl/validate

---

Purpose

Validate an IMAT_FEM object

Syntax

```
fem=validate(femin)
[fem,isvalid]=validate(femin,'silent')
[fem,isvalid,msg]=validate(femin)
```

Description

VALIDATE checks the input objects to make sure they are valid, and returns the output in an imat_fem.

FEMIN is an imat_fem to validate.

Passing in the string `'silent'` will suppress output.

FEM is an imat_fem containing the validated FEM. If none of the inputs are valid, FEM will be empty.

ISVALID is an optional output specifying whether the input data was a valid imat_fem.

If MSG is requested, it is a cell array of strings containing messages for what entities are not valid (empty if the object is valid). If not requested, VALIDATE will issue an error message.

Examples

```
>> fem=readunv('fem_file.unv')

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
       527 nodes
       675 elements
         0 tracelines

>> fem=validate(fem)

fem =

IMAT_FEM Finite Element Model
         1 coordinate system
       527 nodes
       675 elements
         0 tracelines

>>
```

IMAT Methods for `imat_group` objects

---

imat_group	Create an IMAT_GROUP object
imat_group/keep	KEEP group entities of the specified type(s).
imat_group/remove	REMOVE group entities of the specified type(s).
imat_group/cat	Concatenate multiple group objects into a single one.
imat_group/extract	Extract entities of the supplied type(s).
imat_group/plus	Add (concatenate) two IMAT_GROUP objects.
imat_group/minus	Subtract one IMAT_GROUP from another.
imat_group/and	Boolean AND.
imat_group/or	Boolean OR.

imat_group/unique	Return unique groups.
imat_group/intersect	Group intersection.
imat_group/union	Group union.
imat_group/setdiff	Group exclusive-or.
imat_group/setxor	Group difference.
imat_group/validate	Validates an IMAT_GROUP object

---

imat_group/imat_group

Purpose

Create an IMAT_GROUP object.

Syntax

```
group=imat_group
group=imat_group(N)
group=imat_group(S)
group=imat_group(T)
```

Description

The IMAT_GROUP class contains FEM group entity information. It provides a number of methods for working with the data.

IMAT_GROUP with no arguments will create a scalar IMAT_GROUP object with no entities. IMAT_GROUP(N), where N is an Nx2 matrix containing the entity information (ID in column 1, type in column 2), will create an IMAT_GROUP with the supplied entities.

IMAT_GROUP(S), where S is a structure containing fields with the same name as the properties of IMAT_GROUP, will convert the structure into an IMAT_GROUP object.

IMAT_GROUP(T), where T is an IMAT_CTRACE, will create a group from the entities in T. Entities with cartesian coordinates (X through RZ) are assumed to be nodes, and all others are assumed to be unknown.

The IMAT_GROUP object contains several properties that define the group entities:

`.id`	Group ID (numeric scalar)
`.name`	Group name (string up to 80 characters)
`.data`	Nx2 matrix of entity IDs (column 1) and their associated entity types (column 2). The rows (combination of ID and entity type) must be unique.
`.type`	Read-only property that returns a structure that defines the numeric data type for each supported entity type

imat_group/keep

Purpose

KEEP group entities of the specified type(s).

Syntax

```
group=keep(group,type)
group=group.keep(group.type.gNode)
```

Description

KEEP will keep entities from the IMAT_GROUP object GROUP that match the types specified in the numeric vector TYPE.

See Also

imat_group/remove

imat_group/remove

Purpose

REMOVE group entities of the specified type(s).

Syntax

```
group=remove(group,type)
group=group.remove(group.type.gNode)
```

Description

REMOVE will remove group entities from the IMAT_GROUP object GROUP that match the types specified in the numeric vector TYPE.

See Also

imat_group/keep

imat_group/cat

Purpose

Concatenate multiple group objects into a single one.

Syntax

```
fout=cat(fem,fem2[,fem3][,fem4][,...]['silent'])
```

Description

CAT will concatenate the supplied groups into a single one. Each GROUP supplied must be a scalar group object. Duplicate entities will be ignored with a warning.

Passing in `'silent'` will suppress output.

---

imat_group/extract

Purpose

Extract entities of the supplied type(s).

Syntax

```
id=extract(group,type)
id=group.extract([group.type.gNode group.type.gElement])
```

Description

EXTRACT will extract the entities of the specified types. TYPE is a numeric vector. EXTRACT only works on scalar group objects.

---

imat_group/plus

Purpose

Add (concatenate) two IMAT_GROUP objects.

Syntax

```
c=plus(a,b)
c=a+b
```

Description

PLUS performs the equivalent of concatenating two IMAT_GROUP objects. Both A and B must be scalar IMAT_GROUP objects.

See Also

imat_group/minus, imat_group/cat

---

imat_group/minus

Purpose

Subtract one IMAT_GROUP from another.

Syntax

```
c=minus(a,b)
c=a-b
```

Description

MINUS removes the entities from the IMAT_GROUP object B from the IMAT_GROUP entities in A. Both A and B must be scalar IMAT_GROUP objects.

See Also

imat_group/plus

---

imat_group/and

Purpose

Boolean AND.

Syntax

```
c=and(a,b)
c=a&b
```

Description

AND returns the IMAT_GROUP based on the input IMAT_GROUP A which has entities common to both A and B. Both A and B must be scalar IMAT_GROUP objects.

See Also

imat_group/or

---

imat_group/or

Purpose

Boolean OR.

Syntax

```
c=or(a,b)
c=a|b
```

Description

OR returns an IMAT_GROUP that contains the entities in A and the entities that are in B that are not in A. Both A and B must be scalar IMAT_GROUP objects.

See Also

imat_group/and

---

imat_group/unique

Purpose

Return unique groups.

Syntax

```
c=unique(a)
c=unique(a,ignorecell)
c=unique(a,{'ignoreid','ignorename'})
```

Description

UNIQUE returns the unique set of groups in the IMAT_GROUP array A.

The optional cell array of strings IGNORECELL can contain the following strings, which specify which properties to ignore when making the comparisons.

String	Action
`'ignoreid'`	ignore the group id
`'ignorename'`	ignore the group name

---

imat_group/intersect

Purpose

Group intersection.

Syntax

```
[c,ia,ib]=intersect(a,b)
```

Description

INTERSECT returns the IMAT_GROUP based on the input IMAT_GROUP A which has entities common to both A and B. Both A and B must be scalar IMAT_GROUP objects. The result will be sorted.

IA and IB are indices such that C = A(IA) and C = B(IB).

See Also

imat_group/union

---

imat_group/union

Purpose

Group union.

Syntax

```
[c,ia,ib]=union(a,b)
```

Description

UNION returns the IMAT_GROUP based on the input IMAT_GROUP A which has entities common to both A and B. Both A and B must be scalar IMAT_GROUP objects. The result will be sorted.

IA and IB are indices such that C is a sorted combination of A(IA) and B(IB).

See Also

imat_group/intersect

---

imat_group/setdiff

Purpose

Group difference.

Syntax

```
c=setdiff(a,b)
[c,i]=setdiff(a,b)
```

Description

SETDIFF returns the IMAT_GROUP entities in IMAT_GROUP A that are not in IMAT_GROUP B. Both A and B must be scalar IMAT_GROUP objects. The result will be sorted.

I is an index vector such that C = A(I).

See Also

imat_group/setxor

---

imat_group/setxor

Purpose

Group exclusive-or.

Syntax

```
c=setxor(a,b)
[c,i]=setxor(a,b)
```

Description

SETXOR returns the IMAT_GROUP entities that are not in the intersection of IMAT_GROUP A and IMAT_GROUP B. Both A and B must be scalar IMAT_GROUP objects. The result will be sorted.

IA and IB are indices such that C is a sorted combination of A(IA) and B(IB).

See Also

imat_group/setdiff

---

imat_group/validate

Purpose

Validate an IMAT_GROUP object.

Syntax

```
out=validate(group)
[out,isvalid]=validate(group,'silent')
[out,isvalid,msg]=validate(group)
```

Description

VALIDATE checks the input objects to make sure they are valid, and returns the output in an IMAT_GROUP. If none of the inputs are valid, OUT will be empty. Passing in the string `'silent'` will suppress output.

OUT is an IMAT_GROUP containing the validated GROUP. ISVALID is an optional output specifying whether the input data was a valid IMAT_GROUP. If MSG is requested, it is a cell array of strings containing messages for what entities are not valid (empty if the object is valid). If not requested, VALIDATE will issue an error message.

IMAT_FNPLOT CLASS

Description

The IMAT_FNPLOT class is a class that consists of a number of methods and properties that provide convenient control of an `imat_fn` plot. When a plot is created, its definition is stored in an IMAT_FNPLOT object, which allows you to change the plot display after the plot has been generated. The axes associated with the plot are associated with a UIPANEL. This allows the panel to be embedded in other GUIs and manipulated accordingly (see UIPLOT). This class also offers a lot of convenience for plotting. For example, you can link different plots together, so that changing an attribute of one (for example the window) will automatically change the axes on the other linked plots.

A contextual menu is created on all IMAT_FNPLOT plots. The menu provides convenient access to the majority of the IMAT_FNPLOT methods. For a complete description of each menu item, see the contextual menu section in the imat_fn/plot documentation.

Templates are a functionality that was once exclusive to UIPLOT. They have now been added to the basic function plotting capabilities. See the imat_fn/plot documentation for more information.

An important thing to understand about an IMAT_FNPLOT is that it consists of three axes that never get destroyed. The first axes of the three is always either the real or magnitude plot. The second axis is always the imaginary or phase plot. The third axis is always the Nyquist plot. Only the axes that are relevant to the currently displayed Complex Option are visible.

Reference Guide

The IMAT_FNPLOT has both properties and methods available to you.

Properties

The top level properties of the IMAT_FNPLOT object are the most basic to its function. The top level properties are series of handles to the different components of the plot, as well as a structure that contains the complete information about its status and format. These properties are described in the table below.

`hpanel`	Handle to uipanel that contains the entire plot.
`ha`	Handles (1x3) to the three axes that comprise the plot.
`hf`	Handle to the figure that contains the plot.
`hl`	Cell array (1x3) containing the handles to the lines on each of the three axes that comprise the plot.
`f`	`imat_fn` object containing the functions that are plotted.
`s`	Structure containing information about the plot settings.

S Structure Fields

The fields in the `.s` field of the IMAT_FNPLOT object contain a complete description of the current status/formatting of the IMAT_FNPLOT display. It should be considered 'read-only', as any changes will not affect the plot itself. Only the object methods functions called from the contextual menu affect the contents of this field.

Attributes with size 1x3 correspond to the three classifications of plots. That is: `'modulus + phase'`, `'real + imaginary'`, and `'nyquist'`. Items such as labeling, windowing, and scaling are stored on an individual basis for the three different types of plots.

`title`	String containing the plot title text. If it is `'default '`, the title is auto-generated based on the type of data shown on the plot. If it is anything else, it overrides the default setting.
`xlabel`	Cell array (1x3) of strings that defines the X axis label for each of the three axes. If it is `'default '`, the labels are auto-generated based on the type of data shown as well as the type of plot. For instance, in a magnitude/phase plot the x-label would appear only on the magnitude axis, and not the phase axis.
`ylabel`	Cell array (1x3) of cell arrays. They contain the Y axis labels for each axis of the three plot classifications. The first two cells are size 1x2 since they are multi-axis plots.
`xscale`	Cell array (1x3) of strings containing the X axis scale for each of the three plot types.
`yscale`	Cell array (1x3) of strings containing the X axis scale for each of the three plot types.
`complex`	String which defines the type of plot. The available types are `'modulus + phase'`, `'modulus'`, `'phase'`, `'real + imaginary'`, `'real'`, `'imaginary'`, and `'nyquist'`.
`window`	Cell array (1x3) of cell arrays storing the window information for each of the three plot classifications. The window information specifies the X and Y limits of the axes. The first two classifications are stored as {xwin ywin1 ywin2}. Nyquist is stored as {xwin ywin}. X-windowing is always the same for multi-axis plots.
`windowhistory`	1x3 structure containing the windowing history for each axis type. The field `.val` contains the windowing information, and the field .ind specifies the current index into .val.
`gridopt`	String which defines which grid lines are shown on the plot. The available options are `'X only'`, `'Y only'`, `'X and Y'`, and `'none'`.
`linkplots`	Logical value which defines whether this plot listens when a notification is sent out indicating another plot has changed.

`linkaxes`	Logical value which defines whether the three axes that make up a plot listen to each other. It is advisable not to turn this off. If this is turned off, axes that belong together (such as magnitude and phase) will not act together.
`cleanphase`	Logical value defining whether the phase has been cleaned.
`tag`	Structure that completely defines any tags that have been applied to the plot.
`style`	Structure that completely defines the formatting of the lines on the plot. This field will be empty until the STYLE method is used. Each style that has been set will then appear as a field of a structure in `.style`. This structure will be an array of the same length as the number of functions currently plotted.
`axis`	Structure that defines the formatting of the axes on the plot. This field will be empty until the AXIS method is used. Each axis property that has been set will then appear as a field of the structure in `.axis`.
`border`	Structure that defines the plot borders in pixels.
`font`	Structure that completely defines the formatting of the various fonts on the plot. It is described in more detail below.
`legend`	Structure that completely defines the legend on the plot. It is described in more detail below.
`xtick`	Structure that completely defines the formatting of the X axis ticks.
`ytick`	Structure that completely defines the formatting of the Y axis ticks.

Tag Fields

`type`	String defining whether the X or Y data (or both) is displayed on the plot when tagging. This can be `'x'`, `'y'`, or `'xy'`.
`cursoron`	Logical value specifying whether cursor is currently on.
`loc`	String defining whether tagging happens on data or grid. This can be `'data'` or `'grid'`.
`cursorh`	Handle to the cursor line if cursoring is currently on.
`title`	String storing the original title of the plot before tagging began.
`tags`	Structure containing information on each of the tags, such as which function the tag is one, what the type of the tag was, etc.
`h`	Handles to the text and marker for each tag.
`print`	Logical value defining whether the tag coordinate will be printed to the command window.

Font Fields

`title`	Structure containing the title font attributes and their current settings.
`xlabel`	Structure containing the X axis label font attributes and their current settings.
`ylabel`	Structure containing the Y axis label font attributes and their current settings.
`axis`	Structure containing the axis font attributes and their current settings.
`default`	Structure containing the default font attributes and their current settings. When fonts are set back to their defaults, they will use these settings.

Legend Fields

<table>
<tr><td>show</td><td>This is a logical flag defining whether the legend is shown.</td></tr>
<tr><td>type</td><td>This is the type of legend that is shown. There are several types: 'default ','idline1','idline4', 'ref/res', and 'custom'.</td></tr>
<tr><td>handle</td><td>This is the handle to the legend.</td></tr>
<tr><td>n</td><td>This is the maximum number of legend items to display when generating the legend. This is not used when the user manually enters a custom legend.</td></tr>
<tr><td>custom</td><td>This is a cell array of strings that define the entries for the custom legend. This is only populated if the type is set to 'custom'.</td></tr>
<tr><td>location</td><td>This is the location of the legend. The available options are the same as MATLAB's LEGEND function.</td></tr>
<tr><td>orientation</td><td>This is the orientation of the legend It is either 'Horizontal' or 'Vertical'.</td></tr>
<tr><td>font</td><td>Structure containing the legend font settings:
<table>
<tr><td>FontName</td><td>Font name (e.g. 'Arial'). Must be a valid MATLAB-supported value.</td></tr>
<tr><td>FontWeight</td><td>Font weight (e.g. 'bold'). Must be a valid MATLAB-supported value.</td></tr>
<tr><td>FontAngle</td><td>Font angle (e.g. 'italic'). Must be a valid MATLAB-supported value.</td></tr>
<tr><td>FontSize</td><td>Font size in points. Must be a valid MATLAB-supported value.</td></tr>
</table></td></tr>
<tr><td>interpreter</td><td>This specifies the interpreter to use for legend labels. The available options are the same as MATLAB's LEGEND function.</td></tr>
</table>

Methods

A number of methods are available for the IMAT_FNPLOT class. These methods allow you to manipulate the plot after it has been generated. In the documentation below, the IMAT_FNPLOT object is denoted by HPLOT.

HPLOT.applytemplate([tmpl][fname])

This method applies a template stored in the template structure TMPL or in the file FNAME to the plot. If neither is input, a file dialog will be shown where the template file can be selected.

HPLOT.axis(attribute1,val1,attribute2,val2,...,)

This method changes the axis properties of the axes that are displayed on an IMAT_FNPLOT. This method directly controls the style of the lines on an IMAT_FNPLOT. ATTRIBUTE1, ATTRIBUTE2, etc are attributes such as `'TickDir'` and `'Box'`. Attributes must be followed by their corresponding values. They are applied as attribute pairs as they would on any set() command. All of the axes on the plot will use these settings.

If other IMAT_FNPLOTs on the same figure are linked (they are by default), then the axes of those plots will also be changed by using this method.

HPLOT.complex(option)

This method changes the complex option of the IMAT_FNPLOT. The allowed options are: `'magnitude + phase'`, `'magnitude'`, `'phase'`, `'real + imaginary'`, `'real'`, `'imaginary'`, and `'nyquist'`.

If other IMAT_FNPLOTs on the same figure are linked (they are by default), then the complex option of those plots will also be changed by using this method.

HPLOT.font(['title','xlabel','ylabel','axis'][,'name',name][,'weight',weight]['angle',angle][,'size',size]['color',color])

This method changes the font of the title, xlabel, ylabel, or axes. The first argument defines which of these are being defined. The following arguments define the name, weight, and angle, size, and color of the font. All don't need to be defined. If one isn't defined, the current font will be used as the starting point.

The name can be any valid font name on the system. The weight is either `'bold'` or `'normal'`. The angle is either `'italic'` or `'normal'`. The size is any real number greater than one.

HPLOT.grid(option)

This method changes the grid lines that are displayed on an IMAT_FNPLOT. The available options are:

`'on'`	Turn on both x and y grid lines
`'off'`	Turn off all grid lines
`'x and y'`	Turn on both x and y grid lines (same as `'on'`)
`'x only'`	Turn on only the x grid lines
`'y only'`	Turn on only the y grid lines
`'none'`	Turn off all grid lines (same as `'off'`)

If other IMAT_FNPLOT on the same figure are linked (they are by default), then the grid lines of those plots will also be changed by using this method.

HPLOT.legend([string1,string2,string3,...][,'Location',location][,'Orientation',orientation][,'Type',type]['Interpreter',interp][,'Visible',visible]['FontName',FN][,'FontWeight',FW][,'FontAngle',FA][,'FontSize',FS][,numlegend])

This method functions much like the standard MATLAB LEGEND function. This one is just used specifically with IMAT_FNPLOT to enable advanced features.

The legend strings may be supplied as a series of string arguments, or as a cell array of strings. Strings specified that do not otherwise match a valid option are assumed to be the labels for the lines on the plot. You can use `%ResponseCoord`, `%ReferenceCoord`, `%Name`, `%IDLine1`, `%IDLine2`, `%IDLine3`, `%IDLine4`, `%FunctionType`, and `%RMS` as part of the legend and they will get replaced with the corresponding function values. The `'Location'`, `'Visible'`, `'Orientation'`, `'Interpreter'`, `'FontName'`, `'FontWeight'`, `'FontAngle'`, and `'FontSize'` arguments act the same a sMATLAB's legend function.

The `'Type'` argument is an added feature beyond the normal legend function. It determines how the legend labels are automatically generated. The allowed options are `'default '`, `'idline1'`, `'idline4'`, and `'ref/res'`. If custom string labels are entered, this is automatically set to `'custom'` and should not be entered.

Passing in a positive numeric scalar NUMLEGEND specifies the number of legend entries to display on the plot.

HPLOT.link(true/false)

This method sets the linking of the plot object. If TRUE, then whenever another IMAT_FNPLOT on the same figure sends its notification that it has changed, it will apply the same change. If FALSE, it simply ignores the notification. It is important to note, that all imat_fnplots on the same figure are linked by default.

HPLOT = HPLOT.loadfig([filename])

This method loads the figure data from a special figure file and recreates the figure from the information stored in the file. This functionality is the IMAT_FNPLOT equivalent of loading from a .fig file.

FILENAME is an optional string or Nx2 cell array of strings. If it is a cell array of strings or it contains wildcards, the user will be prompted for the filename with a graphical file dialog.

HPLOT.print()

This method prints the current plot to the printer, using MATLAB's print dialog. If you select *File -> Print* from the figure, it will bring up the print dialog. Otherwise, it will bring up the print preview dialog.

HPLOT.remove()

The REMOVE method removes the plot from the figure. It does all of the clean up that is necessary for a plot's deletion not to impact the stablity of the rest of the figure.

HPLOT.replace(G)

The replace method replaces the functions that are plotted on the IMAT_FNPLOT with those in the `imat_fn`, G, keeping the format as similar as possible. If the number of functions in G is less than or equal to the number currently plotted, the formats will be applied in order. If the number of functions in G exceeds the number of formats currently plotted, the existing formatting will be applied for all possible, with the extras having the default formatting.

HPLOT.saveas([filename])

This method saves the IMAT_FNPLOT to a file. It is called when you select *File* -> *Save As* from the figure menu. If FILENAME is not supplied, or is a cell array of strings, or contains wildcards, the user will be prompted with a graphical file dialog.

HPLOT.savefig([filename])

This method saves the IMAT_FNPLOT to a special figure file that can then be loaded to recreate the figure. This functionality is the IMAT_FNPLOT equivalent of saving to a `.fig` file.

FILENAME is an optional string or Nx2 cell array of strings. If it is a cell array of strings or it contains wildcards, the user will be prompted for the filename with a graphical file dialog.

HPLOT.savetemplate([filename])

This method saves the formatting of the current plot to a template file. A filename to save the template to, FILENAME, can be optionally passed in as well.

HPLOT.style(attribute1,val1,attribute2,val2,...,[,index])

This method directly controls the style of the lines on an IMAT_FNPLOT. ATTRIBUTE1, ATTRIBUTE2, etc are properties such as `'LineWidth'`, `'Marker'`, or `'LineStyle'`. You can set any of the Primitive Line Properties available in MATLAB. Please refer to the MATLAB documentation for more details.

Properties must be followed by their corresponding values. They are applied as property/value pairs as they would on any set() command. The INDEX argument controls which lines the properties will apply to. The index directly corresponds to the order of the functions as they were initially plotted (and as stored in the function as they were passed in). If the index is not supplied, then the style properties will be applied to all of the lines on the plot.

HPLOT.tag_style([index,]attribute1,val1,attribute2,val2,...)

This method directly controls the style of the tags on an IMAT_FNPLOT. ATTRIBUTE1, ATTRIBUTE2, etc are attributes such as 'FontSize', `'FontName'`, or 'Color'. They are applied as attribute pairs as they would on any set() command.The INDEX argument controls which tags the attributes will apply to. The index directly corresponds to the order of the tags as they were initially displayed. If the index is not supplied, then the style commands will apply to all of the tags on the plot.

HPLOT.title(title[,'Property1',value1,'Property2',value2,...])

This method adds a title to an imat_fnplot. If the complex option of the plot is changed, the title will always appear over the correct axis.

You can use `%ResponseCoord`, `%ReferenceCoord`, `%Name`, `%IDLine1`, `%IDLine2`, `%IDLine3`, `%IDLine4`, and `%FunctionType` as part of the label and they will get replaced with the value from the first function plotted.

In addition to the title, you can also specify Property/Value pairs. Valid pairs are any that MATLAB's TITLE function accepts. These properties and their values are stored as fields in `hplot.s.font.title`.

If other imat_fnplots on the same figure are linked (they are by default), then the xlabel of those plots will also be changed by using this method (assuming they are the same complex option).

HPLOT.uistyle()

This method opens a window where the line color, style and width can be edited on a line by line basis.

To edit the line style, select the line from the table and change the entries in the pulldowns in the table. Once the style values are set, press **Apply** to set the changes to the plot. When completely done, press **Done** to set the changes and close the window.

HPLOT.xlabel(label[,'Property1',value1,'Property2',value2,...])

This method changes the xlabel of an imat_fnplot. It works slightly differently than the typical MATLAB xlabel. Here, LABEL is the desired label for the plot. If it is entered without any associated INDEX, then the label is applied to all of the axes of the plot. If an index is specified, that label is applied to that specific axis only. INDEX can range from 1 to 3. An index of 1 corresponds to the `'magnitude'` or `'real'` axis. An index of 2 corresponds to the `'phase'` or `'imaginary'` axis. An index of 3 corresponds to the `'nyquist'` axis.

LABEL can be entered as a cell array of strings as long as INDEX is a vector of integers of the same size.

You can use `%ResponseCoord`, `%ReferenceCoord`, `%Name`, `%IDLine1`, `%IDLine2`, `%IDLine3`, `%IDLine4`, and `%FunctionType` as part of the label and they will get replaced with the value from the first function plotted.

In addition to the xlabel, you can also specify Property/Value pairs. Valid pairs are any that MATLAB's XLABEL function accepts. These properties and their values are stored as fields in `hplot.s.font.xlabel`.

If other imat_fnplots on the same figure are linked (they are by default), then the xlabel of those plots will also be changed by using this method (assuming they are the same complex option).

HPLOT.xlim(xlim[,index])

This method changes the x limits of an imat_fnplot. XLIM is a 1x2 vector defining the new limits. If XLIM is entered without any associated INDEX, then the limit is applied to the current complex option. If an index is specified, the limit is applied to that specific axis only. INDEX can range from 1 to 3. An index of 1 corresponds to the 'magnitude' or 'real' axis. An index of 2 corresponds to the 'phase' or 'imaginary' axis. An index of 3 corresponds to the `'nyquist'` axis.

Passing INF in for XLIM will fit the X limits to the data plotted, which is the same as the `'tight'` option. Passing in `'auto'` or [] for XLIM sets the limit to `'auto'`.

If other imat_fnplots on the same figure are linked (they are by default), then the limits of those plots will also be changed by using this method (even if they are not the same complex option).

HPLOT.xscale(scale)

This method changes the X axis scale of the currently visible plot.

SCALE is a string containing `'lin'`, `'log'`, or `'default'`.

If other IMAT_FNPLOTs on the same figure are linked (they are by default), then the X axis scale of those plots will also be changed by using this method (assuming they are the same complex option).

HPLOT.xtick([,'value',value][,'label',label][,'valuemode',valuemode][,'labelmode',labelmode])

This method sets the formatting of the xticks on an IMAT_FNPLOT. The xtick values and labels can be set manually using the `'value'` and `'label'` arguments, or they can be set so MATLAB sets them automatically using the `'valuemode'` and/or `'labelmode'` options.

The VALUE argument must be a numeric vector. This is the location where tick marks will be placed. The LABEL argument is a manual override of how the ticks are labeled. This must be a cell array of strings. If the size of the cell array is less than the number of ticks, then the cell array will be repeated so that all ticks have a label. If the size of the cell array is more than the number of ticks, only the labels up to the number of ticks will be used, starting from the beginning of the cell array.

VALUEMODE and LABELMODE can be set to `'manual'` or `'automatic'`. If values or labels are set, then these modes are set to `'manual'` automatically.

It is important to note that the xticks will automatically be applied to all axes associated with the current complex option.

HPLOT.ylabel(label[,index][,'Property1',value1,'Property2',value2,...])

This method changes the ylabel of an imat_fnplot. It works slightly differently than the typical MATLAB ylabel. Here, LABEL is the desired label for the plot. If it is entered without any associated INDEX, then the label is applied to all of the axes of the plot. If an index is specified, that label is applied to that specific axis only. INDEX can range from 1 to 3. An index of 1 corresponds to the `'magnitude'` or `'real'` axis. An index of 2 corresponds to the `'phase'` or `'imaginary'` axis. An index of 3 corresponds to the `'nyquist'` axis.

LABEL can be entered as a cell array of strings as long as INDEX is a vector of integers of the same size.

You can use `%ResponseCoord`, `%ReferenceCoord`, `%Name`, `%IDLine1`, `%IDLine2`, `%IDLine3`, `%IDLine4`, and `%FunctionType` as part of the label and they will get replaced with the value from the first function plotted.

In addition to the ylabel, you can also specify Property/Value pairs. Valid pairs are any that MATLAB's YLABEL function accepts. These properties and their values are stored as fields in `hplot.s.font.ylabel`.

If other imat_fnplots on the same figure are linked (they are by default), then the xlabel of those plots will also be changed by using this method (assuming they are the same complex option).

HELOT.ylim(ylim[,index])

This method changes the y limits of an IMAT_FNPLOT. Here, YLIM is the desired y limits for the plot. If it is entered without any associated INDEX, then the limits will be applied as appropriately as possible. That is, they will be applied to all axes shown, except in the case of a magnitude/phase plot, where only the magnitude will be windowed.

If an index is specified, then the y limits are applied to that specific axis only. INDEX can range from 1 to 3. An index of 1 corresponds to the `'magnitude'` or `'real'` axis. An index of 2 corresponds to the `'phase'` or `'imaginary'` axis. An index of 3 corresponds to the `'nyquist'` axis.

YLIM can be entered as a cell array of strings as long as INDEX is a vector of integers of the same size.

Passing INF in for YLIM will fit the Y limits to the data plotted, which is the same as the `'tight'` option. Passing in `'auto'` or [] for YLIM sets the limit to `'auto'`.

If other imat_fnplots on the same figure are linked (they are by default), then the y limits of those plots will also be changed by using this method (assuming they are the same complex option).

HPLOT.yscale(scale,axind)

This method changes the Y axis scale of the currently visible plot. Note that you cannot change the scale of the phase axis on a magnitude/phase plot.

SCALE is a string containing `'lin'`, `'log'`, or `'default'`.

AXIND is the index of the axis to change. For multi-axis plot (e.g. magnitude/phase), 1 is the bottom axis and 2 is the top axis.

If other IMAT_FNPLOTs on the same figure are linked (they are by default), then the X axis scale of those plots will also be changed by using this method (assuming they are the same complex option).

HPLOT.ytick(index[,'value',value][,'label',label][,'valuemode',valuemode][,'labelmode',labelmode])

This method sets the formatting of the yticks on an IMAT_FNPLOT. The ytick values and labels can be set manually using the `'value'` and `'label'` arguments, or they can be set so MATLAB sets them automatically using the `'valuemode'` and/or `'labelmode'` options.

INDEX is a required argument that defines which axis the yticks will be applied to. It will only be stored for that axis in the current complex option. An index of 1 corresponds to the `'magnitude'` or `'real'` axis. An index of 2 corresponds to the `'phase'` or `'imaginary'` axis. An index of 3 corresponds to the `'nyquist'` axis.

The VALUE argument must be a numeric vector. This is the location where tick marks will be placed. The LABEL argument is a manual override of how the ticks are labeled. This must be a cell array of strings. If the size of the cell array is less than the number of ticks, then the cell array will be repeated so that all ticks have a label. If the size of the cell array is more than the number of ticks, only the labels up to the number of ticks will be used, starting from the beginning of the cell array.

VALUEMODE and LABELMODE can be set to `'manual'` or `'automatic'`. If values or labels are set, then these modes are set to `'manual'` automatically.

It is important to note that the xticks will automatically be applied to all axes associated with the current complex option.

See Also

imat_fn/plot

IMAT Methods and Functions for `imat_vtkplot` objects (+FEA)

Please see these instructions for VTK Setup if you are having any issues using vtkplot.

imat_vtkplot	`imat_vtkplot` class constructor.
imat_vtkplot/backgroundcolor	Set the background color.
imat_vtkplot/text	Place arbitrary 2-D text on the plot.
imat_vtkplot/text3	Place arbitrary 3-D text on the plot.
imat_vtkplot/textcolor	Set the text color.
imat_vtkplot/colorrange	Set the colorbar range.
imat_vtkplot/setcontourcomponent	Set the contour display component.
imat_vtkplot/resultlocation	Set the result data location for the displayed contour results.
imat_vtkplot/cutplane	Add and control cutplanes on plot.
imat_vtkplot/displaystyle	Set mesh visibility.
imat_vtkplot/elementstyle	Set element style.
imat_vtkplot/nodestyle	Set node style.
imat_vtkplot/tracelinestyle	Set traceline style.
imat_vtkplot/ctracestyle	Set coordinate trace style.
imat_vtkplot/export	Export current display or animation to a file.

imat_vtkplot/getlabels	Get labeled entities from a VTK plot.
imat_vtkplot/labelelements	Label elements on undeformed or deformed plot.
imat_vtkplot/labelnodes	Label nodes on undeformed or deformed plot.
imat_vtkplot/interactiveselector	Select entities interactively.
imat_vtkplot/measure	Measure the distance between two nodes.
imat_vtkplot/transparent	Set mesh transparency.
imat_vtkplot/view	Set the model view.
imat_vtkplot/visible	Set the mesh visibility.

imat_vtkplot/imat_vtkplot (+FEA)

Purpose

Advanced FEM/result plotting class

Syntax

```
imat_vtkplot
```

Description

The IMAT_VTKPLOT class is a class that consists of a number of methods and properties that provide convenient control of an `imat_fem`, `imat_shp`, or `result` plot. An instance of the class is created by using the VTKPLOT method of either of the three objects.

The IMAT_VTKPLOT object has several useful public properties. These properties store the important parts of the plot for use / reference.

Property	Description
`hp`	Handle to the uipanel that contains the plot.
`hf`	Handle to the figure that the plot is on.
`s`	Structure containing all of the information defining the plot. Most is self explanatory, but for more information see the detailed help.
`v`	Handle to the VTK object.
`fem`	Structure containing a copy of the FEM that is plotted.

`contour`	`imat_shp` or `result` containing the contours for the plot.
`displacement`	`imat_shp` containing the displacements for the plot.

---

imat_vtkplot/backgroundcolor (+FEA)

Purpose

Set the background color.

Syntax

```
hplot.backgroundcolor(color)
```

Description

This method sets the background color of the VTK plot.

COLOR is a string or 1x3 RGB vector in the range of [0,255]. The RGB value of the background color is stored in the `displaycolor.background` field of the `s` structure of HPLOT.

---

imat_vtkplot/text3 (+FEA)

Purpose

Place arbitrary 3-D text on the plot.

Syntax

```
hplot.text3(coord,label)
hplot.text3(color,label,'scale',scale,'color',color)
```

Description

This method displays arbitrary 3D text at the specified locations on the plot.

COORD is an Nx3 matrix of coordinates at which to place the text.

LABEL is a string or cell array of strings containing the text strings to place on the plot.

SCALE is an optional numeric scalar specifying the text size. The default is 1.

COLOR is an optional Nx3 numeric matrix containing the text colors as RGB values. These values must be between 0 and 1. The default color is black ([0 0 0]).

Examples

```
>> hplot.text3([0 0 0],'label_text')
>> hplot.text3([0 0 0; 1 1 1],{'label 1'; 'label 2'},'scale',3,'color',[1 0 0; 1 0 0])      % Make
text red
```

See also

imat_vtkplot/text

---

imat_vtkplot/textcolor (+FEA)

Purpose

Set the text color.

Syntax

```
hplot.textcolor(color)
```

Description

This method sets the color of text on the VTK plot.

COLOR is a string or 1x3 RGB vector in the range of [0,255]. The RGB value of the color is stored in the `displaycolor.text` field of the `s` structure of HPLOT.

---

imat_vtkplot/colorrange (+FEA)

Purpose

Set the colorbar range.

Syntax

```
hplot.colorrange(range)
hplot.colorrange('on')
hplot.colorrange([range][,'on'][,'off'])
```

Description

COLORRANGE controls the range of values on the color bar, as well as its visibility.

RANGE is a 1x2 numeric vector with the minimum value being the first member, and the maximum value being the second member. To let the plot determine the max or min value, set the corresponding value to Inf.

Passing in the string 'on' will enable the colorbar display, and passing in the string `'off'` will turn off the colorbar display.

If no values are passed in, a GUI will be presented where the values can be entered, and the colorbar can be enabled if it isn't already.

imat_vtkplot/setcontourcomponent (+FEA)

Purpose

Set the contour display component.

Syntax

```
hplot.setcontourcomponent(comp)
```

Description

SETCONTOURCOMPONENT sets the contour result component being displayed.

COMP is either a string containing the component (or 'element' to use element colors), or a numeric scalar containing the index into the component list found in

```
HPLOT.contour.components
```

Examples

```
>> h.setcontourcomponent('Y')
>> h.setcontourcomponent(2)
>> h.setcontourcomponent('element')
```

imat_vtkplot/resultlocation (+FEA)

Purpose

Set the result data location for the displayed contour results.

Syntax

```
hplot.resultlocation(value)
hplot.resultlocation('centroidal')
```

Description

RESULTLOCATION sets the result data location for the displayed contour results.

VALUE is the result location value. Valid choices are

Value	**Meaning**
`'centroidal'`	Element centroidal
`'mean'`	Element nodal average
`'min'`	Element nodal maximum
`'max'`	Element nodal minimum
`'absmax'`	Element nodal absolute maximum

See Also

imat_result/criterion

---

imat_vtkplot/cutplane (+FEA)

Purpose

Add and control cutplanes on plot.

Syntax

```
hplot.cutplane(dir,action)
```

Description

This method controls the cut planes available on the IMAT_VTKPLOT. There a total of four cutplanes that can be created and modified. They are specified by using the DIR argument. The available directions for the cut planes are `'x'`, `'y'`, `'z'`, and `'g'`. `'x'`, `'y'`, and `'z'` are cut planes through their respective axis. `'g'` is a general cutplane that can be dragged to cut through any plane.

ACTION can be `'toggle'`, `'visible'`, or `'flip'`. Each of these actions are applied independently on all four available cut planes, based on the cutplane specified by the DIR argument.

The `'toggle'` action toggles whether the cutplane is active. This action will add the cutplane to the plot and cut plane will be active. Calling `'toggle'` again will remove the cutplane from the plot from the figure and the model will return to its uncut state.

The `'visible'` action toggles the visibility of the cutplane. That is, when the visibility is toggled off the model will remain cut, butthe cut plane and its associated controls will disappear.

The `'flip'` action flips the side of the plane that is cut.

imat_vtkplot/displaystyle (+FEA)

Purpose

Set mesh visibility.

Syntax

```
hplot.displaystyle(mode,toggle)
```

Description

This method changes whether the mesh is shown as a shaded or line view. MODE is either `'undeformed'` or `'deformed'`. TOGGLE is either `'line'` or `'shaded'`.

imat_vtkplot/csstyle (+FEA)

Purpose

Set element style.

Syntax

```
hplot.csstyle('visible',visible,'dirlabels',dirlabels,'cslabels',cslabels,'scale',scale)
```

Description

This method changes the style of the coordinate systems on the undeformed FEM.

VISIBLE is a logical specifying whether the coordinate systems should be displayed.

DIRLABELS is a logical specifying whether the coordinate system X, Y, and Z direction labels should be displayed.

CSLABELS is a logical specifying whether the coordinate system labels (IDs) should be displayed.

SCALE is a number greater than 0 that sets the coordinate system sizing scale factor.

imat_vtkplot/elementstyle (+FEA)

Purpose

Set element style.

Syntax

```
hplot.elementstyle(mode,'width',width,'style',style,'edgecolor',edgecolor,'facecolor',facecolor)
```

Description

This method changes the style of the elements for either the undeformed or deformed FEM.

MODE controls which display the style changes will be applied to -- `'undeformed'` or `'deformed'`.

WIDTH changes the width of the element borders. This can be any number greater than zero.

STYLE controls the style of the element edges. This can be `'-'` for solid lines, `'--'` for dashed lines, `'-.'` for dash-dot lines, `':'` for dotted lines, or `'none'` for no lines.

EDGECOLOR sets the edge color for all of the elements. It must be a 1x3 vector with values from 0-255. This specifies the RGB value of the color. Similarly, FACECOLOR specifies the face color of the elements. It must be an Nx3 vector, where N is the total number of elements.

---

imat_vtkplot/nodestyle (+FEA)

Purpose

Set node style.

Syntax

```
hplot.nodestyle('undeformed','size',3)
hplot.nodestyle('deformed,'color',[255 0 0])
hplot.nodestyle(mode,'size',size,'color',color)
hplot.nodestyle('both','color',ideas_colormap(fem.node.color))
```

Description

This method changes the style of the nodes on the mesh.

MODE controls which display the style changes will be applied to -- `'undeformed'`, `'deformed'`, or `'both'`.

SIZE is a number greater than 0 which sets the size of the nodes.

COLOR is an Nx3 RGB vector which sets the color of the nodes. N must be 1 for `'undeformed'` or `'deformed'` modes, and can be 1 or the number of nodes for `'both'`. The values must be between 0 and 255.

See Also

ideas_colormap

imat_vtkplot/tracelinestyle (+FEA)

Purpose

Set traceline style.

Syntax

```
hplot.tracelinestyle('undeformed','width',3)
hplot.tracelinestyle('deformed,'color',[255 0 0])
hplot.tracelinestyle(mode,'width',width,'style',style,'color',color)
```

Description

This method changes the style of the tracelines on the mesh.

MODE controls which display the style changes will be applied to -- `'undeformed'` or `'deformed'`.

WIDTH changes the width of the tracelines. This can be any number greater than zero.

STYLE controls the style of thelines. This can be `'-'` for solid lines, `'--'` for dashed lines, `'-.'` for dash-dot lines, `':'` for dotted lines, or `'none'` for no lines.

COLOR is an Nx3 RGB vector which sets the line color of the tracelines. N must either be 1 or the number of tracelines. The values must be between 0 and 255.

---

imat_vtkplot/ctracestyle (+FEA)

Purpose

Set element style.

Syntax

```
hplot.csstyle('visible',visible,'dirlabels',dirlabels,'nodelabels',nodelabels,'scale',scale)
```

Description

This method changes the style of the coordinate traces on the undeformed FEM.

VISIBLE is a logical specifying whether the coordinate traces should be displayed.

DIRLABELS is a logical specifying whether the coordinate system X, Y, and Z direction labels should be displayed.

NODELABELS is a logical specifying whether the coordinate trace node labels (IDs) should be displayed.

SCALE is a number greater than 0 that sets the coordinate tracesizing scale factor.

---

imat_vtkplot/export (+FEA)

Purpose

Export current display or animation to a file.

Syntax

```
hplot.export()
hplot.export(FNAME[,MAG][,'transient']
```

Description

This method exports the current VTK image or animation to a file. It determines the file type based on the filename extension. Valid file types are

`.png`	Portable Network Graphics (recommended)
`.tiff`	Tagged Image File Format
`.jpg`	Joint Photographic Experts Group
`.bmp`	Bitmap
`.pnm`	Portable Any Map
`.avi`	Audio Video Interleave (animation).

If the AVI extension is used, the current result animation will be saved. Otherwise a static image will be saved. AVI support is not available on Mac or Linux.

FNAME is an optional string specifying the filename to create. If it is not supplied, is empty, or contains wildcards, you will be prompted for the filename with a graphical file dialog.

MAG is an optional scalar specifying the image magnification for file creation. Higher magnification results in better image quality at the expense of bigger file size. The default is 1.

The optional input string `'transient'` directs EXPORT to create a transient animation, regardless of the VTKPLOT menu setting. This is only valid for AVI export.

---

imat_vtkplot/getlabels (+FEA)

Purpose

Get labeled entities from a VTK plot.

Syntax

```
val = hplot.getlabels(type,mode)
val = hplot.getlabels;
```

Description

This method returns the IDs of the labeled entities on a VTK plot.

TYPE specifies the type of entity for which to extract the labels -- `'node'` (default) or `'element'`.

MODE controls which display the labels will be extracted from -- `'undeformed'` (default) or `'deformed'`.

VAL is a vector of the entity IDs in the order in which they were picked.

Examples

```
>> val = hplot.getlabels('node');
>> val = hplot.getlables('elem','def');
```

---

imat_vtkplot/labelelements (+FEA)

Purpose

Label elements on undeformed or deformed plot.

Syntax

```
hplot.labelelements(mode,id)
hplot.labelelements('undeformed',id,'size',20)
hplot.labelelements('deformed','clear')
```

Description

This method labels the elements of the model. MODE controls which model the labels will be associated to -- `'undeformed'` or `'deformed'`.

ID is a 1-D vector of node IDs to be labeled. To clear all of the labels, pass in `'clear'` as the ID.

The optional input string `'size'` followed by a numeric scalar sets the label size. The default size is 12. Please note that the size also affects the node label size.

---

imat_vtkplot/labelnodes (+FEA)

Purpose

Label elements on undeformed or deformed plot.

Syntax

```
hplot.labelnodes(mode,id)
hplot.labelnodes('undeformed',id,'size',20)
hplot.labelnodes('deformed','clear')
```

Description

This method labels the nodes of the model. MODE controls which model the labels will be associated to -- `'undeformed'` or `'deformed'`.

ID is a 1-D vector of node IDs to be labeled. To clear all of the labels, pass in `'clear'` as the ID.

The optional input string `'size'` followed by a numeric scalar sets the label size. The default size is 12. Please note that the size also affects the element label size.

imat_vtkplot/interactiveselector (+FEA)

Purpose

Select entities interactively.

Syntax

```
out = hplot.measure(type,mode,cbfcn)
out = hplot('off');
```

Description

This method provides an interactive selector for nodes and elements. You may provide a callback function that will be called whenever the user selects an entity of the type specified.

TYPE is a string specifying the entity type. Valid choices are `'node'` (default) and `'element'`.

MODE specifies the model from which the entities are extracted -- `'undeformed'` (default) or `'deformed'`.

CBFCN is either a function handle or a cell array containing the callback function handle or name and valid input arguments to the callback function. The entity ID will be passed in as the last input argument to the function.

If no callback function is specified, INTERACTIVESELECTOR will return with the entity label selected. Otherwise, it will stay in the interactive selector mode until the user cancels by pressing <ESC>.

Passing in the string 'off' causes INTERACTIVESELECTOR to disable interactive selection. This is useful for situations where you want to disable the interactive selector from another external callback.

OUT contains the ID of the last entity selected. If the user cancels, OUT will be empty.

Examples

```
>> out = hplot.measure('node','undeformed',{@my_function, 1, 'argument'})
>> out = hplot.measure('elem','deformed',@my_function)
```

---

imat_vtkplot/measure (+FEA)

Purpose

Measure the distance between two nodes.

Syntax

```
out = hplot.measure(mode)
out = hplot.measure('deformed')
```

Description

This method measures the distance between two nodes in a model. The user selects the two nodes on the plot. These nodes will be labeled as they are selected. The labels will remain until you change the display or issue another command, so you can see what was selected. To force a display refresh after using MEASURE, you can use

```
>> hplot.v.canvas.repaint();
```

Pressing the <ESC> key will cancel the measurement command.

MODE specifies the model from which the entities are extracted -- `'undeformed'` (default) or `'deformed'`.

OUT is a 1x3 vector containing the first and second nodes selected, and the distance between them. If the user cancels, OUT will be empty.

---

imat_vtkplot/transparent (+FEA)

Purpose

Set mesh transparency.

Syntax

```
hplot.transparent(mode,value)
```

Description

This method changes the transparency of either the undeformed or deformed mesh.

MODE controls which display the transparency will be applied to -- `'undeformed'` or `'deformed'`.

VALUE is a value between 0 and 100 that specifies the level of transparency desired. 0 is opaque, and 100 is completely transparent.

---

imat_vtkplot/view (+FEA)

Purpose

Set the model view.

Syntax

```
hplot.view(v)
```

Description

This method changes the view of the model. The view is specified by V, which can be `'xy'`, `'xy-'`, `'yz'`, `'yz-'`, `'xz'`, `'xz-'`, `'iso'`, or a 1x2 numeric vector containing [AZ EL], the azimuth and elevation. Setting V to `'reset'` will reset the zoom so that the model fits in the panel.

Other valid inputs for the view option specify whether the view is a parallel or perspective projection. This can be controlled by passing in `'parallel'` or `'perspective'`, respectively.

---

imat_vtkplot/interactionstyle (+FEA)

Purpose

Select entities interactively.

Syntax

```
hplot.interactionstyle(style)
```

Description

INTERACTIONSTYLE sets the style of interaction with the VTK window. This specifies what mouse and keyboard controls to use to manipulate the VTK display (pan, zoom, and rotate).

STYLE is one of the following strings. RMB means Right Mouse Button, MMB means Middle Mouse Button, and LMB means Left Mouse button. MW means Mouse Wheel.

Style		
`'vtk' or 'default'`	Use the default VTK controls.	
	Pan	MMB
	Zoom	RMB
	Rotate	LMB (Ctrl+LMB rotates about screen)
	Spin	Ctrl+LMB
`'femap'`	Behave like Siemens FEMAP.	
	Pan	Ctrl+MMB
	Zoom	Shift+MMB or RMB
	Rotate	MMB
	Spin	Alt+MMB
`'nx'`	Behave like Siemens NX. I-deas STYLE also works with NX.	
	Pan	Shift+MMB
	Zoom	F2+MMB or Ctrl+MMB or RMB
	Rotate	MMB
	Spin	Alt+MMB
`'ideas'`	Behave like I-deas.	
	Pan	F1+LMB
	Zoom	F2+LMB
	Rotate	F3+LMB

The following is applicable for all styles.

F4 snaps to the nearest orthogonal or iso view. For all styles, the mouse wheel zooms. For all except `'vtk'`, LMB and MMB can be used interchangeably.

F1+MW rotates the view about the model X axis in 5 degree increments. F2+MW rotates about the Y axis, and F3+MW rotates about the Z axis. You can also use Ctrl, Shift, and Ctrl+Shift or Alt instead of the F1, F2, and F3 keys for this functionality.

---

imat_vtkplot/visible (+FEA)

Purpose

Set the mesh visibility.

Syntax

```
hplot.visible(mode,toggle)
hplot.visible('undeformed',true)
```

Description

This method changes the visibility of either the undeformed or deformed mesh.

MODE controls which display the visibility toggle will be applied to -- `'undeformed'` or `'deformed'`.

TOGGLE is a logical that specifies the mesh visibility.

IMAT Methods and Functions for `fcn` objects

The methods referenced on this page are specific to FCN objects. All of the imat_fn methods are also available for FCN.

fcn	Create an `fcn` object
fcn/load	Get one or more attributes of an `imat_fn` object
fcn/save	Set one or more attributes of an `imat_fn` object

---

fcn/fcn

Purpose

Create an `fcn` object.

Syntax

```
f=fcn
f=fcn(m,n,p,...)
f=fcn(m,n,p,...,'Attr',value,...)
```

Description

FCN is a subclass of IMAT_FN that was designed for use with Vibrata. It accesses all of the properties and methods of IMAT_FN. It also has some additional properties. These are described here.

The FCN constructor operates the same way as the IMAT_FN constructor.

To see the methods available for FCN, type '`methods(fcn)`'.

FCNs are stored in a file with the *.fcn* extension. It is a MATLAB *.mat* file with contents specific to FCN. Use fcn/load to read FCNs from a `.fcn` file. Use fcn/save to save FCNs to a `.fcn` file.

Examples

```
>> f = fcn(3,4,2);

>> f = fcn(3,'FunctionType','Frequency Response Function');
```

Load and save examples.

```
>> f = load(fcn);

>> f = load(fcn,'file.fcn');

>> save(f);

>> save(f,'file.fcn');
```

See Also

fcn/load, fcn/save, imat_fn

User's Guide

---

fcn/load

Purpose

Load FCN and function sets from a *.fcn* file into a structure.

Syntax

```
out = load(fcn)
out = load(fcn,filename)
[out,filename] = load(f,filename,'checksum','function_sets','silent')
```

Description

LOAD reads a *.fcn* file into MATLAB. Since it is an FCN method, you must supply a FCN object to invoke this method. Any FCN object, including an empty FCN returned by the FCN constructor function will suffice. FILENAME is an optional string specifying the filename to read. If it is not provided, you will be prompted with a graphical file selector.

Passing in the string `'checksum'` will load only the UIDs and associated function checksums, and return the output in a cell array. If the checksum data is not stored in the FCN file, load will load the functions and create the checksum data.

Passing in the string `'function_sets'` will load only the functions sets stored in the file.

Passing in the string `'silent'` suppresses output. load will otherwise display a warning message for any function set it loads where not all of the functions referenced by that set are found.

OUT is a structure containing the contents of the *.fcn* file. One field contains the FCN object. The name of the field is the name of the variable that was saved to the *.fcn* file. If a function set was also saved to the *.fcn* file, it is returned as a second field in OUT called `.function_sets`.

See Also

fcn/save, fcn/md5sum

---

fcn/save

Purpose

Save FCN (and optionally function sets) to a *.mat* file with extension *.fcn*.

Syntax

```
save(f)
save(f,'file.fcn')
filename = save(f,FILENAME,FSET,'-append')
```

Description

SAVE writes FCN objects and optionally function sets to a *.fcn* file. Only a single FCN object can be saved to a given FCN file. FCN is the FCN object to save. FILENAME is an optional string containing the filename. If it is not provided, you will be prompted with a graphical file selector. FSET is an optional function set structure.

By default the *.fcn* file will be overwritten with the new data. If the string `'-append'` is provided, the supplied variables are appended to the file. If an empty FCN object is provided, it will not overwrite the existing functions in the file.

For example, to overwrite a function set in an existing file, you can use

```
save(fcn([]),'filename.fcn',fset_new,'-append');
```

The example below, where F is a non-empty FCN, writes the new FCN to the file but does not overwrite the existing function set in the file.

```
save(f,'filename.fcn','-append');
```

See Also

fcn/load, fcn/md5sum

IMAT Functions for import and export

---

readadf	Import functions, time histories, or shapes from Associated Data File
writeadf	Export functions, time histories, or shapes to Associated Data File
readunv	Import from a Universal file
writeunv	Export data to a Universal file
writesubst	Create a substructure Universal file
readnas **(+FEA)**	Import Nastran Output2, bulk data, and punch files
writenas **(+FEA)**	Write to a Nastran Output2 file
writeop4 **(+FEA)**	Write matrices to Nastran Output4 format
create_op2table **(+FEA)**	Create table for use with WRITENAS
readneu **(+FEA)**	Import data from FEMAP Neutral files.

writefemap **(+FEA)**	Export functions, shapes, results, groups, and FEM geometry to a FEMAP Neutral file or directly to a FEMAP session through the COM interface.
femap_invoke **(+FEA)**	Invoke a FEMAP API method through the COM interface
readodb **(+FEA)**	Import data from Abaqus ODB files

---

readadf

Purpose

Import functions or shapes from I-DEAS or NX ADF.

Syntax

```
f=readadf('filename')
f=readadf('*.afu','range',[0 4])
f=readadf('adfheader')
f=readadf('silent','noprogbar')
f=readadf('*.afu','headersonly')
[f,fname]=readadf('*.afu',[])
f=readadf('*.ati',[1:10])
f=readadf
```

Description

The READADF function reads all records from an I-DEAS ADF into the variable `f`. The filename should include the extension, either `.afu` for a function ADF, `.ati` for a time history ADF, or `.ash` for a shape ADF. The output variable `f` will be either an `imat_fn` or an `imat_shp` variable, with all records from the ADF in a single column. The function or shape data will be converted to the current unit system, as selected by setunits. If no units have been selected, the user will be prompted to choose the unit system.

If the filename has wildcards, a standard file dialog will be presented to allow the user to choose the file to read.

Calling READADF with no arguments is equivalent to using a filename of `'*.a*'`.

To read a subset of records from the ADF, you may pass in an optional numeric vector. The vector specifies records to read from the ADF. The third syntax example above will attempt to read the first 10 time history records from the ATI file selected from the file selection dialog. If are not sure which records you want to read from the ADF, you can pass in an empty vector. In this case READADF will read the record headers from the ADF and present you with a GUI, allowing you to select which records you wish to read.

To read in just the function headers and not the function data, pass in the string argument `'headersonly'`. READADF will return an `imat_fn` or `imat_shp` containing all of the attributes but no data.

To read a range of abscissa values from an ATI or AFU file, pass in the string argument `'range'` followed by a numeric vector containing the minimum and maximum abscissa values to read. Only the function data within the specified range will be imported. This request will be ignored for ASH files or if you also pass in `'headersonly'`. If you pass in the string `'silent'`, READADF will suppress output. The optional input string `'noprogbar'` will suppress the progress bar.

If you pass in the string `'adfheader'`, READADF will return an nx8 matrix containing the ADF header information for valid records in the file. The data in each column is as follows:

Column	1	2	3	4	5	6	7	8
AFU and ATI	record number	starting block	number of blocks	response node	response direction	reference node	reference direction	version number
ASH	record number	starting block	number of blocks	0	0	0	0	0

READADF supports an optional output variable. The second output from READADF is the path and filename of the ADF selected for reading.

By default, the raw data in an ADF is in SI units. READADF will read the data and convert it to the current units system specified by setunits. NX has the ability to write ADFs for which the raw data is written in units other than SI. READADF supports these files by first converting the raw data to SI, then to the current units system.

Examples

```
>> f=readadf('/users/testdata/run01.afu');
>> s=readadf('*.ash');
```

See Also

writeadf, setunits, getunits

---

writeadf

Purpose

Export functions or shapes to I-DEAS ADF.

Syntax

```
writeadf('filename',f,g,...)
writeadf('filename',f,'delete')
writeadf('*.afu',f,'silent','noprogbar')
writeadf('*.afu',f,g,'overwrite',[1 2 4 6])
```

Description

WRITEADF writes one or more `imat_fn` or `imat_shp` variables to an I-DEAS ADF. The filename should include the extension, either `.afu` for a function ADF, `.ati` for a time history ADF, or `.ash` for a shape ADF. The variables to be written should be compatible with the desired ADF type. If you pass in the string `'silent'`, WRITEADF will suppress output. The optional input string `'noprogbar'` will suppress the progress bar.

If the specified file exists, then by default the records will be appended to the file. Otherwise, a new file will be created. The optional input string `'delete'` will delete the ADF before writing to it, if it exists.

If the `'overwrite'` option is used, WRITEADF will overwrite records in the ADF rather than appending them to the end of the ADF. RECIND is a vector of indices into the ADF for which records to overwrite. It must be the same length as the total valid number of records to write, and must contain valid record numbers. To see a list of valid record numbers in the ADF, see column 1 of the ADF header matrix returned by READADF with the `'adfheader'` option.

If the filename has wildcards, or no filename argument is provided, then a standard file dialog will be presented to allow the user to choose the file to write. Note that if you choose an existing ADF from the file dialog, you will get a warning that the file will be replaced. Please ignore this message, as it is hard-wired into MATLAB's file dialog.

Examples

```
>> writeadf('/users/testdata/run01_updated.afu', f);
>> writeadf('file.afu', f, 'overwrite', [1 4]);
>> writeadf('*.ash', s);
```

See Also

readadf, getunits, setunits

---

readunv

Purpose

Import data from a Universal file.

Syntax

```
f=readunv('filename')
f=readunv('filename','dialogtitle')
f=readunv('silent','noprogbar')
f=readunv('asresult')
f=readunv('shponly')
f=readunv('filename',[datasets])
[f,fname]=readunv
```

Description

The READUNV function is used to read various types of universal files from I-deas.

The filename argument is a string specifying the universal file to be read. If wildcards are included in the filename, then a standard file dialog will be presented, allowing the user to choose the file. The optional dialog title will be used as a title for the file dialog window. The optional input string `'silent'` suppresses output to the command window. The optional input string `'noprogbar'` suppresses the progress bar.

Calling READUNV without any arguments is equivalent to supplying a filename of `'*.unv'`.

You may also specify a list of datasets (see below) to either skip or read. This list is a numeric vector containing a list of dataset numbers. To skip a dataset, make the dataset number negative in DATSETS. To read only a given dataset list, pass the dataset list in as positive numbers. Positive dataset numbers in DATASETS implicitly assume that READUNV should skip all other datasets.

By default READUNV will read mode shapes in dataset 2414 into an `imat_shp`, and all other results into an `imat_result`. Two optional input strings can alter this behavior. Passing in the string `'asresult'` will cause READUNV to read all dataset 2414 results into an `imat_result`. Passing in the string `'shponly'` will cause READUNV to read only mode shapes from dataset 2414 into an `imat_shp`.

FNAME is an optional output argument containing the filename of the Universal file read.

The following universal file formats are supported:

Dataset	**Description**	**READUNV Output**
164	Units	none (used to adjust units of result)
58	Functions/time histories	`imat_fn` array
58b	Functions/time histories (binary)	`imat_fn` array
55	Shapes from Test	`imat_shp` array
2414	Shapes from Simulation	`imat_shp` array , and/or result object
1802	Coordinate trace	`imat_ctrace` object
757	DOF set	`imat_ctrace` object
247	DOF set	substructure entity (see below)
252	Entity definition matrix	substructure entity (see below)
611, 612	Matrix for the matrix package	substructure entity (see below)
2453	Results matrix	substructure entity (see below)
18	Coordinate system	`imat_cs` object in `imat_fem` object property (`.cs`)
2420	Coordinate system	`imat_cs` object in `imat_fem` object property (`.cs`)
15	Node	`imat_node` object in `imat_fem` object property (`.node`)
781	Node	`imat_node` object in `imat_fem` object property (`.node`)
2411	Node	`imat_node` object in `imat_fem` object property

		(`.node`)
82	Traceline	`imat_tl` object in `imat_fem` object property (`.tl`)
2416	Traceline	`imat_tl` object in `imat_fem` object property (`.tl`)
2431	Traceline	`imat_tl` object in `imat_fem` object property (`.tl`)
780	Element	`imat_elem` object in `imat_fem` object property (`.elem`)
2412	Element	`imat_elem` object in `imat_fem` object property (`.elem`)

It is also possible to write your own dataset reader plugins, so you can extend READUNV to import datasets not natively supported. Please refer to the Extending Readunv page for more information on this useful capability.

In most cases, only one type of output will be generated from a single universal file (this should be the case if the universal file was written by I-DEAS Test). Functions or time histories will be output in a single `imat_fn` variable with as many elements as necessary. Shapes will also be output in a single `imat_shp` variable. A universal file containing a single coordinate trace will be read into an `imat_ctrace` variable. A universal file containing multiple coordinate traces will be read into a cell array with one `imat_ctrace` in each cell element. The coordinate trace or DOF set name in the Universal file will be stored as the `imat_ctrace` name.

A substructure can be specified using different dataset types. Dataset 247 and 252 go together. If the substructure is stored using these datasets, the Universal file should contain both a dataset 247 (identifying the degrees of freedom in the substructure) and one or more datasets 252 (matrices). If the substructure is stored in dataset 2453, this dataset should appear consecutively with the degree of freedom matrix and entity matrices. Datasets 611 and 612 contain only matrices; the corresponding degree of freedom information is not available. When reading such a universal file, READUNV will create a MATLAB structure variable with the following fields:

`f.aset`	`imat_ctrace` of independent DOF (size Na)
`f.oset`	`imat_ctrace` of constrained (dependent) DOF (size No)
`f.sset`	`imat_ctrace` of restrained DOF (size Ns)
`f.mass`	Na x Na mass matrix
`f.stiffness`	Na x Na stiffness matrix
`f.viscous`	Na x Na viscous damping matrix
`f.hysteretic`	Na x Na hysteretic damping matrix
`f.constraint`	No x Na back expansion matrix

(Some of these fields may be empty if the associated matrix was not included in the substructure universal file.) Note that the s-set partition of the matrices will be discarded by READUNV, if present.

FEM data entities will be stored in an imat_fem with properties that contain individual FEM entity objects with the entity data (i.e. nodes and elements). For a full description of these structures, please refer to the FEM Data Format Reference. Please note that `readunv` will return the node ordering for parabolic elements in Nastran-centric ordering, which is different from the ordering specified in the Universal file. Also note that coordinate system 1 (global Cartesian) in the Universal file will be renumbered to 0 on import.

In the situation when mixed types of data are included in the same universal file, READUNV will create a cell array for the output argument. The cell array will contain, in this order, an `imat_fn` (all functions read), an `imat_shp` (all shapes read), the MATLAB structure variable described above, followed by one `imat_ctrace` per coordinate trace. Only those elements that were present in the universal file will be placed in the output cell array.

Some third party vendors write Universal files with incorrect formatting. In particular, dataset delimiter lines (containing '   -1') are written to have trailing spaces. READUNV automatically detects files with this formatting problem and utilizes a Perl script to clean the files. It creates a new file, given the filename with `'_fix'` appended to the end. READUNV will warn you if this happens and will proceed to read the cleaned file, which it leaves on the system after it is finished.

Examples

```
>> f=readunv('/ms5/examples/tda/air_func.unv');
>> f=readunv('*.unv','Select file to read');
>> f=readunv('*.unv','Select file to read',[55 58]);  % Read only functions and shapes
```

See Also

writeunv, writesubst

---

readunv

Purpose

Import data from a Universal file.

Syntax

```
f=readunv('filename')
f=readunv('filename','dialogtitle')
f=readunv('silent','noprogbar')
f=readunv('asresult')
f=readunv('shponly')
f=readunv('filename',[datasets])
[f,fname]=readunv
```

Description

The READUNV function is used to read various types of universal files from I-deas.

The filename argument is a string specifying the universal file to be read. If wildcards are included in the filename, then a standard file dialog will be presented, allowing the user to choose the file. The optional dialog title will be used as a title for the file dialog window. The optional input string `'silent'` suppresses output to the command window. The optional input string `'noprogbar'` suppresses the progress bar.

Calling READUNV without any arguments is equivalent to supplying a filename of `'*.unv'`.

You may also specify a list of datasets (see below) to either skip or read. This list is a numeric vector containing a list of dataset numbers. To skip a dataset, make the dataset number negative in DATSETS. To read only a given dataset list, pass the dataset list in as positive numbers. Positive dataset numbers in DATASETS implicitly assume that READUNV should skip all other datasets.

By default READUNV will read mode shapes in dataset 2414 into an `imat_shp`, and all other results into an `imat_result`. Two optional input strings can alter this behavior. Passing in the string `'asresult'` will cause READUNV to read all dataset 2414 results into an `imat_result`. Passing in the string `'shponly'` will cause READUNV to read only mode shapes from dataset 2414 into an `imat_shp`.

FNAME is an optional output argument containing the filename of the Universal file read.

The following universal file formats are supported:

Dataset	**Description**	**READUNV Output**
164	Units	none (used to adjust units of result)
58	Functions/time histories	`imat_fn` array
58b	Functions/time histories (binary)	`imat_fn` array
55	Shapes from Test	`imat_shp` array
2414	Shapes from Simulation	`imat_shp` array , and/or result object
1802	Coordinate trace	`imat_ctrace` object
757	DOF set	`imat_ctrace` object
247	DOF set	substructure entity (see below)
252	Entity definition matrix	substructure entity (see below)
611, 612	Matrix for the matrix package	substructure entity (see below)
2453	Results matrix	substructure entity (see below)
18	Coordinate system	`imat_cs` object in `imat_fem` object property (`.cs`)
2420	Coordinate system	`imat_cs` object in `imat_fem` object property (`.cs`)
15	Node	`imat_node` object in `imat_fem` object property (`.node`)
781	Node	`imat_node` object in `imat_fem` object property (`.node`)
2411	Node	`imat_node` object in `imat_fem` object property (`.node`)
82	Traceline	`imat_tl` object in `imat_fem` object property (`.tl`)
2416	Traceline	`imat_tl` object in `imat_fem` object property (`.tl`)
2431	Traceline	`imat_tl` object in `imat_fem` object property (`.tl`)
780	Element	`imat_elem` object in `imat_fem` object property (`.elem`)
2412	Element	`imat_elem` object in `imat_fem` object property (`.elem`)

It is also possible to write your own dataset reader plugins, so you can extend READUNV to import datasets not natively supported. Please refer to the Extending Readunv page for more information on this useful capability.

In most cases, only one type of output will be generated from a single universal file (this should be the case if the universal file was written by I-DEAS Test). Functions or time histories will be output in a single `imat_fn` variable with as many elements as necessary. Shapes will also be output in a single `imat_shp` variable. A universal file containing a single coordinate trace will be read into an `imat_ctrace` variable. A universal file containing multiple coordinate traces will be read into a cell array with one `imat_ctrace` in each cell element. The coordinate trace or DOF set name in the Universal file will be stored as the `imat_ctrace` name.

A substructure can be specified using different dataset types. Dataset 247 and 252 go together. If the substructure is stored using these datasets, the Universal file should contain both a dataset 247 (identifying the degrees of freedom in the substructure) and one or more datasets 252 (matrices). If the substructure is stored in dataset 2453, this dataset should appear consecutively with the degree of freedom matrix and entity matrices. Datasets 611 and 612 contain only matrices; the corresponding degree of freedom information is not available. When reading such a universal file, READUNV will create a MATLAB structure variable with the following fields:

`f.aset`	`imat_ctrace` of independent DOF (size Na)
`f.oset`	`imat_ctrace` of constrained (dependent) DOF (size No)
`f.sset`	`imat_ctrace` of restrained DOF (size Ns)
`f.mass`	Na x Na mass matrix
`f.stiffness`	Na x Na stiffness matrix
`f.viscous`	Na x Na viscous damping matrix
`f.hysteretic`	Na x Na hysteretic damping matrix
`f.constraint`	No x Na back expansion matrix

(Some of these fields may be empty if the associated matrix was not included in the substructure universal file.) Note that the s-set partition of the matrices will be discarded by READUNV, if present.

FEM data entities will be stored in an imat_fem with properties that contain individual FEM entity objects with the entity data (i.e. nodes and elements). For a full description of these structures, please refer to the FEM Data Format Reference. Please note that `readunv` will return the node ordering for parabolic elements in Nastran-centric ordering, which is different from the ordering specified in the Universal file. Also note that coordinate system 1 (global Cartesian) in the Universal file will be renumbered to 0 on import.

In the situation when mixed types of data are included in the same universal file, READUNV will create a cell array for the output argument. The cell array will contain, in this order, an `imat_fn` (all functions read), an `imat_shp` (all shapes read), the MATLAB structure variable described above, followed by one `imat_ctrace` per coordinate trace. Only those elements that were present in the universal file will be placed in the output cell array.

Some third party vendors write Universal files with incorrect formatting. In particular, dataset delimiter lines (containing '   -1') are written to have trailing spaces. READUNV automatically detects files with this formatting problem and utilizes a Perl script to clean the files. It creates a new file, given the filename with `'_fix'` appended to the end. READUNV will warn you if this happens and will proceed to read the cleaned file, which it leaves on the system after it is finished.

Examples

```
>> f=readunv('/ms5/examples/tda/air_func.unv');
>> f=readunv('*.unv','Select file to read');
>> f=readunv('*.unv','Select file to read',[55 58]);  % Read only functions and shapes
```

See Also

writeunv, writesubst

Extending READUNV

READUNV can easily be extended to import datasets that are not natively supported. To do so, simply create an m-file named using the convention *readunv_XXXX.m*, where XXXX is the Universal dataset number you wish to support. For example, the file to read dataset 180 would be called *readunv_180.m*. Each time READUNV is invoked, it will search for files in your MATLAB path that follow this format. You cannot override any of the datasets currently supported by READUNV. If any of these files match a dataset currently supported by READUNV, they will be ignored and you will receive a warning message.

Every time READUNV encounters a dataset supported by a custom routine, it will store the output in a new cell of the cell array output by READUNV. The contents of the cell will be the contents of the output from your*readunv_XXXX.m* function.

The following sections describe the functions that you can use to read data from the Universal file. You are also provided with an example function that shows you how to use several of the provided function calls.

READUNV API Function Calls

Several functions are provided to access the Universal file. These functions utilize buffered reads for increased performance, so you should use these rather than the MATLAB-provided `fread`, `frwrite`,`fscanf`, etc. Using these functions will result in unpredictable behavior. The functions provided for your use are listed below. These functions can be accessed by using function handles, which requires the use of the `feval`command.

The following functions are summarized in the following table, and are described in more detail below.

unv_getl	Read lines from a Universal file.
unv_scanf	Scan lines with the specified format to get up to a specified number of values.
unv_scands	Scan to the end of the dataset using with the specified format.
unv_err	Display an error message and terminate the Universal file read.
e_sscanf	Scan floating point data and return the results in a numeric vector.

unv_getl

Read lines from the Universal file.

Usage: `line=unv_getl(n)`

n is the number of lines to read. If ***n*** is not specified, it defaults to 1. ***line*** is a character array containing the lines read. It is a single vector, and line breaks consist of newline characters embedded in the string. If the end of the file is found before reading all ***n*** lines, an error results.

unv_scanf

Scan lines with the specified format to get up to a specified number of values.

Usage: `[x,xcount]=unv_scanf(s,nc,nl)`

This function is intended to be used with numeric data. ***s*** is the specified format (i.e. `'%d'` or `'%e'`). If the format is specified as `'%e'`, it will call `e_sscanf` internally. ***nc*** is the number of values to read, and ***nl*** is the maximum number of lines to read. ***x*** is a vector of numeric values read, and ***xcount*** is a count of the number of values read. If the end of the file is found before finding the end of the dataset, an error results.

unv_scands

Scan to the end of the dataset using with the specified format.

Usage: `[x,xcount,nlread]=unv_scands(s)`

This function is intended to be used for numeric data. It is a very efficient way to read in an entire dataset. ***s*** is the specified format (i.e. '%d' or '%e'). ***x*** is a vector of numeric values read. ***xcount*** is a count of the number of values read, and ***nlread*** is a count of the number of lines read. Both ***xcount*** and ***nlread*** should be checked for consistency and expected values. If the end of the file is found before finding the end of the dataset, an error results.

unv_err

Display an error message and terminate the Universal file read.

Usage: `unv_err(s)`

This function should be called if an error condition (such as an unexpected number of values read) is found when reading the Universal file. ***s*** is the string containing the error message to be displayed.

e_sscanf

Scan floating point data and return the results in a numeric vector.

Usage: `[a,...]=e_sscanf(s,...)`

This function expects a string ***s*** as input. The input and output arguments follow the calling convention of the MATLAB builtin function`sscanf`. `e_sscanf` is simply a wrapper that ensures that double-precision floating point representations using D are converted to E so that the numbers can be correctly converted by `sscanf`.

Function Format

The function header for your supplied universal reader function has a specific number of input and output arguments. The function header must follow the form

```
function out=readunv_180(fh,uid_cur,ufact_cur,uid,ufact);
```

fh is a structure containing function handles for the available functions described in the previous section. Each field of the structure matches the function name, and the content of each field is the function handle to that function. You will use `feval`to call each function, as described below. ***uid_cur*** is a scalar containing the number of the current units in the MATLAB session as specified by setunits. ***ufact_cur*** is a 4x1 vector containing the scale factors for the current units set. The contents of ***ufact_cur*** are defined by getunits. ***uid*** and ***ufact*** contain the units number and scale factors for the current units set specified by dataset 164 in the Universal file. These units factors are useful for converting the Universal file contents to the current units system when importing.

out can be defined however you like. The contents of out will be assigned to a cell in the cell array output by READUNV.

Calling the READUNV API functions

Since the READUNV API functions are internal to READUNV, the only way to have access to them is to use the function handles supplied in the input argument ***fh***. You can use `feval` to call these functions. For example, if you wanted to call unv_scanf, instead of calling it this way:

```
[x,xcount]=unv_scanf(s,nc,nl);
```

Using `feval`, you would call it this way:

```
[x,xcount]=feval(fh.unv_scanf,s,nc,nl);
```

Example

The following example shows how to use the API to read dataset 180, which is the Y2K mapping dataset written to the top of each Universal file exported by I-DEAS Master Series. Dataset 180 looks like the following, which has been truncated for brevity. The full dataset is 100 lines long.

```
-1
180
00 2000
01 2001
02 2002
...
68 2068
69 2069
70 1970
71 1971
...
98 1998
99 1999
-1
```

The following m-file, which can also be found in the examples directory of your IMAT installation, highlights several of the functions that can be used to read the datasets.

```
function out=readunv_180(fh,uid_cur,ufact_cur,uid,ufact);
% Read UNV dataset 180
%
% This example m-file provides a sample of how to write a Universal dataset plugin
% for READUNV.  It reads dataset 180, which is the Y2K mapping dataset written to
% the top of each Universal file written by I-DEAS Master Series.

% Update the progress dialog to specify the dataset currently being processed

ideas_progress(2,'Reading dataset 180 ...');
```

```
11
12  % Read a single line from the file to the string 'line'
13  % Display an error message and stop processing if the line is empty
14
15  line=feval(fh.unv_getl,1);
16  if isempty(line), feval(fh.unv_err,'Incomplete dataset 180');, end
17
18  % Convert the line into a number using sscanf.  We will use READUNV's e_sscanf function,
19  % which will also convert double-precision exponential format (1.0D+00) to E format, so
20  % MATLAB's sscanf function will work properly.  e_sscanf takes the same arguments as the
21  % builtin MATLAB sscanf function.
22
23  [a,count]=feval(fh.e_sscanf,line,'%d',2);
24  if count~=2, feval(fh.unv_err,'Incomplete dataset 180');, end
25
26  % Read up to 8 values from 4 lines using the '%d' format
27  % Print an error message and stop processing if 8 values were not read
28
29  [b,count]=feval(fh.unv_scanf,'%d',8,4);
30  if count~=8, feval(fh.unv_err,'Incomplete dataset 180');, end
31  b=reshape(b,2,[]);
32
33  % Read the entire dataset using the '%d' format.  This is the most efficient way to read
34  % the dataset.  Display error messages and stop processing if the incorrect number of values
35  % or lines were read.
36
37  [x,xcount,nlread]=feval(fh.unv_scands,'%d');
38  if xcount~=190, feval(fh.unv_err,'Incomplete dataset 180');, end
39  if nlread~=95, feval(fh.unv_err,'Incomplete dataset 180');, end
40
41  % Build the output and return
42
43  out=[a'; b'; reshape(x,2,[])'];
44  return;
```

Line 1 is the function declaration. The listed arguments are required, even if they are not used inside of the function.

Line 10 is a useful call that updates the string displayed in the progress dialog. You should call this to specify that you are beginning to read the specific dataset.

Line 15 called unv_getl, and line 16 displays an error message if an unexpected string (empty) is returned.

Line 23 called e_sscanf to convert the string read in line 15 to a vector of numbers. Line 24 displays an error message if an unexpected number of values is returned.

Line 29 called unv_scanf to read 4 lines into the numeric vector ***b***. It expects a total of 8 integers ('%d' format). Line 30 displays an error message if an unexpected number of values is returned. Line 31 reshapes the output to a 2x4 matrix.

Line 37 called unv_scands to read the rest of the dataset into a numeric vector. This function is the only one that needed to be called to read in this dataset, but the others were used to provide more examples. Lines 38 and 39 display an error message if an unexpected number of values is returned.

Line 43 assembles the dataset into a 100x2 matrix containing the data stored in dataset 180.

---

writeunv

Purpose

Export data to a Universal file.

Syntax

```
writeunv('filename',f1,f2,...)
writeunv('binary','filename','dialogtitle',f1,f2,...)
writeunv('noprogbar','filename',f1,f2,...)
writeunv('silent','filename',f1,f2,...)
writeunv('append','filename',f1,f2,...)
writeunv('binary','filename',f1,f2,...)
writeunv('filename','dataset15',f1,f1,...)
writeunv('filename','dialogtitle',f1,f2,...)
fname=writeunv(...)
```

Description

The WRITEUNV function is used to write functions, mode shapes, coordinate traces, and FEM geometry to a Universal file.

The filename argument is a string specifying the file to be written. If a wildcard is present in the filename, then a standard file dialog will be presented to allow the user to choose the output file. The filename may be omitted, in which case `'*.unv'` will be assumed.

If a dialog title is provided, it will be used to title the standard file dialog.

If you pass the string `'binary'` into WRITEUNV , functions will be written in binary format. If you pass in `'append'`, the new data will be appended to the end of the UNV file if it already exists. Otherwise a new UNV file will be created. Passing in `'dataset15'` will cause WRITEUNV to export nodes to dataset 15, rather than the default dataset 2411. Passing in `'silent'` will suppress output. The optional input string `'noprogbar'` will suppress the progress bar.

IF an output argument is requested, it will return the fully qualified filename of the file written.

The variables `f1`, `f2`, ... to be written can be any of the following:

Variable type	*Description*	*Datasets written*
`imat_fn`	Function or time history	58 (or 58b)
`imat_shp`	Mode shape	55
`imat_ctrace`	Coordinate trace	1802

`imat_cs`	Coordinate systems	2420
`imat_node`	Nodes	2411 or 15
`imat_elem`	Elements	2412
`imat_tl`	Tracelines	82
`imat_group`	Groups	2477

For function or shape arrays, WRITEUNV will write all functions or shapes to the corresponding dataset.

The universal file will be prefaced with a dataset 164 (units), and the data will be written in the currently selected unit system.

Note that the FEM global Cartesian coordinate system 0 will be renumbered to 1 prior to export, to be consistent with I-deas. If the FEM already contains a coordinate system 1, it will be renumbered to the maximum ID + 1. Nodes that reference either of these will be renumbered accordingly.

Examples

```
>> writeunv('/users/testdata/run01_modified.unv',f);
>> writeunv('binary','*.unv','Select binary file to write',f);
>> filename = writeunv('*.unv','Select file to write','append','silent',f);
```

See Also

readunv, writesubst

---

writesubst

Purpose

Create a substructure universal file for I-DEAS Test/Correlation

Syntax

```
writesubst('filename',v)
writesubst('filename','aset',aset,'oset',oset,'mass',maa,...)
```

Description

The WRITESUBST function allows you to create a substructure universal file that can be imported into I-DEAS Test to allow orthogonality calculations and back expansion. The universal file will contain a dataset 164 (units), dataset 2411 (nodes), dataset 247 (structural DOF), and datasets 252 (matrices). The resulting file should be imported into I-DEAS Test using File/Import and selecting 'Substructure Universal File'. You will need to select the same filename twice.

The filename argument is a string specifying the universal file to write. If wildcards are included in the filename, then a standard file dialog will be presented, allowing the user to choose the file.

The data to write to the universal file can be provided either in a MATLAB structure variable `v` (syntax 1), or with alternating entity names and values (syntax 2). The possible fields and entity names, and the associated values, are as follows:

`v.aset`	`imat_ctrace` of independent DOF (size Na)
`v.oset`	`imat_ctrace` of constrained (dependent) DOF (size No)
`v.sset`	`imat_ctrace` of restrained DOF (size Ns)
`v.mass`	Na x Na mass matrix
`v.stiffness`	Na x Na stiffness matrix
`v.viscous`	Na x Na viscous damping matrix
`v.hysteretic`	Na x Na hysteretic damping matrix
`v.constraint`	No x Na back expansion matrix

(Some of these fields may be omitted if the associated matrix or set is not to be included in the substructure universal file.) Note that the matrix rows and columns should be in the same order as the associated coordinate traces. The MATLAB structure is in the same format as that returned by readunv.

When writing the substructure universal file, WRITEUNV will move any s-set degrees of freedom into the o-set in order to minimize the size of the substructure. It will also fill in any missing degrees of freedom (including rotational degrees of freedom). This minimizes problems during back expansion in I-DEAS Test.

Important note: unlike most universal files, I-DEAS Test ignores the units in a substructure universal file. Therefore, it is important that you have the correct units selected in I-DEAS when you import the file.

Examples

```
>> v.aset=imat_ctrace('1x','2x','2y','2z');
>> v.mass=eye(4);
>> writesubst('test_subst.unv',v);
```

See Also

readunv, writeunv

---

readnas (+FEA)

Purpose

Import data from Nastran Output2, Output4, bulk data, and punch files.

Syntax

```
f=readnas('filename')
f=readnas
f=readnas('directoryonly')
f=readnas('supported')
[f,fname,opdir,opt]=readnas('*.op2','blocks',{'OUGV1','GEOM1'})
f=readnas(filename,'blocks',BLOCK_CELL,OPT,'silent','noprogbar','logfile')
f=readnas(filename,'modes',1:5)
```

Description

READNAS reads data from a Nastran file. READNAS supports Output2 (OP2), bulk data (BLK or DAT), PCH (XYPUNCH, SORT1, and SORT2), and Output4 (OP4) files. OP2 files must be created with the PARAM,POST,-2 or PARAM,POST,-1 option to read geometry and results. READNAS supports geometry, property, matrix, and nodal result data blocks. Geometry data is output in the IMAT geometry structure format. Physical and material property data is returned as structures. Results are output as `imat_shp` for nodal results, and as result objects for all other result types. Matrices are output as structures. The OGPWG table(s) are output as a structure, which is described in detail below. Unrecognized datablocks are read in as a structure of INT32 vectors.

READNAS determines the file type by the file's extension. If your file does not have a standard extension (i.e. .op2, .pch, .dat, .blk, .op4), READNAS will default to treating the file as BLK. To force READNAS to read the file as a different type, pass in the appropriate string listed below (i.e. `'isop2'`, `'ispch'`).

Several optional input arguments are supported. FILENAME is a string containing the filename to be read. It must include the extension of the file to read. If FILENAME is not specified, or FILENAME contains wildcards (i.e. '*.op2'), a standard file dialog will appear. If FILENAME contains wildcards, it will be used as a filter for the file dialog.

Several string input arguments are also supported.

`'blocks'`	Must be followed by a cell array containing data block names to read. If the cell array is scalar and contains an empty matrix, all of the datablocks will be read.
`'modes'`	Must be followed by a numeric vector of mode numbers to read from each SORT1 result datablock
`'subcases'`	Must be followed by a numeric vector of subcase IDs to read from each SORT1 result datablock
`'entityids'`	Must be followed by a numeric vector of entity IDs that will be used to filter results. These IDs represent node IDs for for nodal results and element IDs for elemental results, including Data At Nodes On Elements results such as stress. Works with PCH and OP2 results. Does not work with the `sort1tosort2` option.
`'silent'`	Suppress output
`'noprogbar'`	Don't display progress bar at all
`'directoryonly'`	Get directory of the file contents (OP2, OP4, and PCH)
`'supported'`	Return a list of the datablocks recognized by READNAS
`'logfile'`	Write external log file (.lis) containing processing details (default is no log file will be written)
`'asresult'`	Return all results as `imat_result` (OP2). The default is for nodal displacement results to be returned as `imat_shp`.
`'asraw'`	Return all table outputs as they are stored in the OP2 file. Unrecognized tables are always returned in this format. You can use one of PARSEOP2TABLE_* helper functions to process some of the tables

	returned in this format.
`'isblk'`	Force READNAS to treat the file as a bulk data file.
`'isop2'`	Force READNAS to treat the file as an Output2 file.
`'ispch'`	Force READNAS to treat the file as a PCH file.
`'isop4'`	Force READNAS to treat the file as an Output4 file.
`'psd'`	Label PCH file functions as PSDs.
`'frf'`	Label PCH file functions as FRFs.
`'time'`	Label PCH file functions as time histories.
`'isascii'`	Force READNAS to treat the OP4 file as ASCII.
`'isbinary'`	Force READNAS to treat the OP4 file as binary.

For PCH files, BLOCK_CELL can be a cell array containing either a single vector of numeric indices of records to read (an empty matrix specifies that all of the records should be read), or a cell array of block names (i.e. `'OUGV1'`, `'PCHBULK'`, etc.). The block names are those returned in the `.name` field of the PCH file directory structure (described here).

OPT is a structure containing special processing options for READNAS. You may also save this structure to a MAT file called `readnas_options.mat`. If READNAS finds this file in the current working directory, and the MAT file contains a structure named 'OPT', and you have not passed in an OPT structure, READNAS will use the OPT structure found in the MAT file. Supported options are shown in the table below. They are also more fully described below.

`OPT.global.femap`	Logical (default is false). Read results in a format suitable for FEMAP. This sets several of the other global options (see here for more details). Setting these other options explicitly will override the default FEMAP option settings.
`OPT.global.ideas`	Logical (default is false). Read results in a format suitable for I-deas. This sets several of the other global options (see here for more details). Setting these other options explicitly will override the default I-deas option settings.
`OPT.global.actualnodenumbers`	Logical (default is false). For Data At Nodes On Elements results (i.e. stress), return the actual node numbers for the result components, rather than 1-N.
`OPT.global.renumbercsys`	Logical (default is false). Renumber coordinate systems 0, 1, and 2. If set to TRUE, coordinate systems will be renumbered, but IMAT functionality such as plotting may not work.
`OPT.global.transformtobasic`	Logical (default is true). Transform results from the coordinate system in the OP2 file to the basic. See below for more details.
`OPT.global.atcentroids`	Logical (default is false). For stress, strain, and element force, return results at the centroid. These results are averaged from the nodal results for 1D elements, and are read directly from the OP2 file for 2D and 3D elements.
`OPT.global.atnodes`	Logical (default is true). For stress, strain, and element force, return results at nodes where they are available (for some linear 2D elements nodal results are not available unless results are output with the CORNER option).
`OPT.global.asraw`	Logical (default is false). Same as the `'asraw'` input option.
`OPT.global.combinesuper`	Logical (default is true). Combine superelement results. Also affects SOL200 results with

	multiple iterations, and also aeroelastic analysis where the output contains both structural and aerodynamic grids, which will be in separate records (set this option to false in this instance).
`OPT.fem.renumberpid`	Logical (default is true). Renumber duplicate property IDs. If set to false, no renumbering will be done. This could cause problems if exporting to external programs such as I-deas.
`OPT.pch.sort1tosort2`	Logical (default is false). Translate PCH file SORT1 records to functions (SORT2 format, equivalent to old behavior).

Four output arguments are supported. F is the returned data. It is a structure with fields that describe the output.

The following fields are possible for BLK, OP2, or PCH files.

Field Name	**BLK**	**OP2**	**PCH**	**Content description**
`.casecontrol`	X	X		Structure containing case control information, including sets from SET= cards (imat_group).
`.ceigen`		X		Structure containing the complex eigenvalue table (CLAMA, CLAMAD, FLAMA, or ULAMA).
`.eigen`		X		Structure containing the eigenvalue table (OLB, LAMA, or XLAMA).
`.fem`	X	X	X	imat_fem containing FEM geometry.
`.functions`		X	X	imat_fn containing results in SORT2 format.
`.loads`	X	X	X	Structure containing the different bulk data loads supported.
`.matrix`	X	X	X	Nx1 structure containing matrix information.
`.mpc`	X	X	X	Structure with the fields `.id`, `.name`, and `.mpc`. The `.id` field contains the list of unique MPC set IDs. The `.name` field contains a cell array with the MPC names. The `.mpc` field contains a structure with the fields `.sid` and `.terms`. The `.sid` field is a vector containing the set IDs for each MPC card imported. The `.terms` field is a cell array, where each cell is an Nx3 matrix of MPC equation terms.
`.properties`	X	X	X	Structure containing physical properties (`.ept`) and material properties (`.mpt`).
`.pvt`	X	X		Nx2 cell array containing the parameter value table. The first column contains the parameter names and the second column contains the parameter values.
`.records`			X	Cell array containing SORT1 results. Each cell contains the output for a given SORT1 record. Nodal results are returned as an `imat_shp`, and elemental results (i.e. stress) are returned as a structure.
`.result`		X		Cell array containing results (imat_shp or imat_result).
`.set`	X	X	X	Structure containing sets (e.g. ASET, BSET, CSET, QSET, USET) read from bulk data cards. Each set will be stored as an imat_ctrace in a separate field of `.set`. The `.uset` field is a cell array containing the sets for each named USET imported.
`.spc`	X	X	X	Nx4 matrix containing SPC definitions. Column 1 is SPC set ID. Column 2 is the

				grid ID. Column 3 is the component list, and column 4 is the enforced displacement value.
`.table`		X		Raw datablock tables (unsupported datablocks import in this format).
`.weight`		X		Weight generator tables(s).

Matrices read from the OP4 file will be returned as a structure array.

FILENAME is a string containing the full path and filename of the file read.

The OP2 directory structure is as follows:

`.blocknames`	Cell array of strings containing the block names.
`.blockdata`	nx6 matrix containing information about each block. The first column contains the starting word number for the block. The second column contains the number of words for the block and the third column specifies whether the block is a duplicate block. The fourth column gives the record type. The fifth column gives the superelement ID. The sixth column gives a number that depends on solution type: (load col num)(101) / mode #(103) / step #(106,112,129) / frequency #(111).
`.endianswap`	Specifies whether byte-swapping occurred due to the OP2 endian-ness being different from the current platform.

The PCH directory structure is as follows:

`.name`	nx1 cell array containing the record names (may be blank).
`.index`	nx3 matrix containing record indices. Generally speaking this field is only of interest to READNAS internally. The first column is the record number, the second column is the row number (only valid for SORT1 records being converted to functions), and the third column is the column number.
`.data`	nx12 matrix. The columns are as follows: Function Type, Subcase, Data Type, Entity Type, Entity Number, Item, Complex Type (1=real, 2=real/imag, 3=mag/phase), # Rows, #Columns, Sort Type, (201=MATRIX, 205=SORT1, 206=SORT2, 207=XYPUNCH, 208=PCHTABLED1, 209=SORT2_3COL, 303=BULK), Superelement ID, and Z Value (time/frequency/0 for static).
`.supported`	nx1 logical specifying whether the record is supported by READNAS.

Supported sort types are listed in the table below.

Sort Type Number	**Sort Type**
201	MATRIX
205	SORT1
206	SORT2
207	XYPUNCH
208	PCHTABLED1

209	SORT2_3COL
303	BULK

For OP4 files, the directory output is a cell array of strings containing the matrix names.

The physical and material property output structure is defined below. Both sets of property data are returned in the `.properties` field of the output. Physical property data is returned in the `.ept` field, and material property data is returned in the `.mpt` field. Each of these fields contains a structure, the format of which is described below. The following physical property cards are supported: PBAR, PBARL, PBEAM, PBEAML, PBEND, PBUSH, PDAMP, PELAS, PRBAR, PROD, PTUBE, PCOMP, PSHEAR, PSHELL, and PSOLID. The following material property cards are supported: MAT1, MAT2, MAT3, MAT4, MAT5, MAT8, MAT9, and MAT10.

`.id`	Nx1 vector of property ID's.
`.type`	Nx1 cell array of strings containing the property type. This matches the physical or material property card name.
`.data`	Nx1 cell array of structures. Each field contains a structure whose fields are specific to the property. There are too many properties and fields to describe here in detail. In general, field names will match the field description names in the Nastran documentation. To understand the fields and their contents, please look up the bulk data entry in the Nastran Quick Reference Guide.

The OGPWG (grid point weight generator) output structure is described below. If there are multiple superelements in the OP2 file, this structure will be an array, where each index contains the table for each superelement.

`.refid`	Reference grid point I, or 0 for the basic origin.
`.title`	Run title.
`.subtitle`	Run subtitle. Contains the superelement ID in the description if applicable.
`.label`	Run label.
`.MO`	Rigid body mass matrix relative to the reference point in the basic coordinate system. 6x6 matrix.
`.S`	Transformation matrix from the basic coordinate system to the principal mass axes. 3x3 matrix.
`.mass`	Principal masses. 3x1 vector.
`.cg`	Center of gravity in the X, Y, and Z directions, presented column-wise. 3x3 matrix.
`.IS`	Inertia matrix about the center of gravity relative to the principal mass axes. 3x3 matrix.
`.IQ`	Principal moments of inertia about the center of gravity. 3x1 vector.
`.Q`	Transformation matrix between the S-axes and the Q-axes. The columns are the unit direction vectors for the corresponding principal inertias. 3x3 matrix.

The LAMA (eigenvalue) output structure is described below. Each field is an NMODESx1 numeric vector, where NMODES is the number of modes extracted during the eigenvalue analysis.

`.modenum`	Mode number.
`.extorder`	Mode extraction order.

`.eigenval`	Eigenvalue (λ).
`.omega`	Omega (ω), which is the square root of the eigenvalue.
`.freq`	Frequency in Hz, which is omega divided by 2π.
`.genmass`	Generalized mass.
`.genstiff`	Generalized stiffness.

Bulk data file TABLED1 cards are imported into the `.tables` field. This field contains another structure containing the `.tabled1` field (in the future the other TAB* cards will be returned in appropriate fields). Each table is contained in a structure with the following fields.

`.id`	TAB* card ID.
`.xaxis`	X Axis type ('LINEAR' or 'LOG' or empty if not specified on the bulk data entry)
`.yaxis`	Y Axis type ('LINEAR' or 'LOG' or empty if not specified on the bulk data entry)
`.data`	Nx2 matrix containing the table values. The first column contains the X axis values, and the 2nd column contains the Y axis values.

OPT is a structure containing the processing options used during the import.

Important notes

The notes below refer to the default behavior. Specifics for each program mask are described further below.

Generally speaking, when you import results from an OP2 file, you need the geometry datablocks present so READNAS can access information relevant to the result processing. If you opt not to transform results to the basic coordinate system or return actual node numbers, you can import nodal and energy results without the geometry.

Nodes will be imported in the definition coordinate system when read from both bulk data files and Output2 files. SPOINTs will import as nodes with the color yellow and coordinates 0,0,0.

Matrices will be contained in a structure, where each element of the structure contains the contents of one of the matrices read. The structure fields are described in the following table. If the matrices being read from the Output2 file were created by the substructure DMAP alter provided by MTS for creating a Guyan-reduced substructure (KXX, MXX, and GO), READNAS will also fill in the DOF field. Otherwise this field will contain empty DOF lists, since Nastran does not store matrix DOF lists directly in the Output2 file.

`.matrix`	Matrices in sparse format.
`.name`	Matrix names.
`.dof`	Row and column DOF, respectively, stored as a 1x2 cell array of `imat_ctrace`.
`.phase`	Logical. True if the matrix is complex-valued and values are given in magnitude/phase form.
`.symmetric`	Logical. True if the matrix is symmetric. READNAS will return the fully populated matrix.
`.matpool`	Logical. True if the matrix was read in from Nastran DMIG and has value 0 if matrix was read in from Nastran DMI. Matrices read from an OP2 file will always have a value of 0.

If you have a DMI matrix, and wish to obtain a matrix where the row and column numbers match the row and column numbers in the DOF fields, you can use the expandDMI function.

CONM1 and CONM2 mass and inertia properties will be imported as physical property tables.

To read a subset of SORT1 results from the OP2 file, use the 'subcases' and/or `'modes'` specifiers. These act as filters on the results. For modes runs, 'modes' specifies the mode(s) to read. `'subcases'` allows you to specify results from specific subcase IDs to read. The subcase numbers match the number specified on the SUBCASE case control cards.

BLK File Output

READNAS reads most information from a bulk data file, including parts of the file management section and executive and case control. It handles INCLUDE statements as well. Relative paths on an INCLUDE statement are treated as relative from the top-level bulk data specified to READNAS.

If bulk data includes a comment line that has the form "$TITLE = #", where column 79 contains a digit, it will assume that this is the start of a punch data record. If this is not intended, then please modify the format of the comment line so that READNAS interprets it correctly as a comment.

PCH File Output

READNAS considers Punch file data to be stored as a series of records. A record consists of a header line or lines, followed by numeric data. SORT1 records are written as a matrix, where the rows are the entity (i.e. node or element), and the columns are the item codes. One record is written for each time or frequency step, and for each step, one record is written for each entity type. SORT2 records are written as a matrix, where the rows are the time or frequency, and the columns are the item codes. One record is written for each entity type. XYPUNCH records are written as a 2-column matrix, where the first column is the time or frequency, and the second column is the data for a specific entity and item code. The XYPUNCH header contains very little information, so for example it is not possible to determine whether the first column contains time or frequency values. READNAS thus allows you to specify what this axis contains.

BULK data found in PCH files will be returned in the fields that are returned when reading bulk data directly. Each section of bulk data found in the PCH file will appear as a separate record. If the same bulk record types are found in multiple bulk sections, READNAS will attempt to merge them. All matrices will be returned, even if they are duplicated. Only the first set of FEM geometry will be returned. Only the first set of each set type will be returned.

Nonlinear CBUSH results are written to the PCH file as a stress record for element type 226. The data has 18 values. The first 9 are translational forces, stresses, and strains, in that order. The following 9 are the rotational forces, stresses, and strains. Note that the Output2 file output is ordered differently. Please see the notes below for datablock OESNLXR.

Punch file results are reordered in the same way, such that the first result values are force, the next 6 are stress, and the final 6 are strain.

It is possible to have READNAS return the SORT1 records in SORT2 format. To do this, pass in the OPT structure with the `.pch.sort1tosort2` field set to `true`.

SORT2 and XYPUNCH records (and SORT1 records converted to functions) are returned in the `.functions` field of the output as `imat_fn`.

Matrix records are returned in the `.matrix` field as an array of structures using the format specified above.

SORT1 records are returned in the `.records` field as a cell array. Each cell contains the data for a specific record. If the record contained nodal data (item codes 3-8), it will be stored as an `imat_shp`. Otherwise, it will be stored as a structure with the following fields:

`.title`	String containing Nastran run title.
`.subtitle`	String containing Nastran run subtitle.
`.label`	String containing Nastran run label.
`.id`	mx1 vector containing entity IDs (i.e. node or element) corresponding to the data rows.
`.item`	1xn vector containing the item codes for the data columns.
`.entitytype`	Scalar specifying the entity type. The corresponds to the Nastran entity type (i.e. CBAR=34).
`.seid`	Scalar specifying the Superelement ID (if specified in the PCH file).
`.datatype`	Scalar specifying the data type. This corresponds to the I-deas nomenclature, which is documented here.
`.subcase`	Subcase number.
`.modenum`	Mode number for modal results or Load ID for static results.
`.zvalue`	Time value or frequency or mode frequency (Hz), depending on the analysis type.
`.eigenvalue`	For modes, contains the eigenvalue. For complex modes, this value with be complex.
`.acode`	Nastran approach code.This is the Analysis Code*10 + Device Code. Please consult your Nastran documentation for more details.
`.data`	MxN matrix containing the numeric SORT1 items, where M is the number of IDs (length of `.id`), and N is the number of values in `.item`, and the contents of each column corresponds to the item code in `.item` for this result. Columns defined with string-based items are set to NaN, and the string contents are stored in the same column in `.datachar`.
`.datachar`	MxN cell array of strings containing the string-based items, where M is the number of IDs (length of `.id`), and N is the highest column number containing a string-based item. N will always be equal to or less than the length of `.id`. Columns with numeric items contain empty matrices. If this result type does not have any string-based item codes, this field contains an empty matrix.

Output2 Import Notes

This section contains general information about how Output2 stores its data. The following subsections define how the import options modify the results from the way they are stored in the Output2 file. If no transformation is requested, results will be returned the way they are stored in the Output2 file. Please see below for details of what transformations occur if they are requested.

Nodal displacement results (OUG) are stored in the displacement coordinate system unless their datablock name begins with 'B', in which case the results are stored in the basic coordinate system.

Nodal force results (OQG) are stored in the displacement coordinate system.

Element forces read from the OEF datablocks are always imported with some coordinate system transformation. 1D elements such as CBAR and CBEAM are always imported in the element coordinate system. Forces are transformed according to the cross section rotation angle. Eccentricities are also accounted for. CELAS and CBUSH forces are always imported as they are stored in the Output2 file. **Please note that for bar/beam elements, Nastran defines moments IN a plane, where IMAT**

defines moments ABOUT an axis perpendicular to the plane. `readnas` returns the moment in plane 1 as *Mz* and the moment in plane 2 as *My*, which means that the moment values will appear flipped from what Nastran reports. There may also be a sign change on the moments, depending on the program mask. Please read the sections on I-deas and FEMAP masks for more important information.

Nastran does not write out stress tensors for 1D elements. IMAT returns the stresses as components of a tensor, but the values returned are not tensor components.

Element Type	**Tensor components and actual storage**	

`CBAR (linear, type 34)`
`CBEND (linear, type 69)`
`CBARAO (linear, type 100)`
	Tensor Component	**Element stress component**

	`XX`	Axial stress
	`XY`	Bending stress at point C
	`YY`	Bending stress at point D
	`XZ`	Bending stress at point E
	`YZ`	Bending stress at point F
`CBEAM (linear, type 2)`	**Tensor Component**	**Element stress component**
	`XY`	Longitudinal stress at point C
	`YY`	Longitudinal stress at point D
	`XZ`	Longitudinal stress at point E
	`YZ`	Longitudinal stress at point F

`CONROD (linear, type 10)`
`CROD (linear, type 1)`
`CTUBE (linear, type 3)`
	Tensor Component	**Element stress component**

	`XX`	Axial stress
	`XZ`	Torsional stress
`CROD (nonlinear, type 89)`	**Tensor Component**	**Element stress component**
	`XX`	Axial stress
	`XY`	Equivalent stress
	`YY`	Total strain
	`XZ`	Effective plastic

<table>
<tr><td></td><td><table>
<tr><td></td><td>strain</td></tr>
<tr><td><code>YZ</code></td><td>Effective creep strain</td></tr>
<tr><td><code>ZZ</code></td><td>Linear torsional stress</td></tr>
</table></td></tr>
<tr><td><code>CBEAM (nonlinear, type 94)</code></td><td><table>
<tr><th>Tensor Component</th><th>Element stress component</th></tr>
<tr><td><code>XX</code></td><td>Axial stress</td></tr>
<tr><td><code>XY</code></td><td>Equivalent stress</td></tr>
<tr><td><code>YY</code></td><td>Total strain</td></tr>
<tr><td><code>XZ</code></td><td>Effective plastic strain</td></tr>
<tr><td><code>YZ</code></td><td>Effective creep strain</td></tr>
</table></td></tr>
</table>

The OESNLXD datablock contains nonlinear output for different element types. NASTRAN mixes the datatypes stored in this datablock. Since results in IMAT can only have a single datatype, READNAS returns OESNLXD results as a force dataset. The implications are that if you export these results to another format (e.g. Universal file), and then change units and read the results back in, non-force results will be scaled incorrectly. The table below describes how the results for the supported element types are returned.

Element Type	**Output storage**
`CGAP`	The first 3 components (1 through 3) contain the X, Y, and Z forces, respectively. Components 4 through 6 contain the X, Y, and Z displacements. The open/close state and the two slip values are not returned from the Output2 file, but are imported from the Punch file.
`CROD`	The 6 components are returned in order of Axial stress, Equivalent stress, Total strain, Effective plastic strain, Effective creep strain, and Linear torsional stress.

Nonlinear CBUSH results are written to the OESNLXR datablock. READNAS returns a stress dataset. Layer 0 contains the force results. Layer 1 contains the stresses, and layer 2 contains the strains. This is a slight reordering from the way the results are stored in the Output2 file, where the results are stored as all translations, then all rotations. Please note that the resulting output is tagged by READNAS as a force, so changing the units on export/import may result in incorrect values for the stresses and strains. Also note that the PCH file import retains the Nastran ordering. See the notes above.

For 2D elements, element stress resultants are stored in the element coordinate system. See the CQUAD and CTRIA documentation in the Nastran Quick Reference Guide for more details on the definition of the element coordinate system, as well as the material coordinate system. Midside node results for parabolic elements are averaged from the two adjacent corner nodes, since the Output2 file does not store midside node results.

Stress and strain data is stored in the Output2 file in the element coordinate system for all element types except PCOMP. Stresses and strains for composite shell elements (PCOMP) are written to the Output2 file in the ply coordinate system.

Grid point force balances are written to the Output2 file in the basic coordinate system. See the notes for GPFORCE in the Nastran documentation. These results are returned as Data on Elements at Nodes results. Special quantities are identified with a non-positive element ID. Please refer to the table below for a list of these quantities and their associated "element" identifier.

Element ID	Quantity
`>0`	Element number
`0`	SPC (reaction) forces
`-1`	Applied loading at node
`-2`	MPC/Couple forces
`-3`	Total force

Output on each element is ordered in the nodal order expected by I-deas. The node order is the same between I-deas and Nastran for linear elements. However, parabolic elements have a different numbering sequence. I-deas interleaves corner and midside nodes, and Nastran orders the corner nodes first, followed by the midside nodes. The node numbering listed in the component listing follow the Nastran convention, but are ordered in the I-deas convention. This is so exporting these results via writeunv will result in a correctly ordered Universal file result dataset. To clarify the ordering, consider a CTRIA6 with corner nodes 1, 2, and 3, and midside nodes 4, 5, and 6. The component listing for the results at this element will be in node order 1 4 2 5 3 6.

Option: FEMAP

This option sets the **transformtobasic** option to *false*, **atnodes** to *true*, **atcentroids** to *true*, and **actualnodenumbers** to *false*. This combination has the effect of returning the data much more closely to the way it is stored in the OP2 file.

Only one spring force is returned for each spring element, and is returned at node 0, the "centroid".

CBAR and CBEAM element Y and Z moment values are switched as compared to the way they are stored and printed in the Nastran output (for example, F06), since IMAT defines beam moments about an axis and Nastran defines moments in a plane. The signs on the moments will match what is displayed in the F06 file. Forces and moments are returned for nodes 1 and 2 (end A and end B of each element), as well as the "centroid". The centroidal values are the average of the node 1 and node 2 values. The axial and shear forces are the same for both nodes, since Nastran only stores one value per element for the axial, shear, and torque. However, the moments may be different.

Shell stress resultants are returned with a layer number of 100. This layer number is used to identify stress resultants when writing a FEMAP Neutral file with writefemap.

Only results at the corners of parabolic elements are returned. No midsize results are returned, since Nastran does not store midside node values in its output.

Option: I-deas

This option sets the **transformtobasic** option to *true*, **atnodes** to *true*, **atcentroids** to *false*, **renumbercsys** to *true*, and **actualnodenumbers** to *false*.

Nodal displacement data read from the OUG datablocks (displacements and pressures) is imported from the Output2 file in the global (basic) coordinate system if geometry information is present in the Output2 file.

Nodal (SPC) forces read from the OQG datablocks are transformed to the global (basic) coordinate system if geometry information is present in the Output2 file.

Midsize node results will be returned for parabolic elements, but since Nastran does not store midside results, these are simply the average of the two corner nodes on that edge.

For bar/beam/bend element forces, the end A *Mz* and end B *My* moments have their signs negated from what is stored in the Output2 file and printed to the F06 file. In addition, the end A axial and shear forces and end A torsion are negated from what is stored in Nastran.

Output on each element is ordered in the nodal order expected by I-deas. The node order is the same between I-deas and Nastran for linear elements. However, parabolic elements have a different numbering sequence. I-deas interleaves corner and midside nodes, and Nastran orders the corner nodes first, followed by the midside nodes. The node numbering listed in the component listing follow the Nastran convention, but are ordered in the I-deas convention. This is so exporting these results via writeunv will result in a correctly ordered Universal file result dataset. To clarify the ordering, consider a CTRIA6 with corner nodes 1, 2, and 3, and midside nodes 4, 5, and 6. The component listing for the results at this element is in node order 1 4 2 5 3 6.

Option: transformtobasic

Coordinate transformations require that geometry information is present in the Output2 file. If this information is not present, results will be returned as they are stored in the Output2 file.

Nodal displacement data read from the OUG datablocks (displacements and pressures) will be transformed to the global (basic) coordinate system. If these results were written to the Output2 file in the basic coordinate system (datablock name begins with 'B'), this option has no effect, since the results are already in the basic coordinate system.

Nodal (SPC) forces read from the OQG datablocks are transformed to the basic coordinate system.

Elemental forces for 1D elements such as CELAS, CBUSH, CBAR and CBEAM are transformed to the basic coordinate system.

Element stress resultants for 2D elements are transformed from the element coordinate system to the basic coordinate system. The transformation angle is defined as the angle between the element X axis and the basic X axis. If the element does not project onto the basic X axis, the basic Z axis is used instead. See the CQUAD4 documentation in the Nastran Quick Reference Guide for more details.

Stress and strain data is always returned in the element coordinate system for 1D elements. For 2D and 3D elements, it is returned in the basic coordinate system for most elements. For CHEX1, CHEX2, CWEDGE, CTRAPRG, and CTRIARG elements, the data is always returned as it is stored in the Output2 file.

Stresses and strains for composite shell elements (PCOMP) are transformed from the element coordinate system (defined by the element first edge) to the basic. This means that there is still a transformation from the ply angle to basic.

Heat flux and heat gradient data for 1D elements will be returned exactly as it is stored in the Output2 file. Data for 2D and 3D elements is transformed to the basic coordinate system.

Grid point force balance (GPFB) data is transformed to the basic coordinate system.

OP2: Generic Tables

Tables not explicitly recognized by READNAS can still be imported. They will be stored in the `.table` field of the output. This field contains one field for each unrecognized table read. The field names are the names of the tables read. Each of these fields is a structure with three fields, `.header`, `.record` and `.trailer`. The `.record` field is a cell array, where each cell contains an INT32 vector corresponding to the records in the table. The `.trailer` field is a 1x6 int32 array containing the 6 trailer words for the table. The `.header` field is a structure with four fields. This structure contains the actual contents of the 16-word header found in OP2 tables. The `.name` field is an 8-character array corresponding to the table name. This corresponds to the first 2 words in the header. The `.fulltrailer` field contains the 7-word trailer that immediately follows the table name. The last 6 words of this trailer correspond to the contents of the `.trailer` field. The `.name2` field contains the contents of the next two words in the header. In most cases this should match the `.name` field, but it does not always do so. The `.date` field contains the last 5 words of the header, which is the date string. In many cases this is empty since it is not set in the OP2 file.

To properly interpret the table records and trailer, please visit the Nastran DMAP Programmer's Guide. The record data is read in as INT32. Since it can be real (float32), double, or character data, it will likely be necessary to recast the data into the appropriate datatype. READNAS can read in Output2 files created on platforms with different endian-ness from the platform on which it is running. READNAS will perform the necessary byte-swapping based on 4-byte words. The `.endianswap` field of the DIR structure specifies whether READNAS performed byte-swapping during the read. IMAT provides the function TYPECAST_OP2 to typecast data from INT32 to another data type, correctly handling any byte-swapping necessary.

IMAT provides several helper routines for processing some of tables in raw format. Please refer to the PARSEOP2TABLE_* functions for useful examples of how to process raw OP2 tables.

Examples

```
>> f=readnas('/users/testdata/output.op2','blocks',{'BOUGV1','OEF1'});
>> s=readnas('*.dat','silent');
```

See Also

expandDMI, readodb, readadf, readunv, setunits, getunits, parseop4, parseop2table_dmig, parseop2table_sets, typecast_op2

---

writenas (+FEA)

Purpose

Write to a Nastran Output2 file.

Syntax

```
writenas(table)
filename=writenas(filename,table,'append','silent','noprogbar','logfile')
```

Description

WRITENAS writes tables to a Nastran OP2 file.

TABLES is a structure containing the output tables and/or matrices to write. The structure must have fieldnames equal to the names of the tables/matrices to write. Each of these fields must be a structure containing the field `'trailer'`. `'trailer'` is a 1x6 INT32 vector containing the table trailer values.

Table structures must also contain the field `'record'`. `'record'` is a cell array of INT32 vectors containing the record contents.

Matrix structures must also contain the fields `'colstart'`, `'rownum'`, and `'start'`. These all contain INT32 vectors containing the sparse format information that defines the matrix.

You are responsible for handling endian formatting of the data. If you are creating a new OP2 file there won't be any endian issues because WRITENAS will create the file with the platform-native endianness. If you are appending to an existing file, you should check for the endianness of the OP2 file before writing to it, and handle your data accordingly.

Several optional input arguments are supported. FILENAME is a string containing the filename to be read. It must include the extension of the file to write. If FILENAME is not specified, or FILENAME contains wildcards (i.e. '*.op2'), a standard file dialog will appear. If FILENAME contains wildcards, it will be used as a filter for the file dialog.

Several string input arguments are also supported.

`'append'`	Append to an existing OP2 file if it exists
`'silent'`	Suppress output
`'noprogbar'`	Don't display progress bar at all
`'logfile'`	Write external log file (.lis) containing processing details (default is no log file will be written)

FILENAME is a string containing the full path and filename of the file written.

See Also

readnas, createop2_table

---

writeop4 (+FEA)

Purpose

Write to a Nastran Output4 file.

Syntax

```
writeop4(mat)
filename = writeop4(fname,mat,ndigits,'append','silent')
```

Description

WRITEOP4 exports matrices to Nastran Output4 (OP4) format. Currently WRITEOP4 handles real square or rectangular matrices.

FNAME is an optional input string containing the name of the OP4 file to write. If not supplied, or it contains wildcard characters ('*' or '?'), the user will be prompted with a graphical file dialog.

MAT is a structure containing the matrices. It must at minimum have the following fields:

`.name`	String (8 characters maximum) containing the matrix name.
`.matrix`	Matrix to export.

NDIGITS specifies the number of digits of precision desired. The default is 7. WRITEOP4 will determine the appropriate format specifier based on this.

The optional string `'silent'` suppresses output. The optional string `'append'` causes WRITEOP4 to append to the file if it exists. The default is to overwrite.

FILENAME is a string containing the full path and filename of the file written.

See Also

readnas

---

create_op2table (+FEA)

Purpose

Create tables for use with WRITENAS.

Syntax

```
table=create_op2table(shp)
table=create_op2table(result,{shp,'BOPHIG'})
```

Description

CREATE_OP2TABLE creates a structure in the format required by WRITENAS for export to a Nastran OP2 file.

Each input can either be an `imat_shp` or `imat_result` object. If you pass in the object directly, CREATE_OP2TABLE will select a table name to use. You can force the table name by passing in a 1x2 cell array, where the first cell is the `imat_shp` or `imat_result`, and the second cell is a string containing the table name.

TABLE is a structure properly formatted for WRITENAS. Each field is the table name that will be exported, and the contents of each field is a structure containing the table data that will be written.

You can import tables into this format by calling READNAS with the `'asraw'` option. This is useful if you just want to make modifications to existing OP2 data.

Supported blocks:

Category	**DataLocation**	**Notes**
OEE	Data On Elements	This includes strain and kinetic energy datablocks (ONRGY1, ONRGY2). Nastran writes out datablocks per element type, but Data On Elements do not have an element type component. This function writes the data for all elements to a single record. One way around this is to set the SEID for each element to the element type.

OES1	Data At Nodes On Elements	This includes stress and strain datablocks such as OES1 and OSTR1X. The following element types are supported: 39 (CTETRA) 67 (CHEXA) 68 (CPENTA) 74 (CTRIA3) 144 (CQUAD4)
OUG1, OQG1	Data At Nodes	This includes disp/vel/acc datablocks (BOPHIG, OUGV1, BOUGV1, RADCONS, etc.), temperature datablocks (TOUGV1), and SPC/MPC force datablocks (OQG1, OQMG1).

Notes:

- If a table name is not directly entered, one will be assumed based on the data. For some desired blocks, such as RADCONS, if the table name is not directly entered then the FEA program reading the OP2 file may misidentify the results.
- CREATE_OP2TABLE uses IDLine2 of the shape for the TITLE, IDLine3 for the SUBTITLE, and IDLine4 for the LABEL.
- If the input is an `imat_shp`, and it has any modes with non-zero or non-unity frequency, CREATE_OP2 will also create a LAMA table.

See Also

readnas, writenas

---

readneu (+FEA)

Purpose

Import data from FEMAP Neutral files.

Syntax

```
f=readneu
f=readneu(filename,datasets,'silent')
f=readneu('neutral_file.neu',[403 404])
f=readneu('silent')
f=readneu('noprogbar')
```

Description

READNEU reads datasets from FEMAP Neutral (.neu) files.

FILENAME is an optional input string containing the filename to read. If not supplied, you will be prompted for the filename with a graphical dialog.

DATASETS is an optional numeric vector containing the dataset numbers to import. If the dataset number is negative, READNEU will read datasets except for that one. For example, if DATASETS is -404, READNEU will read all datasets except for dataset 404.

Passing in the string `'silent'` suppresses output to the screen during processing.

Passing in the string `'noprogbar'` suppresses the progress bar during processing.

The following datasets are supported:

Dataset Number	**Description**	**Notes**
`403`	Node	
`404`	Element	See below for a detailed list of supported element types. Note that properties from dataset 402 will be imported and applied to the element output if it exists.
`405`	Coordinate system	
`408`	Groups	

F is a cell array containing the outputs. FEM entities will be returned as an imat_fem, and group entities will be returned as an imat_group. If nothing was read, DATA will be -1.

Supported Element Types

READNEU does not support all element types. The following table lists the element types not currently supported.

FEMAP Element Type	**Name**
`4`	Link
`6`	Spring
`7`	DOF Spring
`10`	Plot only
`11`	Shear
`12`	Shear (parabolic)
`28`	Mass mat
`29`	Rigid
`30`	Stiff mat
`32`	Plot plate
`33`	Slide line

`34`	Contact
`35`	Axisymmetric shell (quad only)
`36`	Axisymmetric shell (parabolic)
`38`	Weld

Examples

```
>> f = readneu('test.neu', [405 408])
>> f = readneu(404)
>> f = readneu('test.neu', 'silent')
```

See Also

writefemap, readunv, writeunv, imat_elem/elemtype_f2i

---

writefemap (+FEA)

Purpose

Export functions, shapes, results, groups, and FEM geometry to a FEMAP Neutral file or directly to a FEMAP session through the COM interface.

Syntax

```
writefemap('filename',f1,f2,...)
writefemap(happ,f1,f2,...)
writefemap('noprogbar','filename',f1,f2,...)
writefemap('silent','filename',f1,f2,...)
writefemap('append','filename',f1,f2,...)
writefemap('filename','title',title,f1,f2,...)
writefemap('*.neu','fnoffset',10,'outoffset',100,f,r,...)
```

Description

WRITEFEMAP writes the supplied entities to a FEMAP Neutral file or alternately directly to a FEMAP session. If the file already exists, it will be overwritten.

FILENAME is an optional string containing the filename to write. If it is not supplied, or it contains wildcards, the user will be given a standard file dialog to determine the output filename. The resulting filename is returned by WRITEFEMAP if outputs are requested.

If the user passes in a FEMAP COM object HAPP instead, the entities will be written directly to the FEMAP session instead of the Neutral file. You can create a new FEMAP session with the command

```
happ = actxserver('femap.model')
```

Alternately, you can connect to an existing session with the command

```
happ = actxGetRunningServer('femap.model')
```

Several optional input arguments are supported.

`'silent'` suppresses output when writing the file. `'noprogbar'` disables the progress bar when writing.

`'append'` will write the new data to the end of the Neutral file if it exists. The default action is to overwrite the file.

`'title'` followed by a string TITLE overrides the default file dialog title.

`'fnoffset'` followed by an integer scalar will apply this offset to the function ID used when writing functions to the Neutral file. The default ID starts at 1 and increments by 1. This is important because if any functions already exist in the .mod file with the same IDs that are in the Neutral file, they will be overwritten by the Neutral file contents.

`'outoffset'` followed by an integer scalar will apply this offset to the output set ID used when writing functions to the Neutral file. The default ID starts at 1 and increments by 1. This is important because if any output sets already exist in the MODFEM file with the same IDs that are in the Neutral file, they will be overwritten by the Neutral file contents.

The following entities can be written. The dataset number is the Neutral file dataset if the output is a Neutral file.

`imat_fn`	All records are written to dataset 420
`imat_shp`	All shape records are written using datasets 450 and 451
`imat_result`	All records are written using datasets 450 and 451
`imat_fem object`	Coordinate systems are written to dataset 405. Nodes are written to dataset 403. Elements are written to dataset 404.
`imat_group object`	Groups are written to dataset 408. See below for specifics on the group object format and contents.

Important Notes

Use readnas with the FEMAP program mask (`opt.global.femap=true`, or in Options | Program Mask: FEMAP) to read data from the OP2 file in a format suitable for FEMAP.

In general, results should be in the format stored in the OP2 file. This generally means that results are stored in local coordinates.

When reading the Neutral file into FEMAP, data (i.e. functions and output sets and output vectors) with the same IDs as the data in FEMAP will overwrite any data in FEMAP. Care must be taken when importing these results. WRITEFEMAP gives you control over the IDs written to the Neutral file.

Below are notes specific to the translation of each data type. In general WRITEFEMAP attempts to duplicate the FEMAP numbering convention for output results.

imat_fn

IMAT analyzes the `imat_fn` AbscissaDataType, AbscissaTypeQual, OrdNumDataType, and OrdNumTypeQual attributes to determine the function type. IDLine1 is used for the function title.

imat_shp

The output set title is derived from the OrdNumDataType, Frequency, and Damping attributes. WRITEFEMAP will write out the individual 3 or 6 component directions. It also calculates and writes the magnitude vectors for translation and also rotation where appropriate. These vectors are necessary for FEMAP to be able to animate the shapes.

WRITEFEMAP handles both real and complex shapes.

imat_result

FEMAP stores all results as individual vectors of components and data types. All of the vectors of all of the data types are stored under a specific output set that contains the overall title and notes for that result.

WRITEFEMAP writes each result to its own output set. Both real and complex results of this type are supported.

Data At Nodes results are written the same way as `imat_shp` output.

Data At Nodes On Elements results are much more difficult to write. Because FEMAP does not calculate derived results such as von Mises stress on the fly. Any results that need to be displayed must already exist as a specific output vector. WRITEFEMAP does not calculate derived results as it exports, so only the individual components will be available for display. Both centroidal and nodal results will be written.

FEMAP writes individual output vectors for spring, beam, shell, and solid results. It also writes vectors containing the shell top and bottom fiber distances. WRITEFEMAP uses the element type information to determine where to write the results. The element types follow Nastran Item Code convention. These are set correctly by readnas when importing Output2 results. If the imat_result came from a different source, you must ensure that this component is set correctly. Elements with layer=100 are assumed to be shell stress resultants. Elements with more than 2 layers are assumed to be composites. Midside node results are not used by FEMAP, so they must not be present when exporting.

Data On Elements results are fairly simple. Each data component in the result is written to a separate output vector. The output vector title is derived from the data type and component name. Vector ID's start at 9,000,000. The component number is added to this.

The best performance will be achieved if the nodes are numbered 1-n for each element.

imat_fem object

Coordinate systems, nodes, and elements are supported. Nodes are first transformed to the global Cartesian system before writing. Colors are mapped from the I-deas color designation to the closest corresponding FEMAP color. User-defined colors are mapped to the default colors.

FEMAP requires that all spring elements have 2 nodes, even if they are springs to ground. When FEMAP imports a Nastran deck it actually creates a new fully SPC'd node for springs to ground. WRITEFEMAP simply uses the spring node twice.

Some element data is lost in translation. In particular, items such as end releases and beam rotation angles and end offsets are not carried over. This is because IMAT does not have a place to store this information. Geometry export to FEMAP is primarily intended so the FEM can be used for display purposes.

imat_group object

Entities currently supported by WRITEFEMAP are 1 = coordinate systems, 7 = nodes, and 8 = elements.

Examples

```
>> writefemap('/users/testdata/run01_modified.neu',f,g,h);
>> writefemap('*.neu','Select file to write','append','silent',f);
```

See Also

readneu, readnas, writenas, readunv, writeunv

---

femap_invoke (+FEA)

Purpose

Invoke a FEMAP API method through the COM interface.

Syntax

```
[RC[,OUT1,OUT2...]]=FEMAP_INVOKE(OBJ,METHOD,[,ARG1,ARG2,...])
```

Description

FEMAP_INVOKE calls a method on the FEMAP object specified in OBJ. This method works basically identically as MATLAB's native method, but is necessary for methods that have both inputs and outputs in the argument list.

FEMAP_INVOKE will first search through its supported list of methods to see if it is one with input and output arguments that has already been implemented. If so, it will call a special handler for the method. For these calls, you must always pass the input arguments to the function in ARG1, ARG2, and so on. The output arguments are specified in the output to FEMAP_INVOKE.

In addition to the methods described in the FEMAP API, FEMAP_INVOKE does provide some additional methods for more efficiently getting and putting object properties. The GetProperties method will return a structure containing all of the object's properties, and the SetProperties method accepts a structure of the same form. It will set the properties for all of the fields of the input structure that it recognizes. The objects for which these methods are provided are shown below.

Object	**Method**	**Description**
`Output`	[RC,OUT] = femap_invoke (OP,'GetProperties')	OP is a feOutput object. It must have its ID and SetID properties set appropriately. OUT is a structure containing the properties.
`Output`	`RC=femap_invoke (OP,'PutProperties',SPROP)`	OP is a feOutput object. It must have its ID and SetID properties set appropriately. SPROP is a structure containing the properties.
`OutputSet`	`[RC,OUT] = femap_invoke (OPS,'GetProperties')`	OP is a feOutputSet object. It must have its ID and SetID properties set appropriately. OUT is a structure containing the properties.
`OutputSet`	`RC=femap_invoke (OPS,'PutProperties',SPROP)`	OP is a feOutputSet object. It must have its ID and SetID properties set appropriately. SPROP is a structure containing the properties.

If the method is not found in the supported list, it will attempt to invoke the method by calling MATLAB's INVOKE function. Thus for methods that have only input arguments in the argument list, INVOKE and FEMAP_INVOKE are interchangeable.

RC is the return call. If an error occurred, RC will be a structure with the fields `.errnum` and `.errstr`. If the method was successful, it will contain the return value from the method.

Please note thta FEMAP_INVOKE by no means supports the exhaustive list of FEMAP API method calls. If the function you are trying to use has output arguments, and FEMAP_INVOKE returns an error and you are calling it correctly, chances are that this method is not yet supported. If so, please file a TIER report.

Example

```
>> happ = actxserver('femap.model');
>> node = invoke(happ,'feNode');
>> node = femap_invoke(happ,'feNode');          % This line is equivalent to the one above it
>> setid = int32(0);
>> [rc,nnod,ids,xyz,lay,col,typ,dcs,ocs,pbc] = femap_invoke(node,'GetAllArray',setid);
```

See Also

writefemap

---

readodb (+FEA)

Purpose

Import data from Abaqus ODB files.

Syntax

```
f=readodb('filename')
f=readodb
[f,fname,odbdir,sel]=readodb('*.odb','dironly','range',RANGE_
CELL,'silent','noprogbar','headeronly')
```

Description

READODB reads data from an Abaqus ODB. History data will be imported as `imat_fn` and Field data will be imported as `imat_shp` or a `result` object, depending on the data type. Several optional input arguments are supported.

`'silent'`	Suppress all output messages
`'noprogbar'`	Don't display progress bar at all
`'dironly'`	Get the ODB directory and return it in a structure. This is the same structure that gets returned in the 3rd output argument ODBDIR. This structure is detailed below.
`'headeronly'`	Only read headers, no data.

`'range'`	Specifies what data to read. The following argument must be an Nx2 cell array, described in more detail below.
`'asresult'`	Force all field output to be an imat_result object.
`'mergeinst'`	Merge all instances into a single result. This only applies to imat_result output.

The range specifier must be an Nx2 cell array, where N is the number of different data types to read. The first column must contain a string specifying the data source. The data source must either be `'xydata'` or the step name (key).

If the data source is `'xydata'`, the second column argument must be a numeric vector containing the indices of the xyData to read. An empty matrix tells READODB to read all of the xyData.

If the data source is a step, the second column argument must be an nx2 cell array. The first element in this cell array must be the string `'history'` or `'field'`. Note that you can only read one `'history'` and one `'field'` type per step per call to READODB.

Type	Description
`'history'`	The second element in this cell array must be a numeric vector containing the indices of the historyObjects to read. An empty matrix tells READODB to read all of the historyObjects associated with this step. The output will be returned as an `imat_fn`.
`'field'`	The second element in this cell array must be a numeric vector containing the indices of the frames to read. An empty matrix tells READODB to read all of the frames associated with this step. The optional second element in this cell array contains the data type(s) to read. If it is not provided, READODB will read all of the supported data types available in each frame. If it is specified, it must be a string or cell array of strings containing the data types to be read. To see if a datatype in your ODB is currently supported, open the ODB with READODB and click on the Select Subset button and view the Data Types list. The datatypes listed here are supported.

Translational and rotational data types will be combined in a single output. Nodal displacement output will be returned as an `imat_shp`. All other field output will be returned as a `result` object. If there are multiple instances, the coefficients for each instance will be stored in a separate result. In this case the output will be MxN, where M is the number of frames selected for that data type, and N is the number of instances. Different data types will be returned as separate cells in the output. The instance name will be stored in IDLine2.

F is the resulting data. If multiple output types are requested, F will be a cell array containing the various outputs. FNAME is an optional output string containing the filename read.

ODBDIR is an optional output structure containing the ODB directory contents. The structure fields are

`.filename`	Name of ODB (string)
`.nxydata`	Number of xyData objects in the ODB (scalar)
`.xydatakeys`	Names of the xyData objects (cell array)
`.nstepdata`	Nx2 array, where N is the number of steps containing supported data. The first column contains the number of historyObjects in that step, and the second contains the number of fieldObjects in the step.
`.stepkeys`	Names of the steps (cell array)

`.desc`	Step descriptions (cell array)
`.domain`	Step domain (cell array)
`.procedure`	Step procedure (cell array)

SEL is a cell array of the same format as RANGE_CELL containing the range specifiers for the data that was selected to read.

Examples

```
>> data=readodb('range',{'xydata',1:10; 'Step-2',{'history',[1 3 10]}});
>> data=readodb('range',{'Step-1',{'field',[1 3 5]}});
>> data=readodb('range',{'Step-1',{'field',1:10,'U'}});
>> data=readodb('range',{'Step-1',{'field',[1 3 10]};'Step-2',{'field',1:20,{'U','UR'}}});
>>
```

See Also

readnas, readunv, readadf, setunits, getunits

IMAT Utility Functions

setunits	Select the desired unit system
getunits	Return information on the current unit system
ideas_colormap	Get MATLAB RGB color code from I-DEAS colors
ideas_datenum	Get MATLAB datenum for an I-DEAS date string
ideas_datestr	Get I-DEAS date string for a MATLAB datenum
imat_dir2num	Convert coordinate direction strings to numeric direction codes
imat_num2dir	Convert numeric direction codes to coordinate direction strings
imat_getfile	Enhanced getfile for IMAT toolbox.
imat_putfile	Enhanced putfile for IMAT toolbox.
ideas_progress	Implements progress window for IMAT
imat_ver	Get the current IMAT version

license_commute	Manage commuter (checkout) licenses
expandDMI **(+FEA)**	Converts DMI matrix read with readnas to full storage
mat2subst	Convert matrices to substructure format
parseop2table_raw2nas	Parse readnas raw tables into Nastran-formatted output
count_freqs_in_band	Count the number of frequencies in 1/N octave bands.
get_octave_bands	Get 1/N octave bands.

setunits

Purpose

Set unit system for IMAT.

Syntax

```
setunits(ustr)
setunits
setunits('US', [Lfact Ffact Tfact Toffset] )
setunits('ing')
setunits(ustr,'silent')
```

Description

SETUNITS defines the unit system you want to work in.

`setunits(ustr)` selects one of eight predefined unit systems. `ustr` should be a two-character string taken from one of the following:

unum	*ustr*	*Name*	*Length*	*Time*	*Mass*	*Force*	*Temp*
1	SI	Metric_Abs_(SI)	meter	sec	kilogram(kgm)	Newton (N)	deg_c:K
2	BG	British_Grav	foot	sec	lbf-sec^2/ft	pound (lbf)	deg_F:R
3	MG	Metric_Grav	meter	sec	kgf-sec^2/m	kilogram (kgf)	deg_C:K
4	BA	British_Abs	foot	sec	pound(lbm)	poundal (pdl)	deg_F:R
5	MM	Modified_SI	mm	sec	kilogram(kgm)	milli-Newton	deg_C:K
6	CM	Modified_SI	cm	sec	kilogram(kgm)	centi-Newton	deg_C:K
7	IN	British_Grav_(mod)	inch	sec	lbf-sec^2/in	pound (lbf)	deg_F:R

8	GM	Metric_Grav_(mod)	mm	sec	kgf-sec^2/mm	kilogram (kgf)	deg_C:K
9	US	User_defined	-	sec	-	-	-
10	MN	Milli Meter (Newton)	mm	sec	tonne (t)	Newton (N)	deg_C:K

If you append a `'g'` to the end of the units string `ustr`, IMAT will treat acceleration quantities as G's and not engineering units. For example, `setunits('sig')` will treat acceleration units as G's and all other quantities with metric engineering units.

`setunits('US',[ Lfact Ffact Tfact Toffset Afact])` sets units to a nonstandard (user-defined) unit system for which *`Lfact`* length units equal one meter, *`Ffact`* force units equal one Newton, and *`Tfact`* temperature units equal one degree Kelvin. *`Toffset`* is the offset of absolute temperature. *`Afact`* is the conversion factor between one acceleration unit and one meter per second squared.

`setunits` with no arguments will bring up a graphical prompt for the desired unit system. The optional input string `'silent'` will suppress output.

The unit system affects all input and output operations to ADF and universal files. On input from an ADF or universal file, data is converted into the selected unit system. On output to an ADF, data is converted back to SI units as required by I-deas. On output to a universal file, a units dataset (164) is written at the beginning of the file to indicate the units of the subsequent datasets.

Note: unlike I-deas, changing the unit system does not automatically convert units for existing functions. You should call SETUNITS before any import/output operations, and maintain consistency throughout a session. If you never call SETUNITS, then SI units will be assumed. If GETUNITS is called before SETUNITS, then the user will prompted for a unit system.

Examples

```
>> setunits('in')
Units set to IN

>>
```

See Also

getunits

---

getunits

Purpose

Return information on the current unit system.

Syntax

```
[ustr,ufact,unum]=getunits('in')
[ustr,ufact,unum]=getunits('silent')
[ustr,ufact,unum]=getunits('in','silent')
[ustr,ufact,unum,ulab]=getunits
```

Description

GETUNITS has two uses: to find out the current unit system, and to obtain information on any predefined unit system. The input argument is a two- or three-character unit string. If the input argument is omitted, the current working units are returned.

The meaning of the output arguments are as follows:

```
ustr  - a vector of conversion factors :
ufact - the two-character unit system name
     ufact(1) = 1 meter/1 model length unit
     ufact(2) = 1 Newton/1 model force unit
     ufact(3) = 1 deg K/1 model temperature unit (relative temperature)
     ufact(4) = temperature offset from absolute zero
     ufact(5) = 1 meter/sec^2/1 model acceleration unit
unum  - the numeric value of the unit system
ulab  - the units label strings
     ulab{1,:}  - length units label strings
     ulab{2,:}  - force units label strings
     ulab{3,:}  - mass units label strings
     ulab{4,:}  - temperature units label strings
     ulab{5,:}  - time units label strings
     ulab{6,:}  - acceleration units label strings
```

All of the output arguments are optional. If no output arguments are requested, then GETUNITS prints a summary of the specified unit system.

The following unit systems are recognized:

unum	*ustr*	*Name*	*Length*	*Time*	*Mass*	*Force*	*Temp*
1	SI	Metric_Abs_(SI)	meter	sec	kilogram(kgm)	Newton (N)	deg_c:K
2	BG	British_Grav	foot	sec	lbf-sec^2/ft	pound (lbf)	deg_F:R
3	MG	Metric_Grav	meter	sec	kgf-sec^2/m	kilogram (kgf)	deg_C:K
4	BA	British_Abs	foot	sec	pound(lbm)	poundal (pdl)	deg_F:R
5	MM	Modified_SI	mm	sec	kilogram(kgm)	milli-Newton	deg_C:K
6	CM	Modified_SI	cm	sec	kilogram(kgm)	centi-Newton	deg_C:K
7	IN	British_Grav_(mod)	inch	sec	lbf-sec^2/in	pound (lbf)	deg_F:R
8	GM	Metric_Grav_(mod)	mm	sec	kgf-sec^2/mm	kilogram (kgf)	deg_C:K
9	US	User_defined	-	sec	-	-	-
10	MN	Milli Meter (Newton)	mm	sec	tonne (t)	Newton (N)	deg_C:K

If the 3rd character of USTR is `'g'`, this indicates that IMAT is treating acceleration units as G's and not engineering units.

The unit system affects all input and output operations to ADF and universal files. On input from an ADF or universal file, data is converted into the selected unit system. On output to an ADF, data is converted back to SI units as required by I-deas. On output to a universal file, a units dataset (164) is written at the beginning of the file to indicate the units of the subsequent datasets.

Passing in the optional string `'silent'` suppresses output.

Note: unlike I-deas, changing the unit system does not automatically convert units for existing functions. You should call SETUNITS before any import/output operations, and maintain consistency throughout a session. If you never call SETUNITS, then SI units will be assumed. If GETUNITS is called before SETUNITS, then the user will prompted for a unit system.

Examples

```
>> getunits('in');
Summary of IN unit system (current default)
Length
.....     39.3701 model units = 1 meter
          1       model unit  =       0.0254 meters
Force
.....    0.224809 model units = 1 Newton
          1       model unit  =      4.44822 Newtons
Mass
.....  0.00571015 model units = 1 kilogram
          1       model unit  =      175.127 kilograms
Temperature
.....         1.8 model units = 1 degree Kelvin
          1       model unit  =     0.555556 degrees Kelvin
Temperature scale
.....         1.8 model units -      459.67 = 1 degree Kelvin
          1       model unit  -      459.67 =     0.555556 degree Kelvin
Acceleration
.....     39.3701 model units = 1 m/s^2
          1       m/s^2       =       0.0254 model units

>> getunits('ing');
Summary of IN unit system (current default)
Length
.....     39.3701 model units = 1 meter
          1       model unit  =       0.0254 meters
Force
.....    0.224809 model units = 1 Newton
          1       model unit  =      4.44822 Newtons
Mass
.....  0.00571015 model units = 1 kilogram
          1       model unit  =      175.127 kilograms
Temperature
.....         1.8 model units = 1 degree Kelvin
          1       model unit  =     0.555556 degrees Kelvin
Temperature scale
.....         1.8 model units -      459.67 = 1 degree Kelvin
          1       model unit  -      459.67 =     0.555556 degree Kelvin
Acceleration
.....     386.089 model units = 1 G
          1       G           =   0.00259008 model units
>>
```

See Also

setunits

ideas_colormap

Purpose

Get MATLAB RGB color designators from I-DEAS color codes.

Syntax

```
n=ideas_colormap(m)
```

Description

IDEAS_COLORMAP is a utility function that converts I-DEAS color code into MATLAB RGB codes. The I-DEAS color code `m` can be a vector. The number of rows in `n` will equal the number of elements in `m`. Any user-defined colors (>15) will return as black.

Examples

```
>> n=ideas_colormap([3 7 11])
n =
         0         0.6600    1.0000
         0         1.0000         0
         1.0000         0         0
>>
```

ideas_datenum

Purpose

Get MATLAB datenum for an I-DEAS date string.

Syntax

```
n=ideas_datenum(s)
```

Description

IDEAS_DATENUM is a utility function that converts an I-DEAS date/time string `s` into a numeric date/time `n`.

I-DEAS date/time strings are used for creation and modification date strings in `imat_fn` objects, and have the format `'DD-MMM-YY HH:MM:SS'`. IDEAS_DATENUM also handles the IRIG Time format, which is `'DDD:HH:MM:SS.SSSSSS'`, where `DDD` is the number of days from the beginning of the year.

MATLAB numeric date/time values are serial days where 1 is 1-Jan-0000.

Examples

```
>> f=imat_fn(1);
>> n=ideas_datenum(f.createdate)
n =
   7.2974e+05
>> datevec(n)
ans =
        1997          12          16           9          24          41
>>
```

See Also

ideas_datestr

---

ideas_datestr

Purpose

Get I-DEAS date string for a MATLAB datenum.

Syntax

```
s=ideas_datestr(n)
s=ideas_datestr(n,'irig')
```

Description

IDEAS_DATESTR is a utility function that converts a numeric date/time `n` into an I-DEAS date/time string `s`.

I-DEAS date/time strings are used for creation and modification date strings in `imat_fn` objects, and have the format `'DD-MMM-YY HH:MM:SS'`.

Passing in the optional input string `'irig'` returns the date string in IRIG Time format.

MATLAB numeric date/time values are serial days where 1 is 1-Jan-0000.

`s=ideas_datestr(now)` returns the current date/time in I-DEAS date/time format. `s=ideas_datestr('irig')` returns the current date/time in IRIG Time format.

Examples

```
>> n=datenum('12/10/97')
n =
      729734
>> f=imat_fn(1);
>> f.createdate=ideas_datestr(n);
>> f.createdate
ans =
      '10-Dec-97   00:00:00'
>> f.irigtime=ideas_datestr(n,'irig');
>> f.irigtime
ans =
      '344:00:00:00.000000'

>>
```

See Also

ideas_datenum

---

imat_dir2num

Purpose

Convert coordinate direction string(s) to number(s).

Syntax

```
x=imat_dir2num(dir)
```

Description

IMAT_DIR2NUM is a utility function that converts direction strings to numeric direction codes.

The direction string `dir` can be either a single string (in which case IMAT_DIR2NUM will return a scalar code); or a cell array of strings (in which case IMAT_DIR2NUM will return a numeric array); or an nxm character array, with m<=4 (in which case IMAT_DIR2NUM will return an nx1 numeric array).

The numeric code returned by IMAT_DIR2NUM for each direction string is given in the following table.

Numeric code	Direction	Numeric code	Direction
1	X+	-1	X-
2	Y+	-2	Y-
3	Z+	-3	Z-
4	RX+	-4	RX-

5	RY+	-5	RY-
6	RZ+	-6	RZ-
0	any other		

Examples

```
>> imat_dir2num('RX-')
ans =
     -4

>> t=imat_ctrace('5y', '3rx', '3z', '4x-', '3z', '5y+')
t =
     '5Y+'
    '3RX+'
    '3Z+'
    '4X-'
    '3Z+'
    '5Y+'

>> d=dir(t)
d =
    'Y+'
    'RX+'
    'Z+'
    'X-'
    'Z+'
    'Y+'

>> imat_dir2num(d)
ans =
     2
     4
     3
    -1
     3
     2

>>
```

See Also

imat_num2dir

---

imat_num2dir

Purpose

Convert direction number(s) to coordinate direction string(s).

Syntax

```
d=imat_num2dir(x)
```

Description

IMAT_NUM2DIR is a utility function that converts numeric direction codes to coordinate direction strings.

The numeric argument x can be either a scalar integer from -6 to 6 (in which case IMAT_NUM2DIR will return a string), or an nx1 array of such values (in which case IDEAS_NUM2DIR will return an nx4 array of type char).

The direction string returned by IMAT_NUM2DIR for each numeric code is given in the following table.

Numeric code	Direction	Numeric code	Direction
1	X+	-1	X-
2	Y+	-2	Y-
3	Z+	-3	Z-
4	RX+	-4	RX-
5	RY+	-5	RY-
6	RZ+	-6	RZ-

Any other value returns a null string.

Examples

```
>> x=(-7:7)'
x =
    -7
    -6
    -5
    -4
    -3
    -2
    -1
     0
     1
     2
     3
     4
     5
     6
     7

>> imat_num2dir(x)
ans =

    RZ-
    RY-
    RX-
    Z-
    Y-
    X-

    X+
    Y+
    Z+
    RX+
    RY+
    RZ+

>>
```

See Also

imat_dir2num

---

imat_getfile

Purpose

Enhanced uigetfile for IMAT toolbox.

Syntax

```
filename = imat_getfile('initfile','dialogtitle')
[filename,filterindex] = imat_getfile('initfile','dialogtitle'[,...])
```

Description

IMAT_GETFILE is a wrapper around UIGETFILE that returns a fully qualified filename.

INITFILE is a string or cell array of strings that contains the file mask. If not supplied, `'*.*'` will be used.

DIALOGTITLE is a string containing the title for the file dialog.

Any additional input arguments will be passed directly to UIGETFILE.

FILENAME is the fully qualified filename selected. If the user cancels, FILENAME is -1. If the `Multiselect` option is turned on, FILENAME is a cell array of strings containing the fully qualified filenames selected.

FILTERINDEX is the index of the filter selected in the dialog box. Please see the help for UIPUTFILE for more details.

Error checking is minimal.

See Also

imat_putfile

---

imat_putfile

Purpose

Enhanced uiputfile for IMAT toolbox.

Syntax

```
filename = imat_putfile('initfile','dialogtitle')
[filename,filterindex] = imat_putfile('initfile','dialogtitle')
```

Description

IMAT_PUTFILE is a wrapper around UIPUTFILE that returns a fully qualified filename.

INITFILE is a string or cell array of strings that contains the file mask. If not supplied, `'*.*'` will be used.

DIALOGTITLE is a string containing the title for the file dialog.

FILENAME is the fully qualified filename selected. If the user cancels, FILENAME is -1.

FILTERINDEX is the index of the filter selected in the dialog box. Please see the help for UIPUTFILE for more details.

Error checking is minimal.

See Also

imat_getfile

---

imat_progress

Purpose

Implements progress window for IMAT.

Syntax

```
imat_progress(0,'Progress title')
imat_progress(0,'Progress title','modal')
imat_progress(1,0.5)
imat_progress(2,'Item progress')
imat_progress(-1)
```

Description

IMAT_PROGRESS is a utility function that displays and updates a progress bar. Only one progress bar may be used at a time.

The first argument to IMAT_PROGRESS is a flag specifying what action to take. The table below describes the additional arguments possible for each action.

First Argument	**Additional Argument(s)**
`'register'`	H_PATCH,H_TEXT[,H_APPSTOP,S_APPSTOP] This allows you to pre-register handles to a patch (H_PATCH) and text object (H_TEXT). If these are registered, when you use the 0 action, IMAT_PROGRESS will use the supplied handles. H_APPSTOP is an optional handle for where to store the appdata for when the user stops. If H_APPSTOP and S_APPSTOP are not provided (or 'register' is not used), H_APPSTOP defaults to H_PATCH and S_APPSTOP defaults to `'IMATProgressStop'`.
`0`	TITLE[,'modal'][,'stopbutton'] This initializes the figure window with given title. If a patch and handle object was not pre-registered, IMAT_PROGRESS will create a new figure. Supplying `'modal'` will make the proogress figure modal. `'stopbutton'` creates a Stop button on the progress bar. If the user clicks on this button, the appdata S_APPSTOP on the handle H_APPSTOP will be set (see `'register'` option for more details). If this appdata is set to true when IMAT_PROGRESS is called with the 1 or 2 action, it will throw an error.
`1`	Sets the progress percentage to the numeric fraction specified by X. X must be between 0 and 1.
`2`	STR is a string specifying the string to place just above the progress bar.

`3`	RGBVEC is a 1x3 vector that sets the progress bar color (pass in [] to reset the color to the default).
`-1`	-none-

Any other value returns a null string.

Examples

```
>> imat_progress(0,'File read','modal');
>> imat_progress(2,'Reading file...');
>> imat_progress(1,0.25);
>> imat_progress(1,0.5);
>> imat_progress(1,0.75);
>> imat_progress(1,1);
>> imat_progress(-1);
```

---

imat_ver

Purpose

Return information on the current version of IMAT.

Syntax

```
imat_ver
v=imat_ver
[v,n]=imat_ver
[v,n,d]=imat_ver
v=imat_ver('matlab')
```

Description

IMAT_VER is a utility function that returns the current version of IMAT. Up to three output arguments can be supplied. If none are supplied, a string containing the version information is returned.

The meaning of the output arguments are as follows:

`v` - character array containing the IMAT version information string

`n` - character array containing the version number

`d` - character array containing the date IMAT was compiled

If you pass in the string 'matlab', IMAT_VER returns a number containing the current MATLABversion multiplied by 10000. The minor version number will be multiplied by 100, and the tertiary version number (if any) will be added as-is. For example, MATLAB 8.1 will be returned as 81000. MATLAB 7.9.1 will be returned as 70901.

Examples

```
>> imat_ver
Interface between MATLAB, Analysis, and Test Version 3.0.0 10/09/2009

>> [v,n,d]=imat_ver
v =
Interface between MATLAB, Analysis, and Test Version 3.0.0 10/09/2009
n =
3.0.0
d =
10/09/2009
```

---

license_commute

Purpose

Manage commuter (checkout) licenses.

Syntax

```
license_commute
imat_progress(0,'Progress title','modal')
imat_progress(1,0.5)
imat_progress(2,'Item progress')
imat_progress(-1)
```

Description

LICENSE_COMMUTE is a management interface for commuting licenses. Commuting means checking out a license from a server to the local machine, so you can take your machine off the network and still use the license.

LICENSE_COMMUTE has several calling sequences, described below.

Calling sequence	Description
`license_commute`	Display a graphical interface that allows you to manage your commuted licenses.
`data = license_commute ('commlist')`	Return a structure containing information about the currently commuted licenses available on the client. DATA is the license structure.
`data = license_commute ('liclist')`	Return a structure containing information about all of the licenses available on the available server(s). DATA is the license structure. This function can be used to determine which licenses are available for commuter checkout.
`license_commute('checkout',LICDATA)`	Check out a commuter license to the client machine. LICDATA is a 1x3 or 1x4 cell array containing the credentials for the license to commute. The first cell is a string containing the Feature Name. The second is a string containing the Feature Version. The 3rd is a numeric scalar containing the number of licenses (or tokens) to commute. The 4th is an

	optional numeric scalar specifying how many days to check out the license. 0 means the maximum period allowed by the license.
`license_commute ('checkin',LICDATA)`	Check in a commuter license. LICDATA is a 1x2 cell array containing the credentials for the license to check in. The first cell is a string containing the Feature Name. The second is a string containing the Feature Version.
`-1`	-none-

Examples

```
>> data = license_commute('liclist');
>> license_commute('checkout',{'ATA_TOKENS','1.0',4,0})
>> license_commute('checkin',{'ATA_TOKENS','1.0'})
```

---

expandDMI

Purpose

Converts DMI matrix read with `readnas` to full storage.

Syntax

```
mat=expandDMI(dmi,fullstorage)
mat=expandDMI(dmi,true)
```

Description

EXPANDDMI is a utility function that expands a DMI matrix created using readnas to a full double. The input argument is a single structure containing the matrix output from readnmat. Note that this function will not accept an array of structs. The struct is documented below. If the structure does not contain a DMI matrix, the return value is empty. If the structure does contain a DMI matrix, the return value is a fully populated double matrix.

Two input arguments are required. DMI is a structure with the fields shown in the table below. FULLSTORAGE is a logical designating whether output should be in full storage or sparse.

Field	**Description**
`matrix`	matrices in sparse format
`name`	matrix names
`dof`	row and column DOF, respectively, stored as a 1x2 cell array of `imat_ctrace`
`phase`	logical flag (0/1) has value of 1 if the matrix is complex-valued and values are given in magnitude/phase form
`symmetric`	logical flag (0/1) has value of 1 if the matrix is stored symmetrically (only half the matrix is stored)
`matpool`	logical flag (0/1) has value of 1 if the matrix was read in from Nastran DMIG and has value 0 if matrix was read in from Nastran DMI

The output MAT is a full or sparse matrix containing data associated with DMI.

See Also

readnas

---

mat2subst

Purpose

Convert matrices to substructure format.

Syntax

```
sub=mat2subst(mat)
sub=mat2subst(mat,'silent')
```

Description

MAT2SUBST takes the matrix structure supplied in MAT and returns a substructure matrix in OUT.

MAT is an array of structures containing matrices in the format returned by READNAS and a few other IMAT functions. At minimum these must have the fields `'name'`, `'matrix'`, and `'dof'`. MAT2SUBST expects the mass matrix to be called `'MXX'`, `'MXXGEXT'`, or `'MAAX'`, the stiffness matrix to be called `'KXX'`, `'KXXGEXT'`, or `'KAAX'`, and the back expansion (constraint) matrix to be called `'GO'`, `'GOGADGX'`, or `'MUG1'`. It will copy the matrices it finds into the output structure. The DOF defined for the mass, stiffness, and columns of the constraint matrix will be placed in the `'aset'` field, and the row DOF for the constraint matrix is placed in the `'oset'` field.

If the mass or stiffness matrix is upper or lower triangular, MAT2SUBST will fill in the missing portion of the matrix.

MAT2SUBST will also make sure the mass and stiffness matrices are the same size and use the union of the two sets of DOF.

The optional input string `'silent'` suppresses output.

OUT is a structure containing the matrices found in the input structure. This structure is the format used by WRITESUBST.

Examples

```
>> f=readnas
Units set to IN
OP2 Name: C:\plate_modes_guyan.op2
Units   : <1> SI
...Opened OP2 file 'C:\plate_modes_guyan.op2'
...Reading OP2 directory block names...18 data blocks from 70 records
...Reading GEOM1 and GEOM2 data from OP2 file...
...Processing GEOM1 data
...Processing GEOM2 data
...Reading matrix 'KXX' from OP2 file...
...Reading matrix 'MXX' from OP2 file...
...Reading matrix 'GO' from OP2 file...
...Finished processing input file
...Done!

f =
    matrix: [1x3 struct]

>> sub=mat2subst(f.matrix)
...Filling in mass matrix
...Filling in stiffness matrix
...Filling in constraint matrix

sub =
          aset: [35x1 imat_ctrace]
          oset: [10x1 imat_ctrace]
          sset: [0x0 imat_ctrace]
          mass: [35x35 double]
     stiffness: [35x35 double]
       viscous: []
    hysteretic: []
    constraint: [10x35 double]

>>
```

See Also

readnas, writesubst

---

parseop2table_raw2nas

Purpose

Parse readnas raw tables into Nastran-formatted output.

Syntax

```
data=parseop2table_raw2nas(table)
data=parseop2table_raw2nas(table,endianflag,'silent')
```

Description

PARSEOP2TABLE_RAW2NAS parses "raw" output from READNAS (from the 'asraw' input option) into structures containing native Nastran-formatted tables. Please see the DDL descriptions for each datablock and the Item Codes chapter of the Nastran Quick Reference Guide for information on interpreting the Nastran output.

Please edit this function and look in the datablock_lookup() function for a list of supported datablocks. If a datablock category is supported, but the specific datablock name in a supporting category is not listed, you can add it to the list in this function to add support.

Currently supported table datablock categories are:

`OUG`	Nodal displacement/velocity/acceleration
`OGF`	Grid point forces
`OEE`	Element energy (strain, kinetic, loss)
`OEF`	Element forces
`OES`	Element stresses or strains
`OPG`	Applied loads
`OQG`	Single or multipoint constraint forces

TABLE is a structure whose fields are the names of the datablocks imported by READNAS and whose contents are the raw data structures.

ENDIANFLAG is an optional input specifying whether the endian should be flipped (true indicates that endians are flipped). This value is an optional output of READNAS, and is generally only true when the OP2 file was created on Unix and you are running Matlab on Windows or Linux, or vice versa.

The optional input string `'silent'` suppresses output.

TABLE is a structure containing the same fields supplied in the input. Datablocks that were translated replace the original content of the fields.

Result datablocks consist of a 146 word header record followed by a data record. This function translates each pair of header and data records into a single output structure. Thus if the input datablock contains 100 records, the output will contain a 50x1 structure. The fieldnames for the output will vary for each result type, since the header record contains different result type-specific information. However, most of the fieldnames will be the same. Some important fields to note are:

`.id`	NIDx1 vector of entity IDs* 10. Though not specified in the Nastran documentation, this corresponds to Item Code 1. Divide this vector by 10 to get the node or element ID as appropriate. The last digit of the ID specifies the ID type (node, scalar point, element, etc.).
`.item`	1xNITEM vector of Item Codes available. These are usually 2-N, but item codes corresponding to character val-

	ues are removed.
`.data`	NIDxNITEM matrix of data values. NID is the number of IDs, and ITEM is the number of Item Codes (2-N). The meaning of each item code is defined for each entity type for each result type in the Nastran Quick Reference Guide. Columns corresponding to character Item Codes are stored as Nan, and the corresponding character values are stored in the `.datachar` field.
`.datachar`	NIDxNCOL cell matrix of strings corresponding to the character strings in the Item Codes. To save memory, NCOL corresponds to the highest character item code for that data type, rather than NITEM.

Examples

```
>> f = readnas('asraw');

>> data = parseop2table_raw2nas(f.table);

>> [f,~,info] = readnas('asraw');

>> data = parseop2table_raw2nas(f.table, info.endianswap);
```

See Also

readnas

---

count_freqs_in_band

Purpose

Count the number of frequencies in 1/N octave bands.

Syntax

```
count=count_freqs_in_band(fmin,fmax)
[count,band]=get_octave_bands(fmin,fmax,n)
[count,band]=get_octave_bands(fmin,fmax,'silent','ansi')
[count,band]=get_octave_bands(fmin,fmax,n,'silent','ansi')
```

Description

COUNT_FREQS_IN_BAND counts the number of frequencies in 1/N octave bands. It handles both formula-calculated frequency bands and ANSI bands. Please see the help for GET_OCTAVE_BANDS for more details.

FREQS is a vector of frequencies to count. Only frequency 1 Hz and up are counted.

The optional input argument `'silent'` suppresses output.

By default COUNT_FREQS_IN_BAND does not use the "preferred" frequencies. Instead, it calculates the frequencies using standard formulas. To use the ANSI/ISO standard frequencies defined in ISO R 266 and ANSI S1.6-1984, pass in the string `'ansi'`.

BANDS is a structure containing the frequency band information. See GET_OCTAVE_BANDS for details of its contents.

COUNT is a vector the length of the number of bins containing a count of the frequencies in FREQS that fall into each of the bins.

See also

get_octave_bands

---

get_octave_bands

Purpose

Get 1/N octave bands.

Syntax

```
band=get_octave_bands(fmin,fmax)
band=get_octave_bands(fmin,fmax,n)
band=get_octave_bands(fmin,fmax,'silent','ansi')
band=get_octave_bands(fmin,fmax,n,'silent','ansi')
```

Description

GET_OCTAVE_BANDS returns the 1/N octave bands defined from the minimum and maximum frequencies supplied. If N is not specified, it will default to 1.

The optional input argument `'silent'` suppresses output.

By default OCTAVEN does not use the "preferred" frequencies. Instead, the following formulas are used:

`center = 1000*2^(i/N)`
where `i = 0` for `f0 = 1000 Hz`

```
lower = center / 2^(1/2/N)
upper = center * 2^(1/2/N)
```

To use the ANSI/ISO standard frequencies defined in ISO R 266 and ANSI S1.6-1984, pass in the string 'ansi'. If octave or third octave reduction is requested, the standard frequencies will be used. Otherwise OCTAVEN will issue a warning and use the above formulas instead.

The output structure BAND contains the 1/N octave band data. Fields are

`.center`	Vector of center frequencies for each band
`.lower`	Vector of starting frequencies for each band

`.upper`	Vector of ending frequencies for each band
`.ansi`	Logical specifying whether ANSI frequency bands were used

Data Attribute Reference

Function Data Attributes

Function Data Attributes (sorted by subject)

IMAT Home Page
Alphabetical listing

Function identifiers

- FunctionType
- ResponseCoord
- ResponseNode
- ResponseDir
- ReferenceCoord
- ReferenceNode
- ReferenceDir
- IDLine1
- IDLine2
- IDLine3
- IDLine4
- CreateDate
- ModifyDate
- Version
- SetRecord

Abscissa information

- Abscissa
- AbscissaSpacing
- NumberElements
- AbscissaMin
- AbscissaInc
- AbscissaDataType
- AbscissaTypeQual
- AbscissaExpLength
- AbscissaExpForce
- AbscissaExpTemp
- AbscissaExpTime
- AbscissaAxisLab
- AbscissaUnitsLab
- AbscissaOffset
- IRIGTime

Ordinate information

- Ordinate
- OrdinateType
- NumberElements
- OrdNumDataType
- OrdNumTypeQual
- OrdNumExpLength
- OrdNumExpForce
- OrdNumExpTemp
- OrdNumExpTime
- OrdDenDataType
- OrdDenTypeQual
- OrdDenExpLength
- OrdDenExpForce
- OrdDenExpTemp
- OrdDenExpTime
- MaxOrdValReal
- MaxOrdValImag
- MinOrdValReal
- MinOrdValImag
- OrdinateAxisLab
- OrdinateUnitsLab
- OrdOffsetReal
- OrdOffsetImag
- OrdScaleReal
- OrdScaleImag
- ExpDampingFact

Z Axis information

- ZGeneralValue
- ZRPMValue
- ZTimeValue
- ZOrderValue
- ZAxisDataType
- ZAxisTypeQual
- ZAxisExpLength
- ZAxisExpForce
- ZAxisExpTemp
- ZAxisExpTime

Measurement information

- MeasurementRun
- SamplingType
- WeightingType
- WindowType
- AmplitudeUnits
- Normalization
- OctaveAvgType
- OctaveFormat
- OctaveOverallRMS
- OctaveWeightedRMS
- PulsesPerRev

Other miscellaneous information

- UserValue1
- UserValue2
- UserValue3
- UserValue4
- LoadCase
- CoordSys
- OwnerName
- ResponseEntity
- ReferenceEntity

Function Data Attributes (sorted alphabetically)

IMAT Home Page
Subject listing

- Abscissa
- AbscissaAxisLab
- AbscissaDataType
- AbscissaExpForce
- AbscissaExpLength
- AbscissaExpTemp
- AbscissaExpTime
- AbscissaInc
- AbscissaMin
- AbscissaOffset
- AbscissaSpacing
- AbscissaTypeQual
- AbscissaUnitsLab
- AmplitudeUnits
- CoordSys
- CreateDate
- ExpDampingFact
- FunctionType
- IDLine1
- IDLine2
- IDLine3

IDLine4
LoadCase
MaxOrdValImag
MaxOrdValReal
MeasurementRun
MinOrdValImag
MinOrdValReal
ModifyDate
Normalization
NumberElements
OctaveAvgType
OctaveFormat
OctaveOverallRMS
OctaveWeightedRMS
OrdDenDataType
OrdDenExpForce
OrdDenExpLength
OrdDenExpTemp
OrdDenExpTime
OrdDenTypeQual
OrdNumDataType
OrdNumExpForce
OrdNumExpLength
OrdNumExpTemp
OrdNumExpTime
OrdNumTypeQual
OrdOffsetImag
OrdOffsetReal
OrdScaleImag
OrdScaleReal
Ordinate
OrdinateAxisLab
OrdinateType
OrdinateUnitsLab
OwnerName
PulsesPerRev
ReferenceCoord
ReferenceDir
ReferenceEntity
ReferenceNode
ResponseCoord
ResponseDir
ResponseEntity
ResponseNode
SamplingType
SetRecord
UserValue1
UserValue2
UserValue3
UserValue4
Version
WeightingType
WindowType

- ZAxisDataType
- ZAxisExpForce
- ZAxisExpLength
- ZAxisExpTemp
- ZAxisExpTime
- ZAxisTypeQual
- ZGeneralValue
- ZOrderValue
- ZRPMValue
- ZTimeValue

Abscissa

The abscissa values are the *x* values in the (*x*,*y*) data points of the function. If the function is unevenly spaced (see AbscissaSpacing), then the abscissa values are stored as a column vector in the function. The abscissa values must be real and monotonic increasing. If the function is evenly spaced, the abscissa values are not actually stored, but are determined by the AbscissaMin and AbscissaInc. The Abscissa data type is stored in AbscissaDataType.

AbscissaAxisLab

This is a character string of maximum length 20 that can be used to override the default abscissa axis label. It will appear in place of the default axis label when the function is graphed in I-DEAS.

AbscissaDataType

Defines the data type for the abscissa. It is always assumed to be a real quantity. The available data types and their corresponding numeric identifier are shown in the table below. The attribute specified determines how the data is converted when it is stored and retrieved by I-DEAS.

Numeric Identifier	Data Type
0	`'Unknown'`
1	`'General'`
2	`'Stress'`
3	`'Strain'`
5	`'Temperature'`
6	`'Heat Flux'`
8	`'Displacement'`
9	`'Force'`
11	`'Velocity'`
12	`'Acceleration'`
13	`'Excitation Force'`
15	`'Pressure'`

16	'Mass'
17	'Time'
18	'Frequency'
19	'RPM'
20	'Order'
21	'Sound Pressure'
22	'Sound Intensity'
23	'Sound Power'
24	'Cycles'
25	'Torque'
26	'Moment'
27	'LoadFactor'
28	'Gravitational Acceleration'
29	'Element Force'
30	'Element Moment'
31	'Signal'
32	'Unitless Scalar'
33	'Unitless Real'
34	'Unitless Integer'
35	'Voltage'
36	'Electric Current'

These same data attributes may be assigned to the Ordinate and Z-Axis as well. The "Unknown" data type assumes unitless data. The "General" data type lets you define your own units to the data using the Abscissa Exponents.

AbscissaExpForce

An integer value which defines the force exponent for the abscissa data type. This needs to be set explicitly only for the "General" AbscissaDataType. For all others, it is defined automatically. If this attribute is changed, the AbscissaDataType will be set to "General". During importing and exporting, the exponents are used for unit conversion.

AbscissaExpLength

An integer value which defines the length exponents for the abscissa data type. This needs to be set explicitly only for the "General" AbscissaDataType. For all others, it is defined automatically. If this attribute is changed, the AbscissaDataType will be set to "General". If AbscissaTypeQual is set to rotation, the length is assumed to be unitless and expressed in radians. During importing and exporting, the exponents are used for unit conversion.

AbscissaExpTemp

An integer value which defines the temperature exponents for the abscissa data type. This applies needs to be set explicitly only for the "General" AbscissaDataType. For all others, it is defined automatically. If this attribute is changed, the AbscissaDataType will be set to "General". The standard abscissa data types use the Kelvin or Rankine scale to express temperature. During importing and exporting, the exponents are used for unit conversion.

AbscissaExpTime

An integer value which defines the time exponents for the abscissa data type. This attribute is not used for any units conversion. Note that there is no place to store this attribute in a Universal file or ADF, so any changes made to this attribute will be lost in a round-trip through these file formats.

AbscissaInc

If AbscissaSpacing is even, this stores the spacing, or increment, of the abscissa values. If AbscissaSpacing is uneven, this stores the maximum abscissa value of the function.

AbscissaMin

This stores the minimum abscissa value. If the AbscissaSpacing is even, all other abscissa values are calculated from this minimum using AbscissaInc. You cannot modify the minimum abscissa value of unevenly spaced data.

AbscissaOffset

OBSOLETE and not available.

AbscissaSpacing

Defines the abscissa spacing as even or uneven. If the spacing is defined as even, the Abscissa is not stored, but is generated as necessary from AbscissaMin and AbscissaInc. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	`'Uneven'`
1	`'Even'`

AbscissaTypeQual

This attribute affects the interpretation of the units exponents. This attribute is necessary for all abscissa data types except for "General". The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type

0	`'Translation'`
1	`'Rotation'`
2	`'Translation Squared'`
3	`'Rotation Squared'`

The difference in interpretation of the units exponent is in the length. For rotation, the length is unitless and assumed to be in radians.

AbscissaUnitsLab

This is a character string of maximum length 20 that can be used to override the default abscissa units label. It will appear in place of the default abscissa units label when the function is graphed in I-DEAS.

AmplitudeUnits

This attribute, together with Normalization, defines the amplitude units for spectral data. The available data types and their corresponding numeric identifier are shown in the table below.

A linear spectrum computed with half-peak units (and 'Units squared' Normalization) will have an amplitude of 0.5 when analyzing a unit sine wave. A spectrum computed with peak units will have unit amplitude when analyzing a unit sine wave. A spectrum computed with RMS units will have an amplitude of sqrt(0.5) when analyzing a unit sine wave.

CoordSys

OBSOLETE and not available.

CreateDate

Lists the date that the function was created. It is automatically stored when the function is created. While it can be modified in MATLAB, it will be overwritten when written to an ADF. It is stored as a character string of length 20, in the format "`DD-MMM-YY HH:MM:SS`".

ExpDampingFact

Defines the damping factor used as the decay rate of an exponential window that has been applied to a time history. If the damping factor is known, the damping values calculated from parameter estimation during a modal analysis can be corrected to their true values.

FunctionType

Defines the type of data stored in the function. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	`'General or Unknown'`

1	'Time Response'
2	'Auto Spectrum'
3	'Cross Spectrum'
4	'Frequency Response Function'
5	'Transmissibility'
6	'Coherence'
7	'Auto Correlation'
8	'Cross Correlation'
9	'Power Spectral Density (PSD)'
10	'Energy Spectral Density (ESD)'
11	'Probability Density Function'
12	'Spectrum'
13	'Cumulative Frequency Distribution'
14	'Peaks Valley'
15	'Stress/Cycles'
16	'Strain/Cycles'
17	'Orbit'
18	'Mode Indicator Function'
19	'Force Pattern'
20	'Partial Power'
21	'Partial Coherence'
22	'Eigenvalue'
23	'Eigenvector'
24	'Shock Response Spectrum'
25	'Finite Impulse Response Filter'
26	'Multiple Coherence'
27	'Order Function'
28	'Phase Compensation'
29	'Harmonic Function'
30	'Octave'
31	'Temperature'
32	'Stress vs Strain'
33	'Life'
34	'Campbell Diagram'

In I-DEAS, some operations may be restricted to certain function types. For example, only functions of type "Time Response", "Peaks Valley", and "Finite Impulse Response Filter" may be exported to an ATI file.

IDLine1

This is also called the function Title, used for descriptive text. It is limited to 80 characters. When a function is plotted in I-DEAS, the IDLine1 appears on the legend by default. If a function was created by measurement in I-DEAS Test, then the IDLine1 will be taken from the label of the response channel in the channel table.

IDLine2

The second function descriptor line, used for descriptive text. It is limited to 80 characters. If a function was created by measurement in I-DEAS Test, then the IDLine2 will be taken from the label of the reference channel in the channel table.

IDLine3

The third function descriptor line, used for descriptive text. It is limited to 80 characters.

IDLine4

The fourth function descriptor line, used for descriptive text. It is limited to 80 characters. When a function is plotted in I-DEAS, the IDLine4 appears on the legend by default. (But it is referred to as "Line 3" in the Legend management form.) If a function was created by measurement in I-DEAS Test, then the IDLine4 will be taken from the Measurement Description for that run.

IRIGTime

The IRIG is a time stamp that can be used to associate a global time value with a particular record. Practically speaking, this can be used to synchronize datasets acquired using different hardware in a lab, or keep an accurate record of when a measurement was made. The IRIG Time format is of the form `'AAA:HH:MM:SS.SSSSSS'`, where AAA is the number of days from the start of the year (with January 1 being day 1).

LoadCase

OBSOLETE and not available.

MaxOrdValImag

OBSOLETE and not available.

MaxOrdValReal

OBSOLETE and not available.

MeasurementRun

Integer value which specifies the run number of the measurement. It is used by I-DEAS Test to indicate groups of data acquired at the same time. It is automatically assigned during measurement so that it is a unique number for the ADF.

MinOrdValImag

OBSOLETE and not available.

MinOrdValReal

OBSOLETE and not available.

ModifyDate

Lists the date that the function was last modified. It is automatically updated when the function is modified. While this attribute can be modified in MATLAB, it will be overwritten when written to an ADF. It is stored as a character string of length 20, in the format "`DD-MMM-YY   HH:MM:SS`".

Normalization

This attribute, together with AmplitudeUnits, is set by I-DEAS Test during data acquisition to document the normalization method used to define a spectrum. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	`'Unknown'`
1	`'Units squared'`
2	`'Units squared/Hz'`
3	`'Units squared sec/Hz'`

The normalization is important for properly computing the RMS of a spectrum. The names are most meaningful for an "Auto Spectrum" FunctionType. For this type of function, if the normalization is "Units squared", then the sum of the spectral values should match the mean square of the time response. If the normalization is "Units squared/Hz", then the sum of the spectral values times the frequency increment should match the mean square of the time response. If the normalization is "Units squared sec/Hz", then the sum of the spectral values times the square of the frequency increment should match the mean square of the time response.

For linear spectra, these characteristics are true of the modulus squared of the spectrum.

NumberElements

Integer value which defines the number values in the function. It is the same as the number of Abscissa and Ordinate values in the function.

OctaveAvgType

Integer value which defines the type of response averaging applied to octave results. This attribute is meaningful only for the Octave function type.

Numeric Identifier	Octave Average Type
0	`'None'`
1	`'Fast'`
2	`'Slow'`
3	`'Impulse'`
4	`'Linear'`

The octave average type is stored as an integer.

OctaveFormat

This attribute sets the octave format that was used to define a spectrum. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	None
1	Octave
3	One third octave (1/3)
n	1/n octave

The octave format is stored as an integer.

OctaveOverallRMS

OBSOLETE and not available.

OctaveWeightedRMS

OBSOLETE and not available.

OrdDenDataType

Defines the data type for the ordinate denominator. The available data types and their corresponding numeric identifier are shown in the table under AbscissaDataType. The attribute specified determines how the data is converted when it is stored and retrieved by I-DEAS.

The denominator is ordinarily a reference or an input excitation. If a reference measurement was not made, the denominator qualifiers are not used (should be set to "Unknown"). The ordinate numerator and denominator data types are stored separately to accommodate functions such as frequency response functions.

OrdDenExpForce

An integer value which defines the force exponents for the ordinate denominator data type. This needs to be set explicitly only for the "General" OrdDenDataType. For all others, it is defined automatically. If this attribute is changed, the OrdDenDataType will be set to "General". During importing and exporting, the exponents are used for unit conversion.

OrdDenExpLength

An integer value which defines the length exponents for the ordinate denominator data type. This needs to be set explicitly only for the "General" OrdDenDataType. For all others, it is defined automatically. If this attribute is changed, the OrdDenDataType will be set to "General". If OrdDenTypeQual is set to rotation, the length is assumed to be unitless and expressed in radians. During importing and exporting, the exponents are used for unit conversion.

OrdDenExpTemp

An integer value which defines the temperature exponents for the ordinate denominator data type. This needs to be set explicitly only for the "General" OrdDenDataType. For all others, it is defined automatically. If this attribute is changed, the OrdDenDataType will be set to "General". The standard ordinate data types use the Kelvin or Rankine scale to express temperature. During importing and exporting, the exponents are used for unit conversion.

OrdDenExpTime

An integer value which defines the time exponents for the ordinate denominator data type. This attribute is not used for any units conversion. Note that there is no place to store this attribute in a Universal file or ADF, so any changes made to this attribute will be lost in a round-trip through these file formats.

OrdDenTypeQual

This attribute affects the interpretation of the ordinate denominator units exponents. This attribute is necessary for all OrdDenDataTypes except for "General". The available data types and their corresponding I-DEAS numeric identifier are shown in the table under AbscissaTypeQual.

OrdNumDataType

Defines the data type for the ordinate numerator. The available data types and their corresponding numeric identifier are shown in the table under AbscissaDataType. The attribute specified determines how the data is converted when it is stored and retrieved by I-DEAS.

The numerator is ordinarily the response to an input excitation. If a reference measurement was not made, the ordinate data should be stored in the numerator. The ordinate numerator and denominator data types are handled separately to accommodate functions such as frequency response functions.

OrdNumExpForce

An integer value which defines the force exponents for the ordinate numerator data type. This needs to be set explicitly only for the "General" OrdNumDataType. For all others, it is defined automatically. If this attribute is changed, the OrdNumDataType will be set to "General". During importing and exporting, the exponents are used for unit conversion.

OrdNumExpLength

An integer value which defines the length exponents for the ordinate numerator data type. This needs to be set explicitly only for the "General" OrdNumDataType. For all others, it is defined automatically. If this attribute is changed, the OrdNumDataType will be set to "General". If OrdNumTypeQual is set to rotation, the length is assumed to be unitless and expressed in radians. During importing and exporting, the exponents are used for unit conversion.

OrdNumExpTemp

An integer value which defines the temperature exponents for the ordinate numerator data type. This needs to be set explicitly only for the "General" OrdNumDataType. For all others, it is defined automatically. If this attribute is changed, the OrdNumDataType will be set to "General". The standard ordinate data types use the Kelvin or Rankine scale to express temperature. During importing and exporting, the exponents are used for unit conversion.

OrdNumExpTime

An integer value which defines the time exponents for the ordinate numerator data type. This attribute is not used for any units conversion. Note that there is no place to store this attribute in a Universal file or ADF, so any changes made to this attribute will be lost in a round-trip through these file formats.

OrdNumTypeQual

This attribute affects the interpretation of the ordinate numerator units exponents. This attribute is necessary for all OrdNumDataTypes except for "General". The available data types and their corresponding I-DEAS numeric identifier are shown in the table under AbscissaTypeQual.

OrdOffsetImag

OBSOLETE and not available.

OrdOffsetReal

OBSOLETE and not available.

OrdScaleImag

OBSOLETE and not available.

OrdScaleReal

OBSOLETE and not available.

Ordinate

The ordinate values are the *y* values in the (*x*,*y*) data points of the function. The ordinate values are stored as a column vector of either real or complex numbers. OrdinateType specifies the type of data stored.

OrdinateAxisLab

This is a character string of maximum length 20 that can be used to override the default Ordinate axis label. It will appear in place of the default axis label when the function is graphed in I-DEAS.

OrdinateType

This attribute determines the ordinate value storage. This attribute is set automatically based on the characteristics of the ordinate. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
2	`'Real Single'` [precision]
4	`'Real Double'` [precision] (not supported currently)
5	`'Complex Single'` [precision]
6	`'Complex Double'` [precision] (not supported currently)

(Although I-DEAS stores data in single precision, all data is stored in MATLAB in double precision.)

OrdinateUnitsLab

This is a character string of maximum length 20 that can be used to override the default Ordinate units label. It will appear in place of the default units label when the function is graphed in I-DEAS.

OwnerName

Character string of length 16 which identifies the username of the person who created the function.

PulsesPerRev

Specifies the number of pulses per revolution or pulses per cycle for a tachometer measurement. If SamplingType is "RPM from Tach" or "Frequency from Tach", this attribute indicates the number of pulses per revolution or cycle that were used to compute the RPM or frequency.

ReferenceCoord

Character string of up to 14 characters which defines the reference coordinate of the function. It is a combination of ReferenceNode and ReferenceDir. Note that setting the ReferenceCoord also sets the ReferenceNode and ReferenceDir, and vice versa.

The reference coordinate normally refers to the physical orientation of the exciter on a tested structure. This information is made up of a location number, direction, and sense. For example, a reference coordinate of '1X+' indicates a location of point number 1 on the structure, with a positive direction along the X axis.

ReferenceDir

String of up to 4 characters which identifies the direction and sense of the reference. Any four characters may be used, but physical meaning is attached to the directions 'X+', 'X-', 'Y+', 'Y-', 'Z+', 'Z-', 'RX+', 'RX-', 'RY+', 'RY-', 'RZ+', and 'RZ-'. ReferenceNode, combined with ReferenceDir, make up ReferenceCoord.

ReferenceEntity

OBSOLETE and not available.

ReferenceNode

Integer value which specifies the node number of the reference. Enter a zero if you do not want a value to appear in the attributes. ReferenceNode, combined with ReferenceDir, make up ReferenceCoord.

ResponseCoord

Character string of up to 14 characters which defines the response coordinate of the function. It is a combination of ResponseNode and ResponseDir. Note that setting the ResponseCoord also sets the ResponseNode and ResponseDir, and vice versa.

The response coordinate normally refers to the physical orientation of a measured response to an input to the structure. This information is made up of a location number, direction, and sense. For example, a response coordinate of '2X+' indicates a location of point number 2 on the structure, with a positive direction along the X axis.

ResponseDir

String of up to 4 characters which identifies the direction and sense of the response. Any four characters may be used, but physical meaning is attached to the directions 'X+', 'X-', 'Y+', 'Y-', 'Z+', 'Z-', 'RX+', 'RX-', 'RY+', 'RY-', 'RZ+', and 'RZ-'. ResponseNode, combined with ResponseDir, make up ResponseCoord.

ResponseEntity

OBSOLETE and not available.

ResponseNode

Integer value which specifies the node number of the response. Enter a zero if you do not want a value to appear in the attributes. ResponseNode, combined with ResponseDir, make up ResponseCoord.

SamplingType

This attribute documents the sampling type that was used during a data measurement. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	`'Dynamic'`
1	`'Static'`
2	`'RPM From Tach'`
3	`'Frequency From Tach'`

Sampling types other than "Dynamic" apply to data obtained during Transient Measurements and Order Tracking.

SetRecord

Integer value specifying the set number the function is associated with. Single records are specified with a SetRecord of 0. When exported, all functions with a given nonzero SetRecord are stored in a Function Set, which can be plotted in I-DEAS with XYZ plotting. When exporting to an existing ADF, if Function Sets already exist in the ADF, SetRecord will specify the offset from the highest set number already written to the ADF.

UserValue1

Specifies a real value of significance to the user. This attribute could be used to store a value that does not fall under any of the other attributes.

UserValue2

Specifies a real value of significance to the user. This attribute could be used to store a value that does not fall under any of the other attributes.

UserValue3

Specifies a real value of significance to the user. This attribute could be used to store a value that does not fall under any of the other attributes.

UserValue4

Specifies a real value of significance to the user. This attribute could be used to store a value that does not fall under any of the other attributes.

Version

Integer value which specifies the function's version number. When exporting to an ADF, the combination of ReferenceCoord, ResponseCoord, and Version must be unique. No such requirement is imposed in MATLAB. If a record with the same combination already exists in the ADF, the version number will be incremented until a unique combination is achieved.

WeightingType

This attribute specifies the weighting function that was applied to generate an acoustic spectrum. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	'None' (or Unknown)
1	`'A weighting'`
2	`'B weighting'`
3	`'C weighting'`
4	`'D weighting'`

WindowType

This attribute documents the type of window that was applied to the data when it was processed. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	`'None'`
1	`'Hanning Narrow'`
2	`'Hanning Broad'`
3	`'Flattop'`
4	`'Exponential'`
5	`'Impact'`
6	`'Impact and Exponential'`

ZAxisDataType

Defines the data type for the ZGeneralValue associated with this function. Z-axis values are used in Function Sets to position functions on an XYZ plot. The available data types and their corresponding I-DEAS numeric identifier are shown in the table under

AbscissaDataType. The attribute specified determines how the data is converted when it is stored and retrieved by I-DEAS.

ZAxisExpForce

An integer value which defines the force exponents for the ZGeneralValue associated with this function. This needs to be specified explicitly only for the "General" ZAxisDataType. For all others, it is defined automatically. If this attribute is changed, the ZAxisDataType will be set to "General". During importing and exporting, the exponents are used for unit conversion.

ZAxisExpLength

An integer value which defines the length exponents for the ZGeneralValue associated with this function. This needs to be specified explicitly only for the "General" ZAxisDataType. For all others, it is defined automatically. If this attribute is changed, the ZAxisDataType will be set to "General". If ZAxisTypeQual is set to rotation, the length is assumed to be unitless and expressed in radians. During importing and exporting, the exponents are used for unit conversion.

ZAxisExpTemp

An integer value which defines the temperature exponents for the ZGeneralValue associated with this function. This needs to be specified explicitly only for the "General" ZAxisDataType. For all others, it is defined automatically. If this attribute is changed, the ZAxisDataType will be set to "General". The standard ordinate data types use the Kelvin or Rankine scale to express temperature. During importing and exporting, the exponents are used for unit conversion.

ZAxisExpTime

An integer value which defines the time exponents for the Z axis data type. This attribute is not used for any units conversion. Note that there is no place to store this attribute in a Universal file or ADF, so any changes made to this attribute will be lost in a round-trip through these file formats.

ZAxisTypeQual

This attribute affects the interpretation of the Z axis units exponents for the ZGeneralValue associated with this function. This attribute is necessary for all ZAxisDataTypes except for "General". The available data types and their corresponding I-DEAS numeric identifier are shown in the table under AbscissaTypeQual.

ZGeneralValue

Specifies a real number for the Z axis value associated with this function (usually a member of a function set). This is used when plotting XYZ or waterfall plots to define the Z axis of the plot.

ZOrderValue

Specifies a real number for the order value associated with this function (usually a member of a function set). This is used when plotting XYZ or waterfall plots to define the Z axis of the plot when the Z axis is of data type Order.

ZRPMValue

Specifies a real number for the RPM value associated with this function (usually a member of a function set). This is used when plotting XYZ or waterfall plots to define the Z axis of the plot when the Z axis is of data type RPM.

ZTimeValue

Specifies a real number for the time value associated with this function (usually a member of a function set). This is used when plotting XYZ or waterfall plots to define the Z axis of the plot when the Z axis is of data type Time.

Shape Data Attributes

Shape Data Attributes (sorted by subject)

IMAT Home Page
Alphabetical listing

Shape identifiers

- IDLine1
- IDLine2
- IDLine3
- IDLine4
- IDLine5
- CreateDate
- ModifyDate

Modal information

- Frequency
- Damping
- ModalMassReal
- ModalMassImag
- ModalDampingReal
- ModalDampingImag
- SolutionMethod
- ReferenceNode
- ReferenceDir
- ReferenceCoord
- ResponseNode
- ResponseDir
- ResponseCoord

Shape information

- Nodes
- NumberNodes
- ShapeType
- DOFType
- CSType
- Shape
- OrdNumDataType
- OrdNumTypeQual
- OrdNumExpLength
- OrdNumExpForce
- OrdNumExpTemp
- OrdDenDataType
- OrdDenTypeQual
- OrdDenExpLength
- OrdDenExpForce
- OrdDenExpTemp

Other miscellaneous information

- AbscissaAxisLab
- AbscissaUnitsLab
- OrdinateAxisLab
- OrdinateUnitsLab
- OwnerName

Shape Data Attributes (sorted alphabetically)

IMAT Home Page
Subject listing

- AbscissaAxisLab
- AbscissaUnitsLab
- CreateDate
- CSType
- DOFType
- Damping
- Frequency
- IDLine1
- IDLine2
- IDLine3
- IDLine4
- IDLine5
- ModalDampingImag
- ModalDampingReal
- ModalMassImag
- ModalMassReal
- ModifyDate
- Nodes
- NumberNodes
- OrdDenDataType

- OrdDenExpForce
- OrdDenExpLength
- OrdDenExpTemp
- OrdDenExpTime
- OrdDenTypeQual
- OrdNumDataType
- OrdNumExpForce
- OrdNumExpLength
- OrdNumExpTemp
- OrdNumExpTime
- OrdNumTypeQual
- OrdinateAxisLab
- OrdinateUnitsLab
- OwnerName
- ReferenceCoord
- ReferenceDir
- ReferenceNode
- ResponseCoord
- ResponseDir
- ResponseNode
- Shape
- ShapeType
- SolutionMethod

AbscissaAxisLab

This is a character string of maximum length 20 that can be used to override the default Abscissa axis label. It is used only with operating deflection shapes.

AbscissaUnitsLab

This is a character string of maximum length 20 that can be used to override the default Abscissa units label. It is used only with operating deflection shapes.

CreateDate

Lists the date that the shape was created. It is automatically stored when the shape is created. While it can be modified in MATLAB, it will be overwritten when exported to an ADF. It is stored as a character string of length 20, in the format "`DD-MMM-YY HH:MM:SS`".

CSType

This attribute defines the coordinate system type in which the shape coefficients are defined. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	Displacement
1	Basic (global)

2	Other

DOFType

This attribute defines the number of degrees of freedom per node, which affects the Shape coefficient storage. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
3	`'3DOF'` [X,Y,Z (translational only)]
6	`'6DOF'` [X,Y,Z,RX,RY,RZ (translational and rotational)]

Damping

Defines the damping of the mode as a fraction of critical damping. For example, 1% of critical damping is stored as 0.01.

Frequency

Defines the undamped natural frequency of the mode in Hertz.

IDLine1

This is also called the shape Title. It is limited to 80 characters.

IDLine2

The second shape descriptor line. It is limited to 80 characters.

IDLine3

The third shape descriptor line. It is limited to 40 characters.

IDLine4

The fourth shape descriptor line. It is limited to 80 characters.

IDLine5

The fifth shape descriptor line. It is limited to 80 characters.

ModalDampingImag

Specifies the imaginary part of the modal damping for this mode. This attribute is not set by I-DEAS until you explicitly normalize the mode, or until you export the shape to a universal file.

For a real mode, this attribute is not used. For a complex mode, ModalDampingImag holds the imaginary part of the "modal B" for this complex mode. For more information, read the "Complex Mode Solution" article in I-DEAS SmartView.

Modal damping is considered to have the same units as the shape coefficients (defined by OrdNumDataType and OrdDenDataType).

ModalDampingReal

Specifies the real part of the modal damping for this mode. This attribute is not set by I-DEAS until you explicitly normalize the mode, or until you export the shape to a universal file.

For a real mode, this attribute is not used. For a complex mode, ModalDampingReal holds the real part of the "modal B" for this complex mode. For more information, read the "Complex Mode Solution" article in I-DEAS SmartView.

Modal damping is considered to have the same units as the shape coefficients (defined by OrdNumDataType and OrdDenDataType).

ModalMassImag

Specifies the imaginary part of the modal mass for this mode. This attribute is not set by I-DEAS until you explicitly normalize the mode, or until you export the shape to a universal file.

For a real mode, the ModalMassImag is zero, and ModalMassReal is the modal mass of the mode. For a complex mode, ModalMassImag holds the imaginary part of the "modal A" for this complex mode. For more information, read the "Complex Mode Solution" article in I-DEAS SmartView.

Modal mass is considered to have the same units as the shape coefficients (defined by OrdNumDataType and OrdDenDataType).

ModalMassReal

Specifies the real part of the modal mass for this mode. This attribute is not set by I-DEAS until you explicitly normalize the mode, or until you export the shape to a universal file.

For a real mode, the ModalMassImag is zero, and ModalMassReal is the modal mass of the mode. For a complex mode, ModalMassReal holds the real part of the "modal A" for this complex mode. For more information, read the "Complex Mode Solution" article in I-DEAS SmartView.

Modal mass is considered to have the same units as the shape coefficients (defined by OrdNumDataType and OrdDenDataType).

ModifyDate

Lists the date that the shape was last modified. It is automatically updated when the shape is modified. While it can be modified in MATLAB, it will be overwritten when exported to an ADF. It is stored as a character string of length 20, in the format "`DD-MMM-YY HH:MM:SS`".

Nodes

Integer column vector of node labels for which Shape coefficients are stored.

NumberNodes

Specifies the number of nodes used in the mode shape definition. This defines the size of Nodes, and when multiplied by DOFType, defines the size of Shape.

OrdDenDataType

Defines the data type for the ordinate denominator of the frequency response functions from which the shape was defined (mode shapes are considered to be in the same units as the FRFs). For mode shapes from test, this would typically be "Excitation Force". For mode shapes from analysis, this would typically be "Unknown" (i.e., unitless). This data type determines unit conversion during import and export.

The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	`'Unknown'`
1	`'General'`
2	`'Stress'`
3	`'Strain'`
5	`'Temperature'`
6	`'Heat Flux'`
8	`'Displacement'`
9	`'Force'`
11	`'Velocity'`
12	`'Acceleration'`
13	`'Excitation Force'`
15	`'Pressure'`
16	`'Mass'`
17	`'Time'`
18	`'Frequency'`
19	`'RPM'`
20	`'Order'`
21	`'Sound Pressure'`
22	`'Sound Intensity'`
23	`'Sound Power'`
24	`'Cycles'`
25	`'Torque'`

26	`'Moment'`
27	`'LoadFactor'`
28	`'Gravitational Acceleration'`
29	`'Element Force'`
30	`'Element Moment'`
31	`'Signal'`
32	`'Unitless Scalar'`
33	`'Unitless Real'`
34	`'Unitless Integer'`
35	`'Voltage'`
36	`'Electric Current'`

OrdDenExpForce

An integer value which defines the force exponents for the data type of the denominator of the shape. This needs to be specified explicitly only for the "General" OrdDenDataType. For all others, it is defined automatically. If this attribute is changed, the OrdDenDataType will be set to "General". During importing and exporting, the exponents are used to convert units.

OrdDenExpLength

An integer value which defines the length exponents for the data type of the denominator of the shape. This needs to be specified explicitly only for the "General" OrdDenDataType. For all others, it is defined automatically. If this attribute is changed, the OrdDenDataType will be set to "General". If OrdDenTypeQual is set to rotation, the length is assumed to be unitless and expressed in radians. During importing and exporting, the exponents are used to convert units.

OrdDenExpTemp

An integer value which defines the temperature exponents for the data type of the denominator of the shape. This needs to be specified explicitly only for the "General" OrdDenDataType. For all others, it is defined automatically. If this attribute is changed, the OrdDenDataType will be set to "General". The standard ordinate data types use the Kelvin or Rankine scale to express temperature. During importing and exporting, the exponents are used to convert units.

OrdDenExpTime

An integer value which defines the time exponents for the ordinate denominator data type. This is a "virtual" attribute, in that it is not actually stored in the object. It is a read-only attribute. The value returned is based on the data type. This attribute is not used for any units conversion.

OrdDenTypeQual

This attribute affects the interpretation of the ordinate numerator units exponents. This attribute is necessary for all OrdDenDataTypes except for "General". The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	`'Translation'`
1	`'Rotation'`

The difference in interpretation of the units exponent is in the length. For rotation, the length is unitless and assumed to be in radians.

OrdNumDataType

Defines the data type for the ordinate numerator of the frequency response functions from which the shape was defined (mode shapes are considered to be in the same units as the FRF's). For mode shapes from test, this would typically be "Acceleration", or other data type corresponding to the measurement. For mode shapes from analysis, this would typically be "Displacement". This data type determines unit conversion during import and export from I-DEAS.

The available data types and their corresponding numeric identifier are shown in the table under OrdDenDataType.

OrdNumExpForce

An integer value which defines the force exponents for the data type of the numerator of the shape. This needs to be specified explicitly only for the "General" OrdNumDataType. For all others, it is defined automatically. If this attribute is changed, the OrdNumDataType will be set to "General". During importing and exporting, the exponents are used to convert units.

OrdNumExpLength

An integer value which defines the length exponents for the data type of the numerator of the shape. This needs to be specified explicitly only for the "General" OrdNumDataType. For all others, it is defined automatically. If this attribute is changed, the OrdNumDataType will be set to "General". If OrdNumTypeQual is set to rotation, the length is assumed to be unitless and expressed in radians. During importing and exporting, the exponents are used to convert units.

OrdNumExpTemp

An integer value which defines the temperature exponents for the data type of the numerator of the shape. This needs to be specified explicitly only for the "General" OrdNumDataType. For all others, it is defined automatically. If this attribute is changed, the OrdNumDataType will be set to "General". The standard ordinate data types use the Kelvin or Rankine scale to express temperature. During importing and exporting, the exponents are used to convert units.

OrdNumExpTime

An integer value which defines the time exponents for the ordinate numerator data type. This is a "virtual" attribute, in that it is not actually stored in the object. It is a read-only attribute. The value returned is based on the data type. This attribute is not used for any units conversion.

OrdNumTypeQual

This attribute affects the interpretation of the ordinate numerator units exponents. This attribute is necessary for all OrdNumDataTypes except for "General". The available data types and their corresponding numeric identifier are shown in the table under OrdDenTypeQual.

OrdinateAxisLab

This is a character string of maximum length 20 that can be used to override the default Ordinate axis label. It is used only for operating deflection shapes.

OrdinateUnitsLab

This is a character string of maximum length 20 that can be used to override the default Ordinate units label. It is used only for operating deflection shapes.

OwnerName

Character string of length 16 which identifies the username of the person who created the shape.

ReferenceCoord

Character string of up to 14 characters which defines the reference coordinate that was used to scale the modal coefficients. It is a combination of ReferenceNode and ReferenceDir. Note that setting the ReferenceCoord changes the ReferenceNode and ReferenceDir, and vice versa.

The reference coordinate normally refers to the physical orientation of the exciter on a tested structure. This information is made up of a location number, direction, and sense. For example, a reference coordinate of '1X+' indicates a location of point number 1 on the structure, with a positive direction along the X axis.

ReferenceDir

String of up to 4 characters which identifies the direction and sense of the reference used to scale the mode shape. Any four characters may be used. ReferenceNode, combined with ReferenceDir, make up ReferenceCoord.

ReferenceNode

Integer value which specifies the node number of the reference used to scale the mode shape. Enter a zero if you do not want a value to appear in the attributes. ReferenceNode, combined with ReferenceDir, make up ReferenceCoord.

ResponseCoord

Character string of up to 14 characters which defines the response coordinate used to scale the modal coefficients. It is a combination of ResponseNode and ResponseDir. Note that setting the ResponseCoord changes the ResponseNode and ResponseDir, and vice versa.

The response coordinate normally refers to the physical orientation of a measured response to an input to the structure. This information is made up of a location number, direction, and sense. For example, a response coordinate of '2X+' indicates a location of point number 2 on the structure, with a positive direction along the X axis.

ResponseDir

String of up to 4 characters which identifies the direction and sense of the response used to scale the mode shape. Any four characters may be used. ResponseNode, combined with ResponseDir, make up ResponseCoord.

ResponseNode

Integer value which specifies the node number of the response used to scale the mode shape. Enter a zero if you do not want a value to appear in the attributes. ResponseNode, combined with ResponseDir, make up ResponseCoord.

Shape

Real or complex column vector of mode shape coefficients. The size of Shape should be NumberNodes times DOFType. All degrees of freedom for each node are stored together, and the ordering of the nodes matches the order of the Nodes attribute.

ShapeType

This attribute defines the mode shape type as either a real mode or complex mode. The shape type affects the interpretation of the modal mass and modal damping attributes. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
1	`'Real'`
2	`'Complex'`

SolutionMethod

Documents the curve-fitting method that was used to generate the modal coefficients. The available data types and their corresponding numeric identifier are shown in the table below.

Numeric Identifier	Data Type
0	`'User Defined'`
1	`'Move Response'`
2	`'Complex Exponential'`

3	`'Circle Fit'`
4	`'Polyreference'`
5	`'SDOF Polynomial'`
6	`'Direct Parameter'`
14	`'Orthogonal Polyreference'`
15	`'Frequency Polyreference'`
20	`'Structural Modification'`
21	`'COMAC'`
22	`'Mode Shape Math'`
23	`'Real Mode Approximation'`
24	`'Generic Move Response'`

IMAT_RESULT Data Attributes

IMAT_RESULT Data Attributes (sorted by subject)

IMAT Home Page
Alphabetical listing

Result identifiers

- AnalysisType
- IDLine1
- IDLine2
- IDLine3
- IDLine4
- IDLine5
- ModelType
- Name

Integer-valued information

- BoundaryCondition
- CreationOption
- DesignSetID
- FrequencyNumber
- IterationNumber
- LoadSet
- ModeNumber
- NumberRetained
- SolutionSetID
- TimeStepNumber

Real-valued information

- EffectiveMass
- Eigenvalue
- EigenvalueImag
- EigenvalueReal
- Frequency
- HystereticDamping
- ModalA
- ModalAImag
- ModalAReal
- ModalB
- ModalBImag
- ModalBReal
- ModalMass
- ModalMassImag
- ModalMassReal
- ParticipationFactor
- Stiffness
- StiffnessImag
- StiffnessReal
- Time
- ViscousDamping

Result data information

- component
- data
- NumberValues
- DataLocation
- ExpForce
- ExpLength
- ExpTemperature
- ExpTime
- ResultType

Result Data Attributes (sorted alphabetically)

IMAT Home Page
Subject listing

- AnalysisType
- BoundaryCondition
- component
- CreationOption
- data
- DataCharacteristic
- DataLocation
- DesignSetID
- EffectiveMass
- Eigenvalue
- EigenvalueImag

- EigenvalueReal
- ExpForce
- ExpLength
- ExpTemperature
- ExpTime
- Frequency
- FrequencyNumber
- HystereticDamping
- IDLine1
- IDLine2
- IDLine3
- IDLine4
- IDLine5
- IterationNumber
- LoadSet
- ModalA
- ModalAImag
- ModalAReal
- ModalB
- ModalBImag
- ModalBReal
- ModalMass
- ModalMassImag
- ModalMassReal
- ModelType
- ModeNumber
- Name
- NumberRetained
- NumberValues
- ParticipationFactor
- ResultType
- SolutionSetID
- Stiffness
- StiffnessImag
- StiffnessReal
- Time
- TimeStepNumber
- ViscousDamping

Analysis Type-specific Attributes

Some attributes are only valid for some AnalysisType values. Please review the table below.

	AnalysisType									
	Unknown	Statics	Normal Modes	Complex Eigenvalue First	Transient	Frequency Response	Buckling	Complex Eigenvalue Second	Static Nonlinear	Response Dynamics (10-

				Order				Order		16)
DesignSetID	X	X	X	X	X	X	X	X	X	X
IterationNumber		X	X							X
SolutionSetID	X	X	X	X	X	X	X	X	X	X
BoundaryCondition	X	X	X	X	X	X	X	X	X	X
LoadSet		X		X	X	X	X	X	X	X
ModeNumber			X	X			X	X		X
TimeStepNumber					X				X	X
FrequencyNumber						X				X
CreationOption	X	X	X	X	X	X	X	X	X	X
NumberRetained	X	X	X	X	X	X	X	X	X	X
Time					X				X	
Frequency			X			X				X
Eigenvalue							X			
ModalMass			X							X
ViscousDamping			X							X
HystereticDamping			X							X
EigenvalueReal				X				X		
EigenvalueImag				X				X		
ModalA				X						
ModalB				X						
Mass								X		
Stiffness								X		

AnalysisType

This attribute defines the type of analysis that generated this result. The numeric identifier and corresponding analysis type are shown in the table below.

Numeric Identifier	Analysis Type
0	`'Unknown'`
1	`'Static'`
2	`'Normal mode'`
3	`'Complex eigenvalue first order'`
4	`'Transient'`
5	`'Frequency response'`
6	`'Buckling'`
7	`'Complex eigenvalue second order'`
9	`'Static non-linear'`
10	`'Craig-Bampton constraint modes'`
11	`'Equivalent attachment modes'`
12	`'Effective mass modes'`
13	`'Effective mass matrix 1'`
14	`'Effective mass matrix 2'`
15	`'Distributed Load Load Distribution'`
16	`'Distributed Load Attachment Modes'`

BoundaryCondition

This is an integer specifying the boundary condition number for this result.

---

component

Lists the components associated with the data in the result. The component format depends on the DataLocation and the DataCharacteristic. The table below describes the format of the component listing based on DataLocation.

CreationOption

This is an integer specifying the creation option for this result. It only has meaning in I-deas.

---

data

Contains the DataLocation object. See here for more specifics.

DataCharacteristic

Defines the data characteristic for the result. This property is a member of the DataLocation objects. See here for details.

DataLocation

This is a special read-only attribute that summarizes the DataLocation object types as a cell array of strings.

DesignSetID

This is an integer specifying the design set ID for this result. It only has meaning in I-deas.

EffectiveMass

This is a 6x1 vector of real numbers containing the effective mass fractions for this result. This is only valid for mode shape results.

Eigenvalue

Real or complex number defining the eigenvalue. This is only valid for frequency-domain results. It is related to the Frequency attribute.

EigenvalueImag

Real number containing the imaginary portion of the Eigenvalue.

EigenvalueReal

Real number containing the real portion of the Eigenvalue.

ExpForce

An integer value which defines the force exponents for the data type of the denominator of the shape. This needs to be specified explicitly only for the "User defined" ResultType. For all others, it is defined automatically. If this attribute is changed, the ResultType will be set to "User defined". The data types use the Kelvin or Rankine scale to express temperature. During importing and exporting, the exponents are used to convert units.

ExpLength

An integer value which defines the length exponents for the data type of the denominator of the shape. This needs to be specified explicitly only for the "User defined" ResultType. For all others, it is defined automatically. If this attribute is changed, the

ResultType will be set to "User defined". The data types use the Kelvin or Rankine scale to express temperature. During importing and exporting, the exponents are used to convert units.

ExpTemperature

An integer value which defines the temperature exponents for the data type of the result. This needs to be specified explicitly only for the "User defined" ResultType. For all others, it is defined automatically. If this attribute is changed, the ResultType will be set to "User defined". The data types use the Kelvin or Rankine scale to express temperature. During importing and exporting, the exponents are used to convert units.

ExpTime

An integer value which defines the time exponents for the data. This needs to be specified explicitly only for the "User defined" ResultType. For all others, it is defined automatically. If this attribute is changed, the ResultType will be set to "User defined". The data types use the Kelvin or Rankine scale to express temperature. During importing and exporting, the exponents are used to convert units.

Frequency

Real number defining the undamped natural frequency. This is only valid for frequency-domain results. It is related to the EigenValue attribute.

For real normal modes (based on the AnalysisType attribute), it is the square root of the magnitude of the eigenvalue divided by 2π.

$$f_n = \text{sqrt}(|E|)/ 2\pi$$

where $|E|$ is the magnitude, which is $sqrt(Er\hat{}2 + Ei\hat{}2)$, Er is the real component of the eigenvalue, and Ei is the imaginary component of the eigenvalue.

For complex modes (based on the AnalysisType attribute), it is the sign of the imaginary component of the eigenvalue multiplied by the magnitude of the eigenvalue divided by 2π.

$$f_n = \text{sgn(Ei)} * |E| / 2\pi$$

FrequencyNumber

An integer value which defines the frequency number for the result. This is only valid for response analysis results in I-deas.

HystereticDamping

A real value which defines the hysteretic damping fraction for the result. It is only valid for frequency domain results. This value is entered as a fraction, so 1% would be entered as 0.01.

IDLine1

This is also called the shape Title. It is limited to 80 characters.

IDLine2

The second descriptor line. It is limited to 80 characters.

IDLine3

The third descriptor line. It is limited to 40 characters.

IDLine4

The fourth descriptor line. It is limited to 80 characters.

IDLine5

The fifth descriptor line. It is limited to 80 characters.

IterationNumber

An integer value which defines the iteration number for the result. This is only valid for results in I-deas.

LoadSet

An integer value which defines the load set number for the result. This is only valid for results in I-deas.

ModalA

Real or complex number defining the modal A value. This is only valid for complex eigenvalue results.

ModalAImag

A real number containing the imaginary portion of the ModalA value.

ModalAReal

A real number containing the real portion of the ModalA value.

ModalB

Real or complex number defining the modal B value. This is only valid for complex eigenvalue results.

ModalBImag

A real number containing the imaginary portion of the ModalB value.

ModalBReal

A real number containing the real portion of the ModalB value.

ModalMass

Real or complex number specifying the modal mass for mode shapes.

For a real mode, the ModalMassImag is zero, and ModalMassReal is the modal mass of the mode.

Modal mass is considered to have the same units as the mode shape coefficients (defined by ResultType).

ModalMassImag

Real number specifying the imaginary portion of the ModalMass for mode shapes.

ModalMassReal

Real number specifying the real portion of the ModalMass for mode shapes.

ModelType

Defines the model type for the result. This defines the analysis domain. The numeric identifier and corresponding model type are shown in the table below.

Model Type Numeric Identifier	ModelType
0	`Unknown`
1	`Structural`
2	`Heat transfer`
3	`Fluid flow`

ModeNumber

An integer value which defines the mode number for the result. This is only valid for mode shape results in I-deas.

Name

String up to length 80 containing the name of the result.

NumberRetained

An integer value which defines the mode number for the result. This is only valid for results in I-deas.

NumberValues

An integer value which specifies the total number of data values in the result. Setting this attribute will either truncate or expand the result. If the result is expanded, the new Data quantities will be 0.

ParticipationFactor

This is a 6x1 vector of real numbers containing the participation factors for this result. This is only valid for mode shape results.

ResultType

Defines the data type for the result. The available data types and their corresponding numeric identifier are shown in the table below. The ResultType determines units conversion factors between units systems.

Numeric Identifier	Component
2	`'Stress'`
3	`'Strain'`
4	`'Element Force'`
5	`'Temperature'`
6	`'Heat Flux'`
7	`'Strain Energy'`
8	`'Displacement'`
9	`'Reaction Force'`
10	`'Kinetic Energy'`
11	`'Velocity'`
12	`'Acceleration'`
13	`'Strain Energy Density'`
14	`'Kinetic Energy Density'`
15	`'Hydrostatic Pressure'`
16	`'Heat Gradient'`
17	`'Code Check Value'`
18	`'Coefficient of Pressure'`
19	`'Ply Stress'`
20	`'Ply Strain'`
21	`'Failure Index for Ply'`
22	`'Failure Index for Bonding'`
23	`'Reaction Heat Flow'`

24	'Stress Error Density'
25	'Stress Variation'
27	'Element Stress Resultant'
28	'Length'
29	'Area'
30	'Volume'
31	'Mass'
32	'Constraint Force'
34	'Plastic Strain'
35	'Creep Strain'
36	'Strain Energy Error Norm'
37	'Dynamic Stress At Nodes'
38	'Heat Transfer Coefficient'
39	'Temperature Gradient'
40	'Kinetic Energy Dissipation Rate'
41	'Strain Energy Error'
42	'Mass Flow'
43	'Mass Flux'
44	'Heat Flow'
45	'View Factor'
46	'Heat Load'
47	'Stress Component'
48	'Green Strain'
49	'Contact Forces'
50	'Contact Pressure'
51	'Contact Stress'
52	'Contact Friction Stress'
53	'Velocity Component'
54	'Heat Flux Component'
55	'Infrared Heat Flux'
56	'Diffuse Solar Heat Flux'
57	'Collimated Solar Heat Flux'
58	'Safety Factor'
59	'Fatigue Damage'

60	'Fatigue Damage With Direction'
61	'Fatigue Life'
62	'Quality Index'
63	'Stress With Direction'
64	'Translation With Direction'
65	'Rotation With Direction'
66	'Force With Direction'
67	'Moment With Direction'
68	'Translational Acceleration With Direction'
69	'Rotational Acceleration With Direction'
70	'Level Crossing Rate With Direction'
71	'Trans Shell Stress Resultant With Direction'
72	'Rot Shell Stress Resultant With Direction'
73	'Failure Index'
74	'Nodal Point Forces'
75	'Displacement Component'
76	'Acceleration Component'
77	'Force Component'
78	'Strain Component'
94	'Unknown Scalar'
95	'Unknown 3DOF Vector'
96	'Unknown 6DOF Vector'
97	'Unknown Symmetric Tensor'
98	'Unknown General Tensor'
99	'Unknown Stress Resultant'
101	'Gap Thickness'
102	'Solid Layer (+ surface)'
103	'Solid Layer (- surface)'
104	'Total Solid Layer'
105	'Flow Vector at Fill'
106	'Bulk Flow Vector'
107	'Core Displacement'
108	'Layered Shear Strain Rate'
109	'Shear Stress'

110	'Heat Flux (+ surface)'
111	'Heat Flux (- surface)'
112	'Layered Temperature'
113	'Bulk Temperature'
114	'Peak Temperature'
115	'Temperature at Fill'
116	'Mass Density'
117	'Pressure'
118	'Volumetric Skrinkage'
119	'Filling Time'
120	'Ejection Time'
121	'No-flow Time'
122	'Weld Line Meeting Angle'
123	'Weld Line Underflow'
124	'Original Runner Diameter'
125	'Optimized Runner Diameter'
126	'Change in Runner Diameter'
127	'Averaged Layered Cure'
128	'Layered Cure'
129	'Cure Rate'
130	'Cure Time'
131	'Induction Time'
132	'Temperature at Cure'
133	'Percent Gelation'
134	'Part Heat Flux (+ surface)'
135	'Part Heat Flux (- surface)'
136	'Part-Wall Temperature (+ surface)'
137	'Part-Wall Temperature (- surface)'
138	'Part Ejection Time'
139	'Part Peak Temperature'
140	'Part Average Temperature'
141	'Parting Temperature (+ surface)'
142	'Parting Temperature (- surface)'
143	'Parting Heat Flux (- surface)'

144	'Parting Heat Flux (+ surface)'
145	'Wall Temperature Convergence'
146	'Wall Temperature (- surface)'
147	'Wall Temperature (+ surface)'
148	'Line Heat flux'
149	'Line Pressure'
150	'Reynold's Number'
151	'Line Film Coefficient'
152	'Line Temperature'
153	'Line Bulk Temperature'
154	'Mold Temperature'
155	'Mold Heat Flux'
156	'Rod Heater Temperature'
157	'Rod Heater Flux'
158	'Original Line Diameter'
159	'Optimized Line Diameter'
160	'Change in Line Diameter'
161	'Air Traps'
162	'Weld Lines'
163	'Injection Growth'
164	'Temp Diff (Celsius)'
165	'Shear Rate'
166	'Viscosity'
167	'Percentage'
168	'Time'
169	'Flow Direction'
170	'Speed'
171	'Flow Rate'
172	'Thickness Ratio'
201	'Maximum Temperature'
202	'Minimum Temperature'
203	'Time of Maximum Temperature'
204	'Time of Minimum Temperature'
205	'Conductive Flux'

206	`'Total Flux'`
207	`'Residuals'`
208	`'View Factor Sum'`
209	`'Velocity Adjusted'`
210	`'Pressure (+ surface)'`
211	`'Pressure (- surface)'`
212	`'Static Pressure'`
213	`'Total Pressure'`
214	`'K-E Turbulence Energy'`
215	`'K-E Turbulence Dissipation'`
216	`'Fluid Density'`
217	`'Shear Stress (+ surface)'`
218	`'Shear Stress (- surface)'`
219	`'Roughness (+ surface)'`
220	`'Roughness (- surface)'`
221	`'Y+ (+ surface)'`
222	`'Y+ (- surface)'`
223	`'Fluid Temperature'`
224	`'Convective Heat Flux'`
225	`'Local Convection Coefficient'`
226	`'Bulk Convection Coefficient'`
301	`'Sound Pressure'`
302	`'Sound Power'`
303	`'Sound Intensity'`
304	`'Sound Energy'`
305	`'Sound Energy Density'`
1001	`'User defined'`

SolutionSetID

An integer value which defines the solution set ID for the result. This is only valid for results in I-deas.

Stiffness

Real or complex number specifying the modal stiffness for mode shapes.

For a real mode, the StiffnessImag is zero, and StiffnessReal is the modal stiffness of the mode.

Modal mass is considered to have the same units as the mode shape coefficients (defined by ResultType).

StiffnessImag

Real number specifying the imaginary portion of the Stiffness for mode shapes.

StiffnessReal

Real number specifying the real portion of the Stiffness for mode shapes.

Time

A real value which defines the time for the result. This is only valid for time-domain results.

TimeStepNumber

An integer value which defines the time step number for the result. This is only valid for time-domain results.

ViscousDamping

Real number defining the viscous damping. This is only valid for frequency-domain results. It is related to the EigenValue attribute.This value is entered as a fraction, so 1% would be entered as 0.01.

The viscous damping ratio is the negative of the real component of the eigenvalue divided by the magnitude of the eigenvalue:

$$\zeta = -Er / sqrt(Er^2 + Ei^2)$$

where *Er* is the real component of the eigenvalue, and *Ei* is the imaginary component of the eigenvalue.

IMAT_RESULT DataLocation Attributes

IMAT_RESULT DataLocation Attributes (sorted by subject)

IMAT Home Page

Properties

- component
- data
- DataCharacteristic
- DataLocation
- NumberElements
- NumberNodes
- NumberValues

---

component

Returns an MxN integer matrix of components. M is the number of data values, and N is the number of components, which varies by DataLocation. See the component attribute help for each DataLocation object.

The following attributes are specific to **Data At Nodes** results.

Property Name	Component Column Number	Description
`node`	1	Node ID. Must be positive.
`dir`	2	Direction. See the table below for specific identifiers.
`seid`	3	Superelement ID (default 0). Must be non-negative.

The following attributes are specific to **Data On Elements** results.

Property Name	Component Column Number	Description
`element`	1	Element ID. Must be positive.
`layer`	2	Layer number. Must be non-negative.
`seid`	3	Superelement ID (default 0) Must be non-negative.

The following attributes are specific to **Data At Nodes On Elements** results.

Property Name	Component Column Number	Description
`element`	1	Element ID. Must be positive.
`node`	2	Node ID.
`comp`	3	Component. See the table below for specific identifiers.
`layer`	4	Layer number. Must be non-negative.
`seid`	5	Superelement ID (default 0)
`eltype`	6	Nastran element type (default 0 for unknown). Must be non-negative.

The following attributes are specific to **Data On Elements At Nodes** results.

Property Name	Component Column Number	Description
`node`	1	Node ID. Must be non-negative.

`element`	2	Element ID.
`comp`	3	Component. See the table below for specific identifiers.
`eltype`	4	Nastran element type (default 0 for unknown). Must be non-negative.

The following attributes are specific to **Data At Points** results.

Property Name	Component Column Number	Description
`point`	1	Point ID. Must be positive.

The table below shows the valid components and their numeric representation for the component or direction for each DataLocation.

Numeric Identifier	Component
0	(scalar)
1	`'X'`
2	`'Y'`
3	`'Z'`
4	`'RX'`
5	`'RY'`
6	`'RZ'`
11	`'XX'`
22	`'YY'`
33	`'ZZ'`
12	`'XY'`
13	`'XZ'`
23	`'YZ'`
21	`'YX'`
31	`'ZX'`
32	`'ZY'`
41	`'Fx'`
42	`'Fy'`
43	`'Fxy'`
44	`'Mx'`
45	`'My'`

46	`'Mxy'`
47	`'Vx'`
48	`'Vy'`
51	`'MaxP'`
52	`'MidP'`
53	`'MinP'`
54	`'MaxS'`
57	`'MagT'`
58	`'MagR'`
59	`'VonM'`

The table below shows Nastran element types recognized and mapped by IMAT. This may not be an exhaustive list; if the element type you are looking for is not in this table, please refer to the Quick Reference Guide in your NASTRAN documentation. The element types are most easily found in the Item Codes section.

Numeric Identifier	Element Type
0	(unknown)
	0-D
11	CELAS1
12	CELAS2
13	CELAS3
14	CELAS4
40	CBUSH1D
102	CBUSH
20	CDAMP1
21	CDAMP2
22	CDAMP3
23	CDAMP4
86	CGAP
	1-D
1	CROD
10	CONROD
89	CRODNL
92	CONRODNL
34	CBAR
100	CBARA

2	CBEAM
94	CBEAMNL
69	CBEND
3	CTUBE
87	CTUBENL
	2-D
74	CTRIA3
75	CTRIA6
33	CQUAD4, CQUAD
64	CQUAD8
70	CTRIAR
82	CQUADR
88	CTRIA3NL
90	CQUAD4NL
95	CQUAD4C
96	CQUAD8C
97	CTRIA3C
98	CTRIA6C
144	CQUADX4
53	CTRIAX6
245	CQUADX8
201	CQUADFD
206	CTRIAFD
211	CTRIAD
214	CQUADXF
215	CQUADYF
	3-D
39	CTETRA
85	CTETRANL
205	CTETRAFD
210	CTETRAD
68	CPENTA
91	CPENTANL
204	CPENTAFD

209	CPENTAD
67	CHEXA
93	CHEXANL
202	CHEXAFD
207	CHEXAD

data

Returns a vector containing the data values for this result.

DataCharacteristic

This determines the valid components for this result. The table below displays the valid data characteristics, their corresponding numeric identifiers, and the valid components for each data characteristic.

Numeric Identifier	Data Characteristic	Valid Components
0	`'Unknown'`	Any
1	`'Scalar'`	Any
2	`'3DOF global translation vector'`	X, Y, Z
3	`'6DOF global translation & rotation vector'`	X, Y, Z, RX, RY, RZ
4	`'Symmetric global tensor'`	XX, YY, ZZ, XY, XZ, YZ
5	`'General global tensor'`	XX, YY, ZZ, XY, XZ, YZ, YX, ZX, ZY
6	`'Stress resultants'`	Fx, Fy, Fxy, Mx, My, Mxy, Vx, Vy

DataLocation

Returns a string containing the DataLocation description corresponding to the DataLocation object IMAT provides the following DataLocation objects.

DataLocation
Data At Nodes
Data On Elements
Data At Nodes On Elements
Data At Points
Data On Elements At Nodes

NumberElements

Number of unique element IDs. Only valid for **Data On Elements** and **Data At Nodes On Elements** result data locations. Setting this will truncate or expand the components and data as appropriate.

NumberNodes

Number of unique element IDs. Only valid for **Data At Nodes** and **Data On Elements At Nodes** result data locations. Setting this will truncate or expand the components and data as appropriate.

NumberValues

Number of data values (and component rows). Setting this will truncate or expand the components and data as appropriate.

FEM Data Attributes

FEM Class Format (sorted by subject)

IMAT Home Page

FEM Classes

Finite Element Models: IMAT_FEM Class
Coordinate Systems: IMAT_CS Class
Nodes: IMAT_NODE Class
Elements: IMAT_ELEM Class
Tracelines: IMAT_TL Class

IMAT_FEM Class

The IMAT_FEM class consists of several properties and methods. Only one FEM is permitted per IMAT_FEM object. In other words, the object must be scalar (1x1). The methods in this class perform checking to ensure that the data is consistent and complete. For example, it checks all of the node IDs referenced by the elements and tracelines to make sure that they all exist in the IMAT_NODE property. It also checks to make sure that all of the coordinate systems referenced in the IMAT_NODE property exist in the IMAT_CS property.

The four properties and their descriptions are shown in the table below. The contents of each of these objects are described in the following sections.

Property Name	Description
`.cs`	Coordinate system class (IMAT_CS)
`.node`	Node data class (IMAT_NODE)
`.elem`	Element data class (IMAT_ELEM)
`.tl`	Traceline data class (IMAT_TL)

Coordinate Systems: IMAT_CS Class

The IMAT_CS class contains coordinate system definitions. This class has the properties listed in the table below.

`.part`	Part information ({1x2} cell array)
`.id`	Coordinate system labels ([nx1] numeric vector)
`.name`	Coordinate system names ({nx1} cell array of strings)
`.color`	Coordinate system colors ([nx1] numeric vector)
`.type`	Coordinate system types ([nx1] numeric vector)
`.matrix`	Coordinate system transformation matrices ([4x3xn] double matrix)

The `part` property is a cell array, where the first element is an integer specifying the part UID, and the second field is a string containing the part name. This field is not used by IMAT, but is provided for completeness.

The `id` property is an nx1 numeric vector containing the coordinate system labels. These labels should be positive and unique. The coordinate system label '1' is reserved for the global (part) Cartesian coordinate system.

The `name` property is an nx1 cell array of strings containing the coordinate system names. Each name is a string. The strings can be varying lengths (hence the need for a cell array). The length of the cell array must be the same as the length of the `id` field.

The `color` property is an nx1 numeric vector containing the coordinate system colors. The colors are positive numbers representing colors using the color scheme defined by I-deas. The length of the `color` vector must be the same as the length of the `id` field.

The `type` property is an nx1 numeric vector containing the coordinate system types. The coordinate system type is 0 for Cartesian, 1 for cylindrical, and 2 for spherical.

The `matrix` property is an 4x3xn double matrix containing the coordinate system transformation matrices. For each transformation matrix, the top 3x3 matrix partition contains the transformation matrix [T] between the global origin and a Cartesian coordinate system aligned with the current coordinate system. The fourth row in the transformation matrix, [o], contains the offset between the global origin and the current coordinate system origin. For example, if [n] a 3x1 vector containing node coordinates in the global Cartesian system and [l] is a 3x1 vector containing node coordinates in the local (Cartesian) system, the equation to transform from global to local coordinates is represented as

$$[l]=[T][n-o']$$

After transforming global Cartesian to local Cartesian, the local coordinates can then be transformed to cylindrical or spherical as necessary.

Nodes: IMAT_NODE Class

The IMAT_NODE class contains node definitions. This class has the properties listed in the table below.

Property Name	Description
`.id`	Node labels ([nx1] numeric vector)
`.cs`	Node coordinate systems ([nx3] numeric vector)
`.color`	Node colors ([nx1] numeric vector)
`.coord`	Node coordinates ([nx3] double vector)

The `id` property is an nx1 numeric vector containing the node labels. These labels should be positive and unique.

The `cs` property is an nx3 numeric matrix containing the node coordinate system labels. The number of rows in this matrix must match the length of the `id` proprety. Column 1 contains the reference (also called export or definition) coordinate system labels. Column 2 contains the displacement coordinate system labels. Both of these coordinate systems are used by I-deas. Column 3 contains coordinate system information for use in MATLAB. These coordinate system labels refer to the coordinate system in which the nodal coordinates in the IMAT_NODE object are stored.

The `color` property is an nx1 numeric vector containing the node colors. The colors are positive numbers representing colors using the color scheme defined by I-deas. This scheme is shown in the table below. The length of the `color` vector must be the same as the length of the `id` field.

Number	Color
0	Black
1	Blue
2	Gray Blue
3	Light Blue
4	Cyan
5	Dark Olive
6	Dark Green
7	Green
8	Yellow
9	Golden Orange
10	Orange
11	Red
12	Magenta
13	Light Magenta
14	Pink
15	White
16+	User-defined

The `coord` property in an nx3 double matrix containing the node coordinates. Each row contains the coordinates for a particular node in the coordinate system specified in column 3 of the `cs` property. If the coordinate system is Cartesian, the coordinates are specified in X-, Y-, and Z-order. If the coordinate system is cylindrical, coordinates are specified in R-, Theta- (degrees), and Z-order. If the coordinate system is spherical, coordinates are specified in R-, Phi-, and Theta-order (degrees). The coordinate system magnitude is assumed to be in the current units system. Angles are stored in units of radians. Please see the I-deas documentation for more information on coordinate system definitions. All of the coordinate systems specified in the `coord` property must be contained in the coordinate system object.

Elements: IMAT_ELEM Class

The IMAT_ELEM class contains all of the relevant element data. The properties and their descriptions are shown in the table below. The contents of each of these properties are described below. Some of the properties in the element structure are not used by IMAT, but are provided so that no data is lost in a round trip between I-deas and MATLAB.

Property Name	Description
`.id`	Element labels ([nx1] numeric vector)
`.type`	Element type ([nx1] numeric vector)
`.color`	Element colors ([nx1] numeric vector)
`.prop`	Element physical and material properties ([nx2] numeric vector)
`.beamdata`	Element beam orientation and cross-sections ([nx3] numeric vector)
`.conn`	Traceline connectivity ({nx1} cell array of numeric row vectors)

The `id` property is an nx1 numeric vector containing the element labels. These labels should be positive and unique.

The `type` property is an nx1 numeric vector containing the element type descriptions. The element types correspond to those used by Nastran. Each type of element has a unique number, as shown in the table below. The length of the `type` vector must be the same as the length of the `id` property. To get a more complete list of element types and classes, use the IMAT_ELEM method get_elemtypes, as shown below. It will return a structure containing information describing the element types.

```
>> et = imat_elem.get_elemtypes()

et =

      type: [1x48 double]
      desc: {1x48 cell}
     nnode: [1x48 double]
nnoderange: {48x1 cell}
  nnodemap: [903x2 double]
     class: [1x903 double]
 classdata: {20x3 cell}
```

Class	Type Number	Element Type
1-D		
	34	CBAR

	100	CBARA
	2	CBEAM
	69	CBEND
	10	CONROD
	92	CONROD (nonlinear)
	1	CROD
	3	CTUBE
	24	CVISC
2-D		
	271	CPLSTN3
	272	CPLSTN4
	273	CPLSTN6
	274	CPLSTN8
	62	CQDMEM
	127	CQUAD
	33	CQUAD4
	64	CQUAD8
	82	CQUADR
	228	CQUADR (newer formulation)
	232	CQUADR (newer formulation)
	243	CQUADX4
	245	CQUADX8
	4	CSHEAR
	74	CTRIA3
	75	CTRIA6
	70	CTRIAR
	227	CTRIAR (newer formulation)
	233	CTRIAR (newer formulation)
	133	CTRIAX
	53	CTRIAX6
	242	CTRAX3
	244	CTRAX6

3-D		
	`67`	CHEXA
	`68`	CPENTA
	`255`	CPYRAM
	`39`	CTETRA
	`76`	CHEXPR
	`77`	CPENPR
	`79`	CPYRAMPR
	`78`	CTETPR
	`93`	CHEXA (nonlinear)
	`91`	CPENTA (nonlinear)
	`85`	CTETRA (nonlinear)
Other		
	`102`	CBUSH
	`40`	CBUSH1D
	`20`	CDAMP1
	`21`	CDAMP2
	`11`	CELAS1
	`12`	CELAS2
	`38`	CGAP
	`86`	CGAP (nonlinear)
	`29`	CONM1
	`30`	CONM2
	`901`	RBAR (numbering is IMAT-only)
	`902`	RBE2 (numbering is IMAT-only)
	`903`	RBE3 (numbering is IMAT-only)

The `color` property is an nx1 numeric vector containing the element colors. The colors are positive numbers representing colors using the color scheme defined by I-deas. The length of the `color` vector must be the same as the length of the `id` field.

The `prop` property is an nx2 numeric vector containing the element physical and material property IDs. This property is not used by IMAT, but is provided for completeness. The first column contains physical property ID numbers, and the second contains material property ID numbers. These are positive numbers. The number of rows in this matrix must be the same as the length of the `id` property.

The `beamdata` property is an nx3 numeric vector containing beam element information. This property is not used by IMAT, but is provided for completeness. The first column contains beam orientation node numbers. The second column contains the fore-end beam cross section number, and the third column contains the aft-end beam cross section number. The `.beamdata` property contains positive numbers for beam elements. Rows that correspond to non-beam elements contain zeros. The number of rows in this matrix must be the same as the length of the `id` field.

The `conn` property is an nx1 cell array of numeric row vectors containing the element connectivity. The element connectivity contains a series of node IDs corresponding to the number of nodes necessary to define that element. The nodes specified in the element should all be contained in the node object. **Please note that the node ordering for parabolic elements is Nastran-centric, where the corner nodes are specified, followed by the midside nodes.**

Tracelines: IMAT_TL Class

The IMAT_TL class contains all of the relevant traceline data. The properties and their descriptions are shown in the table below. The contents of each of these properties are described below.

Property Name	Description
`.id`	Traceline labels ([nx1] numeric vector)
`.color`	Traceline colors ([nx1] numeric vector)
`.desc`	Traceline descriptions ({nx1} cell array of strings)
`.conn`	Traceline connectivity ({nx1} cell array of numeric row vectors)

The `id` property is an nx1 numeric vector containing the traceline labels. These labels should be positive and unique.

The `color` property is an nx1 numeric vector containing the traceline colors. The colors are positive numbers representing colors using the color scheme defined by I-deas. The length of the `color` vector must be the same as the length of the `id` field.

The `desc` property is an nx1 cell array of strings containing the traceline descriptions. Each description is a string. The description strings can be varying lengths (hence the need for a cell array). The length of the cell array must be the same as the length of the `id` property.

The `conn` property is an nx1 cell array of numeric row vectors containing the traceline connectivity. The traceline connectivity contains a series of node IDs. The traceline can contain breaks, which are specified by a node number of 0. When a break is encountered, the traceline will stop being drawn at the node before the break and will continue starting with the node after the break. You must have between 2 and 250 nodes in the traceline (breaks are counted as a node), and you can have multiple breaks. The nodes specified in the traceline should all be contained in the node object.

Group Data Attributes

Group Data Attributes

IMAT Home Page

- id
- name
- data
- type

---

id

The ID is a numeric scalar specifying the group ID. It is not used by IMAT, but is useful for round-tripping to a Universal file.

---

name

The NAME a string containing the group name. It can be up to 80 characters in length.

---

data

The DATA property is an Nx2 matrix containing the entity IDs (column 1) and their corresponding type (column 2). Supported entity types are listed in the table below.

Entity Type	Description
0	Unknown
1	Coordinate System
7	Node
8	Element
27	Traceline

TYPE is a special property that returns enumerations for each of these entity types.

---

type

TYPE is a special dependent property that returns an enumeration for each of the entity types. We recommend that you use this rather than the entity type numbers directly. It will better self-document your code, and will also make your code robust to any potential changes in the entity type numbers. The following code snippet shows how to use this. Note that the static method get_type_data also returns this information in a different format with some additional information.

```
>> g=imat_group;
>> g.type

ans =
     gUnknown: 0
        gCSys: 1
        gNode: 7
     gElement: 8
   gTraceline: 27

>> g.type.gCSys     % Use this nomenclature when referencing entity types

ans =

     1
```

FCN Data Attributes

FCN Data Attributes

The attributes listed on this page are specific to FCN objects. In addition to these, FCN attributes also have all of the imat_fn attributes.

IMAT Home Page

- Children
- Filename
- InterpolationType
- Name
- Parent
- RSpectQ
- UID

Children

This attribute is NOT USED.

Cell array of strings containing a list of child functions.

Filename

String containing the filename from which the function was imported.

InterpolationType

A string specifying the interpolation type. This is stored as a numeric equivalent in the IMAT_FN UserValue4 property.

Numeric Identifier	Data Type
0	`'LinLin'`
1	`'LinLog'`
2	`'LogLin'`
3	`'LogLog'`

Name

String containing the function name.

Parent

Cell array of strings containing a list of parent functions. These are the functions that were used to generate the current function. For example, if two functions were added together, both of these functions will be listed as parents of the resulting function. The string is a combination of function Filename and UID separated by the pipe character (|).

RSpectQ

A numeric value containing the Q value used in Response Spectrum analysis for this function. It is stored in the IMAT_FN UserValue3 property.

UID

Read-only attribute. A 64-bit integer containing a unique identifier. The UID must be unique for all functions in a given variable. The FCN methods ensure that this is always true.

FCN Data Attributes

FCN Data Attributes

The attributes listed on this page are specific to FCN objects. In addition to these, FCN attributes also have all of the imat_fn attributes.

IMAT Home Page

- Children
- Filename
- InterpolationType
- Name
- Parent
- RSpectQ
- UID

Children

This attribute is NOT USED.

Cell array of strings containing a list of child functions.

Filename

String containing the filename from which the function was imported.

InterpolationType

A string specifying the interpolation type. This is stored as a numeric equivalent in the IMAT_FN UserValue4 property.

Numeric Identifier	Data Type
0	`'LinLin'`
1	`'LinLog'`
2	`'LogLin'`
3	`'LogLog'`

Name

String containing the function name.

Parent

Cell array of strings containing a list of parent functions. These are the functions that were used to generate the current function. For example, if two functions were added together, both of these functions will be listed as parents of the resulting function. The string is a combination of function Filename and UID separated by the pipe character (|).

RSpectQ

A numeric value containing the Q value used in Response Spectrum analysis for this function. It is stored in the IMAT_FN User-Value3 property.

UID

Read-only attribute. A 64-bit integer containing a unique identifier. The UID must be unique for all functions in a given variable. The FCN methods ensure that this is always true.

Function and Method Index

This index provides links to the functions and methods available in IMAT.
