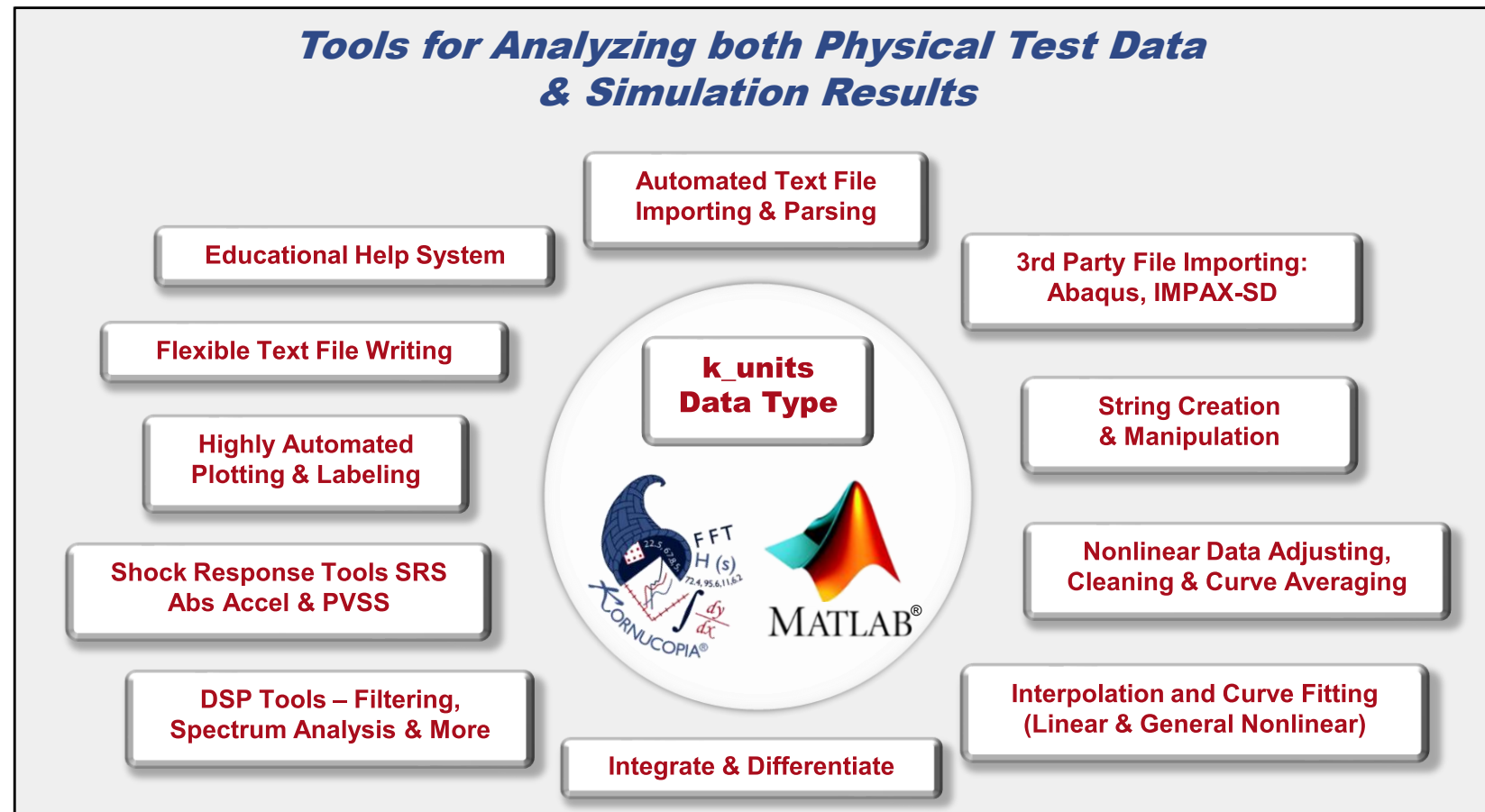


# ***Bodie Technology, Inc***

***Work Smarter, Work Faster™***

## **Kornucopia® ML™ Quick Start**



# Course Content

- Kornucopia at a Glance
- Getting Help/Guidance and Finding Functions & Examples
- An Introduction to the *k\_units* Data Type
  - A variety of topics related to the syntax of using *k\_units* data type are discussed.
- More on Kornucopia Syntax
  - ADV parameter
  - Flexible XY data input syntax
  - Smooth, Curley, & Square Braces plus , ; and : syntax
- Importing & Exporting Data Files with Kornucopia Functions
- Plotting Curves
- *Kornucopia Results Workbook*™ via Tabbed Plots
- Workshop Examples of Complete Kornucopia Workflows

# Legal Notices

- The information in this document is subject to change without notice and should not be construed as a commitment by Bodie Technology, Inc.
- Bodie Technology, Inc. assumes no responsibility for any errors that may appear in this document.
- Copyright © 2004 – 2022 Bodie Technology, Inc.
  - Printed in U.S.A. All rights reserved.
  - Printing: June 2022, rev 01.
- Kornucopia is a registered trademark of Bodie Technology, Inc.  
The following are trademarks of Bodie Technology, Inc:  
The Kornucopia logo, “Faster data analysis with greater understanding”, “Work Smarter – Work Faster”, “Smart-Tools for Analysis”, and “Smart-Tools for Analyzing Noisy and Challenging Problems”.  
The “pot of gold” logo is a service mark of Bodie Technology, Inc.
  - All other brand or product names are trademarks or registered trademarks of their respective companies or organizations.
- This training will be recorded, and an edited version of the video will be available on our website at a later date.



# Bodie Technology, Inc.

We specialize in solving complex problems in nonlinear mechanics by employing a proven mix of computational and testing knowledge in novel ways.

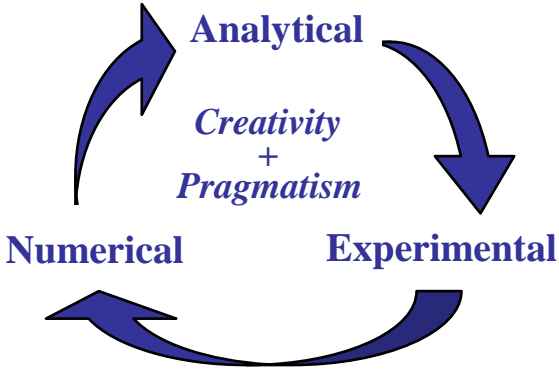


- **Kornucopia® ML™ Software**
- **Customized Training**
- **Expert Consulting**

***Faster data analysis with greater understanding™***

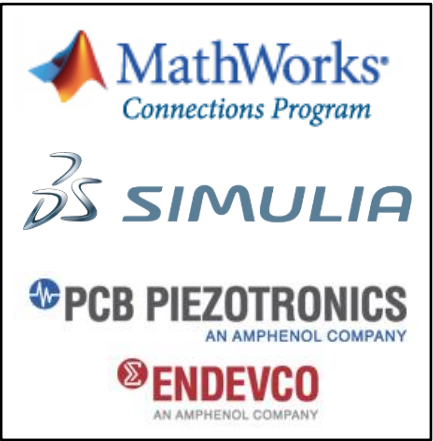
*“Bodie Technology provides engineers with excellent software, training, and consulting resources to help analyze complex nonlinear mechanics problems, especially those involving problematic or noisy datasets.”*

Steve Levine, Chief Strategy Officer, SIMULIA



[www.BodieTech.com](http://www.BodieTech.com)  
[info@BodieTech.com](mailto:info@BodieTech.com)

## Alliances



***Bodie Technology, Inc***  
***Work Smarter, Work Faster™***

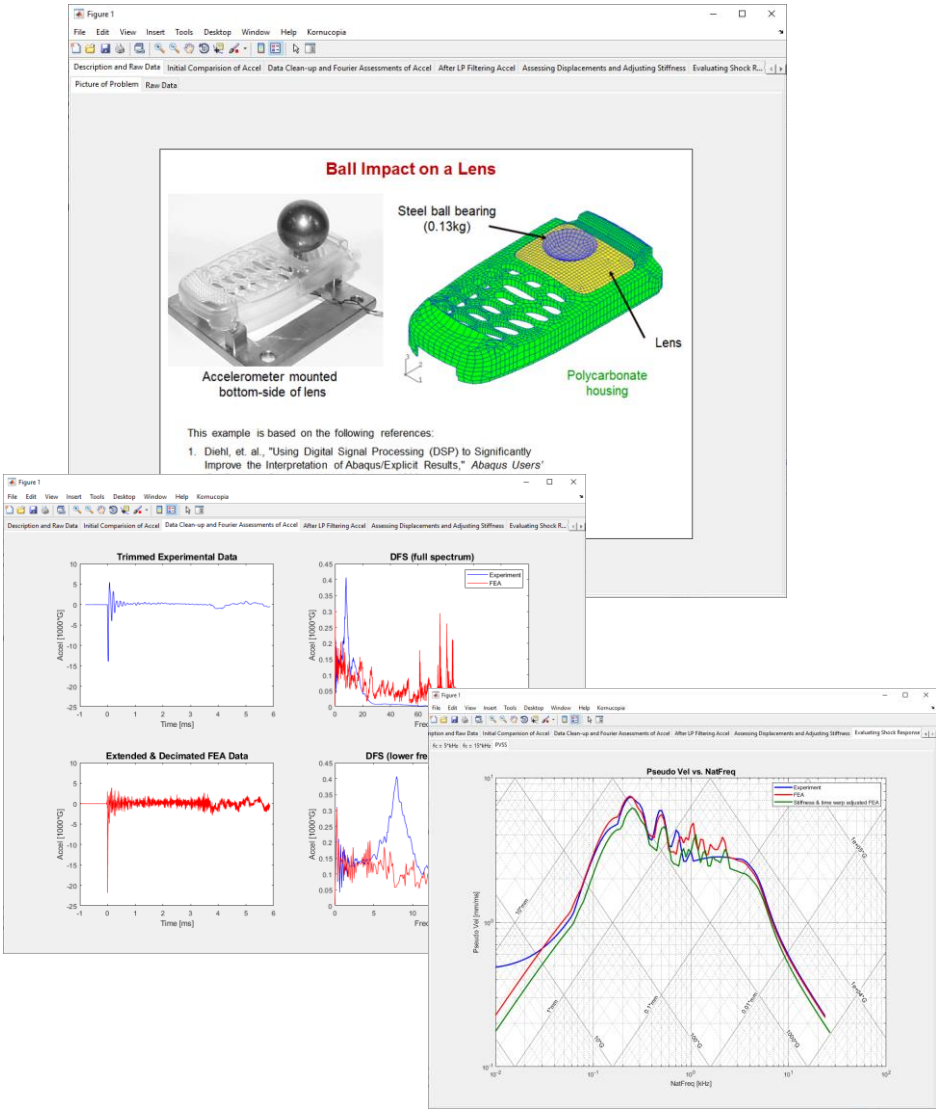
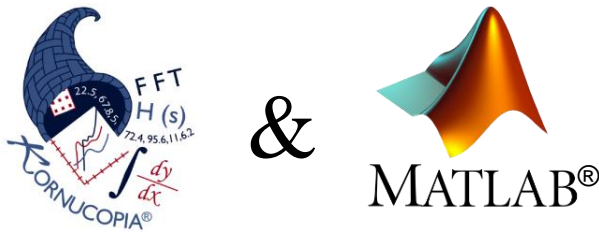


## **Kornucopia at a Glance**

# Kornucopia® ML™

## Focus of the Software

- Work naturally with ALL our information
  - Data, Units, Images and Meta-Info
- Data analysis of physical testing & transient FEA
  - Flexibility, Accuracy, & Confidence
- Tools to process messy and noisy data
  - Solve real-world problems
- Efficient plotting with automatic labeling
  - Easily manage tons of plots to clearly communicate your results



	Time [msec]	A3 [m/s^2]	U3 [mm]	NE11_SNEG [%]
1259	0.9982	-2.6053e+05	-17.3595	0.0731
1260	0.9990	-2.6022e+05	-17.3684	0.0774
1261	0.9998	-2.1613e+05	-17.3775	0.0815
1262	1.0006	-1.3953e+05	-17.3867	0.0856

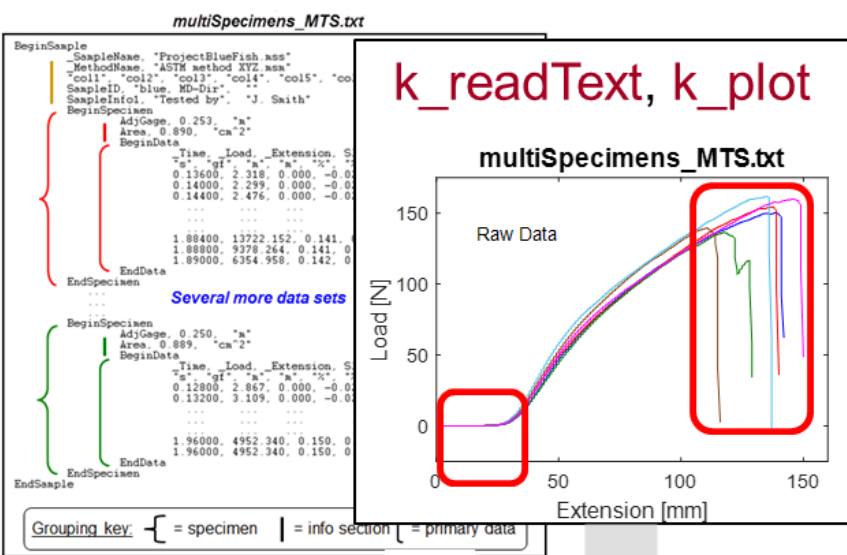
```
Vo = -20e3*mm/s; % Initial vel

v = k_integrate(odbReg('Time'), odbReg('A3'), Vo);
d = k_integrate(odbReg('Time'), v, 0*mm);
```



# Kornucopia® ML™ - Typical Usage Scenarios

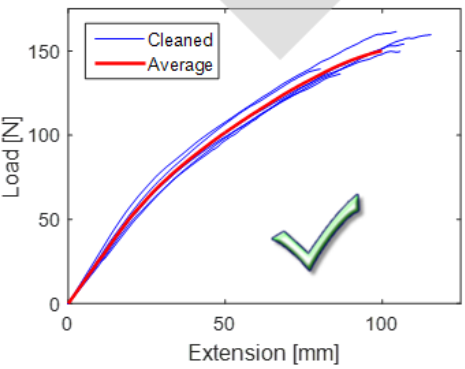
Readily Analyze Messy Data from Tests and Simulations



trim, tweak, shift  
averageXY, fitFunc

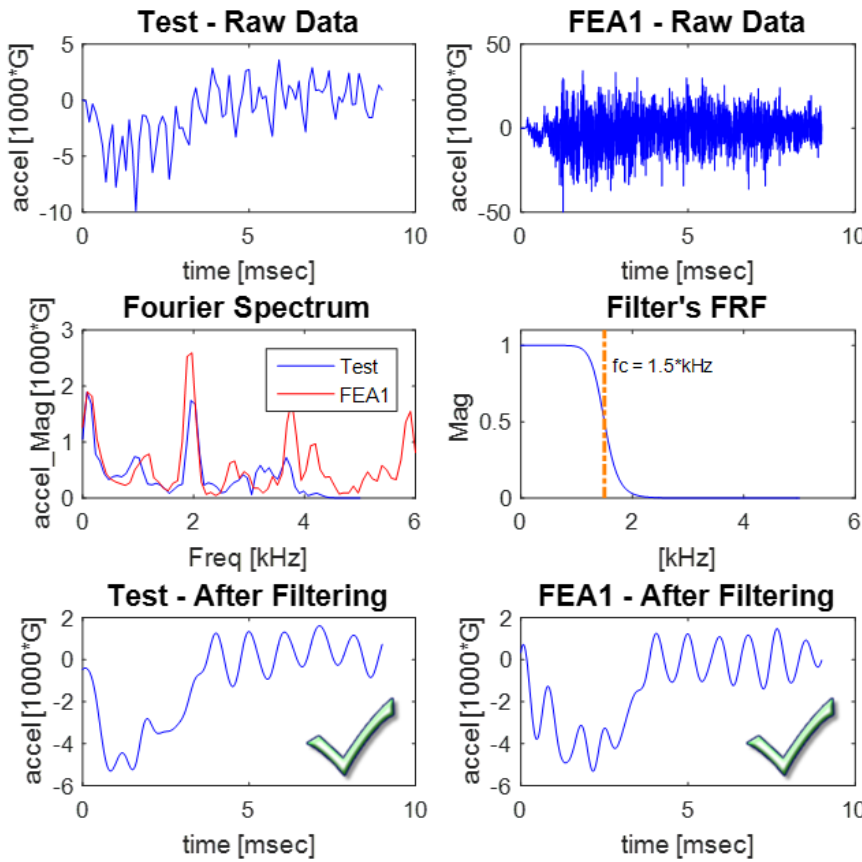
$$F(x, [K, C1]) = K \cdot x + C1 \cdot (x.^2)$$
$$Rsqr = 0.9975$$

K	C1
[N/mm]	[N/(mm^2)]
2.6509	-0.011737



Analysis and Filtering of Noisy Data to Correlate Test and Simulation

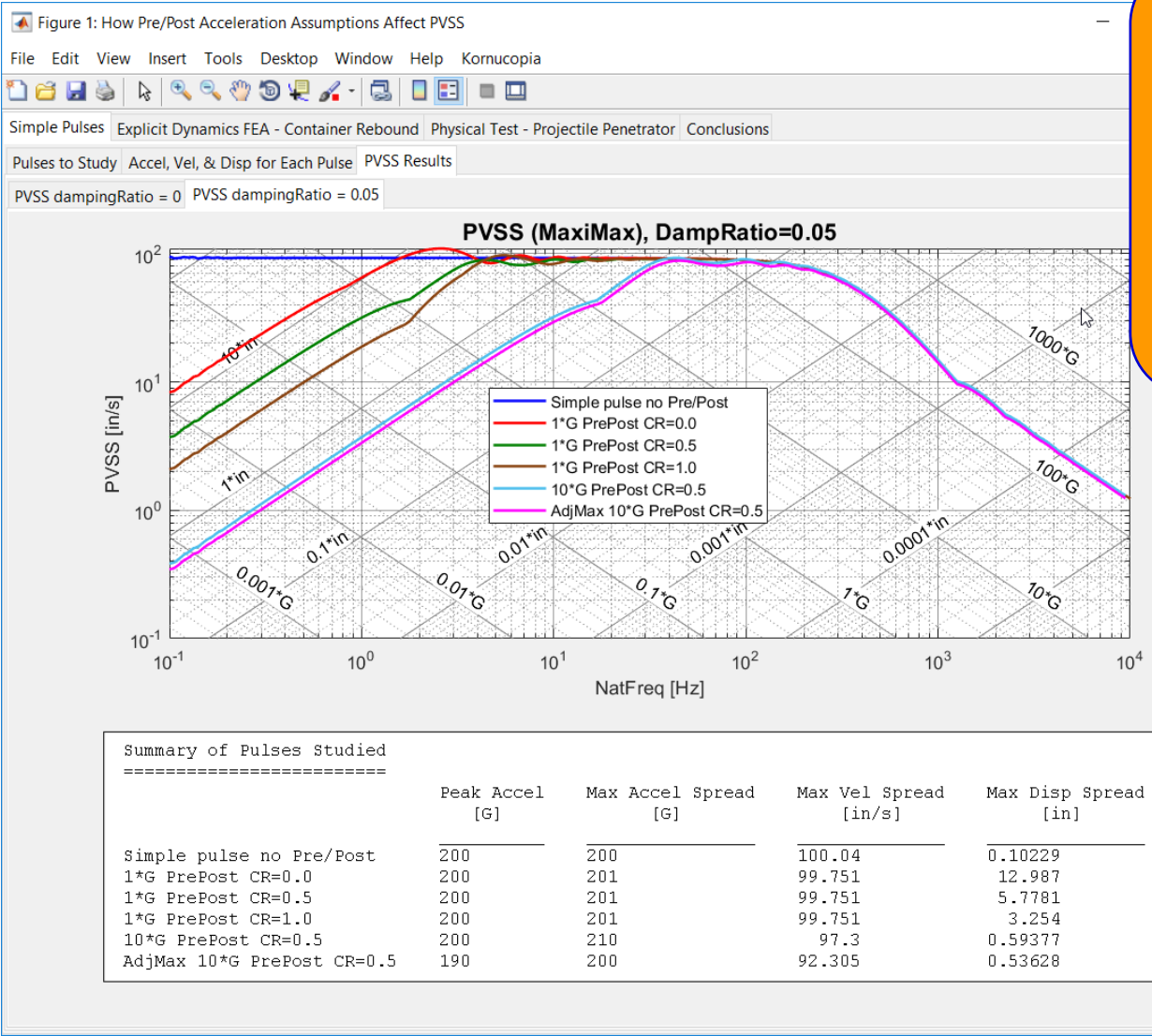
From a Mess to Success!





Results Workbook - Easily Create and Manage Large Amounts of Data

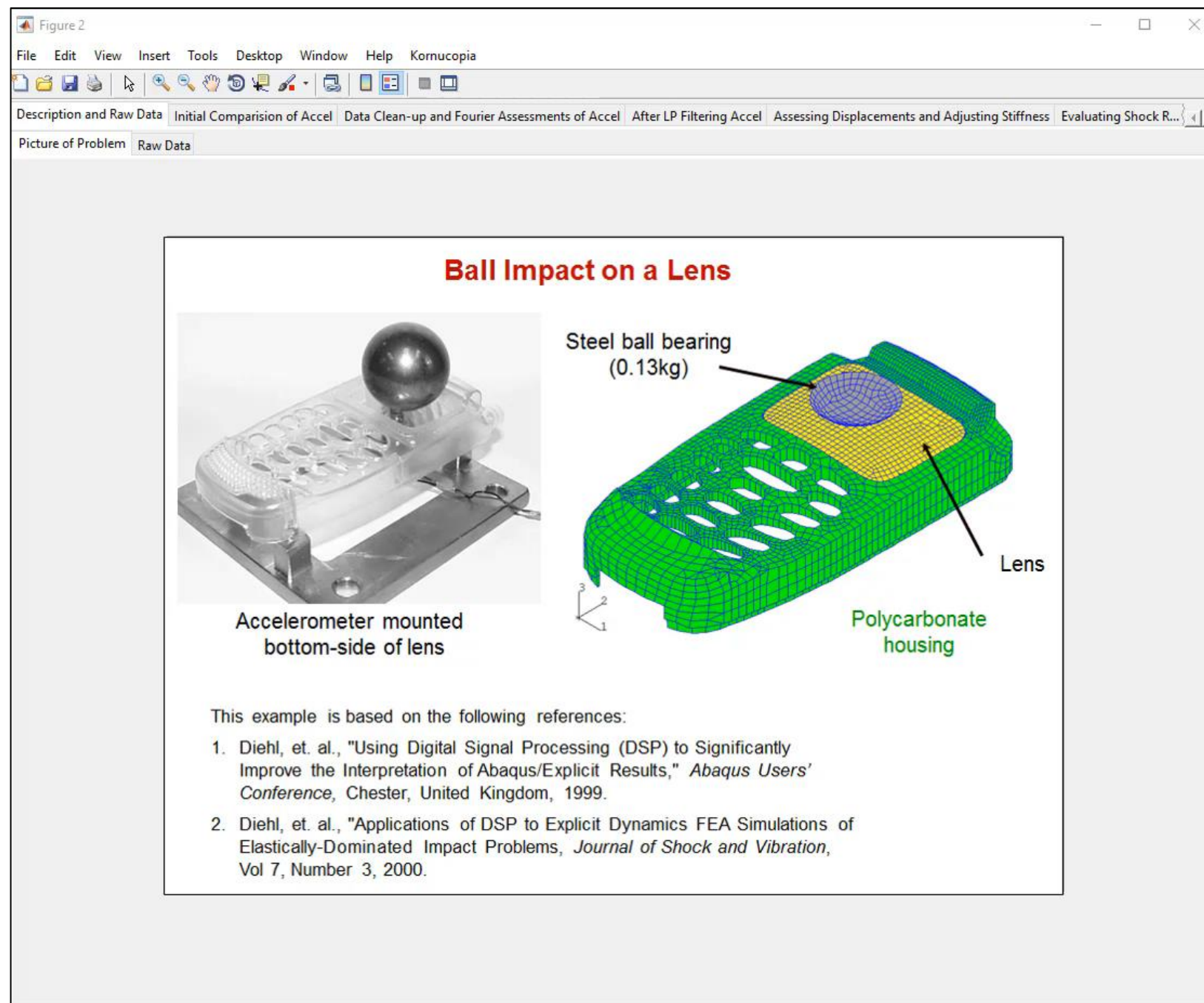
1. Easily create tabbed figures
2. Results presented in orderly manner
3. Quickly compare results with Kornucopia's *pop-out* tools
4. Easily retrieve from figure any curves or tables, including their original meta-data
5. Attach virtually any other files to the figure for access later
6. Visual, interactive, pseudo database & archive
7. Easily generate PDF of all the tabs to share with non-MATLAB users



This 1 figure window could contain as many as 100 plots or more within its nested tabs!







# ***Kornucopia ML – For use w/ MATLAB®***

## ***The Horn of Plenty***

**Tools for both  
Physical Test Data &  
Simulation Results**

**Automated Text File  
Importing & Parsing**

**Educational Help  
System**

**3<sup>rd</sup> Party File Importing:  
Abaqus, IMPAX-SD**

**Flexible Text  
File Writing**

**k\_units  
Data Type**

**Nonlinear Data Adjusting,  
Cleaning & Curve Averaging**

**Highly Automated  
Plotting & Labeling**

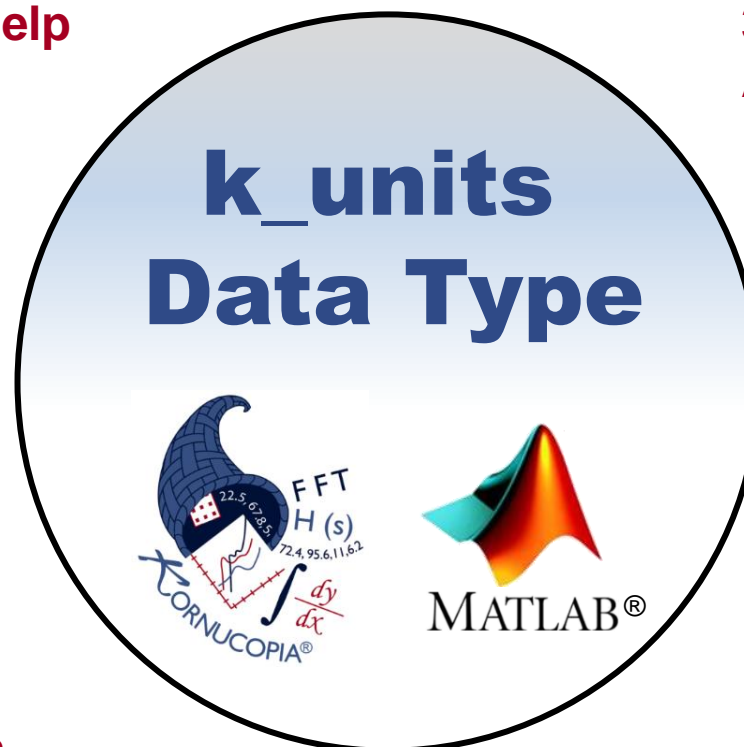
**Interpolation and Curve Fitting  
(Linear & General Nonlinear)**

**Shock Response Tools  
SRS Abs Accel & PVSS**

**String Creation & Manipulation**

**Integrate & Differentiate**

**DSP Tools – Filtering,  
Spectrum Analysis & More**



# Kornucopia ML – Powerful Features that Work Naturally in MATLAB

## 80+ Overloaded MATLAB Functions

### Arithmetic math operators and functions

- +, -, \*, /, \, ^, .\*, ./, .\, .^
- sqrt, sum, prod, diff, ceil, floor, round

### Relational operators

- >, >=, <, <=, ==, ~=

### Trigonometry

- sin, cos, tan, cot, sec, csc
- asin, acos, atan, acot, asec, acsc
- sinh, cosh, tanh, coth, sech, csch
- asinh, acosh, atanh, acoth, asech, acsch

### Logs and Exponents

- log, log10, exp

### Complex numbers and angles

- abs, real, imag, conj, angle, unwrap

### Descriptive statistics

- mean, median, std, min, max

### Reshaping arrays

- cat, horzcat, vertcat
- flipud, fliplr, flipdim, flip

### Is something

- isempty, iscolumn, isrow, isvector, isscalar
- isreal, isfinite, isinf, isnan

### Matrix operations

- transpose, ctranspose
- cross, dot

### Display functions

- display, disp

### Syntax operators and functions

- subsref, subsasgn - functions for overloading indexing with ( ), and { } parentheses, as well as using "." notation.

All these  
MATLAB  
functions  
understand  
the **k\_units**  
data type

## Kornucopia® ML™ Help System

- + All About the k\_units Data Type
- + Details on Kornucopia Features
- Functions
  - 📄 Alphabetical Listing of All Functions
  - Function Categories
    - + Help and Examples
    - + Licensing Related Functions
    - + Comparing Curves and Variables
    - + Data Adjusting and Averaging
    - + Derivatives and Integrals
    - + DSP - Filtering, Smoothing, Spectrum Analysis
    - + Fitting and Interpolation
    - + Interacting with Other Software
    - + k\_units Related Functions
    - + Misc and Cool Stuff
    - + Plotting and Displaying Results
    - + Principal and Mises Values
    - + Reading and Writing Files
    - + Row Vector Operations
    - + Shock Response Spectra and Pulses
    - + Strain Gages
    - + String (Char) Creation/Manipulation
    - + Tables
- + Examples Library

125+ native  
Kornucopia ML  
functions



# Helpful Kornucopia Videos

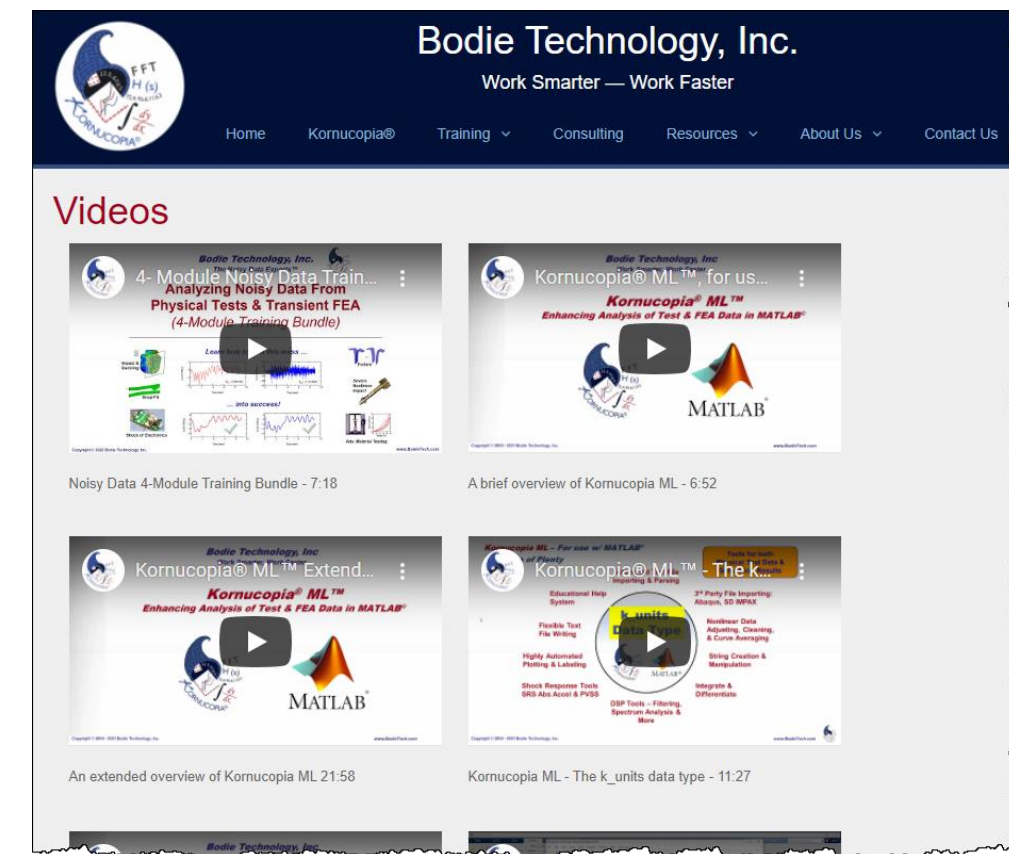
<https://bodietech.com/index.php/resources/videos>

- The following videos that reside on the link above provide overviews of Kornucopia's capabilities:

1. An extended overview of Kornucopia ML - 21:58
2. Kornucopia ML - The k\_units data type - 11:27
3. Kornucopia ML - Analyzing Noisy Data - 9:41
4. If you use the SD VIDAS system for data acquisition please additionally watch these two videos:

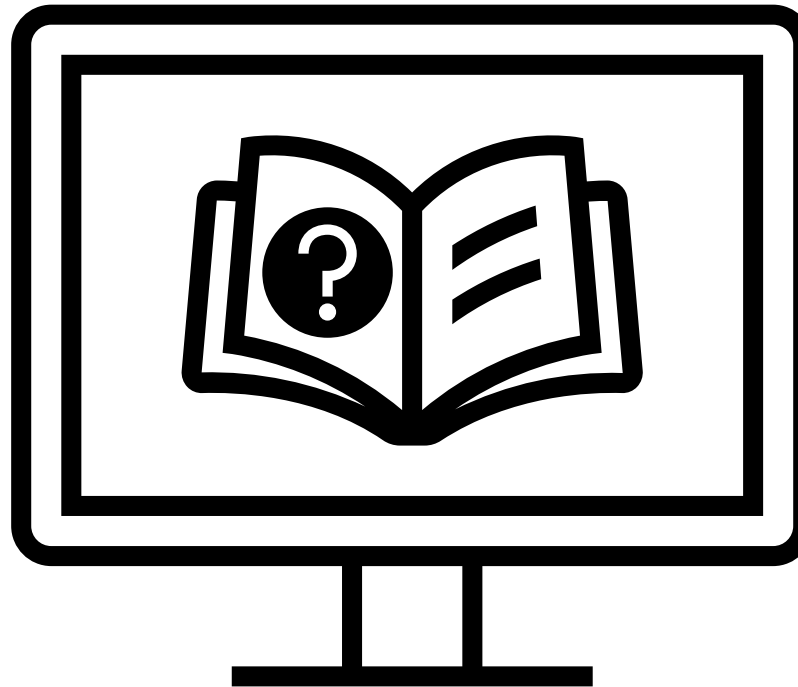
- Intro to Processing of VIDAS Files w/ Kornucopia ML - 3:56
- General Processing of VIDAS Files w/ Kornucopia ML - 12:17

- Recommendation** – You should watch these videos before continuing with this training course.



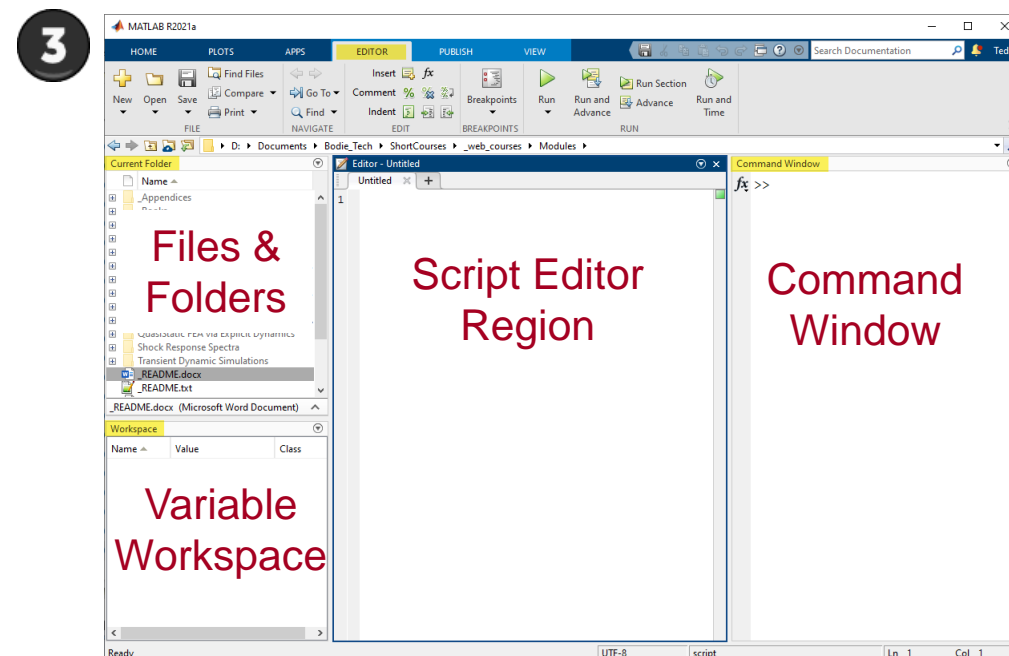
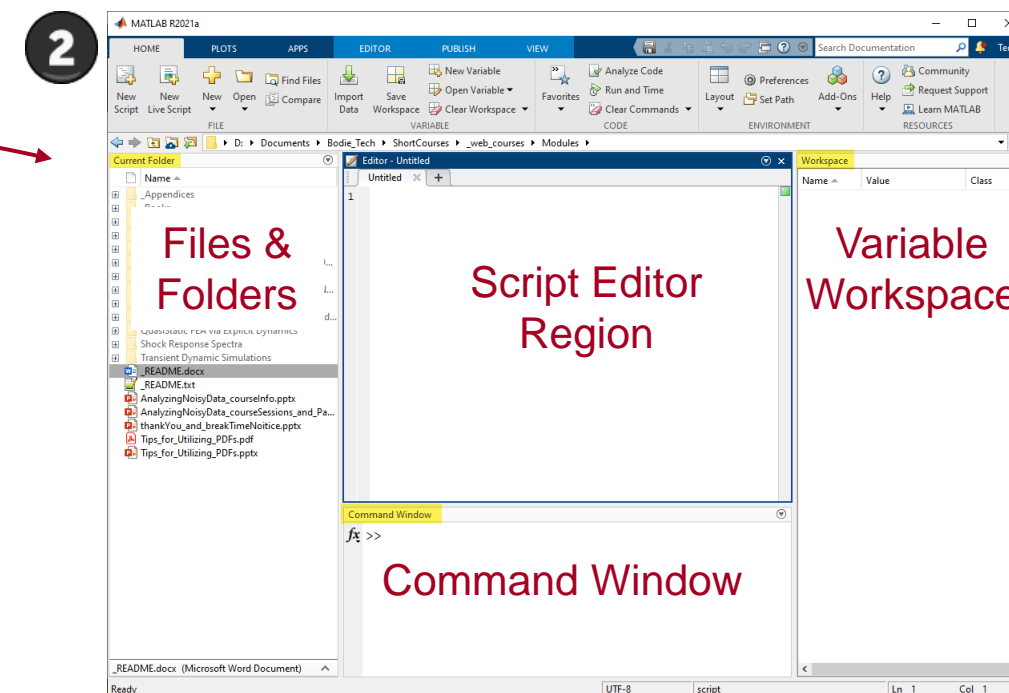
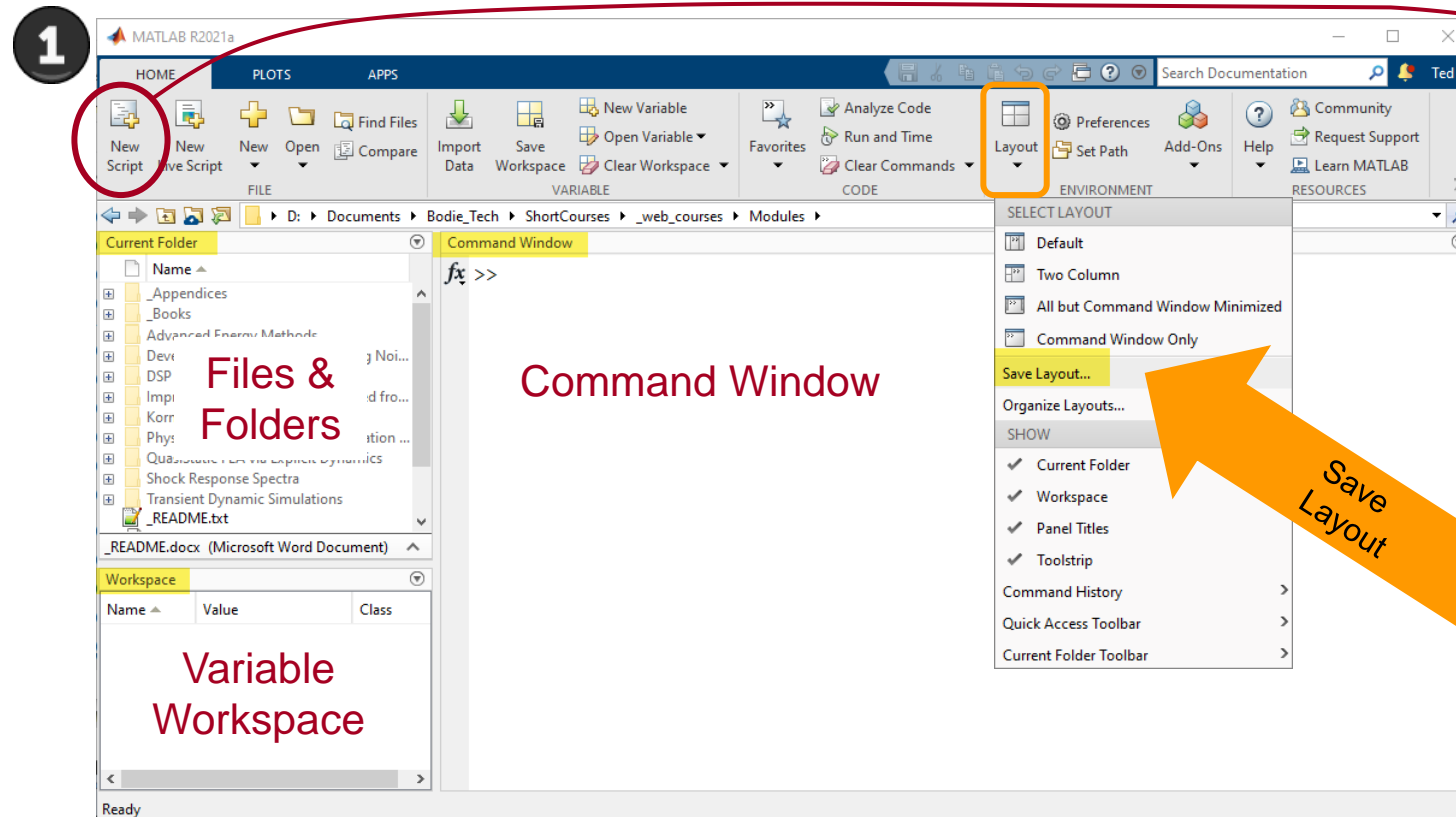


## Getting Help/Guidance and Finding Functions & Examples





# The MATLAB Environment



1. The MATLAB environment before a script is open.
2. MATLAB default layout with the traditional script editor open.
3. Recommended Kornucopia layout for MATLAB environment.
  - Click and drag regions from (2) to get preferred layout (3), then save the layout from the HOME tab.



# The Kornucopia HTML Help System

- Utilize any of the following to access help for Kornucopia
  - Click on *Kornucopia ML Help* icon on your desktop
  - In MATLAB command window, type **k\_2help**
  - Use *F1* key with cursor over a Kornucopia function

3

ballLensImpact.m

```
156
157 % Decimating the FEA data
158 decFactor = fs_regFea / fs_Exp;
159 decFea = k_decimate(extendedFea, [], decFactor);
160 decFea.Props.
161 fs_decFea = k
162
163 % Plotting th
164 currentTabH =
165 k_figTabsClea
166
167 txt = {'No ex
168 'compa
169 k_displayOnFi
170 'parent',
171 'box', 'o
172 'alignHor
173
174 subplot(2,2,2
175 k_plot(regFea
176 'depend',
177 'title', 'user', 'Regularized FEA data'}}
```

Hit **F1** key on k\_decimate

k\_decimate - MATLAB File Help

View code for k\_decimate

### k\_decimate

k\_decimate decimates the input data including proper antialiasing protection ('on' by default). The ADV options provide a variety of choices to control aspects of the algorithms used in the decimation process.

Calling Syntax Options

```
[xNew, yNew] = k_decimate(xData, yData, decimateFactor, ADV)
yNew = k_decimate([], yData, decimateFactor, ADV)
xyNew = k_decimate(xyData, [], decimateFactor, ADV)
```

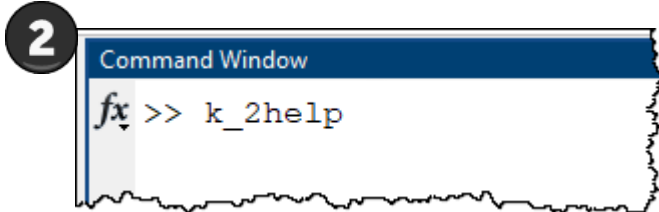
[CLICK FOR DETAILED HELP PAGE](#)

Kornucopia® software is Copyright 2003 - 2021, Bodie Tech. All rights reserved. [www.BodieTech.com](http://www.BodieTech.com)

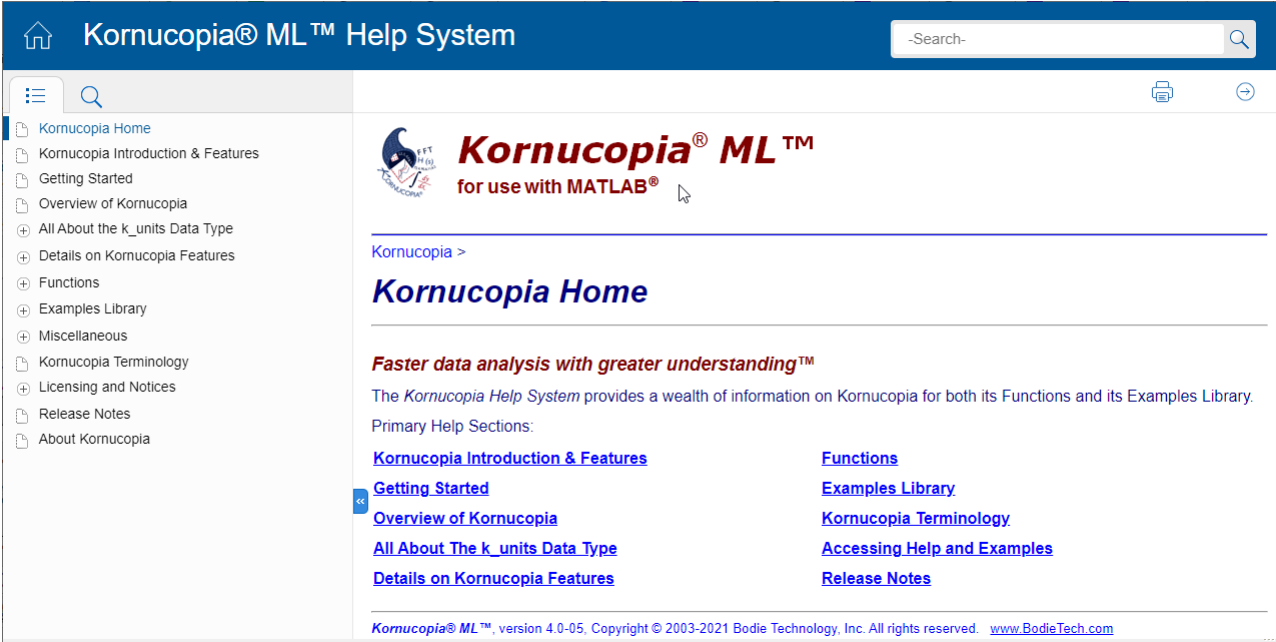
[Open Help Browser](#)

Click here for full help page

Note: Kornucopia Help System is separate from the MATLAB Help System.



Only native MATLAB help is available here



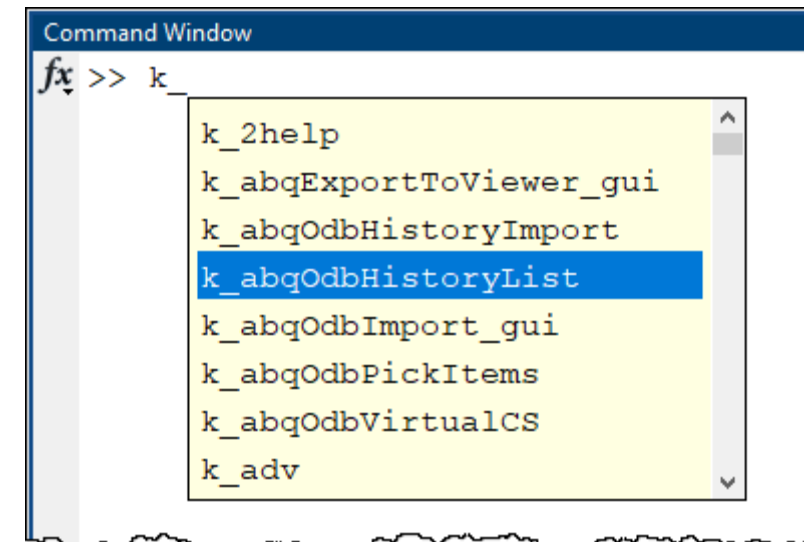
## Finding Kornucopia Functions

- **Live in MATLAB**

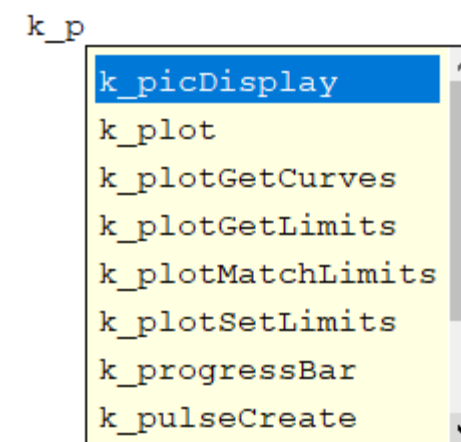
- From command window, type **k\_** then hit the tab key.
- From script editor (MATLAB R2021a and older)
  - type **k\_** then hit the tab key to get selectable alphabetical list of functions.
- From script editor (MATLAB R2021b & R2022a)
  - The list is NOT alphabetical, it is based on AI logic and what you most recently utilized.
  - In these versions of MATLAB use the command window and type **k\_** then hit the tab key to get the full alphabetical selectable list of function.

- **From Kornucopia Help System**

- Listed alphabetically and also by categories.



**Over 125  
functions in  
the list**



**Functions named so that  
they naturally self-group:**

k\_abq\* - Abaqus functions  
k\_plot\* - Plotting functions  
k\_pulse\* - related to pulses  
...

# Detailed Help Pages for Every Function

The screenshot displays the Kornucopia® ML™ Help System interface. The top navigation bar includes a home icon, the title "Kornucopia® ML™ Help System", and a search bar. A left sidebar contains a tree view of the help system, with "Functions" expanded and "Function Categories" selected. The "DSP - Filtering, Smoothing, Spectrum Analysis" category is highlighted, and the "k\_filter" function is selected. The main content area shows the "k\_filter" function page, which includes a breadcrumb trail, a description of the function, a list of advanced DSP features, and a "Calling Syntax" section with three primary options. A "Quick links on this help page" section provides links to related topics, and a "Related links" section provides additional resources.

**Kornucopia® ML™ Help System**

Search:

Kornucopia > Functions > Function Categories > DSP - Filtering, Smoothing, Spectrum Analysis

## k\_filter

This function performs single or bi-directional filtering using cascaded digital filter coefficients, typically created by the function [k\\_clIR](#) or [k\\_cFIR](#).

The function has several advanced DSP features and options that include:

1. Robust cascade filtering approach. This approach is more numerically stable than traditional non-cascaded filtering algorithms when processing massively oversampled data that is commonly output from explicit dynamics simulations and similar.
2. Easily controlled single or bi-directional (zero phase) filtering via the `nPasses` input argument.
3. Advanced end-effect distortion minimization algorithms that are controlled via ADV options. The default prediction-based algorithms typically perform well over a large array of scenarios, but you always have the option of using other ADV option settings as needed.

**Quick links on this help page**

- [Calling Syntax](#)
- [Optional ADV Arguments](#)
- [Simple Depictions of Function Use](#)
- [More Examples of Function Usage](#)

**Related links**

- [Working with Units and k\\_units Data Types](#)
- [Overview of Kornucopia ADV Parameter](#)
- [Flexible XY Data Input Syntax](#)
- [Plotting and Viewing Data](#)
- [Kornucopia Syntax](#) and [Kornucopia Terminology](#)

### Calling Syntax

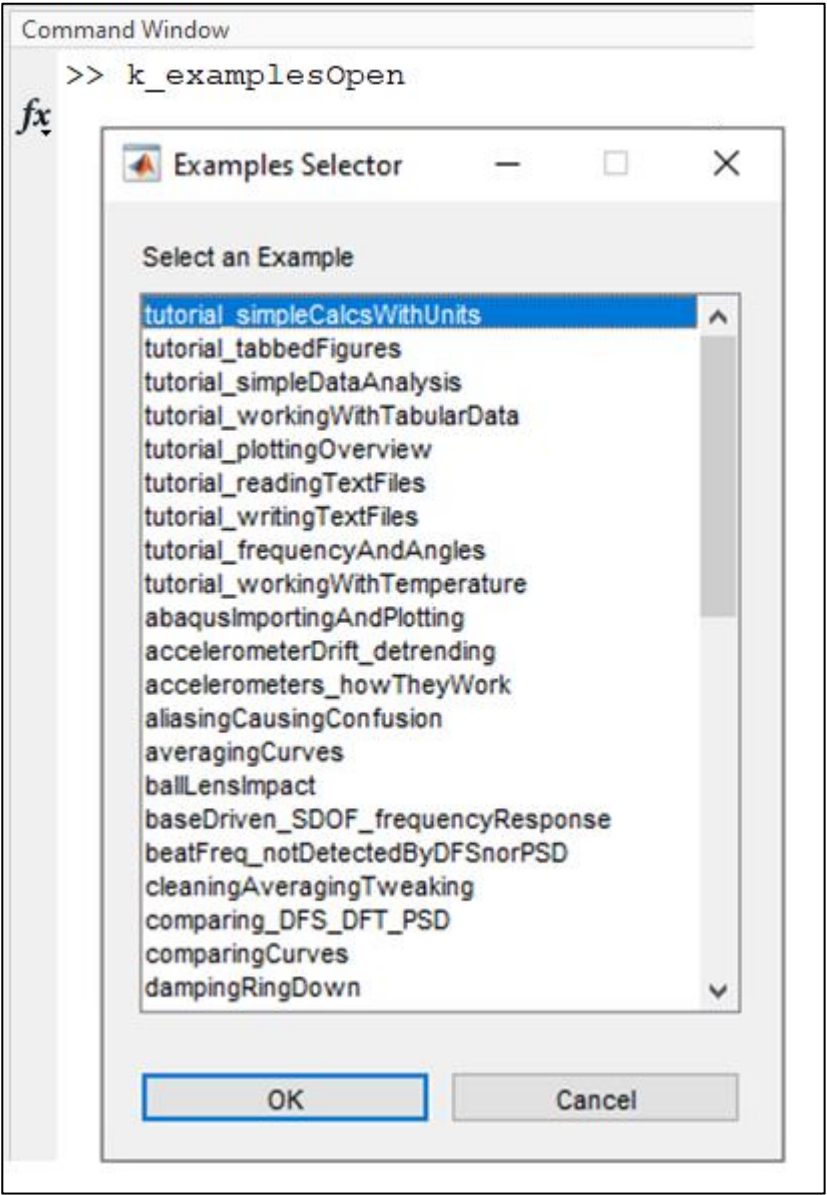
The function has three primary calling syntax options. *Note: the MATLAB function hint feature shows the first calling syntax option only.*

1. `yFilt = k_filter(xData, yData, filterCoeffs, nPasses, ADV)`
2. `yFilt = k_filter([], yData, filterCoeffs, nPasses, ADV)`
3. `xyFilt = k_filter(xyData, [], filterCoeffs, nPasses, ADV)`

- **Sections in every help page**
  - Calling syntax
  - Input arguments explanations
  - Output results explanations
  - Optional ADV arguments
  - Simple depictions of function use
  - Links to full examples that utilize the specific function
- **When using a function, consider **READING** its help page!**
  - You will learn a lot and be able to take full advantage of all of its features.

# Finding Kornucopia Tutorials & Examples

- **Open live in MATLAB via `k_examplesOpen`**
  - Examples open live in MATLAB for you to explore and use as templates for your own work.
- **From Kornucopia Help System**
  - Listed alphabetically, useful for quick look at “published” HTML version of examples.



9 tutorials listed first, on top

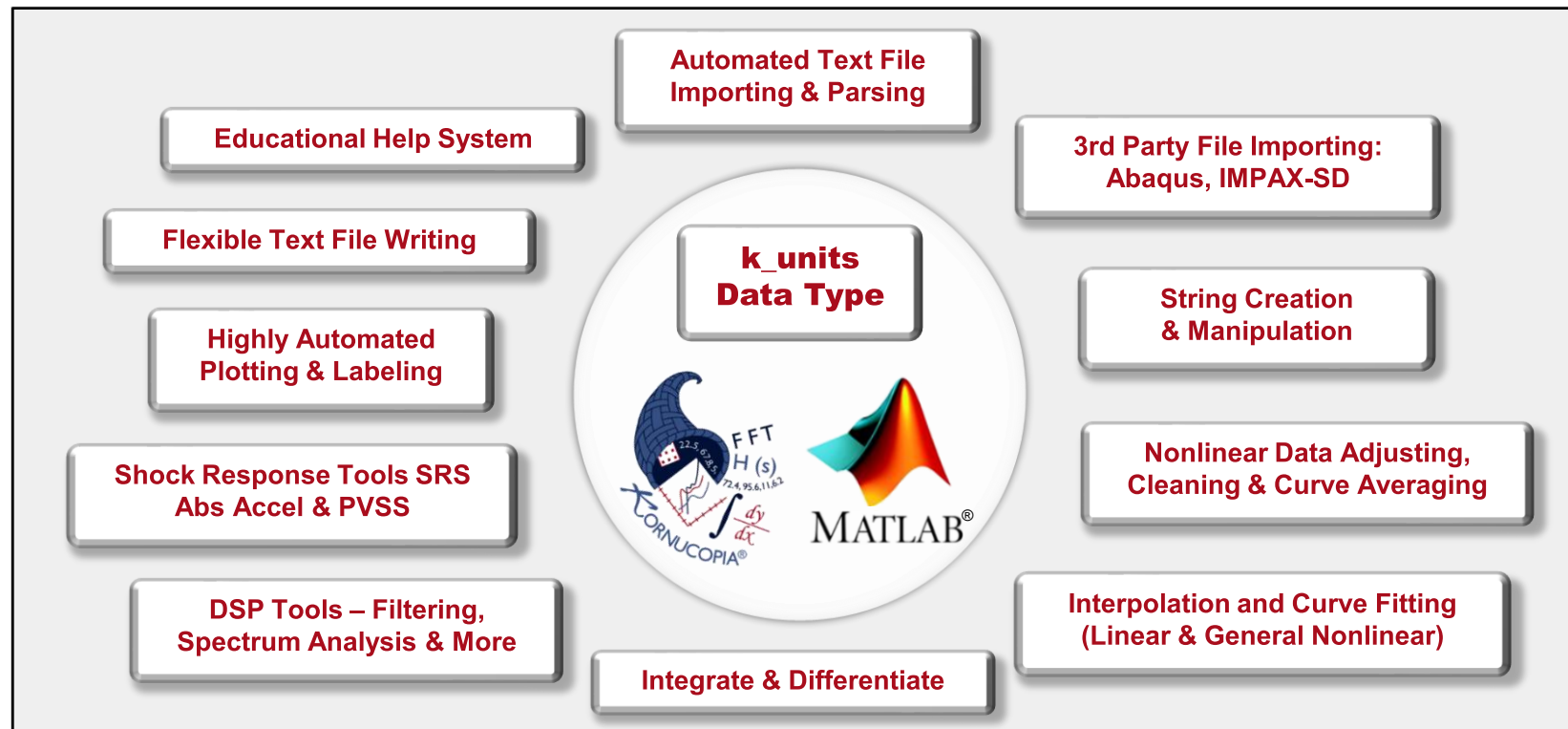
Remainder of list are ~40 general purpose examples listed alphabetically





# An Introduction to the *k\_units* Data Type

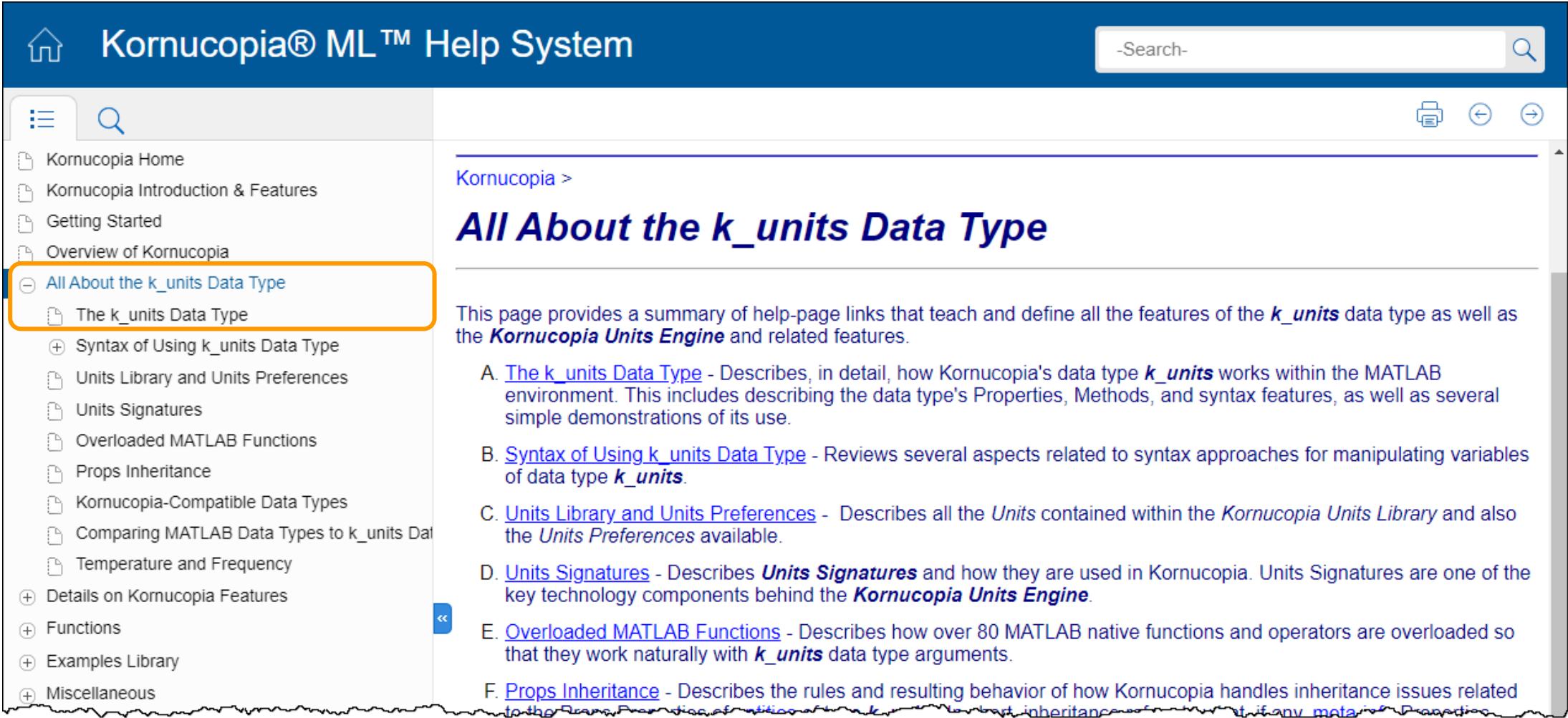
The heart of how Kornucopia holds data





# All About the *k\_units* Data Type

- This help section provides detailed discussions and numerous syntax examples to explain all you need to know to effectively use this powerful data type.



Want to be a power user?

Read these help pages to learn all the powerful syntax features and behaviors of this flexible capability.



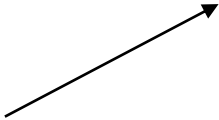


# Typical Data Types for Kornucopia Workflows

- **MATLAB data types** – See MATLAB Help System

- **double** – Holds scalar and array data in double precision.
- **char** – A character array such as 'Here is some stuff'.
- **string** – Holds string arrays such as "Here is example".
  - Generally NOT used in Kornucopia. Use **char** instead.
- **logical** – Holds scalar or array logicals of `true` and/or `false`, often displayed as 1 and/or 0.
- **struct** – Holds structure which has named fields that can contain any data type.
- **cell** – An array container whose elements can hold any data type.
  - **cellstr** is a cell array in which all elements hold `char` data type.
- **table** – Holds tabular data including various Properties for meta-info. The primary tabular data must be of the same data type in a given column, but each column can be of nearly\*\* any data type.

\*\* To hold `k_units` data type in a MATLAB table column, it must be wrapped inside a `cell` array.



```
A =  
    25.1327    3.1416    18.8496  
     9.4248    15.7080    21.9911  
    12.5664    28.2743     6.2832
```

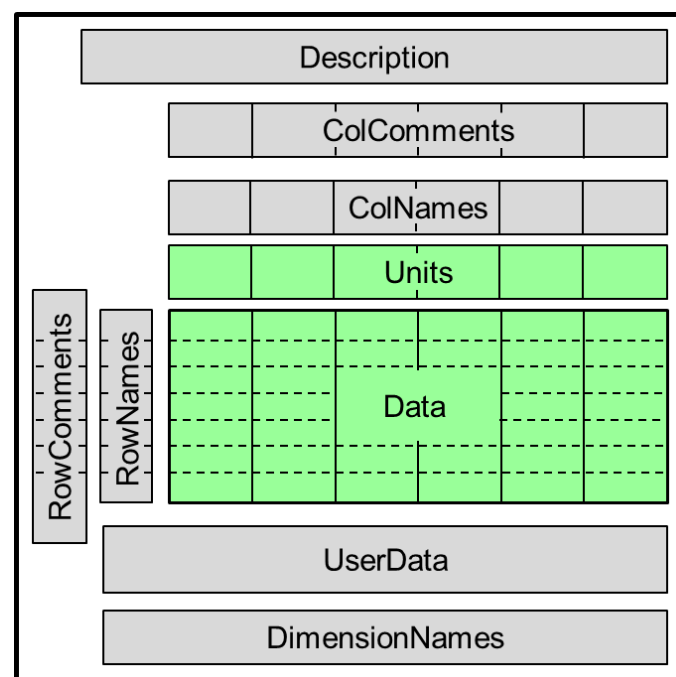
Note: due to historical reasons, Kornucopia terminology uses the word *string* to mean *char*. Also, *char* use single quotes, MATLAB *string* use double quotes.

- **Kornucopia data types** – See Kornucopia Help System

- **k\_units** – Tabular data type which holds numeric data, units, and other meta-info.
- **k\_func** – This data type is returned from Kornucopia curve fitting and interpolation functions.
- **k\_figTabsH** – This data type is created by Kornucopia's `k_figTabsCreate` function.

## The *k\_units* Data Type

- Hold ALL the information for a variable in a single "container"
  - Numeric data, Units
  - Descriptive Text & Meta-Information
- Coupled to a seamless *Units Engine*
- Cleaner MATLAB code, less mistakes



k\_varViewer displaying raw

File Edit View Insert Tools Desktop Window Help

k\_varViewer displaying raw

=== ADDITIONAL PROPS INFO ===

\*\*Props.Description

Results for impact test B2054

	Time [ms]	A3 [1000*G]	Velocity [mm/s]	NE11 [%]
445842	7.0802	7.7065	1.8250e+04	0.1182
445843	7.0802	7.6914	1.8251e+04	0.1182
445844	7.0802	7.6762	1.8252e+04	0.1183
445845	7.0803	7.6610	1.8254e+04	0.1183
445846	7.0803	7.6457	1.8255e+04	0.1183
445847	7.0803	7.6304	1.8256e+04	0.1184
445848	7.0803	7.6150	1.8257e+04	0.1184
445849	7.0803	7.5996	1.8258e+04	0.1185

*How this all works (and much more) will be explained in this training*

Editor - Untitled\*

Untitled\* x +

```

3 % Max effective stress from Accel
4 Meff = 5*lbm;
5 r = 25*cm;
6 Area = pi*r^2;
7 Stress = Meff * raw('A3') / Area;
8 maxStress = max(Stress)

```

Command Window

```

maxStress = 7.8922*MPa

```

# The *k\_units* Class

- k\_units* is a new class for MATLAB provided by Kornucopia® ML™.
- The data type in this class has both Properties and Methods, accessed by “dot” notation.
  - `matls.Props.ColNames = ...  
    'E, nu, CTE, dens'`
  - `matls('E') = ...  
    matls('E').convert('ksi')`
- While the *k\_units* data type is inspired by MATLAB’s `table` data type, the *k\_units* data type has many additional enhancements:
  - Seamless *Units Engine*
  - Efficient additional syntax behaviors
  - Over 80 overloaded MATLAB functions to support natural use of *k\_units* in typical MATLAB calculations.

```
matls =  
Some typical material properties  
=====
```

	E	nu	CTE	dens
	[GPa]		[1e-6*(m/m)/degC]	[kg/m^3]
Aluminum	72	0.33	23.5	2700
Copper	118	0.33	16.9	8960
Glass	69	0.24	9.2	2500

**\*\* Note: Other Props exists but are not displayed.**

Variable name

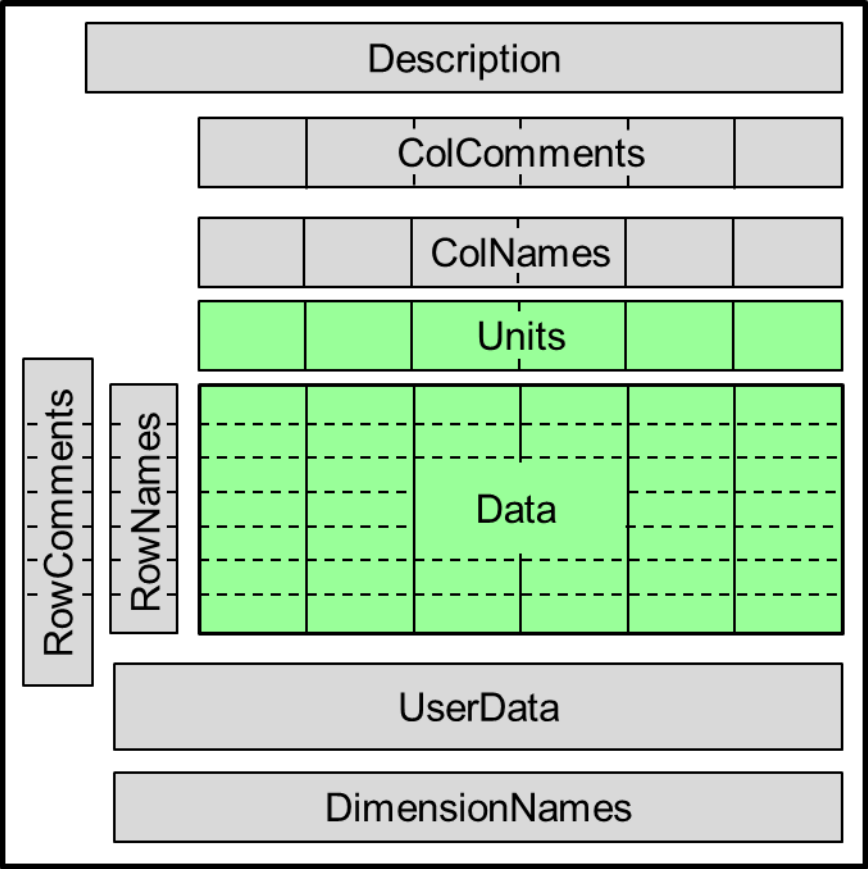
Props.Description

Props.ColNames

Units

Data

Props.RowNames



## k\_units Properties

- Data
- Units
- Props
  - Description
  - ColNames
  - RowNames
  - ColComments
  - RowComments
  - UserData
  - DimensionNames

## k\_units Methods

- convert
- dataSize
- toStruct
- toTable



## Ways to Create a Variable Holding **k\_units** Data Type

- Four primary ways to create or obtain a variable of data type `k_units`

1. The `k_units` class constructor function.

```
v = k_units(data, units, ADV)
```

`data` is a numeric scalar or 2-D numeric array.

`units` defines the units for the entire array with either 1 unit or different units, by column.

`ADV` are optional advanced arguments to define properties such as `ColNames` and others.

2. Performing any math (+, -, \*, /, ^, sin, log, etc.) on entities where one or more of the entities is of data type `k_units`.

```
Area = w*h (if w or h is of data type k_units, then Area is returned as type k_units)
```

3. Importing a data file into MATLAB using one of Kornucopia's file reading or importing functions.

4. Using nearly any Kornucopia function. With only a few exceptions, nearly all Kornucopia functions return a `k_units` data type, even when none of the input arguments are of data type `k_units`.

- See *Kornucopia Help System* page "**The `k_units` Data Type**" for full details.

# Creating a Few Variables of Data Type **k\_units** Via the **k\_units** Constructor

Command Window

>> Lx = k\_units(25, {'mm'})

Lx = 25.00\*mm

>> vals = [0, 10, 0.8; 5, 15, 1.2]

vals =

0	10.0000	0.8000
5.0000	15.0000	1.2000

>> B = k\_units(vals, {'s', 'N/m^2', 'lbf'})

B =

[s]	[N/m^2]	[lbf]
0.000	10.00	0.8000
5.000	15.00	1.200

>> C = k\_units(vals, 's, N/m^2, lbf')

C =

[s]	[N/m^2]	[lbf]
0.000	10.00	0.8000
5.000	15.00	1.200

Creating a scalar of **k\_units** data type

Creating an array of numbers (MATLAB data type double)

Creating an array of **k\_units** data type

Creating an array of **k\_units** data type using the **Kornucopia string-list syntax** instead of cell-string for defining the units.

Note: the leading **>>** is the Command Window prompt, you do not type that.

This is normally how the top single unit entry would be typed (no cell arrays)

Command Window

>> Lx = k\_units(25, 'mm')

Lx = 25.00\*mm

Note: The 3 characters **, ; :** are reserved as delimiters for easily defining multiple *units*, *ColNames*, and *RowNames* using a single string (called a *string-list*).



# Adding/Modifying .Props to Existing k\_units Type Variables

- Starting with variable C from before:

Command Window

>> C

C =

[s]	[N/m^2]	[lbf]
0.000	10.00	0.8000
5.000	15.00	1.200

>> C.Props

ans =

struct with fields:

Description: {}  
    ColNames: {}  
    ColComments: {}  
    RowNames: {}  
    RowComments: {}  
    DimensionNames: {}  
    UserData: []

RowComments

RowNames

Description

ColComments

ColNames

Units

Data

UserData

DimensionNames

>> C.Props.ColNames = 'Time, pressure, left Force';  
>> C.Props.Description = 'My 1st k\_units variable';  
>> C

C =

My 1st k\_units variable  
=====

Time	pressure	left Force
[s]	[N/m^2]	[lbf]
0.000	10.00	0.8000
5.000	15.00	1.200

>> C.Props

ans =

struct with fields:

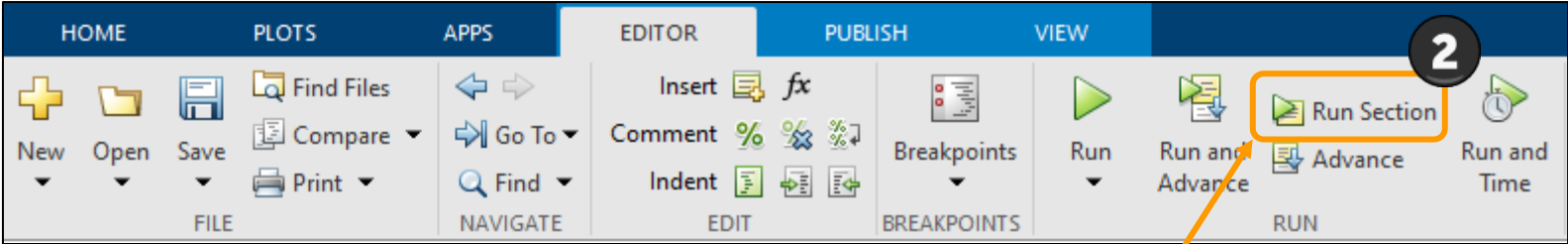
Description: {'My 1st k\_units variable'}  
    ColNames: {'Time' 'pressure' 'left Force'}  
    ColComments: {}  
    RowNames: {}  
    RowComments: {}  
    DimensionNames: {}  
    UserData: []





# Defining .Props Directly Via the k\_units Constructor

- Let's also start to use MATLAB's Script Editor.



**1** Type this in the editor, then push **Run Section** button

```
1
2 - titleTxt = {
3     'My 2nd k_units variable'
4     'Multiple line descriptions are allowed!'
5 };
6
7 - D = k_units(vals, 's, N/m^2, lbf', ...
8     'ColNames', 'Time, pressure, left Force', ...
9     'Description', titleTxt);
10
11 - display(D)
12
```

syntax format

```
k_units(Data, units, ...
        keyWord, value, ...
        keyWord, value)
```

**3** Result displayed in the command window

Command Window

D =

My 2nd k\_units variable  
Multiple line descriptions are allowed!  
=====

Time	pressure	left Force
[s]	[N/m^2]	[lbf]
0.000	10.00	0.8000
5.000	15.00	1.200

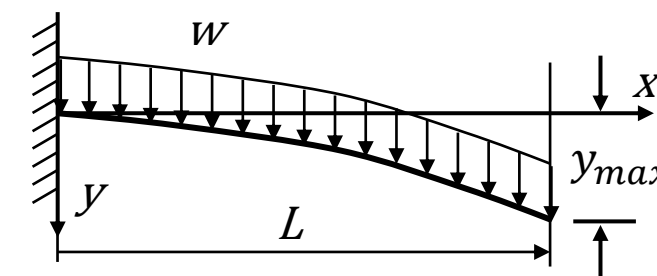


# Doing Math with **k\_units** Variables – Cantilever Beam

## • Given

- Beam parameters:  $L = 1 \cdot \text{ft}$ ,  $h = 3 \cdot \text{mm}$ ,  $b = 1 \cdot \text{in}$ ,  $E = 29 \cdot 10^3 \text{ksi}$ ,  
 $\rho = 7860 \cdot \text{kg/m}^3$ , distributed load  $w$  is self weight.

- Beam deflection equation:  $y(x) = \frac{w}{24 \cdot E \cdot I} \cdot (x^4 - 4 \cdot L \cdot x^3 + 6 \cdot L^2 \cdot x^2)$



rectangular cross-section,  
thickness  $h$  and depth  $b$ .

- **Compute**  $y_{max}$  in terms of units of `mm` and `in`.

**1** cantileverBeam\_selfWeight\_k\_units.m

```

1 - k_cleanup();
2 - k_unitsPreferenceActivate('mm_N_s');
3
4 - %% Define the parameters
5 - L = k_units(1, 'ft');
6 - h = k_units(3, 'mm');
7 - b = k_units(1, 'in');
8 - E = k_units(29e3, 'ksi');
9 - rho = k_units(7860, 'kg/m^3');
```

to be explained shortly

**3** Command Window

```

Units Preference now activated: 'mm_N_s'

yMax = 0.5545*mm
yMax = 0.02183*in
```

**2**

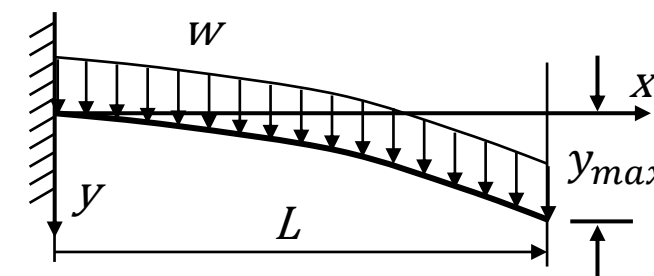
```

12 - %% Additional calculations
13 - % Bending moment of inertia, I
14 - I = (1/12)*b*h^3;
15
16 - % Beam total mass
17 - A = b*h; % beam cross-sectional area
18 - M_beam = rho*A*L;
19
20 - % Distributed load, w
21 - G = k_units(1, 'G'); % Gravity
22 - w = M_beam*G/L;
23
24 - % yMax is at x = L
25 - x = L;
26 - yMax = (w/(24*E*I))*(x^4 - 4*L*x^3 + 6*L^2*x^2);
27 - display(yMax)
28 - yMax = yMax.convert('in');
29 - display(yMax)
```

using .convert method to change a variable's Units

## Doing Math with **k\_units** Variables – Cantilever Beam

- Slightly change the script for using function **k\_unitsVariables**
  - Easily define a number of variables to represent units.



rectangular cross-section,  
thickness  $h$  and depth  $b$ .

**1**

```
cantileverBeam_selfWeight_k_unitsVariables.m
1 - k_cleanup();
2 - k_unitsPreferenceActivate('mm_N_s');
3 - k_unitsVariables('ft, mm, in, ksi, kg, m, G');
4
5 - %% Define the parameters
6 - L = 1*ft;
7 - h = 3*mm;
8 - b = 1*in;
9 - E = 29e3*ksi;
10 - rho = 7860*kg/m^3;
```

This is easier to type  
than previous  
approach

**2**

```
12 - %% Additional calculations
13 - % Bending moment of inertia, I
14 - I = (1/12)*b*h^3;
15
16 - % Beam total mass
17 - A = b*h; % beam cross-sectional area
18 - M_beam = rho*A*L;
19
20 - % Distributed load, w
21 - w = M_beam*G/L;
22
23 - % yMax is at x = L
24 - x = L;
25 - yMax = (w/(24*E*I))*(x^4 - 4*L*x^3 + 6*L^2*x^2);
26 - display(yMax)
27
28 - yMax = yMax.convert('in');
29 - display(yMax)
```

**3** Command Window

```
Units Preference now activated: 'mm_N_s'

yMax = 0.5545*mm
yMax = 0.02183*in
```

Same results  
as before

## Kornucopia Can Find Units-Related Mistakes

- Continuing with previous example, what if we typed the beam deflection equation incorrectly into MATLAB?
- The equation is supposed to be

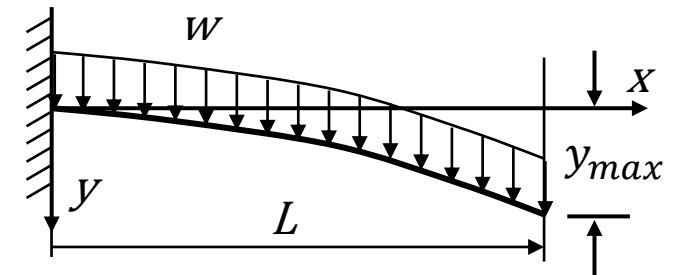
```
23 % yMax is at x = L
24 - x = L;
25 - yMax = (w/(24*E*I))*(x^4 - 4*L*x^3 + 6*L^2*x^2);
26 - display(yMax)
```

- What if we mistakenly typed the following instead

```
23 % yMax is at x = L
24 - x = L;
25 - yMax = (w/(24*E*I))*(x^4 - 4*L*x^2 + 6*L^2*x^2);
26 - display(yMax)
```

**Kornucopia tells you that around the subtraction operator a Units-related error has occurred.**

- Kornucopia error messages are information rich. You will find that if you read them, they will be very helpful.



rectangular cross-section,  
thickness  $h$  and depth  $b$ .

### Command Window

```
Error using -
Incompatible units error. The following two units do not
resolve to the same primary units signature:
    'mm^4' and 'mm^3'.
Note: A viable conversion would be the following:
    'mm^4' to '(mm^3)*(mm)'.
Click this link for detailed help on k\_units
Click this link for details on overloaded MATLAB functions.

Error in cantileverBeam_selfWeight_k_unitsVariables_eqnType
(line 25)
yMax = (w/(24*E*I))*(x^4 - 4*L*x^2 + 6*L^2*x^2);
```

# A Note About the Units Your Variables May Appear With When Output

- With the **k\_units** constructor, the variables are kept in the Units you define the variables with.
  - When math is performed, the resulting Units of a variable depend on the active *Units Preference*.

```
1 k_cleanup();
2 k_unitsPreferenceActivate('mm_N_s');
3 k_unitsVariables('ft, kN')
4
5 %%ok<*NOPTS> Supress warnings of no ; at end of commands
6
7 %% Define some parameters
8
9 l = k_units(30, 'in')
10 f = k_units(3, 'kN')
11
12 L = 2.5*ft
13 F = 3*kN
14
15 % Do a simple calc
16 lf = l * f
17 LF = L * f
```

**1** using **k\_units** constructor

**2** Doing math on values and variables

**3** Both of these calcs yield the same result as expected!

Workspace		
Name ▲	Value	Class
ft	1x1 k_units	k_units
kN	1x1 k_units	k_units

```
Command Window
Units Preference now activated: 'mm_N_s'
>> ft
ft = 1.000*ft
>> kN
kN = 1.000*kN
```

manually showing what variables **ft** and **kN** are holding.

```
Command Window
l = 30.00*in
f = 3.000*kN
L = 762.0*mm
F = 3000*N
lf = 2.286e+06*N*mm
LF = 2.286e+06*N*mm
```

**1**

**2**

**3**

Outputs 2 & 3 came from math operations, and thus they are in the requested *Units Preference*



# Clarifying What the Function `k_unitsVariables` Is Doing and Is Not Doing

When you issue a command like

```
k_unitsVariables('m, lbf, G')
```

Kornucopia internally does the following for you:

- 1. It makes variables with the names of the Units you request.
- 2. Before placing these variables into the MATLAB workspace, it confirms the variables do not already exist.
  - If they do, it confirms they are representing the same Units as being requested.
- 3. Passing the above check, the variables are placed into the workspace.

```
m = k_units(1, 'm');  
lbf = k_units(1, 'lbf');  
G = k_units(1, 'G');
```

Ending ; suppresses variable echo to Command Window

You can then use these variables in math statements as needed to create equations or other variables as needed.

```
w = 320.5 * lbf; % Define a weight  
M = w / G; % Compute a mass
```

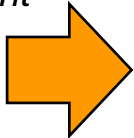
Workspace		
Name ▲	Value	Class
G	1x1 k_units	k_units
lbf	1x1 k_units	k_units
m	1x1 k_units	k_units

Be careful to not unintentionally override a Units Variable as shown below.

```
m = w / G % Compute a mass  
L_a = k_units(10, 'm')  
L_b = 10 * m
```

Assignment

Math



```
Command Window  
m = 0.1454*Mg  
L_a = 10.00*m  
L_b = 1.454*Mg
```

Variable m

Unit m

Kornucopia still understands the Unit 'm', what has been changed is the variable m, which now represents a mass.





## Kornucopia's Units Engine

- **True *Units Engine* using a variable's *Units Signature***
  - Much more than just a simple Units conversion table
  - Properly tracks Units compatibility in all calculations
    - Allows mixed Units to be utilized
    - Robustly processes temperatures, including affine Units conversions
- **Hundreds of Units pre-defined in the Library**
  - m, N, s, mm, lbf, in, kg, snail, slinch, furlong, fortnight, cottonCount ...
- **Easily add Units definitions**
  - `k_unitsDefine('c = 299.792458e6*m/s')`
  - `k_unitsDefine('Mil = 3.375*arcmin')`
- **It is simple to control Units within your output via *Units Preferences* or the *.convert method***
  - Use one of the predefined *Units Preferences* or easily create your own

### Primary Units Signatures

- Time
- Mass
- Length
- Current
- Substance
- Luminous Intensity
- Currency
- Absolute Temperature

### Additional Signatures

- Celsius
- Fahrenheit
- Angle
- Frequency
- Percent

## Kornucopia's Large Units Library with Over 230 Pre-defined Units

- Each of the units is clearly defined and documented.

# Kornucopia® ML™ Help System

## Detailed Listing of Kornucopia Unit Definitions

Listed below in the table are the detailed Units definitions for each of the installed Units in Kornucopia.

- To see this list *live* in MATLAB, use the function [k\\_unitsList](#). Seeing the list live, you will also be able to see any user-defined units too!
- In the table below, the column **Original Formula** displays the given Unit as it was originally defined in Kornucopia. The column **Base Definition** lists the Units definition in terms of Kornucopia's fundamental Units.
- The table, as presented, shows the unit's conversion factors to an accuracy consistent with the default MATLAB `format short` setting. All of the unit's conversion factors are actually accurate to the full double precision of MATLAB. To see the more accurate representation, issue the MATLAB command `format long` and then the Kornucopia function `k_unitsList('-all')`. Also note that the live list will display a column called **Source** which will show Kornucopia as the source for all Units that are not user-defined.

### Default Kornucopia Units, Detailed Definitions List

Unit	Original Formula	Base Definition (Short Format)	Description	Unit Groups
%	% = 0.01	% = 0.01	percent	percent, miscellaneous
1	1 = 1	1 = 1	unity is dimensionless	miscellaneous
A	fundamentalUnit	A = 1*A	ampere	current, fundamental
acre	acre = 43560*ft^2	acre = 4047*m^2	acre	area
arcmin	arcmin = 1/60*deg	arcmin = 0.0002909*rad	arc minute	angle
arcsec	arcsec = 1/3600*deg	arcsec = 4.848e-06*rad	arc second	angle
atm	atm = 1.013250e5*Pa	atm = 1.013e+05*kg/(m*s^2)	atmosphere	pressure
bar	bar = 1e5*Pa	bar = 1e+05*kg/(m*s^2)	bar	pressure
blob	blob = 12*slug	blob = 175.1*kg	blob	mass
BTU	BTU = 1.055055852620*kJ	BTU = 1055*m^2*kg/s^2	International BTU	energy



## Important Notes About Units Related to grams and Gravity

- In Kornucopia there is NO default Unit called g.
  - This is because some would think it is a gram and others would think it is a unit of gravity.
- Instead, Kornucopia has the following:
  - gm is 1 gram mass
  - kg is 1 kilogram mass
  - gf and gmf are both 1 gram force
  - gpd is 1 gram force per denier
  - grn is 1 grain mass
  - G is 1 unit of acceleration of standard gravity =  $9.806650 \text{ m/s}^2$
  - kG is  $1000 * G$
  - mG is  $(1/1000) * G$

## Units Preferences and Units Systems

- Kornucopia's *Units Engine* knows how to properly work with mixed Units in calculations. This means you can do things like

$$A = 5*\text{mm} + 3*\text{in} \quad \text{or} \quad B = 7.3*\text{N}/\text{ft}^2.$$

- A Kornucopia *Units Preference* is like a Units System, but more flexible. The basic idea of a Units Preference is that you are telling the software what Units you want results to be transformed into when calculations are performed.
  - You can also easily change the Units of any variable at any time via the `k_units` method of `.convert` or the function `k_unitsConvert`.
  - The added flexibility of a Units Preference is that by default you are allowed to define a list of mixed units for a given Preference, meaning you can have a Preference that returns results where length (and displacement) are in `mm`, loads in `lbf`, and stress in `GPa`.
  - Many of the installed Kornucopia Units Preferences define a consistent set of units (exceptions include the Units Preferences of ending in `_v` and the Preferences starting with `ShockVib_`).
  - There are four specific functions related to Units Preferences: `k_unitsPreferenceList`, `k_unitsPreferenceActivate`, `k_unitsPreferenceActive`, and `k_unitsPreferenceDefine`.

# Many Default *Units Preferences* to Choose From

- See them via `k_unitsPreferenceList`
- Reference them via their abbreviated names when using the function `k_unitsPreferenceActivate` or the `.convert` method.
- Create your own Units Preference via function `k_unitsPreferenceDefine`

Command Window	
>> k_unitsPreferenceList	
UnitsPreference Name	Units in the Preference
none	
fundamental	m, kg, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %
ft_lbf_s	ft, slug, A, mol, cd, currency, s, R, degC, degF, rad, Hz, %, lbf, lbf*ft, lbf/ft, ft/lbf, psf, l/psf
ft_lbf_s_V	ft, slug, A, mol, cd, currency, s, R, degC, degF, rad, Hz, %, lbf, lbf*ft, lbf/ft, ft/lbf, psf, l/psf, V, ohm, S, C, F, H, Wb, T
in_lbf_slinch_s	in, slinch, A, mol, cd, currency, s, R, degC, degF, rad, Hz, %, lbf, lbf*in, lbf/in, in/lbf, psi, l/psi
in_lbf_slinch_s_V	in, slinch, A, mol, cd, currency, s, R, degC, degF, rad, Hz, %, lbf, lbf*in, lbf/in, in/lbf, psi, l/psi, V, ohm, S, C, F, H, Wb, T
in_lbf_snail_s	in, snail, A, mol, cd, currency, s, R, degC, degF, rad, Hz, %, lbf, lbf*in, lbf/in, in/lbf, psi, l/psi
in_lbf_snail_s_V	in, snail, A, mol, cd, currency, s, R, degC, degF, rad, Hz, %, lbf, lbf*in, lbf/in, in/lbf, psi, l/psi, V, ohm, S, C, F, H, Wb, T
m_N_s	m, kg, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %, l/Pa, V, ohm, S, C, F, H, Wb, T
mm_N_ms	mm, gm, A, mol, cd, currency, ms, K, degC, degF, rad, Hz, %, MPa, l/MPa, V, ohm, S, mC, mF, mH, mWb, kT
mm_N_s	mm, Mg, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %, l/MPa, mV, mohm, kS, C, kF, mH, mWb, kT
nm_nN_ms	nm, mg, mA, mol, cd, currency, ms, K, degC, degF, rad, Hz, %, nm/nN, GPa, l/GPa, pV, nohm, GS, uC, uV, uohm, MS, mC, kF, nH, nWb, kT
ShockVib_ft_ms_G	ft, slug, A, mol, cd, currency, ms, K, degC, degF, rad, Hz, %, lbf/ft, ft/lbf, psf, l/psf, V, ohm, S, C, F, H, Wb, T
ShockVib_ft_ms_kG	ft, slug, A, mol, cd, currency, ms, K, degC, degF, rad, Hz, %, lbf*ft, lbf/ft, ft/lbf, psf, l/psf, V, ohm, S, C, F, H, Wb, T
ShockVib_ft_s_G	ft, slug, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %, ft/lbf, psf, l/psf, V, ohm, S, C, F, H, Wb, T
ShockVib_ft_s_kG	ft, slug, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %, lbf/ft, ft/lbf, psf, l/psf, V, ohm, S, C, F, H, Wb, T

ShockVib_ft_s_kG	ft, slug, A, mol, cd, currency, s, R, degC, degF, rad, Hz, %, kG, lbf, lbf*ft, lbf/ft, ft/lbf, psf, l/psf, V, ohm, S, C, F, H, Wb, T
ShockVib_in_ms_G_snail	in, snail, A, mol, cd, currency, ms, R, degC, degF, rad, kHz, %, in/s, G, lbf, lbf*in, lbf/in, in/lbf, psi, l/psi, V, ohm, S, C, F, H, Wb, T
ShockVib_in_ms_kG_snail	in, snail, A, mol, cd, currency, ms, R, degC, degF, rad, kHz, %, in/s, kG, lbf, lbf*in, lbf/in, in/lbf, psi, l/psi, V, ohm, S, C, F, H, Wb, T
ShockVib_in_s_G_snail	in, snail, A, mol, cd, currency, s, R, degC, degF, rad, Hz, %, G, lbf, lbf*in, lbf/in, in/lbf, psi, l/psi, V, ohm, S, C, F, H, Wb, T
ShockVib_in_s_kG_snail	in, snail, A, mol, cd, currency, s, R, degC, degF, rad, Hz, %, kG, lbf, lbf*in, lbf/in, in/lbf, psi, l/psi, V, ohm, S, C, F, H, Wb, T
ShockVib_m_ms_G	m, kg, A, mol, cd, currency, ms, K, degC, degF, rad, kHz, %, G, N, N*m, N/m, m/N, Pa, l/Pa, V, ohm, S, C, F, H, Wb, T
ShockVib_m_ms_kG	m, kg, A, mol, cd, currency, ms, K, degC, degF, rad, kHz, %, kG, N, N*m, N/m, m/N, Pa, l/Pa, V, ohm, S, C, F, H, Wb, T
ShockVib_m_s_G	m, kg, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %, G, N, N*m, N/m, m/N, Pa, l/Pa, V, ohm, S, C, F, H, Wb, T
ShockVib_m_s_kG	m, kg, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %, kG, N, N*m, N/m, m/N, Pa, l/Pa, V, ohm, S, C, F, H, Wb, T
ShockVib_mm_ms_G	mm, gm, A, mol, cd, currency, ms, K, degC, degF, rad, kHz, %, G, N, N*mm, N/mm, mm/N, MPa, l/MPa, V, ohm, S, mC, mF, mH, mWb, kT
ShockVib_mm_ms_kG	mm, gm, A, mol, cd, currency, ms, K, degC, degF, rad, kHz, %, kG, N, N*mm, N/mm, mm/N, MPa, l/MPa, V, ohm, S, mC, mF, mH, mWb, kT
ShockVib_mm_s_G	mm, Mg, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %, G, N, N*mm, N/mm, mm/N, MPa, l/MPa, mV, mohm, kS, C, kF, mH, mWb, kT
ShockVib_mm_s_kG	mm, Mg, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %, kG, N, N*mm, N/mm, mm/N, MPa, l/MPa, mV, mohm, kS, C, kF, mH, mWb, kT
SI_partial	m, kg, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %, N, N*m, N/m, m/N, Pa, l/Pa, V, ohm, S, C, F, H, Wb, T
um_mN_ms	um, gm, A, mol, cd, currency, ms, K, degC, degF, rad, kHz, %, mN, mN*um, mN/um, um/mN, GPa, l/GPa, uV, uohm, MS, mC, kF, nH, nWb, kT
um_Mg_ms	um, Mg, A, mol, cd, currency, s, K, degC, degF, rad, Hz, %, mN, mN*um, mN/um, um/mN, GPa, l/GPa, nV, nohm, GS, C, GF, nH, nWb, kT
um_uG_ms	um, ug, A, mol, cd, currency, us, K, degC, degF, rad, MHz, %, mN, mN*um, mN/um, um/mN, GPa, l/GPa, mV, mohm, kS, uC, mF, nH, nWb, kT

Below is a Units Preference commonly used for severe shock data.

ShockVib_mm_ms_kG	mm, gm, A, mol, cd, currency, ms, K, degC, degF, rad, kHz, %, kG, N, N*mm, N/mm, mm/N, MPa, l/MPa, V, ohm, S, mC, mF, mH, mWb, kT
-------------------	---



# Some Comments on Units Conversions

Consider the following specification and conversion:

1

```
vals = [
    0, 10, 0.8
    5, 15, 1.2
];
```

2

```
P = k_units(vals, 's, N/m^2, in', ...
    'colNames', 'Time, stress, Vert Length')

P('stress') = P('stress').convert('ksi')
```

1

Time	stress	Vert Length
[s]	[N/m^2]	[in]
0.000	10.00	0.8000
5.000	15.00	1.200

2

Time	stress	Vert Length
[s]	[ksi]	[in]
0.000	1.450e-06	0.8000
5.000	2.176e-06	1.200

The following command would convert all columns to the specified units

```
P = P.convert('hr, GPa, mm')
```

The following command would convert all columns to the Units Preference named 'mm\_N\_s'

```
P = P.convert('mm_N_s')
```

And this command would convert specified columns to the active Units Preference

```
P('Time, Vert Length') = ...
P('Time, Vert Length').convert()
```

To see active Units Preference

```
Command Window
>> k_unitsPreferenceActive()

Current active Units Preference is: 'm_N_s'
```

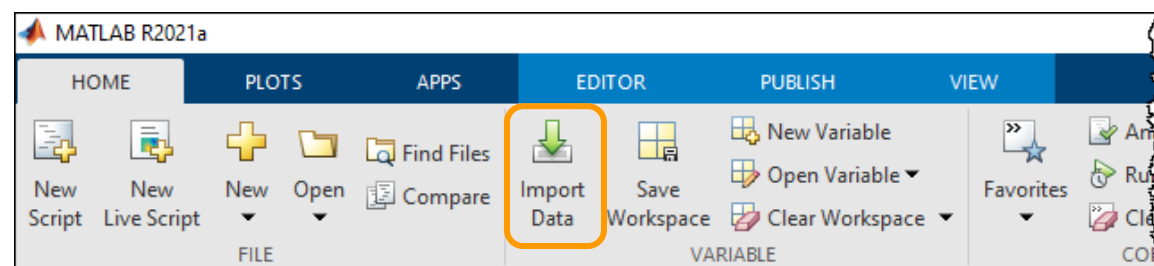




# Creating Variable of type **k\_units** Via Reading In Data Files

- There are a variety of ways to import data files into MATLAB and Kornucopia.

## ■ MATLAB functions & its import wizard



## ■ Kornucopia import functions creating **k\_units**

- General ascii text
  - `k_readText`
  - `k_unpackHeader`
- Special formats
  - Abaqus ODB: `k_abqOdbImport_gui`
  - SD Impax: `k_sdImpaxImport` & `k_sdImpaxRtmImport`

### Ways to Create a Variable Holding **k\_units** Data Type

- Four primary ways to create or obtain a variable of data type **k\_units**

#### ✓ 1. The **k\_units** class constructor function.

```
v = k_units(data, units, ADV)
```

`data` is a numeric scalar or 2-D numeric array.

`units` defines the units for the entire array with either 1 unit or different units, by column.

`ADV` are optional advanced arguments to define properties such as `ColNames` and others.

#### ✓ 2. Performing any math (+, -, \*, /, ^, sin, log, etc.) on entities where one or more of the entities is of data type **k\_units**.

`Area = w*h` (if `w` or `h` is of data type **k\_units**, then `Area` is returned as type **k\_units**)

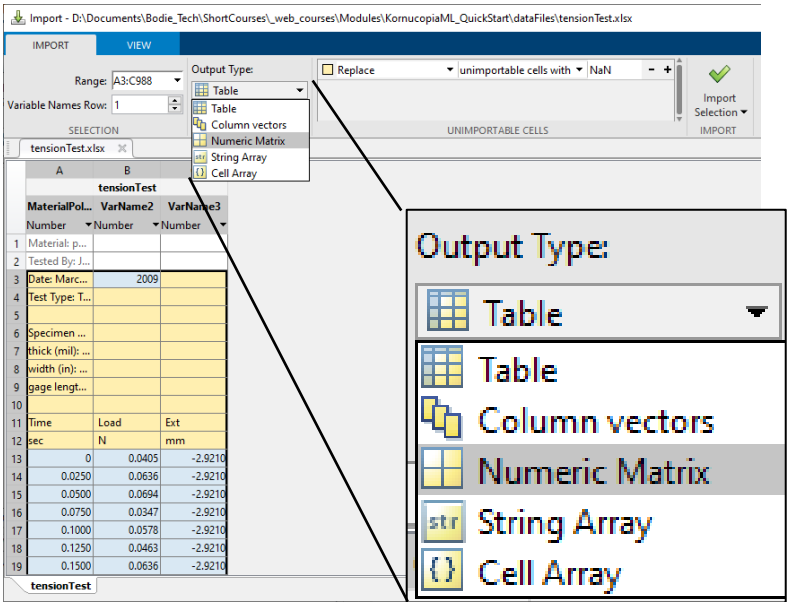
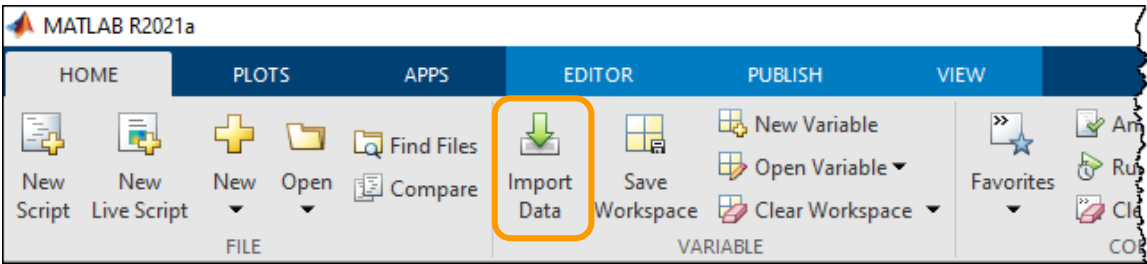
#### 3. Importing a data file into MATLAB using one of Kornucopia's file reading or importing functions.

#### 4. Using nearly any Kornucopia function. With only a few exceptions, nearly all Kornucopia functions return a **k\_units** data type, even when none of the input arguments are of data type **k\_units**.

- See *Kornucopia Help System* page "**The *k\_units* Data Type**" for full details.

# Import XLSX File Via MATLAB Import Wizard

- Consider the file `dataFiles/tensionTest.xlsx`
  - `dataFiles` folder is in the training docs you downloaded.
- Importing with MATLAB wizard has a few options



## File viewed in MS Excel

AutoSave Off

File Home Insert Page Layout Formulas

Paste Clipboard Font

Calibri 11

B I U

J8

	A	B	C
1	Material: polyA		
2	Tested By: Joe Smith		
3	Date: March 7	2009	
4	Test Type: Tension with initial slack		
5			
6	Specimen dimensions		
7	thick (mil): 4.0		
8	width (in): 0.5		
9	gage length (in): 5.000		
10			
11	Time	Load	Ext
12	sec	N	mm
13	0	0.0405	-2.921
14	0.025	0.0636	-2.921
15	0.050	0.0694	-2.921
16	0.075	0.0347	-2.921
17	0.1	0.0578	-2.921
18	0.125	0.0463	-2.921
19	0.150	0.0636	-2.921

tensionTest

Header rows

Useful values

Main tabular data section



## Import XLSX File As Numeric Matrix Via MATLAB Import Wizard

- To the right is an image of the import wizard with settings to just import Numeric Matrix of data.
  - The Output Type is set to "Numeric Matrix".
  - In this approach, all the top header info is ignored.
  - We had to adjust the Range to isolate just the numeric tabular data region (ignoring column names and Units).
  - Only the rectangular-shaped numeric data section is imported
- Once the numeric data is imported (as `tensionTest`), simply create a well described `k_units` type variable as shown below.

```
raw_1 = k_units(tensionTest, 'sec, N, mm', ...
    'ColNames', 'Time, Load, Ext', ...
    'Descrip', 'Tension Test');

raw_1 % This will echo the variable to command window
k_varViewer(raw_1) % Explore in Kornucopia variable viewer
```

Import - D:\Documents\Bodie\_Tech\ShortCourses\\_web\_courses\Modules\KornucopiaML\_QuickStart\dataFiles\tensionTest.xlsx

IMPORT VIEW

Range: A13:C988 Output Type: Numeric Matrix

Variable Names Row: 1

Replace unimportable cells with NaN

Import Selection

SELECTION IMPORTED DATA UNIMPORTABLE CELLS IMPORT

tensionTest.xlsx

	A	B	C
	tensionTest		
1	Material: p...		
2	Tested By: J...		
3	Date: Marc...	2009	
4	Test Type: T...		
5			
6	Specimen ...		
7	thick (mil): ...		
8	width (in): ...		
9	gage lengt...		
10			
11	Time	Load	Ext
12	sec	N	mm
13	0	0.0405	-2.9210
14	0.0250	0.0636	-2.9210
15	0.0500	0.0694	-2.9210
16	0.0750	0.0347	-2.9210
17	0.1000	0.0578	-2.9210
18	0.1250	0.0463	-2.9210
19	0.1500	0.0636	-2.9210
20	0.1750	0.0752	-2.9210
21	0.2000	0.0405	-2.9210

tensionTest

This is the output variable name (set to this by default by the wizard). You could change this to some other variable name you deemed appropriate.

This section of header information is ignored and NOT imported.

# Import XLSX File As Numeric Matrix Via MATLAB Import Wizard (Continued)

Assessing variable `raw_1`.

## Suggested Kornucopia Approaches

Command Window

```
raw_1 =  
Tension Test  
=====
```

Time [sec]	Load [N]	Ext [mm]
0.000	0.04050	-2.921
0.02500	0.06360	-2.921
0.05000	0.06940	-2.921
0.07500	0.03470	-2.921
0.1000	0.05780	-2.921
0.1250	0.04630	-2.921
0.1500	0.06360	-2.921
0.1750	0.07520	-2.921

... 968 rows not shown.  
See "n" rows via `k_set('dispBrief', n)`.

Easy to see key essence of the variable, or use the `k_varViewer` for a scrollable view of it all.

k\_varViewer displaying ...

Kornucopia

k\_varViewer displaying raw\_1

=== ADDITIONAL PROPS INFO ===  
\*\*Props.Description  
Tension Test

	1 Time [sec]	2 Load [N]	3 Ext [mm]
1	0	0.0405	-2.9210e+00
2	2.5000e-02	0.0636	-2.9210e+00
3	5.0000e-02	0.0694	-2.9210e+00
4	7.5000e-02	0.0347	-2.9210e+00
5	1.0000e-01	0.0578	-2.9210e+00
6	1.2500e-01	0.0463	-2.9210e+00
7	1.5000e-01	0.0636	-2.9210e+00
8	1.7500e-01	0.0752	-2.9210e+00
9	2.0000e-01	0.0405	-2.9210e+00
10	2.2500e-01	0.0405	-2.9210e+00
11	2.5000e-01	0.0520	-2.9210e+00
12	2.7500e-01	0.0520	-2.9210e+00
13	3.2500e-01	0.0463	-2.9210e+00

## Traditional MATLAB Variable Editor

Workspace

Name	Value	Class
adv	1x1 struct	struct
raw_1	1x1 k_units	k_units
tensionT...	976x3 double	double

raw\_1

1x1 k\_units

Property	Value
Data	976x3 double
Units	1x3 cell
Props	1x1 struct

raw\_1

raw\_1.Data

	1	2	3
1	0	0.0405	-2.9210
2	0.0250	0.0636	-2.9210
3	0.0500	0.0694	-2.9210
4	0.0750	0.0347	-2.9210
5	0.1000	0.0578	-2.9210
6	0.1250	0.0463	-2.9210
7	0.1500	0.0636	-2.9210
8	0.1750	0.0752	-2.9210

raw\_1

raw\_1.Data

raw\_1.Props

raw\_1.Props.ColNames

	1	2	3	4	5	6
1	Time	Load	Ext			
2						

Variable's contents viewed like a struct.

Warning: do NOT edit values using these tools.

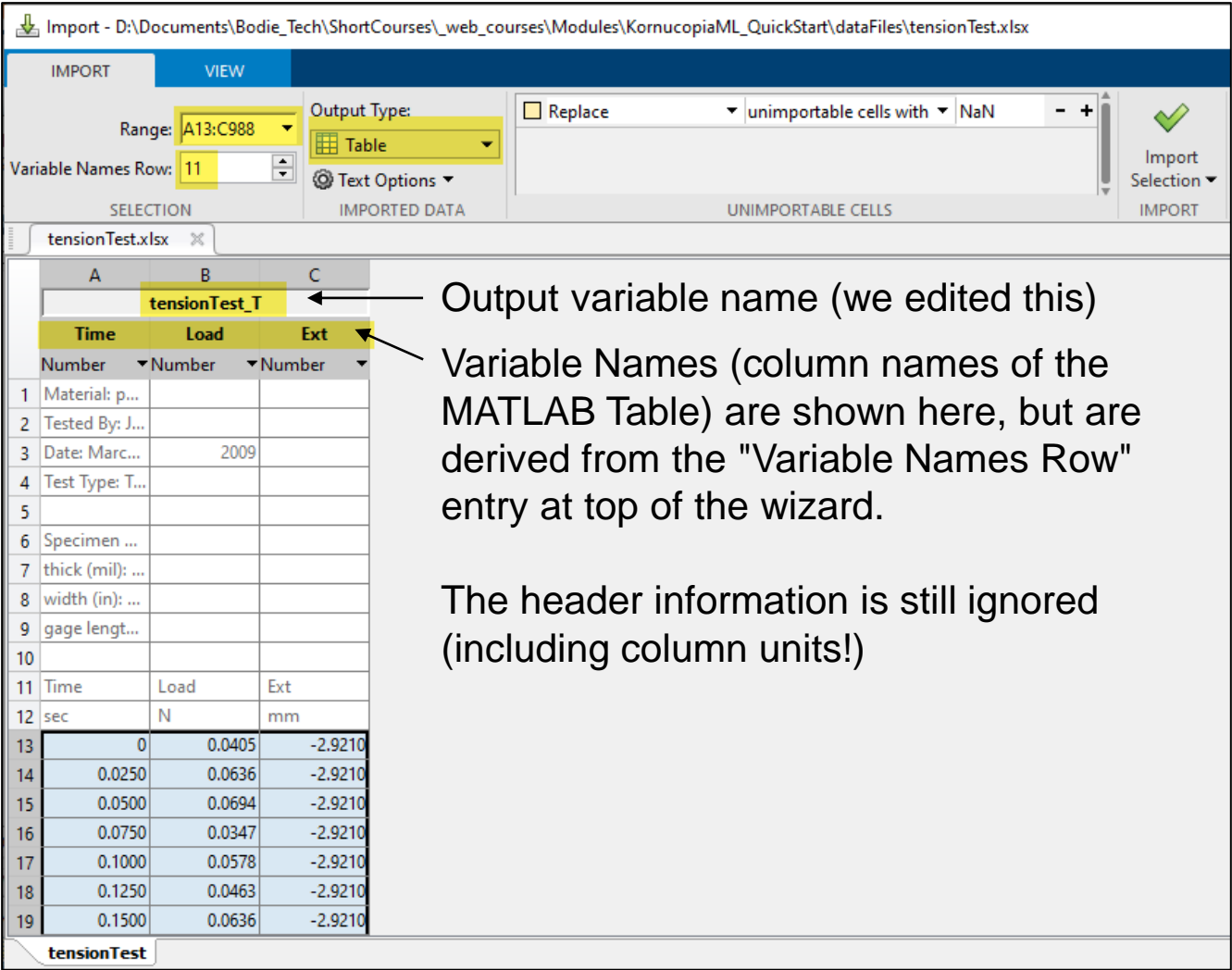
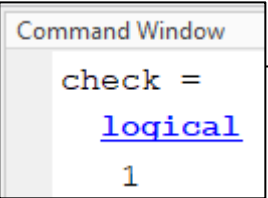


# Import XLSX File As MATLAB Table Via MATLAB Import Wizard

- To the right is an image of the import wizard with settings to import as a MATLAB Table data type.
  - The Output Type is set to "Table".
  - In this approach, all the top header info is ignored, accept we get to reference the row of column names.
  - Adjust the Range to isolate just the numeric tabular data region.
- Once the Table is imported (as `tensionTest_T`), simply create a well described `k_units` type variable as shown below.

`{:,:}` gets all rows & cols of data from table

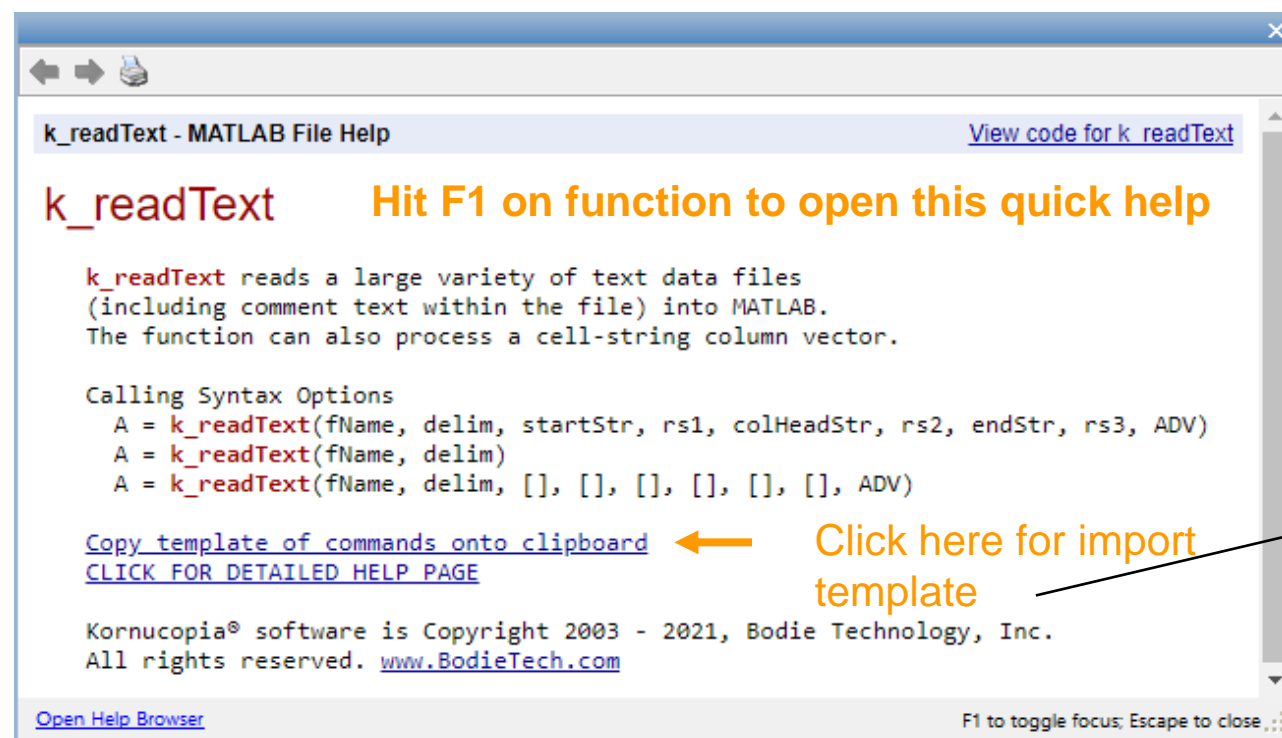
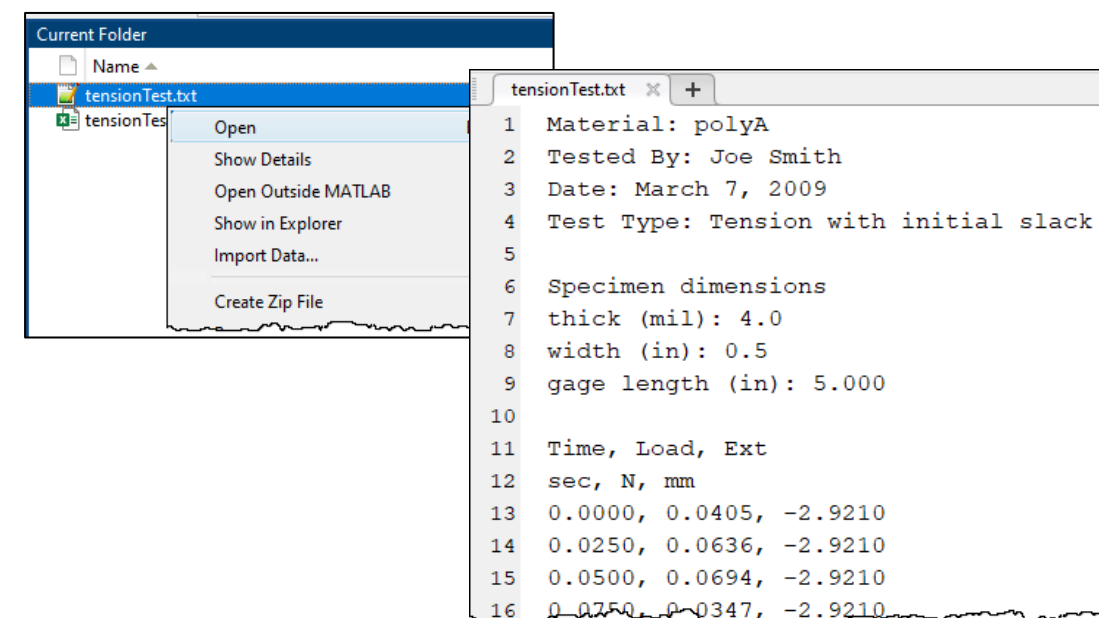
```
raw_2 = k_units(tensionTest_T{:,:}, 'sec, N, mm', ...  
    'ColNames', tensionTest_T.Properties.VariableNames, ...  
    'Descrip', 'Tension Test');  
  
check = isequal(raw_1, raw_2)
```





## Importing Txt File with `k_readText` Function

- Consider the file `dataFiles/tensionTest.txt`
  - `dataFiles` folder is in the training docs you downloaded.
- First open file as text in MATLAB editor to see its contents.
- Next, we use `k_readText` function to import the file



```

% The template below is set to read a local file named 'myFile.dat'
% that is comma delimited with the following section identifications:
% o The data set section begins one row below the row with the string
%   'Specimen dimensions' contained within it.
% o The column header section begins at the row with the string
%   'Time' within it.
% o No specific delimiter is specified for the end of the data set
%   section. Internal automatic logic will be used.
%
% If ADV options are needed, just add them after the rs3 argument in
% the usual way.

dataDir = ''; % Place path to file here (leave as-is if local file)
fileName = 'myFile.dat'; % Use string '?' for file selection popup
fName = fullfile(dataDir, fileName);
delim = ',';
startStr = 'Specimen dimensions';
rs1 = 1;
colHeadStr = 'Time';
rs2 = 0;
endStr = [];
rs3 = [];
raw = k_readText(fName,delim,startStr,rs1,colHeadStr,rs2,endStr,rs3);
  
```



## Importing Txt File with `k_readText` Function (cont)

- Modify a few values accordingly
- Then run the section to import the file.
  - The Kornucopia function will import the entire file, including header info!

```
dataDir = '../dataFiles';
fileName = 'tensionTest.txt';
fName = fullfile(dataDir, fileName);
delim = ',';
startStr = 'Specimen dimensions';
rs1 = 1;
colHeadStr = 'Time';
rs2 = 0;
endStr = [];
rs3 = [];
raw_3 = k_readText(fName,delim,startStr,rs1,colHeadStr,rs2,endStr,rs3);
```

```
Command Window
>> raw_3

raw_3 =
    tensionTest.txt
    =====
    Time      Load      Ext
    [sec]      [N]      [mm]

    0.000      0.04050    -2.921
    0.02500      0.06360    -2.921
    0.05000      0.06940    -2.921
    0.07500      0.03470    -2.921
    0.1000      0.05780    -2.921
    0.1250      0.04630    -2.921
    0.1500      0.06360    -2.921
    0.1750      0.07520    -2.921

... 968 rows not shown.
See "n" rows via k_set('dispBrief', n).
** Other Props exist that are not displayed.
View other Props via dot syntax or functions k_summary or k_varViewer.
```

```
>> raw_3.Props.UserData
ans =
    struct with fields:
        FileInfo: {3×1 cell}
        MainFileHeader: {5×1 cell}
        DatasetHeader: {3×1 cell}
        DatenumID: '736157.537476851837709546089172'
        DatasetNum: 1
```

```
>> raw_3.Props.UserData.DatasetHeader
ans =
    3×1 cell array

    {'thick (mil): 4.0'      }
    {'width (in): 0.5'      }
    {'gage length (in): 5.000'}
```

## Importing Txt File with `k_readText` Function (cont)

- Easily convert the DatasetHeader into a k\_units table via `k_unpackHeader`
  - The function is very flexible – see the function's help page for more details.
- Then perform a few additional calcs
  - Compare area from the header parameters
  - Compute raw strain and stress
    - Convert strain output to units of %.

```
DatasetHeader = raw_3.Props.UserData.DatasetHeader;  
params = k_unpackHeader(DatasetHeader, ':')
```

Command Window		
params =		
thick	width	gage length
[mil]	[in]	[in]
<hr/>		
4.000	0.5000	5.000

```
Area = params('width') * params('thick')  
  
raw_3('Strain') = raw_3('Ext') / params('gage length');  
raw_3('Stress') = raw_3('Load') / Area;  
raw_3('Strain') = raw_3('Strain').convert('%');  
  
raw_3
```

# Importing Txt File with k\_readText Function (cont)

- The warnings come from doing various forms of math where it is not obvious what the ColNames or other Props should be for the resulting output.
  - Example: For the Area calc, what should code give as the resulting ColName?
    - The software is not psychic nor human.
  - See Kornucopia Help, **Props Inheritance** to learn more and how to turn off this warning if desired.
    - See next slide for more info.

```
Command Window
Warning: While using the "*" operator, the output result did NOT inherit
any of the "Props" from the input argument(s).
Click this link to learn more.

Area = 1.290*mm^2

Warning: While using the "/" operator, the output result did NOT inherit
any of the "Props" from the input argument(s).
Click this link to learn more.

Warning: While using the "/" operator, the output result did NOT inherit
any of the "Props" from the input argument(s).
Click this link to learn more.

raw_3 =
  tensionTest.txt
=====
      Time      Load      Ext      Strain      Stress
      [sec]      [N]      [mm]      [%]      [MPa]
-----
    0.000    0.04050    -2.921    -2.300    0.03139
    0.02500    0.06360    -2.921    -2.300    0.04929
    0.05000    0.06940    -2.921    -2.300    0.05379
    0.07500    0.03470    -2.921    -2.300    0.02689
    0.1000    0.05780    -2.921    -2.300    0.04480
    0.1250    0.04630    -2.921    -2.300    0.03588
    0.1500    0.06360    -2.921    -2.300    0.04929
    0.1750    0.07520    -2.921    -2.300    0.05828

... 968 rows not shown.
See "n" rows via k_set('dispBrief', n).
** Other Props exist that are not displayed.
View other Props via dot syntax or functions k_summary or k_varView
```



## Props Inheritance

- Kornucopia's `k_units` variables hold more than just numbers & units, they hold other properties (`Props`):
  - `Props.Description`
  - `Props.ColNames`
  - `Props.ColComments`
  - `Props.RowNames`
  - `Props.RowComments`
  - `Props.UserData`
- Consider the calculation  $C = A .* B$ ; where `A` and `B` are potentially arrays with column names and other `Props`.
  - How is the software supposed to determine the new `Props` of `C`?
    - Kornucopia has rules known as *Props Inheritance*.
- The basic goal of *Props Inheritance* is to smartly create, when reasonably viable, `Props` for an output that are inherited from the arguments of any inputs when math or function calls are used.

In some cases, Props Inheritance is not able to smartly derive `Props` for the output and Kornucopia will issue a warning that they were not inherited.

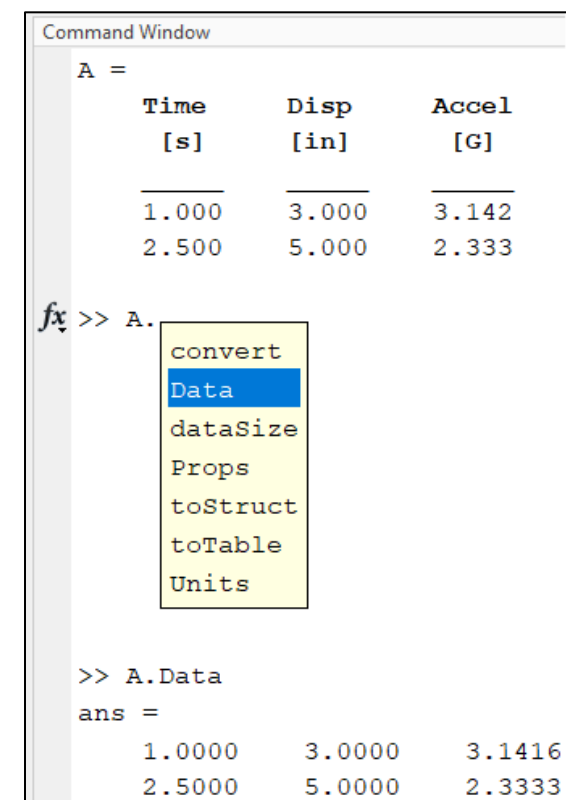
- The Kornucopia Help System provides full details on this topic.
- Turn off the warnings via:  
`k_set('inheritPropsAutoWarning', 'off')`

The screenshot shows the Kornucopia® ML™ Help System interface. The top navigation bar includes a home icon, the title 'Kornucopia® ML™ Help System', a search bar, and navigation icons. The left sidebar contains a table of contents with the following items: Kornucopia Home, Kornucopia Introduction & Features, Getting Started, Overview of Kornucopia, All About the k\_units Data Type (expanded), The k\_units Data Type, Syntax of Using k\_units Data Type, Units Library and Units Preferences, Units Signatures, Overloaded MATLAB Functions, Props Inheritance (highlighted), Kornucopia-Compatible Data Types, Comparing MATLAB Data Types to k\_units Data Types, Temperature and Frequency, Details on Kornucopia Features, Functions, Examples Library, Kornucopia Terminology, and Licensing and Notices. The main content area displays the 'Props Inheritance' page, which includes the Kornucopia logo, the title 'Kornucopia > All About the k\_units Data Type', and the heading 'Props Inheritance'. The text explains that the page discusses the issue of Props Inheritance, meaning the rules and resulting behavior of how Kornucopia handles inheritance issues related to the Props Properties of entities of type k\_units. It notes that inheritance refers to what, if any, meta-info Properties (Description, ColNames, ColComments, RowNames, RowComments, and UserData) stored in the Props of an input argument are inherited by (or copied to) an output argument of a function or operation. A note at the bottom states: 'Note: If you have come to this help page because you clicked on a "learn more" link in a warning stating "... the output result did NOT inherit any of the "Props" from the input argument(s).", this page will help you understand why the warning was issued. If you are looking to disable the warning, simply issue'.



## Accessing Just the Tabular Data from Variables of Type **k\_units**

- There are a few scenarios where you may need to quickly access just the numeric data from a variable of data type `k_units`. Doing so returns a MATLAB `double` data type.
  - When a function does not recognize the `k_units` data type.
  - Certain highly intensive looping calculations with a very large number of loops that computes slowly and cannot be improved by logical indexing or similar techniques.
- Typical **.Data** syntax examples to get just data portion of a variable of type `k_units`
  - `myData = A.Data` retrieves all rows and columns of the tabular data portion.
  - `myVec = A.Data(:,2)` retrieves all rows from 2<sup>nd</sup> column of the tabular data portion of the variable.
  - `myVec_b = A.Data('Accel')` retrieves the tabular data portion of the entire *Accel* column.
- **Warning** – Do not over-use `.Data`
  - Do not retreat to the “`.Data` approach” when Kornucopia gives you a Units-related error or similar. It is trying to help you! Retreating to the old ways of “just numbers” removes that guidance/safety.



Command Window

```
A =
```

Time [s]	Disp [in]	Accel [G]
1.000	3.000	3.142
2.500	5.000	2.333

```
fx >> A.
```

- convert
- Data**
- dataSize
- Props
- toStruct
- toTable
- Units

```
>> A.Data
```

```
ans =
```

1.0000	3.0000	3.1416
2.5000	5.0000	2.3333

# Array Size for Variables of Type **k\_units**

- Consider the file we recently imported with MATLAB's Importing Wizard as well as via Kornucopia's k\_readText function.
  - Notice that ALL the variables of Class `k_units` are shown in the MATLAB Workspace Variable list as `1x1 k_units`. MATLAB views the entire variable as "1 entity".
  - Notice the variable `tensionTest` and `tensionTest_T` both have a size of `976x3` listed. MATLAB knows they each have 976 rows and 3 columns of *primary* data.

- For variables of type `k_units`, use the `.dataSize` method to get its size.
  - This will give you the size of the *primary* Data stored in the variable.

```
Command Window
>> [r,c] = raw_2.dataSize
r =
    976
c =
     3
>> r = raw_2.dataSize(1)
r =
    976
>> c = raw_2.dataSize(2)
c =
     3
```

Workspace		
Name ▲	Value	Class
adv	1x1 struct	struct
Area	1x1 k_units	k_units
check	1	logical
colHeadStr	'Time'	char
dataDir	'./dataFiles'	char
DatasetHeader	3x1 cell	cell
delim	','	char
endStr	[]	double
fileName	'tensionTest.txt'	char
fName	'..\dataFiles\tensionT...	char
params	1x1 k_units	k_units
raw_1	1x1 k_units	k_units
raw_2	1x1 k_units	k_units
raw_3	1x1 k_units	k_units
rs1	1	double
rs2	0	double
rs3	[]	double
startStr	'Specimen dimensions'	char
tensionTest	976x3 double	double
tensionTest_T	976x3 table	table





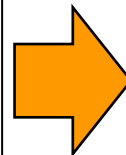
## Kornucopia-Compatible Data Types

- Any data type or (entity A) that can be successfully converted to a `k_units` data type by the following syntax is deemed a *Kornucopia-compatible data type*.

**B = k\_units(A)**

- Examples of entities that are Kornucopia-compatible
  - Any `k_units` data type is itself Kornucopia-compatible.
  - Any scalar or 2-D array which can be converted to a MATLAB double via `double(A)` is a Kornucopia-compatible data type. This will result in a `k_units` data type with dimensionless units.
  - Strings such as the following:

```
A = 'mm';  
B = k_units(A)  
  
C = '5*N/mm^2';  
D = k_units(C)  
  
E = '[1, 3, 7]*lbf/mm';  
F = k_units(E)  
  
G = '10';  
H = k_units(G)
```



Command Window

```
B = 1.000*mm  
D = 5.000*N/mm^2  
F =  
    [lbf/mm]    [lbf/mm]    [lbf/mm]  
    -----  
    1.000      3.000      7.000  
H = 10.00
```

- Also, variables of MATLAB data type `table` are Kornucopia-compatible when their primary data contents only holds data that can be converted to double data type via `double(A{:,:})`.

# Working with Arrays

- Depicted below are some typical **MATLAB** arrays

## A 3x3 **double** array

```
a =  
  
    25.1327    3.1416   18.8496  
     9.4248   15.7080   21.9911  
    12.5664   28.2743    6.2832
```

## A 2x3 **cell** array

```
f =  
  
2x3 cell array  
  
    'Some stuff'    [ 15.3000]    [ 1]  
[1x1 struct]    [3x3 double]    {2x1 cell}
```

## A 3x3 **table**

```
e =
```

Time	disp	NomStiff
25.133	3.1416	18.85
9.4248	15.708	21.991
12.566	28.274	6.2832

Note: The columns in table above may have Units assigned, but they are not displayed when echoed to Command Window with MATLAB table data type



# Working with Arrays

- Depicted below are some typical Kornucopia arrays

A 3x3 **k\_units** array (dimensionless units)

b =		
25.133	3.1416	18.85
9.4248	15.708	21.991
12.566	28.274	6.2832

A 3x3 **k\_units** array (w/ units, but no ColNames)

c =		
[sec]	[m]	[N/m]
25.133	3.1416	18.85
9.4248	15.708	21.991
12.566	28.274	6.2832

A 3x3 **k\_units** array (w/ units & ColNames)

d =		
Time	disp	Nom Stiff
[sec]	[m]	[N/m]
25.133	3.1416	18.85
9.4248	15.708	21.991
12.566	28.274	6.2832

Note for size of variable of type k\_units

- MATLAB function `size(b)` returns `[1,1]` for all variables of data type `k_units`.
- Use `k_units` method `.dataSize` to get size of primary tabular data within variable
  - `b.dataSize`



## Manipulating Rows and Columns of Variables of Type k\_units

- Examples of working with array elements, rows, or columns with a variable of type k\_units.
- We can reference rows and columns with traditional MATLAB index referencing syntax

```
A = raw(:, 2);
B = raw(1:10, :);
raw(:, 3) = raw(:, 3) - raw(1, 3);
```

- Kornucopia also supports using the column names (and row names if they exist) as shown below

```
D = raw('Load');
E = raw('Load : Strain');
raw(1:20, 'Stress') = -raw(1:20, 'Stress');
```

1<sup>st</sup> statement set D = to the entire Load column

2<sup>nd</sup> statement set E = to columns Load through Strain

last statement negated the 1<sup>st</sup> 20 rows of column Stress

Command Window

```
>> raw = raw_3
```

*Data we imported previously*

raw =  
tensionTest.txt  
=====

Time [sec]	Load [N]	Ext [mm]	Strain [%]	Stress [MPa]
0.000	0.04050	-2.921	-2.300	0.03139
0.02500	0.06360	-2.921	-2.300	0.04929
0.05000	0.06940	-2.921	-2.300	0.05379
0.07500	0.03470	-2.921	-2.300	0.02689
0.1000	0.05780	-2.921	-2.300	0.04480
0.1250	0.04630	-2.921	-2.300	0.03588
0.1500	0.06360	-2.921	-2.300	0.04929
0.1750	0.07520	-2.921	-2.300	0.05828

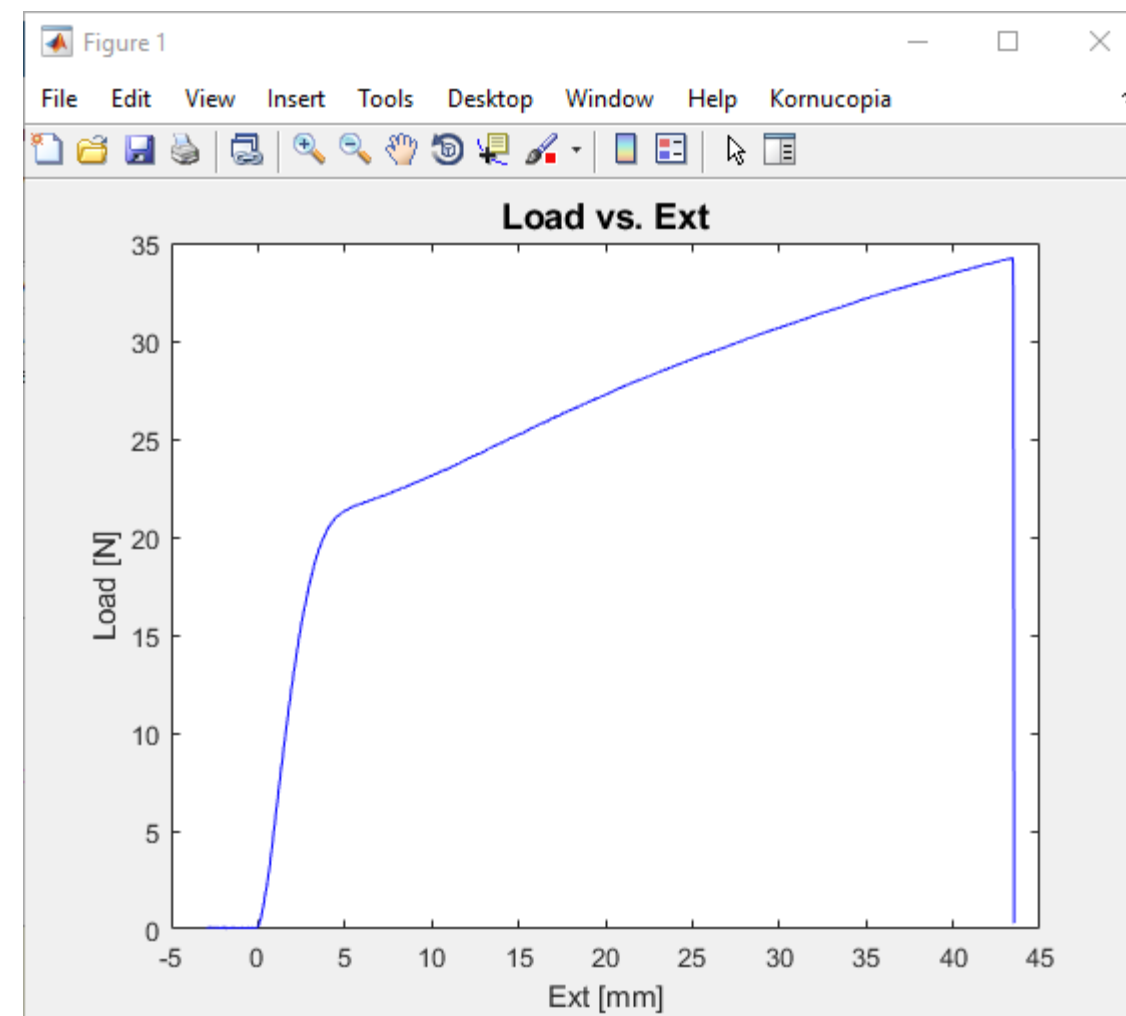
... 968 rows not shown.  
See "n" rows via k\_set('dispBrief', n).  
\*\* Other Props exist that are not displayed.  
View other Props via dot syntax or functions k\_summary or k\_varViewer.

## Quick Plot of a Variable

- Here is a quick plot of Load vs. Ext from our data

```
figure  
k_plot(raw_3('Ext, Load'))
```

- In the commands above we are:
  - Using the `k_plot` function.
  - Feeding in 2 columns of array `raw_3`, the `Ext` and `Load` columns.
- The `k_plot` function automatically uses `ColNames` and `Units` from the variable to label the plot.
  - We can easily modify these labels and other parameters of the plot through optional arguments of the `k_plot` function.
  - More about this shortly.



# Working with Temperature and Frequency

- Working with units of Temperature and Frequency requires special care and user awareness

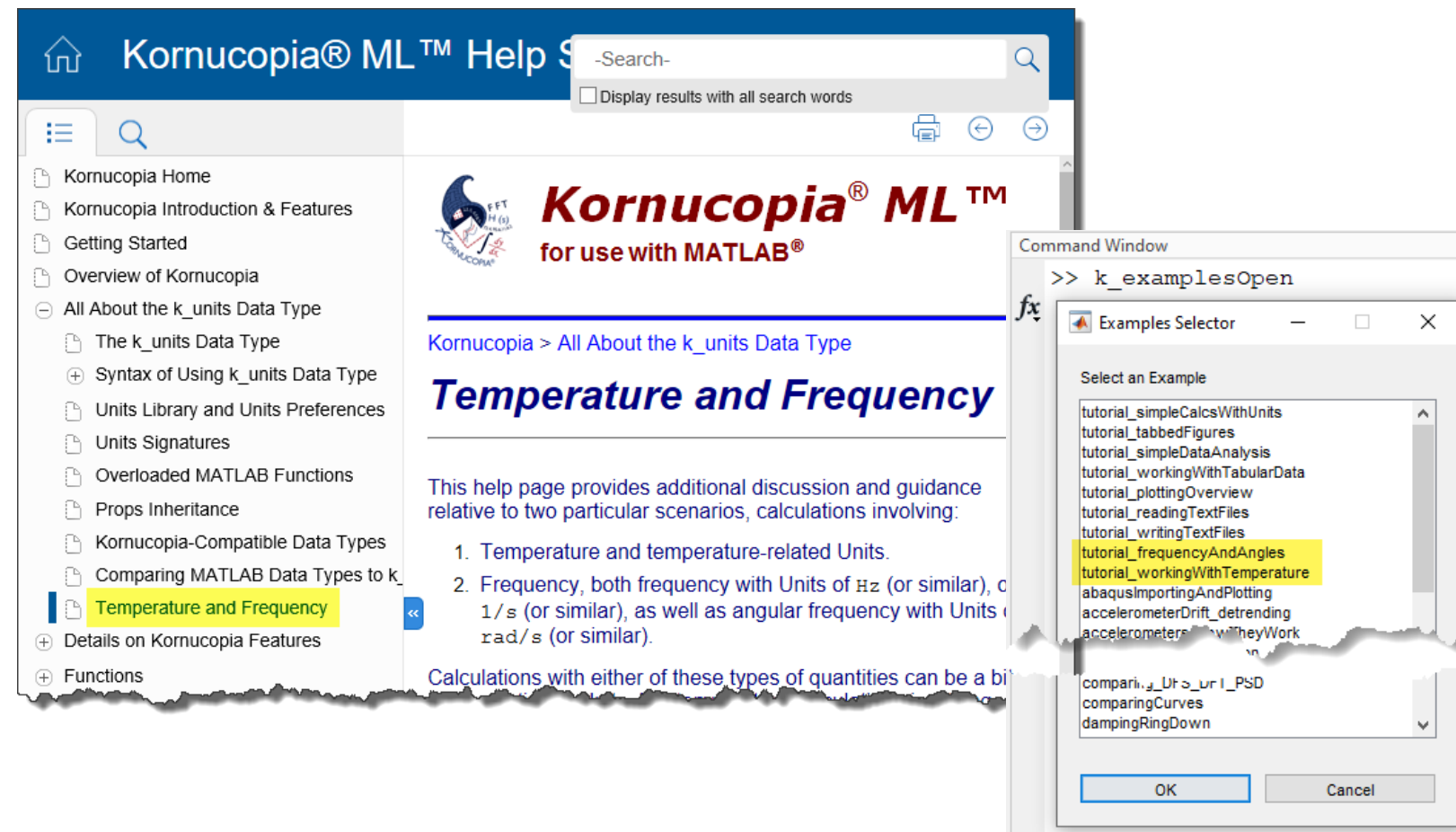
## Temperature

- Units of kelvin and rankine – No issues because they are absolute temperatures (multiplicative Units rules).
- Units of celsius and fahrenheit – They can be **affine** (include an additive offset in their conversion formula) or **non-affine** (no offset, as in delta temperature).
- Kornucopia's Units Engine has advanced artificial intelligence to help you properly navigate these issues in your calculations.

## Frequency

- Kornucopia's Units Engine has advanced artificial intelligence to help you properly navigate Hz, 1/s, and rad/s units in your calculations.

See Kornucopia Help System and Tutorials to learn more about how to properly work with Temperature and Frequency







## **More on Kornucopia Syntax**

## Where to Learn More

- Look at detailed discussions in the Kornucopia help system as shown for details on the various features listed to the right.
- In this training, we will focus on providing an *Awareness Overview* for the syntax features listed below:
  - ADV parameter
  - Flexible XY data input syntax
  - Smooth, Curley, and Square Braces

Kornucopia® ML™ Help System

-Search-

Kornucopia Home  
Kornucopia Introduction & Features  
Getting Started  
Overview of Kornucopia  
All About the k\_units Data Type  
**Details on Kornucopia Features**  
Gui & Argument-Oriented Functionality  
ADV Parameter  
Flexible XY Data Input Syntax  
Anchor Locations  
Reading and Writing Data Files  
Plotting and Viewing Data  
Other Features (DSP & More)  
Splines  
Merit and Error Measures  
Default Fill-ins as Empties  
myKornucopia Folder  
Setting MATLAB Path  
Customizing Kornucopia Behavior  
Transitioning from Past Kornucopia Versions  
Functions  
Alphabetical Listing of All Functions  
Function Categories

**Kornucopia® ML™**  
for use with MATLAB®

Kornucopia >

### Details on Kornucopia Features

This section contains a variety of help pages describing details of the various Kornucopia features.

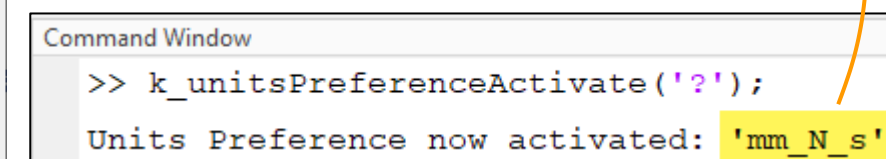
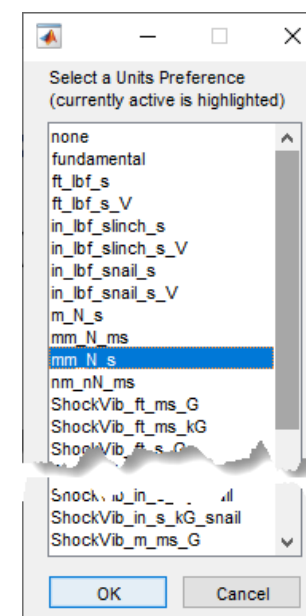
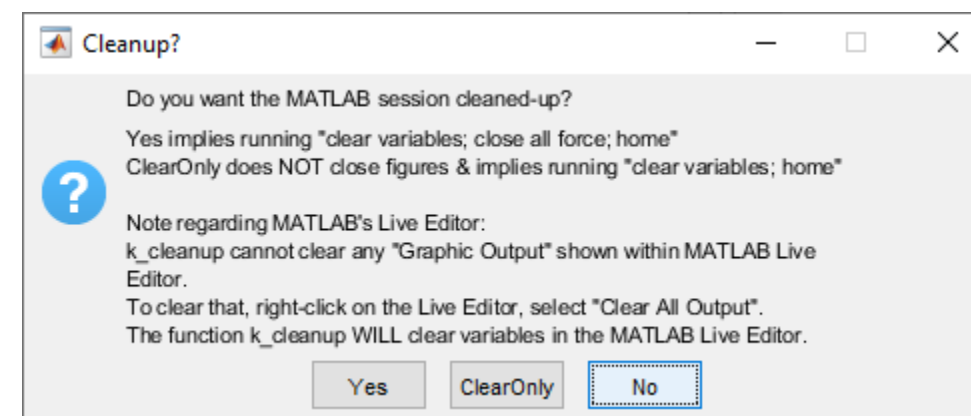
- [Gui & Argument-Oriented Functionality](#) - This help page describes Kornucopia's *gui-oriented* functionality in addition to its original *argument-oriented* functions.
- [ADV Parameter](#) - This help page describes the Kornucopia ADV features which strives to make functions robust and powerful, yet easy to use. This is done by distinguishing between primary input arguments and advanced ones (via the ADV parameter). This help page describes the ADV feature in general, how to determine what ADV options exist for a given function, and how to use the ADV functionality when you desire to override default settings.
- [Flexible XY Data Input Syntax](#) - This help pages discusses Kornucopia's flexible one or two argument XY data input options. In short, for any Kornucopia function that processes XY data input, you have the option of supplying the data as a single XY data array via one input argument or as separate X and Y data input arrays via two input arguments.
- [Anchor Location](#) - This page describes Kornucopia's flexible and unified syntax approach to specify how certain graphic objects are anchored (located) relative to their parent objects (also referred to as parent containers). The Kornucopia syntax is different than traditional MATLAB syntax. This help page provides an overview of the topic of *anchor locations* and explains the associated syntax options.



## A Few Typical Commands Often Found Near the Top of a Kornucopia Script

- These 4 commands (and a few others sometimes) are typically placed at the top of a Kornucopia script.
- `k_cleanup`
  - Creates interactive popup to control cleaning up the MATLAB environment.
- `k_unitsPreferenceActivate`
  - Activates the stated Units Preference.
    - Initially use `k_unitsPreferenceActivate('?')` for popup selection tool. Then copy the echoed value from Command Window into function call.
- `k_unitsVariables`
  - Define whatever Units Variables you need.
- `adv = k_adv()`
  - A list of advanced options. To be described next.

```
k_cleanup();  
k_unitsPreferenceActivate('mm_N_s');  
k_unitsVariables('mm, MPa, GPa, G')  
adv = k_adv();
```



## Kornucopia's ADV Parameter

- Nearly all Kornucopia functions have an optional last argument denoted in the function syntax as the ADV parameter.
- When supplied by the user, the ADV parameter allows you to modify advanced settings for the given function.
- The ADV parameters are specified in pairs of ADV **ParamName** (a keyword) and ADV **ParamVal**.
  - For a given pair, the **ParamName** must be first, followed by the **ParamVal**.
  - Various pairs of ADV parameters can be supplied in any order.
    - If the repeated ADV ParamName pairs are provided for a given function call, the last one will override earlier values.
- The various ADV parameter options are fully described in each function's help page.
- The next few slides shows some examples. Additional demonstrations of ADV use will be presented in the workshops at the end of the training.

Kornucopia > Functions > Function Categories > Derivatives and Integrals

### ***k\_derivative***

#### **Calling Syntax**

The function has two primary calling syntax options. *Note: the MATLAB function hint feature shows the first calling syntax option only.*

1. `derData = k_derivative(xData, yData, ADV)`

2. `xderData = k_derivative(xyData, [], ADV)`

# Example Showing ADV Parameter for function k\_regularize

Command Window

`f> >> k_regularize`

With cursor clicked on function name, hit F1-key

k\_regularize - MATLAB File Help

[View code for k\\_regularize](#)

## k\_regularize

`k_regularize` interpolates data to a constant sampling increment, making the data viable for use with DSP functions that require constant sampling rate.

Calling Syntax

```
[xDataReg, yDataReg] = k_regularize(xData, yData, ADV)
xyDataReg = k_regularize(xyData, [], ADV)
```

[CLICK FOR DETAILED HELP PAGE](#)

Command Window

`>> k_regularize('-adv')`

Quickly view ADV options

Optional ADV settings

Kornucopia function: `k_regularize(xData, yData, ADV)`

Note: to use all ADV defaults, skip ADV argument above

ParamName	ParamVal (default & other possibilities)
<code>direction</code>	<code>'auto'</code> (current default) <ul style="list-style-type: none"><li><code>'auto'</code></li><li><code>'pos'</code></li><li><code>'neg'</code></li></ul>
<code>independentCol</code>	<code>1</code> (current default) <ul style="list-style-type: none"><li><code>'** any valid ColName as string or col index a</code></li></ul>
<code>index</code>	<code>[]</code> (current default) <ul style="list-style-type: none"><li><code>'** any real number representing an index coun</code></li><li><code>'** any matrix of real numbers representing in</code></li><li><code>'** a single character string to represent an</code></li></ul>
<code>interpolateVia</code>	<code>'spline'</code> (current default) <ul style="list-style-type: none"><li><code>'linear'</code></li><li><code>'spline'</code></li><li><code>'cleanOnly'</code></li></ul>
<code>regularizeTo</code>	<code>'mean'</code> (current default) <ul style="list-style-type: none"><li><code>'mean'</code></li><li><code>'median'</code></li><li><code>'min'</code></li><li><code>'stdev1'</code></li><li><code>'stdev2'</code></li><li><code>'stdev3'</code></li></ul>
<code>relativeTol</code>	<code>0.00285</code> (current default) <ul style="list-style-type: none"><li><code>'** any positive number including 0. Value typ</code></li></ul>

## Default ADV options (from full Help page)

ADV

This is the optional advanced parameters argument. Omitting this argument results in default pairs of "ParamName, ParamVal" settings being used as listed below.

ADV installed defaults:

- `'direction', 'auto'`
- `'independentCol', 1`
- `'index', []`
- `'interpolateVia', 'spline'`
- `'regularizeTo', 'mean'`
- `'relativeTol', 0.00285`
- `'skipIncRelTol', 0.0003`
- `'splineEnds', 'clamped'`
- `'warnings', 'on'`

See [k\\_regularize ADV Details](#) for full details on these optional ADV arguments.

Additional detailed descriptions for each ADV option exist on the full Help page at this link



# Example of using ADV Parameter

## Function call taking default ADV options

```
reg = k_regularize(raw);
```

## Function call examples overriding default ADV options

```
reg_a = k_regularize(raw, [], ...
    'interpolateVia', 'linear', ...
    'regularizeTo', 'stdev1');
```

```
reg_b = k_regularize(raw, [], ...
    'relativeTol', 0.005, ...
    'independentCol', 'Ext', ...
    'splineEnds', 'zero');
```

```
reg_c = k_regularize(raw, [], ...
    'relativeTol', 0.005, ...
    'independ', 3, ...
    'spline', 'zero');
```

All ADV pairs are **ParamName, ParamVal**.

Sometimes ParamVals are strings, sometimes they are numeric, other times (not here) they are cell arrays or other data types.

You can also "type short" any ParamName provided it is still unique for that function.

Command Window

`>> k_regularize`

With cursor clicked on function name, hit F1-key

k\_regularize - MATLAB File Help

[View code for k\\_regularize](#)

k\_regularize

k\_regularize interpolates data to a constant sampling increment, making the data viable for use with DSP functions that require constant sampling rate.

Calling Syntax

[xDataReg, yDataReg] = k\_regularize(xData, yData, ADV)

xyDataReg = k\_regularize(xyData, [], ADV)

[CLICK FOR DETAILED HELP PAGE](#)

Command Window

`>> raw = raw_3`

Data we imported previously

raw =

tensionTest.txt

=====

Time [sec]	Load [N]	Ext [mm]	Strain [%]	Stress [MPa]
0.000	0.04050	-2.921	-2.300	0.03139
0.02500	0.06360	-2.921	-2.300	0.04929
0.05000	0.06940	-2.921	-2.300	0.05379
0.07500	0.03470	-2.921	-2.300	0.02689
0.1000	0.05780	-2.921	-2.300	0.04480
0.1250	0.04630	-2.921	-2.300	0.03588
0.1500	0.06360	-2.921	-2.300	0.04929
0.1750	0.07520	-2.921	-2.300	0.05828

... 968 rows not shown.

See "n" rows via k\_set('dispBrief', n).

\*\* Other Props exist that are not displayed.

View other Props via dot syntax or functions k\_summary or k\_varViewer.





# The k\_adv() Function

- As stated earlier, we often define the variable **adv = k\_adv()** in the 1<sup>st</sup> section of a script.
- This creates a single variable, **adv**, of MATLAB type **struct**, whose fieldnames are the same as all the Kornucopia functions. Within each field is that function's ADV options (their ParamNames and all the possible ParamVals for each ParamName).

```
k_cleanup();  
k_unitsPreferenceActivate('mm_N_s');  
k_unitsVariables('mm, MPa, GPa, G')  
adv = k_adv(); ←
```

Echo the adv variable and using dot-tab syntax to view adv variable

```
Command Window  
>> adv  
adv =  
  struct with fields:  
  
          k_2help: [1x1 struct]  
k_abqExportToViewer_gui: [1x1 struct]  
  k_abqOdbHistoryImport: [1x1 struct]  
    k_abqOdbHistoryList: [1x1 struct]  
    k_abqOdbImport_gui: [1x1 struct]  
    k_abqOdbPickItems: [1x1 struct]  
    k_abqOdbVirtualCS: [1x1 struct]  
          k_arcLength: [1x1 struct]
```

```
Command Window  
fx >> adv.k_  
k_2help  
k_abqExportToViewer_gui  
k_abqOdbHistoryImport  
k_abqOdbHistoryList  
k_abqOdbImport_gui  
k_abqOdbPickItems  
k_abqOdbVirtualCS  
k_arcLength
```

From before, our list of all 125 Kornucopia functions

```
Command Window  
fx >> k_  
k_2help  
k_abqExportToViewer_gui  
k_abqOdbHistoryImport  
k_abqOdbHistoryList  
k_abqOdbImport_gui  
k_abqOdbPickItems  
k_abqOdbVirtualCS  
k_adv
```



# The `k_adv()` Function (Example Use for `k_regularize`)

- Using dot-tab syntax, it is easy to narrow down to the function, ParamName, and finally ParamVal of interest to use.
- This makes it easy to explore and apply ADV options as needed to various function calls.

```
Command Window
fx >> adv.k_
    k_pulseCreate
    k_pulseDetails
    k_pulseDuration
    k_readTaggedText
    k_readText
    k_readTextToCellstr
    k_regularize
    k_resample
```

```
Command Window
>> adv.k_regularize
ans =
    struct with fields:
        direction: [1x1 struct]
        independentCol: [1x1 struct]
        index: [1x1 struct]
        interpolateVia: [1x1 struct]
        regularizeTo: [1x1 struct]
        relativeTol: [1x1 struct]
        skipIncRelTol: [1x1 struct]
        splineEnds: [1x1 struct]
        warnings: [1x1 struct]
```

```
Command Window
>> adv.k_regularize.interpolateVia
ans =
    struct with fields:
        linear: {'interpolateVia' 'linear'}
        spline: {'interpolateVia' 'spline'}
        cleanOnly: {'interpolateVia' 'cleanOnly'}
        CURRENTval: {'interpolateVia' 'spline'}
```

```
Command Window
>> adv.k_regularize.interpolateVia.linear
ans =
    1x2 cell array
        {'interpolateVia'}    {'linear'}
```



## The `k_adv()` Function (Example Use for `k_regularize`)

- The following two statements defining variable `reg_a` are equivalent.
  - The 1<sup>st</sup> one uses char strings for each ParamName.
  - The 2<sup>nd</sup> statement used the `adv` variable approach for the `interpolateVia` ParamName and its ParamVal.

```
reg_a = k_regularize(raw, [], ...  
    'interpolateVia', 'linear', ...  
    'regularizeTo', 'stdev1');  
  
reg_a = k_regularize(raw, [], ...  
    adv.k_regularize.interpolateVia.linear, ...  
    'regularizeTo', 'stdev1');
```

### Remember

```
Command Window  
  
>> adv.k_regularize.interpolateVia.linear  
ans =  
    1×2 cell array  
    {'interpolateVia'} {'linear'}
```

### Bottom line

- The ADV syntax is very flexible, and you can utilize various forms as desired.
- Using an `adv` variable as demonstrated allows you a quick, selectable way to pick ADV options for various functions.

## Flexible XY Data Input

- Nearly all Kornucopia functions that expect XY data support flexible input of such data as one argument or two arguments.
- Two-argument data input (`xData`, `yData`)**
  - This input syntax style is explicit, you enter separate arguments for the `xData` and `yData` inputs. The `xData` must be a single column vector and the `yData` can be one or more columns of data.
- One-argument data input (`xyData`)**
  - This input syntax style allows the user to enter a single input argument that contains both the `xData` and the `yData`, all within a single multi-column array. With this input style, the determination of which column is the independent X vector is determined by the given function's ADV ParamVal associated with ParamName `'independentCol'`. Its default setting is the value 1, meaning the first column of `xyData`.
    - Please note: When using the one-argument `xyData` input style, you may need to enter the empty matrix, `[]`, in the location of the `yData` argument if you intend to specify additional arguments beyond the argument location of the `yData` argument, such as any ADV arguments, or for functions with more than just `xData` and `yData` arguments.

Kornucopia > Functions > Function Categories > Fitting and Interpolation

### ***k\_fitFunc***

This function creates a *curve-fit* function for a supplied data-set based on a user-supplied fit-basis equation. Supported fit-basis forms include polynomial, constrained polynomial, linear combination of basis functions, and general nonlinear basis functions.

Summary of function's features:

- The function is useful when you desire to create a function that can represent, in a best-fit sense, a XY curve defined by a set of discrete points.
  - The curve-fit function computed from the underlying data is NOT guaranteed to exactly go through any of the underlying data points, but rather is a best-fit representation of the data points for the given basis equation applied to the underlying data.

### **Calling Syntax**

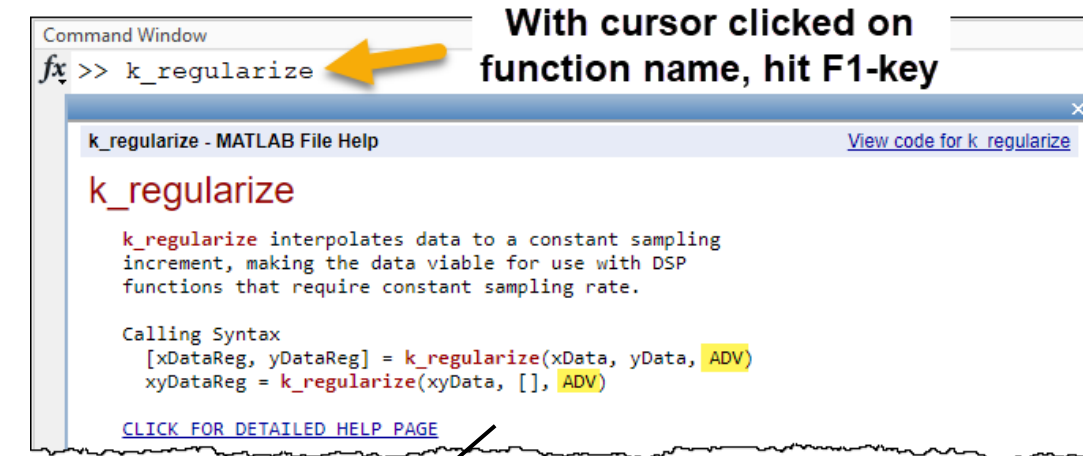
The function has two primary calling syntax options. *Note: the MATLAB function hint feature shows the first calling syntax option only.*

- `Fy = k_fitFunc(xData, yData, fitBasis, ADV)`
- `Fxy = k_fitFunc(xyData, [], fitBasis, ADV)`

The second calling syntax option presented above takes advantage of Kornucopia's [Flexible XY Data Input Syntax](#).

## Flexible XY Data Input – Example with `k_regularize`

- Starting with variables
  - `t` (nx1 vector containing time)
  - `v` (nx3 array of velocity values)
  - `Raw = [t, v]` (Raw is a multi-column array)
  - `goofy = [v, t]`
- MATLAB's Quick Argument Help shows the function arguments as:
  - `k_regularize(xData, yData, ...)`
- F1 brief help shows the function argument syntax options as:
  - `[xDataReg, yDataReg] = k_regularize(xData, yData, ADV)`
  - `xyDataReg = k_regularize(xyData, [], ADV)`



## Flexible XY Data Input – Example with `k_regularize`

For the different styles of input variables `t`, `v`, `raw`, and `goofy`, typical user syntax use would be:

- **Two-argument input**

```
[tReg, vReg] = k_regularize(t, v)
```

- **One-argument xyData input**

```
reg = k_regularize(raw)
```

```
goofyReg = k_regularize(goofy, [], ...  
    'independentCol', 4)
```

- Note: for the "goofy" case, the output `goofyReg` is ordered the same as the original input data's order.

### Bottom line

- For typical Kornucopia workflows, most users find the one-argument xyData input format the easiest to utilize.



## Smooth, Curley, & Square Braces Plus , ; and :

*Lions, tigers, and bears, oh my!*

- **Smooth braces `()`**
  - Used to reference an element in the array, as in `A(3,2)`. The result is generally the same data type.
- **Curly braces `{}`**
  - **Tip** – Curly braces step in and get/place content from/in the element.
  - `Cell` arrays: used to reference content of elements in cell arrays `B{3,2}`. This would return whatever was in location `3,2` of the cell array.
    - Note: smooth braces `B(3,2)` references the elements themselves, returning a cell array.
  - MATLAB `table`: used to reference content of primary tabular data of variable `C{3,2}`.
  - Kornucopia `k_units`: only used on left hand side (LHS) of equals, means to keep current `Props` and `Units` and only place the contents of RHS into variable (with appropriate Units conversions).
- **Square brackets `[]`**
  - Used to concatenate scalars, vectors and arrays to make larger arrays.
  - Used to delete rows or columns of an array, or to make an entire variable empty.
- **Comma, semicolon and colon `, ; :`**
  - They are used in several syntax scenarios in MATLAB. See MATLAB help for various uses.
  - In addition to MATLAB interpretations, they are delimiters and range specifier for Kornucopia string-lists.

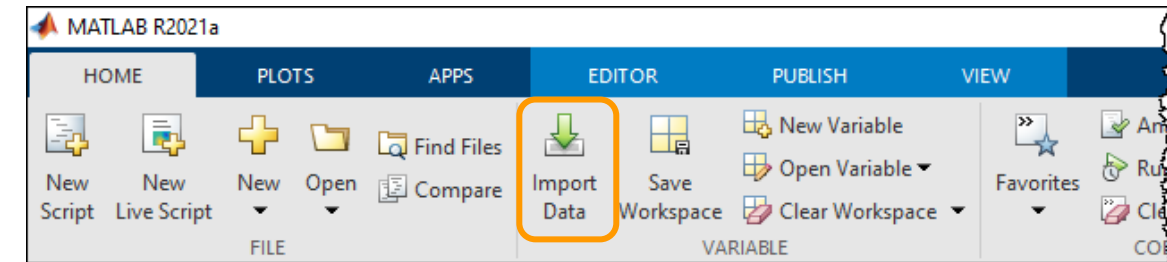


# **Importing & Exporting Data Files with Kornucopia Functions**

# Various Ways to Import/Export Data with MATLAB and Kornucopia

- **MATLAB native features**

- Several functions and import wizard to text and spreadsheet data.
- Functions `save` and `load` for saving/loading workspace variables to a file.
  - This works with `k_units` data type variables too!



- **Kornucopia import functions**

- General ascii text

- `k_readText`
- `k_readTextToCellstr`
- `k_unpackHeader`

- Special formats

- Abaqus ODB: `k_abqOdbImport_gui`
- SD Impax: `k_sdImpaxImport` & `k_sdImpaxRtmImport`

- **Kornucopia export functions & methods**

- `k_writeText`
- For a variable `A` of type `k_units`, the following methods:
  - `A.toStruct`
  - `A.toTable`

# Kornucopia ML can easily read a broad array of Text files

1

File with just numeric data

*DCB\_benchmark\_mm\_N.xy*

```
0,0
0.390652,127.530513709519
0.419608,123.037809878105
0.44958,118.856481559761
0.480822,114.942046538331
0.512826,111.294504813818
0.5461,107.869374170067
0.58039,104.622172390927
0.615442,101.59738169255
0.651764,98.7505198587831
0.689102,96.0371046734742
0.727456,93.4571361366231
```

2

Single data set with header sections & tabular data

*tensionTest.csv*

```
Material: polyA
Tested By: Joe Smith
Date: March 7, 2009
Test Type: Tension with initial slack

Specimen dimensions
thick (mil): 4.0
width (in): 0.5
gage length (in): 5.000

Time, Load, Ext
sec, N, mm
0.0000, 0.0405, -2.9210
0.0250, 0.0636, -2.9210
0.0500, 0.0694, -2.9210
0.0750, 0.0347, -2.9210
```

3

File with beginning and ending markers for each dataset

*multiSpecimens\_MTS.txt*

```
BeginSample
_SampleName, "ProjectBlueFish.asm"
_MethodName, "ASTM method XYZ.asm"
_col1, "col2", "col3", "col4", "col5", "col6"
_SampleID, "blue_MD-Dir", ""
_SampleInfo, "Tested by", "J. Smith"
BeginSpecimen
_AdjGage, 0.253, "m"
_Area, 0.890, "cm^2"
BeginData
_Time, Load, Extension, SlackExt, Sensor 1, Sensor 2
"s", "gf", "m", "m", "%", "%"
0.13600, 2.318, 0.000, -0.02468, 0.00404, -2.46793
0.14000, 2.299, 0.000, -0.02468, 0.00401, -2.46793
0.14400, 2.476, 0.000, -0.02468, 0.00431, -2.46793
...
1.88400, 13722.152, 0.141, 0.11642, 23.90619, 11.64224
1.88800, 9378.264, 0.141, 0.11676, 16.33844, 11.67631
1.89000, 6354.958, 0.142, 0.11694, 11.07135, 11.69353
EndData
EndSpecimen
...
A bunch more specimens
...
BeginSpecimen
_AdjGage, 0.250, "m"
_Area, 0.889, "cm^2"
BeginData
_Time, Load, Extension, SlackExt, Sensor 1, Sensor 2
"s", "gf", "m", "m", "%", "%"
0.12800, 2.867, 0.000, -0.02418, 0.00499, -2.41846
0.13200, 3.109, 0.000, -0.02418, 0.00542, -2.41846
...
1.96000, 4952.340, 0.150, 0.12552, 8.62777, 12.55221
1.96000, 4952.340, 0.150, 0.12552, 8.62777, 12.55221
EndData
EndSpecimen
EndSample
```

Grouping key: { = specimen | = info section [ = primary data

4

File with only beginning markers for each dataset

*multiCurves\_style1.ASC*

```
Project: BigShot
Testing Method: PRS22-5
Tested by: J. Smith
Date: 5/17/2001

Time[s]: Load [gf]: Disp [in]: Dymac Disp [in]: Disp [in]:
0.0000000000: 0.005009: -0.000732: 0.000000: -0.000733:
0.0333333333: 0.005009: -0.000732: 0.000000: -0.000733:
0.0666666667: 0.005009: -0.000732: 0.000000: -0.000733:
0.1000000000: 0.005009: -0.000732: 0.000000: -0.000733:
...
49.0666666667: 84.534256: 0.368347: 0.003967: 0.364380:
49.1000000000: 84.550400: 0.368591: 0.003968: 0.364623:
49.1333333333: 84.550400: 0.368820: 0.003968: 0.364852:
...
Time[s]: Load [gf]: Disp [in]: Dymac Disp [in]: Disp [in]:
0.0000000000: -0.107955: -0.000015: -0.000005: -0.000010:
0.0333333333: -0.107955: -0.000015: -0.000005: -0.000010:
0.0666666667: -0.107955: -0.000015: -0.000005: -0.000010:
...
A bunch more datasets
...
Time[s]: Load [gf]: Disp [in]: Dymac Disp [in]: Disp [in]:
0.0000000000: -0.075679: -0.000351: -0.000004: -0.000347:
0.0333333333: -0.075679: -0.000351: -0.000004: -0.000347:
0.0666666667: -0.075679: -0.000351: -0.000004: -0.000347:
...
49.8666666667: 131.946808: 0.368698: 0.006192: 0.362506:
49.9000000000: 131.914520: 0.368973: 0.006191: 0.362782:
49.9333333333: 131.882248: 0.369232: 0.006189: 0.363043:
49.9666666667: 131.866119: 0.369492: 0.006189: 0.363303:
```

Grouping key: | = info section [ = data per specimen.

## Kornucopia functions

- k\_readText
- k\_unpackHeader
- k\_readTextToCellstr

Finite Element Input Deck (a text file)

*Model1-1s.inp*

```
** Units: Time=sec, Length=mm , mass=Mg (mega grams)
** ** Force=N, Stress = MPa, Density = Mg/(mm^3)
*Heading
** Job name: Model-1s Model name: Model-1s
** Generated by: Abaqus/CAE 6.9-1
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
**
*Part, name=bottom
*Node
1, 0., 0., 0., 0.
2, 0., 0., 1.80836117, 0.
3, 0., 0., 1.61672235, 0.
...
*Element, type=C3D8R
1, 19, 20, 26, 25, 1, 2, 8, 7
2, 20, 21, 27, 26, 2, 3, 9, 8
3, 21, 22, 28, 27, 3, 4, 10, 9
...
** INTERACTIONS
**
** Interaction: CP-1-bottom-1-top-1
*Contact Pair, interaction=IntProp-Standard
CP-1-top-1, CP-1-bottom-1
** Interaction: LeadingEdge
*Contact Pair, interaction=IntProp-Standard
Pick's Ser 50 CNS CP-1-top-1
```



# The k\_readText Function

- Function designed to smartly and flexibly import a large variety of text files ranging from:
  - Simple table of pure numeric data
  - Multi-sectioned file with lots of header regions & tabular regions.

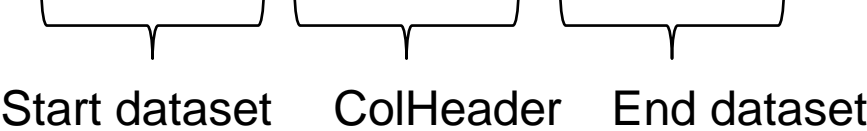
- Syntax

- Simple for pure numeric file

```
A = k_readText(fName, delim)
```

- Flexible for more complicated files

```
A = k_readText(fName, delim, ...  
               sStr, rs1, cStr, rs2, eStr, rs3, ADV)
```



**Search Identifiers for Regions**  
(Marker strings & row shift factors)

Advanced options

File with just numeric data

DCB\_benchmark\_mm\_N.xy

```
0,0  
0.390652,127.530513709519  
0.419608,123.037809878105  
0.44958,118.856481559761  
0.480822,114.942046538331  
0.512826,111.294504813818  
0.5461,107.869374170067  
0.58039,104.622172390927  
0.615442,101.59738169255  
0.651764,98.7505198587831  
0.689102,96.0371046734742  
0.727456,93.4571361366231
```

File with beginning and ending markers for each dataset

multiSpecimens\_MTS.txt

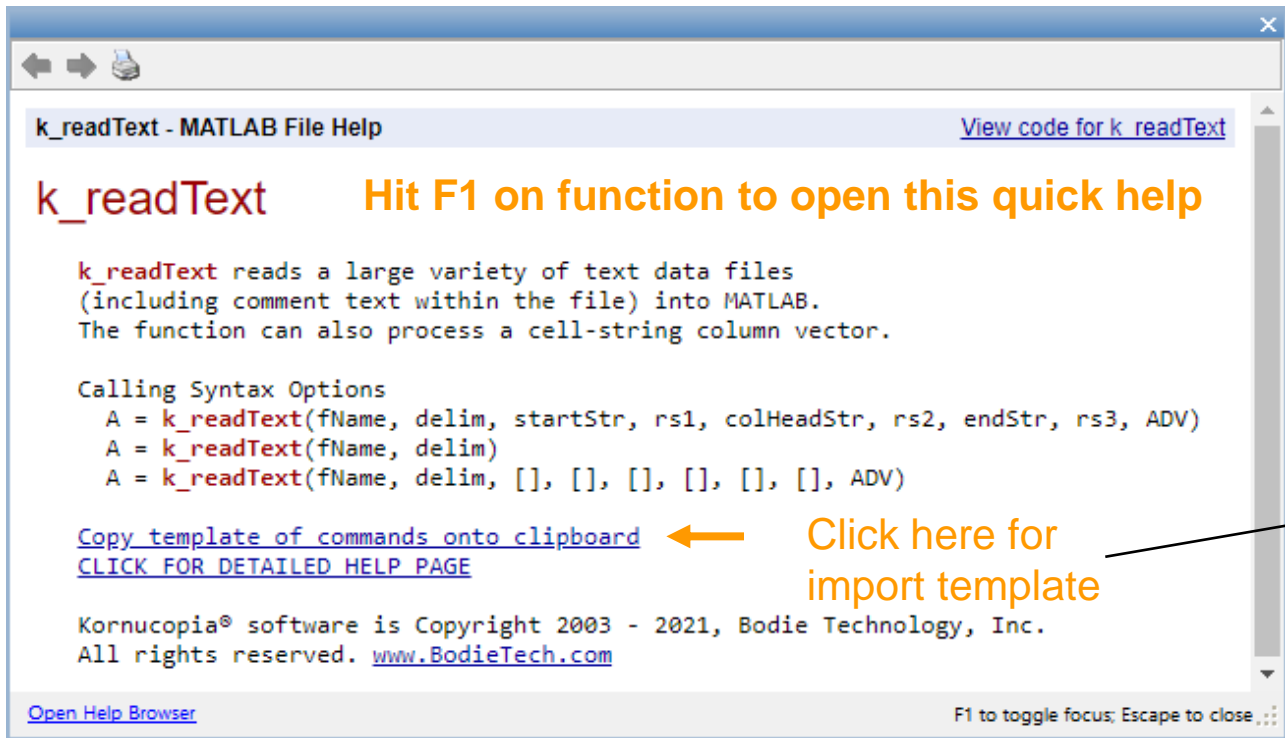
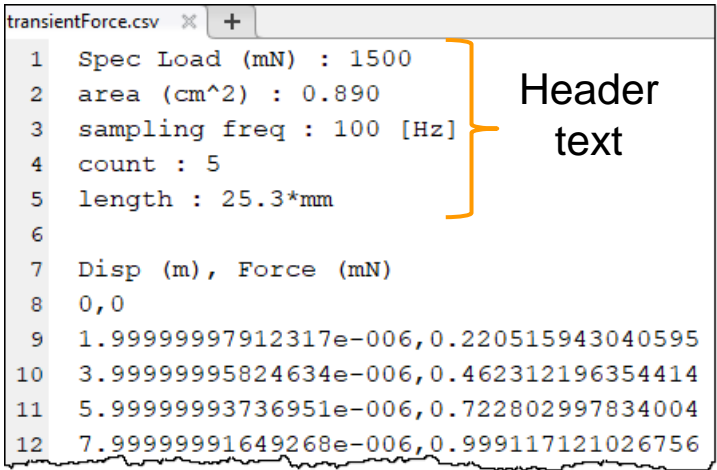
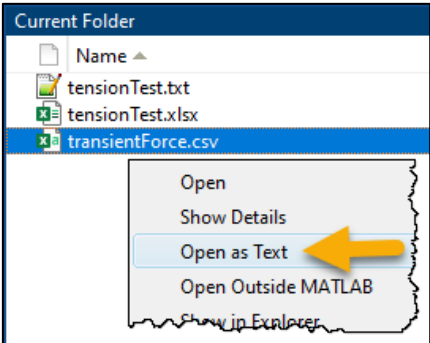
```
BeginSample  
_SampleName, "ProjectBlueFish.nss"  
_MethodName, "ASTM method XYZ.nsa"  
_col1, "col2", "col3", "col4", "col5", "col6"  
_SampleID, "blue_MD-Dir", ""  
_SampleInfo1, "Tested by", "J. Smith"  
BeginSpecimen  
AdjGage, 0.253, "n"  
Area, 0.890, "cm^2"  
BeginData  
Time, Load, Extension, SlackExt, Sensor 1, Sensor 2  
"s", "gf", "n", "n", "%", "%"  
0.13600, 2.318, 0.000, -0.02468, 0.00404, -2.46793  
0.14000, 2.299, 0.000, -0.02468, 0.00401, -2.46793  
0.14400, 2.476, 0.000, -0.02468, 0.00431, -2.46793  
...  
1.88400, 13722.152, 0.141, 0.11642, 23.90619, 11.64224  
1.88800, 9378.264, 0.141, 0.11676, 16.33844, 11.67631  
1.89000, 6354.958, 0.142, 0.11694, 11.07135, 11.69353  
EndData  
EndSpecimen  
...  
A bunch more specimens  
...  
BeginSpecimen  
AdjGage, 0.250, "n"  
Area, 0.889, "cm^2"  
BeginData  
Time, Load, Extension, SlackExt, Sensor 1, Sensor 2  
"s", "gf", "n", "n", "%", "%"  
0.12800, 2.867, 0.000, -0.02418, 0.00499, -2.41846  
0.13200, 3.109, 0.000, -0.02418, 0.00542, -2.41846  
...  
1.96000, 4952.340, 0.150, 0.12552, 8.62777, 12.55221  
1.96000, 4952.340, 0.150, 0.12552, 8.62777, 12.55221  
EndData  
EndSpecimen  
EndSample
```

Grouping key: { = specimen    | = info section    [ = primary data



# Importing CSV File with `k_readText` Function

- Consider the file `dataFiles/transientForce.csv`
  - `dataFiles` folder is in the training docs you downloaded.
- First open file as text in MATLAB editor to see its contents.
- Next, we use `k_readText` function to import the file



```
% The template below is set to read a local file named 'myFile.dat'
% that is comma delimited with the following section identifications:
% o The data set section begins one row below the row with the string
%   'Specimen dimensions' contained within it.
% o The column header section begins at the row with the string
%   'Time' within it.
% o No specific delimiter is specified for the end of the data set
%   section. Internal automatic logic will be used.
%
% If ADV options are needed, just add them after the rs3 argument in
% the usual way.

dataDir = ''; % Place path to file here (leave as-is if local file)
fileName = 'myFile.dat'; % Use string '?' for file selection popup
fName = fullfile(dataDir, fileName);
delim = ',';
startStr = 'Specimen dimensions';
rs1 = 1;
colHeadStr = 'Time';
rs2 = 0;
endStr = [];
rs3 = [];
raw = k_readText(fName,delim,startStr,rs1,colHeadStr,rs2,endStr,rs3);
```





## Importing CSV File with `k_readText` Function (cont)

- Modify a few values accordingly
- Then run the section to import the file.
  - The Kornucopia function will import the entire file, including header info!

```
dataDir = '../dataFiles';
fileName = 'transientForce.csv';
fName = fullfile(dataDir, fileName);
delim = ',';
startStr = 'Spec Load';
rs1 = 0;
colHeadStr = 'Disp';
rs2 = 0;
endStr = [];
rs3 = [];
A = k_readText(fName, delim, startStr, rs1, colHeadStr, rs2, endStr, rs3);
```

```
Command Window
>> A

A =
    transientForce.csv
    =====
           Disp           Force
           [m]           [mN]

    0.000         0.000
    2.000e-06     0.2205
    4.000e-06     0.4623
    6.000e-06     0.7228
    8.000e-06     0.9991
    1.000e-05     1.288
    1.200e-05     1.587
    1.400e-05     1.892

... 9993 rows not shown.
See "n" rows via k_set('dispBrief', n).
** Other Props exist that are not displayed.
View other Props via dot syntax or functions k_summary or k_varViewer.
```

```
Command Window
>> A.Props.UserData
ans =
    struct with fields:
        FileInfo: {3x1 cell}
        MainFileHeader: []
        DatasetHeader: {5x1 cell}
        DatenumID: '736222.264317129622213542461395'
        DatasetNum: 1

>> A.Props.UserData.DatasetHeader
ans =
    5x1 cell array
    {'Spec Load (mN) : 1500' }
    {'area (cm^2) : 0.890' }
    {'sampling freq : 100 [Hz]' }
    {'count : 5' }
    {'length : 25.3*mm' }
```



# Use the `k_unpackHeader` Function to Map Header Into `k_units` Data Type Too!

1

```
transientForce.csv  X  +
1  Spec Load (mN) : 1500
2  area (cm^2) : 0.890
3  sampling freq : 100 [Hz]
4  count : 5
5  length : 25.3*mm
6
7  Disp (m), Force (mN)
8  0,0
9  1.99999997912317e-006,0.220515943040595
10 3.99999995824634e-006,0.462312196354414
11 5.99999993736951e-006,0.722802997834004
12 7.99999991649268e-006,0.999117121026756
```

Header  
text

2

Import via `k_readText`.

```
Command Window
>> A

A =
transientForce.csv
=====
      Disp      Force
      [m]      [mN]
-----
      0.000      0.000
2.000e-06  0.2205
4.000e-06  0.4623
6.000e-06  0.7228
8.000e-06  0.9991
1.000e-05  1.288
1.200e-05  1.587
1.400e-05  1.892

... 9993 rows not shown.
```

3

`k_readText` nicely isolated the dataset header text.  
And `k_unpackheader` will map it into a useful array.

```
headerTxt = A.Props.UserData.DatasetHeader;
B = k_unpackHeader(headerTxt, ':');
```

```
Command Window
>> headerTxt

headerTxt =
5x1 cell array

    {'Spec Load (mN) : 1500'    }
    {'area (cm^2) : 0.890'      }
    {'sampling freq : 100 [Hz]' }
    {'count : 5'                }
    {'length : 25.3*mm'         }
```

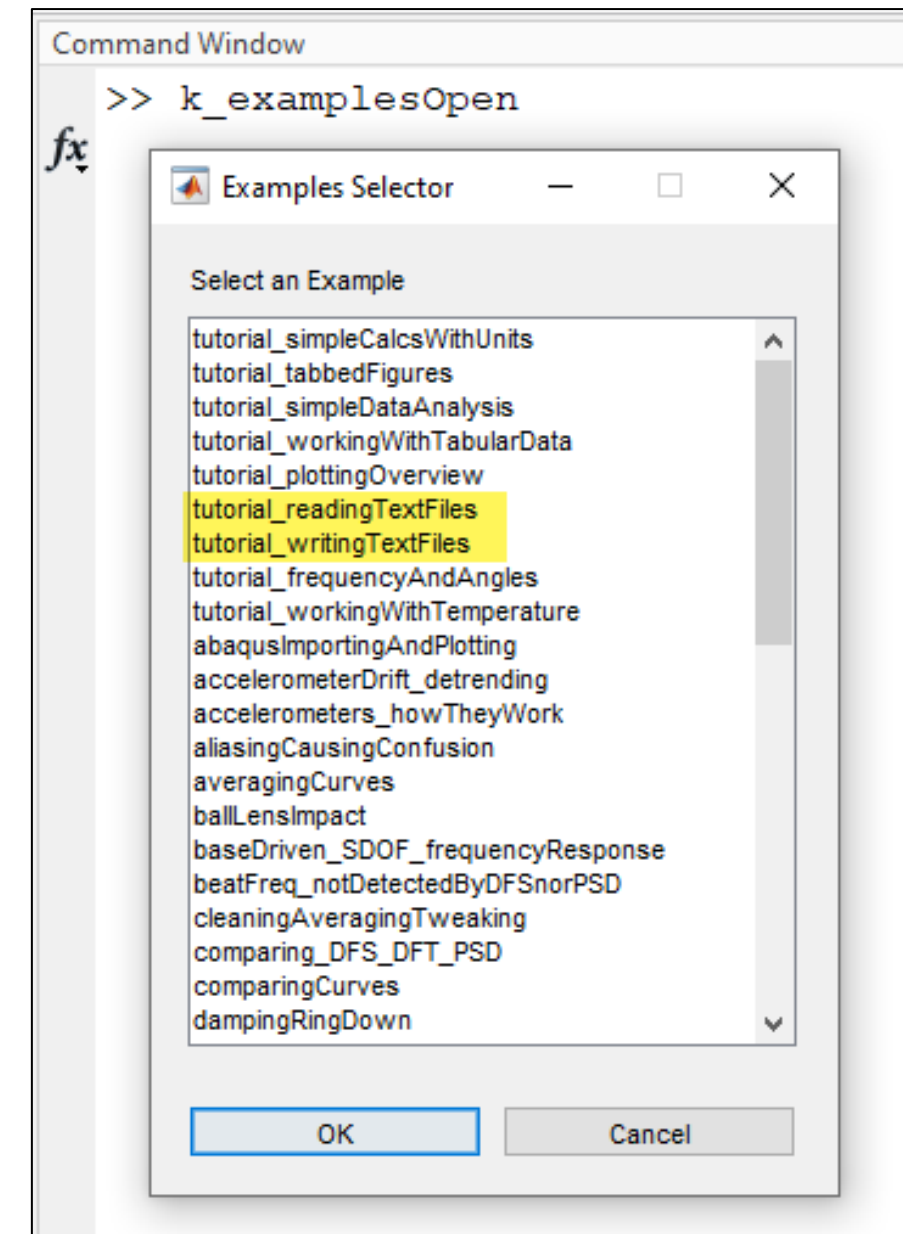
```
>> B

B =
      Spec Load      area      sampling freq      count      length
      [mN]      [cm^2]      [Hz]
-----
1500      0.8900      100.0      5.000      25.30
```



## More on Reading and Writing Text Data Files

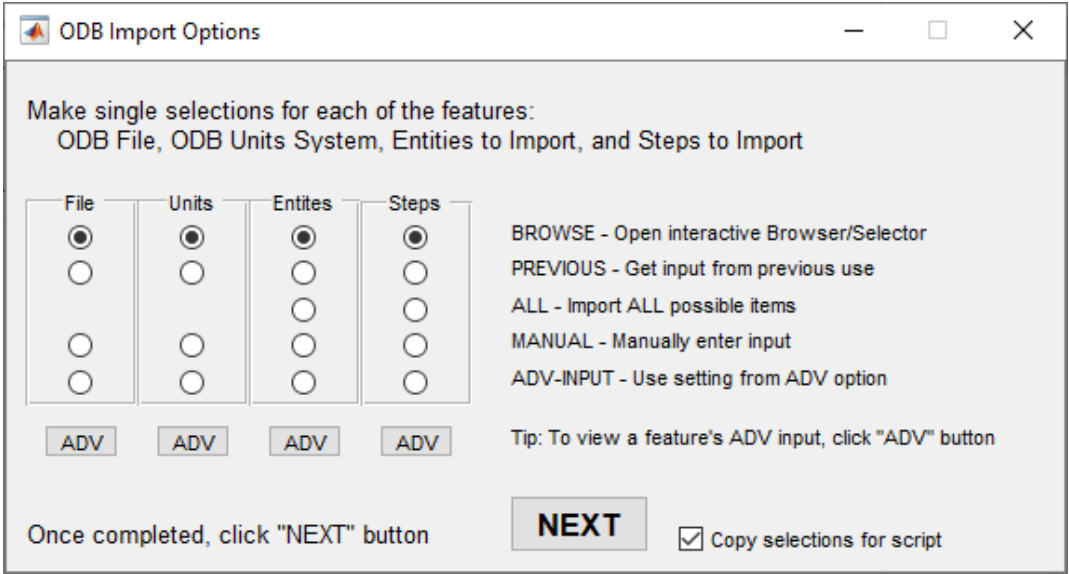
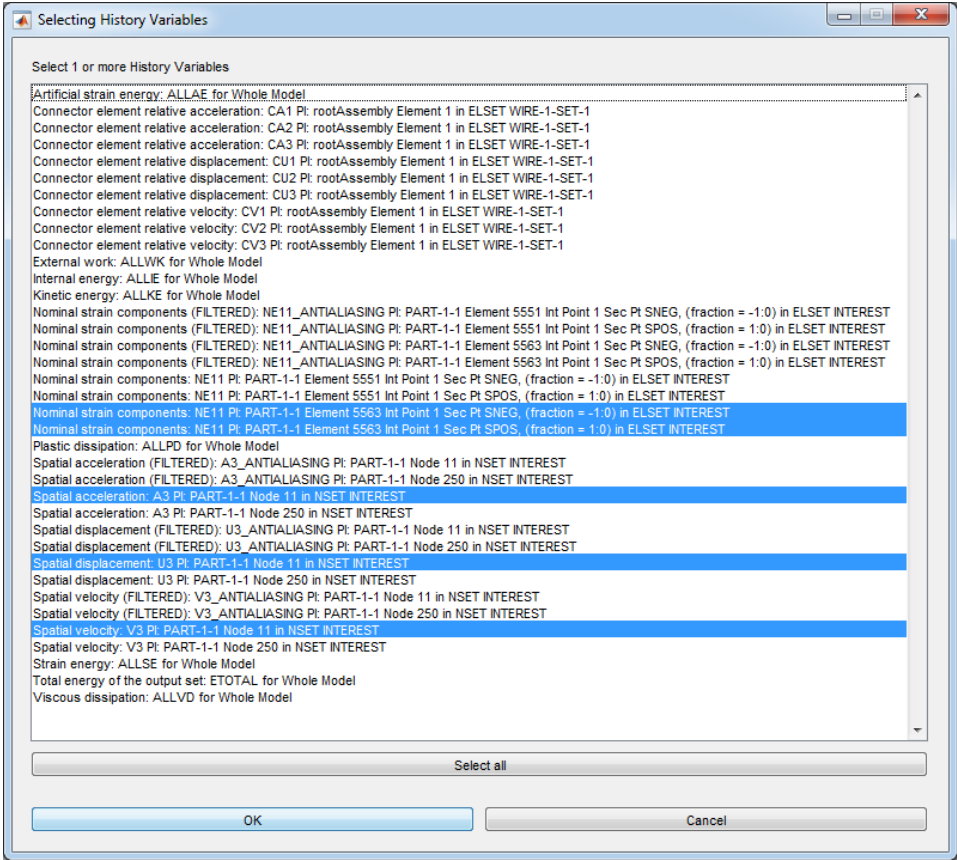
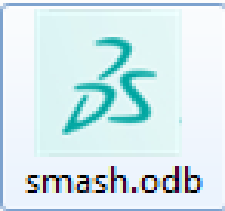
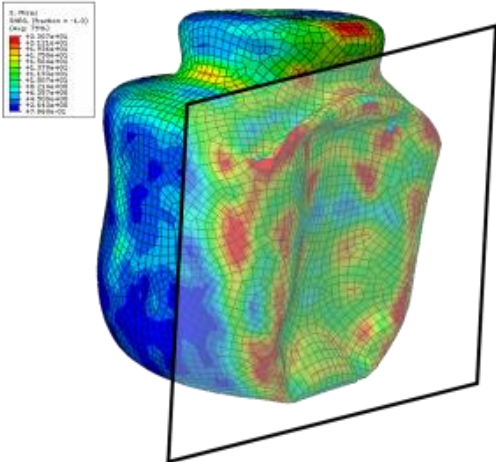
- See these 2 Tutorials for more on reading/writing text data files.
  - Tutorial on Reading Text Files goes over 12 vastly different types of data files.
  - Tutorial on Writing Text Files goes over 8 different scenarios, including reading/writing FEA input decks.
- Also check out the Kornucopia Help System function help pages.
  - `k_readText`
    - Having problems with Units not reading in properly? Look at ADV option 'unitsStrMap' on the function's help page.
  - `k_readTextToCellstr`
  - `k_unpackHeader`
  - `k_writeText`
  - For a variable `A` of type `k_units`, the `k_units` help page has details on the following methods:
    - `A.toStruct`
    - `A.toTable`



# Kornucopia ML Automates and Improves Abaqus Work-Flows

From MATLAB, use `k_abqOdbImport_gui` to import

- 1. Browse to ODB file
- 2. Select units system for model
- 3. Select history entities to import
- 4. Select steps to import



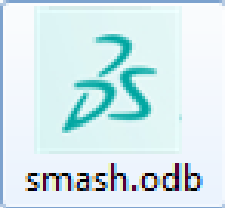
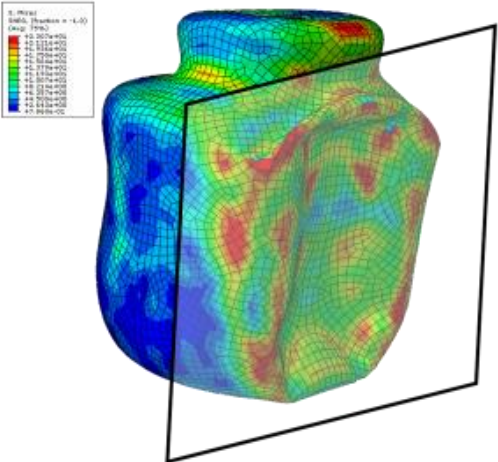
Note: depending on how Abaqus is configured on your computer, you may have to modify ADV option

'`abqNoguiCmd`' for the function `k_set`.

- This controls how Kornucopia calls Abaqus for the various `k_abq*` functions.



# Kornucopia ML Automates and Improves Abaqus Work-Flows



During import, **Kornucopia ML** automatically handles all column naming, documentation, and units.

Result of import via **k\_abqOdbImport\_gui**

k\_varViewer displaying odbRaw

File Edit View Insert Tools Desktop Window Help

k\_varViewer displaying odbRaw

=== ADDITIONAL PROPS INFO ===

\*\*Props.ColComments

For column: Time  
Time

For column: A3  
Spatial acceleration: A3 Pl: PART-1-1 Node 190 in NSET INTEREST

For column: U3  
Spatial displacement: U3 Pl: PART-1-1 Node 190 in NSET INTEREST

For column: NE11\_1  
Nominal strain components: NE11 Pl: PART-1-1 Element 1553 Int Point 1 Sec Pt SNEG, (fraction = -1:0) in ELSET INTEREST

For column: NE11\_2  
Nominal strain components: NE11 Pl: PART-1-1 Element 1553 Int Point 1 Sec Pt SPOS, (fraction = 1:0) in ELSET INTEREST

Important meta-info about each column automatically retrieved

	Time [s]	A3 [mm/s^2]	U3 [mm]	NE11_1	NE11_2
1	0	0	0	0	0
2	5.2637e-07	9.8604e-11	-0.0105	0	0
3	1.0527e-06	9.8935e-11	-0.0211	-1.9581e-25	1.9581e-25
4	1.5791e-06	1.2624e-10	-0.0316	-5.6049e-25	5.6049e-25
5	2.1055e-06	2.1943e-10	-0.0421	-9.3042e-25	9.3042e-25
6	2.6318e-06	3.0313e-10	-0.0526	-9.6874e-25	9.6874e-25
7	3.1582e-06	4.4961e-10	-0.0632	-2.6724e-25	2.6724e-25
8	3.6846e-06	5.9852e-10	-0.0737	1.6894e-24	-1.6894e-24
9	4.2110e-06	7.4743e-10	-0.0842	5.2486e-24	-5.2486e-24

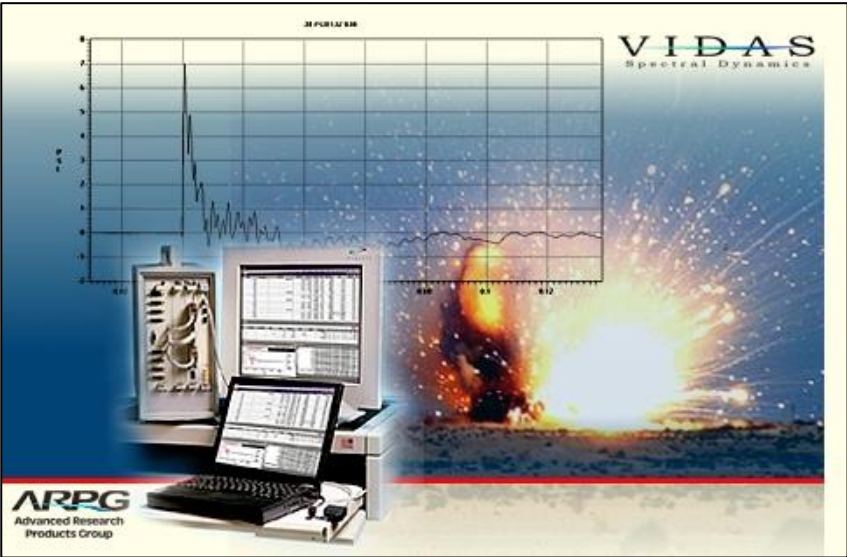
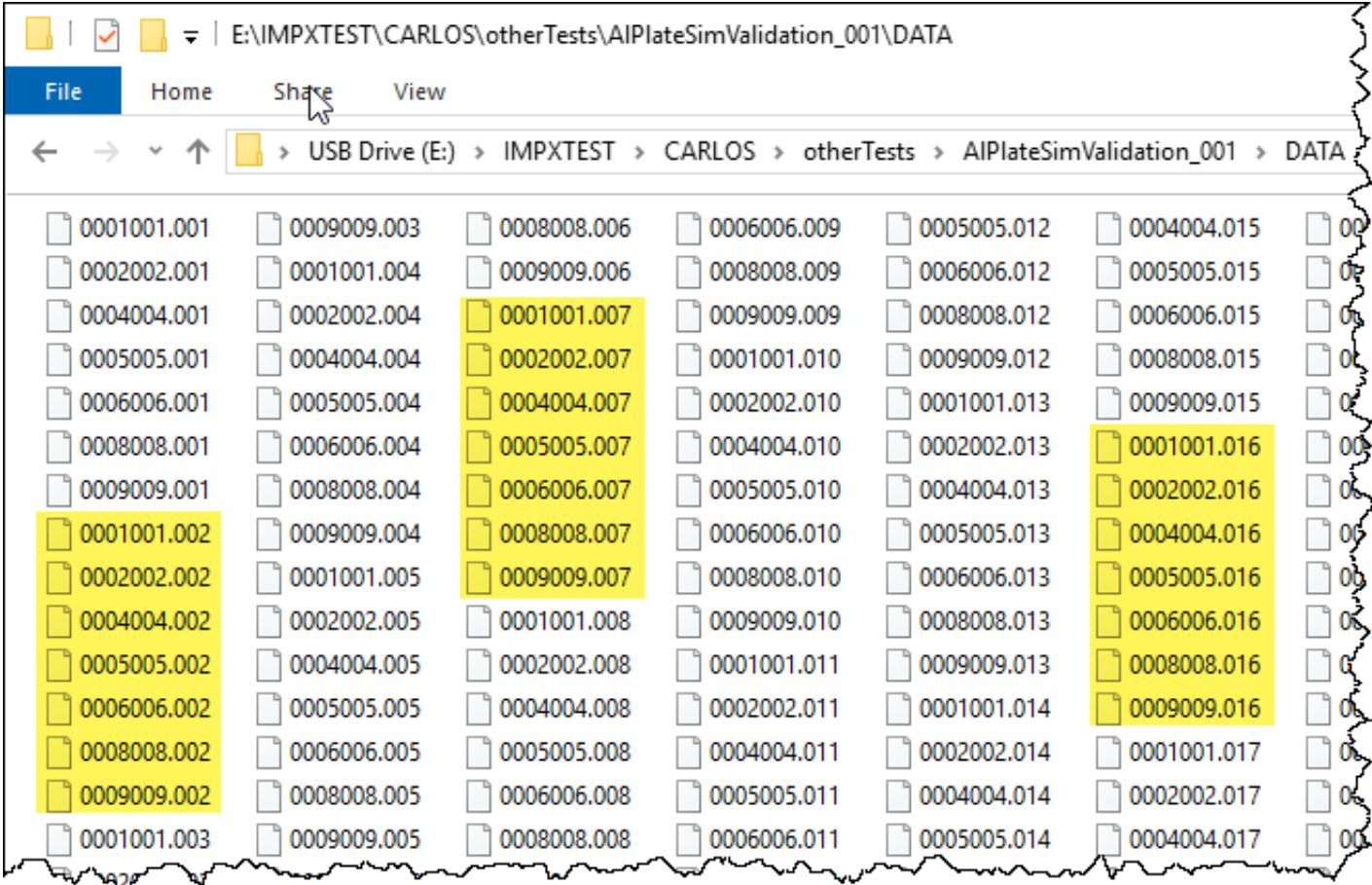
Col Names and Units





# Kornucopia ML Easily Processes Large Amounts of Impax-SD Data Files

- After a day of testing, there are a ton of files to process.
  - Data files for each test (event) get unique extension defined by the event number.



Impax-SD - Signal Directory E:\IMPXTEST\CARLOS\OTHERTESTS\ALPLATESIMVALIDATION\_001

Test Signal Setup Checkout Run View Report Storage Directory Window Utility Registration Knock Analysis Abs

Idle No current operation

Ref	DasC	Xdcr	FullScal	Unit	Status	P	Type	Position	Location	Component
1	001	A-10553	10000	G	Ready	N	AccelZ1	89*mm, 3.86*mm	AluminumPlate	
2	002	A-10358	10000	G	Ready	N	AccelZ2	176*mm, 49.5*mm	AluminumPlate	
3	003	VVS	5000	mV	Ready	N				
4	004	S-2.140-1ARM	1000	uStr	Ready	N	StrainY1	2.54*mm, 50.18*mm	AluminumPlate	
5	005	S-2.140-1ARM	1000	uStr	Ready	N	StrainX2	89.9*mm, 50.16*mm	AluminumPlate	
6	006	S-2.140-1ARM	1000	uStr	Ready	N	StrainX3	80.05*mm, 98*mm	AluminumPlate	
7	007	VVS	5000	mV	Ready	N				
8	008	K-C3610015H	1	mm	Ready	N	DispZ1	89*mm, 3.86*mm	AluminumPlate	
9	009	K-C3610007H	1	mm	Ready	N	DispZ2	176*mm, 49.5*mm	AluminumPlate	
10	010	VVS	5000	mV	Ready	N				





Easily import IMPAX data files via Kornucopia functions `k_sdImpaxImport` and `k_sdImpaxRtmImport`

During import, **Kornucopia ML** handles all column naming, Units, and storing of ALL the related meta-data.

k\_varViewer displaying DB

File Edit View Insert Tools Desktop Window Help

k\_varViewer displaying DB

	1 EventNum	2 EventDesc	3 Notes	4 Date	5 DatumID	6 Folder	7 CommonInfo	8 FsRaw [kHz]	9 SDRaw	10 SDComputed
1	6	To validate simulation models with p...		06-Jul-2017 ...	736882.7731...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
2	8	To validate simulation models with p...		06-Jul-2017 ...	736882.7773...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
3	9	To validate simulation models with p...		06-Jul-2017 ...	736882.7782...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
4	10	To validate simulation models with p...		06-Jul-2017 ...	736882.7786...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
5	11	To validate simulation models with p...		06-Jul-2017 ...	736882.7789...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
6	12	To validate simulation models with p...		20-Jul-2017 ...	736896.6743...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
7	13	To validate simulation models with p...		20-Jul-2017 ...	736896.6804...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
8	25	To validate simulation models with p...	Faster stick pull	20-Jul-2017 ...	736896.7173...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
9	27	To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4290...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
10	32	To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4306...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
11	33	To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4308...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
12	34	To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4311...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
13	40	To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4430...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
14	42	To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4436...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
15	44	To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4440...	D:\MPXTEST\TESTED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	

```
Command Window
>> DB.SDRaw

ans =

    16x1 cell array

    [1x1 k_units]
    [1x1 k_units]
    [1x1 k_units]
    [1x1 k_units]
    [1x1 k_units]
```

The screenshot shows the k\_varViewer application window. The title bar reads "k\_varViewer displaying DB". The menu bar includes File, Edit, View, Insert, Tools, Desktop, Window, and Help. The toolbar contains various icons for file operations and editing. The main window displays a table with 10 columns: EventNum, EventDesc, Notes, Date, DatenumID, Folder, CommonInfo, FsRaw [kHz], SDRaw, and SDComputed. The table contains 15 rows of data, each representing a simulation event. A green arrow points to the window title bar.

1	2	3	4	5	6	7	8	9	10
EventNum	EventDesc	Notes	Date	DatenumID	Folder	CommonInfo	FsRaw [kHz]	SDRaw	SDComputed
1	6 To validate simulation models with p...		06-Jul-2017 ...	736882.7731...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
2	8 To validate simulation models with p...		06-Jul-2017 ...	736882.7773...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
3	9 To validate simulation models with p...		06-Jul-2017 ...	736882.7782...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
4	10 To validate simulation models with p...		06-Jul-2017 ...	736882.7786...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
5	11 To validate simulation models with p...		06-Jul-2017 ...	736882.7789...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
6	12 To validate simulation models with p...		20-Jul-2017 ...	736896.6743...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
7	13 To validate simulation models with p...		20-Jul-2017 ...	736896.6804...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
8	25 To validate simulation models with p...	Faster stick pull	20-Jul-2017 ...	736896.7173...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
9	27 To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4290...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
10	32 To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4306...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
11	33 To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4308...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
12	34 To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4311...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
13	40 To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4430...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
14	42 To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4436...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	
15	44 To validate simulation models with p...	Faster stick pull	26-Jul-2017 ...	736902.4440...	D:\IMPXTEST\TIED\PLATEIMPACT\DATA	**struct**	5000	**k_units**	

```

Command Window
>> DB.SDRaw{4}

ans =
    AluminumPlate [010]
=====
    Time          AccelZ1      AccelZ2      StrainY1      StrainX2      StrainX3      DispZ1      DispZ2
    [msec]         [G]         [G]         [uStrain]     [uStrain]     [uStrain]     [mm]        [mm]

    -1.024         1.8936      -14.227     10.269        8.2225       -5.5388      0.0018686   0.0030274
    -1.0238        -5.6809     -7.8247     -2.5834       -3.0192      1.5457       0.0016461   0.0021815
    -1.0236         0.94681    -11.381     1.7438        8.6079       2.8338       0.002091    0.0028938
    -1.0234        -0.94681    -12.804     0.90419       5.0106       -4.5727      0.0023579   0.0024931
    -1.0232        -3.7873     -11.381     -7.3627       -0.70662     -1.5457      0.0015126   0.0025822
    -1.023         -2e-07      -17.783     0.77502       0.64238      -1.4813      0.0021355   0.0031164
    -1.0228         1.8936      -12.093     5.1668        3.3404       3.0914       0.0014237   0.0027157
    -1.0226        -1.8936     -12.804     1.55          3.4046       -0.38643     0.002269    0.0028048

... 55288 rows not shown.
See "n" rows via k_set('dispBrief', n).
** Other Props exists that are not displayed.
View other Props via dot syntax or functions k_summary or k_varViewer.

```



## Sharing Kornucopia Results With Colleagues

- **Sharing with other Kornucopia users**
  - You can save & share variables with MATLAB's `save` and `load` functions.
  - Save text-based output files via `k_writeText`.
- **Sharing with non-Kornucopia users**
  - Save `k_units` variables to MATLAB `table` or MATLAB `struct` data formats via `.toTable` and `.toStruct` methods.
  - Save text-based output files via `k_writeText`.



## Plotting Curves

## Plotting with `k_units` Data Type

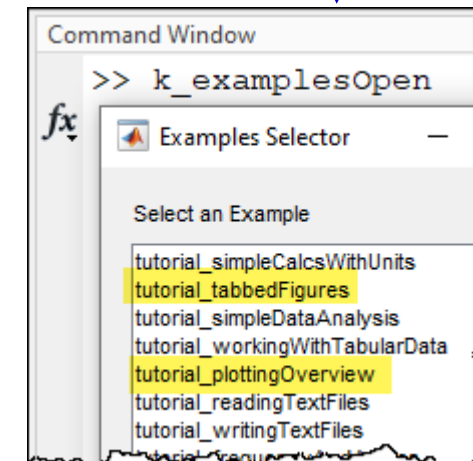
See Kornucopia Help System and Tutorial Examples to fully learn how plotting with `k_units` works in MATLAB.

- **Best Practice Tip**

- Always issue MATLAB `figure` (or `axis` or `subplot`) command to create an object in which plots will be placed.
  - If you do not do this and other figure windows already exist, the plot might be placed in one of those figure windows (which might be confusing to the user).

- **XY plotting**

- Use functions `k_plot`, `k_figZoom`, `k_displayOnFigure`, and many others.
- Kornucopia's `k_plot` function is designed to:
  1. Automatically handle Units issues and label the plot axes, title, and legend.
  2. Easily plot one data curve or many curves by simply supplying a cell array of data sets.
    - No need for for-loops and “hold on” syntax.
  3. Easily plot data sets in different compatible Units.
  4. Easily place well formatted summary tables in figure windows and plots via `k_displayOnFigure`.
- For 3-D plots and other plotting styles, you will need to plot just the data via `A.Data` Property.



# Typical Plotting Example

1

Starting with cell array of 5 data sets

```
rawA =  
5x1 cell array  
{1x1 k_units}  
{1x1 k_units}  
{1x1 k_units}  
{1x1 k_units}  
{1x1 k_units}
```

```
>> rawA{1}  
  
ans =  
Sample_01.dat  
=====
```

Time [s]	Nom Strain [%]	Nom Stress [MPa]
0.000	0.000	0.000
0.007353	0.2008	0.7866
0.01471	0.3476	1.296
0.02206	0.4943	1.738

```
... 133 rows not shown.
```

Details of 1<sup>st</sup> dataset

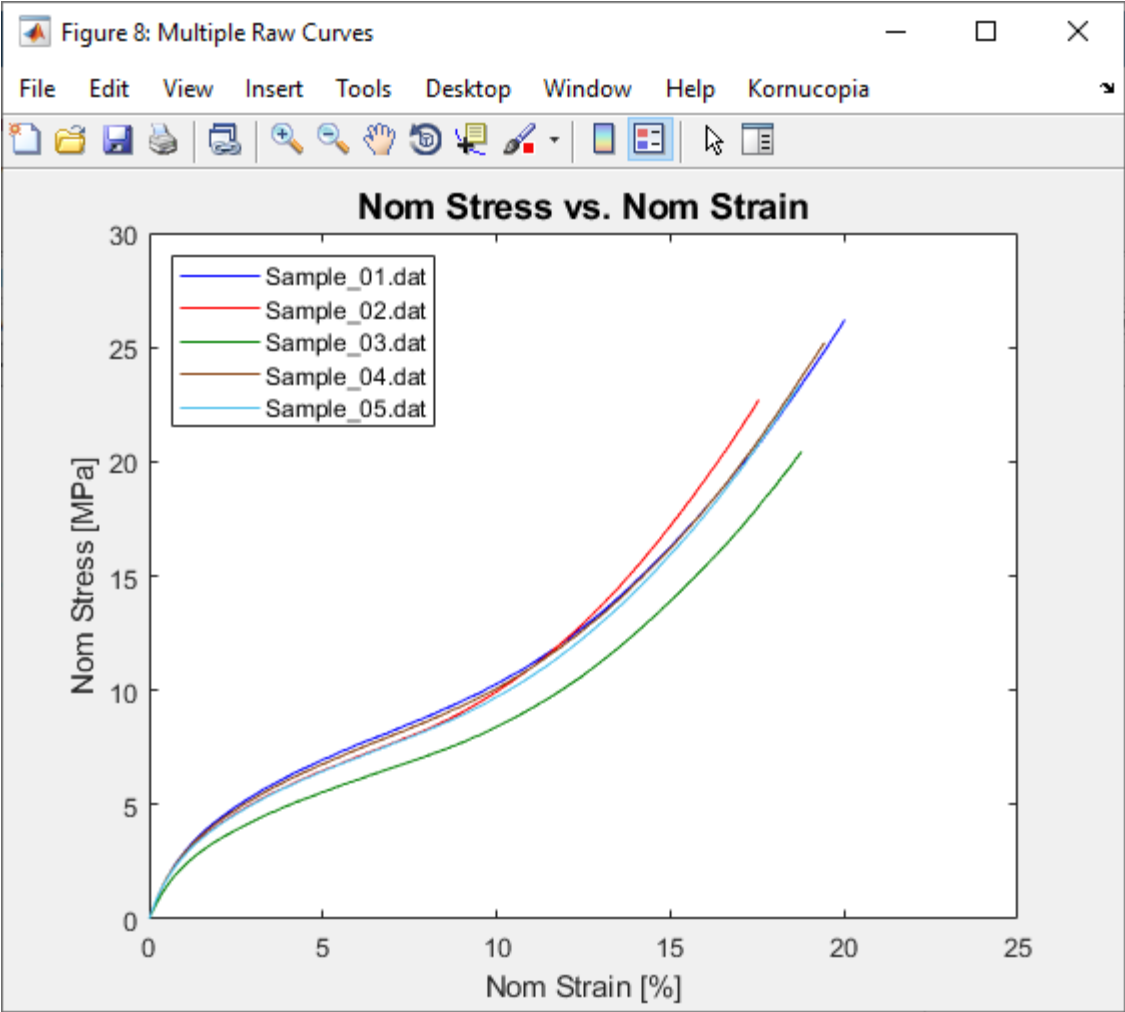
2

Using this figure and k\_plot command

```
figure('name', 'Multiple Raw Curves')  
  
k_plot(rawA, [], ...  
    'independentCol', 'Nom Strain', ...  
    'dependentCol', 'Nom Stress')
```

3

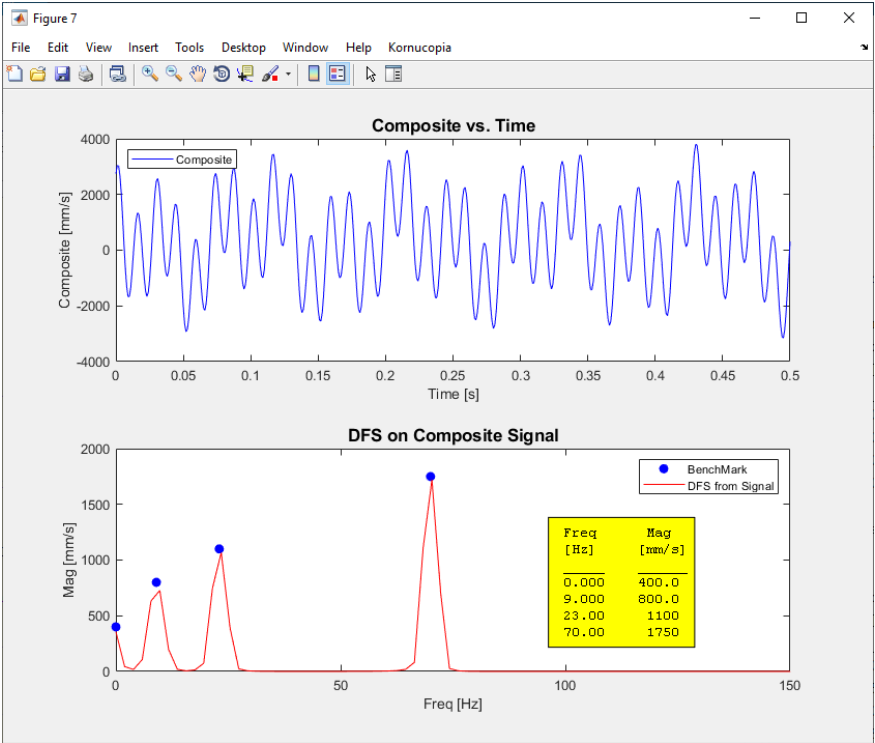
Produces this automatically-formatted plot





# Controlling Plotting Attributes with Kornucopia

- As listed on the right, there are a large number of Kornucopia functions to support creating well-labeled plots and output.
- For the function `k_plot` there are 71 ADV options for this function alone that control all kinds of aspects of plots.
  - See the function's help page and Tutorials & Examples for full details.
- Many of these functions and capabilities will be demonstrated in the examples during the later parts of this course.
- The figure presented here utilized:
  - `figure` along with `k_figZoom`
  - `subplot` (twice)
  - `k_plot` (twice) plus 9 ADV options
  - `k_displayOnFigure` plus 2 ADV options



## Plotting and Displaying Results

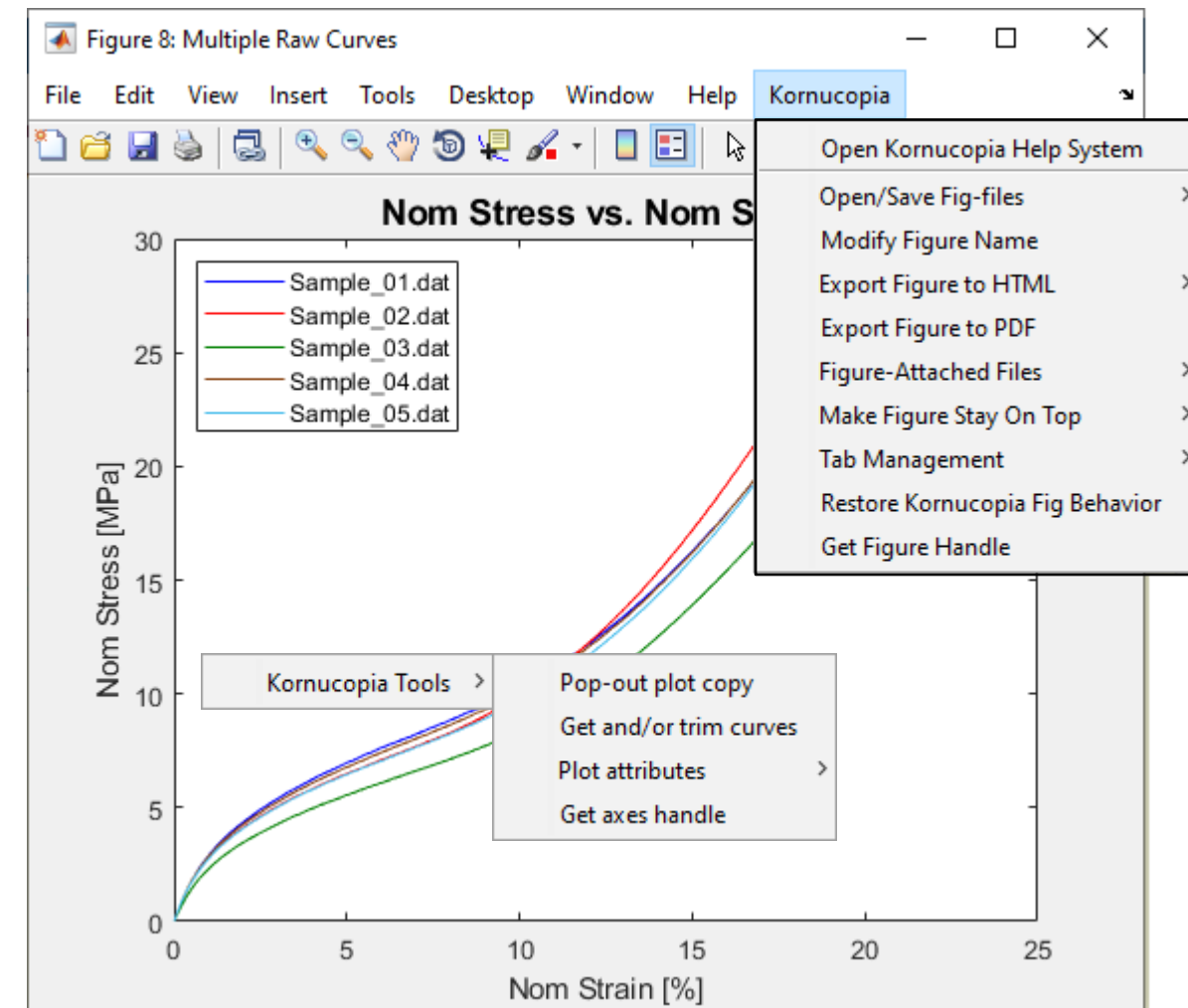
- 📄 `k_colorNames`
- + `k_displayOnFigure`
- + `k_figAttachedFiles`
- + `k_figExportHtml`
- + `k_figExportPDF`
- + `k_figOpen`
- + `k_figSave`
- + `k_figStaysOnTop`
- + `k_figTabsClear`
- + `k_figTabsCreate`
- + `k_figTabsDelete`
- + `k_figTabsDisplay`
- + `k_figTabsInsert`
- 📄 `k_figTabsH`
- + `k_figTabsList`
- + `k_figTabsManage`
- + `k_figZoom`
- + `k_picDisplay`
- + `k_plot`
- + `k_plotGetCurve`
- 📄 `k_plotGetLimits`
- + `k_plotMatchLimits`
- 📄 `k_plotSetLimits`
- + `k_progressBar`
- 📄 `k_stackForPlot`
- + `k_varViewer`

for tabbed figures



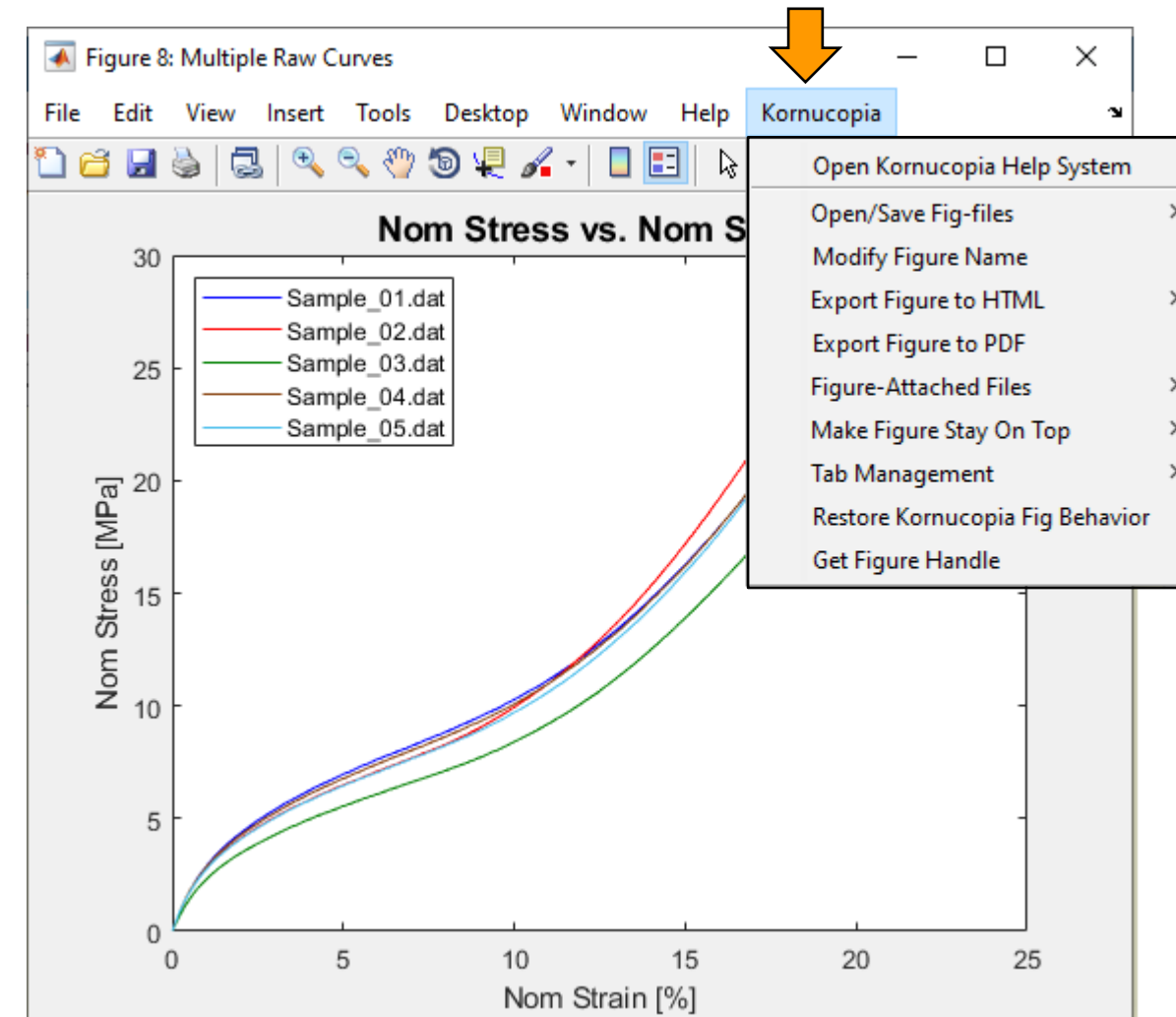
## Right-Click Features and the Kornucopia Menu

- **Right-click** on Kornucopia-created plots to get **Kornucopia Tools**. You get a variety of features including:
  - Pop-out a copy of the plot into another figure window.
  - Highlight curves and easily extract their data (including Units, ColNames, etc) back into the MATLAB session.
  - Modify some of the plot attributes.
  - Note: Ensure you do not have the figure in zoom or pan mode. If you do, the Kornucopia Tools will not appear until you click on the zoom or pan tool to turn off that mode.
- Use the Figure's **Kornucopia menu** to:
  - Save the figure to a file.
  - Easily create a PDF of the figure.
  - Attach virtually any type of files to the figure.
  - and more!



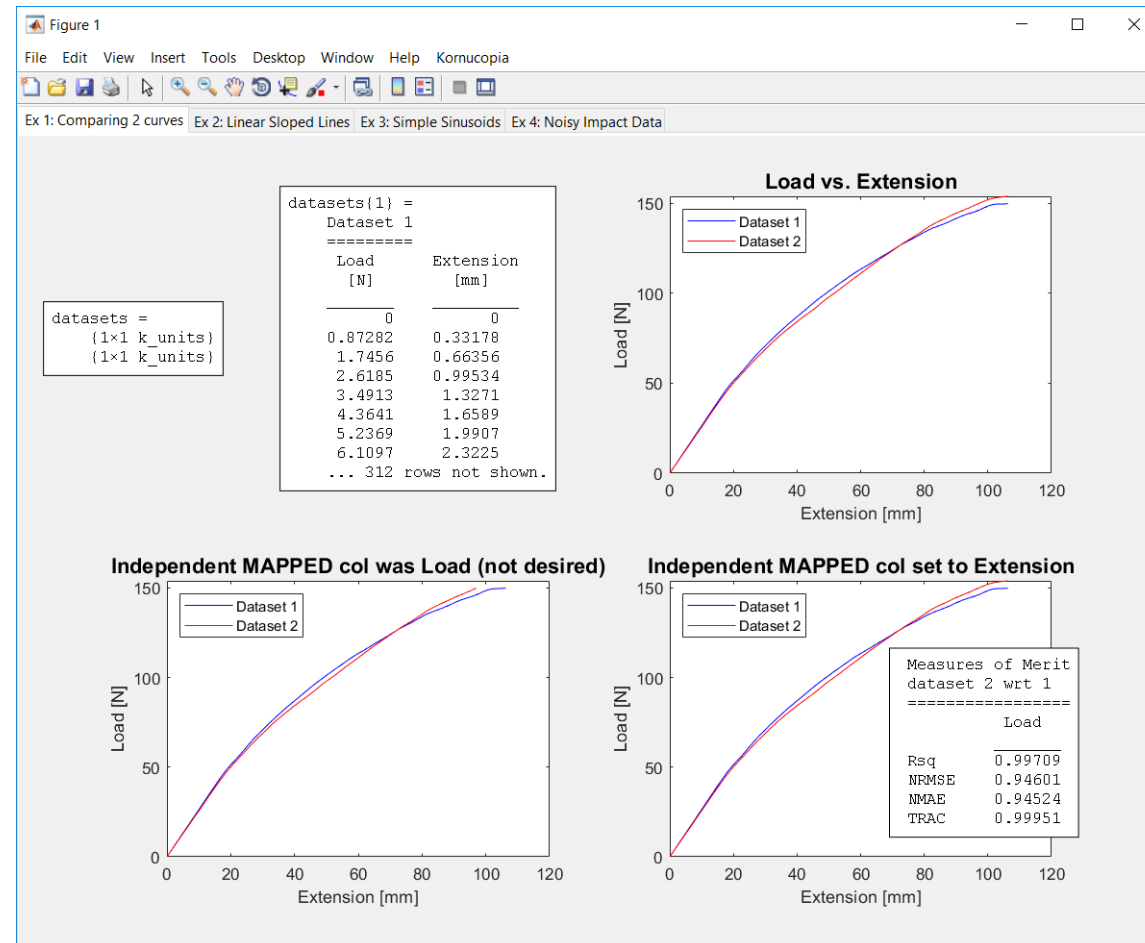
## Sharing Plotting Results With Colleagues

- You can save & share the **Figure file** via the *Kornucopia menu*.
  - Using the Kornucopia menu on the figure window properly prepares the figure-file so that any MATLAB user can view the figure file.
    - Kornucopia users will have full feature options when they open it.
  - **Do NOT use the MATLAB File Save or Save icons**, they may not properly prepare the figure relative to Kornucopia features.
- Easily create a PDF of the Figure, including any Kornucopia-created tabs
  - Create the PDF via the Kornucopia menu.
    - This is shareable with anyone.

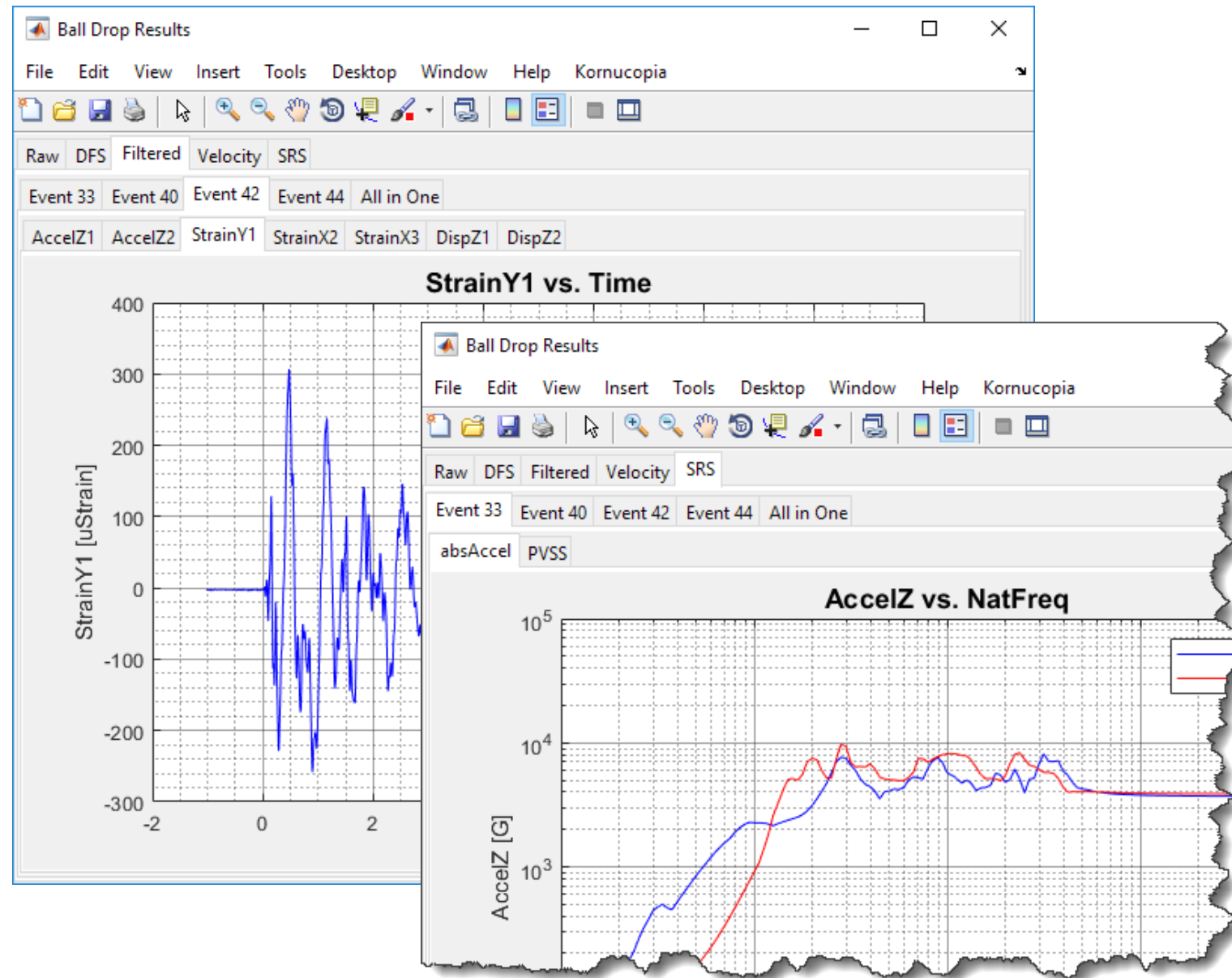




# Kornucopia Results Workbook™ via Tabbed Plots



## Kornucopia Results Workbook™ via a Tabbed Figure



- **5 Stages to analysis**  
Raw, DFS, Filtered, Velocity, SRS
  - **Multiple Events for each Stage**  
4 events (33, 40, 42, 44) & “All in One”
    - 3<sup>rd</sup> Tab row quantities depend on Stage Selection in 1<sup>st</sup> Tab row
  - **This particular example contains in ONE figure what would traditionally require 100 figures!**
  - **Additional Benefits**
    1. You can save it all as one fig file or export to PDF output
    2. The saved fig-file serves as a database for accessing data later.
    3. You can save m-file and other files attached to (inside) the Fig-file for traceability.

Pictures, Tables, Plots – All in One Place

Right-click on any of the Kornucopia-created graphic entities to access interactive *Kornucopia Tools*:

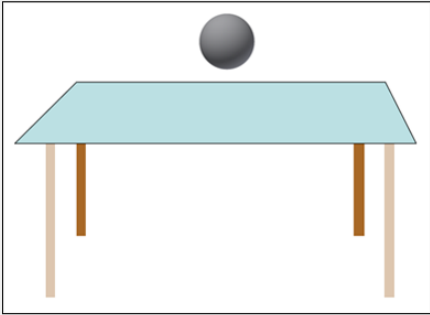
- Tabs, Pictures, Tables, Comments, & Plots

Aluminum Plate Ball Drop Results

File Edit View Insert Tools Desktop Window Help Kornucopia

Test Setup and Events Database Comments Some Results Comparing Displacements

Dropping Steel Ball on Alum Plate



EventNum	Date	ClipCheck	results
42	'26-Jul-2017 10:38'	'OK'	[1×1 k_units]
44	'26-Jul-2017 10:39'	'OK'	[1×1 k_units]
46	'26-Jul-2017 10:40'	'OK'	[1×1 k_units]
48	'26-Jul-2017 10:40'	'OK'	[1×1 k_units]

Aluminum Plate Ball Drop Results

File Edit View Insert Tools Desktop Window Help Kornucopia

Test Setup and Events Database Comments Some Results Comparing Displacements

The following adjustments to the raw data were already made:

1) Some signals were multiplied by -1 to have there results properly aligned with other sensor data.

2) The Keyence displacement sensors have an ADA ("Analog-to-Digital-to-Analog") internal DSP architecture. This means their ouput as seen by the SD Vidas DAQ will have a time delay relative to other non-Keyence sensors. The "Results" data presented here has NOT been corrected for this yet.

3) Data was decimated (with AA protection) from its original 5\*MHz sampling rate to the current sampling rate.

4) No corrections or adjustments have been made yet to the Accel data relative to low frequency drift or offsets.

Aluminum Plate Ball Drop Results

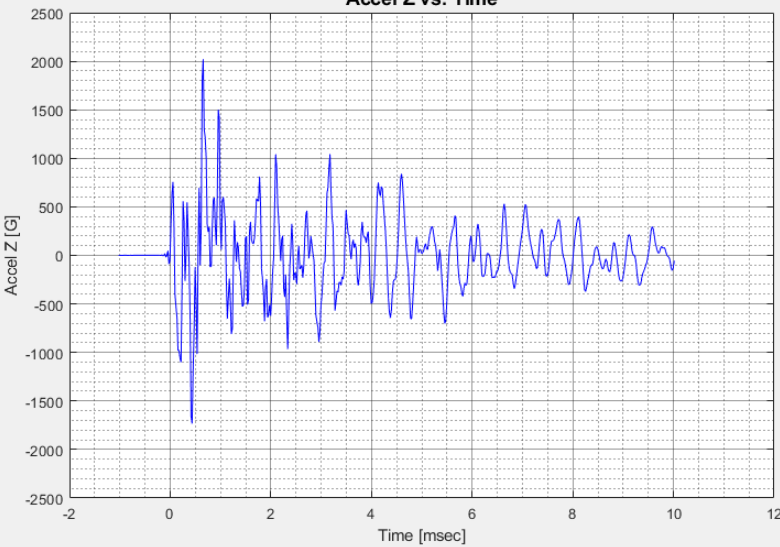
File Edit View Insert Tools Desktop Window Help Kornucopia

Test Setup and Events Database Comments Some Results Comparing Displacements

Event 42 Event 44 Event 46 Event 48

Accel Z Strain Y Disp Z

Accel Z vs. Time



via k\_picDisplay

via k\_displayOnFigure

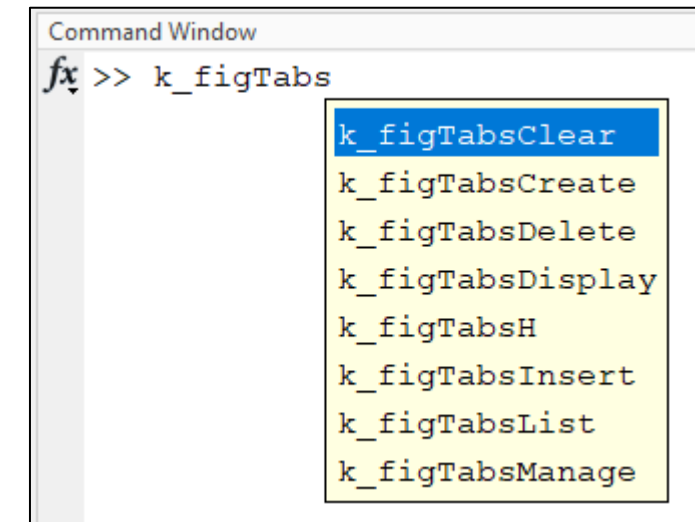
via k\_plot





## ***Kornucopia Results Workbook™ via Tabbed Plots***

- Kornucopia has a number of functions to make creating and managing tabbed figures easy.
  - This includes automatic Tab Management technology to keep the MATLAB figure window "snappy" even when a large number of tabs with large amount of data are within a single figure window.
- You have a lot of flexibility with Kornucopia's `k_figTab*` family of functionality.
  - A key ADV option you will need to use in a variety of MATLAB and Kornucopia plotting functions when using tabbed figures is the 'parent' keyword. This is used to tell MATLAB within what tab you desire a specific plot to appear.



## Kornucopia Results Workbook™ via Tabbed Plots – Template Syntax

```

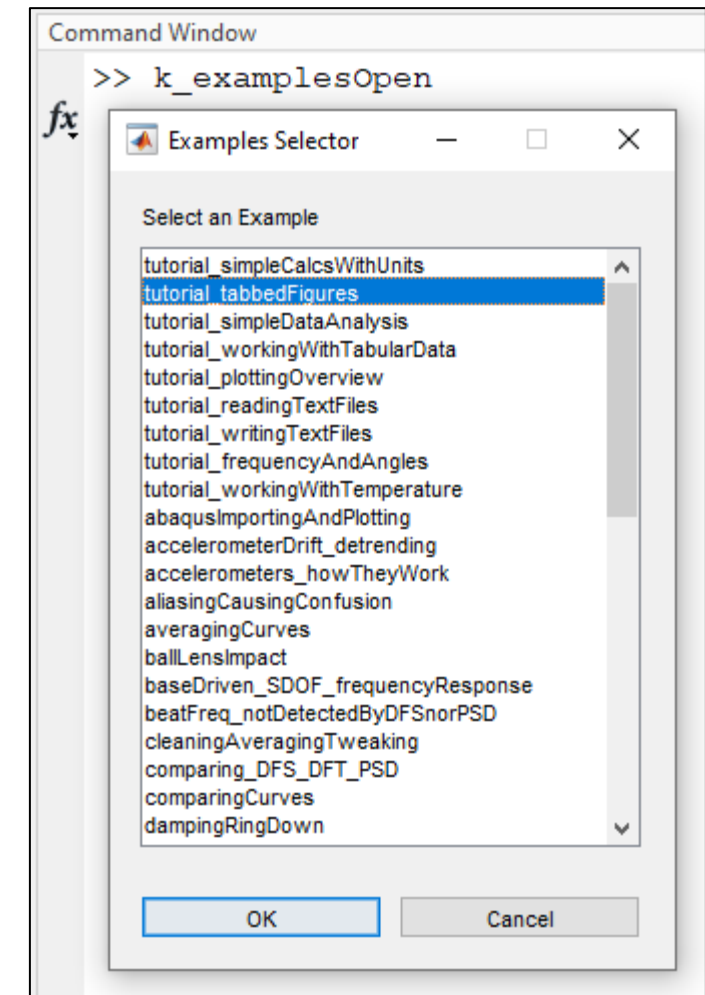
exemplar_tabbedFigure_commands.m  x  +
1      % Exemplar Tabbed Figure Plotting Template
2
3  -   k_cleanup();
4  -   k_unitsPreferenceActivate('mm_N_s');
5      % k_unitsVariables('mm, MPa');
6  -   adv = k_adv();
7
8      % Create a figure with 10 tabs on it
9  -   figH = figure('Name', 'My Notebook', ...
10         'position', k_figZoom(2, 1.5));
11
12  -   tabStuff = k_figTabsCreate(10, figH);
13
14      %% Plot in a tab
15
16  -   currentTabH = tabStuff.TabsH(2);
17  -   k_figTabsClear(currentTabH);
18  -   currentTabH.Title = 'Some Raw Data';
19
20      % Exemplar plotting command for variable toPlt
21  -   k_plot(toPlt, [], ...
22         'parent', currentTabH);
23
24  -   k_figTabsDisplay(currentTabH);

```

- In the top section, create a single figure window.
  - `figH` is the variable name associated with the Figure's *handle*. This is how we later reference the figure object.
  - The variable `tabStuff` represents a special `k_figTabsH` variable type that returns a struct-like output of *handles* related to the tabs you are creating.
- In the *Plot in a tab* region of the script we are:
  1. Selecting a current tab handle.
  2. Ensuring the tab selected is cleared (emptied).
  3. Giving the tab a title (a name).
  4. Issuing whatever plot-like commands are needed. The plotting functions use the ADV `'parent', currentTabH` to ensure the content goes to the desired location.
  5. The last command makes the `currentTabH` be displayed (so we see the plotted results).

## More Details on *Kornucopia Results Workbook™* via a Tabbed Figure

- See the tutorial on **tabbedFigures**.
- Look are a variety of Kornucopia examples.
  - Many of them utilize tabbed figures.
- The exemplar work-flows demonstrated next in this training will utilize tabbed figures.





## **Workshop Examples of Complete Kornucopia Workflows**

## Workshop Examples

- The following 4 workshop examples are provided
  - **Typical Shock Accel Processing**
  - **Cleaning Messy Tension Data**
  - **Ball-Lens Impact**
  - **Moon Penetrator**
- Data files, including completed Kornucopia scripts are in separate folders within the `exampleScripts` folder provided with the training documents.
  - The various examples walk the user through typical Kornucopia work-flows that import one or more data files and then perform a variety of data analysis and clean-up tasks.
  - Please refer to the specific example folders for solutions (m-files and output results).
    - The remaining few slides provide brief overviews of each problem.
- In addition to these workshop examples, feel free to explore the many tutorial and general examples of Kornucopia.

## Typical Shock Accel Processing – Overview

- This example demonstrates the typical steps for analyzing accelerometer data physically measured in a drop test of a consumer electronic device. The example takes the user through the key steps of data analysis:
  1. Initially assessing the raw data.
  2. Making adjustments for DC bias in the accel data.
  3. Computing a Fourier spectrum to assess frequency content.
  4. Applying lowpass filtering to improve data interpretation by removing distortions due to sensor resonance.
    - Set  $f_c$  of LP filter to  $0.1 \cdot f_{n\_sensor}$  as determined from DFS.
  5. Integrating acceleration to velocity, and then performing a sanity check relative to theoretical bounds for velocity change.
  6. Computing SRS for the data.
- **Your Job**
  - Import the raw accel text file that is supplied and perform the key steps outlined above.



## Cleaning Messy Tension Data – Overview

- This example demonstrates the typical steps for cleaning up mess and slightly noisy uniaxial test data that might occur when measuring at the low end of a load cell and/or when measuring challenging materials like a non-woven sheet. The example takes the user through the key steps of data analysis:
  1. Importing multiple data files, 1 per sample.
  2. Cleaning each raw data set of force vs displacement, including
    - Trimming off initial distortion and force-drop after failure.
    - Smoothing each curve to remove noise
    - Tweaking each curve to "heal" initial trimming.
  3. Converting the cleaned Force vs Disp curves into nominal Stress vs Strain curves.
  4. Averaging the Stress vs. Strain curves to a single curve.
  5. Isolate the initial region and compute an initial curve fit to estimate the initial modulus.
- **Your Job**
  - Import the raw text data files that are supplied and perform the key steps outlined above.

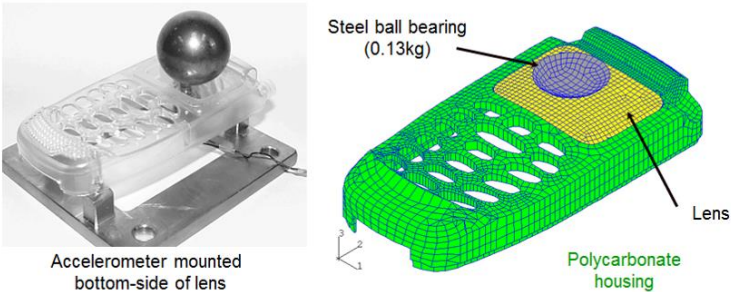
# Ball-Lens Impact – Overview

- Shock measurements of a steel ball impacting on the lens of an older generation cell phone housing are provided. The data are measured on the under-side of the plastic lens, directly at the point of impact.
  - The physical test was collected at 250\*kHz sampling frequency with a proper DAQ that included an AA filter.
  - The FEA data was output at every time increment from an Explicit Dynamics simulation. No filtering or processing has been applied to the raw FEA data.

- **Your Job**

- Compare the two results. How well do they correlate?

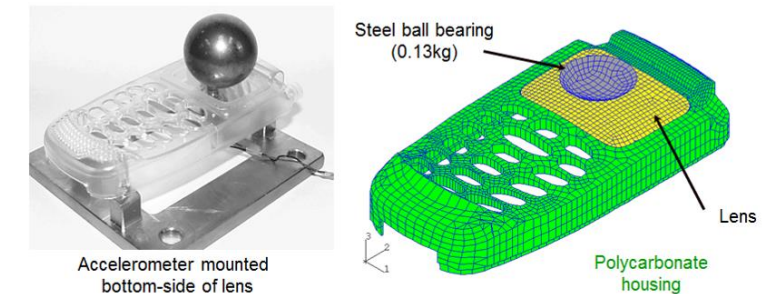
experiment.txt	fea.txt
1 Experiment	1 FEA model
2 Time, Accel	2 Time, Accel, Disp, Strain
3 msec, G	3 sec, mm/sec^2, mm, m/m
4 -8.029320000e-01, -1.250000000e+01	4 0.000000000e+00, 0.000000000e+00, 0.000000000e+00, 0.000000000e+00
5 -7.989320000e-01, -1.250000000e+01	5 1.122280000e-07, -6.518803100e+05, 0.000000000e+00, 0.000000000e+00
6 -7.949320000e-01, -1.250000000e+01	6 2.244260000e-07, -4.384346490e+06, -8.208800320e-09, 5.146795650e-10
7 -7.909320000e-01, -1.250000000e+01	7 3.366550000e-07, -1.538233660e+07, -7.162746430e-08, 4.487320420e-09
8 -7.869320000e-01, -1.250000000e+01	8 4.488680000e-07, -3.892847060e+07, -3.287481080e-07, 2.074711800e-08
9 -7.829320000e-01, -1.250000000e+01	9 5.610960000e-07, -8.016434480e+07, -1.076075320e-06, 6.875398250e-08
10 -7.789320000e-01, -1.250000000e+01	10 6.732940000e-07, -1.425521240e+08, -2.832871910e-06, 1.839639430e-07
11 -7.749320000e-01, -1.250000000e+01	11 7.855070000e-07, -2.264302060e+08, -6.384755130e-06, 4.228287350e-07
12 -7.709320000e-01, -6.250000000e+00	12 8.877350000e-07, -3.280466310e+08, -1.278795910e-05, 8.663322140e-07



## Ball-Lens Impact – Overview (continued)

Consider doing the following during your analysis:

1. Plot the raw data (Accel vs Time). How well does the physical data and FEA compare?
2. Regularize the FEA data and then decimate it to similar sampling freq of physical data.
3. Perform Fourier Analysis to help understand frequency content of the two data sets. Look for frequency bands where correlation is possible.
4. Use the whittling method to LP filter the data with different cutoff frequencies. You are looking for a cutoff frequency to apply that results in the filtered data between the FEA and physical test matching well (Accel vs Time).
5. Using the LP filtered results from above, integrate the physical experiment's Accel vs Time to yield Disp vs Time. Compare this result to the filtered FEA Disp vs Time result.
  - You should see there is discrepancy in both the displacement amplitude and time scale. This suggests the FEA stiffness is not quite correct.
  - Make a correction to the FEA data via a stiffness adjustment, which will also result in a time-scale warping. (See instructor's solution if you are not sure how to do this).
6. From the adjusted FEA displacement data, re-calculate an adjusted FEA acceleration. Then compute PVSS of the Physical Experimental Accel and the two FEA Accels (before and after adjustment), as a final comparison.



## Moon Penetrator – Overview

- A space agency design team is working on a probe that can be shot through a thin layer of ice on a distant planet's moon.
  - The electronics in the probe must survive the severe penetration event. Once under the ice, they are to send data back to earth.
  - A physical experiment (on earth) is created where a penetrator is shot through a simulated layer of the moon's ice.
    - Acceleration data inside the penetrator is recorded.
  - An Explicit Dynamics FEA model of the penetrator is created to simulate the impact and penetration event.
    - Two versions of a model are created and run (model1 and model2).
      - *Plasticity, material failure, and erosion are included in the models.*
      - Acceleration data is reported from the models at the same location as in the physical experiment.
- **Your Job**
  - Import the raw text data files that are supplied and determine which version of the FEA best matches the physical test data.

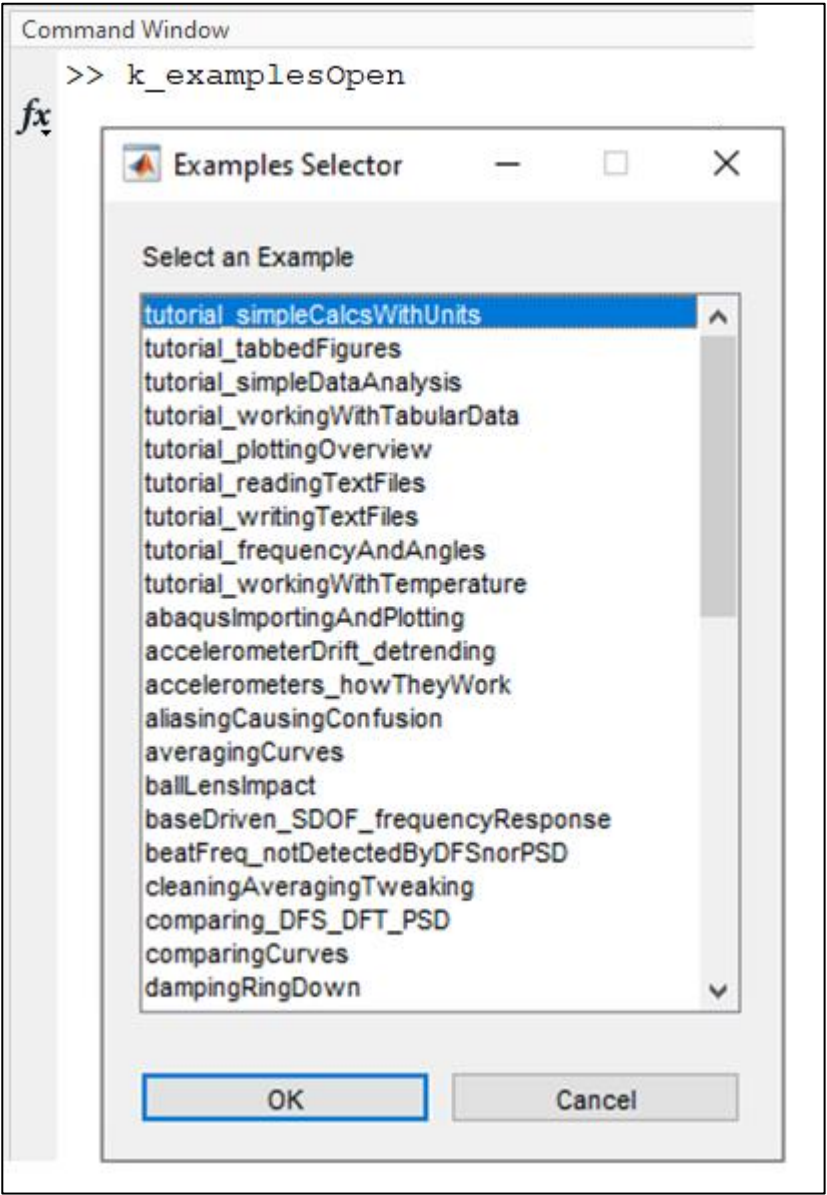
## Moon Penetrator – Overview (continued)

- Basic Guidance and Approach
  - Read in and plot the acceleration versus time data from the test as well as the two FEA models
    - Ensure you are looking at a similar duration of time for all data curves and that acceleration is in the same units for all data.
  - Assess the sample rates of the different datasets. You need to get all three datasets to a similar sampling rate.
    - Decimation will help here.
  - Make a fair comparison between the test and models. You might consider utilizing:
    - Fourier analysis
    - Decimation
    - Lowpass filtering
    - SRS analysis

Tip: You might find that upsampling of the physical test data will be helpful too.

# More Examples Beyond the Workshops Via Kornucopia Tutorials & Examples

- **Open live in MATLAB via `k_examplesOpen`**
  - Examples open live in MATLAB for you to explore and use as templates for your own work.
- **From Kornucopia Help System**
  - Listed alphabetically, useful for quick look at “published” HTML version of examples.



9 tutorials listed first, on top

Remainder of list are ~40 general purpose examples listed alphabetically

