CSI ANSYS Tip of the Week

Two Ansys APDL Macros

Mike Rife
Collaborative Solutions Inc.

Introduction:

The use of the Ansys Parametric Design Language, APDL, to create simple but reusable macros can be a significant time saver. This Tip of the Week will show two examples of macros written for current Ansys users. These macros may be extended or used as templates for future macros.

Note: When creating macros several points should be kept in mind.

1) The exclamation point (!) can be used for comments. Anything after the ! is ignored by Ansys.
2) The dollar sign ($) can be used to put more than one command on a line.
3) When naming parameters, let the last character be the underscore (_). ANSYS Inc. uses an underscore to start all of their parameters. Using an underscore as the last character will help avoid naming conflicts.

Introduction (Continued)

Macro Search Path

By default, ANSYS searches for a user macro file (.mac extension) in the following locations:

•The ANSYSnn/docu directory.
•The directory (or directories) designated by the ANSYS_MACROLIB environment variable (if defined) or the login (home) directory. This environment variable is documented in the ANSYS installation and configuration guide for your platform.
•The directory designated by /PSEARCH command (if defined). This directory is searched before the login directory, but after the directory designated by the ANSYS_MACROLIB environment variable.
• The current directory.

You can place macros for your personal use in your home directory. Macros that should be available across your site should be placed in the ANSYSnn/docu directory or some commonly accessible directory that everyone can reference through the ANSYS_MACROLIB environment variable.

Introduction (Continued)

Passing Arguments to a Macro

There are 19 scalar parameters that you can use to pass arguments from the macro execution command line to the macro. These scalar parameters can be reused with multiple macros; that is, their values are local to each macro. The parameters are named ARG1 through AR19 and they can be used for any of the following items:

- Numbers
- Alphanumeric character strings (up to eight characters enclosed in single quotes)
- Numeric or character parameters
- Parametric expressions

For example, the following simple macro requires four arguments, ARG1, ARG2, ARG3, and ARG4:

```
/prep7
block,,arg1,,arg2,,arg3
sphere,arg4
vsbv,1,2
```

To execute this macro, a user might enter the following in the INPUT WINDOW.

```
mymacro,4,3,2.2,1
```

Objective: A reusable macro that can be used to create geometry not readily accessible in the preprocessor.

Example: Cylindrical Body with a Helical Fin

In this example, the user (call him John Doe) frequently created helical fins on a cylindrical body. However, no helical 2D primitive exists to facilitate the modeling. A macro was created to build the cylindrical body with a helical fin based on parameters specified by the John. The macro, named fins.mac, uses the cylinders length, radius, the ribs radius and pitch as parameters with 1, 1, 2 and.2 being the default values, respectively. If zero is given for any parameter, the macro will prompt the user for a value.

Process:

Step1: Determine if the given parameters are valid.
Step2: Group all existing geometry/mesh entities and unselect. This will allow for an uncluttered view of the new geometry.
Step3: Create the Cylinder
Step 4: Create the Helical Fin
Step 5: Glue all of the resulting areas together. This will insure mesh connectivity.
Step 6: Clean up all internal parameters.

Commands used in this example:

Process Control Commands

***if**	Allows commands to be conditionally read by Ansys.
***endif**	Closes the *if statement.
***else**	Adds additional control in an *if/*endif statement.
***ask**	Prompts the user to input a value for a parameter.
***do/*enddo**	Forms a looping statement.

Components

cm	Creates a component out of the currently selected entities.
cmgrp	Groups components into an assembly.
cmsel	Used to select components & assemblies by name.

Commands used in this example (Continued)

Geometry creation

k	Creates a key point (kp)
csys	Used to set the type of coordinate system
l	Creates a straight line
wpstyl	Sets the style of the working plane. 2D primitives are created in the WP.
kplo	plots the key points
circle	Creates a circle area primitive
/view	Sets viewing direction for plots.
adrag	Drags a line(s) along a guiding line to form an area
nummrg	Compresses the numbering of entities. Useful if there are gaps in numbering sequences.

Inquiry Commands

***get**	Used to “get” information from the Ansys database

Listing of the macro fins.mac.

Step 1: Determine if the given parameters are valid.

Note the usage of if-then-else-endif constructs.

```
! USAGE:
! fins, length, cylinder radius, fin radius, fin pitch

/nopr  ! Suppresses printing to output window

! Check to see if users needs to enter data

*if,arg1,eq,0,then
*ask,length_,What is the cylinders length?,1
*else
length_=arg1
*endif
```

Pg 1

```
*if,arg2,eq,0,then
*ask,crad_,What is the cylinders radius?,1
*else
crad_=arg2
*endif

*if,arg3,eq,0,then
*ask,frad_,What is the fin radius?,2
*else
frad_=arg3
*endif

*if,arg4,eq,0,then
*ask,pitch_,What is the fin pitch?,.2
*else
pitch_=arg4
*endif
```

Pg 2

Step2: Group all existing geometry/mesh entities and unselect. This will allow for an uncluttered view of the new geometry.

```
cm,k_,kp
cm,l_,line
cm,a_,area
cm,v_,volu
cm,n_,node
cm,e_,elem
cmgrp,all_,k_,l_,a_,v_,e_,n_
cmsel,u,all_
```

Pg 3

Step3: Create the Cylinder

```
/prep7
/view,1,1,1,1! Change to an iso view
numcmp,all  ! Compress numbers of all entities
wpcsys,-1,0 ! Changes the WP to global CS but keeps
            ! the isoview.
! create cylinder
k,,0,0,0
kpnc_=_return   ! Returns kp number of kp just created
kplo
circle,kpnc_,crad_,
*get,l4_,line,0,num,max
l3_=l4_-1
l2_=l4_-2
l1_=l4_-3
k,,0,0,length_
kpnl_=_return
l,kpnc_,kpnl_
lextr_=_return
adrag,l1_,l2_,l3_,l4_,,,lextr_
```

Pg 4

Step 4: Create the Helical Fin

```
! create helixes using polar coordinates
wpstyl,,,,,,1  ! WP uses polar c.s.
csys,4         ! Change local c.s. to the WP c.s.
k,,crad_,0,0
k1_=_return
k,,frad_,0,0
k2_=_return
l,k1_,k2_
l1_=_return

ii_=length_/pitch_  !determines the number of fin areas for
                    !looping
*do,i_,1,ii_,1
k,,frad_,90*i_,pitch_*i_
k3_=_return
l,k2_,k3_
l2_=_return
adrag,l1_,,,,,,l2_
k2_=k3_
l1_=l2_+1
*enddo
```

Pg 5

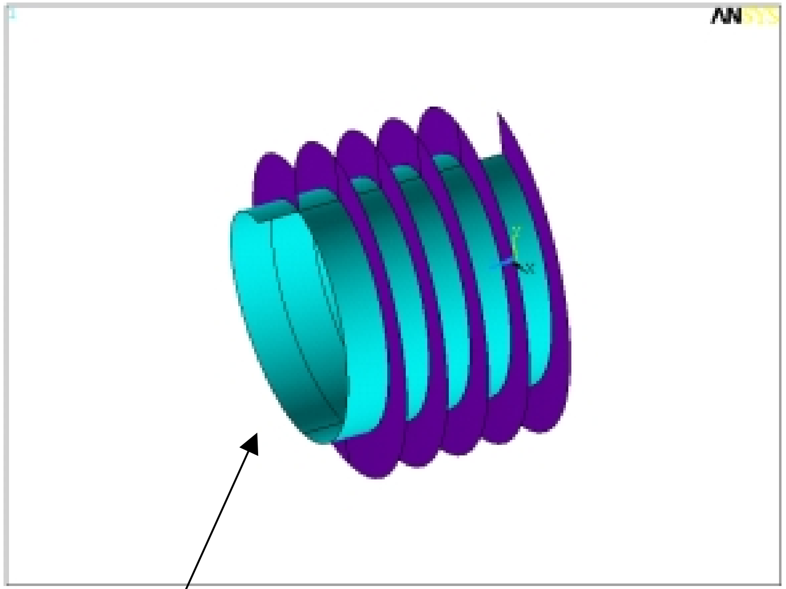
Step 5: Glue all of the resulting areas together. This will insure mesh connectivity.

```
aglue,all
csys
```

Step 6: Clean up all internal parameters.

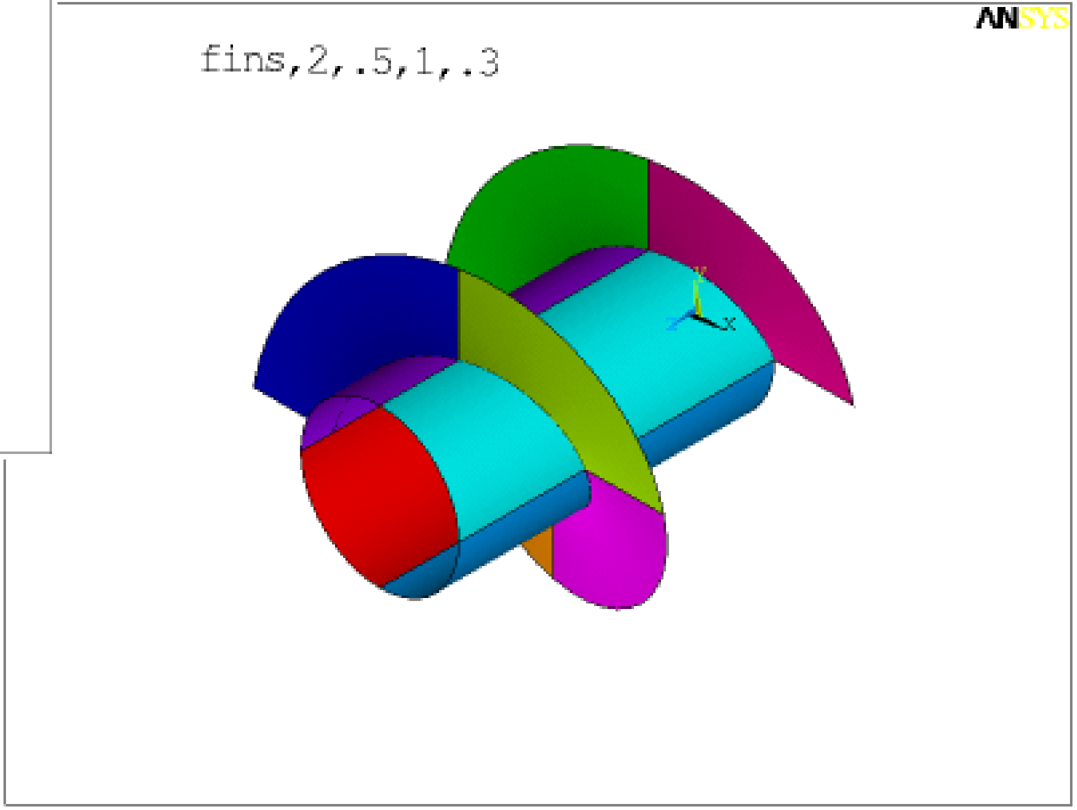
```
! Clean up all variables
! $ allows multiple commands on one line
length_= $ crad_= $ frad_= $ pitch_=
l1_= $ l2_= $ l3_= $ l4_=
kpnc_= $ kpnl_= $ lextr_=
k1_= $ k2_= $ k3_=
i_= $ ii_=

/gopr          ! Resumes printing to the output
window
```

Pg 6



An example of the cylinder built by the John D.

Sample output of the macro.



Objective: A reusable macro that can be used to process results not readily available to them in the time history post processor.

Example: Plotting Total Strain Energy versus Time

In this example, the user (Jane Doe) wanted to plot the total strain energy over time for a structural transient analysis. The time history post processor does not have an option to plot this quantity, therefore a summed strain energy element table is created for each load case, the value put into an array then the array can plotted. The plots show reversed video, but this command is only accessible from the GUI. To reverse the video, follow this menu path:

Utility Menu -> Plot Cntrls -> Style -> Colors -> Reverse Video

Additionally, the strain energy table is not deleted. Therefore the values could be written out to a file for use in another application (a common occurrence with Jane). See the CSI Tip of the Week "Formatting Output Text" for more information on creating custom output.

Process:

Step1: Determine if the given parameter is valid.
Step2: Dimension a table that will store the load step number, time and total strain energy values.
Step3: Set the table to plot ramped values.
Step 4: Utilizing a *do loop, for each load step create a element table of strain energy, sum the table and store the values in the table dimensioned in Step 2.
Step 5: Plot the table.
Step 6: Clean up all internal parameters.

Additional commands used in this example:

***dim**	Dimensions a vector, table or an array

Additional commands used in this example (Continued)

Post Processing Commands

set	Defines which load step results to read.
etable	Creates an table of element results
ssum	Sums the previously created table
/gropt	Used to set grid options
/axlab	Used to label the axis of an x-y plot
/grid	Turns on the grid for x-y plot
***vplot**	Plots the values of a vector (table, array) in an x-y format.

Listing of the macro strne.mac:

Step1: Determine if the given parameter is valid.

```
/nopr           ! Turn off output window printing
/post1          ! go to the results post processor
```

! check that the user inputted number of load steps in macro command

```
*if,arg1,eq,0,then
*ask,nls_,How many load steps?,1
*else
nls_=arg1
*endif
```

Pg 1

Step2: Dimension a table that will store the load step number, time and total strain energy values.

```
! dimension an table to store the strain energy terms
! first column=time, second = total strain energy, ! ! ! !
third = load step number
*dim,strne_,table,nls_,3
sesum_=0    ! sesum_ is sum of element strain energy
```

Step3: Set the table to plot ramped values.

! Set tables to give a ramped plot instead of the default bar !graph

```
*vfill,strne_(0,0),RAMP,0,1,
*do,i_,1,3,1
*vfill,strne_(0,i_),RAMP,i_,1,
*enddo
i_=
```

Pg 2

Step 4: Utilizing a *do loop, for each load step create a element table of strain energy, sum the table and store the values in the table dimensioned in Step 2.

```
! loop through all load steps
*do,i_,1,nls_,1
set,i_
*get,curtim_,active,0,set,time
etable,se_,sene
ssum
*get,sesum_,ssum,0,item,se_
strne_(i_,3)=i_
strne_(i_,1)=curtim_
strne_(i_,2)=sesum_
etable,eras
sesum_=0
*enddo
```

Pg 3

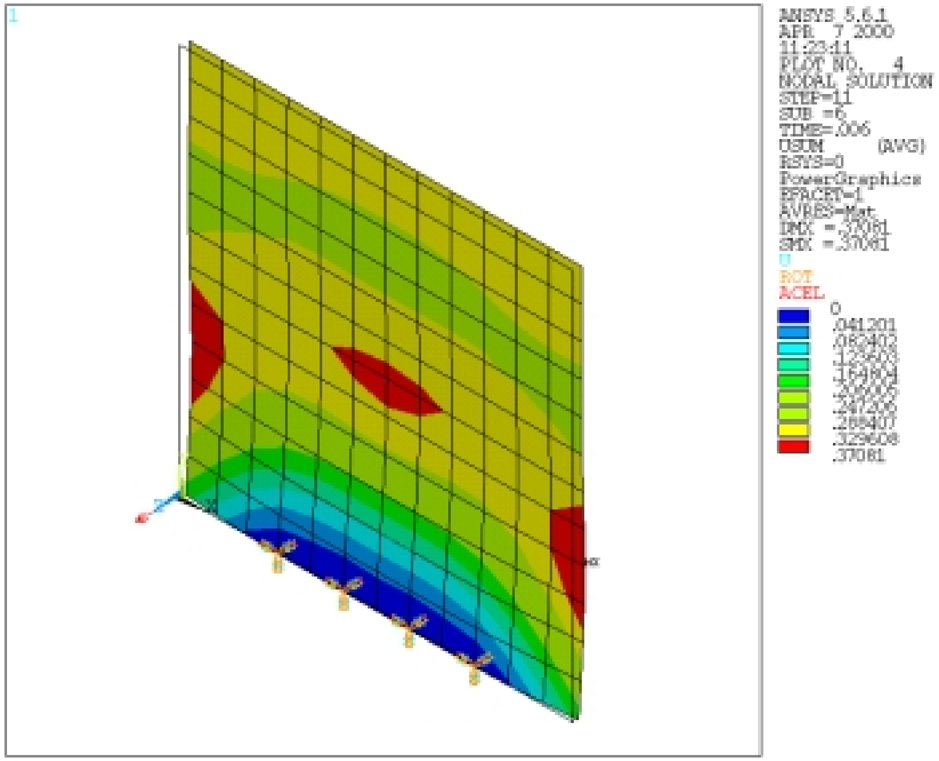
Step 5: Plot the table.

```
! plot strne_ table

/axlab,x,time                ! label x axis as time
/axlab,y,strain|energy       ! label y axis as strain-energy
/grid,1                      ! turn on the grid
/gropt,fill,on               ! fill the area under the curve
*vplot,strne_(1,1),strne_(1,2)
```

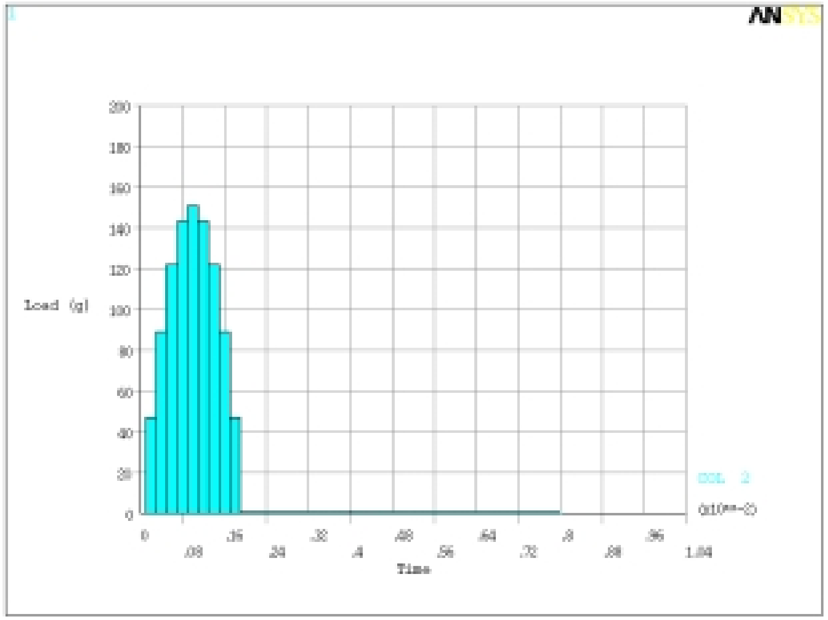
Step 6: Clean up all internal parameters.

```
! zero out all variables except the strain energy table
curtim_=
sesum_=
i_=
se_=
nls_=

/gopr
```

Pg 4

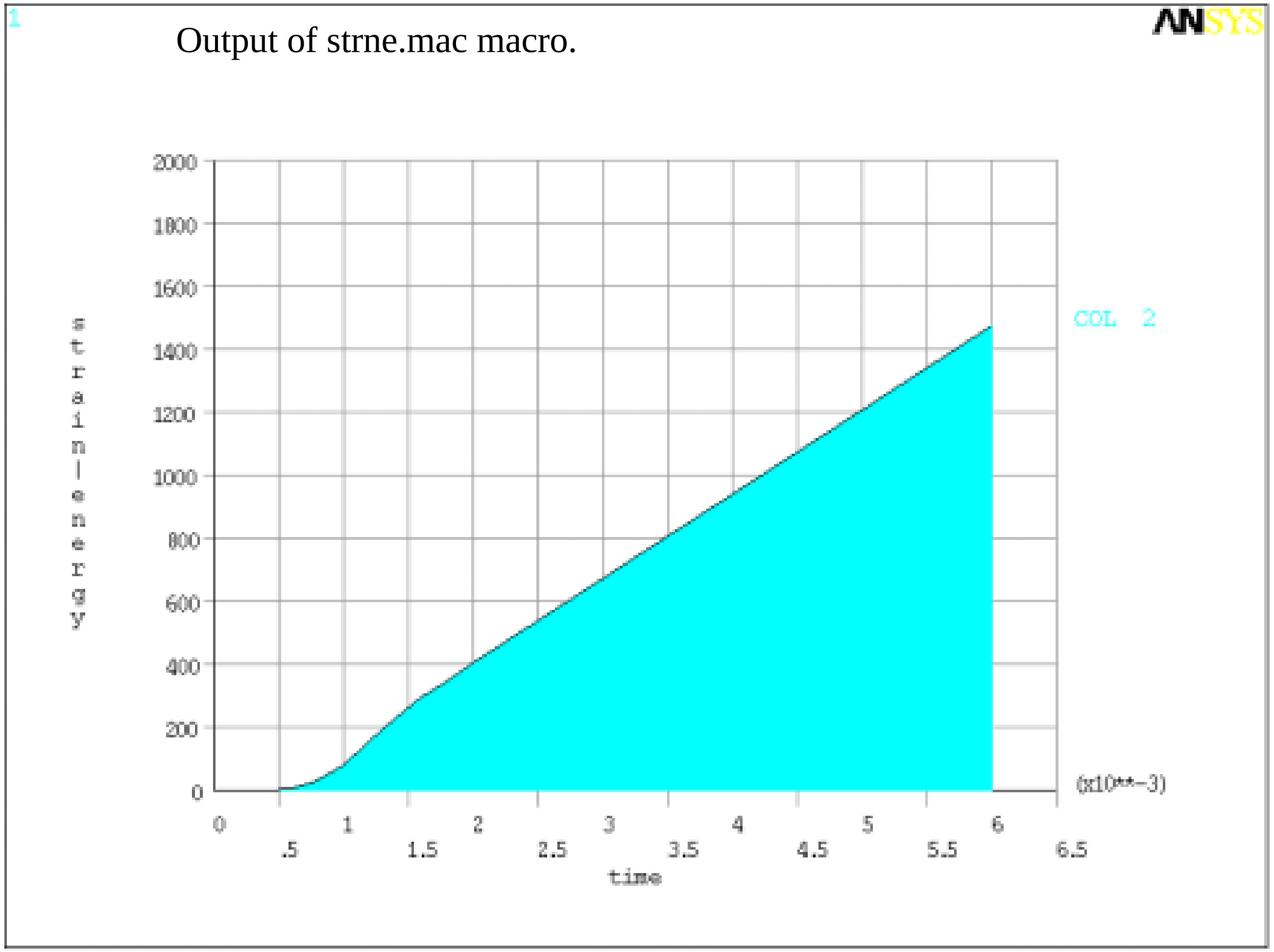


Plot of applied G load.



Test case for Example #2. 12”x12” plate with a 150g sinusoidal impulse load. Time duration 0.003 sec.

Output of strne.mac macro.



Summary

These two examples could be expanded upon to include additional control or used as a template for future macros. For example, the fins.mac macro did not check if the user entered a negative value for the cylinders radius. This could easily be accomplished by adding either an *elseif command to the appropriate *if loop, or writing another *if loop entirely. The macro could also be easily changed to create, say a elliptical cylinder. Other conic sections, such as parabolas and hyperbolas could be created. This, as they say, is left up to the student.

The strne.mac macro could incorporate a “plot dump” to a file. This way plots could be automatically created in, say, .tiff format for later inclusion in Power Point documents.