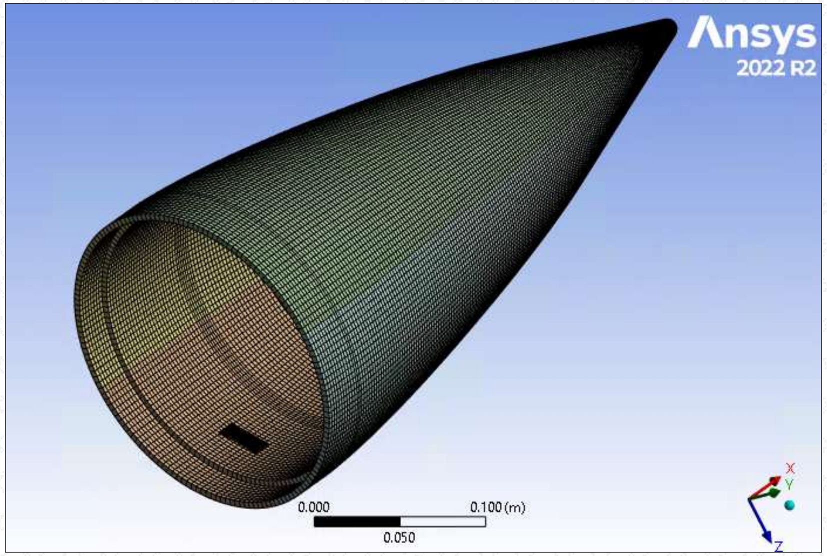
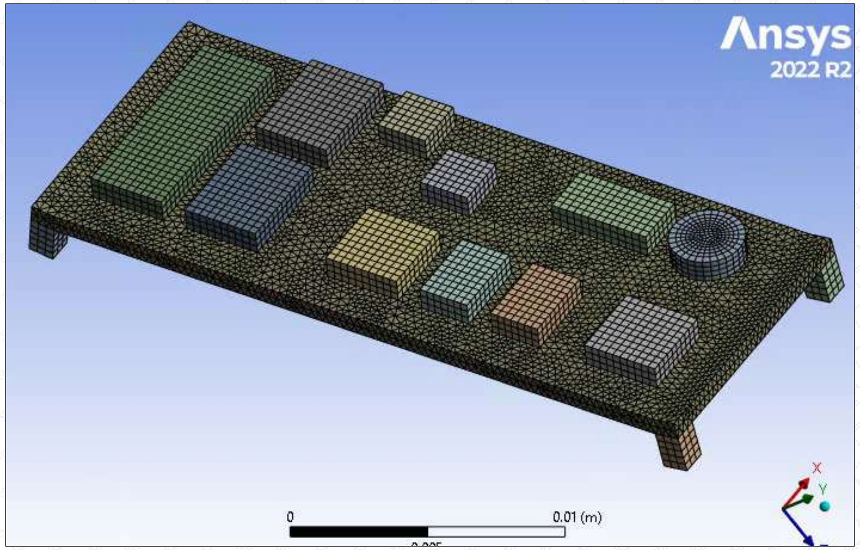
Mapping Data in Ansys Mechanical

An Introduction to PyAnsys





Alex Grishin, PhD
PADT, Inc
2/1/2024

Background

- Most Ansys users are aware of the various automation tools available –notably the Ansys Customization Toolkit (ACT), as well as the older Ansys Parametric Design Language (APDL)
- Within the last 10 years, Ansys has been developing a new automation framework wtih a different approach and different goals: PyAnsys
- While the older, more familiar automation frameworks are meant to be used *within* the Ansys environment to automate or customize simulation functionality, PyAnsys operates *outside* of the application itself.
- With this philosophy, automation occurs at the OS-level, with software connecting to either an Ansys database or the application itself (or even several Ansys databases at once)
- Among other possibilities, this allows users to easily incorporate simulation tasks into larger CAE processes

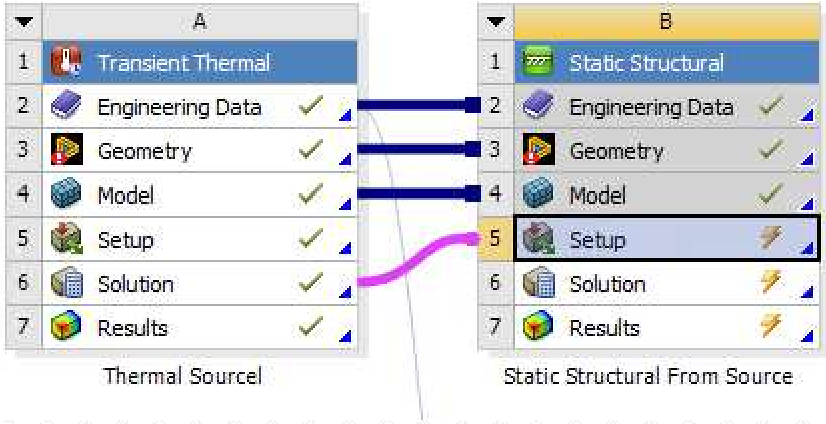
- In this article, we'll focus on a less ambitious goal: We'll see how the PyAnsys framework may be used to automate a frequent but cumbersome task: Mapping data from one simulation source to another
- We'll do so by first exploring options without PyAnsys

Part 1: Mapping Data:

Using Ansys ACT and APDL

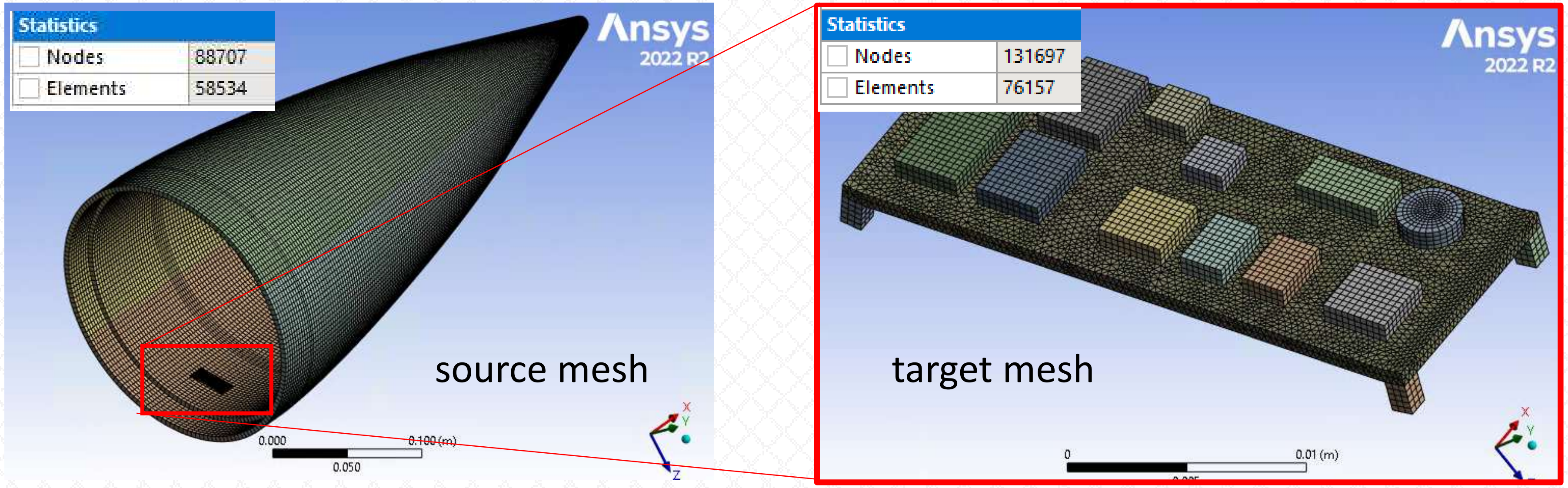
The Problem Statement: Data Mapping

- What we mean by "mapping data" from one simulation to another is the following:
  - we have some physical quantity calculated or prescribed over one geometric domain (represented by a mesh, or even just a set of points) in some environment which must be interpolated onto another domain or mesh which shares the same space *and serves as a load or input in the second environment* (we're usually not just interpolating a result onto different meshes. We want to calculate a new result with the source quantity interpreted as a load)
- The 'quantity' can be any scalar, vector, or tensor quanity (i.e. pressure, temperature, stress, etc)
- Ansys has a seemless load transfer mechanism which involves simply dragging the Solution cell from one analysis system (the 'source' data to be mapped) to the Setup cell of the target analysis system (the target model onto which the quantity is to be mapped). This results in a connection highlighted below



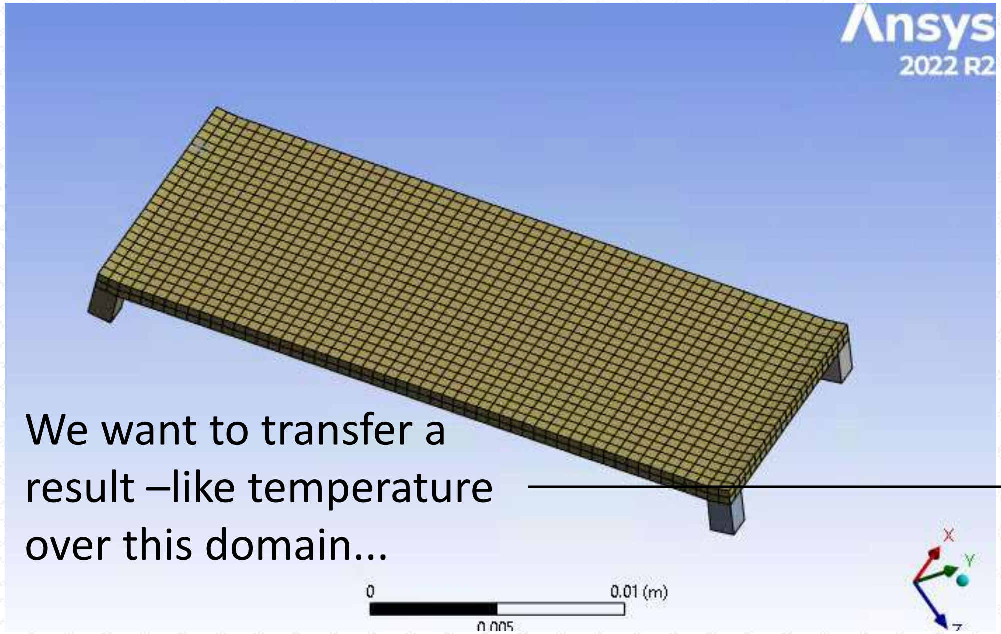
The Problem Statement: Data Mapping

- Although the Solution->Setup transfer mechanism will operate on meshes which are not identical, it will only do so within a fairly strict tolerence around the source mesh
- This mechanism is ideal for transferring results between different analysis environments but having the same mesh (as in the example on the previous slide), or for submodeling (different mesh, but falling within the spatial domain of the source mesh), but not for interpolating data onto meshes which may lie *outside* the source domain
- The example we'll use in this article is shown below



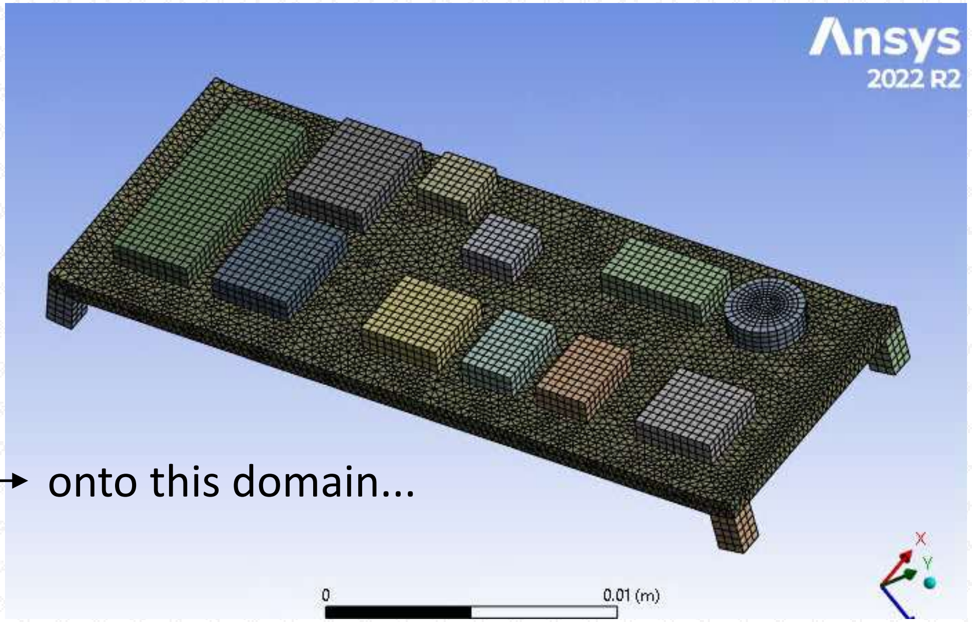
The Problem Statement: Data Mapping

- Within the source mesh domain, there exists a simplified electronic PCB model which contains no TTL component detail, while the target model has much more detail (which lies outside of the source main bounding box)



We want to transfer a result –like temperature over this domain...

source PCB model

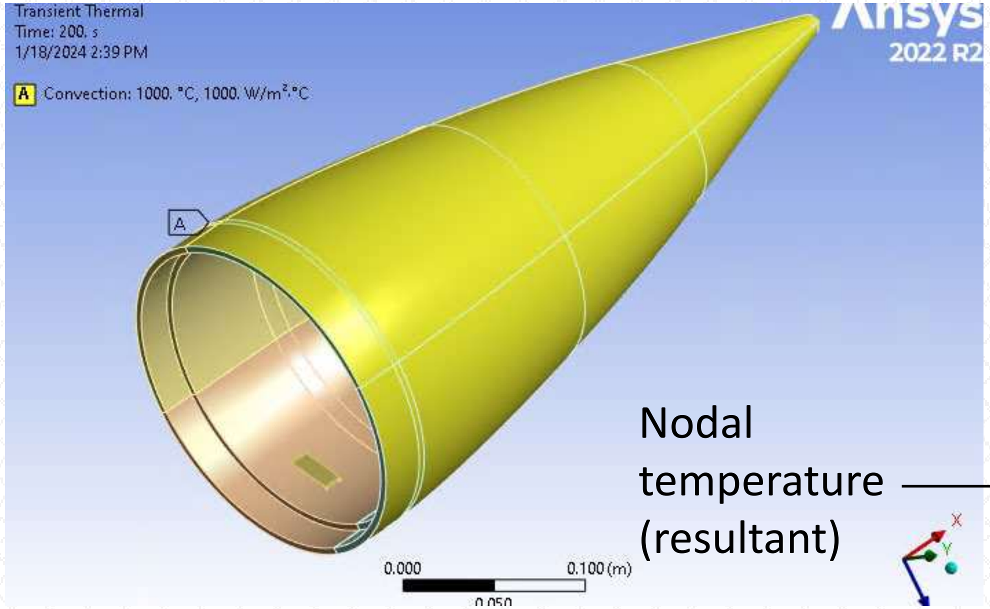


onto this domain...

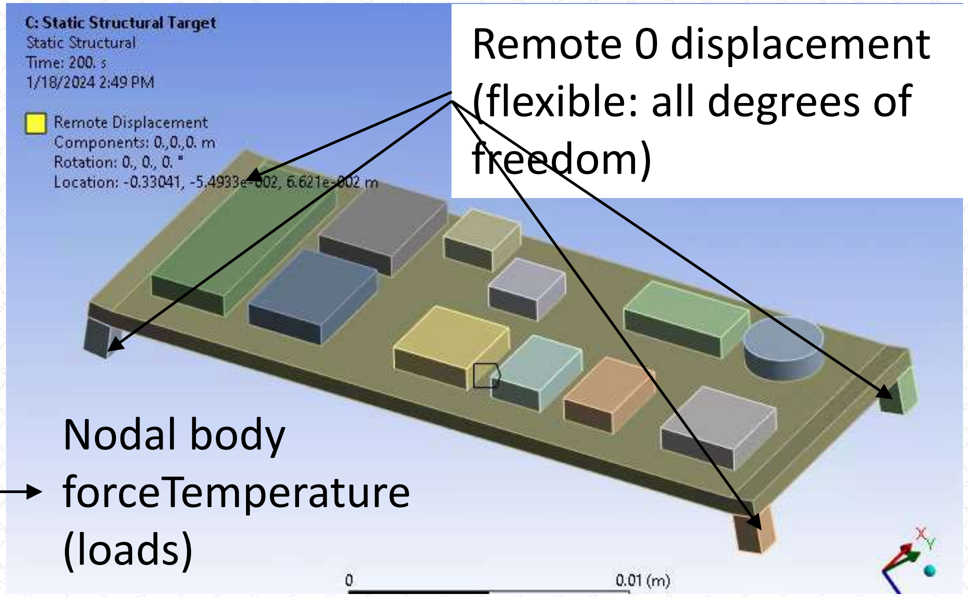
target PCB model

The Problem Statement: Data Mapping

- The source model is a transient thermal study of a nosecone (containing within it a small PCB component) subject to a convection of h=1000 W/m^2 C @ 1000 C ambient temperature for 200 seconds, starting from room temperature (22 C)
- The target model consists of the detailed PCB component subject to the mapped temperatures from the source model. The model is fixed at four solder locations with a remote (flexible) constraint



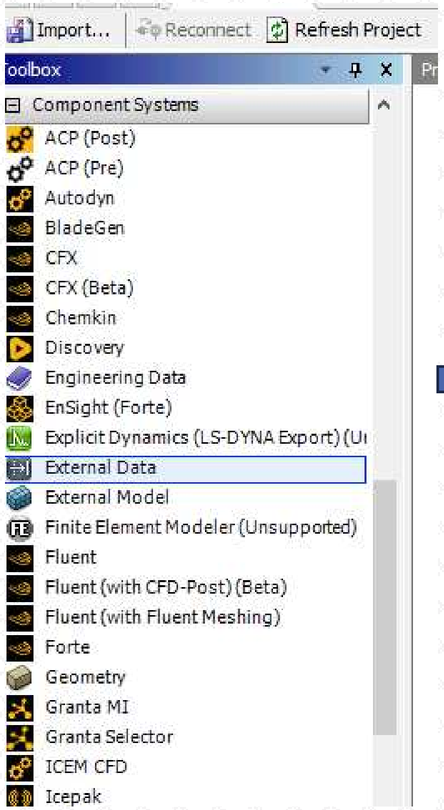
source model (transient thermal)

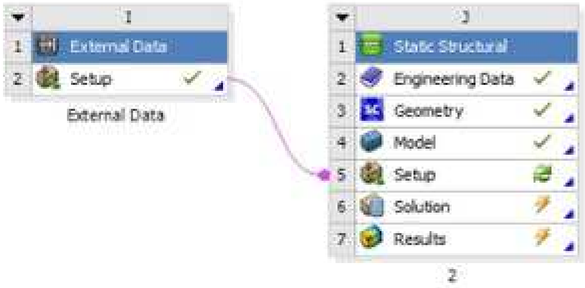


target model static structural model (static structural)

The Problem Statement: Data Mapping

- As mentioned previously, the data transfer mechanism shown on slide 3 is not appropriate for this type of mapping (the entire target domain will not be interpolated)
- Instead, Ansys supplies users with the External Data tool for this purpose (from the Toolbox in the Workbench Project Schematic)





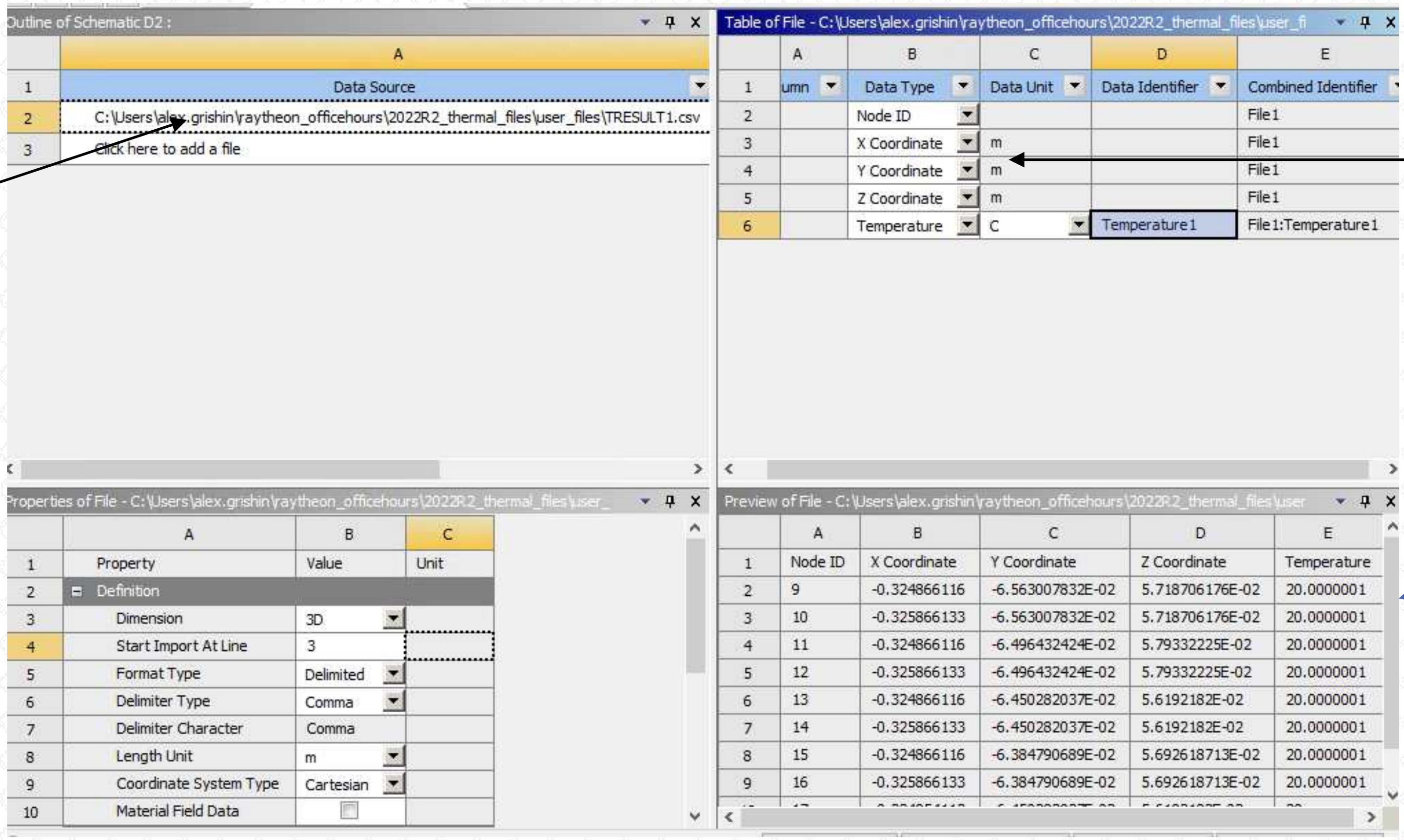
- The External Data tool may be used to transfer a result in the form of ascii text files from one system to another
- But the input text files still need to be generated...

The Problem Statement: Data Mapping

- The External Data tool is meant to be a general purpose tool for interpolating data defined at any collection of spatial points onto the nodes of any ansys mesh
- The data must be stored in a column-delimited text file

- Browse to file location

- Specify what data is stored in each column



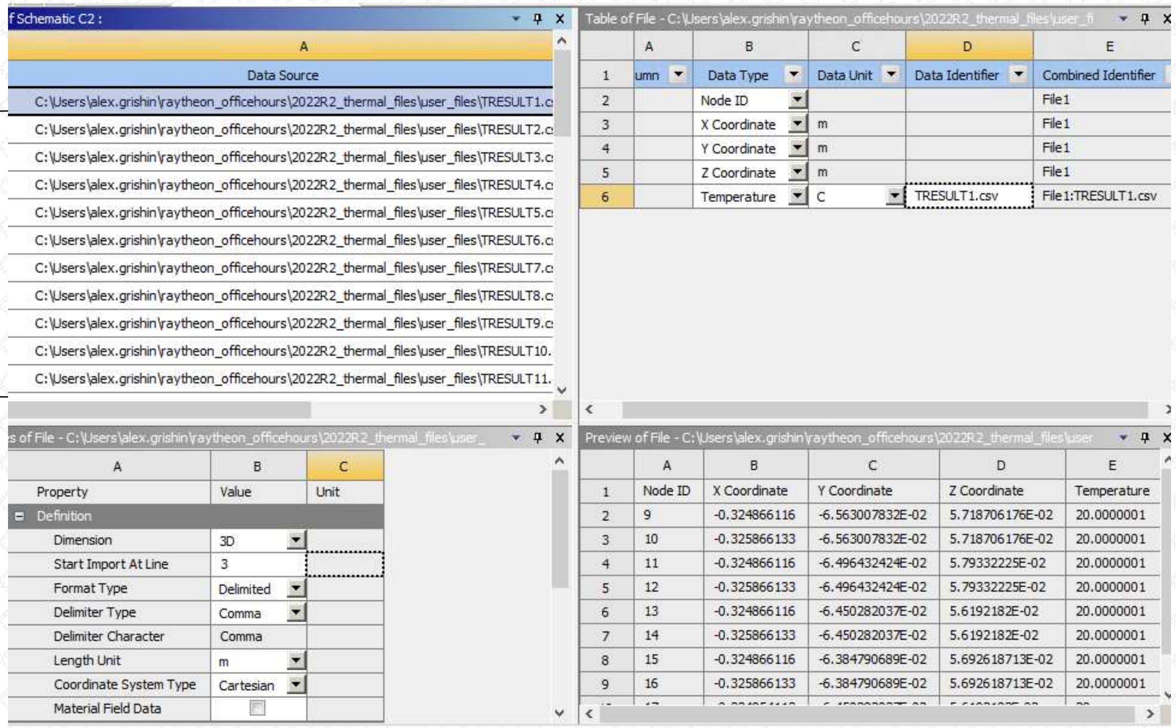
- Specify dimensionality of data, coordinate system, and delimiter type here

- Parsed results are shown here

The Problem Statement: Data Mapping

- But what if you want to map data for multiple load steps (time points)?
- The External Data tool requires that the user create and read the nodal (point) data for each time point –that is: one file per load step as shown below
- This is obviously very tedious for many time steps. So, we'll begin by showing how this process may be automated using ACT and APDL

- one file per load step



The Problem Statement: Data Mapping -Automating Data Export

- There are two main tasks we have to automate here. The first is the export of the data (temperature in our case. In this example, we'll be mapping temperatures from an ANSYS model to another ANSYS model)
- This can be done (at least) two ways
    1. Using Mechanical's Python scripting capability (part of the ACT framework)
    2. Using and APDL Commands object
- We'll start with the Python scripting approach
- We've provided a Python script ('textport.py'), as well as the APDL Commands ('writetemps.inp') for writing out the necessary (temperatures) which accompany this article

textport.py

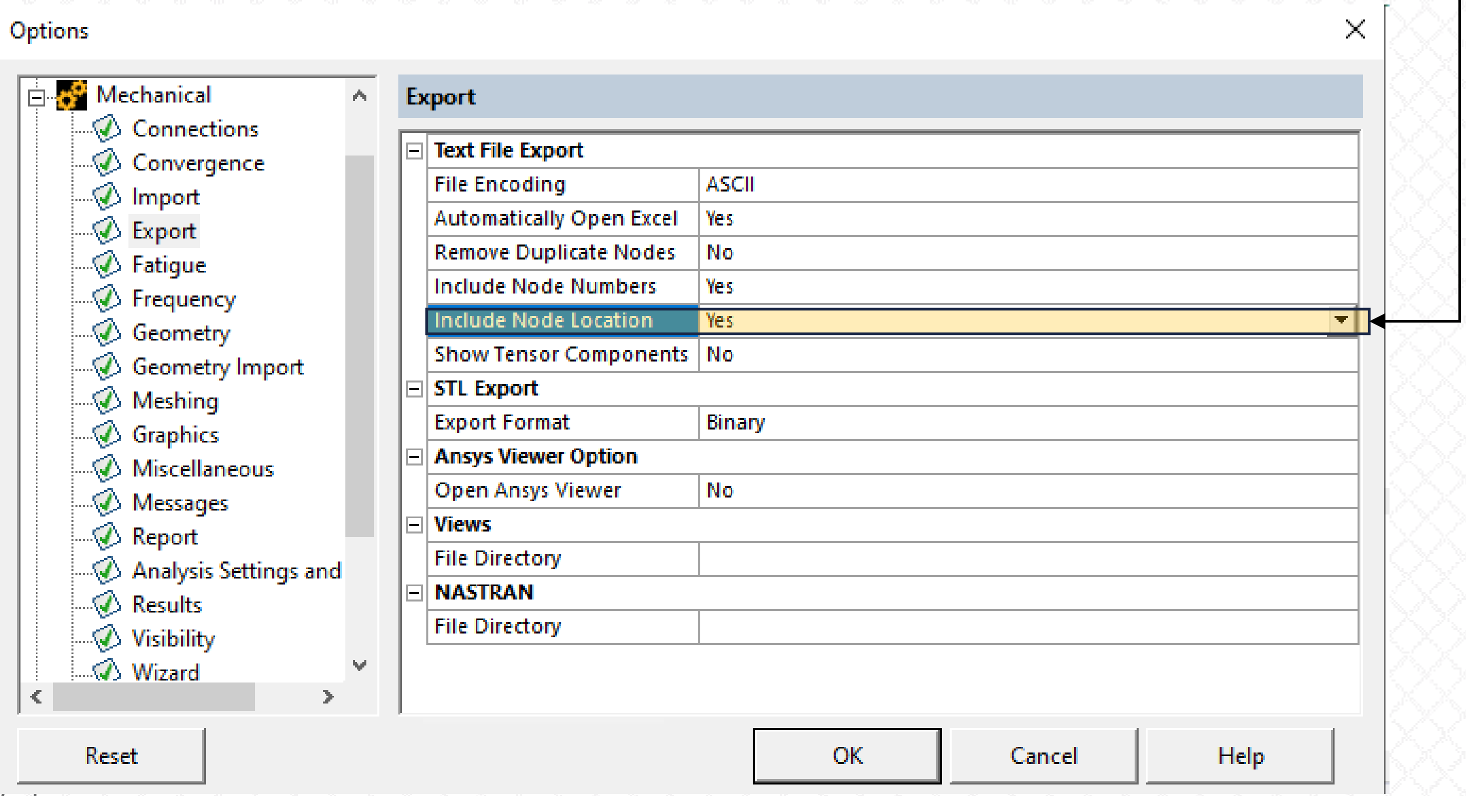
```
'''
This is a Python script used to export scalar results to N
text files (where N is the number of load steps in the result)
from a Workkench thermal or structural model from the Mechanical
interface (either from the Automation->Scripting editor
or withe use of a custom button). To make the data interpolatable
(using the External Data tool, for example), users should first modify
the Export settings to include location information. this can be done
by going to File->Options->Export and setting "Include Node Locations"
to "yes".

The text data should be in the following format:
column 1     column 2     column 3     column 4     Column5
--------     --------     --------     --------     -------
Node Id      X-coord.     Y-coord.     Z-coord.     Temperature

'''
# Script:
import os
import shutil

fpath = ExtAPI.DataModel.Project.Model.Analyses[0].AnalysisSettings.SolverFilesDirectory
reader = ExtAPI.DataModel.Project.Model.Analyses[0].GetResultsData()

userpath = os.path.dirname(fpath)
for i in range(3):
    userpath = os.path.dirname(userpath)
userpath = os.path.join(userpath,'user_files')
```

writetemps.inp

```
!This is an APDL script used to export temperature results to N
!text files (where N is the number of load steps in the result)
!from an Ansys thermal or structural model from the Mechanical
!interface or from the MAPDL interface. If using the Workbench
!interface (from a Commmands object), users should first make
!sure that the Ansys db file is getting written. This can be
!accomplished by setting Analysis Settings->Analysis Deta
!Management->Save MAPDL db to "Yes" in the Mechanical tree
!outline
!
!The text data should be in the following format:
!column 1    column 2     column 3     column 4     Column5
!--------     --------    --------     --------     -------
!Node Id        X-coord.     Y-coord.     Z-coord.     Temperature

fini
resume

/post1
*get,nsets,active,,set,nset
set,last
allsel,
*get,maxn,node,0,num,max
*del,nnums,,nopr
*del,nmask,,nopr
*del,coords,,nopr
*dim,nnums,,maxn
*dim,nmask,,maxn
```

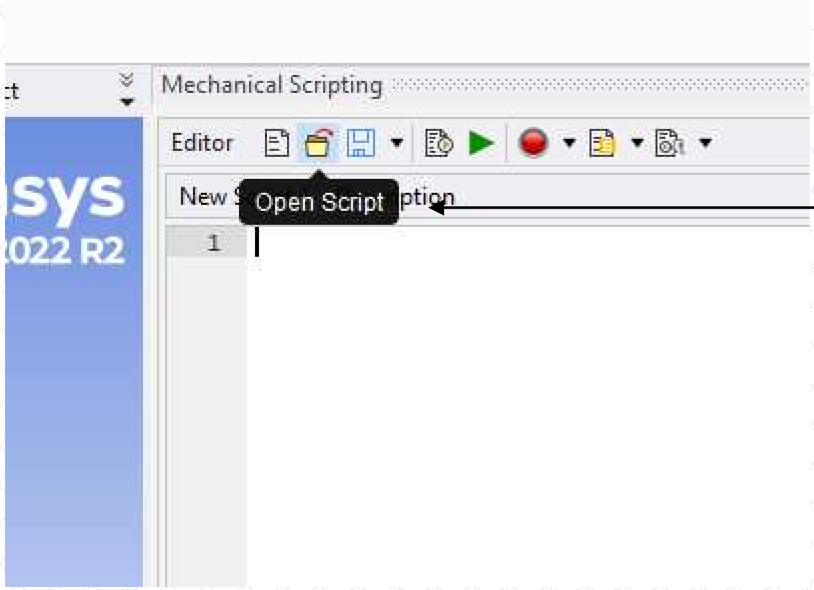
The Problem Statement: Data Mapping
-Automating Data Export: Python Script

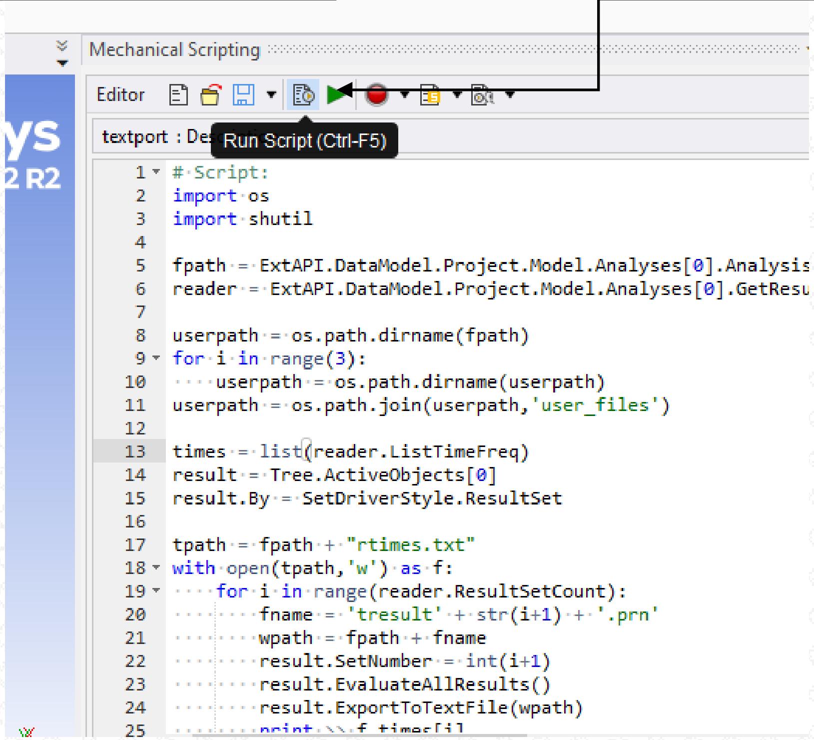
- Before using the 'texport.py' data exporting script, users should first ensure that exported data will include nodal locations (the Mechanical default settings do not do so).
- This can be done by going to File->Options->Export and set 'Include Node Location' to 'Yes' from the Mechanical interface



The Problem Statement: Data Mapping
-Automating Data Export: Python Script

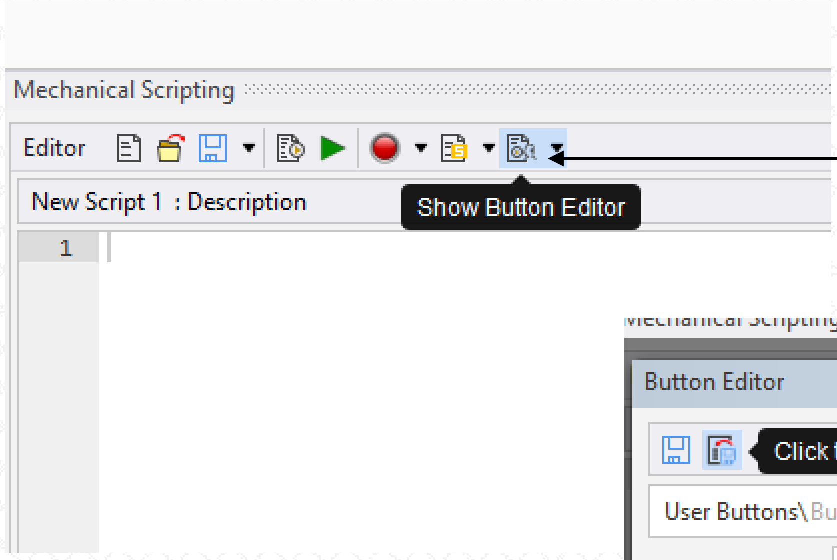
- Users can then invoke ‘textport.py’ to export temperatures by first highlighting temperature result to be written (the Python script identifies which result is currently highlighted) and then going to Automation->Scripting from the top Mechanical Menu tab
- In the Mechanical Scripting editor, browse to ‘textport.py’ by selecting ‘open script’, followed by ‘Run Script’



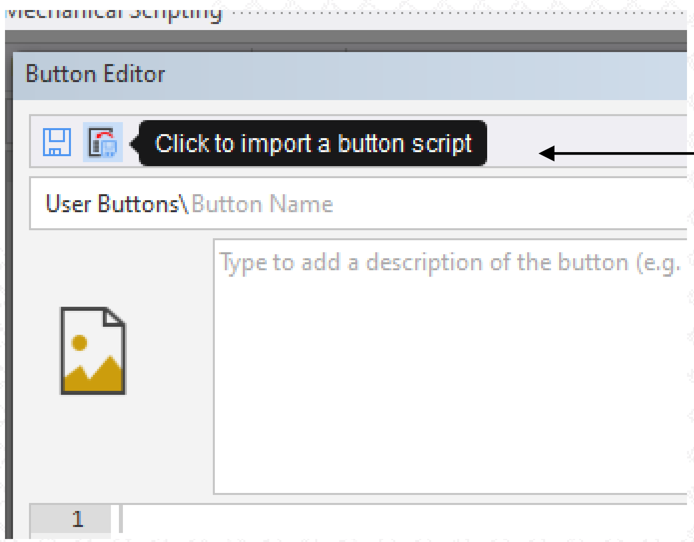


The Problem Statement: Data Mapping
-Automating Data Export: Python Script

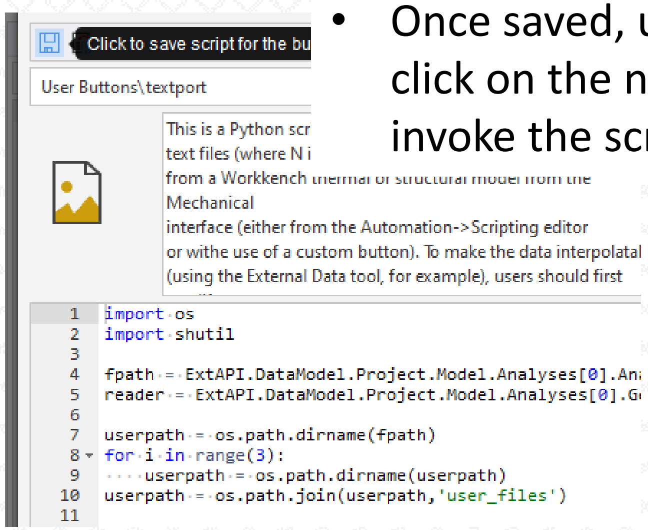
- Users can also create button to invoke this Python script by clicking ‘Show Button Editor’



- Next, click the button to import a button script

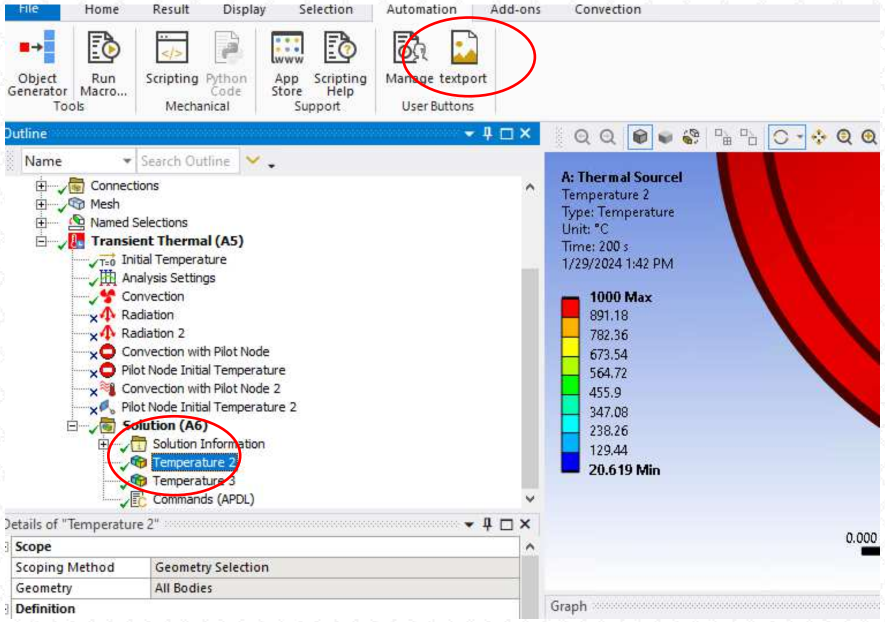


- Finally, click to save the button
- Once saved, users can just click on the new button to invoke the script



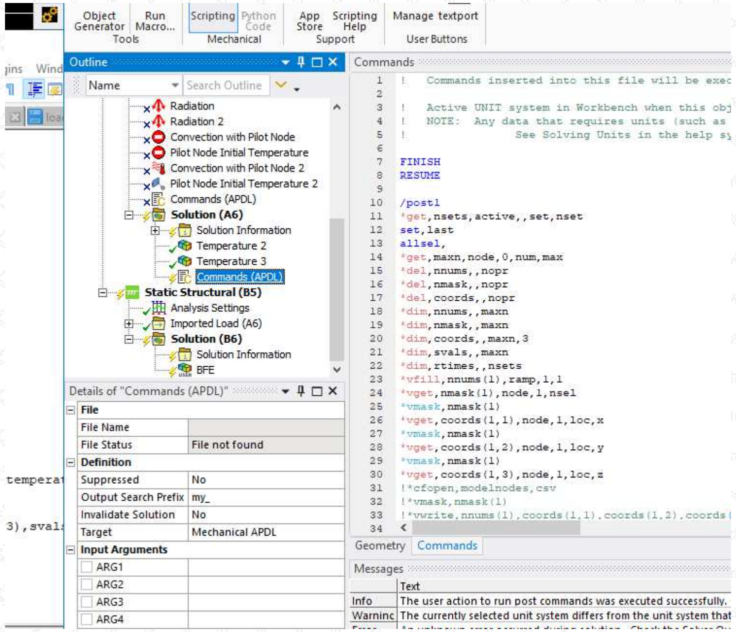
The Problem Statement: Data Mapping
-Automating Data Export: Python Script

- Highlight the result object you want to server as the source of the temperature data and just hit the newly created ‘textport’ button
- The script will run through all the time steps available for the result object and export files in a format readable by the External Data tool
- For the purpose of this tutorial, the script performs one additional operation: It moves the resulting files to the Project’s user_files folder



The Problem Statement: Data Mapping
-Automating Data Export: APDL Commands

- The APDL Commands used to export temperature data for all the nodes of the model are stored in file 'writetemps.inp' accompanying this article
- An APDL Commands object using those commands has also been provided in the Workbench archive accompanying this article: 2022R2_Themal.wbpz (shown below)



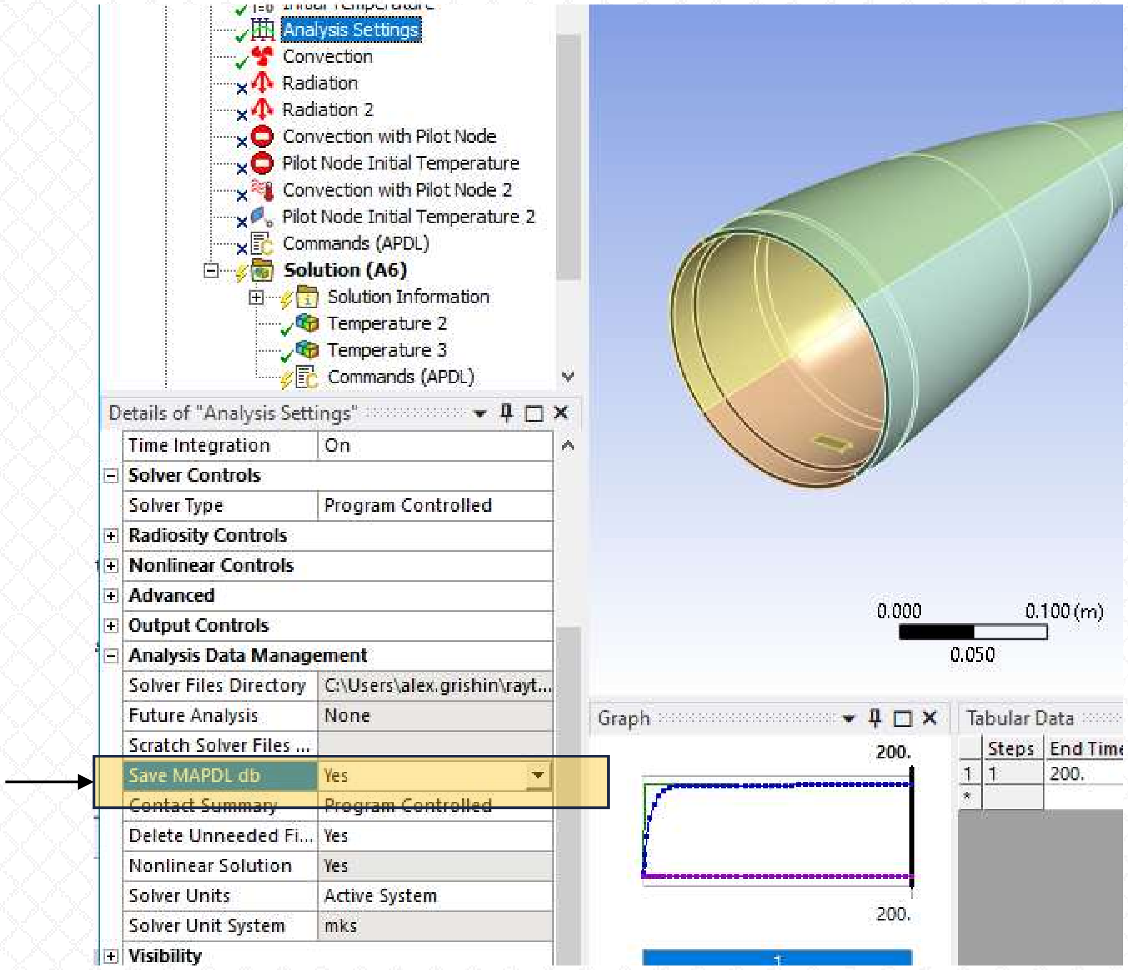
The Problem Statement: Data Mapping
-Automating Data Export: APDL Commands

- Before this script can be used (after download, for example), users should make sure that the Mechanical application is configured to write the MAPDL db file

- This can be done by going to Analysis Settings and setting 'Save MAPDL db' to 'Yes' under 'Analysis Data Management' (this is already set in the archive accompanying this article, but it's good practice to double-check)

- The Commands object should run automtically after the model is solved (and it is unsuppressed)

- To invoke the script between solves, right-click on it and select "Execute Post Commands"



The Problem Statement: Data Mapping
-Automating Data Export

- The text file data format using the Python script is different than the one using the APDL code simply because the default method for exporting Mechanical results to text file (ExportToTextFile()) does not provide a delimiter option (getting around this limitation would result in considerably more code, so we'll omit that exercise for now)
- Instead, this function simply creates **tab** (^t) delimited files
- But Microsoft Excel also knows how to read tab-delimited files, and already associates the file extension 'xls' with such a format (which is why we –and presumably the developers --use this extension for exporting text files*)
- The text file data format used by the APDL script is comma-delimited (which is more convenient in that language), but note that both file formats nevertheless use th e'xls' file extension, because excel recognizes both delimiters when encountering this extension

*The xls file extension for spreadsheets is a legacy format (97-2003), and users will always recieve a warning when opening one of these files with newer versions of Microsoft Excel

The Problem Statement: Data Mapping -Automating Data Transfer

- Once the solution data from the ‘source’ model has been created, it then needs to be read into an External Data object
- As we mentioend on slide 9, doing this manually can be very tedious (it is not uncommon to generate output for hundreds of solution times), so we’ve supplied users with a Workbench Journal script for doing this automatically*

loadfiledata.wbjn

```
#SetScriptVersion(Version="22.2.192")
"""
This is a Python Journal script which creates an External Data object populated by ascii
text output files residing the user_files folder of this project. these files are themselves
created by other scripts. Finally, the script transfers the data content of the external files
to an imported load object in a 'target system' within the same project
"""
import string
import glob
import os

targetSystemName = "Static Structural Target"   #target system for imported load object
#path to source data files (currently the user_files folder of this project)
filepath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\user_files"
#delimiterIs="Tab"               #for reading files written by the ACT extension
#delimiterStringIs=r"\t"         #for reading files written by the ACT extension
delimiterIs="Comma"              #for reading files written by the APDL Commands object
delimiterStringIs=","            #for reading files written by the APDL Commands object
ext = "xls"                      #for ACT extension OR APDL: The xls extension can accomodate comma or tab delimiter
lunit = "m"                      #length units for loads data transfer
startLine = 2                    #Starting line to parse exteran data files

#path to source data times
timepath = os.path.join(filepath,'RTIMES.txt')

template1 = GetTemplate(TemplateName="External Data")
```

*This is still an IronPython script which falls under the common ACT paradigm, but for some reason, the developers call scripting at the Project Schematic level ‘journaling’ and have adopted a ‘wbjn’ file extension for these scripts. The only difference between coding at this level and within a specific application (Mechanical, for example) is that you’re in a different namespace (don’t have access to local modules like ExtAPI)

The Problem Statement: Data Mapping -Automating Data Transfer

- The journal file ‘loadfiledata.wbjn’ does most of the ‘heavy lifting’ of both reading in the ascii text data to be transfered, as well a sending the necessary ACT code to the target system for populating the downstream ‘imported load’ object in the system tree outline

- Lines 12 thru 21 contain global variables which control most of the behavior users may need modify
- For example, line 14 defines the ‘filepath’ variable which contains the path to the source file data (the ascii text files to be read)

```
12  targetSystemName = "Static Structural Target"   #target system for imported load object
13  #path to source data files (currently the user_files folder of this project)
14  filepath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\user_files"
15  #delimiterIs="Tab"                #for reading files written by the ACT extension
16  #delimiterStringIs=r"\t"          #for reading files written by the ACT extension
17  delimiterIs="Comma"               #for reading files written by the APDL Commands object
18  delimiterStringIs=","             #for reading files written by the APDL Commands object
19  ext = "xls"                       #for ACT extension OR APDL: The xls extension can accomodate comma or tab delimiter
20  lunit = "m"                       #length units for loads data transfer
21  startLine = 2                     #Starting line to parse exteran data files
```

The Problem Statement: Data Mapping -Automating Data Transfer

- In addition to creating the necessary External Data object, ‘loadfiledata.wbjn’ also ‘sends’ ACT code necessary to configure and populate the target ‘imported load’ object
- It does so using the ‘SendCommand’ method of the target model container
- This, in turn, relies on a technique of packaging the downstream code in a formatted raw string, as demonstrated here (that blog post provided the inspirpation for this solution)
- The string in question may be found on lines 88 thru 117 (shown below)
- We’re also including a standalone python file containing this code (‘importACTscript.py’) if users want to modify this for their own purposes

loadfiledata.wbjn

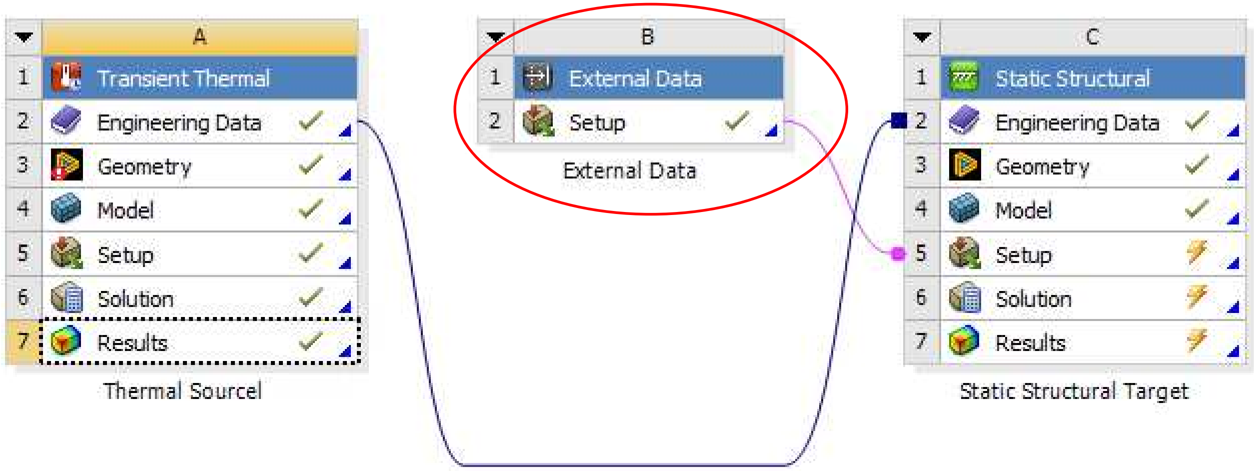
```
88  scriptCommands=r"""
89  with Transaction():
90      import glob
91      import os
92  
93      datapath = r'{0}'
94      timepath = r'{1}'
95      fext = '{2}'
96      resultfiles = glob.globl(datapath,"*."+fext)
97      resultfiles.sort(key=lambda f: int(''.join(filter(str.isdigit, f))))
98      analysis = ExtAPI.DataModel.Project.Model.Analyses[0]
99      with open(timepath,"r") as r:
100         data = r.read().strip()
101         datalist = data.split('\n')
102     times = map(float,datalist)
103     numfilestoload = len(resultfiles)
104     importedloadobjects = [child for child in analysis.Children if child.DataModelObjectCategory.ToString() == "Imp
105     lastimportedloadobj = importedloadobjects[-1]
106     importedTemperature = lastimportedloadobj.AddImportedBodyTemperature()
107     selection = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
108     selection.Ids = [6, 22, 27, 46, 65, 83, 102, 121, 139, 158, 177, 539, 540, 541, 542, 543]
109     importedTemperature.Location = selection
110     table = importedTemperature.GetTableByName("")
111     for i in range(numfilestoload-1):
112         table.Add(None)
113     for i in range(numfilestoload):
114         table[i][0]="File"+str(i+1)+":"+str(resultfiles[i])
115         table[i][1]=times[i]
116     importedTemperature.ImportLoad()
117 """.format(filepath,timepath,ext)
```

importACTscript.py

```
import glob
import os

datapath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\user_files"
timepath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\user_files\RTIMES.txt"
fext = "prn"
resultfiles = glob.globl(datapath,"*."+fext)
resultfiles.sort(key=lambda f: int(''.join(filter(str.isdigit, f))))
analysis = ExtAPI.DataModel.Project.Model.Analyses[0]
with open(timepath,"r") as r:
    data = r.read().strip()
    datalist = data.split('\n')
times = map(float,datalist)
numfilestoload = len(resultfiles)
importedloadobjects = [child for child in analysis.Children if child.DataModelObjectCategory.ToString() == "Import
lastimportedloadobj = importedloadobjects[-1]
importedTemperature = lastimportedloadobj.AddImportedBodyTemperature()
selection = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
selection.Ids = [6, 22, 27, 46, 65, 83, 102, 121, 139, 158, 177, 539, 540, 541, 542, 543]
importedTemperature.Location = selection
table = importedTemperature.GetTableByName("")
for i in range(numfilestoload-1):
    table.Add(None)
for i in range(numfilestoload):
    table[i][0]="File"+str(i+1)+":"+str(resultfiles[i])
    table[i][1]=times[i]
importedTemperature.ImportLoad()
```

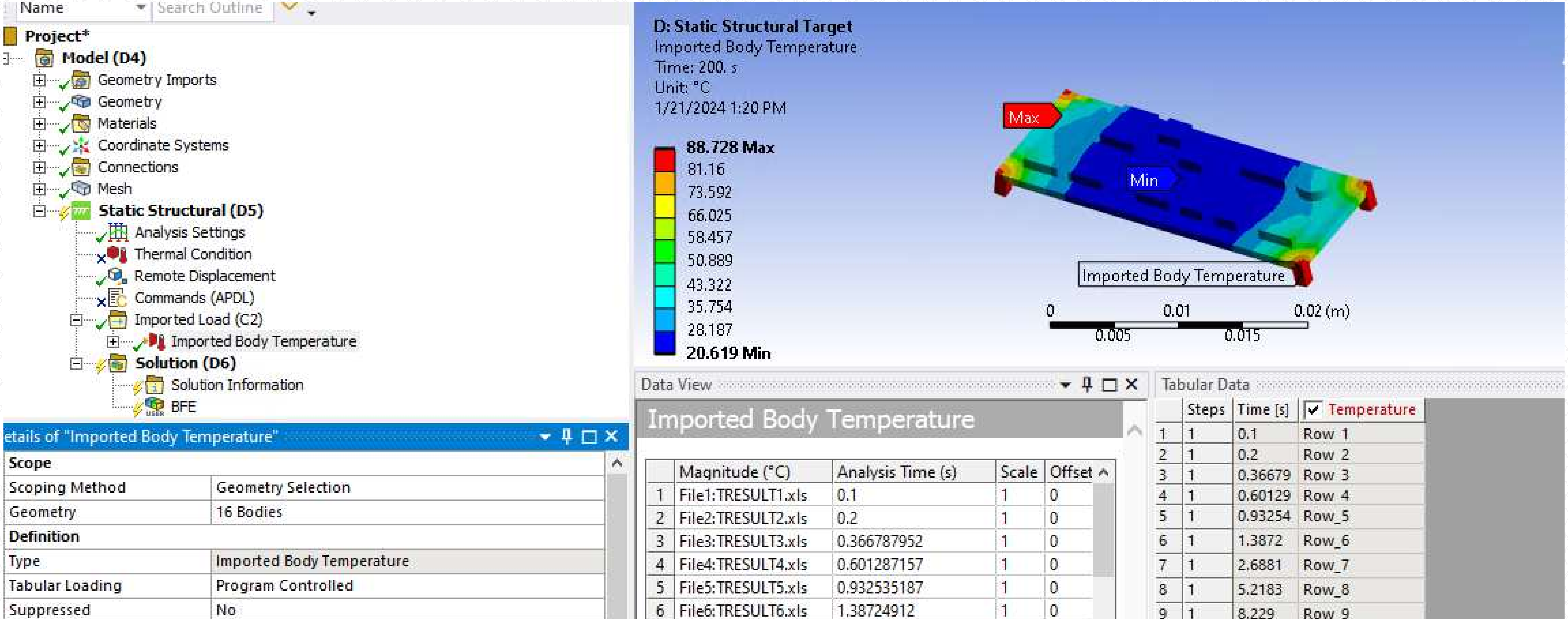
The Problem Statement: Data Mapping -Automating Data Transfer

- Once the necessary ascii text result files have been generated (or otherwise exist), the journal file ‘loadfiledata.wbjn’ may be invoked from the Project Schematic by going to File->Scripting->Run Script File and navigating to ‘loadfiledata.wbjn’
- Upon doing so, users will notice a new External Data object connected to the Setup cell of the target system as shown below



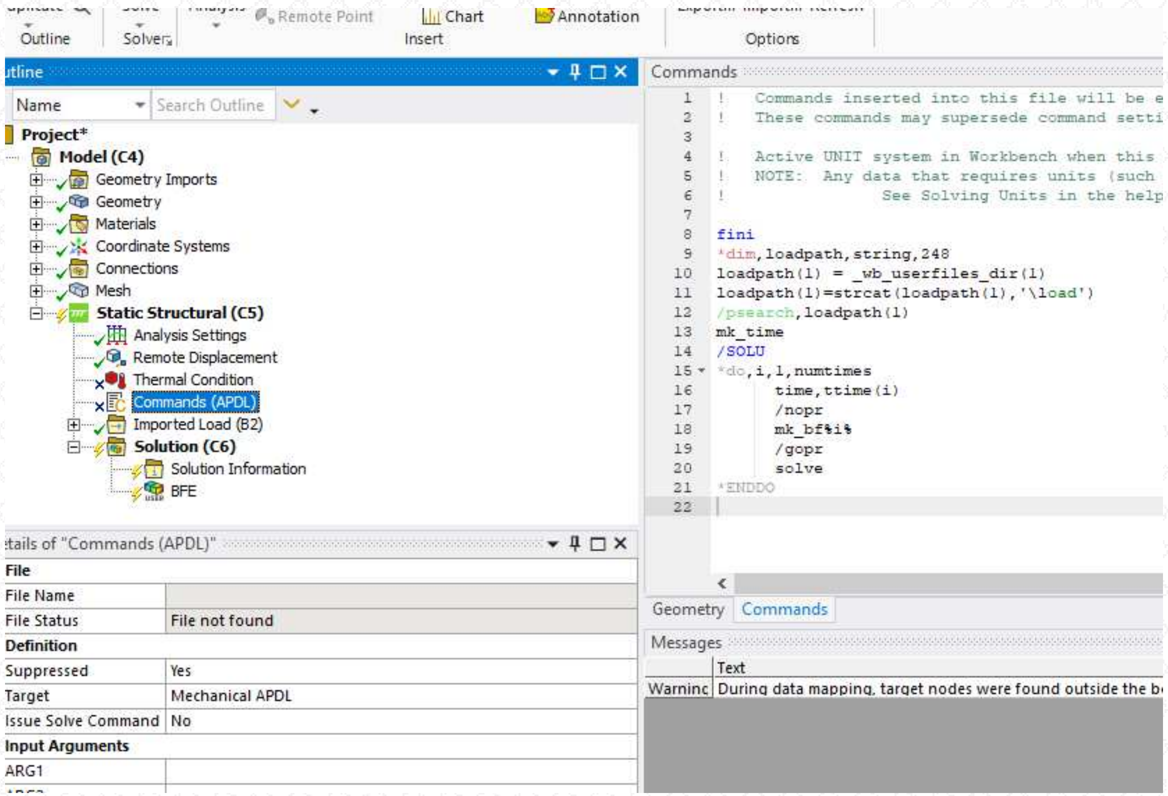
The Problem Statement: Data Mapping -Automating Data Transfer

- In addition, the target Mechanical system is opened (by line 86: setup2.Edit()), and the ACT script shown on slide 19 is transferred to the setup cell, and the imported load object is configured and upated....



The Problem Statement: Data Mapping -Automating Data Transfer

- To run the model, just make sure that the Commands object (used for the PyAnsys mapping method) is suppressed



The Problem Statement: Data Mapping -Automating Data Transfer

- For this particular model problem, the process runs fairly quickly. The model statistics are as follows:
    - Source Data: 88707 points
    - Source Times (Load Steps): 48
    - Target Space: 131697 points

- The data transfer step (running file 'loadfiledata.wbjn') took approximately 2 ½ minutes
- But this problem does not scale well (much worse than linear) using this technique
- A common requirement is to map CFD temperature and fluid prssures onto a structural model. Such models may contain millions of source points over hundreds of load steps. Such a problem could easily take hours (or more) to transfer data using the techniques shown here
- A much more efficient (and compact) method involves the newer PyAnsys tools (discussed next). We recommend the second option (using ansys.dpf.post to map temperatures read from text files) for that scenario

Part 2: Mapping Data:

PyAnsys:

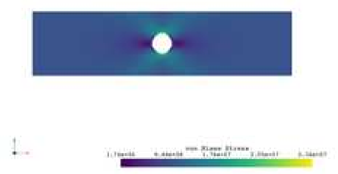
- ansys.mapdl.reader
- ansys.dpf.post

Data Mapping with PyAnsys
-ansys.mapdl.reader

- The PyAnsys Suite offers several tools which are capable of mapping data from one mesh to another
- As we'll see shortly, this is because in order to so, they all rely on a very powerful and general mesh-manipulation utility called PyVista (which is itself a high-level Python wrapper around the popular VTK). This utility comes with a very efficient data mapping functionality which we'll make use of this section
- We'll focus first on PyMAPDL Reader
- Ansys tells us this is a 'legacy' reader (because it provides an older direct link to the ansys database instead of using the newer DPF technology)
- The only reason we're doing this is becasue the newer DPF Post will not read a thermal result database (.rth file) –presumably, this is planned, but not yet available while the legacy reader *will* do this now (see below)

The `ansys-mapdl-reader` module supports the following formats:

- `*.rst` - Structural analysis result file
- `*.rth` - Thermal analysis result file
- `*.emat` - Element matrice data file
- `*.full` - Full stiffness-mass matrix file
- `*.cdb` or `*.dat` - MAPDL ASCII block archive and Mechanical Workbench input files



PyMAPDL Reader

Pythonic interface to read legacy MAPDL result files (MAPDL 14.5 and later)

Data Mapping with PyAnsys
-ansys.mapdl.reader

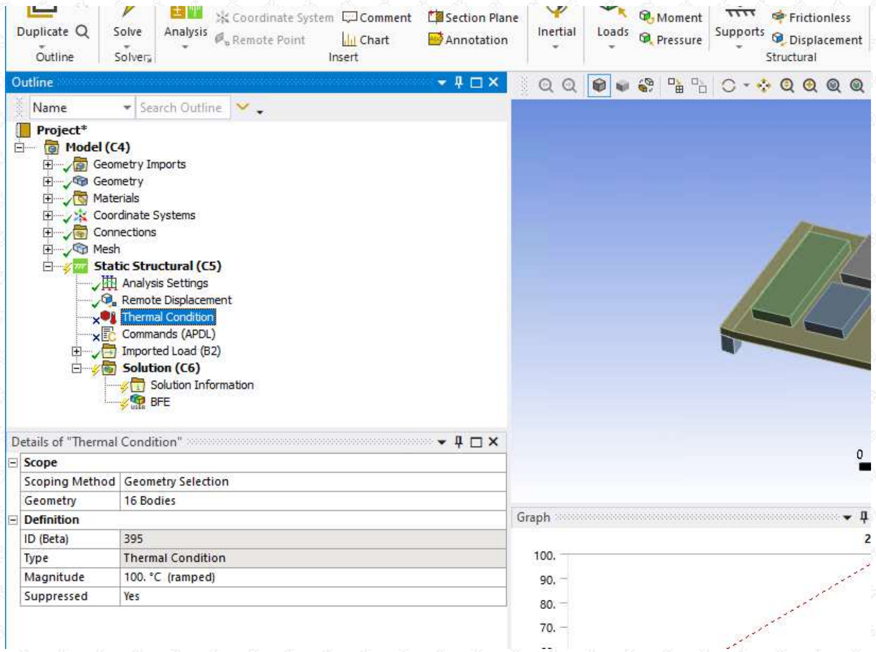
- We provide a Python script called ‘tmap.py’ accompanying this article which maps the temperature solution from the source model onto the target model mesh, and then generates corresponding temperature loads for the target model in the form of APDL commands
- In other words: it does everything that the Workench project script ‘loadfiledata.wbjn’ does, but in a different way.

tmap.py

```
"""
This script imports a 'source' model, copies all its temperature results to its PyVista 'grid'
and then interpolates the results onto the PyVista grid of the target model. It then writes out
the interpoated temperature data to a 'loads' folder (under the current project's user_files folder)
in the form of APDL loads to be applied to the target model.

"""

import os
import ansys.dpf.core as dpf
from ansys.dpf import post
from ansys.mapdl.core import launch_mapdl
from ansys.mapdl import reader
import numpy as np
import pyvista as pv

#path to target model result file (.rst file. This is the Mechanical solution folder)
#rpath = r"C:\Users\alex.grishin\raytheon_officehours\baseline_files\dp0\SYS\MECH\file.rth"
#path to 'source' model (solved thermal model with temperatures to be mapped)
spath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\dp0\SYS\MECH\file.rth"
#path to 'target model (on which to map temperatures as structural loads)
tpath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\dp0\SYS-2\MECH\file.rst"
#path to store APDL commands for target model
loadpath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\user_files\load"

#get solution and mesh from source model
ssol = reader.read_binary(spath)
#get mesh from target model
tsol = reader.read_binary(tpath)
```

Data Mapping with PyAnsys
-ansys.mapdl.reader

- Before testing this script, note that lines 27 and 29 read the source and target result files
- For this to run successfully, make sure that both models have been run (that the necessary result files exist)
- If you haven't already solved both models, do so before running this script.



- If you don't want to run all 48 load steps, you can suppress the 'Imported Load', and unsupress the 'Thermal Condition
- The only point of this step is to provide a result file to PyAnsys
- And only the source model results matter
- from the target model, we only need the model information (not the actual results)

Data Mapping with PyAnsys -ansys.mapdl.reader

- Once PyAnsys has been installed, users can easily invoke the script from the DOS window by activating a python session as below

Command Prompt - python

```
icrosoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

:\>python
ython 3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)] on win32
ype "help", "copyright", "credits" or "license" for more information.
>> _
```

- Next, append the path to tmap.py and invoke it with the lines below

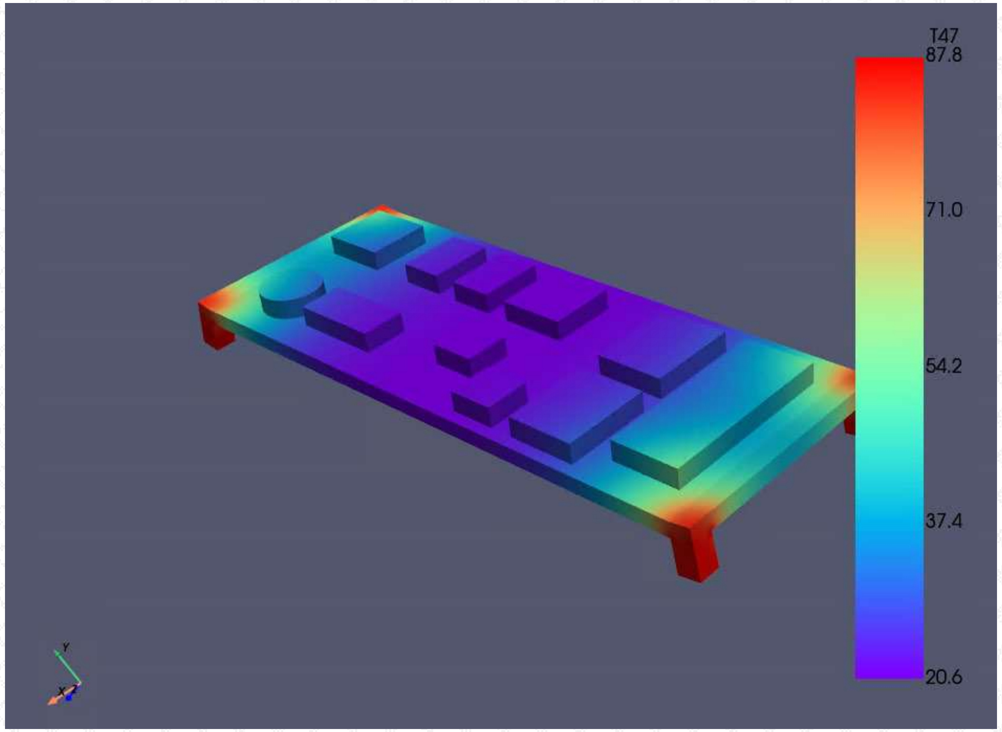
Command Prompt - python

```
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

H:\>python
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path.append(r"C:\Users\alex.grishin\raytheon_officehours")
>>> import tmap
>>> _
```

Data Mapping with PyAnsys
-ansys.mapdl.reader

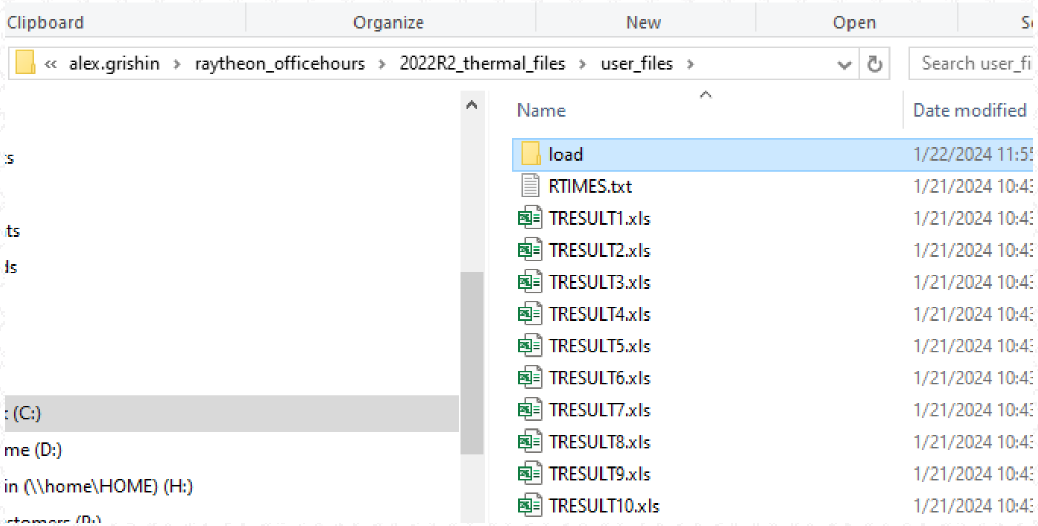
- When executing this script from the command, line 38 will plot the mapped temepratures for load step 48 for verification (below). The script will pause until this window is closed. Simply close it to continue...

```
38  tmesh.plot(scalars='T47',cmap='rainbow')
```

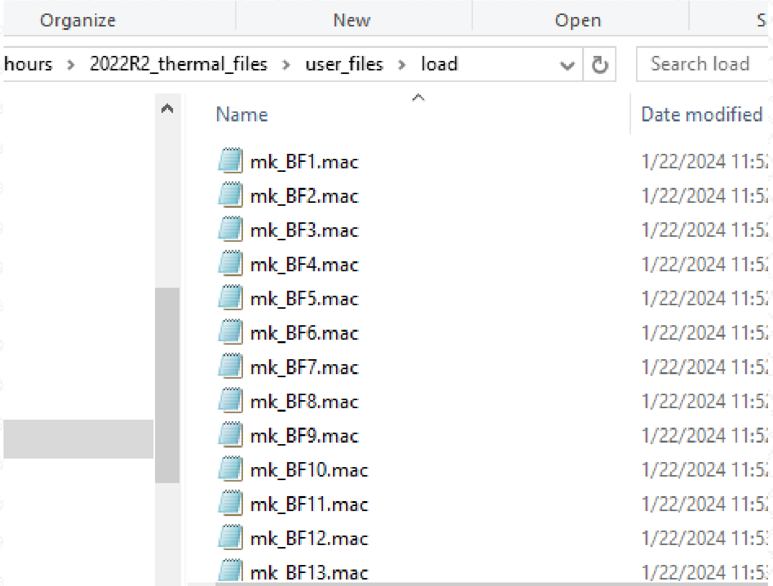


Data Mapping with PyAnsys -ansys.mapdl.reader

- When finished, users should see a new subfolder called ‘load’ in the project’s user_files folder (the same folder in which the source data files are stored)

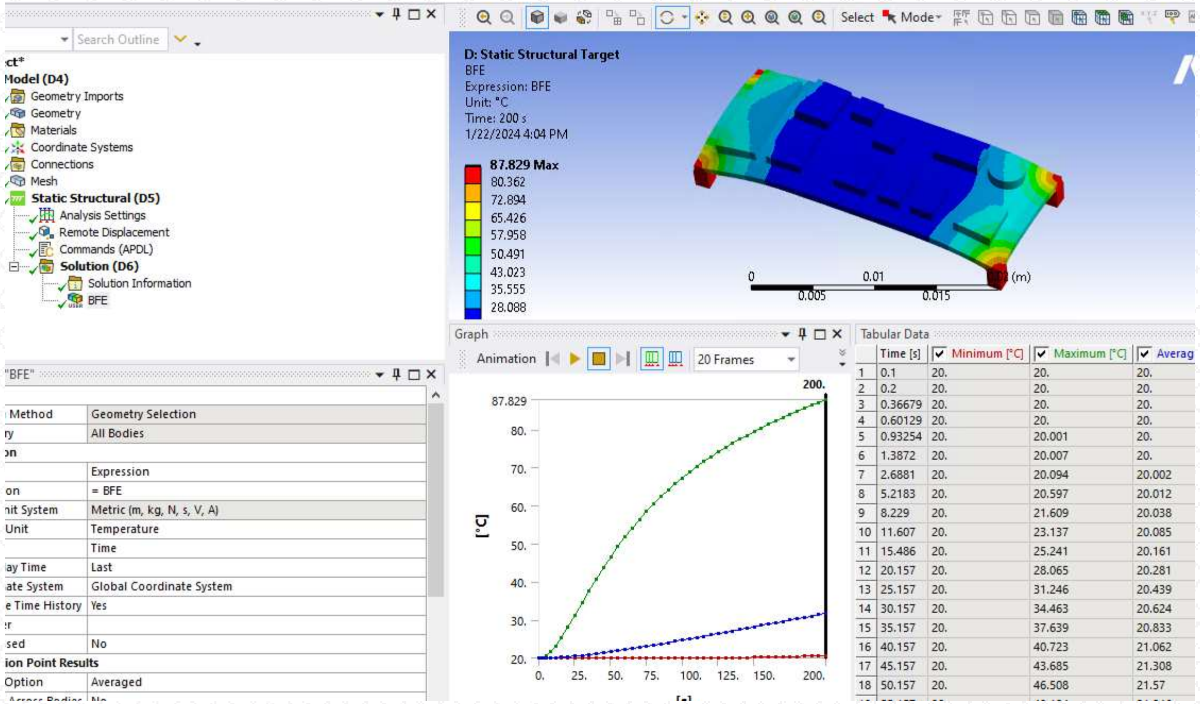


- This folder should contain the APDL load and time definitions (one file for each load step)



Data Mapping with PyAnsys
-ansys.mapdl.reader

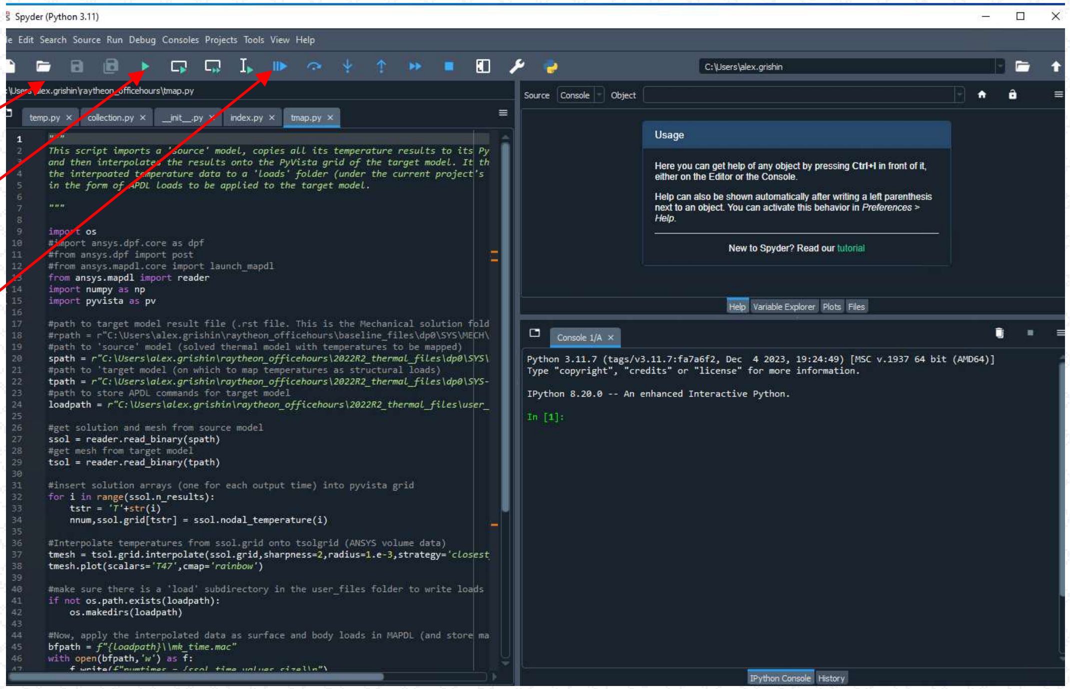
- To run the model, just unsuppress the APDL Commands object and suppress the uniform temperature load object (used to generate a 'dummy' rst file)
- Suppress or delete the Imported Load object (if it exists from a prior study) and solve
- You can verify the temperatures with the user-defined (BFE) result object supplied with the model which plots nodal temperatures from body-forces (with the APDL BFE command)



Data Mapping with PyAnsys
-ansys.mapdl.reader. How does it work?

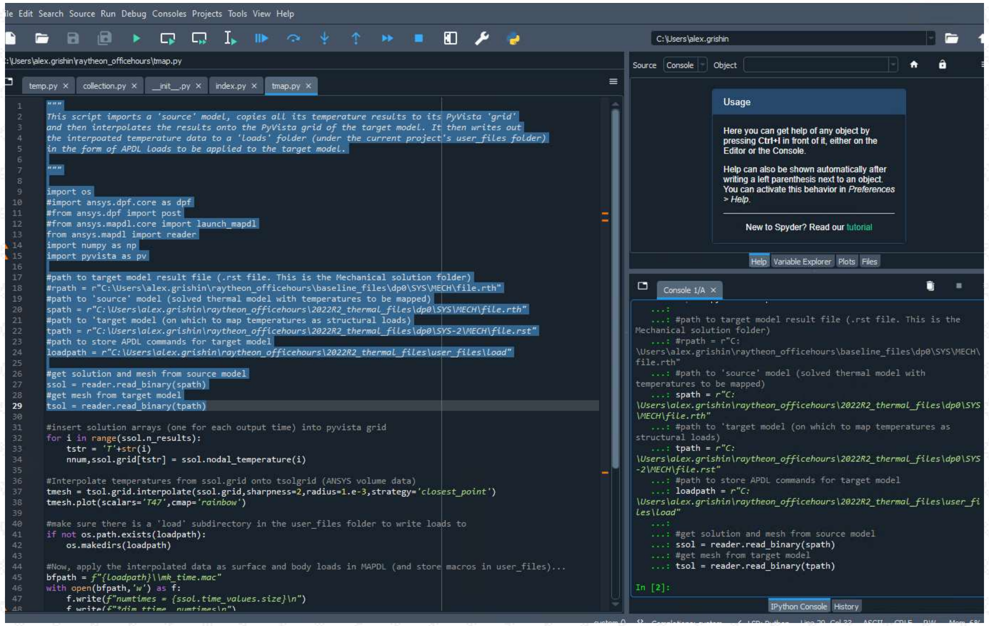
- Running Python scripts in this way is efficient (quick)
- However, when writing, editing, or testing these scripts, it is very helpful to use a modern smart editor. At PADT, we like [Spyder]
- Open 'tmap.py' in the editor. At this point, you can 'run', 'debug', or simply cut-and-paste lines into the console to test what they do.

  - open files
  - run files
  - debug files



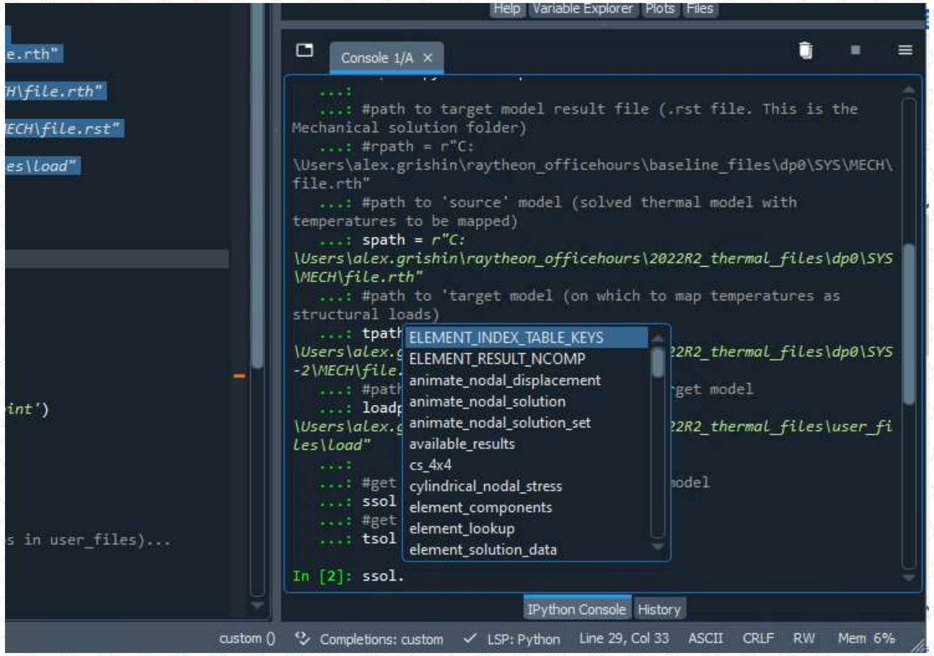
Data Mapping with PyAnsys
-ansys.mapdl.reader. How does it work?

- Cut-and-paste lines 1 thru 29 into the console (this loads the source and target databases into the session).
- Note that this requires that both source and target have already been solved (because we need a result file)
- This is why we've provided the target model with a dummy body load (suppress this before running the mapped solution)



Data Mapping with PyAnsys
-ansys.mapdl.reader. How does it work?

- ssol and tsol are the Python interfaces to the source and target models objects respecively
- Spyder provides code-completion tooltips when you type an object followd by ‘dot(.)’ and hit the ‘tab’ key
- Thus, we can see what variables and methods are available in the ssol and tsol objects using the tooltips (below)



- You can scroll through the tooltips to select the one you want

Data Mapping with PyAnsys -ansys.mapdl.reader. How does it work?

- The next three lines access the PyVista Unstructured Grid object of the source model. They loop through the stored load steps (n_results), make a string identifier (tstr) of each, and then store the corresponding nodal temperature in an array name with that identifier

```
#insert solution arrays (one for each output time) into pyvista grid
for i in range(ssol.n_results):
    tstr = 'T'+str(i)
    nnum,ssol.grid[tstr] = ssol.nodal_temperature(i)
```

- Since we use the PyVista grid functionality to interpolate the results, the first step is to fill the PyVista grid with the source solution
- To understand this a little better, pause for a momont to explore the grid obect
- Make a new object called sgrid by typing ‘sgrid = ssol.grid.copy() <enter>’

```
In [3]: sgrid = ssol.grid.copy()
```

- Type ‘print(sgrid) <enter>’

```
In [4]: print(sgrid)
UnstructuredGrid (0x15ff4c8bd00)
  N Cells:    95334
  N Points:   88707
  X Bounds:   -3.762e-01, 5.597e-02
  Y Bounds:   -9.628e-02, 9.628e-02
  Z Bounds:   -9.628e-02, 9.628e-02
  N Arrays:   28
```

Data Mapping with PyAnsys
-ansys.mapdl.reader. How does it work?

- Type ‘type(sgrid)<enter>’

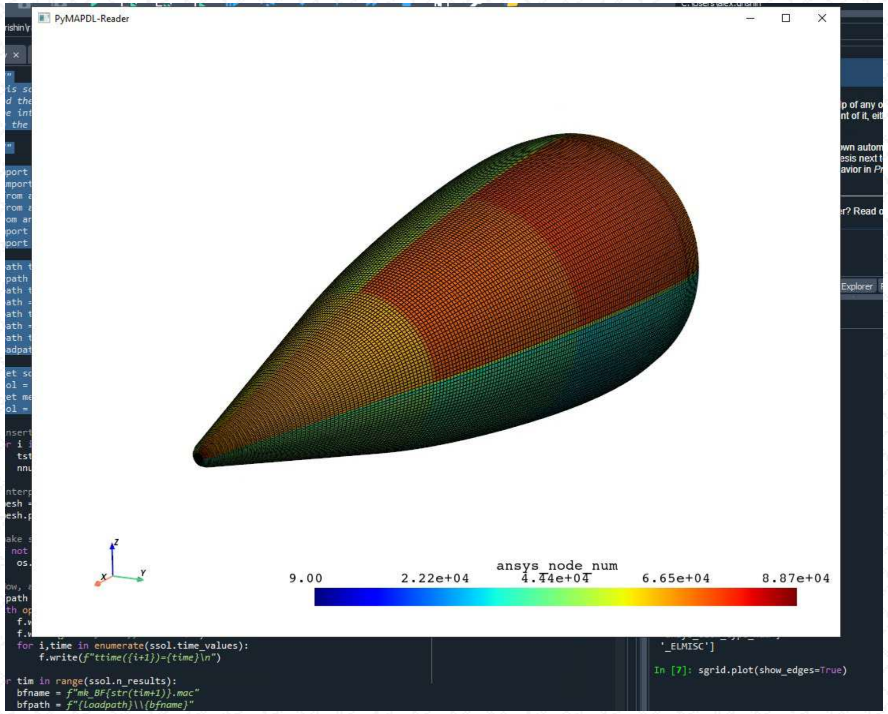
```
In [5]: type(sgrid)
Out[5]: pyvista.core.pointset.UnstructuredGrid
```

- So, we know that ‘sgrid’ is a PyVista Unstructured Grid object containing 95334 ‘Cells’ (elements), and 88707 ‘Points’ (nodes)
- We also know that it already has 28 arrays arrays stored within it
- To see what these array are, type ‘sgrid.array_names<enter>’
- The Points are indexed from 0 to 88706 while the elements are indexed from 0 to 95333
- The stored arrays are indexed accordingly
- Notice that the first array is called ‘ansys_node_num’. This array returns the ansys node id corresponding to the Point index

```
In [6]: sgrid.array_names
Out[6]:
['ansys_node_num',
 'CONTACT_REGION_5_CONTACT',
 'CONTACT_REGION_5_TARGET',
 'CONTACT_REGION_6_CONTACT',
 'CONTACT_REGION_6_TARGET',
 'CONTACT_REGION_7_CONTACT',
 'CONTACT_REGION_7_TARGET',
 'CONTACT_REGION_8_CONTACT',
 'CONTACT_REGION_8_TARGET',
 'RADIATION_SURFACES',
 'SELECTION',
 'SELECTION_2',
 'SELECTION_3',
 'SELECTION_4',
 'SELECTION_5',
 'SELECTION_6',
 'SELECTION_7',
 'SELECTION_8',
 'SELECTION_9',
 'angles',
 'origid',
 'VTKorigID',
 'ansys_elem_num',
 'ansys_real_constant',
 'ansys_material_type',
 'ansys_etype',
 'ansys_elem_type_num',
 '_ELMISC']
```

Data Mapping with PyAnsys
-ansys.mapdl.reader. How does it work?

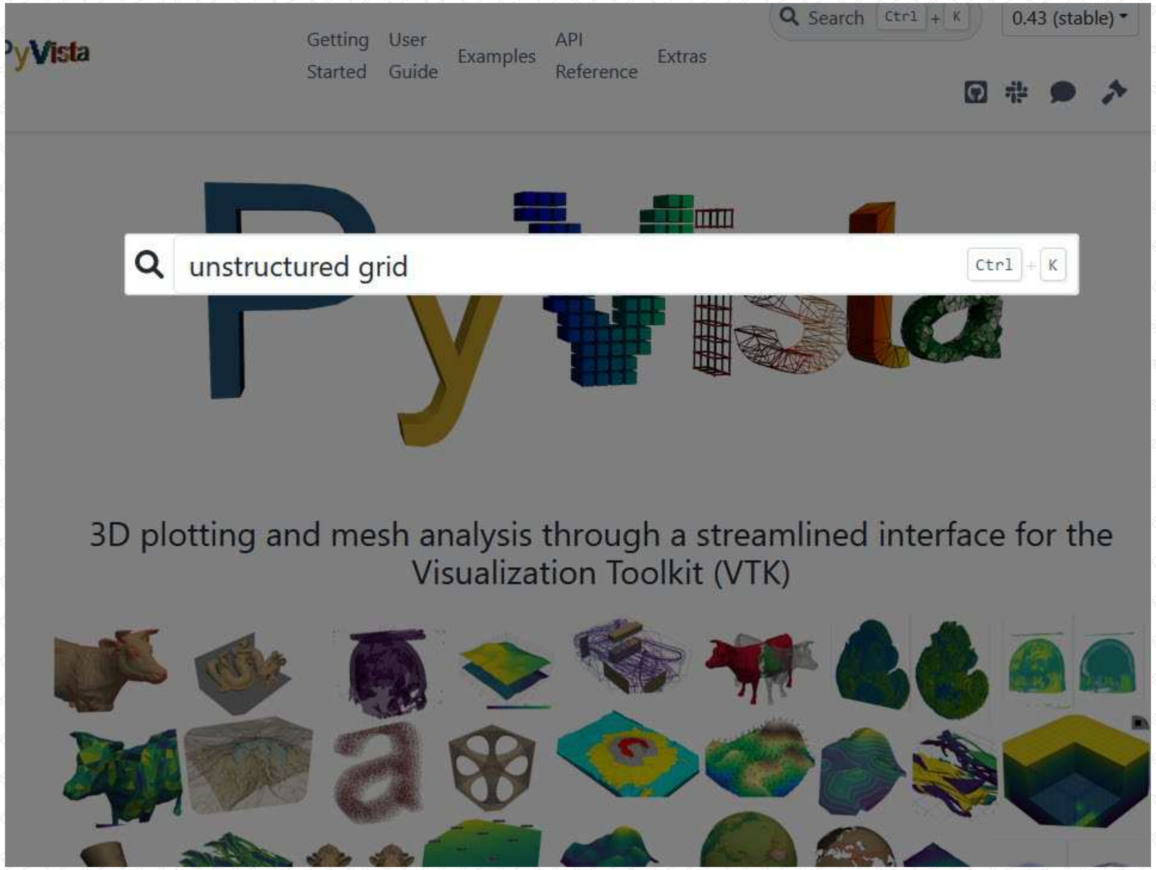
- View the mesh by typing ‘sgrid.plot(show_edges=True)<enter>’



- We’re getting a contour plot, which is simply plotting the values in the first stored array (because we didn’t specify), which happens to be the Ansys node number

Data Mapping with PyAnsys
-ansys.mapdl.reader. How does it work?

- To understand more about what you can do in PyVista, simply type your query in this link



Data Mapping with PyAnsys
-ansys.mapdl.reader. How does it work?

- Let’s continue exploring ‘tmap.py’
- Cut-and-paste the next three lines into the console (followed by <enter>)

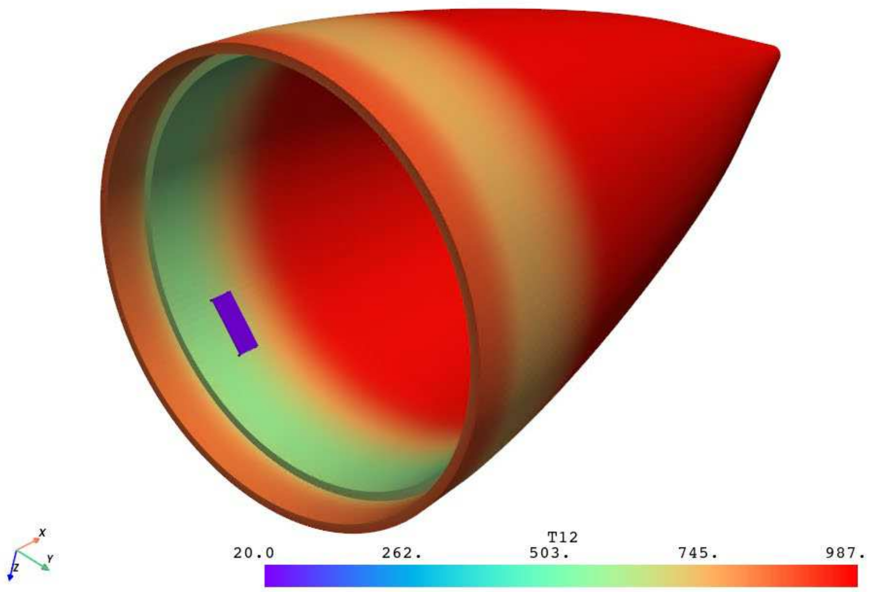
```
In [8]: #insert solution arrays (one for each output time) into
pyvista grid
   ...: for i in range(ssol.n_results):
   ...:     tstr = 'T'+str(i)
   ...:     nnum,ssol.grid[tstr] = ssol.nodal_temperature(i)
   ...:
```

- These lines copy the ‘nodal_temperature’ result (which is a tuple of node numbers and nodal temperature values for load step i) into arrays labeled ‘TN’, where N is the load step number (starting from 0)
- Thus, the N temperature results are now *also* stored in the PyVista grid object. We’ll see why this is necessary in a moment.
- Type ‘ssol.grid.array_names<enter>’ to see the new arrays we’ve created...

Data Mapping with PyAnsys
-ansys.mapdl.reader. How does it work?

- Review one of the temperature results (let's say, the 13th one), by typing the following

```
In [10]: ssol.grid.plot(scalars='T12',cmap='rainbow')
```

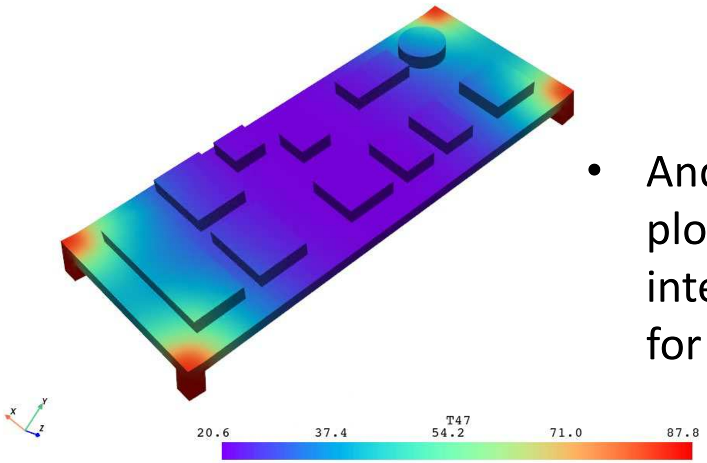


Data Mapping with PyAnsys
-ansys.mapdl.reader. How does it work?

- Now cut-and-paste the next two lines into the console.
- This is the heart the script. Line 37 interpolates the source grid arrays (which we've been reviewing. All of them) onto the target grid object
- To learn more about how this function works, see here.
- Most important. Note how long this took (a few seconds at worst). This is why this is PADT's recommended method of interpolating Ansys data (or external data onto Ansys meshes)

```
In [11]: #Interpolate temperatures from ssol.grid onto tsolgrid (ANSYS volume data)
    ...: tmesh = tsol.grid.interpolate(ssol.grid,sharpness=2,radius=1.e-3,strategy='closest_point')
    ...: tmesh.plot(scalars='T47',cmap='rainbow')
```

- The only thing you may have to modify on other models is this parameter ('radius=1.e-3')
- it defines an interpolation radius, which should be roughly equal to the target mesh element size



- And line 38 (see slide 29) plots the last interpolated time step for us for verification

Data Mapping with PyAnsys
-ansys.mapdl.reader. How does it work?

- Lines 41 and 42 check to see if the 'load' subdirectory (under 'user_files') exists. If it doesn't, create it. We're going to generate thermal loads in the form of APDL commands, and we want those to reside in their own folder

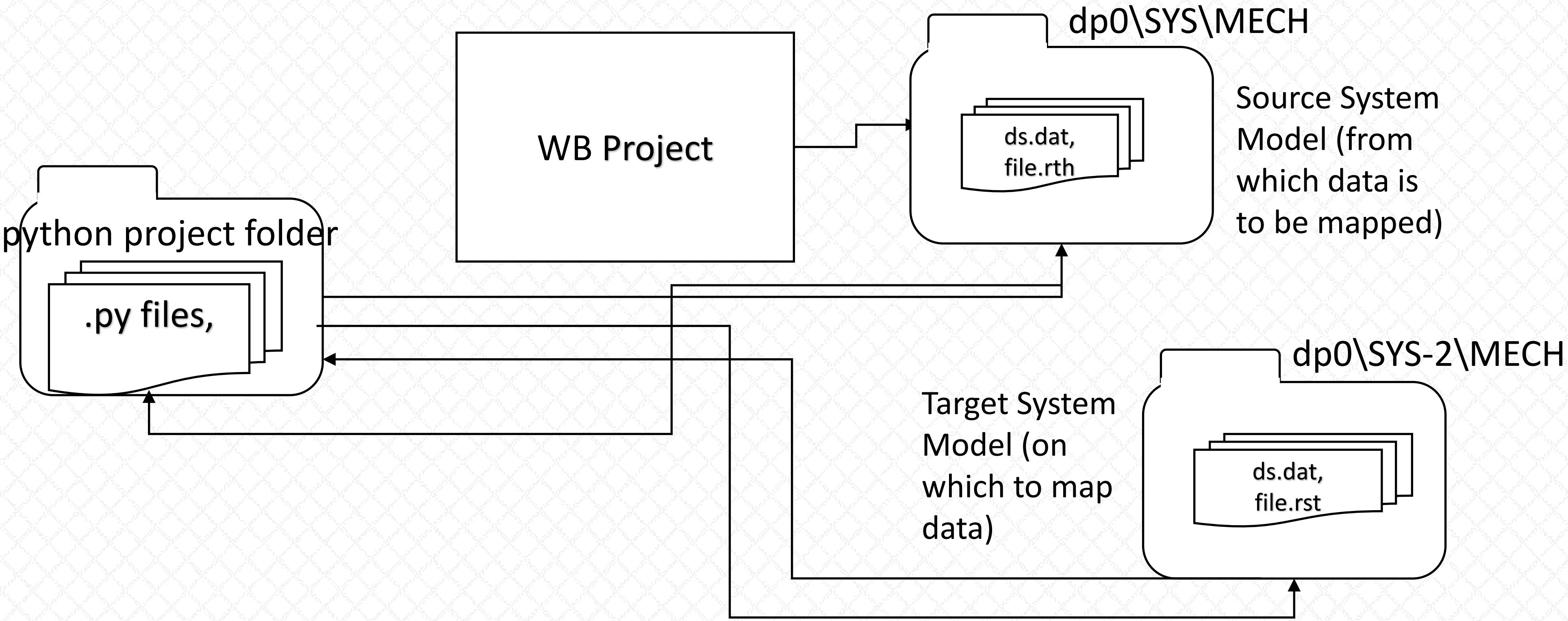
```
40  #make sure there is a 'load' subdirectory in the user_files folder to write loads to
41  if not os.path.exists(loadpath):
42      os.makedirs(loadpath)
```

- Lines 45 thru 60 generate the APDL commands in the form of text files with the node, temperature, and time data we need to run the analysis

```
44  #Now, apply the interpolated data as surface and body loads in MAPDL (and store macros in user_files)...
45  bfpath = f"{loadpath}\\mk_time.mac"
46  with open(bfpath,'w') as f:
47      f.write(f"numtimes = {ssol.time_values.size}\n")
48      f.write(f"*dim,ttime,,numtimes\n")
49      for i,time in enumerate(ssol.time_values):
50          f.write(f"ttime({i+1})={time}\n")
51  nodenums = tmesh['ansys_node_num']
52  for tim in range(ssol.n_results):
53      bfname = f"mk_BF{str(tim+1)}.mac"
54      bfpath = f"{loadpath}\\{bfname}"
55      tstr = f"T{str(tim)}"
56      tvals = tmesh[tstr]
57      with open (bfpath,'w') as f:
58          for j in range(nodenums.size):
59              anstr = f"bf,{str(int(nodenums[j]))},temp,{str(tvals[j])}\n"
60              f.write(anstr)
```

Data Mapping with PyAnsys

- Below is a diagram of how it all works
- It's important to understand (and remember) that PyAnsys runs *outside of* Ansys and either connects directly with a database (in the case of MAPDL reader), or establishes a DPF connection with one (as in the case of DPF Post –discussed next)

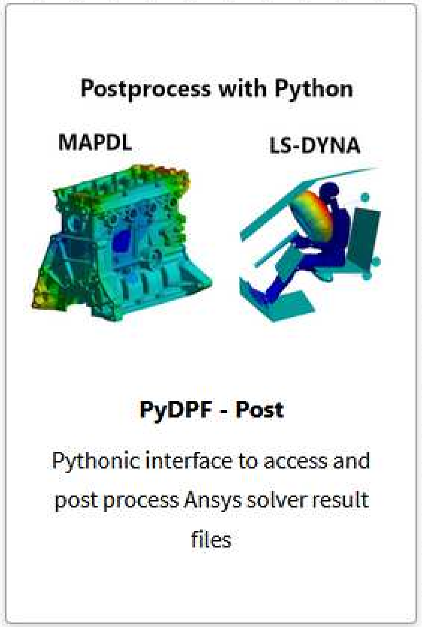


Data Mapping with PyAnsys
-ansys.dpf.post

- In the previous example, we showed users how to map temperature data from a thermal anlaysis system onto a structural model (to be applied as a thermo-elastic load) using a direct connection (MAPDL reader)
- But as we mentioned earler, suppose the temperatures to be mapped come from outside of Ansys?
- PyAnsys is also ideal for such cases, and we supply yet another Python script, called 'tmap2.py' to accomplish this

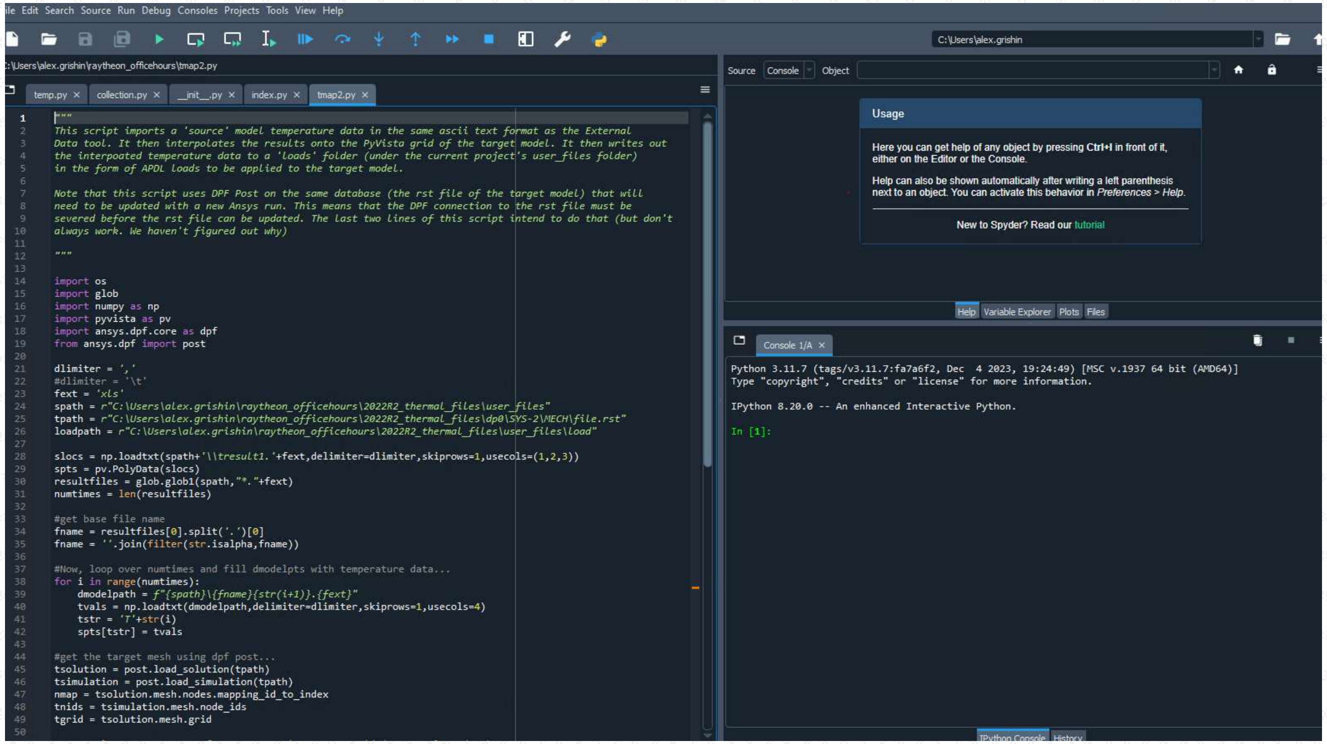
tmap2.py

```python
"""
This script imports a 'source' model temperature data in the same ascii text format as the External
Data tool. It then interpolates the results onto the PyVista grid of the target model. It then writes out
the interpoated temperature data to a 'loads' folder (under the current project's user_files folder)
in the form of APDL loads to be applied to the target model.

Note that this script uses DPF Post on the same database (the rst file of the target model) that will
need to be updated with a new Ansys run. This means that the DPF connection to the rst file must be
severed before the rst file can be updated. The last two lines of this script intend to do that (but don't
always work. We haven't figured out why)

"""

import os
import glob
import numpy as np
import pyvista as pv
import ansys.dpf.core as dpf
from ansys.dpf import post

dlimiter = ','
#dlimiter = '\t'
fext = 'xls'
spath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\user_files"
tpath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\dp0\SYS-2\MECH\file.rst"
loadpath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\user_files\load"
```



Data Mapping with PyAnsys
-ansys.dpf.post

- Of course, ‘tmap2.py’ can be run from a python shell as before(see slide 28)
- This code is very similar to that of ‘tmap.py’ (utilizing the same interpolation method), but this time, it uses DPF Post to get the target database and generates it’s own PyVista source object
- Open a new Spyder session and open file ‘tmap2.py’

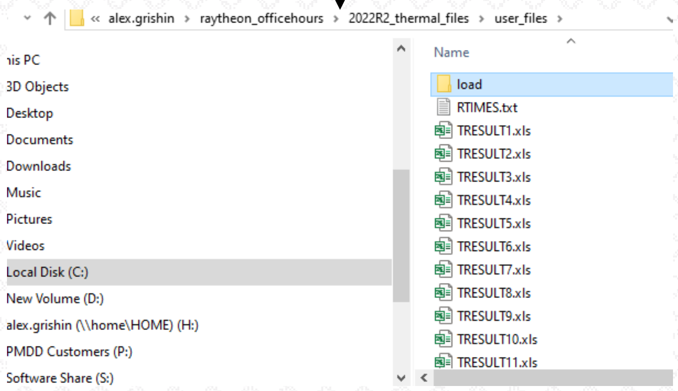


- In this example, we’re using the comma-delimited files generated by the APDL scripts (slides 15 – 17)
- To change that, just uncomment line 22

Data Mapping with PyAnsys
-ansys.dpf.post

- We see the first major differences between this code and ‘tmap.py’ in lines 28 – 42.
- Notice also that the source path (spath) now points to the user_files folder. This code requires that folder be populated with temperature data as described in the first half of this article

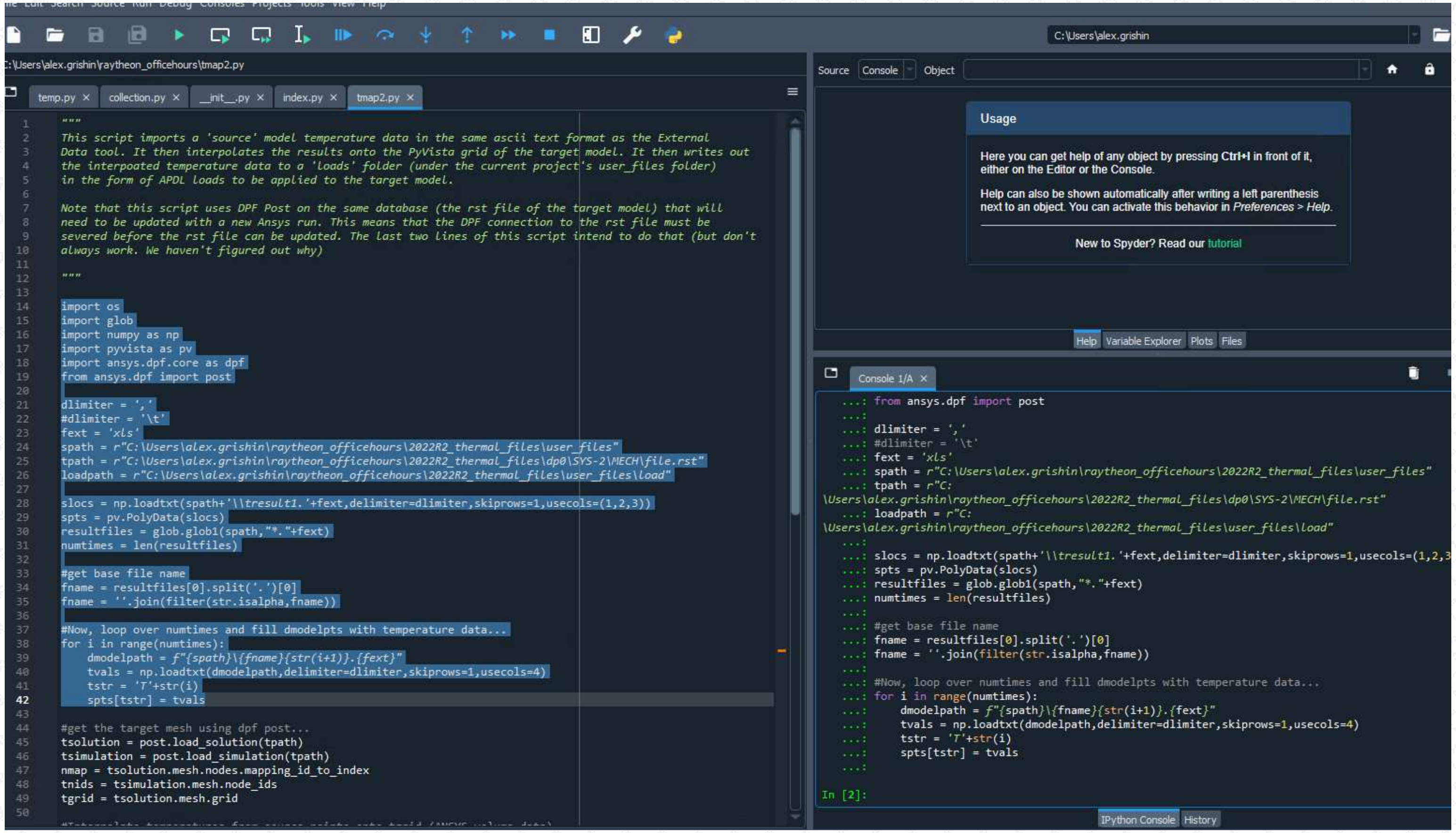
```
24      spath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\user_files"
25      tpath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\dp0\SYS-2\MECH\file.rst"
26      loadpath = r"C:\Users\alex.grishin\raytheon_officehours\2022R2_thermal_files\user_files\load"
27
28      slocs = np.loadtxt(spath+'\\tresult1.'+fext,delimiter=dlimiter,skiprows=1,usecols=(1,2,3))
29      spts = pv.PolyData(slocs)
30      resultfiles = glob.glob1(spath,"*."+fext)
31      numtimes = len(resultfiles)
32
33      #get base file name
34      fname = resultfiles[0].split('.')[0]
35      fname = ''.join(filter(str.isalpha,fname))
36
37      #Now, loop over numtimes and fill dmodelpts with temperature data...
38      for i in range(numtimes):
39          dmodelpath = f"{spath}\{fname}{str(i+1)}.{fext}"
40          tvals = np.loadtxt(dmodelpath,delimiter=dlimiter,skiprows=1,usecols=4)
41          tstr = 'T'+str(i)
42          spts[tstr] = tvals
```



- Line 28 gets the node data one time (so that it doesn’t have to keep reading that for every time step
- Line 29 generates a PyVista PolyData object out of the source nodes (instead of a full mesh)

Data Mapping with PyAnsys -ansys.dpf.post

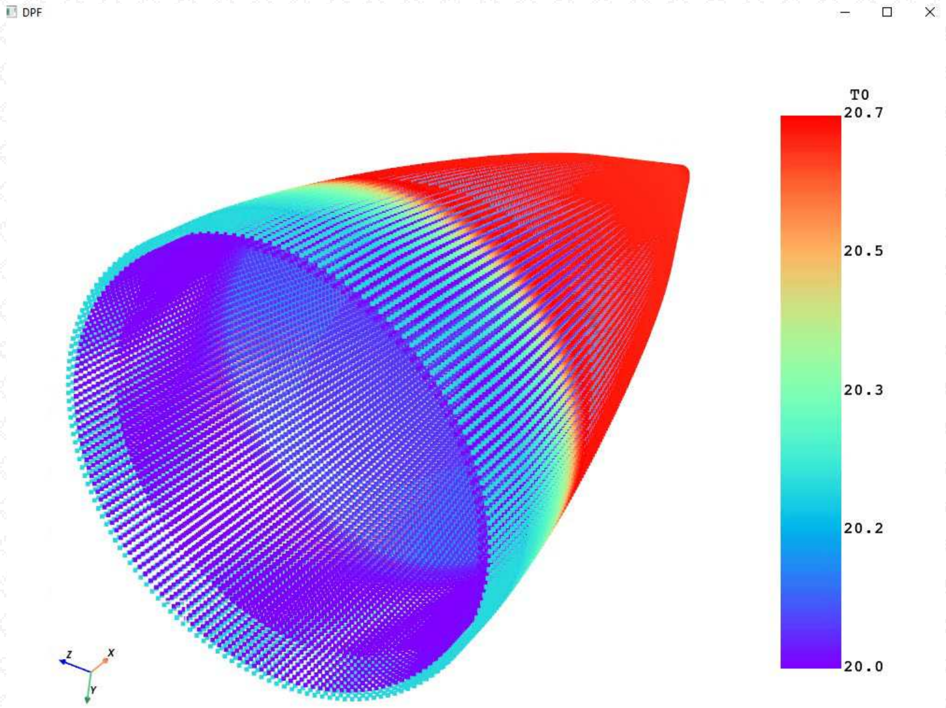
- Cut-and-past lines 14 thru 42 as below



Data Mapping with PyAnsys -ansys.dpf.post

- Now, in the console, type ‘spts.plot(scalars=‘T0’,cmap=‘rainbow’)<enter>’ to plot the first load step

```
In [2]: spts.plot(scalars='T0',cmap='rainbow')
```



- This is the temperature data (just the point data) at the first load step
- Note that the point data is all we really need for nodal interpolation (we don’t need an entire mesh)

Data Mapping with PyAnsys
-ansys.dpf.post

- Lines 45 – 52 are similar to what we've seen before, except that the dpf post interface differs from that of the MAPDL Reader slightly
- for documentation on this interface, see here
- Users may continue cutting and pasting if desired
- One big difference between DPF Post and MAPDL reader is that lines 45 and 46 establish a DPF connection to the target rst
- If we want to run this analysis again, we'll have to break the DPF connection, or else the existing rst file will be *locked* (preventing the creation of a new one!)
- Lines 86 and 87 have been added to do this

```
44  #get the target mesh using dpf post...
45  tsolution = post.load_solution(tpath)
46  tsimulation = post.load_simulation(tpath)
47  nmap = tsolution.mesh.nodes.mapping_id_to_index
48  tnids = tsimulation.mesh.node_ids
49  tgrid = tsolution.mesh.grid
50
51  #Interpolate temperatures from source points onto tgrid (ANSYS volume data)
52  tmesh = tgrid.interpolate(spts,sharpness=2,radius=1.e-3,strategy='closest_point')
53
54  tmesh.plot(scalars='T47',cmap='rainbow')
```

- The same interpolation function is being used

```
84  #shut down grpc server (if you don't do this, the result file won't get updated)
85  #The dpf server can always be restarted wtih dpf.server.connect_to_server()
86  tsimulation.release_streams()
87  dpf.server.shutdown_global_server()
```

Appendix

Installing PyAnsys

Appendix

Installing PyAnsys

Option 1

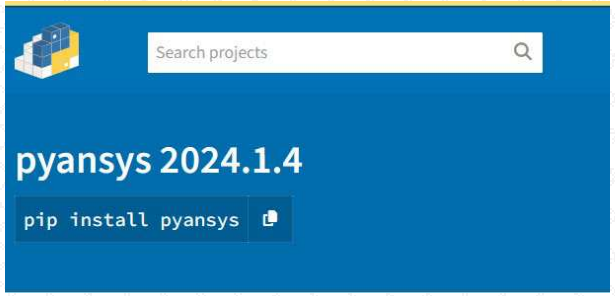
Only PyAnsys, Python, and the Spyder IDE

Appendix

Installing PyAnsys: Option 1

- This article was written the week of 1/22/2024
- At that time, the latest PyAnsys installation targeted Ansys 2024 R1, and for some reason, this build broke some functionality that should have worked with earlier releases (specifically with DPF Post). What this means is that it's probably not safe to just install PyAnsys without regard to version as shown here ('pip install pyansys')
- We'll have to assume that this will happen from time to time, and so what follows is a 'best practice' for installing PyAnsys on a Windows desktop –by targeting specific versions of PyAnsys

- First, make sure to install a supported Python version (supports all versions between 3.9 and 3.11 inclusive as of this writing)



Appendix

Installing PyAnsys: Option 1

- In this example, we'll install Python 3.11 (64-bit)
- This is easiest with the installer (shown below)

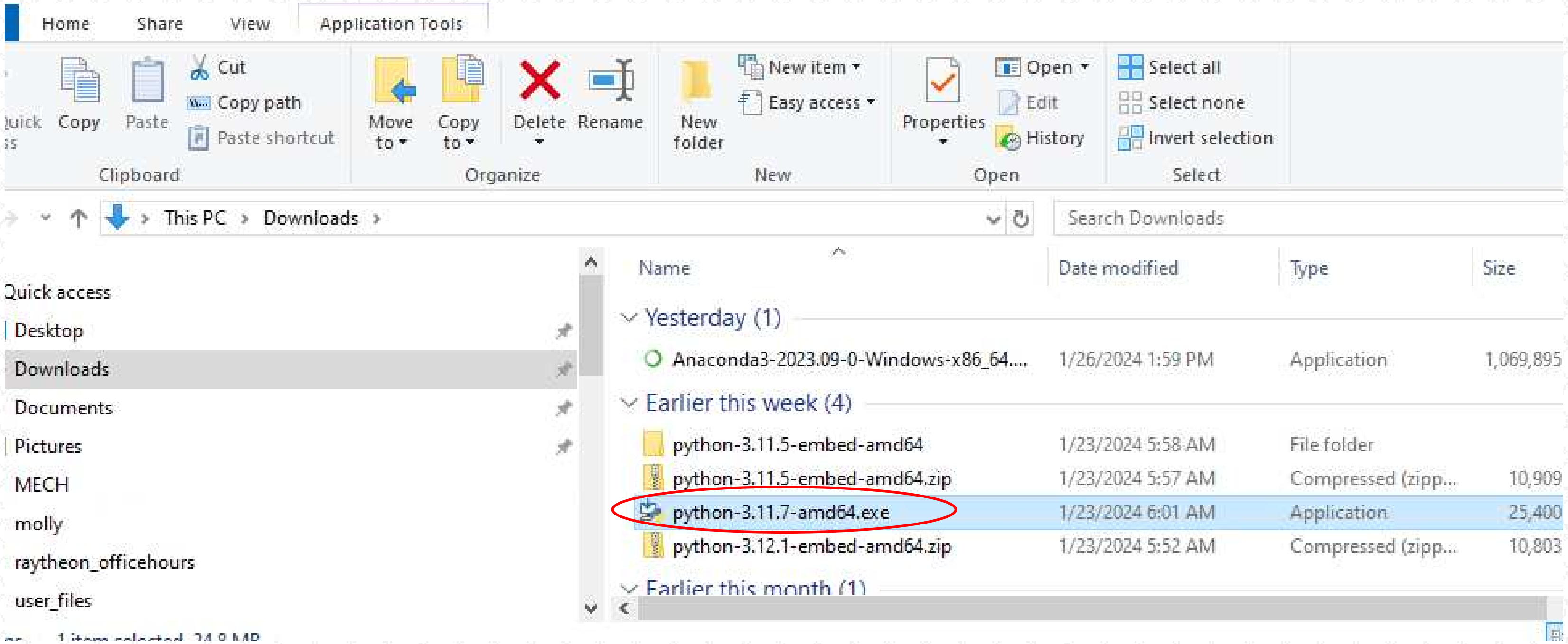
Files

Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore
Gzipped source tarball	Source release		ef61f81ec82c490484219c7f0ec96783	26601929	SIG	.sigstore
XZ compressed source tarball	Source release		d96c7e134c35a8c46236f8a0e566b69c	20074108	SIG	.sigstore
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	89b63192da4def3d0d4f17ff06a33064	44555492	SIG	.sigstore
Windows embeddable package (32-bit)	Windows		f6fa152aa4259f51604f5bbaf5a5f4c4	10075424	SIG	.sigstore
Windows embeddable package (64-bit)	Windows		696ae7fa834526523ba5492d3a1ead14	11198184	SIG	.sigstore
Windows embeddable package (ARM64)	Windows		f3a6296650c51e3e64ae7d41999b4a78	10461852	SIG	.sigstore
Windows installer (32 -bit)	Windows		8a52f3859989f0b1313f4baaa6936410	24722192	SIG	.sigstore
Windows installer (64-bit)	Windows	Recommended	6ebd889155ac3261308202b29d39c5a4	26009544	SIG	.sigstore
Windows installer (ARM64)	Windows	Experimental	216803e75bf3944c183873adf135c459	25272216	SIG	.sigstore

Appendix

Installing PyAnsys: Option 1

- Once downloaded, double-click on the installer to run it with ordinary user priveleges (executing it 'As Adminstrator' won't bring any advantages, and in fact it may be safer to execute it without those privileges)



Appendix

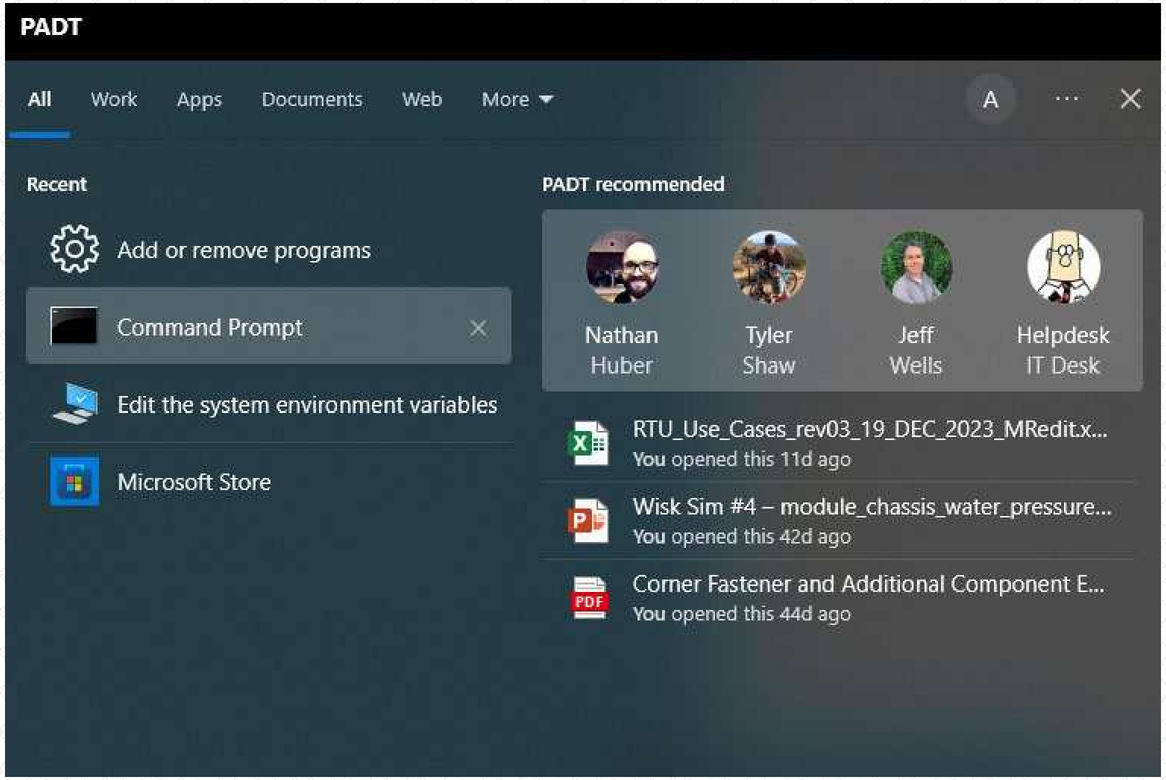
Installing PyAnsys: Option 1

- If you have no other versions of Python installed (or if this is to be the 'main' one used), check both boxes below and hit 'Install Now'

Appendix

Installing PyAnsys: Option 1

- Open a DOS Window (by typing ’cmd’ in Start Menu search Window)



Appendix

Installing PyAnsys: Option 1

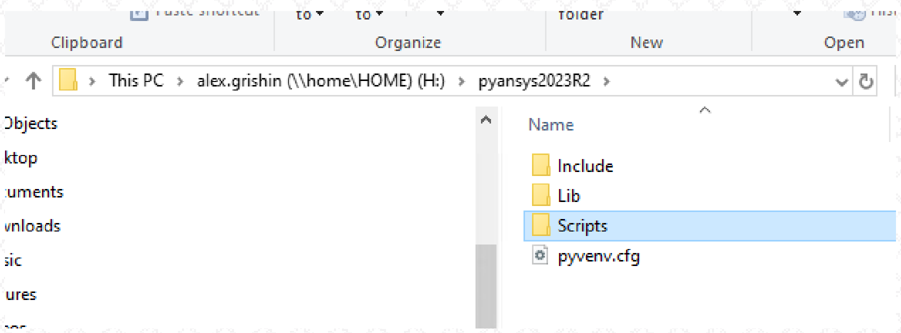
- Make a Virtual Environment for this PyAnsys installation (targeted at Ansys2023R2) by entering the following (followed by <enter>):

  `python –m venv pyansys2023R2`

```
H:\>python -m venv pyansys2023R2
Actual environment location may have moved due to redirects, links or junctions.
  Requested location: "H:\pyansys2023R2\Scripts\python.exe"
  Actual location:    "\\home\HOME\alex.grishin\pyansys2023R2\Scripts\python.exe"

H:\>
```

- By default, Python will place the virtual environment in the current directory location (in this case, your 'home' drive as shown below). You can change this by supplying a path as shown here

- To learn more about wha ta virtual environement is and why we're doing this, see here



Appendix

Installing PyAnsys: Option 1

- Now, activate this environment to begin installing PyAnsys by typing and entering:

```
pyansys2023R2\Scripts\activate
```

```
H:\>pyansys2023R2\Scripts\activate

(pyansys2023R2) H:\>_
```

- Install PyAnsys for your version of Ansys as shown here by typing:

```
pip install pyansys==2023.2.0
```

```
(pyansys2023R2) H:\>pip install pyansys==2023.2.0
Collecting pyansys==2023.2.0
  Obtaining dependency information for pyansys==202
```

- For future installs, you can obtain a list of available PyAnsys versions by typing:

```
pip index versions pyansys
```

```
(pyansys2023R2) H:\>pip index versions pyansys
WARNING: pip index is currently an experimental command. It may be removed/changed in a future release without prior war
ning.
pyansys (2024.1.4)
Available versions: 2024.1.4, 2024.1.3, 2024.1.2, 2024.1.1, 2024.1.0, 2023.2.11, 2023.2.10, 2023.2.9, 2023.2.8, 2023.2.7
, 2023.2.6, 2023.2.5, 2023.2.4, 2023.2.3, 2023.2.2, 2023.2.1, 2023.2.0
  INSTALLED: 2023.2.0
  LATEST:    2024.1.4
```

Appendix

Installing PyAnsys: Option 1

- Follow the suggestion provided by pip at the end of the install:

```
[notice] A new release of pip is available: 23.2.1 -> 23.3.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

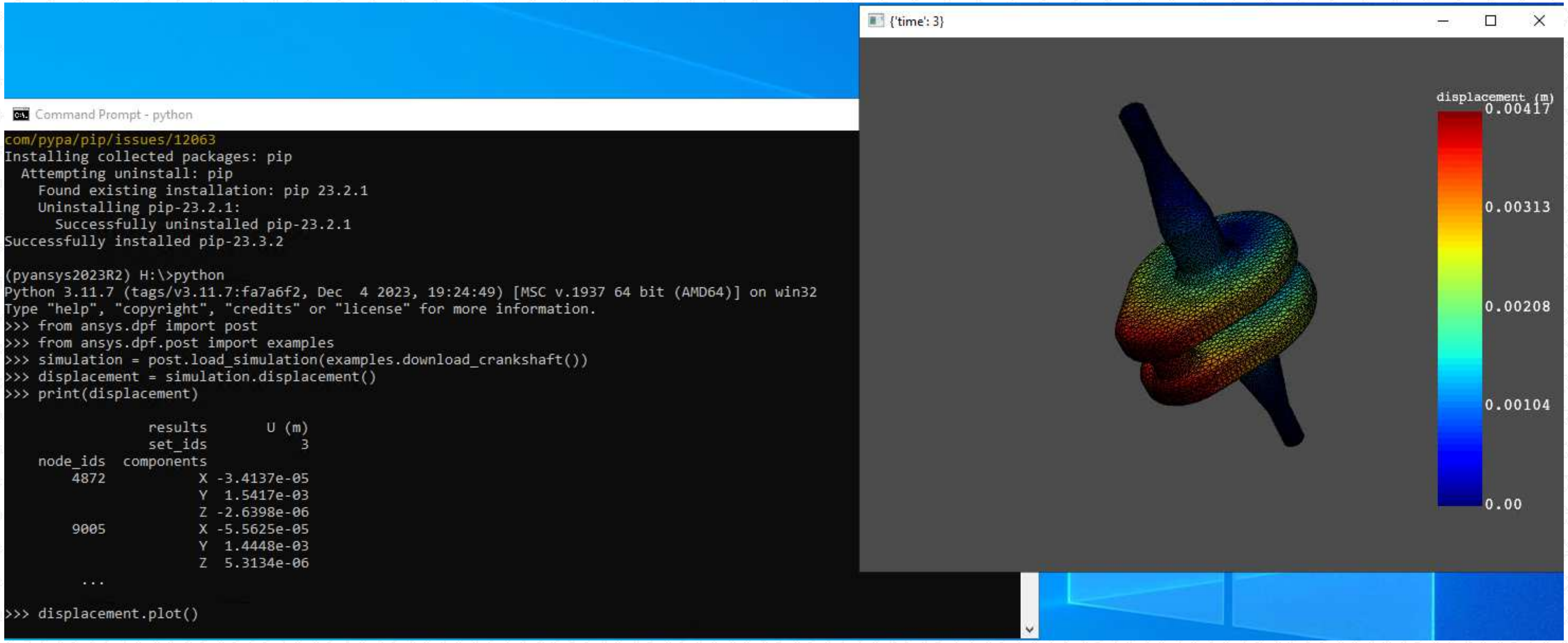
```
(pyansys2023R2) H:\>python -m pip install --upgrade pip
Requirement already satisfied: pip in h:\pyansys2023r2\lib\site-packages (23.2.1)
Collecting pip
  Obtaining dependency information for pip from https://files.pythonhosted.org/packages/15/aa/3f4c7bcee2057a76562a5b33ec
bd199be08cdb4443a02e26bd2c3cf6fc39/pip-23.3.2-py3-none-any.whl.metadata
  Using cached pip-23.3.2-py3-none-any.whl.metadata (3.5 kB)
Using cached pip-23.3.2-py3-none-any.whl (2.1 MB)
DEPRECATION: ansys-dpf-core 0.8.0 has a non-standard dependency specifier ansys-dpf-gate>=0.3.*. pip 23.3 will enforce t
his behaviour change. A possible replacement is to upgrade to a newer version of ansys-dpf-core or contact the author to
 suggest that they release a version with a conforming dependency specifiers. Discussion can be found at https://github.
com/pypa/pip/issues/12063
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.2.1
    Uninstalling pip-23.2.1:
      Successfully uninstalled pip-23.2.1
Successfully installed pip-23.3.2
```

- Now, with this installation of PyAnsys, test it out with at least a few examples from the PyAnsys website.

Appendix

Installing PyAnsys: Option 1

- In particular, make sure to test modules that you know you will use in a Python shell as below
- Testing this base installation in January of 2024 on the example found here results in the following.
- That seems to work...



Appendix

Installing PyAnsys: Option 1

- But the MAPDL Reader seems to be broken when we try the installation on this example (this is why you should test everything you know you're going to use)

```
>>> from ansys.mapdl import reader
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "H:\pyansys2023R2\Lib\site-packages\ansys\mapdl\reader\__init__.py", line 15, in <module>
    from ansys.mapdl.reader.archive import (
  File "H:\pyansys2023R2\Lib\site-packages\ansys\mapdl\reader\archive.py", line 10, in <module>
    from pyvista import CellType
ImportError: cannot import name 'CellType' from 'pyvista' (H:\pyansys2023R2\Lib\site-packages\pyvista\__init__.py)
```

- This happened in January 2024, but may not happen in future releases. If it does happen, the broken modules and depencies must be repaired (future users may skip the following if nothing's broken)
- If the above error does occur, exit Python and type and enter the following:

```
pip install --upgrade pyvista
pip install --upgrade ansys-dpf-core
pip install --upgrade ansys-mapdl-core
```

Appendix

Installing PyAnsys: Option 1

- After we make the three ‘upgrades’, we get the following message:

```
Requirement already satisfied: certifi>=2017.4.17 in h:\pyansys2023r2\lib\site-packages (from requests<3.0.0.dev0,>=2.18
.0->google-api-core!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.0,<3.0.0.dev0,>=1.31.5->google-api-python-client->ansys-dpf-core) (2023
.11.17)
Using cached ansys_dpf_core-0.10.1-py3-none-win_amd64.whl (6.2 MB)
Installing collected packages: ansys-dpf-core
  Attempting uninstall: ansys-dpf-core
    Found existing installation: ansys-dpf-core 0.8.0
    Uninstalling ansys-dpf-core-0.8.0:
      Successfully uninstalled ansys-dpf-core-0.8.0
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviou
r is the source of the following dependency conflicts.
ansys-dpf-composites 0.2b2 requires pyvista<0.37.0,>=0.36.1, but you have pyvista 0.43.2 which is incompatible.
pyansys 2023.2.0 requires ansys-dpf-core==0.8.0, but you have ansys-dpf-core 0.10.1 which is incompatible.
pyansys 2023.2.0 requires ansys-mapdl-core==0.64.1, but you have ansys-mapdl-core 0.67.0 which is incompatible.
pyansys 2023.2.0 requires ansys-math-core==0.1.1, but you have ansys-math-core 0.1.3 which is incompatible.
Successfully installed ansys-dpf-core-0.10.1
```

- Certainly not what we want to see, but so far, we haven’t noticed anything still broken
- Proceed to install the IDE

Appendix

Installing PyAnsys: Option 1

- We’re going to install the Spyder IDE (editor). We recommend doing this outside of any virtual environment and NOT using pip (we’re going to download the installer). But before we do that, we’ll use pip within the pyansys environment to supply Spyder with the dependencies it will need to run in that environment (by the way: you’ll need to do this with all virtual environments you create in the future)
- Type:

```
pip install spyder-kernels==2.5.*
```

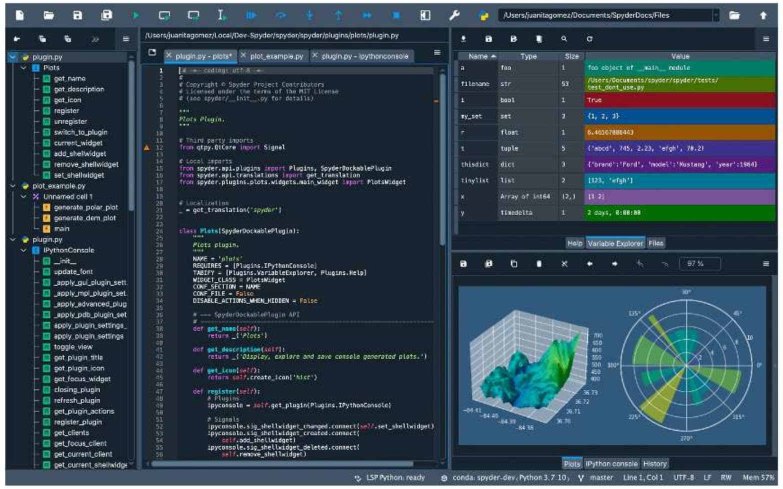
```
(pyansys2023R2) H:\>pip install spyder-kernels==2.5.*
Collecting spyder-kernels==2.5.*
```

Appendix

Installing PyAnsys: Option 1

- Navigate to the [Spyder downloads page] , scroll to the bottom, and hit the 'Download For Windows' button (or click the link below)

spyder | The Scientific Python Development Environment

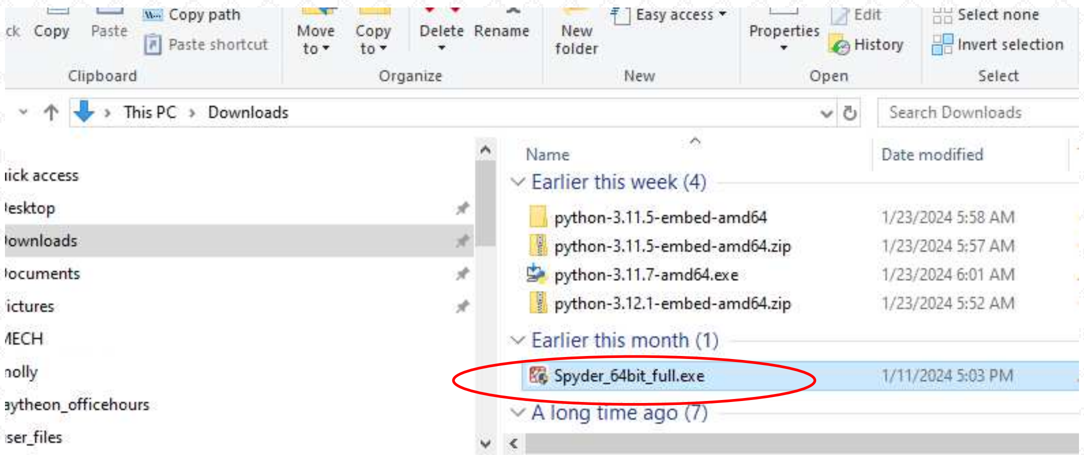


Download for Windows

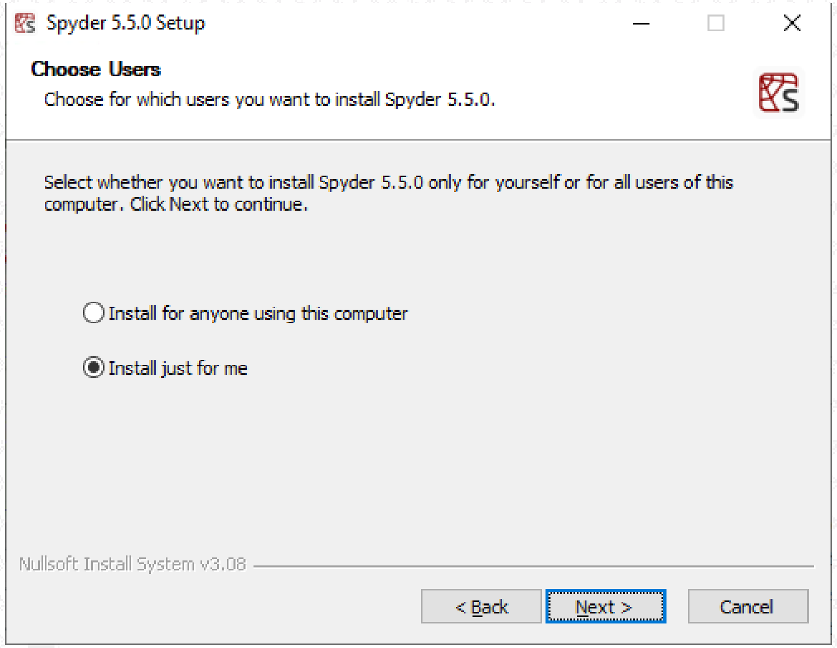
Appendix

Installing PyAnsys: Option 1

- Double-click on the 'Spyder_64bit_full.exe' installer
- Accept all defaults as they pop up (except for this one)...



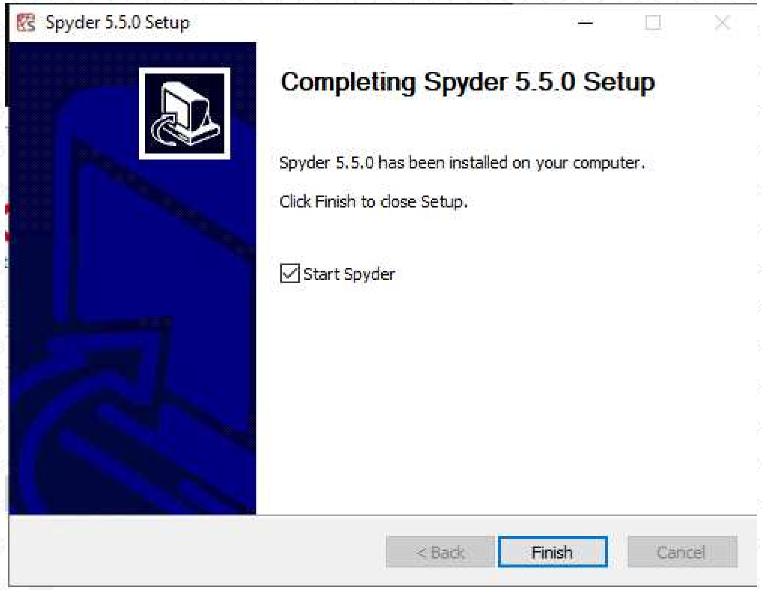
- If you're not installing on a system server (just for your own use on your workstation), select 'Install just for me'



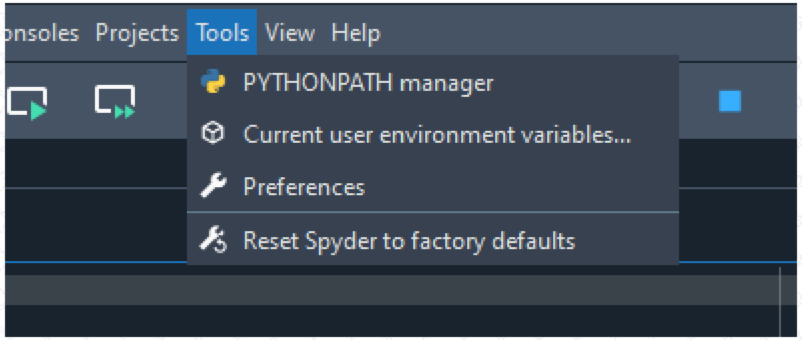
Appendix

Installing PyAnsys: Option 1

- Open Spyder by clicking ‘Finish’ (when finished) with the ‘Start Spyder’ option checked as below



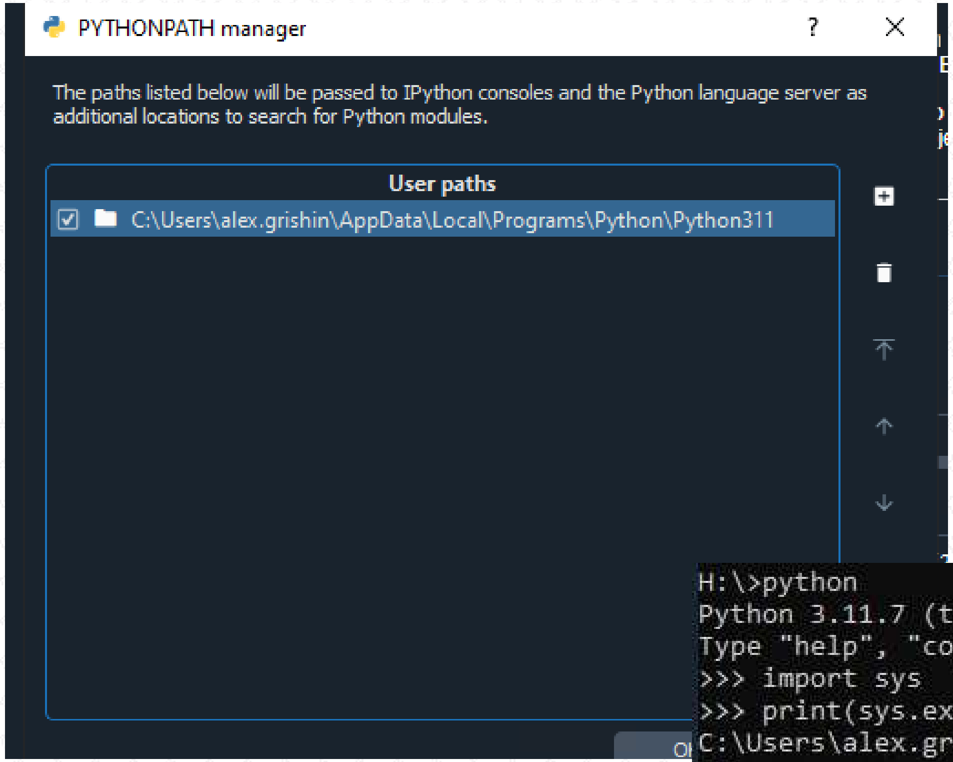
- Spyder installs with its own version of Python pre-installed
- This will NOT be what you want
- Go to Tools->PYTHONPATH manager



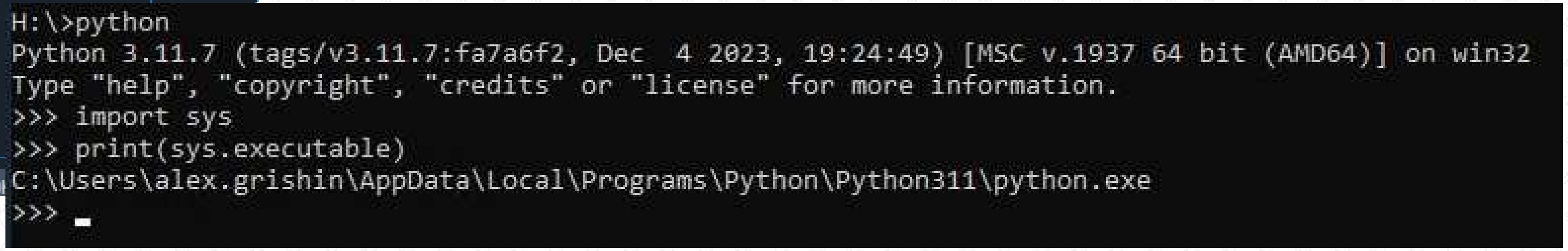
Appendix

Installing PyAnsys: Option 1

- In the PYTHONPATH manager, click on the ‘+’ button on the right to ‘Add path’
- Browse to add the base Python installation path as shown here



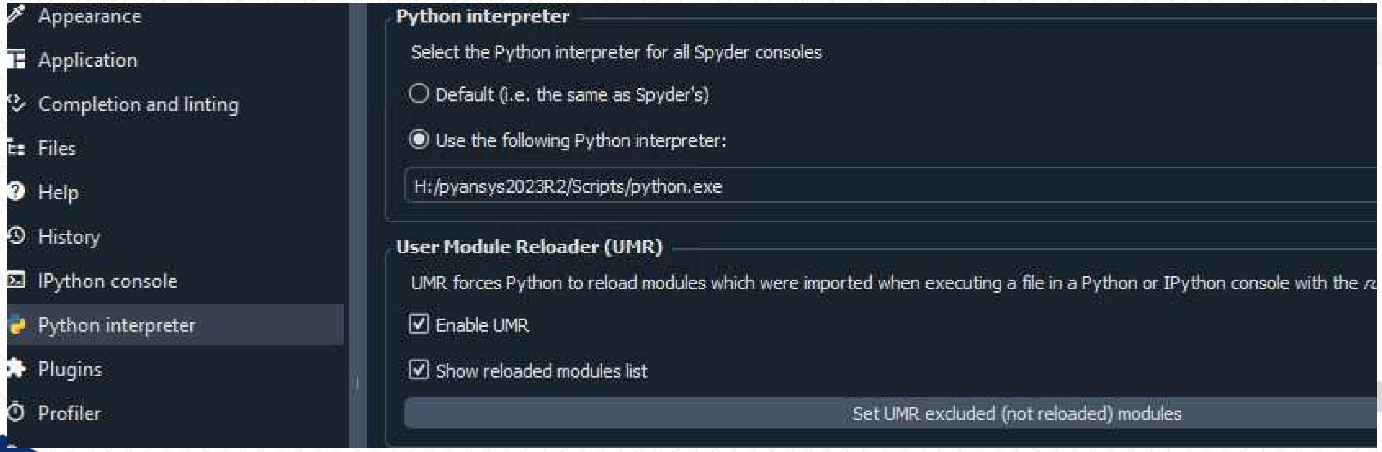
- **Pro Tip**: To discover where your base installation path is, there is a much better way than that shown in the link above
- Just open a command prompt (you don’t want to be in the virtual environment. Deactivate it if you are), and type ‘python’ to launch a python shell session
- Then type ‘import sys’ followed by ‘print(sys.executable)’
- The path PYTHONPATH is looking for is the parent of this executable (the ‘Python311’ folder)

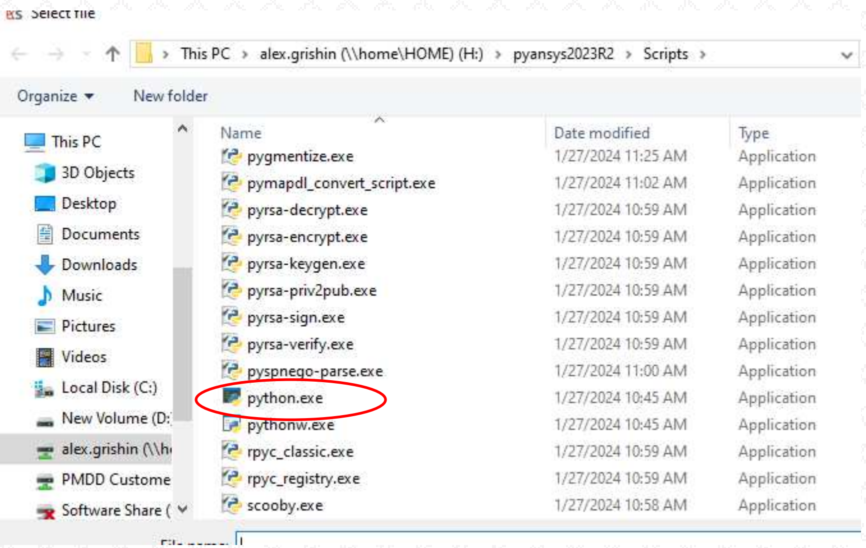


Appendix

Installing PyAnsys: Option 1

- from the Spyder top menu, go to Tools->Preference->Python interpeter and browse to select the python executable path in the virtual environment Scripts folder (this time including the executable itself. You can use the same pro tip as before to find the path, but this time from within the pyansys2023R2 virtual environement)
- To summarize: In the previous step, we told Spyder where the base installation is so it can use that instead of its own Python installation. In this step, we're telling Spyder to use the Python executable in our PyAnsys virtual environment instead of the base installation. This is Spyder's way of supporting virtual environments

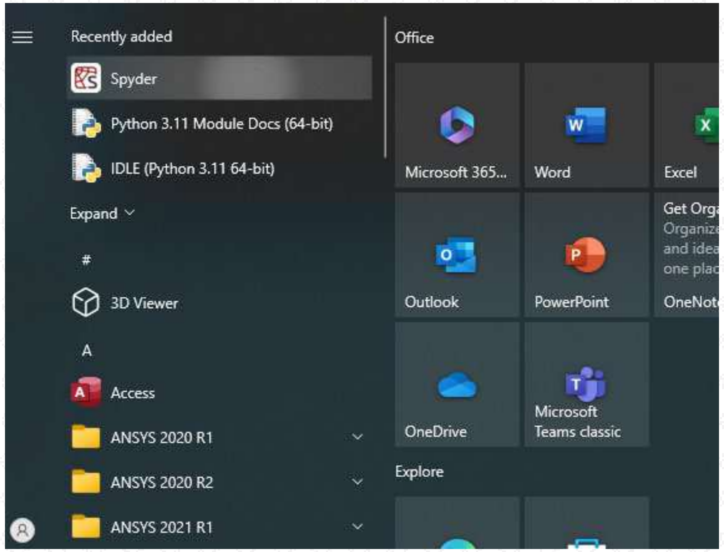


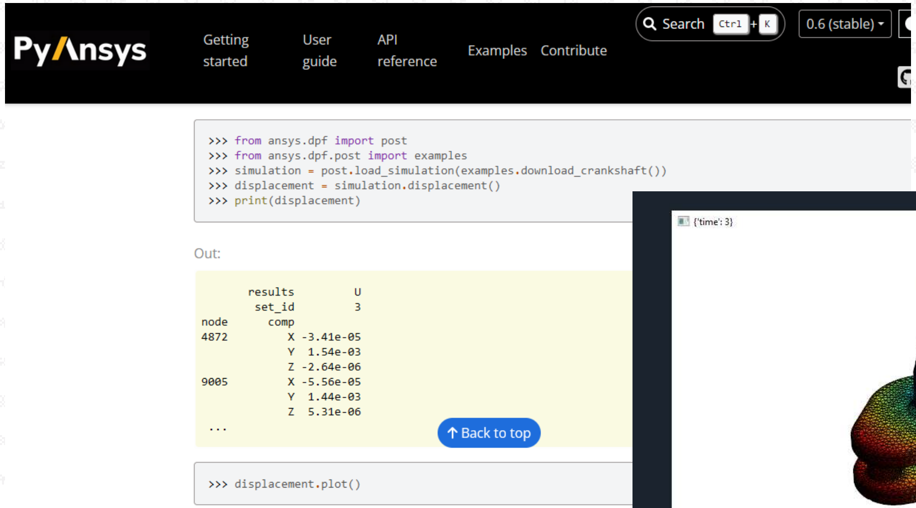


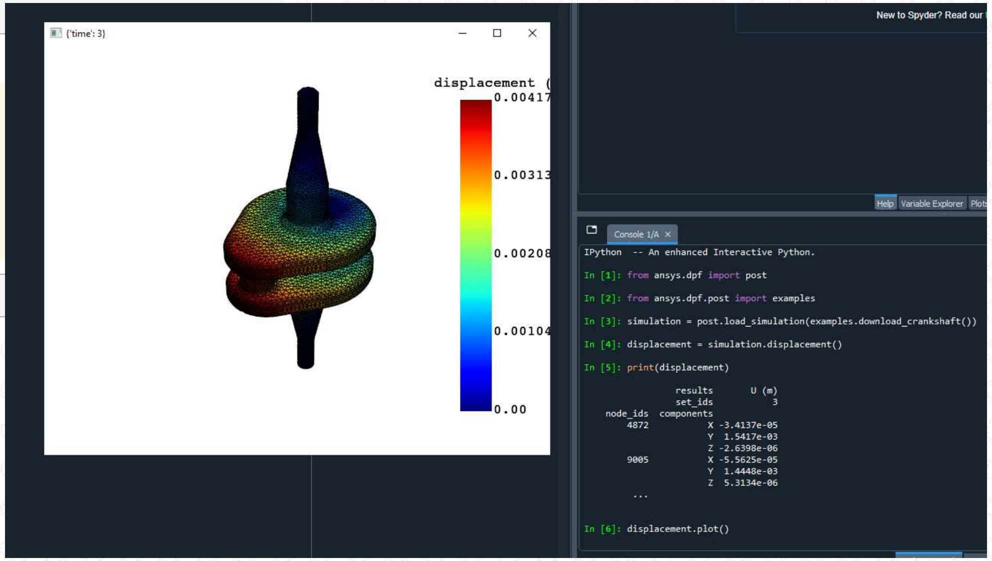
Appendix

Installing PyAnsys: Option 1

- To try the new settings, close the Spyder app and relaunch it from the start menu
- Now, try a simple DPF Post example in the console window to test the installation (for example, the one on slide 62



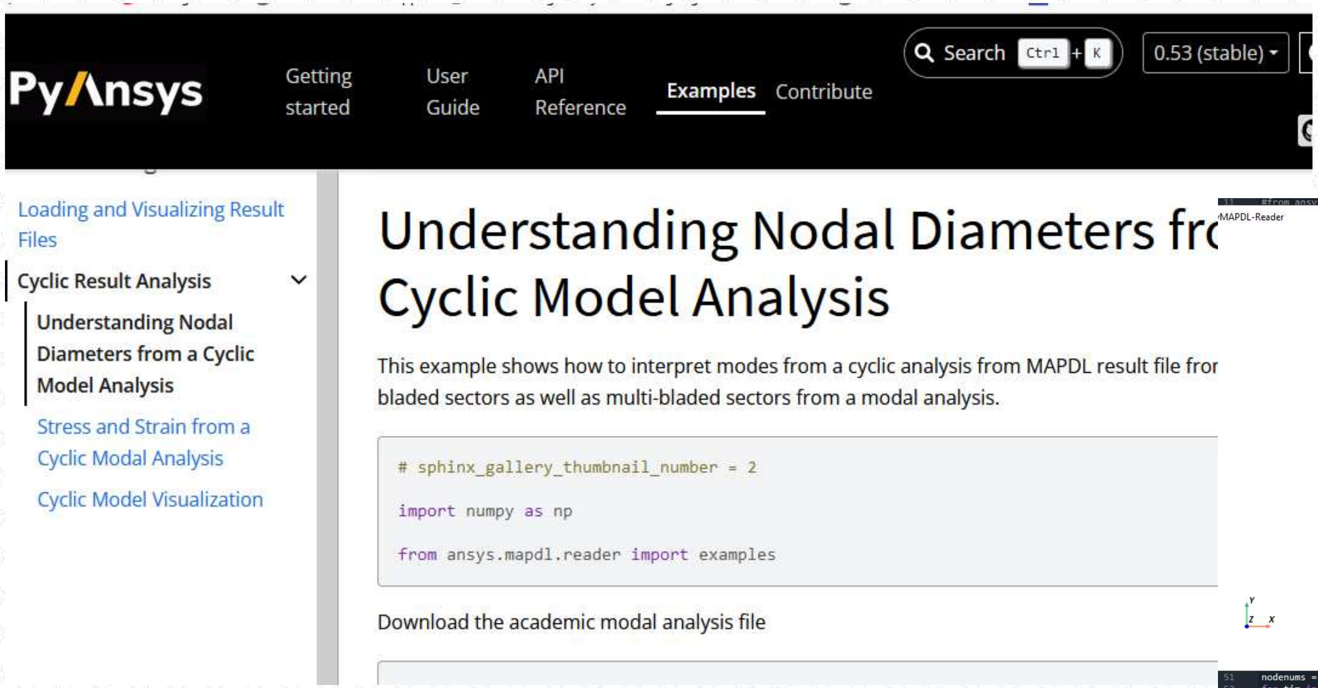


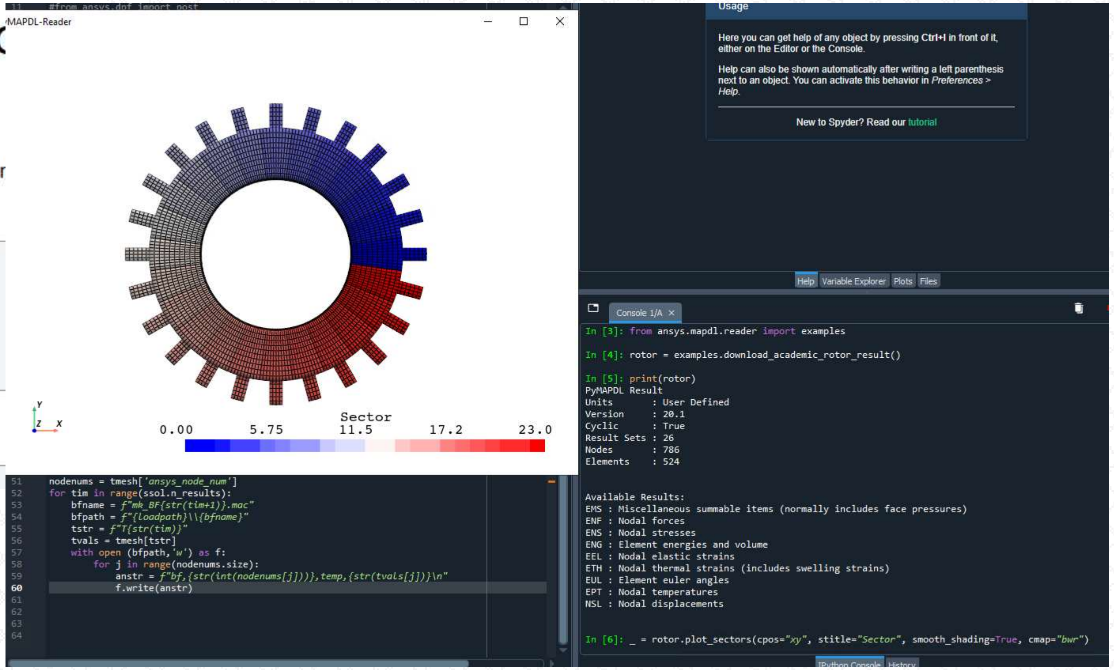


Appendix

Installing PyAnsys: Option 1

- This time, the example from slide 63 (MAPDL Reader) seems to work





Appendix

Installing PyAnsys: Option 1

- This seems like a lot of work, but it’s worth it if everything works.
- Future releases may fix things. If so, then users can omit the steps taken on slide 63 (the ‘upgrade’ of PyVista, DPF Core, and MAPDL Reader), but we recommend all the other steps taken here to ensure a robust functioning installation of PyAnsys

Appendix

Installing PyAnsys

Option 2

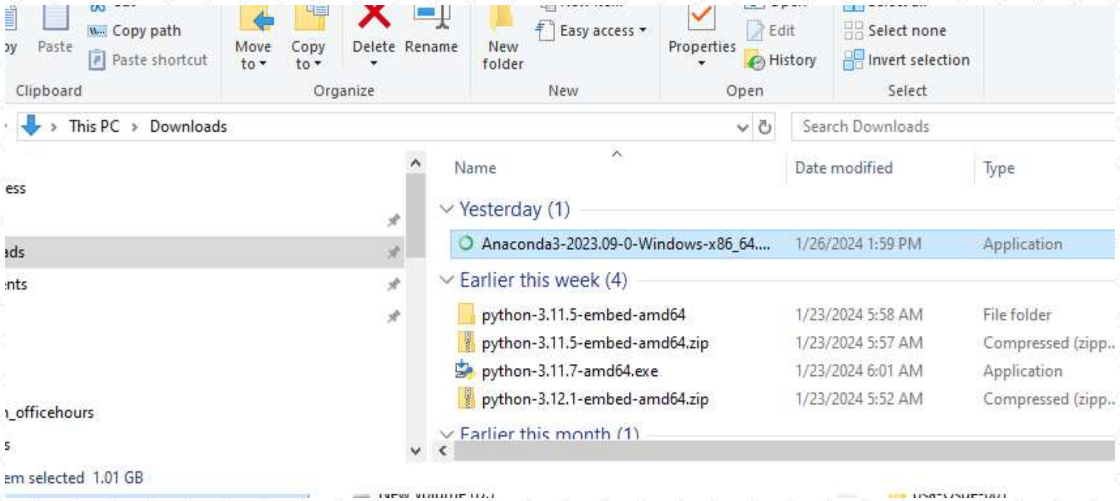
Anaconda

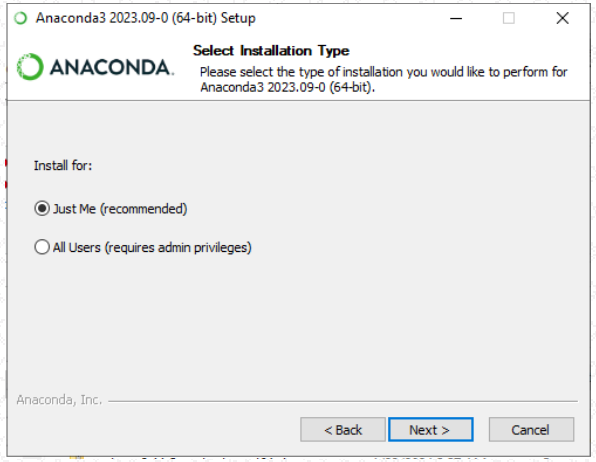
Appendix

Installing PyAnsys: Option 2

- Once again, accept all default options and keep hitting ‘next’

- Download Anaconda from [this website]
- Now, run the Anaconda installer. If you have Adminstrator privileges, run it ‘As Administrator’* (by-right-clcking on it. In that case, you will also want to open all subsequent Anaconda Prompts As Adminstrator). If not, simply double-click to run it.
- If you’re installing it ‘As Administrator’, click on ‘All Users’ in the next dialog box
- If not, select ‘Just Me’
- Accept all other defaults



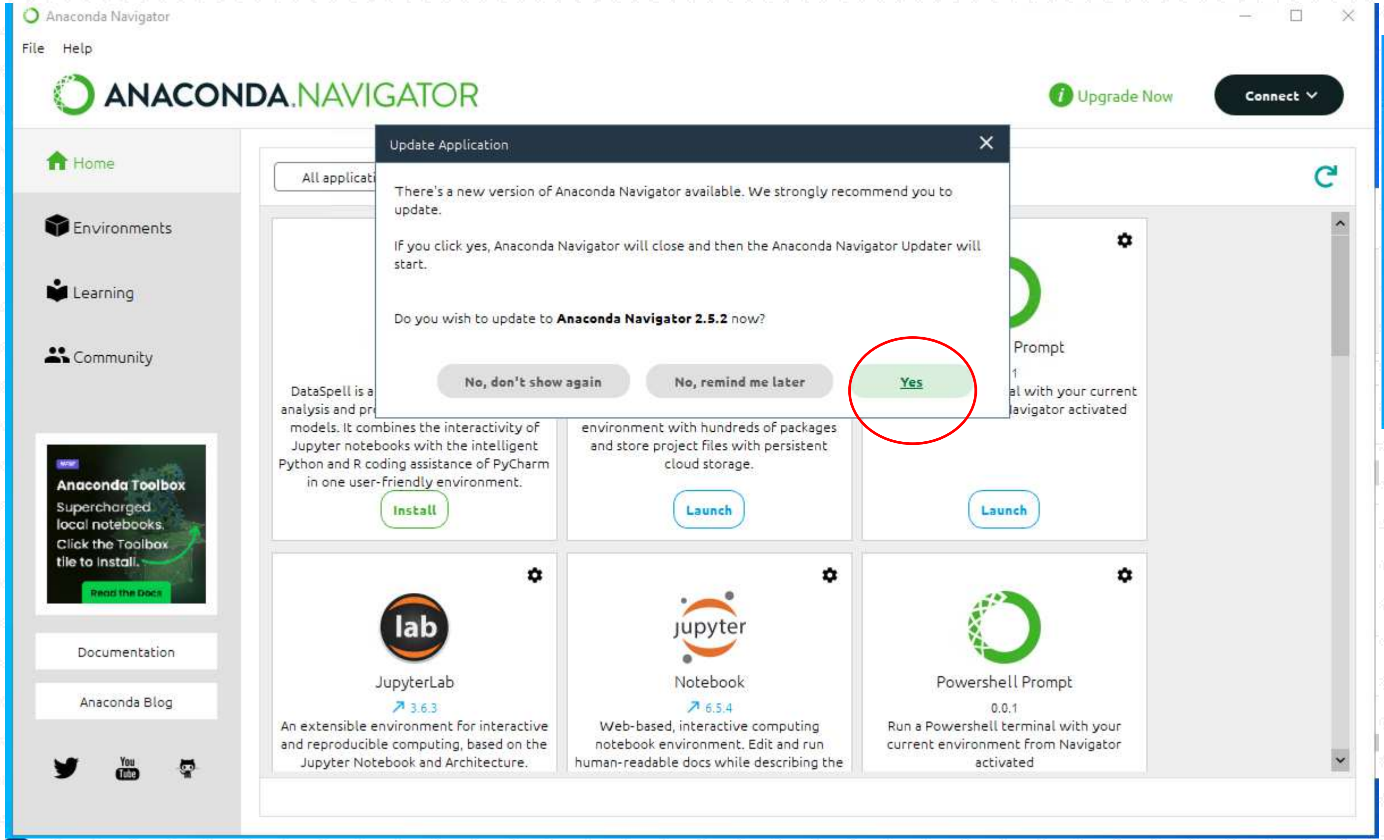


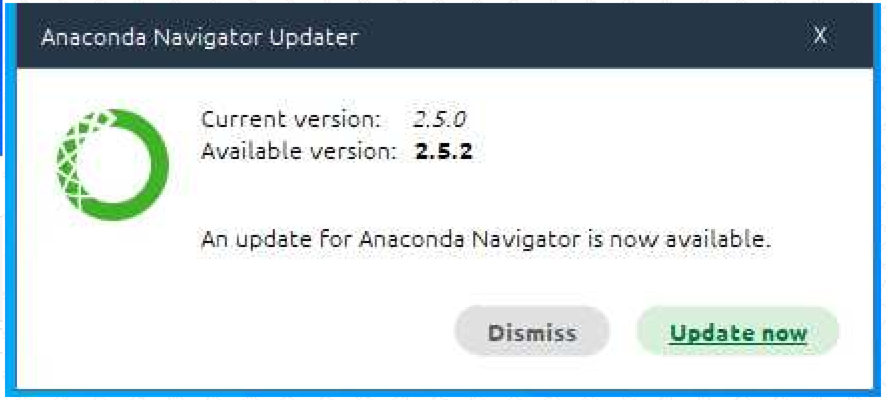
* We recommend NOT installing ‘As Administrator’ if you can avoid it

Appendix

Installing PyAnsys: Option 2

- When the installer finishes, you'll be prompted to 'update'. Click 'Yes' (followed by 'Update Now')



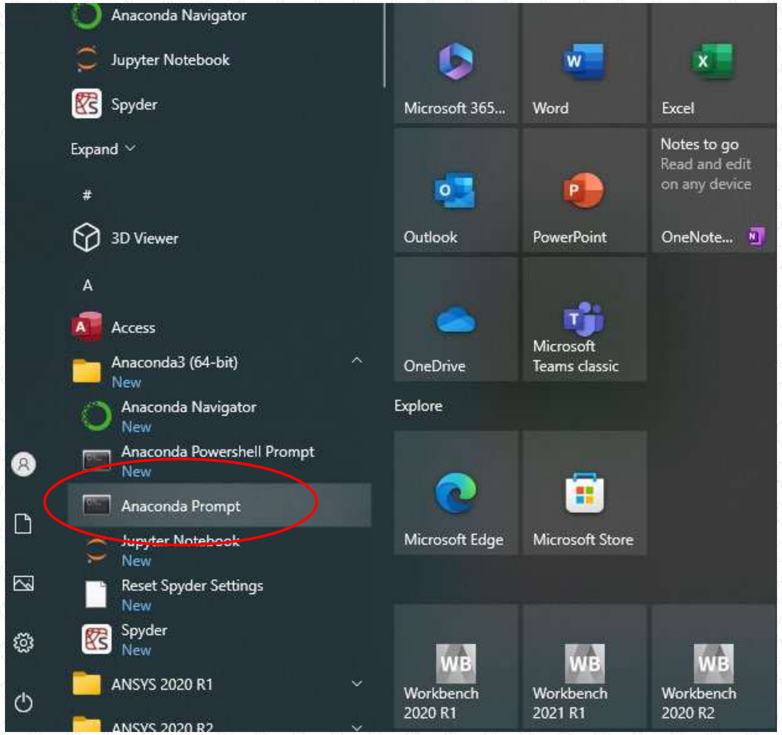


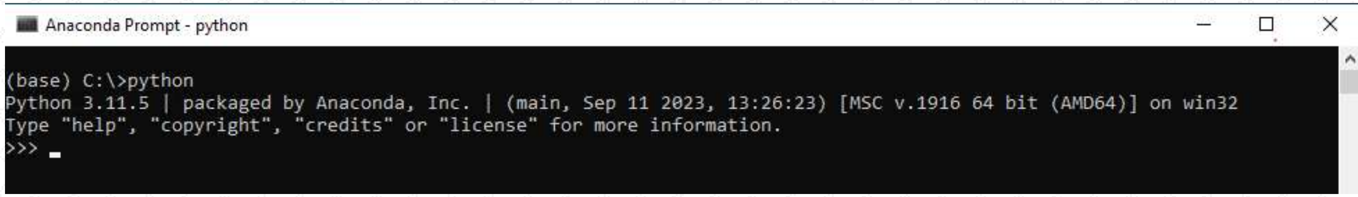
- you will be repeatedly asked if you want to 'sign in' or otherwise register for Anaconda's cloud services
- This is up to you, but not strictly necessary for using Anaconda
- When finished, you will be prompted to launch navigator. Do so

Appendix

Installing PyAnsys: Option 2

- After Anaconda has installed and updated, go to your Wndows Start Menu and select the 'Anaconda Prompt'
- Type 'python' into the prompt window...





- So, in January 2024, it looks like Anaconda is providing us with Python 3.11.5 as the Python version
- This is fine for our current purposes (we may have to amend this for future users)

Appendix

Installing PyAnsys: Option 2

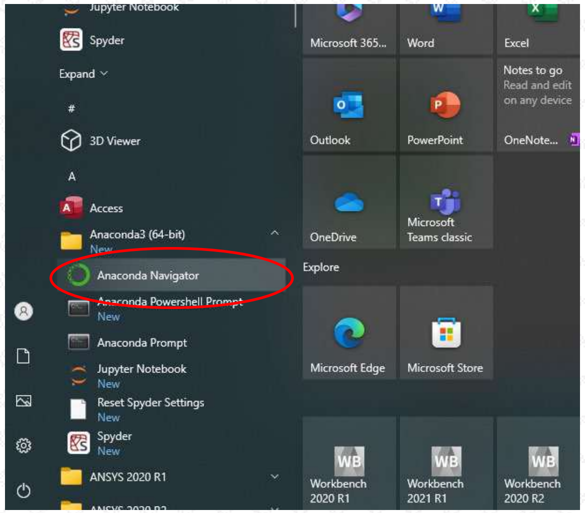
- Notice the ‘(base)’ next to the prompt
- this tells us that we are in the ‘base’ environment
- Unlike a ‘raw’ or independent installation of Python, in which one navigates environments with the base installation Python, Anaconda works with its own external environment and package-management systen (conda)
- Windows does not know where the Python executable is (that location will depend on whether you installed with adminstrator privileges or not). Anaconda manages its own Python ecosystem.
- To see where the base installation is, once again type and enter ‘import sys’ followed by ‘print(sys.executable)’

```
(base) C:\>python
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> print(sys.executable)
C:\Users\alex.grishin\AppData\Local\anaconda3\python.exe
>>> _
```

Appendix

Installing PyAnsys: Option 2

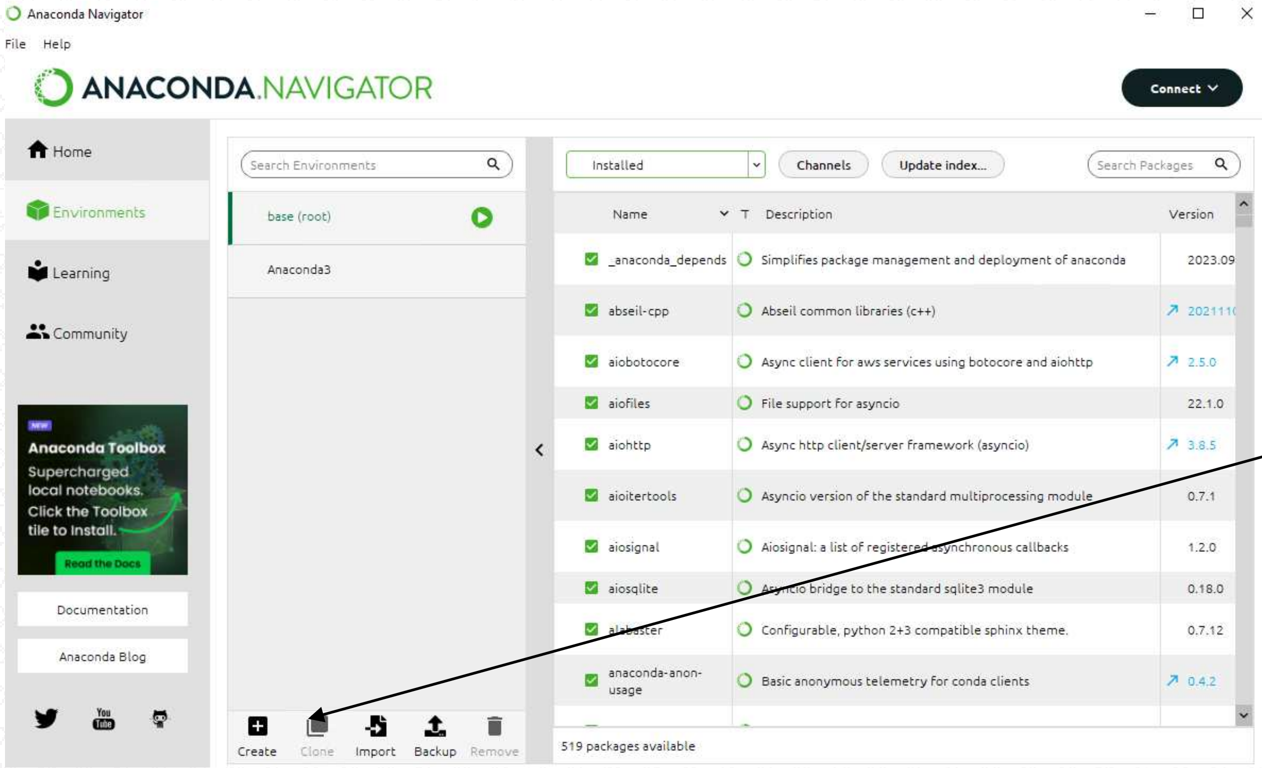
- Ok. Time to install PyAnsys
- Everything will proceed almost exactly as before, except this time we'll create the pyansys2023R2 environment with Conda (Anaconda's package manager)
- This can be done two ways
  1. Graphically, with the Navigator
  2. In the Anaconda Prompt shell
- We'll do it with Navigator (which should already be open. See slide 75)
- If not, launch it from the Start Menu...



Appendix

Installing PyAnsys: Option 2

- Create the environment by clicking on ‘Environments’ in the left tool bar and also click on the ‘base (root)’ environment in the middle menu group
- When you do, all the packages that you install are shown at teh right (there are over 500!)

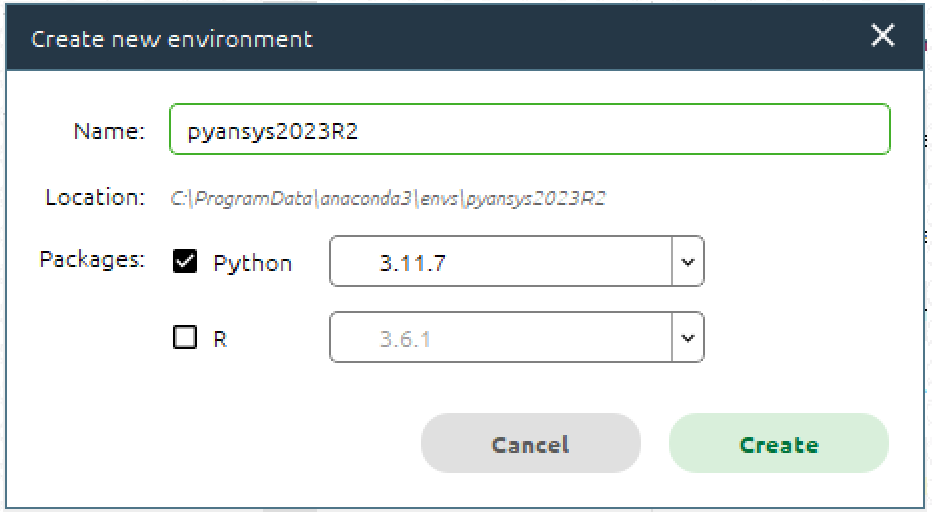


- Hit the ‘+Create’ button at the bottom of the middle screen

Appendix

Installing PyAnsys: Option 2

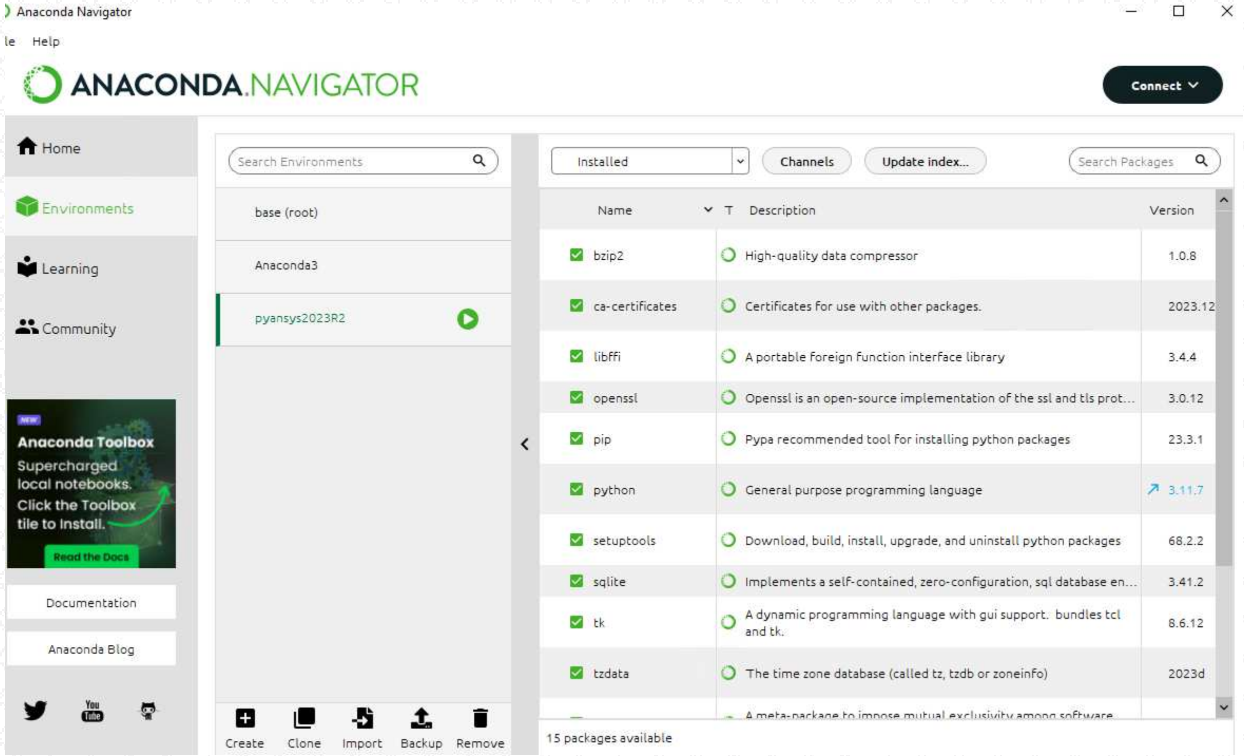
- Fill in the ‘Create new environment’ dialog box as shown below



Appendix

Installing PyAnsys: Option 2

- Once we do this, notice that there are now only 15 packages available in our newly created environment (that’s about to change)
- These correspond to ‘core’ packages that Anaconda deems essential for any environment



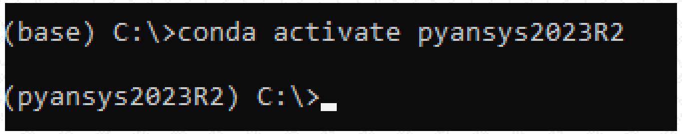
Appendix

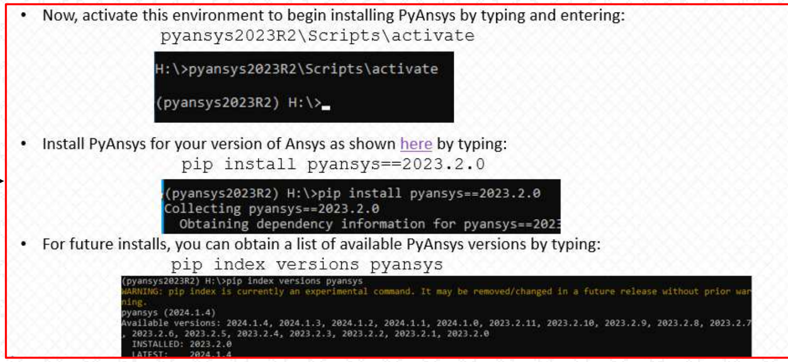
Installing PyAnsys: Option 2

- We'll continue using the Anaconda Prompt. Launch the Anaconda Prompt (close the earlier one if it is still open and launch a new one)
- Type 'conda activate pyansys2023R2 <enter>'
- Now type 'pip install pyansys==2023.2.0 <enter>' as on slide 59
- From there, follow the same directions as found on slides 59 – 64
- But change the spyder-kernels version to 2.4.* (this release of Anaconda uses an earlier releae of Spyder). So change the line on slide 64 to:

```
pip install spyder-kernels==2.4.*
```

- In particular, don't forget to perform the examples check we did on slides 61,62. And if nothing is broken, you don't have to perform the upgrades on slide 62. But as of this writing, users should see the same Error as shown on slide 62, and so the same remedy as shown there must applied here
- Note that conda has its own package installer (conda), which works with syntax similar to pip and should normally be used to install packages
- However, not all Python packages support conda. PyAnsys is one of these, and so pip *must* be used in those cases

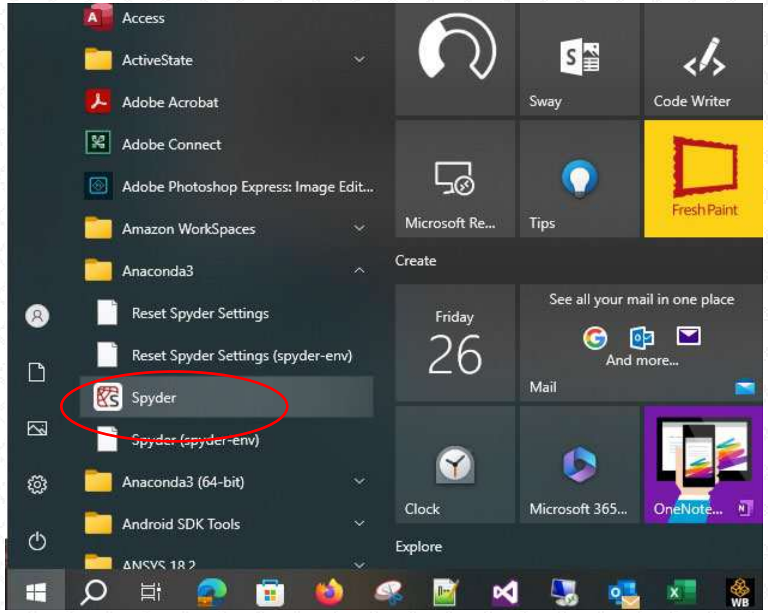


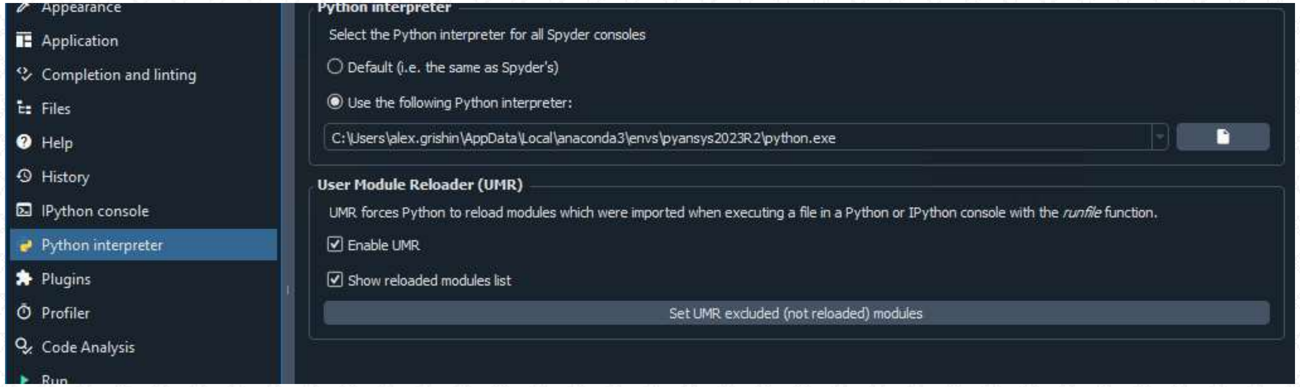


Appendix

Installing PyAnsys: Option 2

- Launch spyder, but this time do it with the Spyder launch icon that we have in the Start Menu (this is another advantage to using Anaconda)
- The next step is to set the Python solver to use the PyAnsys environement we've created. This is under Tools->Preferences as on slide 69 (we don't have to set the PYTHONPATH in Anaconda, so we can skip the instructions on slide 68)

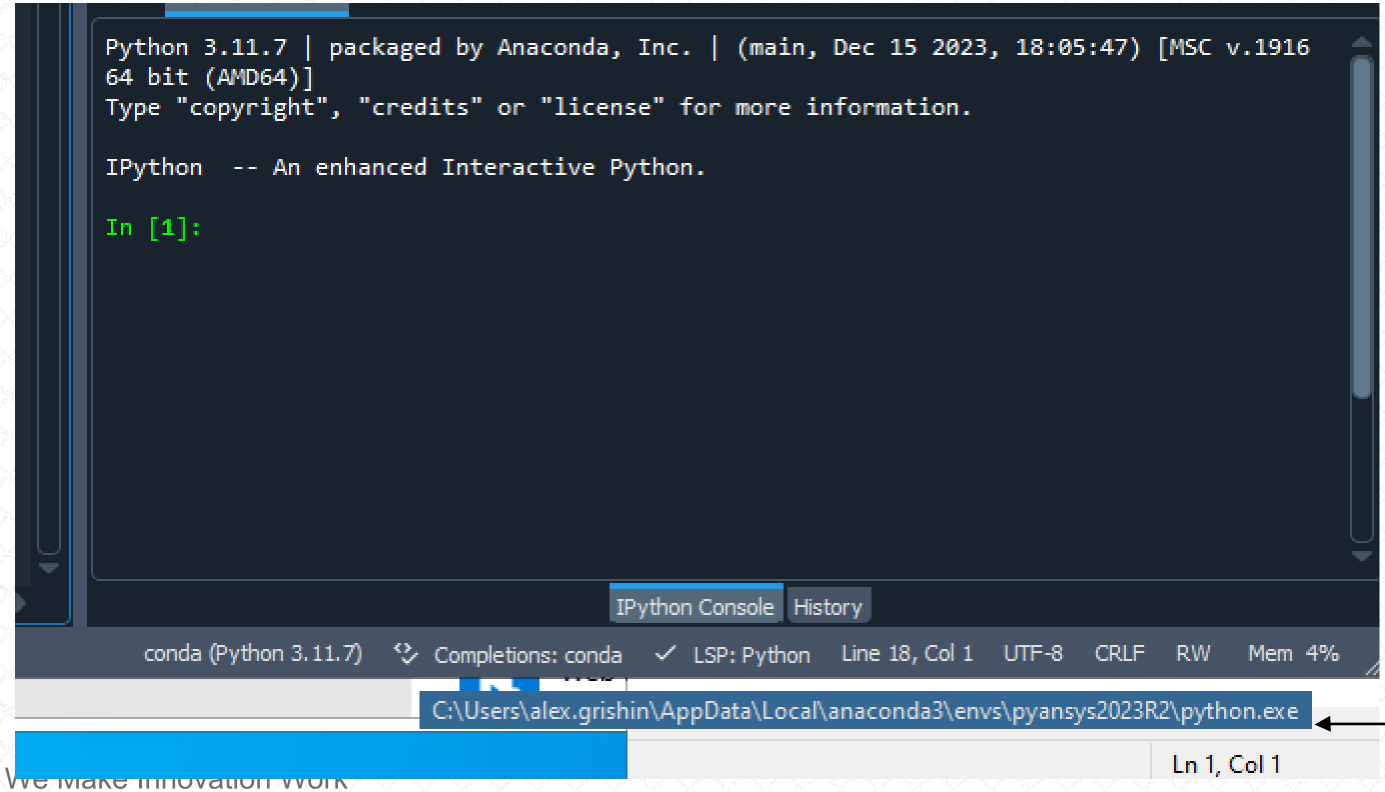




Appendix

Installing PyAnsys: Option 2

- Once the Python Interpreter has been set to the PyAnsys environment, close Spyder
- Relaunch Spyder to ensure the change have taken effect.
- You'll know you're successful when you see the following in your console window



- Python version is now 3.11.7

- Hovering cursor over this field at the bottom confirms that the PyAnssy environment is bing used

Appendix

Installing PyAnsys: Option 2

- Test the install with the PyAnsys with the examples on slide 62 and slide 63