



Powering Innovation That Drives Human Advancement

PyAnsys Workshop for Structural Engineers

Sandeep Medikonda (Senior Manager Application Engineering)
Mohamed Koubaa (Principal R&D Engineer – MBU)

Agenda

- Scripting Overview
- PyAnsys Overview & What's available
- Ansys Python Manager: Demo
- Mechanical Scripting & PyMechanical
 - Hands on Demo
 - Workshop
- Intro to PyDPF
 - Hands on Demo
 - Workshop
- Documentation, Help & Other Resources
- Conclusions



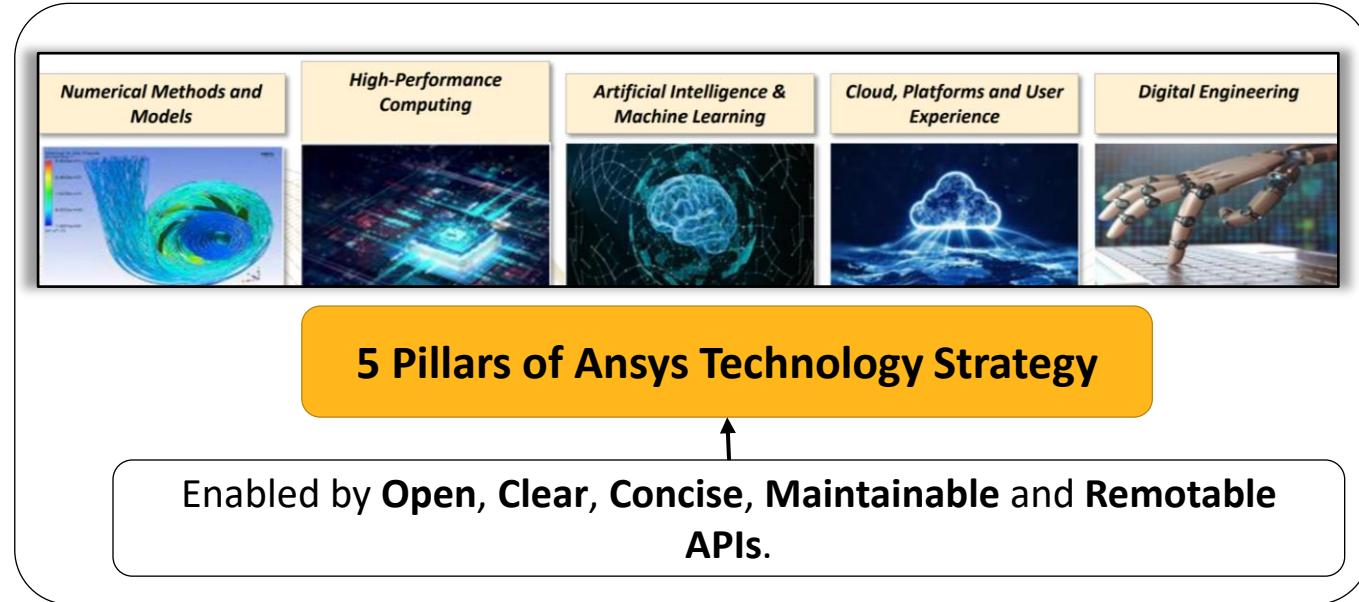
Need for APIs?

Evolving Ansys Customer Needs:

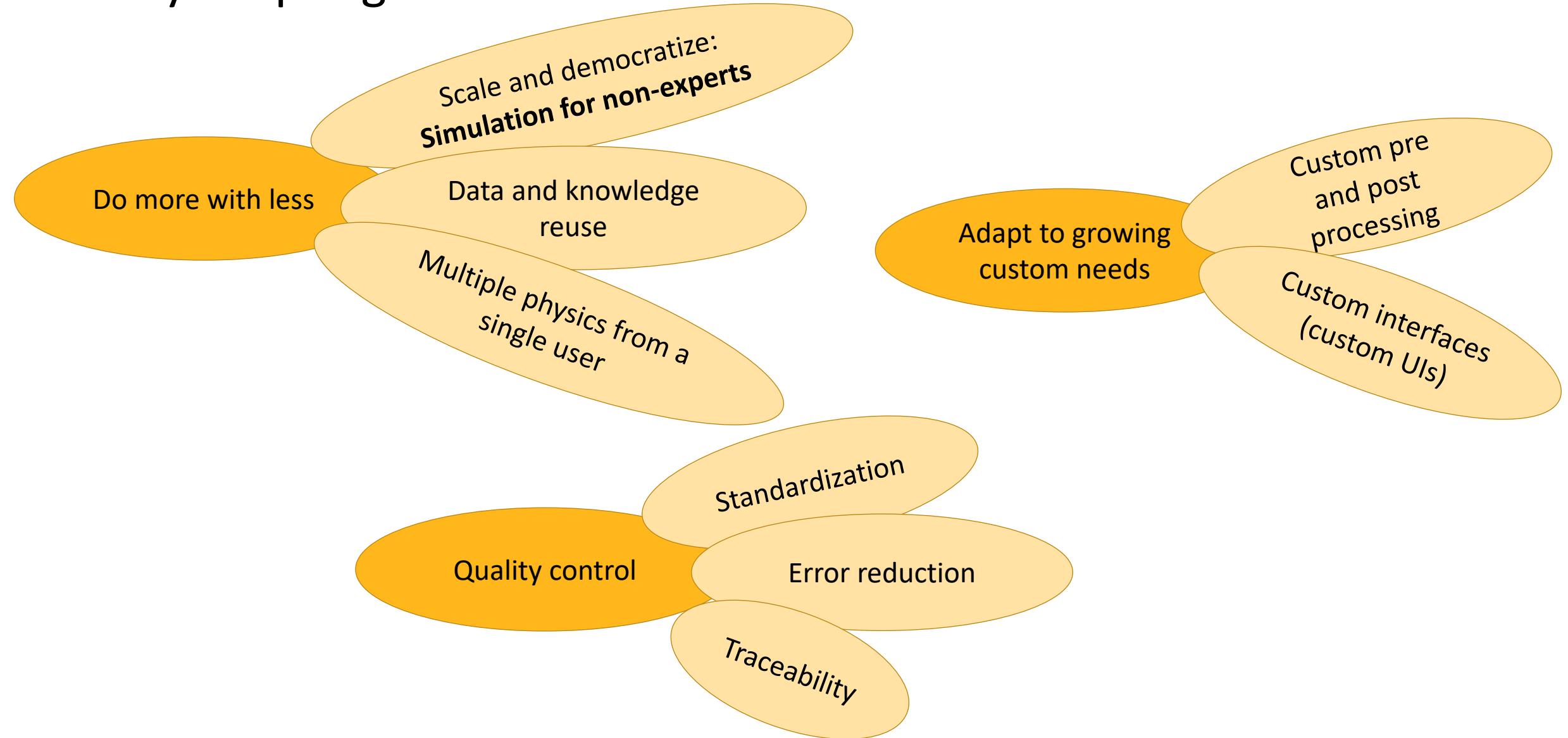
- Democratization of simulation (including non-engineers)
- Accelerate predictable insights
- Connect Ansys technologies to wider enabling technologies such as AI/ML

Ansys Assessment:

- Requirements can be classified into 5 Pillars:
Numerical Solvers, High-Performance Computations, AI/ML, Cloud and Digital Engineering.
- Need for strong and open APIs



Why scripting?



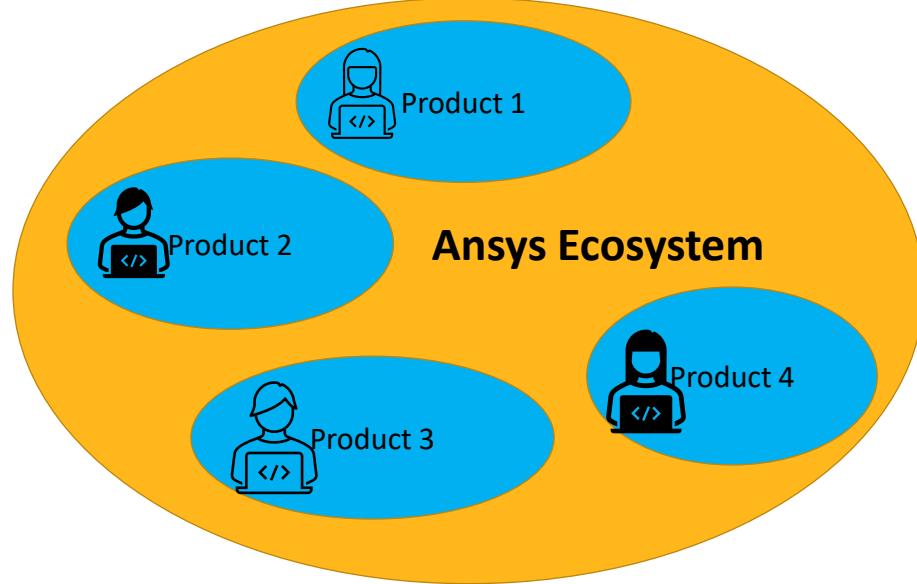
Why scripting?

It's all about increasing the value of our tools for
customers (*)

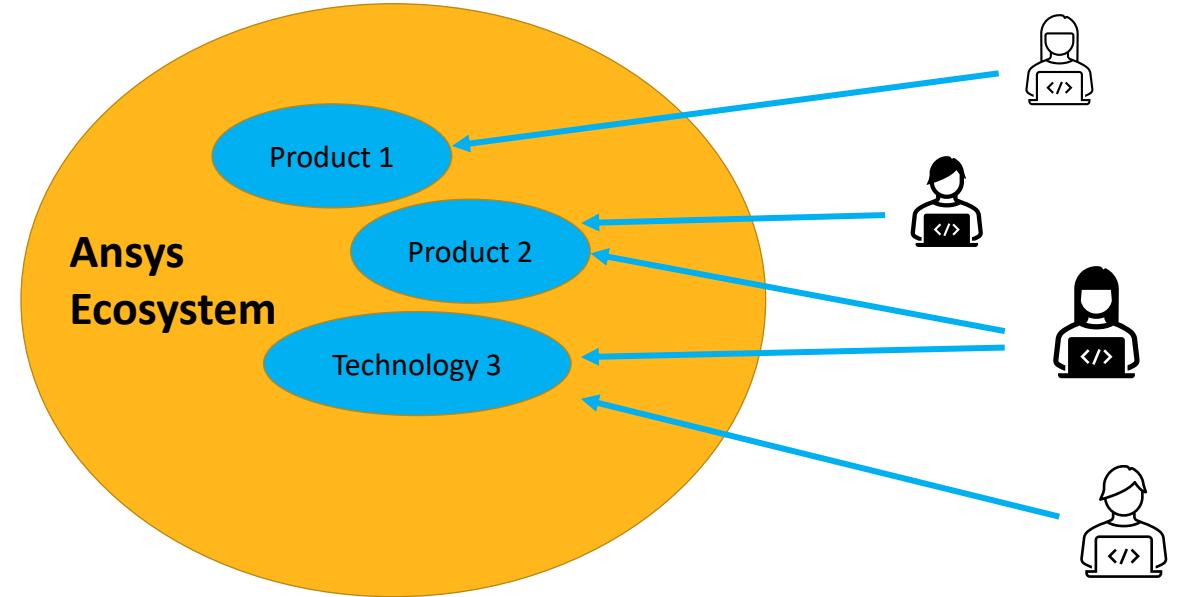
(*) and saving ourselves time as well 😊

Everyone buys the same software from us but uses it in a different way.
Not every usage is covered by “standard” features.

Two possible approaches to scripting with Ansys

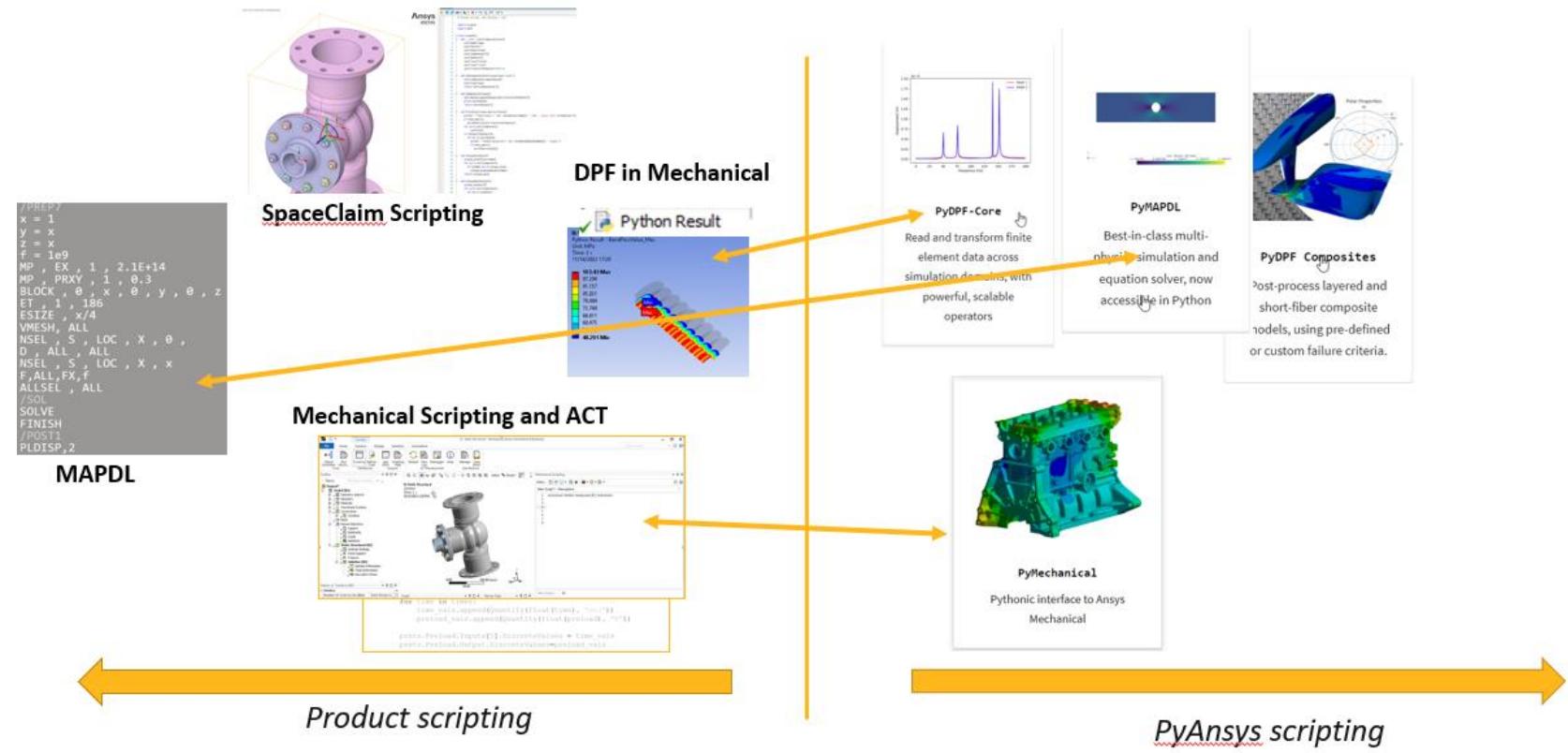


Product scripting



PyAnsys scripting

With a lot of similarities:



PyAnsys



What is PyAnsys?



=



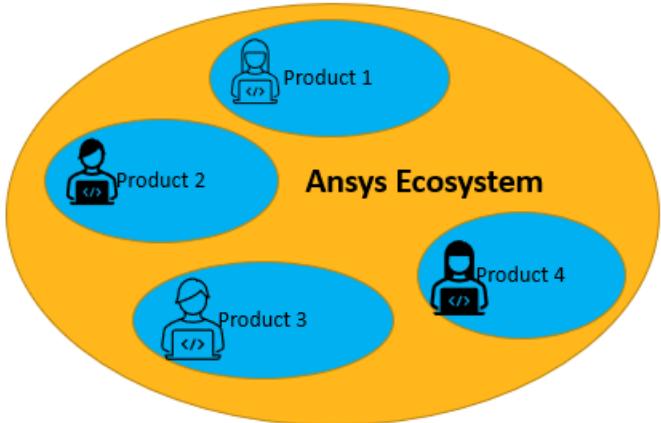
Python

Ansys

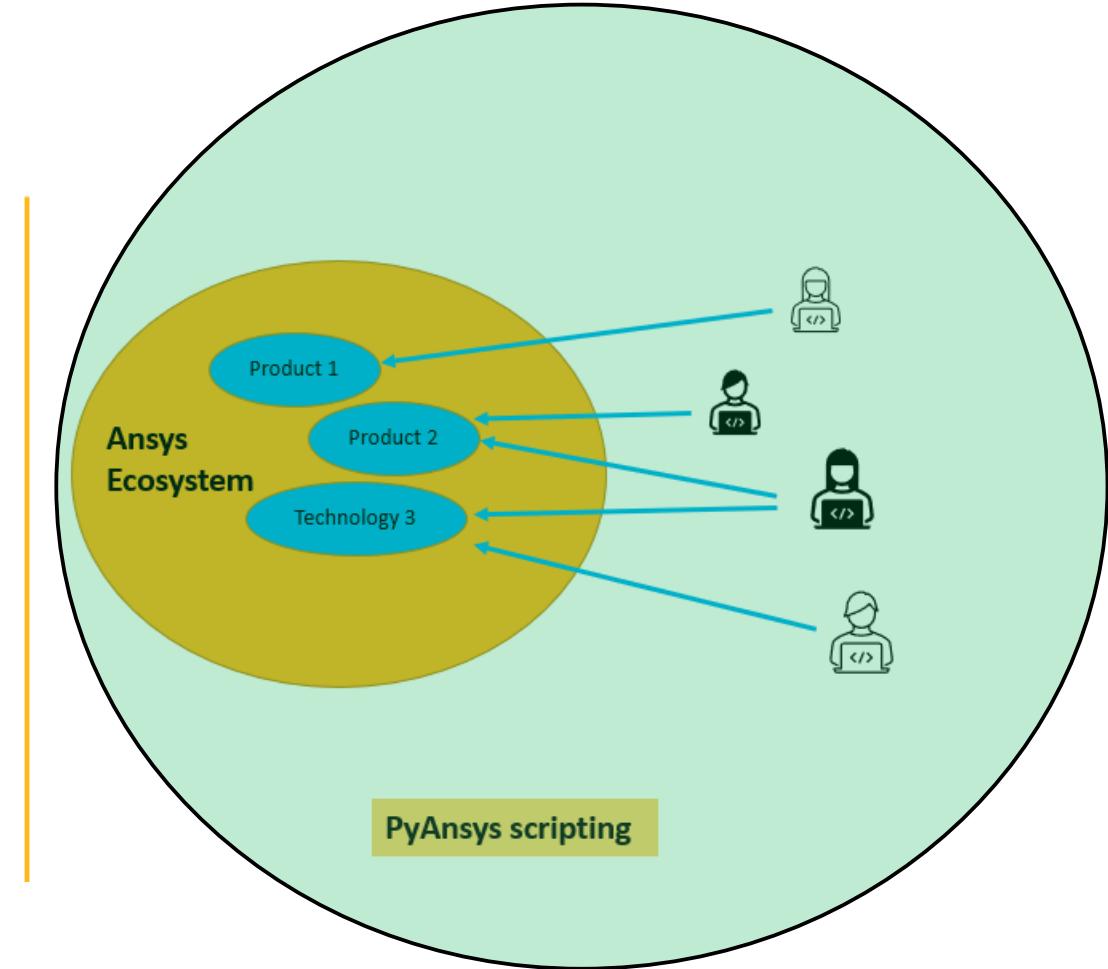
PyAnsys

Set of technologies that allow the user to interface with Ansys products pythonically

PyAnsys scripting



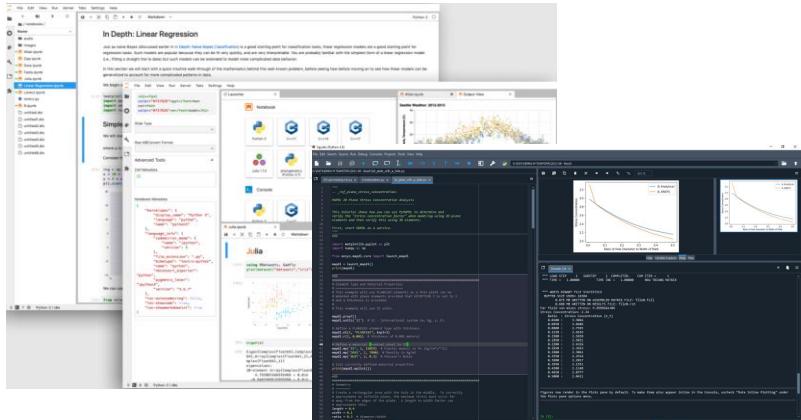
Product scripting



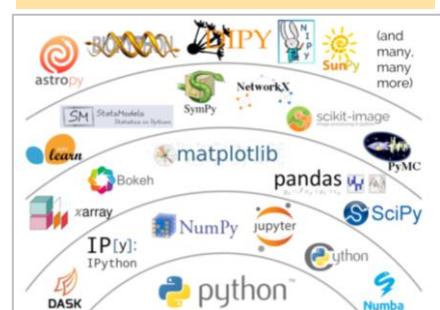
PyAnsys scripting

Scripting from within YOUR tools

Any tool from which you can write Python (UI is yours to choose)



Possibility to use other useful Python tools



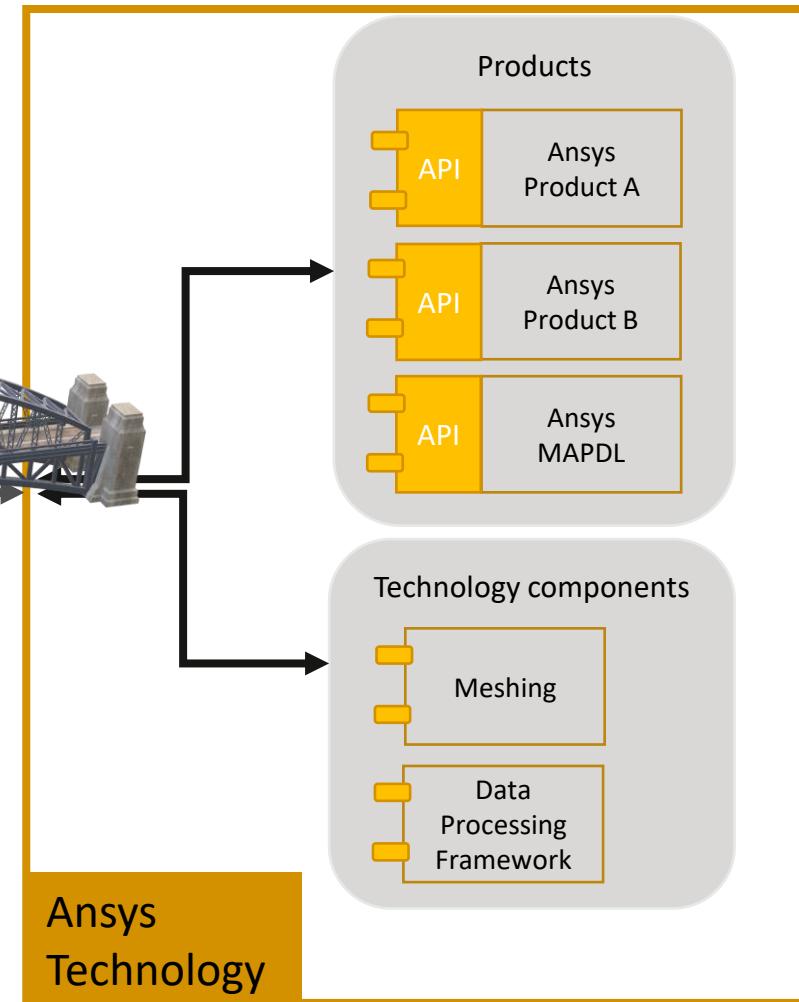
Customer tools



PyAnsys
technology



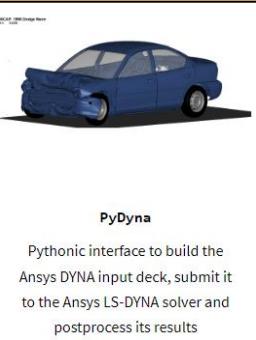
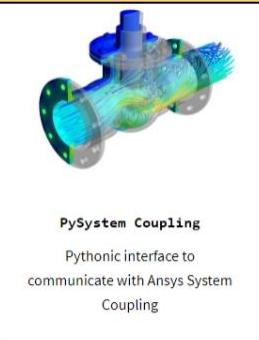
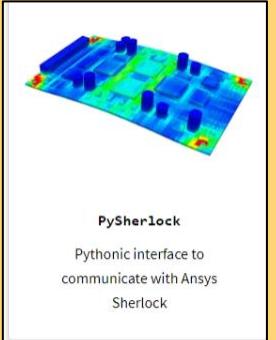
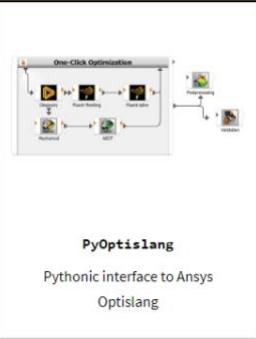
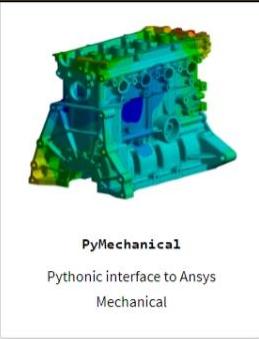
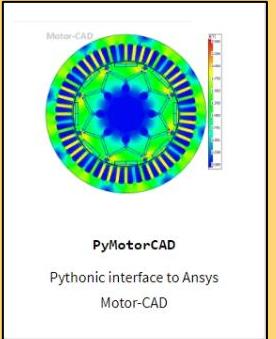
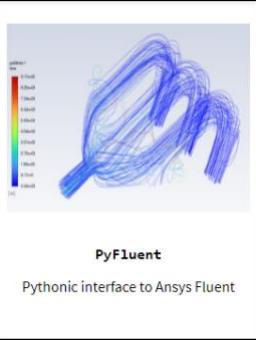
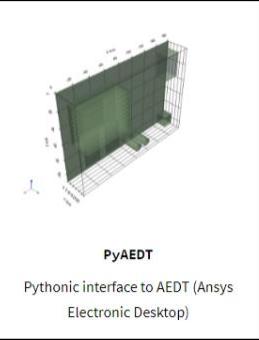
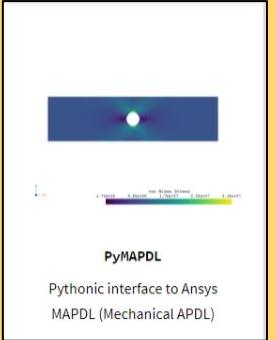
Ansys
Technology



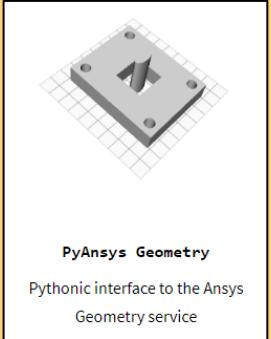
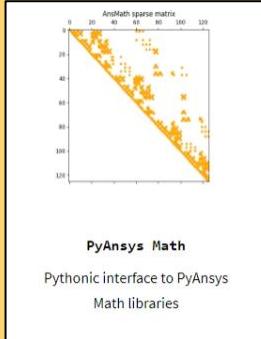
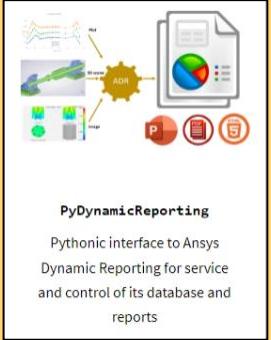
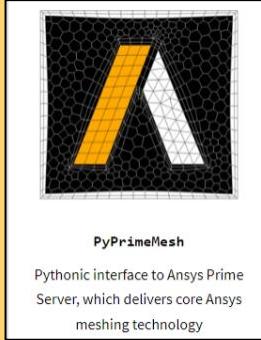
What Packages Are Available?

Check out all available packages in the [PyAnsys doc](#)

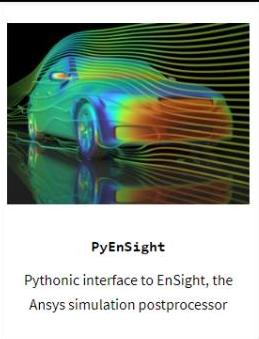
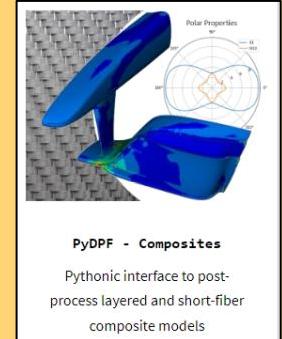
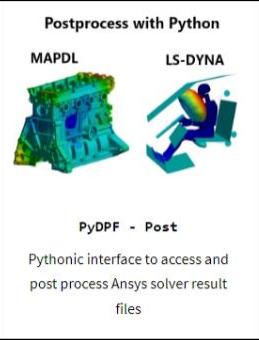
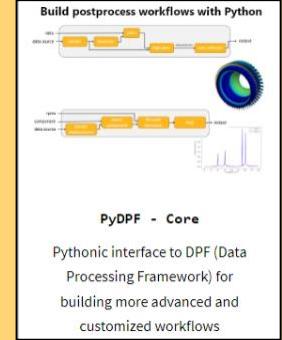
Simulation Libraries



Utility Libraries

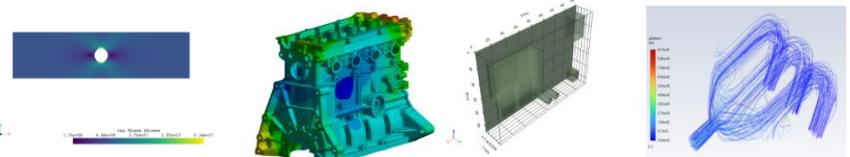


Postprocessing Libraries



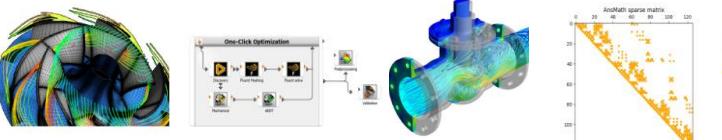
PyAnsys: What's available?

Simulation libraries

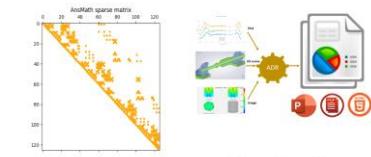


PyMAPDL
Pythonic interface to Ansys MAPDL (Mechanical APLD)
PyMechanical
Pythonic interface to Ansys Mechanical
PyAEDT
Pythonic interface to AEDT (Ansys Electronic Desktop)
PyFluent
Pythonic interface to Ansys Fluent

Utility libraries

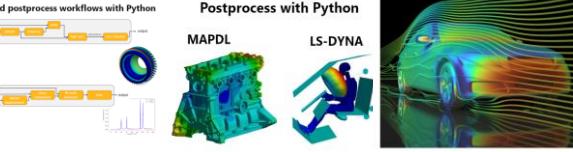


PyPrimeMesh
Pythonic interface to Ansys Prime Server, which delivers core Ansys meshing technology
PyOptislang
Pythonic interface to Ansys Optislang
PySystem Coupling
Pythonic interface to Ansys System Coupling



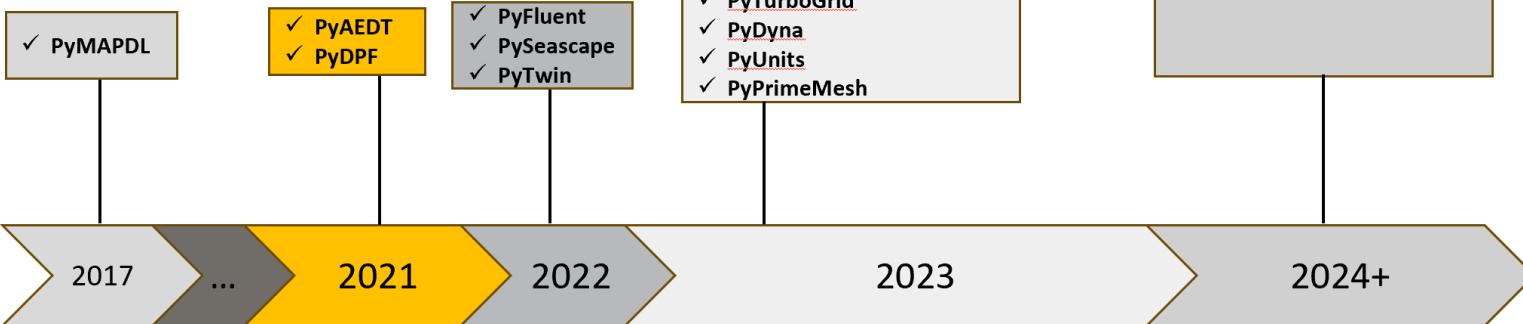
PyAnsys Math
Pythonic interface to PyAnsys Math libraries
PyDynamicReporting
Pythonic interface to Ansys Dynamic Reporting for service and control of its database and reports
PyDPF - Core
Pythonic interface to DPF (Data Processing Framework) for building more advanced and customized workflows

Post-processing libraries



Postprocess with Python
Build postprocess workflows with Python
PyDPF - Post
Pythonic interface to access and post process Ansys solver result files
PyEnSight
Pythonic interface to EnSight, the Ansys simulation postprocessor

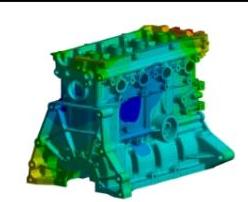
5-year strategy...



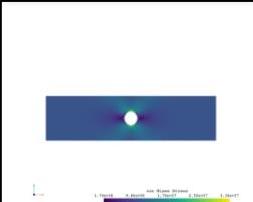
Keep an eye on...
docs.pyansys.com

And If I'm a Structural Engineer?

Simulation Libraries



PyMechanical
Pythonic interface to Ansys Mechanical

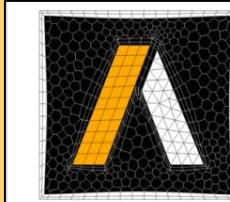


PyMAPDL
Pythonic interface to Ansys MAPDL (Mechanical APDL)



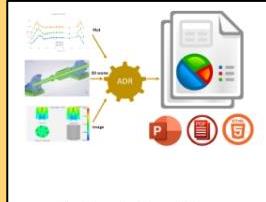
PyDyna
Pythonic interface to build the Ansys DYNA input deck, submit it to the Ansys LS-DYNA solver and postprocess its results

Utility Libraries

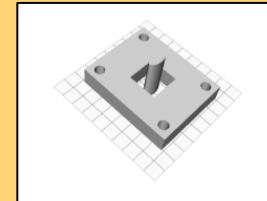


PyPrimeMesh

Pythonic interface to Ansys Prime Server, which delivers core Ansys meshing technology



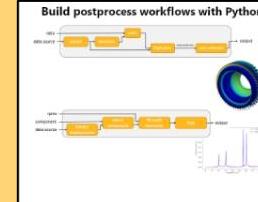
PyDynamicReporting
Pythonic interface to Ansys Dynamic Reporting for service and control of its database and reports



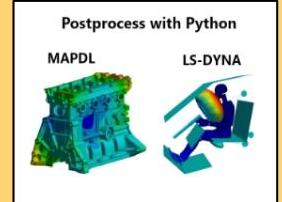
PyAnsys Geometry

Pythonic interface to the Ansys Geometry service

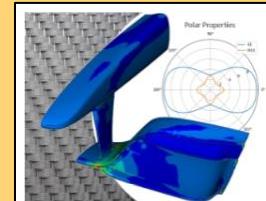
Postprocessing Libraries



PyDPF - Core
Pythonic interface to DPF (Data Processing Framework) for building more advanced and customized workflows

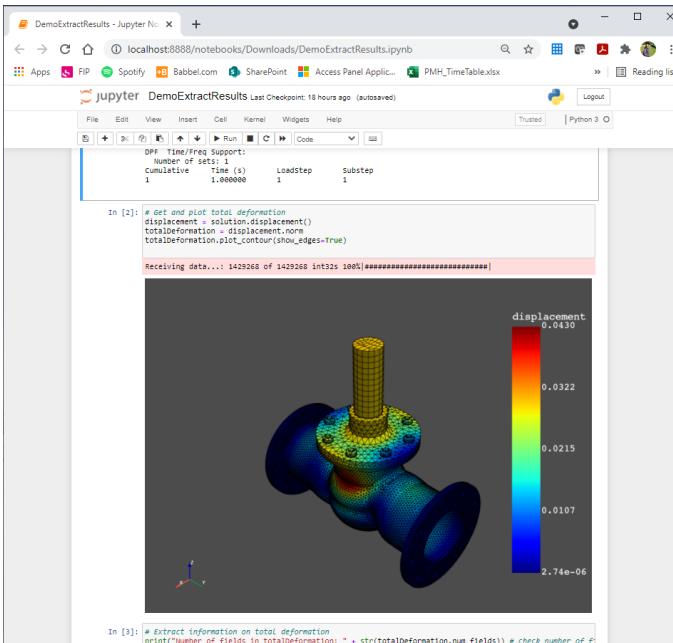


PyDPF - Post
Pythonic interface to access and post process Ansys solver result files

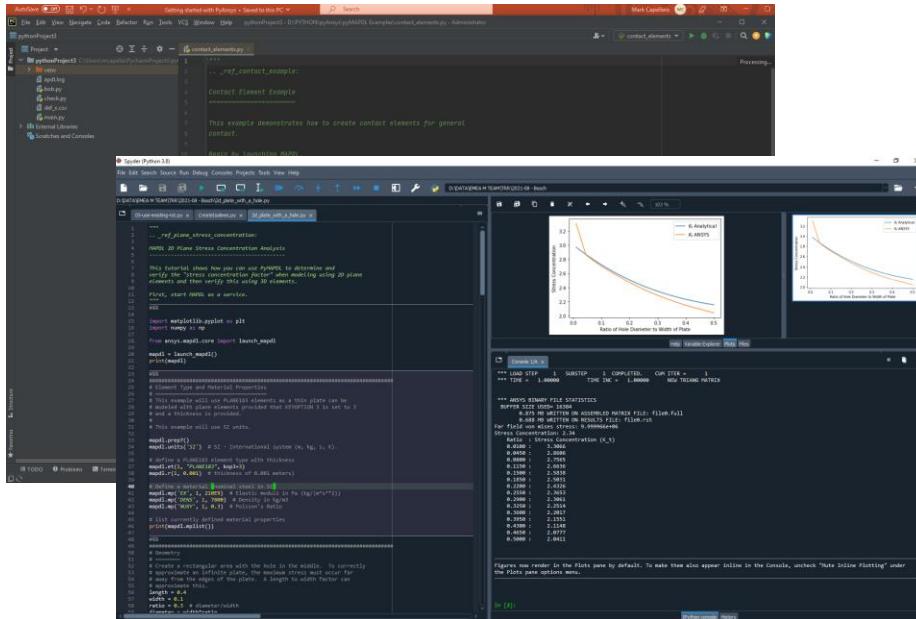


PyDPF - Composites
Pythonic interface to post-process layered and short-fiber composite models

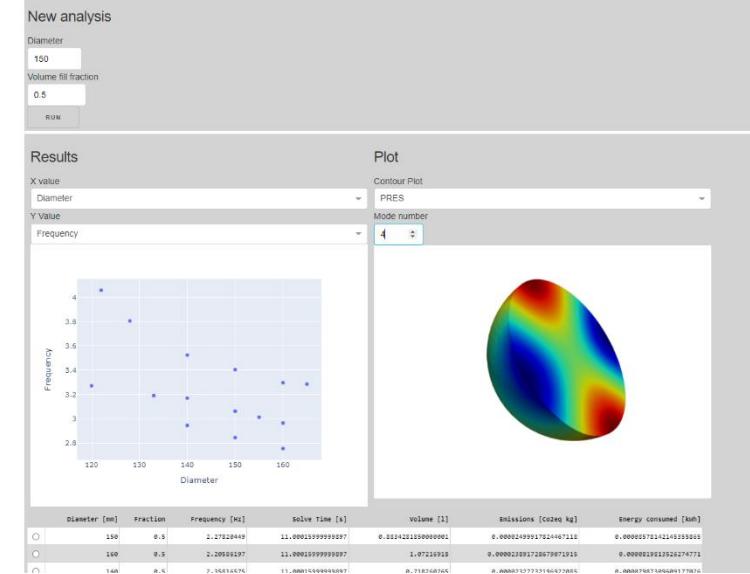
Is there a user interface (UI)?



Code can be written from a web-page thanks to Jupyterlab

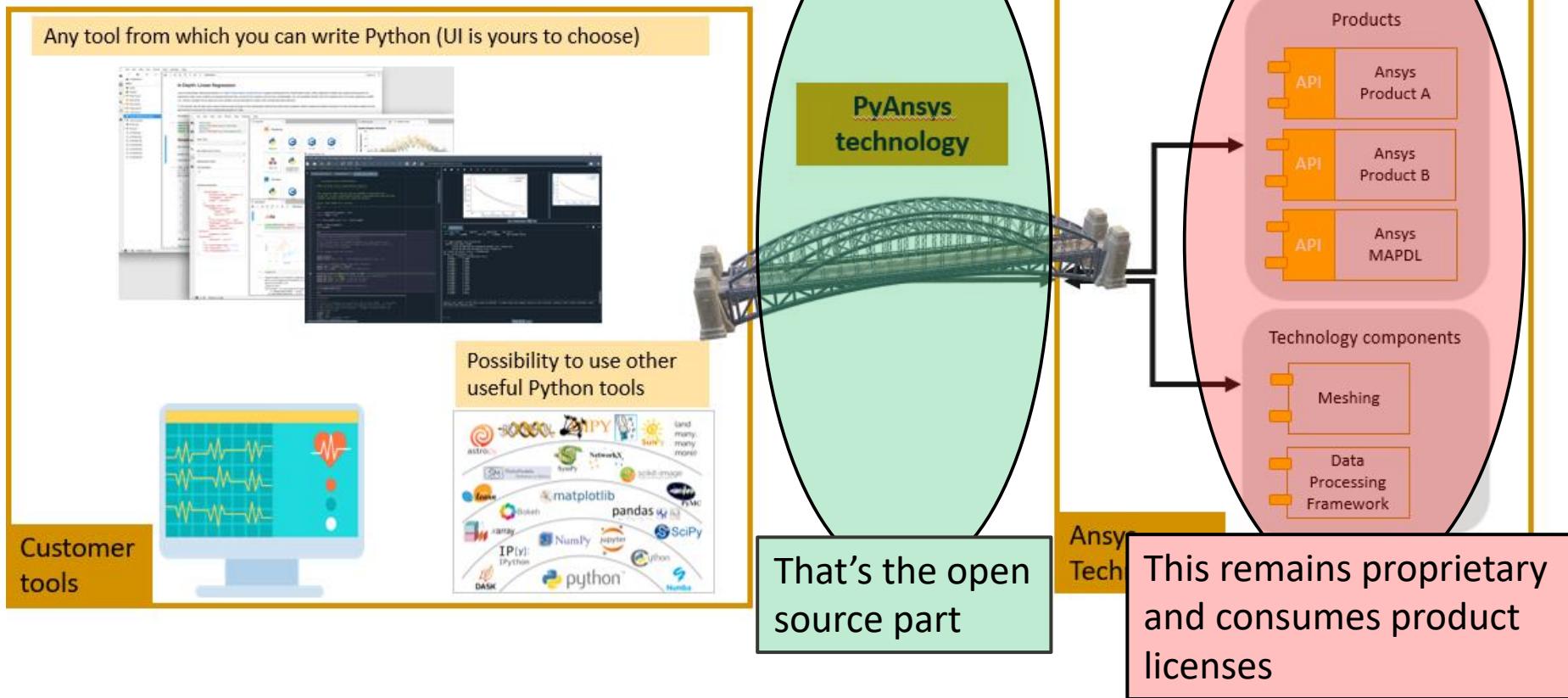


An IDE can be used (PyCharm, Spyder, ...)



User can also develop apps by coding both the front end and the back end

Open source ?! What's in it for us?



We're just offering a new way of using our products!
This will allow existing users to use our products differently, effectively and efficiently.

PyAnsys: Getting Started



Use Ansys Python Manager – Graphical Installation (windows & linux)

Need:

- Similar to the **Anaconda GUI installer**: Transition to a **Commercial Business Model** was a major roadblock for Ansys Customers.
- Reduce Impact for Beginner & Intermediate Python users.



Solution/Tool Created:

- A simple 3-step installer app.

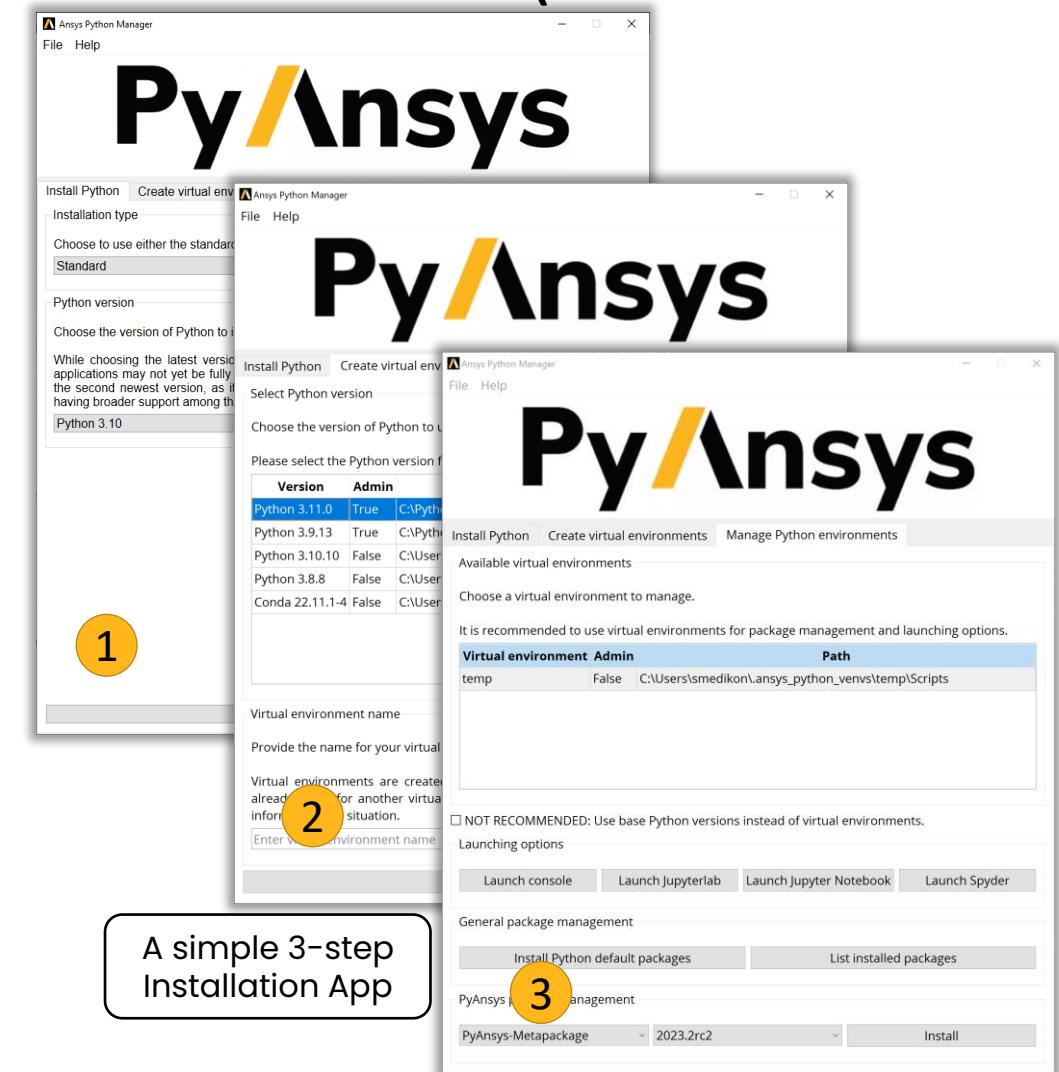


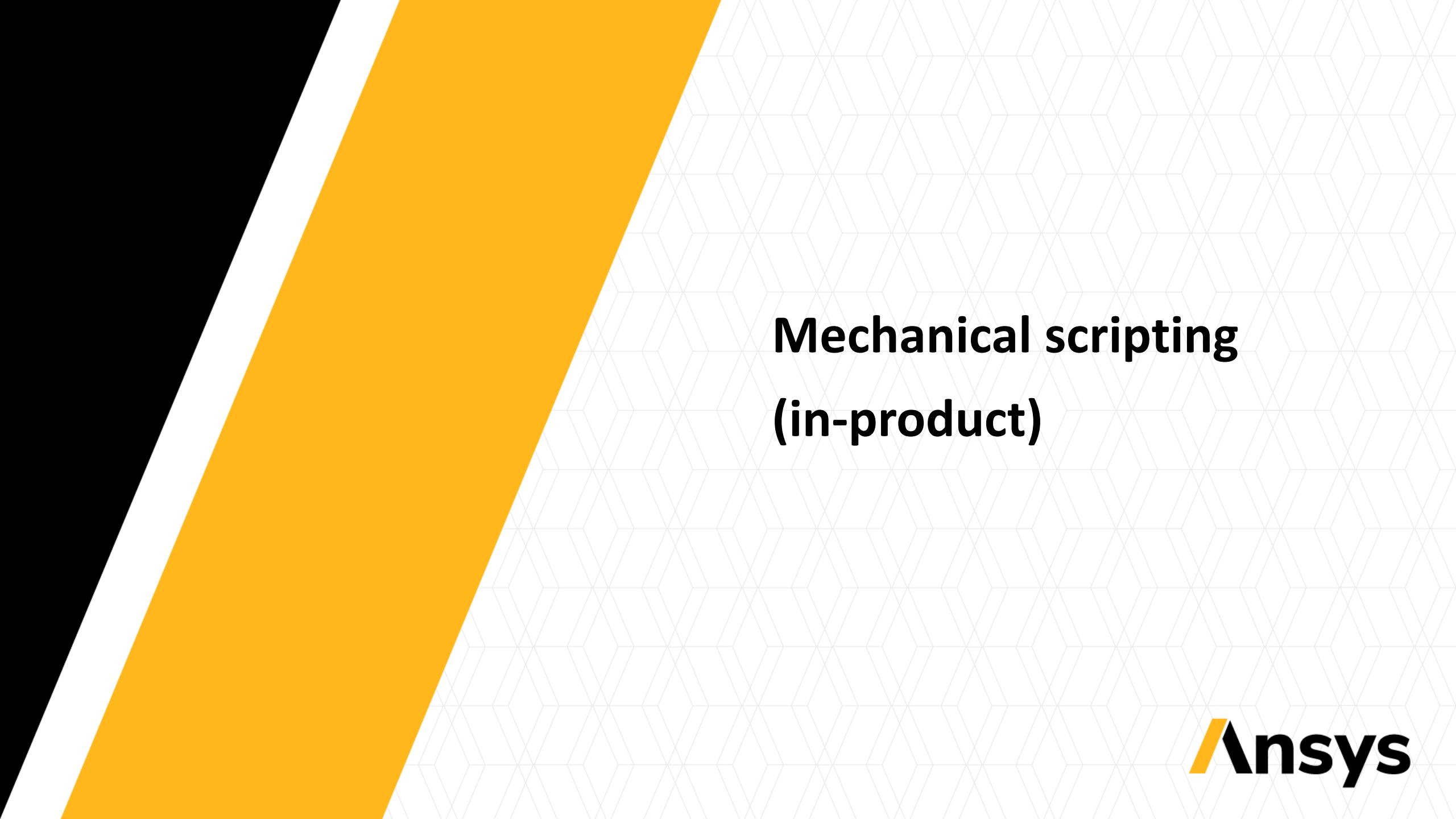
- Using Ansys Python Manager:

Download it from:

[Releases · ansys/python-installer-qt-gui
\(github.com\)](https://github.com/ansys/python-installer-qt-gui/releases)

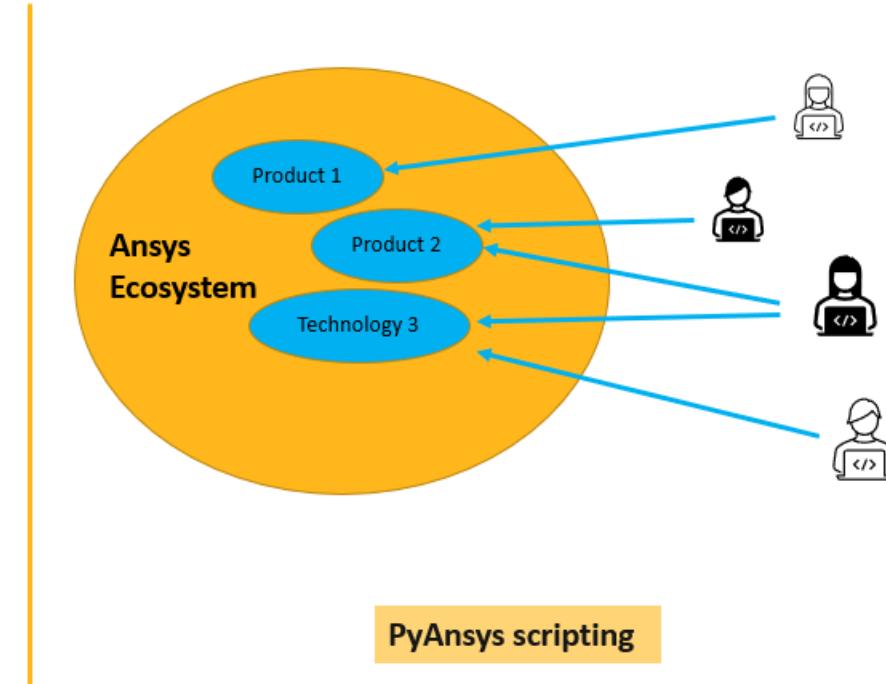
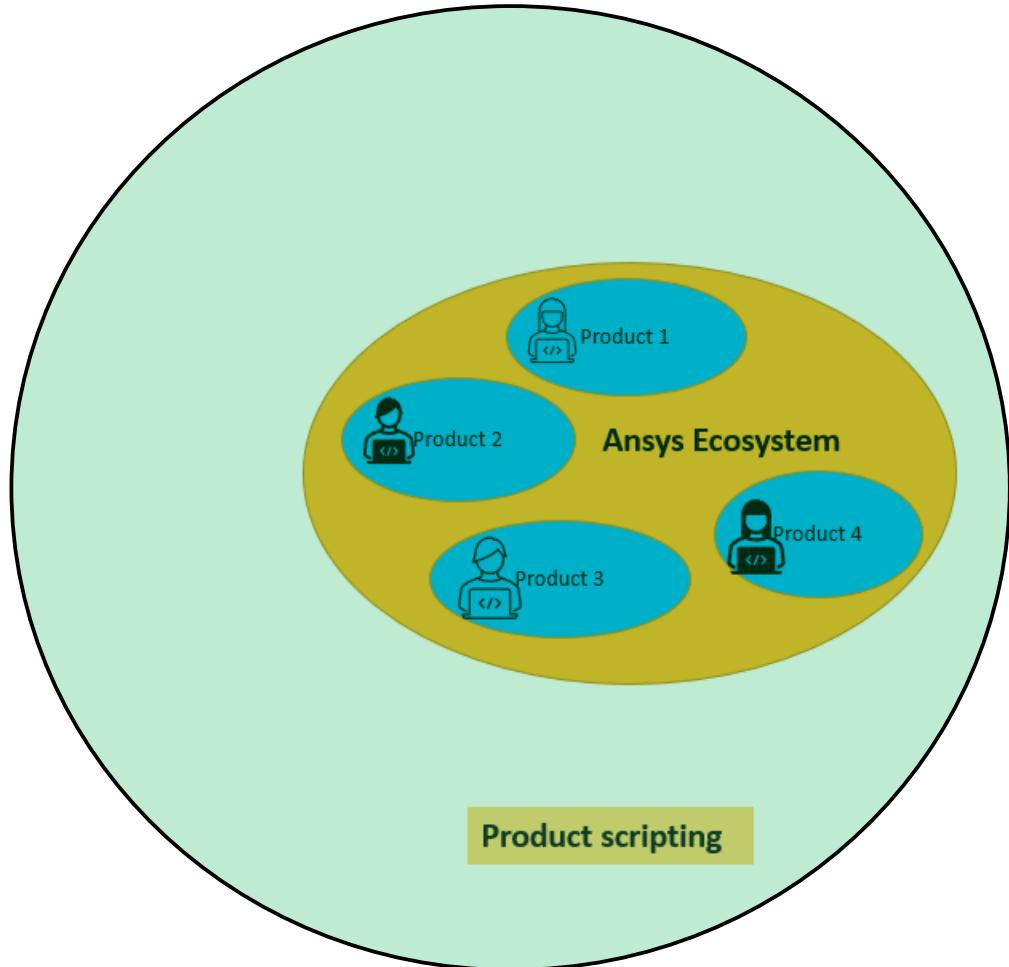
<https://developer.ansys.com/ansys-python-manager>



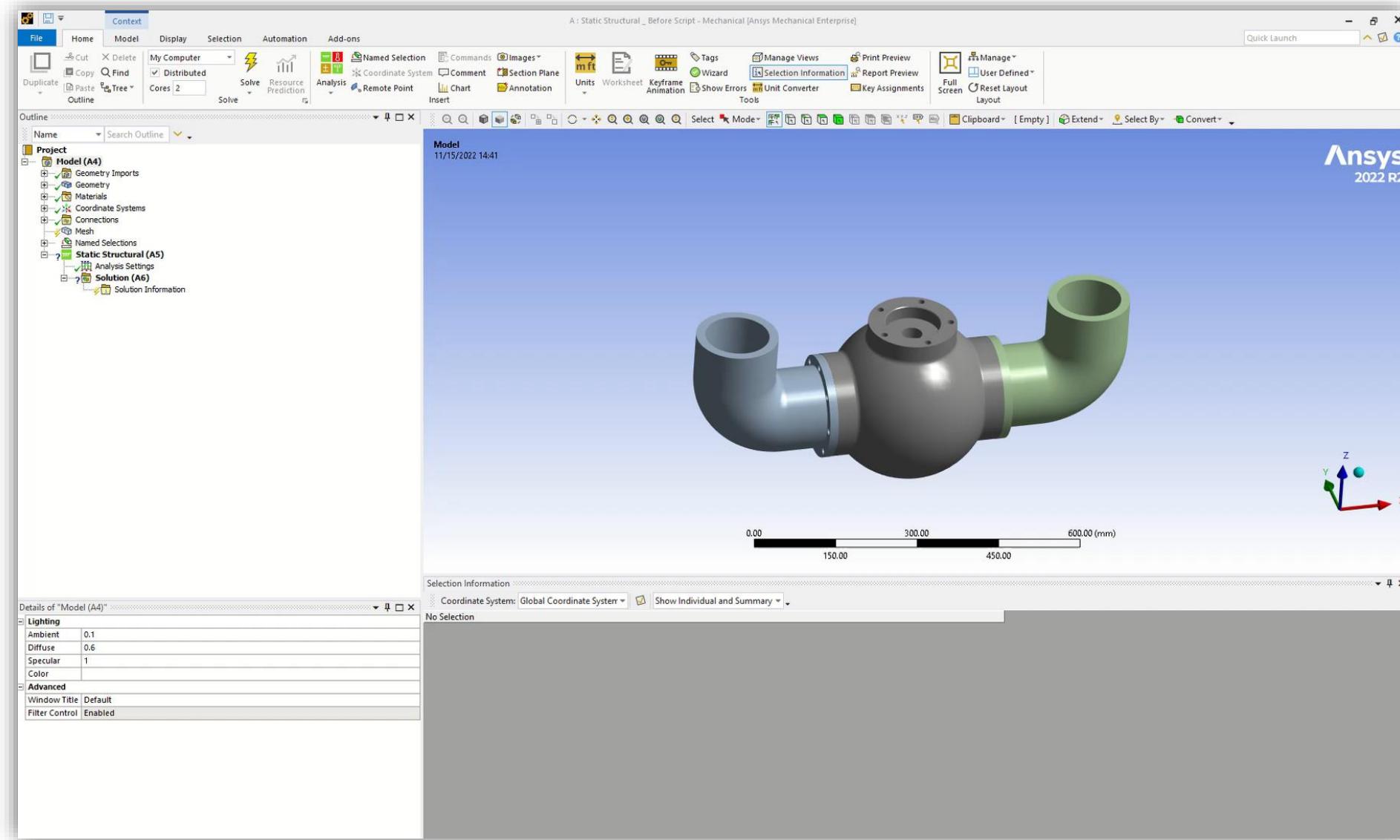


Mechanical scripting (in-product)

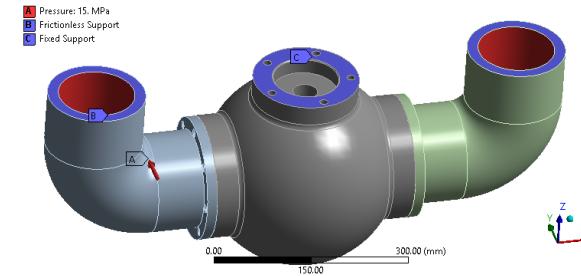
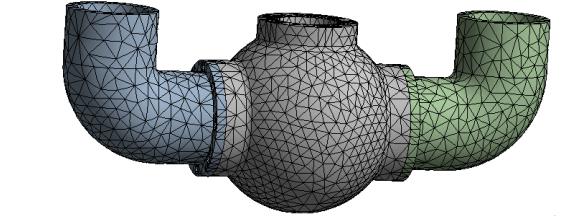
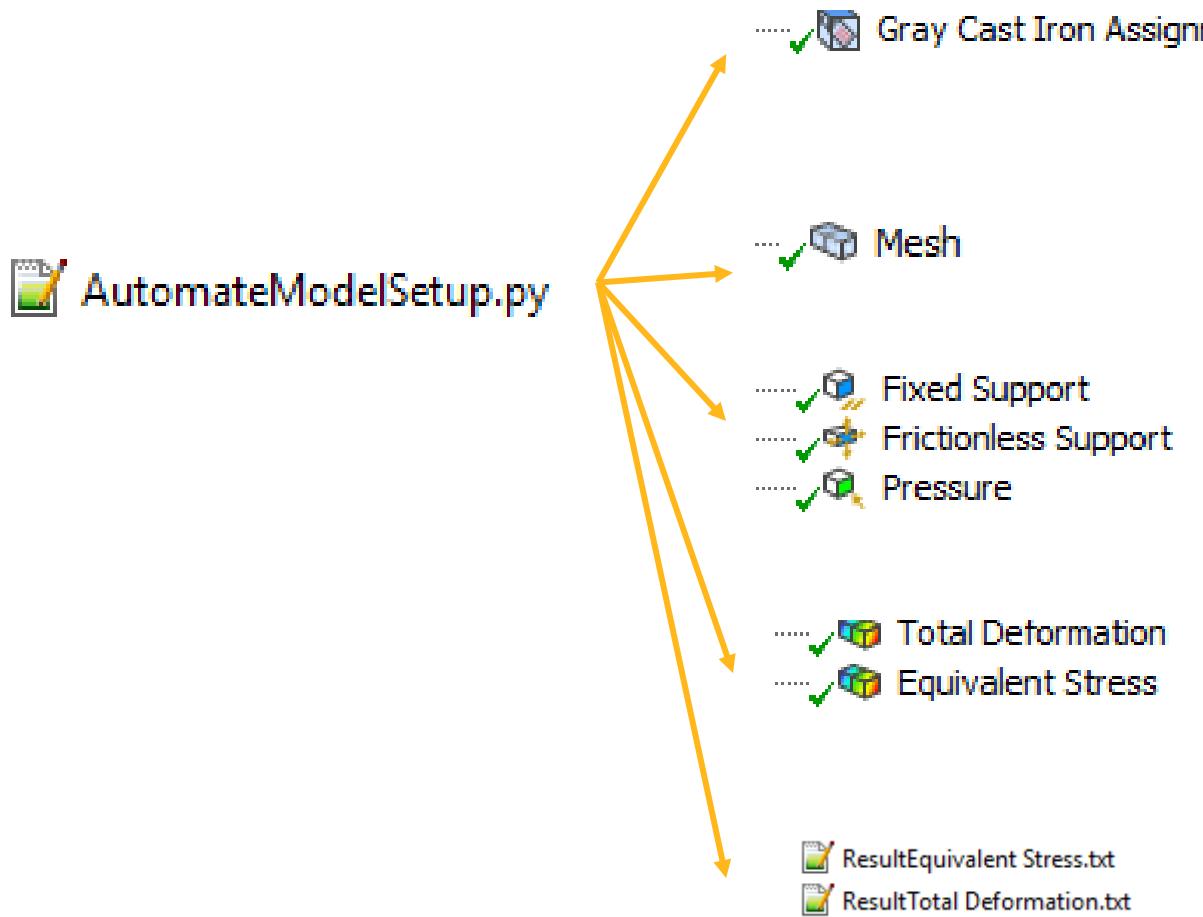
Product scripting



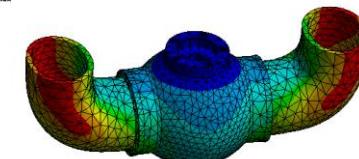
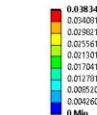
Mechanical scripting demo: complete model setup



Mechanical scripting demo: complete model setup



A: Static Structural
Total Deformation
Type: Total Deformation
Unit: mm
Time: 1
05/10/2010 17:55



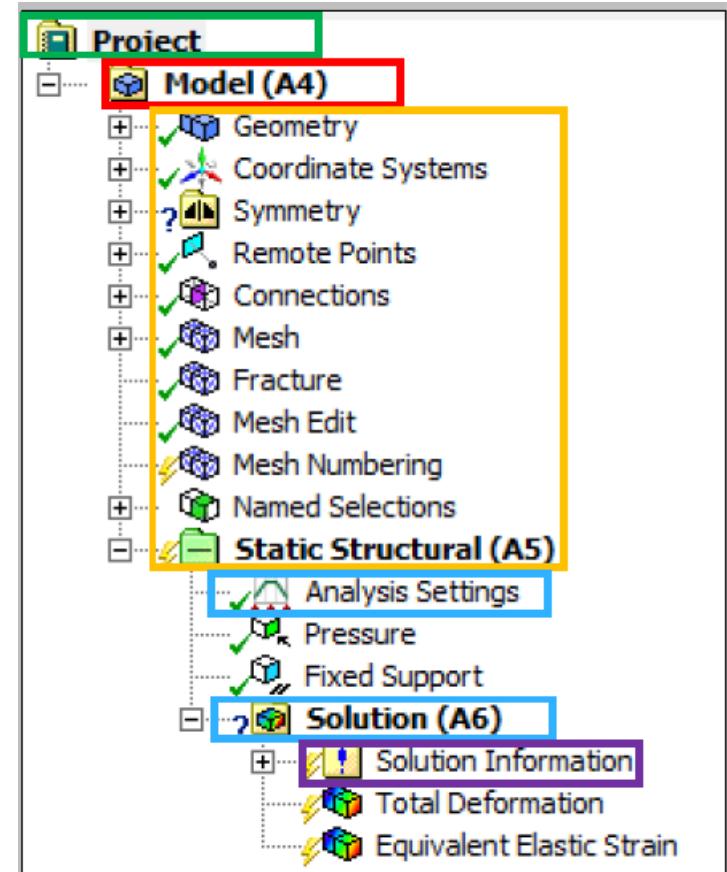
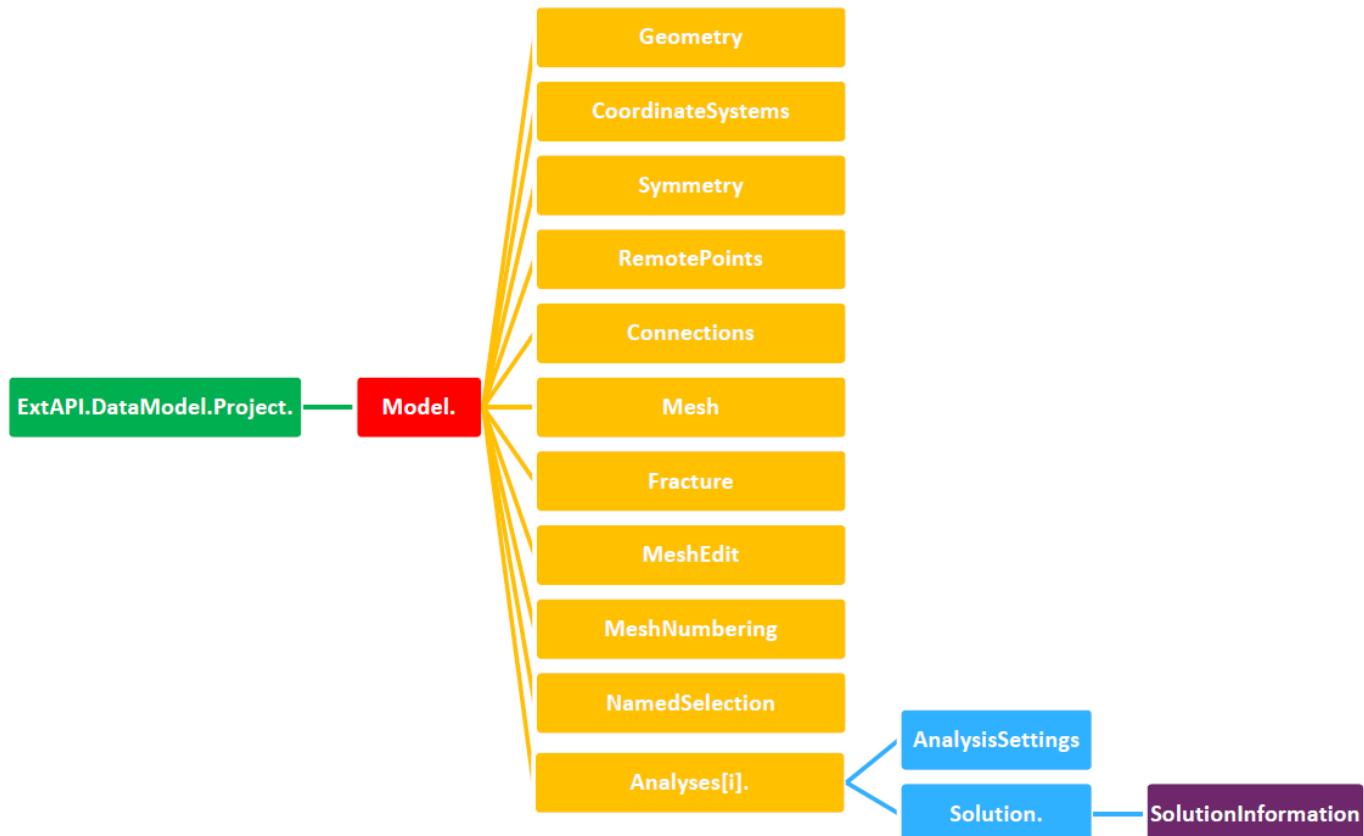
0.00 400.00 (mm)

z
y
x

Node Number	Equivalent (von-Mises) Stress (MPa)
11136	4.9244
11137	5.2469
11138	4.3853
11139	5.9214
11140	5.5261
11141	4.1676

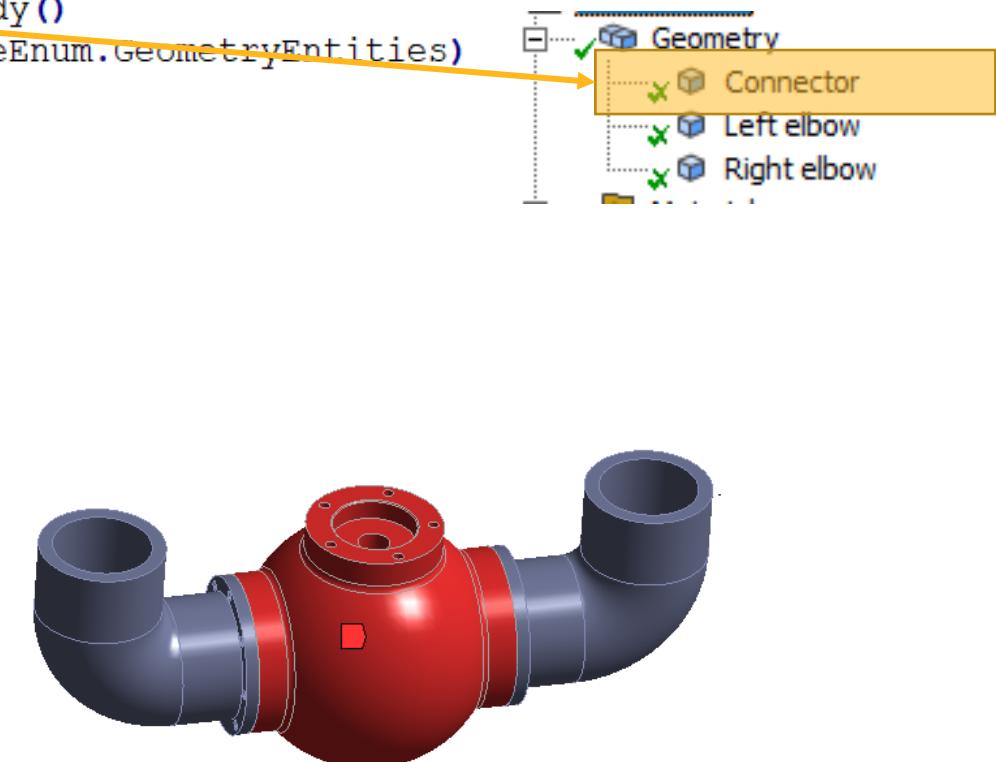
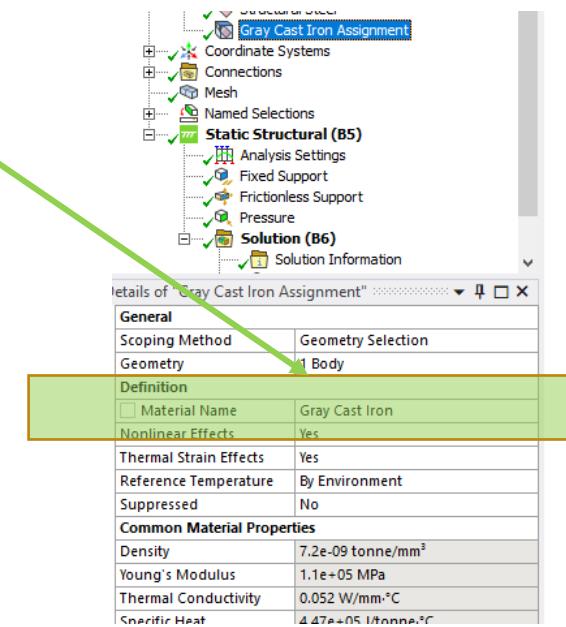
Mechanical Automation API

```
# Reference model  
model = ExtAPI.DataModel.Project.Model
```



Let's dive into the script: define material assignment

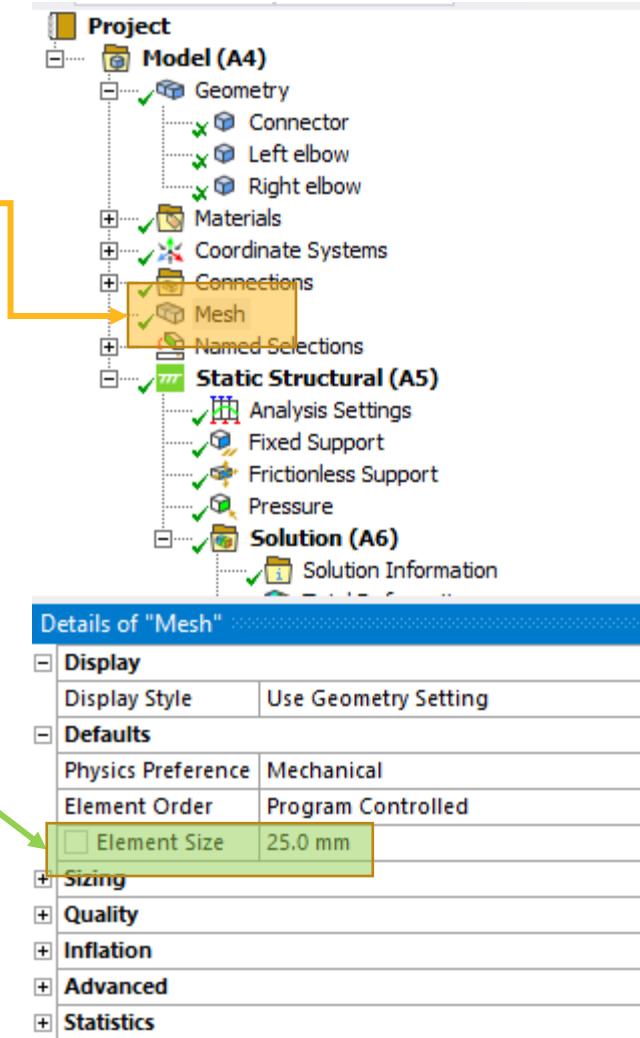
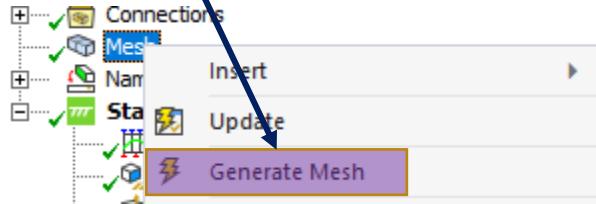
```
# Change material assignment for body named "Connector"
import materials
matAssignment = model.Materials.AddMaterialAssignment()
body = ExtAPI.DataModel.GetObjectsByName("Connector")[0].GetGeoBody()
tempSel = ExtAPI.SelectionManager.CreateSelectionInfo(SelectionTypeEnum.GeometryEntities)
tempSel.Ids = [body.Id]
matAssignment.Location = tempSel
matAssignment.Material = "Gray Cast Iron"
ExtAPI.DataModel.Tree.Refresh()
```



Let's dive into the script: define mesh settings

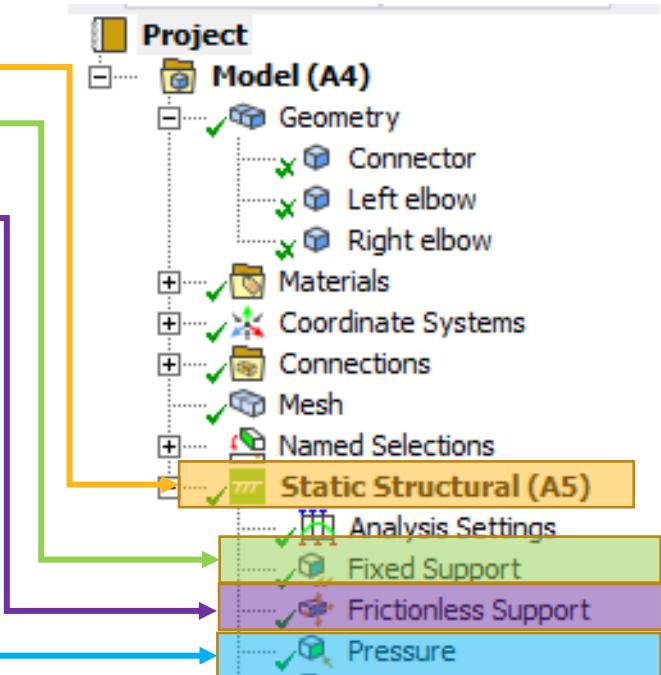
```
# Define mesh settings
```

```
mesh = model.Mesh  
mesh.ElementSize = Quantity('25 [mm]')  
mesh.GenerateMesh()
```



Let's dive into the script: define boundary conditions

```
# Define boundary conditions:  
analysis = model.Analyses[0]  
  
fixedSupport = analysis.AddFixedSupport()  
fixedSupport.Location = ExtAPI.DataModel.GetObjectsByName("NSFixedSupportFaces")[0]  
  
frictionlessSupport = analysis.AddFrictionlessSupport()  
frictionlessSupport.Location = ExtAPI.DataModel.GetObjectsByName("NSFrictionlessSupportFaces")[0]  
  
pressure = analysis.AddPressure()  
pressure.Location = ExtAPI.DataModel.GetObjectsByName("NSInsideFaces")[0]  
pressure.Magnitude.Inputs[0].DiscreteValues = [Quantity("0 [s]"), Quantity("1 [s]")]  
pressure.Magnitude.Output.DiscreteValues = [Quantity("0 [Pa]"), Quantity("15 [MPa]")]
```



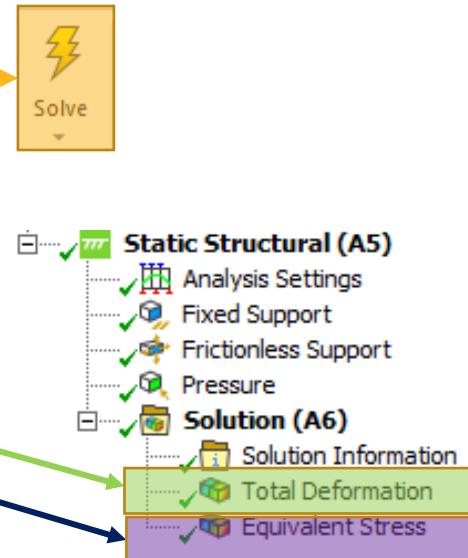
The screenshot shows three tabs in the interface:

- Details of "Fixed Support"**: Shows Scoping Method as Named Selection (`NSFixedSupportFaces`) and Definition ID (Beta) as 92.
- Details of "Frictionless Support"**: Shows Scoping Method as Named Selection (`NSFrictionlessSupportFaces`) and Definition ID (Beta) as 94.
- Details of "Pressure"**: Shows Scoping Method as Named Selection (`NSInsideFaces`) and Definition ID (Beta) as 96. The **Definition** section includes a **Tabular Data** table:

Steps	Time [s]	Pressure [MPa]
1	0.	0.
2	1.	15.
*		

Let's dive into the script: solve and postprocess

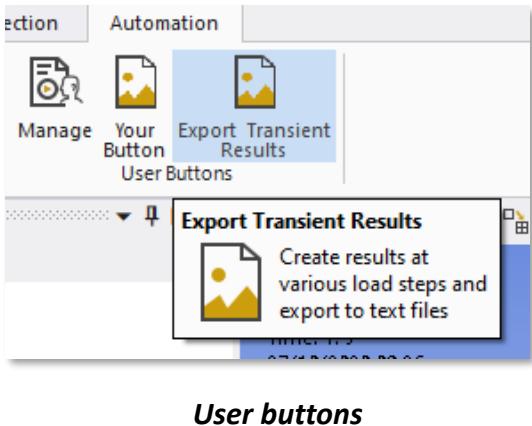
```
# Solve model  
model.Solve()  
  
# Add results  
solution = analysis.Solution  
solution.AddTotalDeformation()  
solution.AddEquivalentStress()  
solution.EvaluateAllResults()  
  
# Export result values to a text file  
filePath=r"D:\Export\Result"  
fileExtension=r".txt"  
results = solution.GetChildren(DataModelObjectCategory.Result, True)  
for result in results:  
    fileName = str(result.Name)  
    result.ExportToTextFile(True, filePath+fileName+fileExtension)
```



Node Number	Equivalent (von-Mises) Stress (MPa)
11136	4.9244
11137	5.2469
11138	4.3853
11139	5.9214
11140	5.5261
11141	4.1676
11142	2.8028
11143	3.7985
11144	20.445
11145	22.654
11146	27.851
11147	36.228
...	...

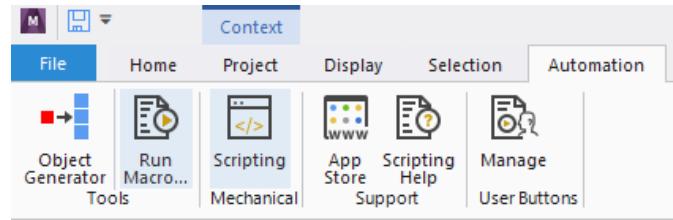
Mechanical Automation Scripting

- Well documented Python APIs with scripting guide and quick start examples
- Shell and multi-line editor to build and debug scripts quickly
- Supports autocomplete and Intellisense
- Ability to easily promote scripts to user buttons
- Recording



29

©2024 ANSYS, Inc.

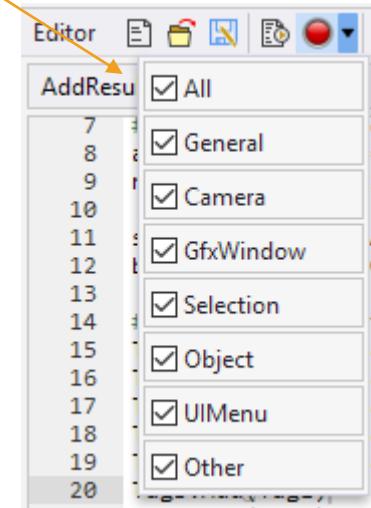


```
AddResult : Description
7 #.Get.the.number.of.steps.for.the.analysis
8 analysisSettings = Model.Analyses[0].AnalysisSetting
9 numSteps = analysisSettings.NumberOfSteps
10
11 solution = Model.Analyses[0].Solution
12 bolt = DataModel.GetObjectsByName("Bolt")[0]
13
14 #.Create.tags.that.we.will.use.later.for.finding.objects
15 Tags = ExtAPI.DataModel.ObjectTags
16 Tag1 = Ansys.Mechanical.Application.ObjectTag ("A1")
17 Tag2 = Ansys.Mechanical.Application.ObjectTag ("Bolt")
18 Tag3 = Ansys.Mechanical.Application.ObjectTag ("U_Sur")
19 Tag4 = Ansys.Mechanical.Application.ObjectTag ("EQV")
20 Tags.Add(Tag1)
21 Tags.Add(Tag2)
22 Tags.Add(Tag3)
23 Tags.Add(Tag4)
24
25 #.For.each.step.add.the.desired.result.objects.with.i
26 for step in range(1,numSteps):
27
28 for step in range(1,numSteps):
29
```

Shell

```
>>> analysisSettings = Model.Analyses[0].AnalysisSettings
>>> numSteps = analysisSettings.NumberOfSteps
>>> numSteps
1
>>>
```

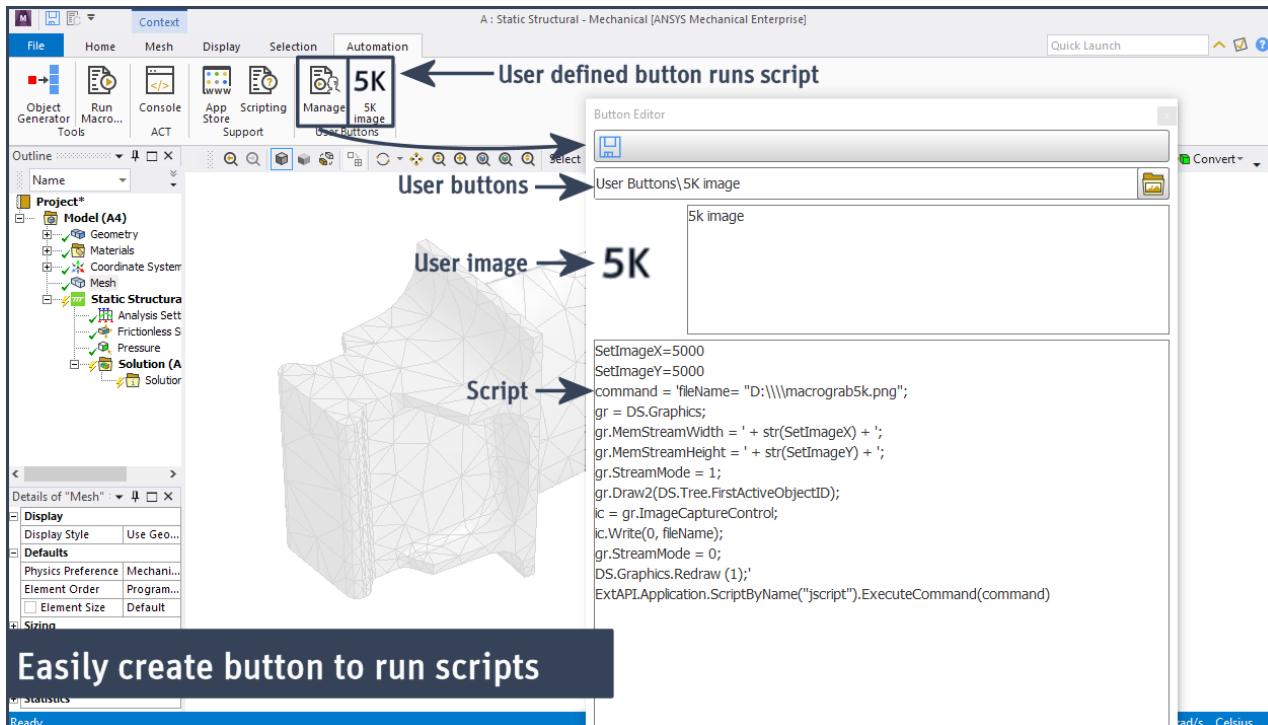
Record button with control of how much to journal



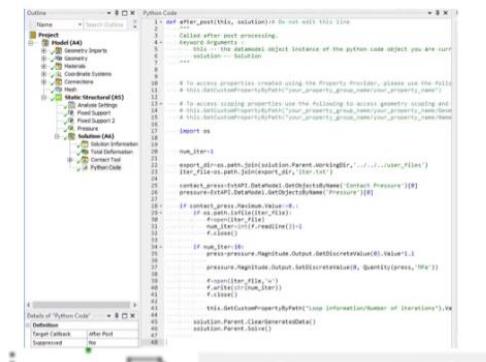
Powering Innovation That Drives Human Advancement

Ansys

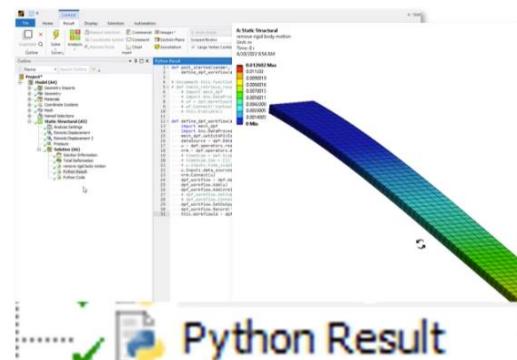
More advanced usage: 1-time execution, Python Code & Python Result Objects



Save script to button
(Script executed when clicked on)



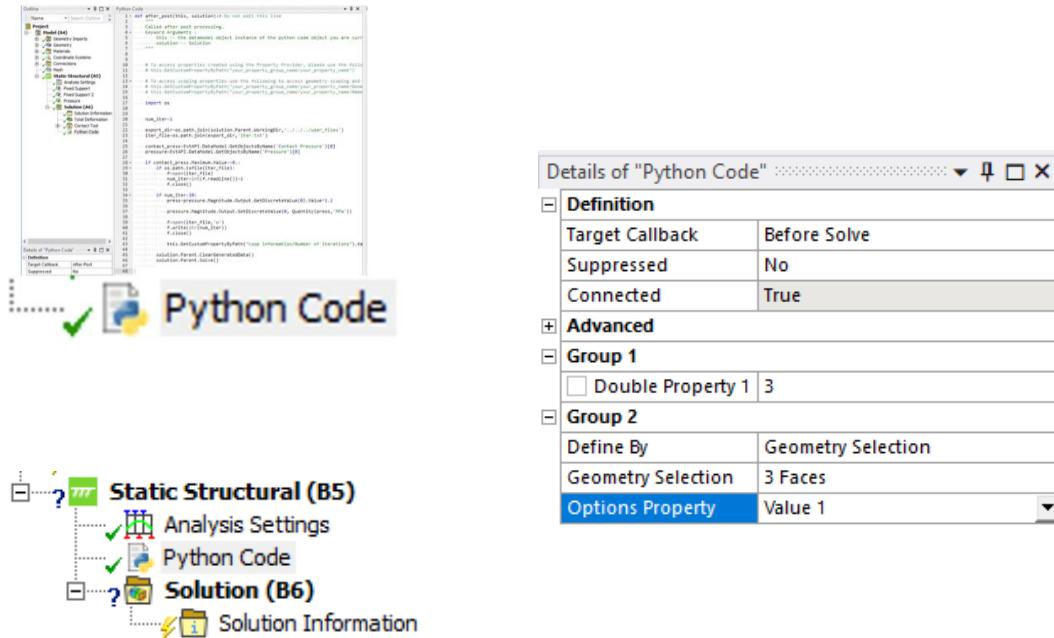
Python Code



Execute code automatically during simulation

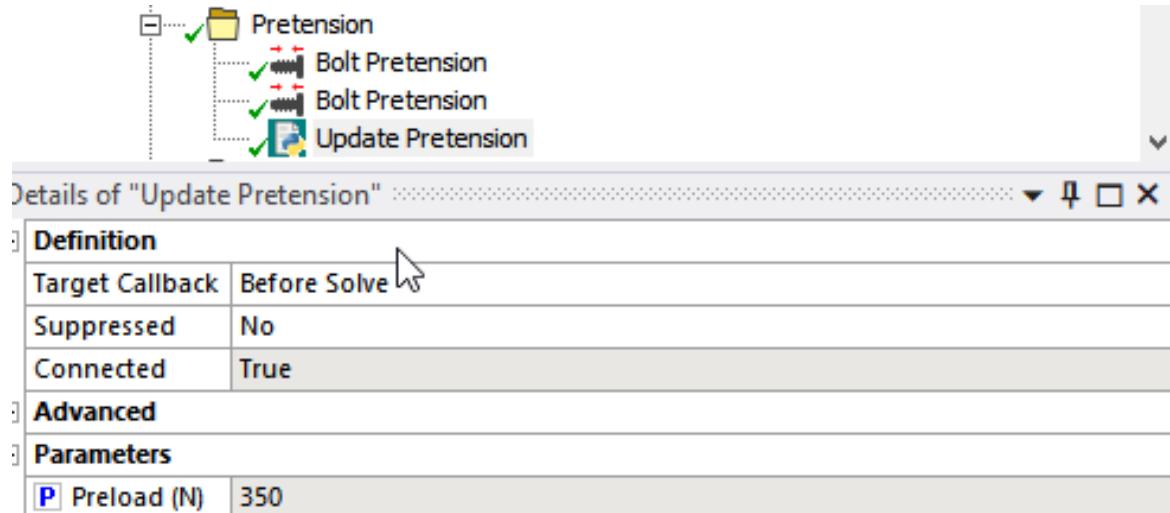
Within the simulation tree to execute with each simulation

Python Code Object



- Execute Python code in response to events occurring in Mechanical (“target callback”)
- Use Python code to inject MAPDL commands into the solver input
- Define custom properties

Python Code example: Create new parameters for design variations



Preload value available as parameter

- *Update can be applied to one or more pretension objects*
- *Called before solving to update value sent to the solver*

```
def before_solve(this, analysis):# Do not edit this line
    """
    Called before solving the parent analysis.
    Keyword Arguments :
        this -- this python object
        analysis -- Static Structural
    """
    preload =
    this.PropertyProvider.GetPropertyByName("Parameters/Preload
(N)").Value

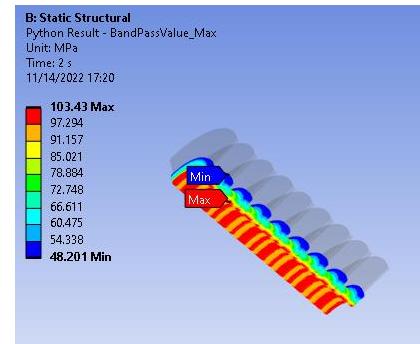
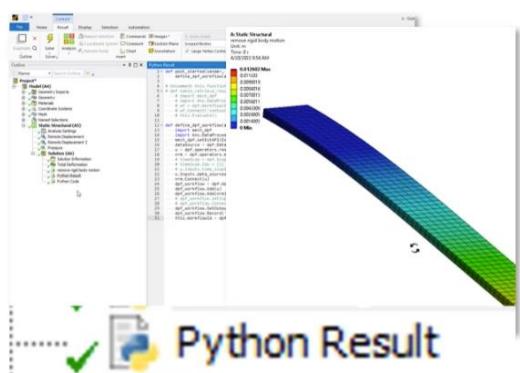
    all_prets = DataModel.GetObjectsByName('Bolt Pretension')

    for prets in all_prets:
        times = range(1,prets.Parent.AnalysisSettings.NumberOfSteps+1)
        preload_vals=[]
        time_vals=[]
        for time in times:
            time_vals.append(Quantity(float(time), "sec"))
            preload_vals.append(Quantity(float(preload), "N"))

        prets.Preload.Inputs[0].DiscreteValues = time_vals
        prets.Preload.Output.DiscreteValues=preload_vals
```

Perform custom postprocessing within the simulation tree to execute with each simulation

Python Result Object

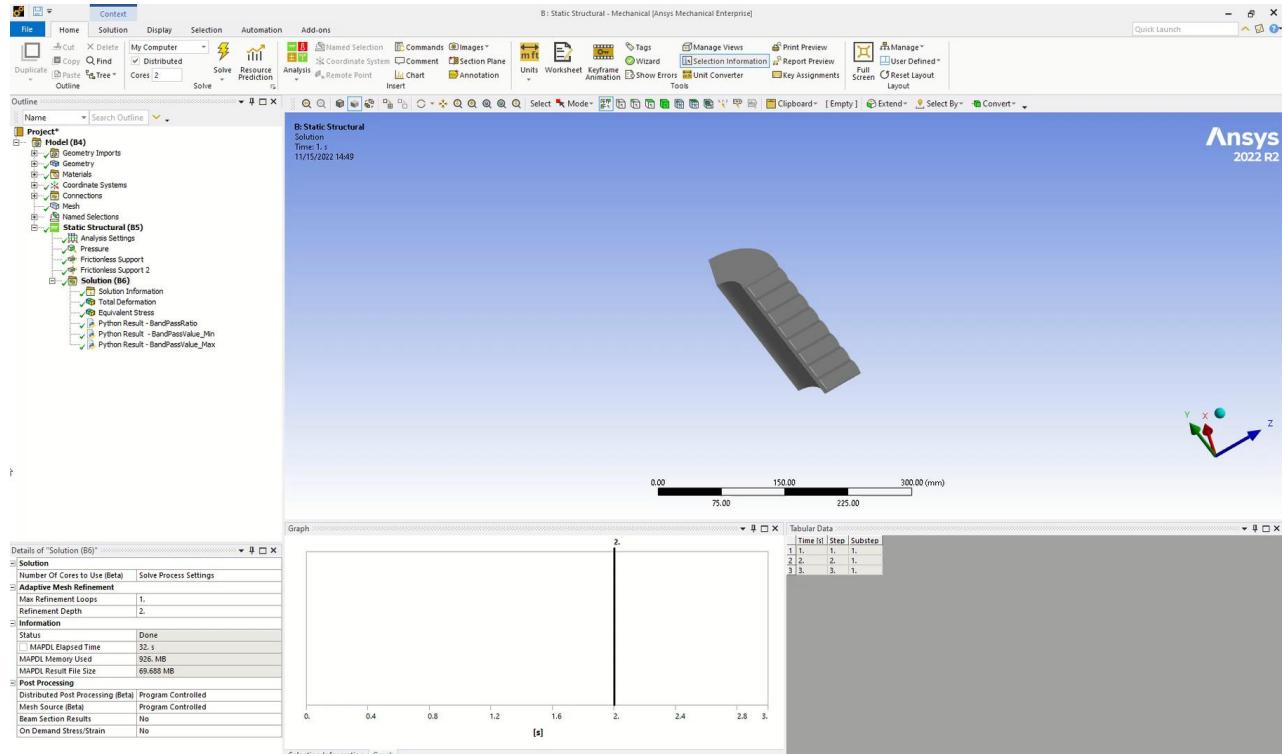
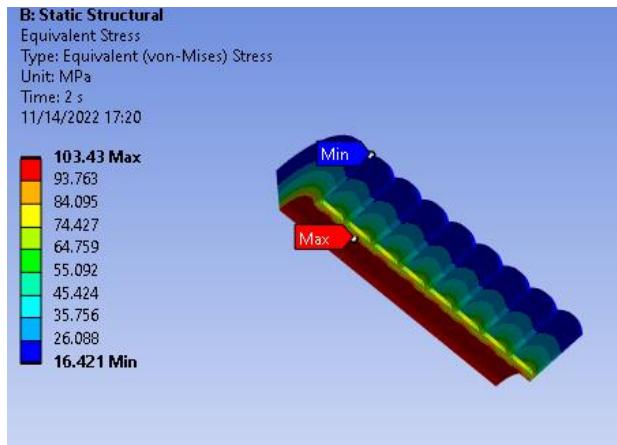


Details of "Python Result - BandPassValue_Max" :::::::	
Definition	
Suppressed	No
Connected	True
Advanced	
Script Execution Scope	<code>__python_code_89__</code>
Engine Type	Iron Python
Group 1	
<input type="checkbox"/> Stress Min Value	48

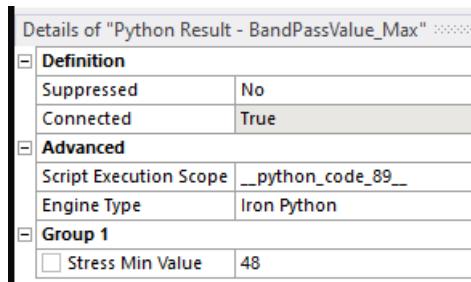
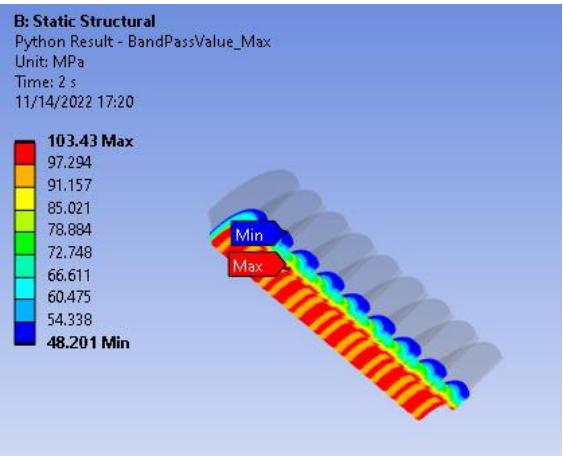
- Create custom postprocessing workflow
- Plot result as if it was a standard Mechanical item
- Add properties in the details view as needed (inputs, outputs, parameters)
- Based on DPF technology (Data Processing Framework)

Python Result example: stress threshold

Standard result: plot stress on complete model



Python result: plot stress above a specified value



Introduction to PyMechanical



Python scripting for Mechanical

1. Classic (IronPython)

- Run inside the scripting window (IronPython)

2. CLI (IronPython)

- Run mechanical from the command line, pass in input script as a command line argument
- Usable without PyMechanical, but easier to use with PyMechanical
- See [documentation](#) here.

3. Embedded instance (CPython)

- An instance of mechanical is embedded into the python process as an object using PyMechanical

4. Remote session (IronPython server, CPython client)

- Start a remote session of Mechanical in its own process
 - The remote session can be running on the local machine.
- Connect to that session from CPython using PyMechanical and send commands as IronPython strings or scripts

PyMechanical: access Mechanical through Python

install via:

```
pip install ansys-mechanical-core
```

Remote session

- Multiple programs:
 - Server = Mechanical
 - Client = Python (PyMechanical)
- Client can send commands to the server as strings, or as .py scripts
- Based on gRPC

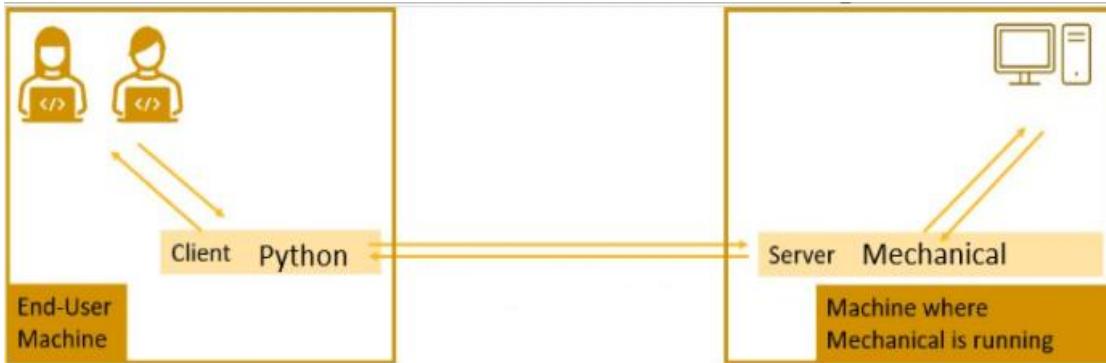
Embedded instance

- Python process only
- Mechanical application instance as a python object
- Use in the same way as Mechanical Scripting
- Based on pythonnet

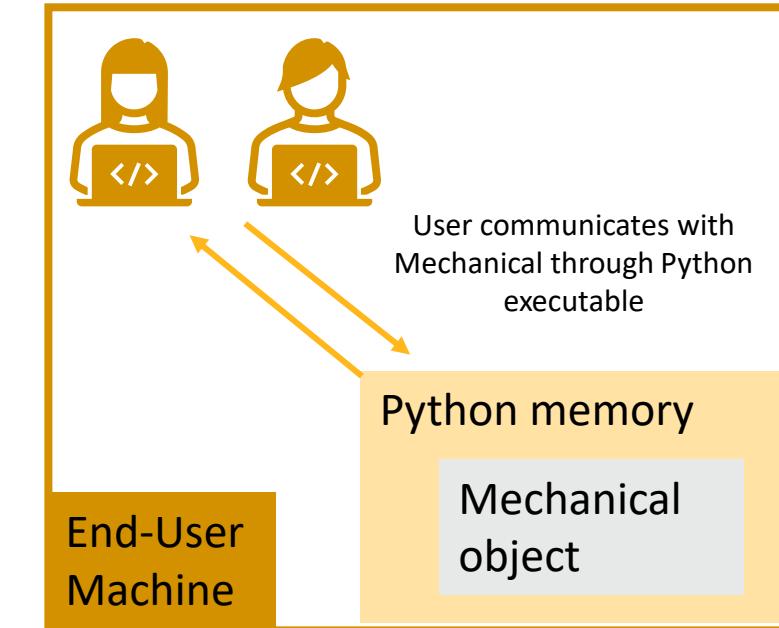
Go [here](#) for an explanation of the Mechanical application architecture and why both interfaces are offered

PyMechanical: access Mechanical through Python

Remote session



Embedded instance



Remote session

Launch a session locally:

```
import os
from ansys.mechanical.core import launch_mechanical

mechanical = launch_mechanical()
```

Interesting options:

- `launch_mechanical(batch = True)`: launches Mechanical in batch mode
- `launch_mechanical(batch = False)` launches a visible Mechanical session

Main methods:

<code>Mechanical.clear ()</code>	Clear the database.
<code>Mechanical.download (files[, target_dir, ...])</code>	Download files from the working directory of the Mechanical instance.
<code>Mechanical.download_project ([extensions, ...])</code>	Download all project files in the working directory of the Mechanical instance.
<code>Mechanical.exit ([force])</code>	Exit Mechanical.
<code>Mechanical.list_files ()</code>	List the files in the working directory of Mechanical.
<code>Mechanical.log_message (log_level, message)</code>	Log the message using the given log level.
<code>Mechanical.project_directory</code>	Get the project directory for the currently connected Mechanical instance.
<code>Mechanical.run_python_script (script_block[, ...])</code>	Run a Python script block inside Mechanical.
<code>Mechanical.run_python_script_from_file (file_path)</code>	Run a Python file inside Mechanical.
<code>Mechanical.upload (file_name[, ...])</code>	Upload a file to the Mechanical instance.
<code>Mechanical.version</code>	Get the Mechanical version based on the instance.

Embedded instance

Launch an embedded instance:

```
from ansys.mechanical.core import App  
  
app = App()  
ns = app.DataModel.Project.Model.AddNamedSelection()
```

Entry points (same as in-product scripting):

ExtAPI: `Application.ExtAPI`
DataModel: `Application.DataModel`
Model: `Application.DataModel.Project.Model`
Tree: `Application.DataModel.Tree`
Graphics: `Application.ExtAPI.Graphics`

Useful tip: import additional entry points, namespaces, and types (Quantity, DataModelObjectCategory, etc.):

```
from ansys.mechanical.core import App, global_variables  
  
app = App()  
# The following line extracts the global API entry points and merges them into your global  
# Python global variables.  
globals().update(global_variables(app))
```

PyMechanical demo for embedded and remote versions

This [example](#) illustrates how to setup the same model, once with embedded instance and once with remote version.

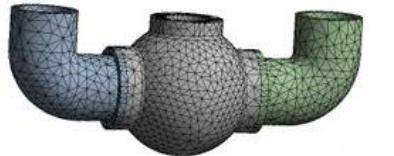
Embedded instance

```
import os
from ansys.mechanical.core import App, global_variables

app = App(version=232)
globals().update(global_variables(app)) # update global variables

# Add static analysis
analysis = Model.AddStaticStructuralAnalysis()
```

Type commands as if in the Mechanical console

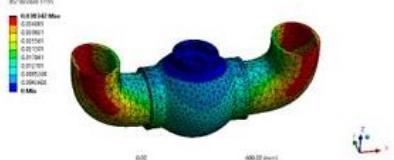
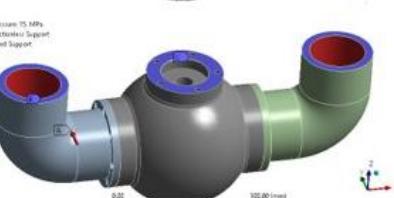


Remote session

```
import os
from ansys.mechanical.core import launch_mechanical

mechanical = launch_mechanical(batch=True, loglevel="DEBUG")
print(mechanical)
# Check working directory
server_project_directory = mechanical.run_python_script("ExtAPI.DataMode")
print(server_project_directory)
# Run mechanical automation script
mechanical_script = os.path.join(work_dir, 'mech_script.py')
result = mechanical.run_python_script_from_file(mechanical_script)
```

Send Mechanical commands as strings or as second Python script



- Import CAD geometry
- Define material assignment
- Define mesh settings
- Define boundary conditions
- Solve
- Post-process
- Export results

Input files: mech_script.py, pymechanical_demo_remote.py, pymechanical_demo_embedded.py, Valve.scdoc

What's coming

- 3D visualization (basic support in version 2024 R2)
- Improved client-server design
- Autocomplete for vscode
- Material modeling API



PyMechanical: Live Demo



PyMechanical: Workshop Challenge

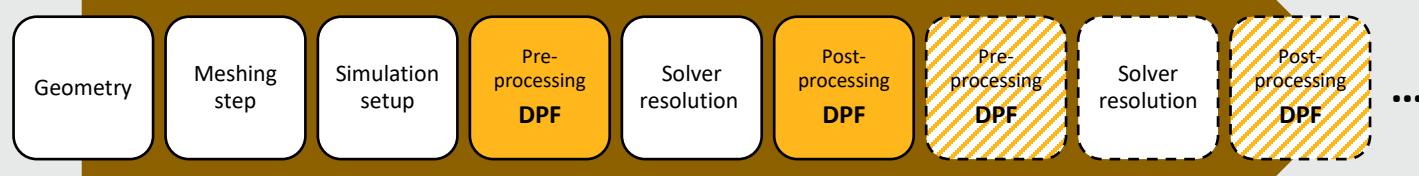
Introduction to PyDPF



Ansys Data Processing Framework (DPF)

Pre & post-processing of simulation data in a modular way

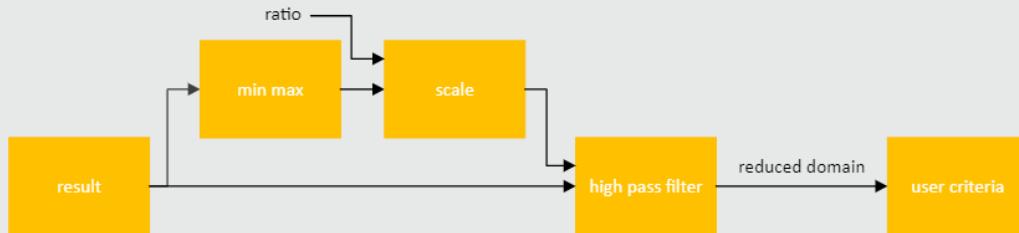
Transforming data, in the simulation process...



... with reusable operators from domain libraries...



... assembled as dedicated workflows.

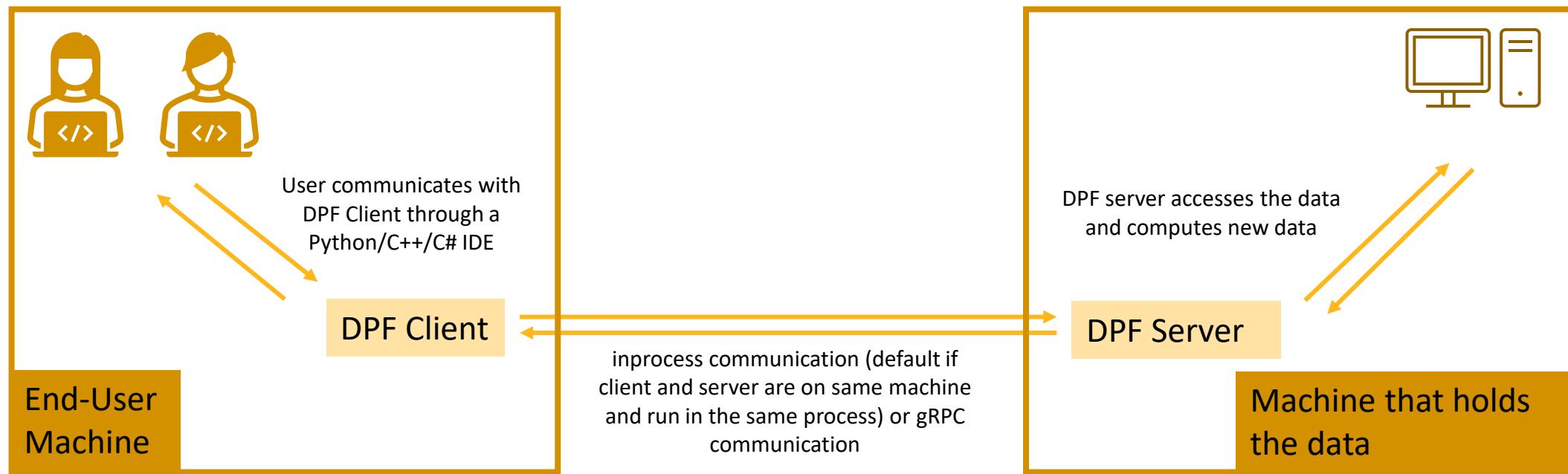


DPF provides numerical **simulation engineers** and **solution developers** with a toolbox for accessing and transforming large amounts of simulation data. DPF is targeted at internal Ansys BU and ACE or external partners and customers.

DPF structures and makes reusable, complex pre & post processing of data for simulation, cross-teams & cross-products sharing, and application specific tool development.

Client/Server architecture

DPF is based on a Client/Server architecture. Client and Server can be on the same machine, or on separate machines.



Where and how to find install files?

DPF Client

- Python packages are usually installed through “pip”
- To install PyDPF-Core and PyDPF-Post from a Python console:

```
pip install ansys-dpf-core
```

```
pip install ansys-dpf-post
```

- It is also possible to download the wheels from Github ([PyDPF-Core](#), [PyDPF-Post](#)) or PyPi ([PyDPF-Core](#), [PyDPF-Post](#))

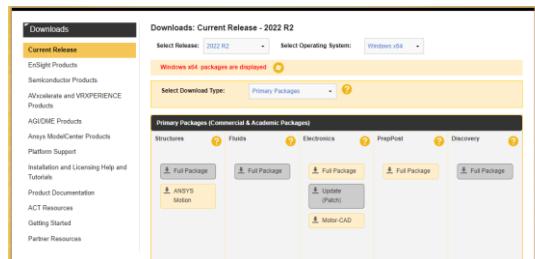
DPF Server

Ansys Unified Installer

- DPF Server is packaged within the standard Ansys installation files
- To download the standard installation files, use your preferred distribution channel, for example the [Ansys Customer Portal](#)

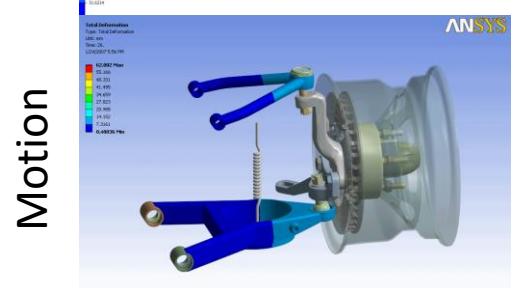
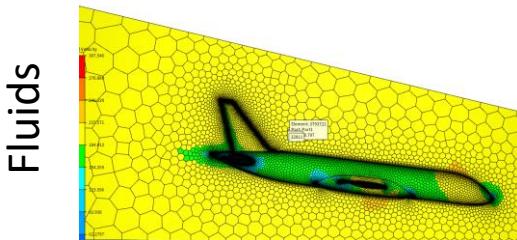
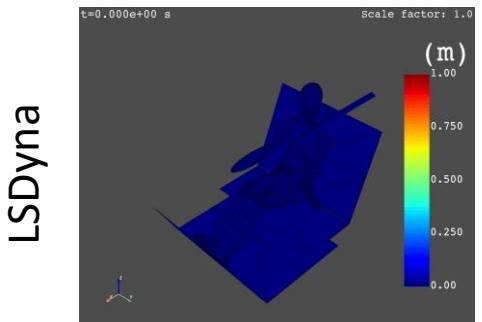
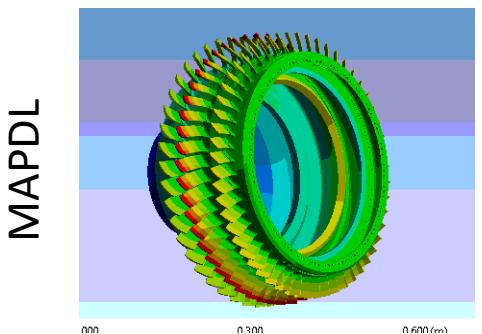
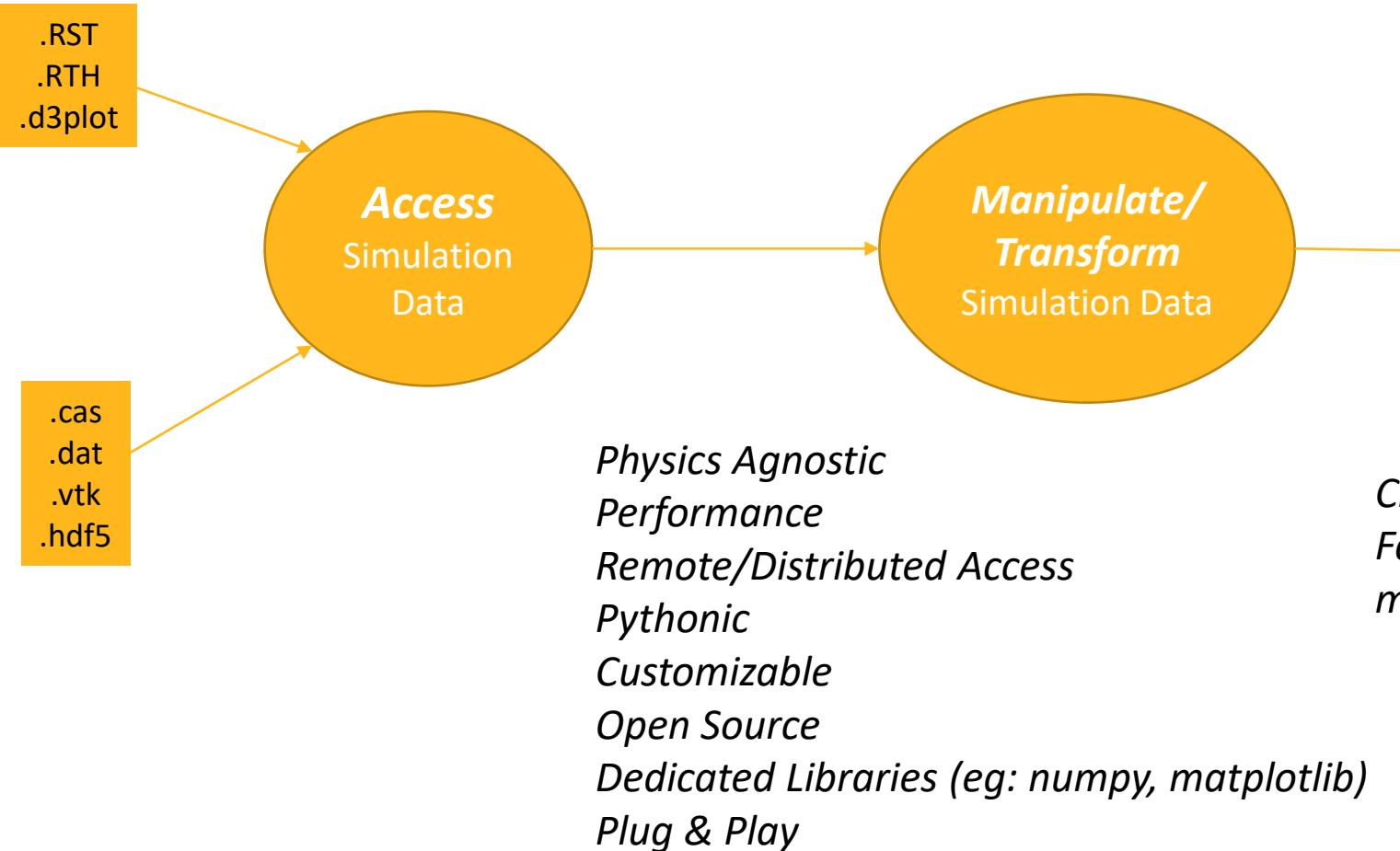
Standalone package

- DPF Server is also available as a standalone package **for pre-release access only**.
- This package can be downloaded from the [DPF Pre-Release page of the Ansys Customer Portal](#)



For further guidance, please refer to the help ([PyDPF-Core](#), [PyDPF-Post](#))

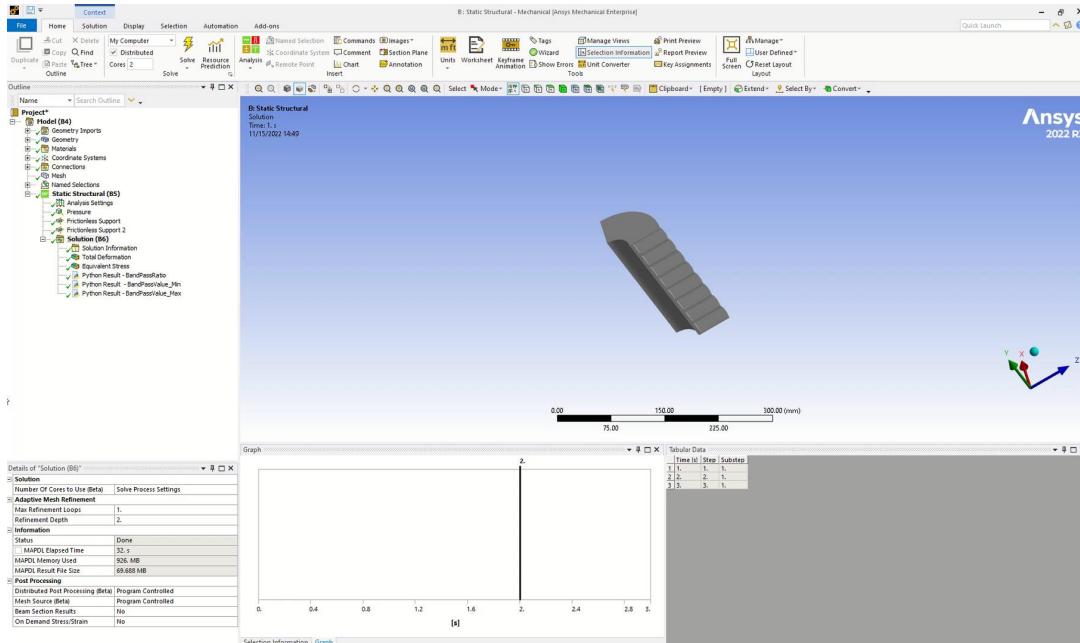
DPF: automate and customize post-processing



Where can DPF be used ?

In Mechanical

Scripting console, Python Code object, **Python Result object**

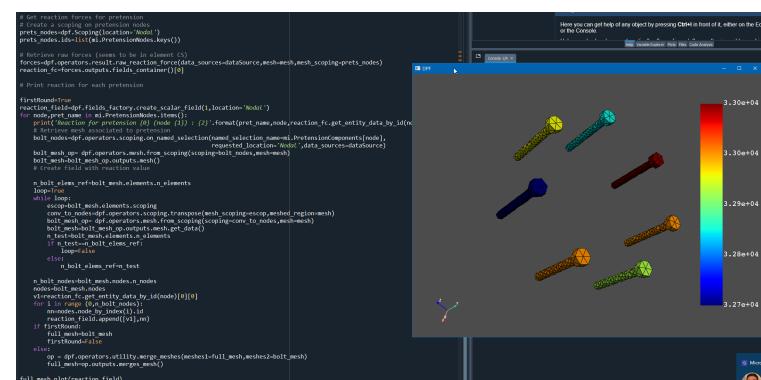


With PyAnsys

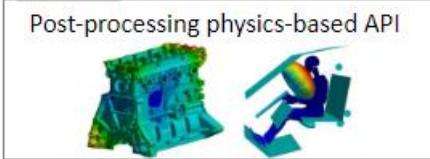
Generic client API



Highly customizable, physics agnostic workflow-based API



Dedicated libraries

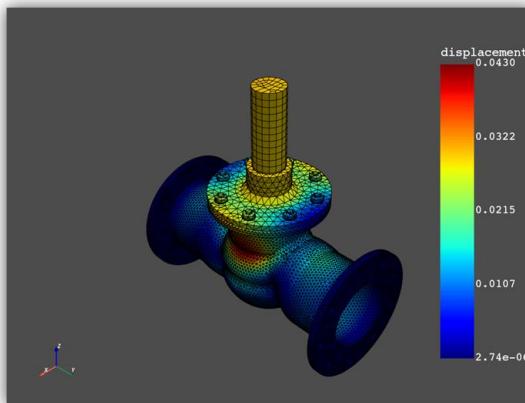
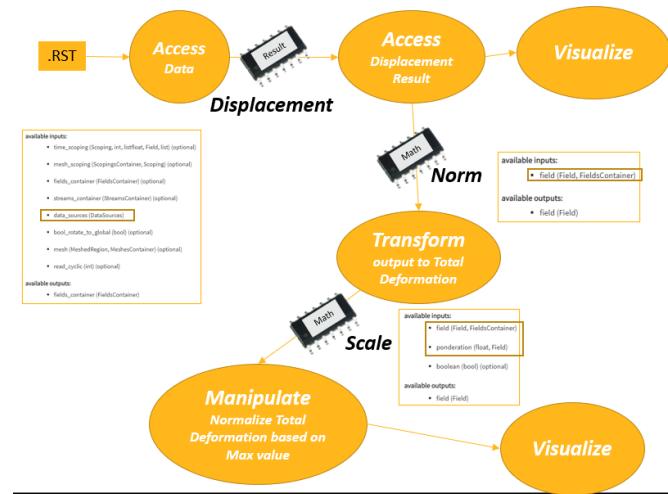


Using DPF in PyAnsys

- PyAnsys project: offers two Python packages based on DPF:
 - PyDPF-Core: Complete exposure of DPF technology to read and transform FE data with powerful operators.
 - PyDPF-Post: Streamlined post-processing API for extracting standard results. Uses PyDPF-Core behind the scenes.

DPF Core

Exposure of complete DPF technology: chain operators together to create a customized postprocessing workflow



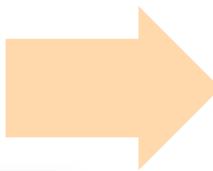
DPF Post

Simple wrapper functions to main DPF capabilities



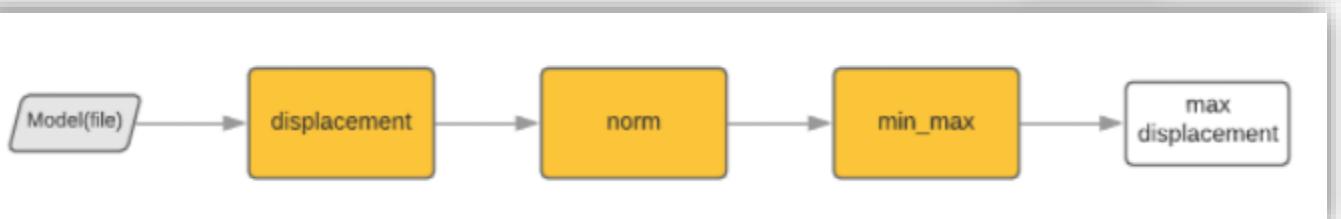
PyDPF-Core

Run simulation in ANSYS and save binary result file (.rst)



Use PyAnsys to:

- Manipulate and create new data easily thanks to DPF-Core



DPF can access data from **solver result files** as well as several neutral formats (**csv, hdf5, vtk**, etc.).

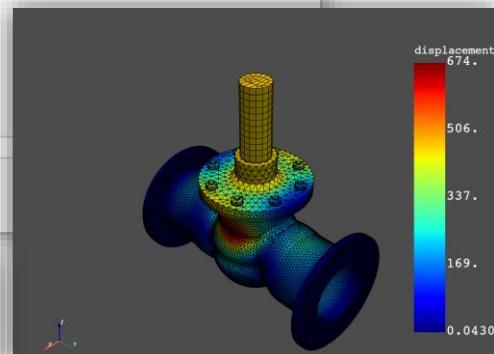
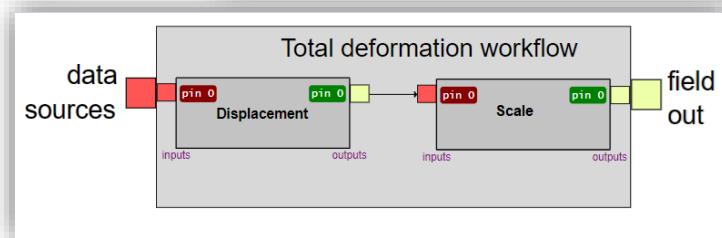
```
## Use DPF core to manipulate data
from ansys.dpf import core

model = core.Model(file_name)
U = model.results.displacement()
field_container = U.outputs.fields_container()
print(field_container[0].data)

op = core.Operator('scale') # operator instantiation
op.inputs.field.connect(fieldContainer[0])
scale_value = float(max_total_deformation/min_total_deformation)
op.inputs.ponderation.connect(scale_value)

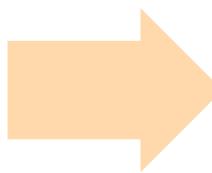
my_field = op.outputs.field()
print(my_field)
print(my_field.data)

mesh = model.metadata.meshed_region
mesh.plot(my_field)
```



PyDPF-Post

Run simulation in ANSYS and save binary result file (.rst)

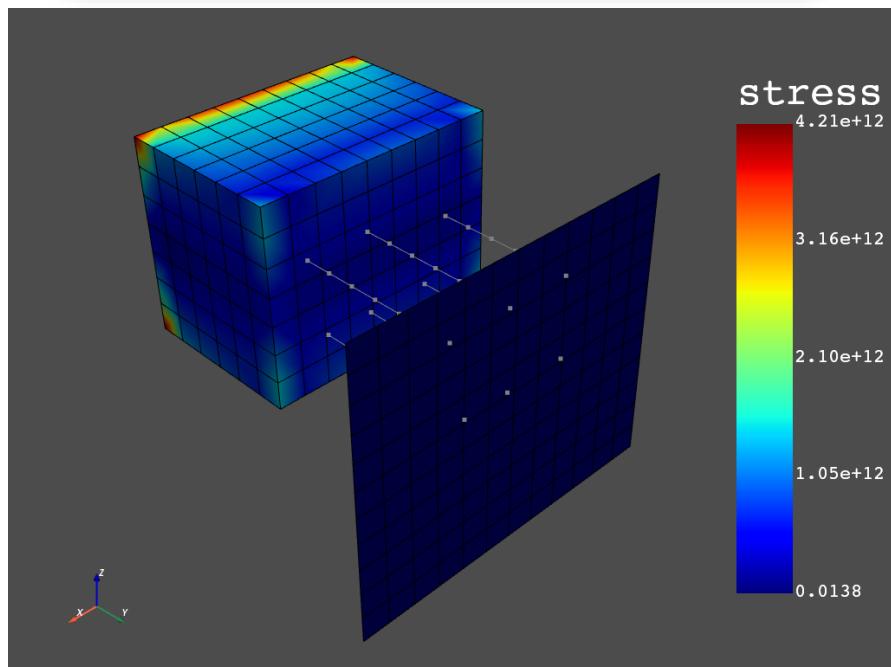


Use PyAnsys to:

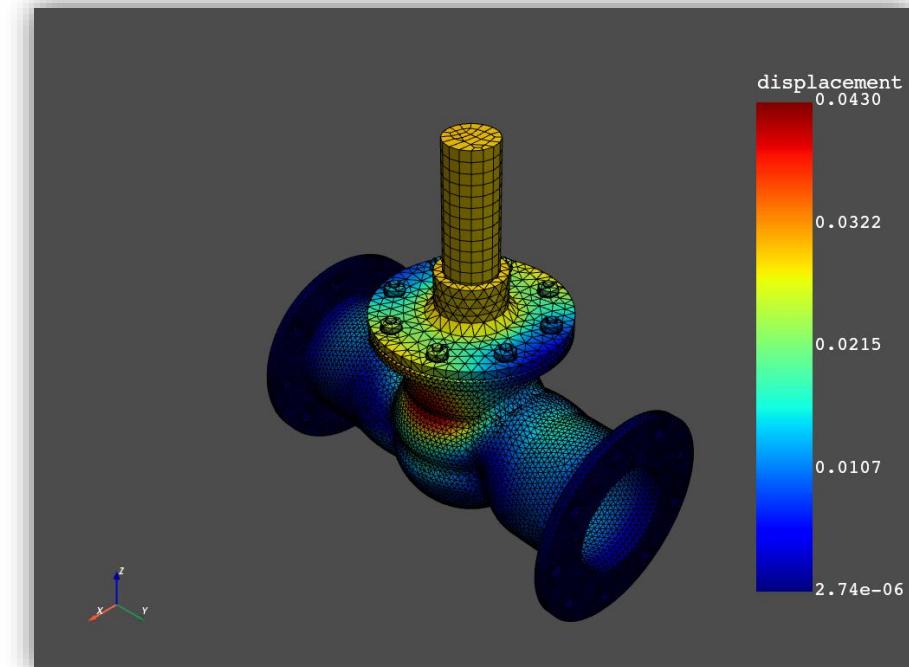
- Plot data in .rst file thanks to DPF-Post

Plot the amplitude contour

```
amplitude = stress_result.tensor_amplitude  
stress.plot_contour()
```



```
# Get and plot total deformation  
displacement = solution.displacement()  
total_deformation = displacement.norm  
total_deformation.plot_contour(show_edges=True)
```



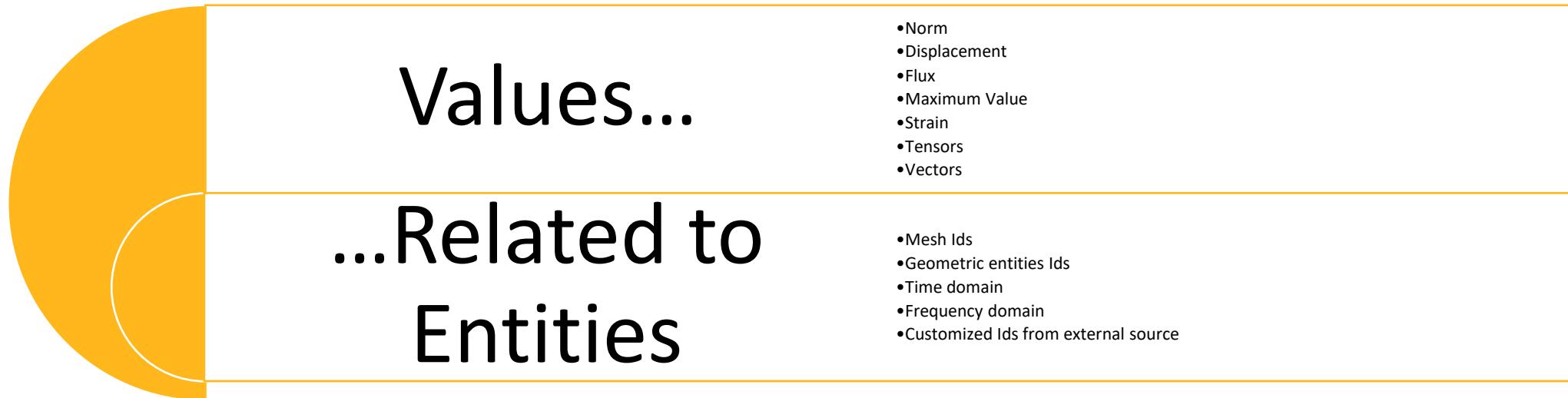
DPF: Key Concepts

PyDPF-Core: Key Concepts

- **Data source:** One or more files containing analysis results.
- **Field:** Main simulation data container.
- **Field container:** For a transient, harmonic, modal, or multi-step static analysis, a set of fields, with one field for each time step or frequency.
- **Location:** Type of topology associated with the data container. DPF uses three different spatial locations for finite element data: `Nodal`, `Elemental`, and `ElementalNodal`.
- **Operators:** Objects that are used to create, transform, and stream the data. An operator is composed of a **core** and **pins**. The core handles the calculation, and the pins provide input data to and output data from the operator.
- **Scoping:** Spatial and/or temporal subset of a model's support.
- **Support:** Physical entity that the field is associated with. For example, the support can be a mesh, geometrical entity, or time or frequency values.
- **Workflow:** Global entity that is used to evaluate the data produced by chained operators.
- **Meshed region:** Entity describing a mesh. Node and element scopings, element types, connectivity (list of node indices composing each element) and node coordinates are the fundamental entities composing the meshed region.

Simulation Data

- First step consists of defining the simulation data.
- Simulation data are collections of arrays of values with their metadata;
- Basic definition of simulation data is



Simulation Data

- Let's take a data set example.
- It is necessary to have the full knowledge of what the data correspond to, since there might be multiple ways to interpret the data set:

Entities Ids

File	Edit	Format	View	Help
1	1.32E+08	-1.13E+07	-2.26E+06	
2	6.41E+07	-6.10E+06	-1.29E+06	
3	3.65E+07	-1.16E+07	-2.39E+06	
4	2.00E+07	-2.43E+07	-1.71E+06	
5	1.18E+07	-4.50E+07	-1.44E+06	
6	8.10E+06	-7.05E+07	-1.42E+06	
7	5.97E+06	-9.81E+07	-1.93E+06	
8	5.20E+06	-1.27E+08	-1.96E+06	
9	2.00E+06	-1.60E+08	-6.26E+06	
10	9.44E+06	-1.92E+08	1.66E+06	
11	-3.56E+07	-2.74E+08	-6.56E+07	
12	1.43E+08	-3.92E+06	6.40E+06	
13	9.98E+07	-1.70E+07	-56967	
14	7.09E+07	-2.49E+07	-9.52E+05	
15	5.08E+07	-3.45E+07	-1.15E+06	
16	3.68E+07	-4.75E+07	-1.32E+06	
17	2.72E+07	-6.42E+07	-1.52E+06	

3 Sets of values related to entities Ids
Per set of data: 1 scalar value is associated to 1 entity

Entities Ids

File	Edit	Format	View	Help
1	1.32E+08	-1.13E+07	-2.26E+06	
2	6.41E+07	-6.10E+06	-1.29E+06	
3	3.65E+07	-1.16E+07	-2.39E+06	
4	2.00E+07	-2.43E+07	-1.71E+06	
5	1.18E+07	-4.50E+07	-1.44E+06	
6	8.10E+06	-7.05E+07	-1.42E+06	
7	5.97E+06	-9.81E+07	-1.93E+06	
8	5.20E+06	-1.27E+08	-1.96E+06	
9	2.00E+06	-1.60E+08	-6.26E+06	
10	9.44E+06	-1.92E+08	1.66E+06	
11	-3.56E+07	-2.74E+08	-6.56E+07	
12	1.43E+08	-3.92E+06	6.40E+06	
13	9.98E+07	-1.70E+07	-56967	
14	7.09E+07	-2.49E+07	-9.52E+05	
15	5.08E+07	-3.45E+07	-1.15E+06	
16	3.68E+07	-4.75E+07	-1.32E+06	
17	2.72E+07	-6.42E+07	-1.52E+06	

3-component vector

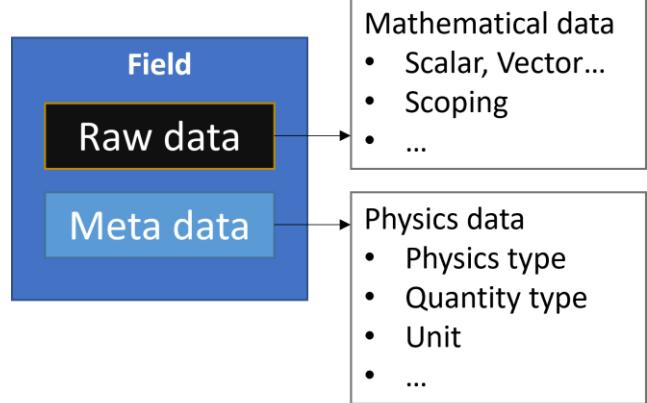
1 Set of values related to entities Ids
1 vector composed of 3 components is associated to 1 entity

Simulation Data in DPF: Fields & Scopings

- Let's introduce DPF language:
 - Values are stored in **Fields**
 - Values are associated to Entities
 - It is often necessary to define a subset of entities. Usually, you will not be working with the entire field
 - The subset of Entities is defined using **Scoping**
- Fields** can be seen as lists of values, and **Scoping** like a subset of a list
- Scoping** can be any type, for example, a subset of:
 - Mesh Ids
 - Geometric entities Ids
 - Time domain
 - Frequency domain

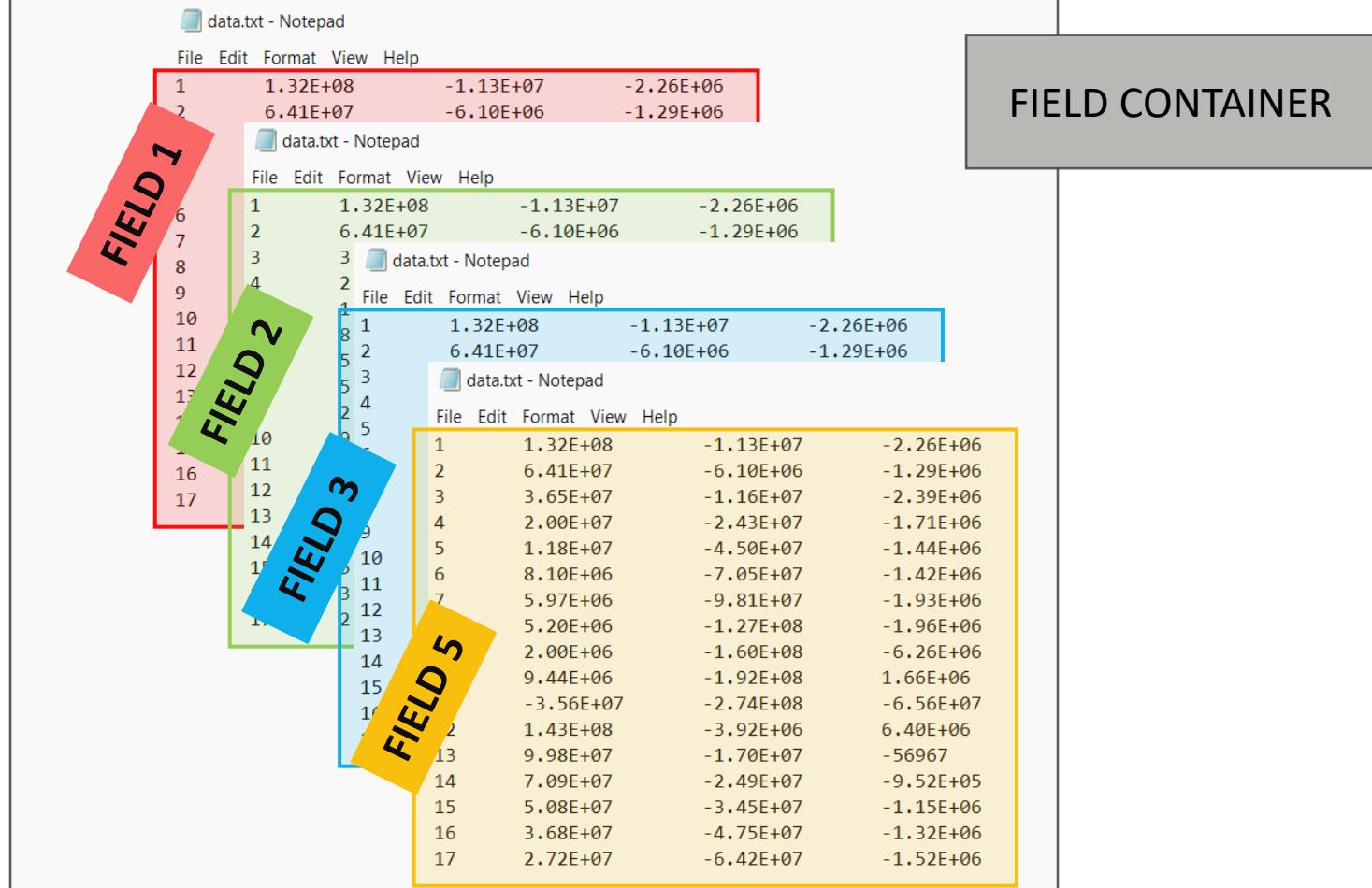
	data.txt - Notepad			
	File	Edit	Format	View
1	1.32E+08	-1.13E+07	-2.26E+06	
2	6.41E+07	-6.10E+06	-1.29E+06	
3	3.65E+07	-1.16E+07	-2.39E+06	
4	2.00E+07	-2.43E+07	-1.71E+06	
5	1.1		-1.44E+06	
6	8.1		-1.42E+06	
7	5.9		-1.93E+06	
8	5.2		-1.96E+06	
9	2.0		-6.26E+06	
10	9.4		1.66E+06	
11	-3.		6.56E+07	
12	1.43E+08	-3.92E+06	6.40E+06	
13	0.08E+07	-1.70E+07	5.60E+07	
14	7.09E+07	-2.49E+07	-9.52E+05	
15	5.08E+07	-3.45E+07	-1.15E+06	
16	3.68E+07	-4.75E+07	-1.32E+06	
17	2.72E+07	-6.42E+07	-1.52E+06	

Working with only this set of values, need to specify a **scoping**: subset of entities from 14 to 17



Fields Container

- The Fields Container is a vector of Fields and all the Fields are ordered with labels and ids. Most commonly, the Fields Container is scoped on "time" label and the ids are the time or frequency sets



Data Sources and Streams

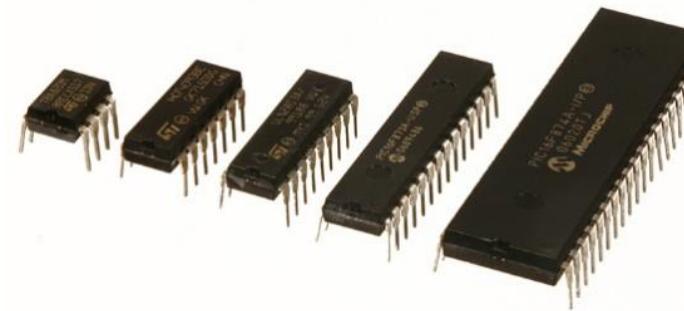
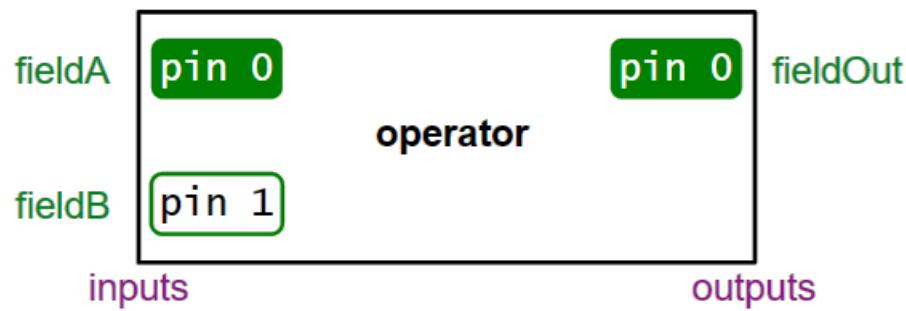
- The **Data Sources** is a container of files on which the analysis results can be found
 - It usually consists of path to access results or data files.
- **Streams** is an entity containing the data sources.
 - Defining **Streams** opens the data files. They stay opened and they keep some data in cache to make the next evaluation faster.
 - Using Streams is convenient when using large files. It saves time when opening/closing files.
 - The streams should be released to close the files.
- **Streams** are associated to a **DataSource**

Transform Data: Operators



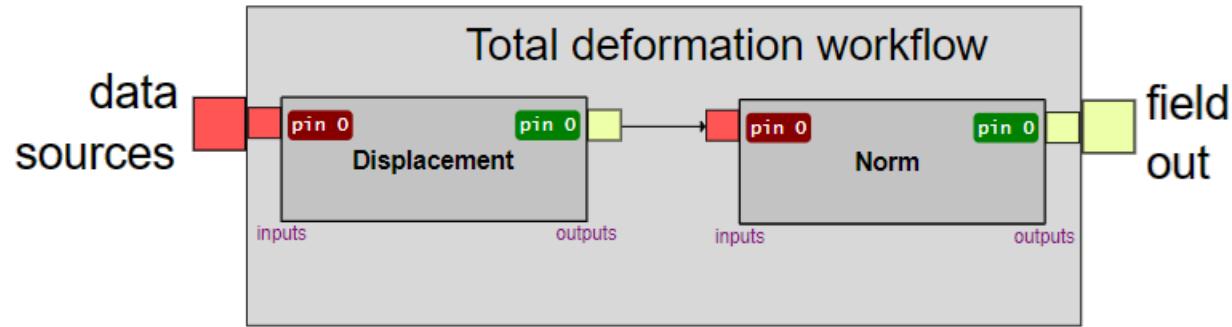
Operators

- The Operator is the only object used to **create and transform the data**.
- It can be seen as an integrated circuit in electronics with a range of pins in input and in output.
- Those pins allow the user to pass on his information to each operator. When the operator is evaluated, it will process the input information to compute its output with respect to its description.



Operators

- Operators can be chained together to create workflows. To do so, the user only needs to connect some operator's outputs to another operator's inputs.



- With workflows, lazy evaluation is performed, which means that when the last operator's outputs are asked by the user, all the connected operators will also be evaluated (and not before) to compute a given result.
- Operator is responsible for checking that is connected to the pin is valid;

Operators

Operators can be used to: import, export, transform and analyze data.

result

- acceleration
- acceleration X
- acceleration Y
- acceleration Z
- accu eqv creep strain
- accu eqv plastic strain
- add rigid body motion (field)
- add rigid body motion (fields container)
- artificial hourglass energy
- cms matrices provider
- co-energy
- contact fluid penetration pressure
- contact friction stress
- contact gap distance
- contact penetration
- contact pressure
- contact sliding distance
- contact status
- contact surface heat flux
- contact total stress

min_max

- incremental over field
- incremental over fields container
- max by component
- max by component
- max over phase
- max over time
- min max by entity
- min max by entity over time
- min max over time
- min over time
- over field
- over fields container
- over label
- phase of max
- time of max
- time of min

utility

- bind support
- bind support (fields container)
- change location
- change shell layers
- convert to field
- convert to fields container
- extract field
- forward
- forward field
- forward fields container
- forward meshes container
- html doc
- python generator

Operators

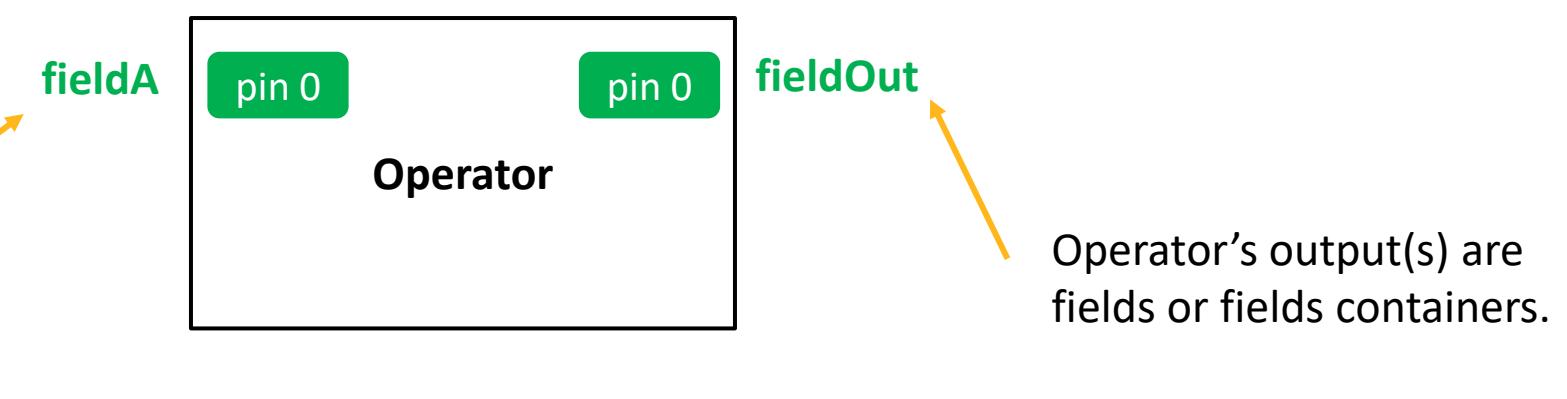
- Using an operator requires to have a look at the help documentation. The way to define the operator is operators' dependent
- Operator definition consists of 3 steps:
 - Operator Initiation
 - Input definition
 - Output storage

Operator's input(s) are usually field(s) or fields container(s).

They can also be scopings

They should be connected to the operator input pin using

That is why the simulation data (fields / fields containers) should be defined prior to defining operators.



Operators

- Example: let's look at Total Sum operators (Math category)

math: total sum

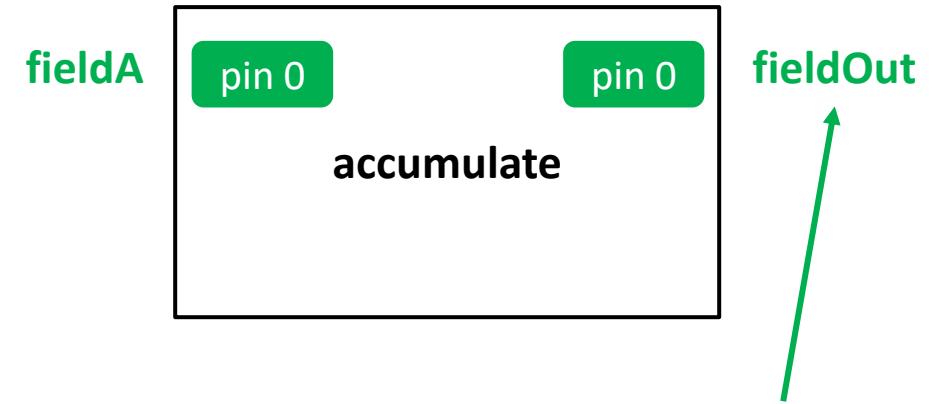
Description
Sum all the elementary data of a field to get one elementary data at the end.

Inputs
`pin 0 fieldA (field | fields_container) : field or fields container with only one field is expected`

Outputs
`pin 0 field (field)`

Configurations
`mutex (bool) false`

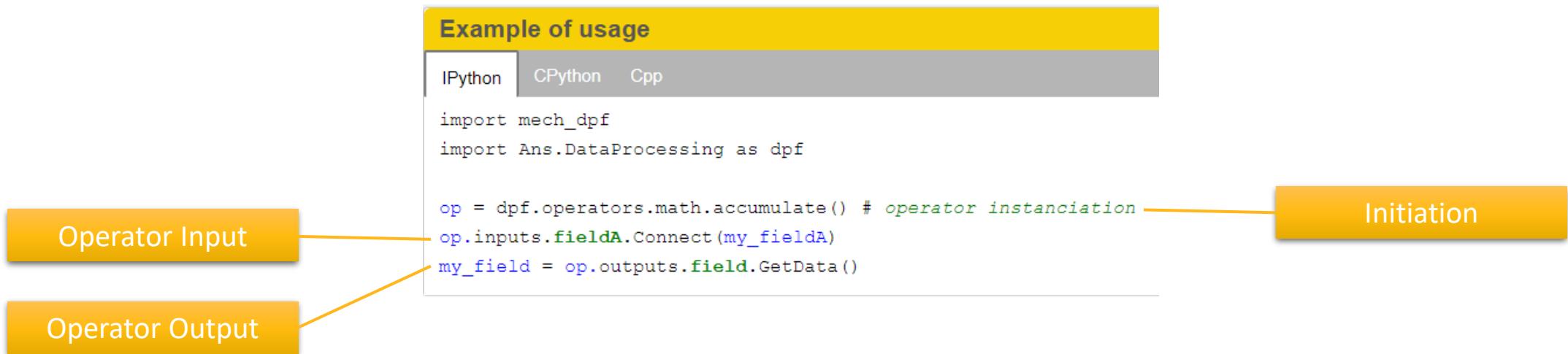
Scripting
`category: math`
`plugin: core`
`scripting name: accumulate`



fieldOut is a scalar field containing the sum of all values from input field

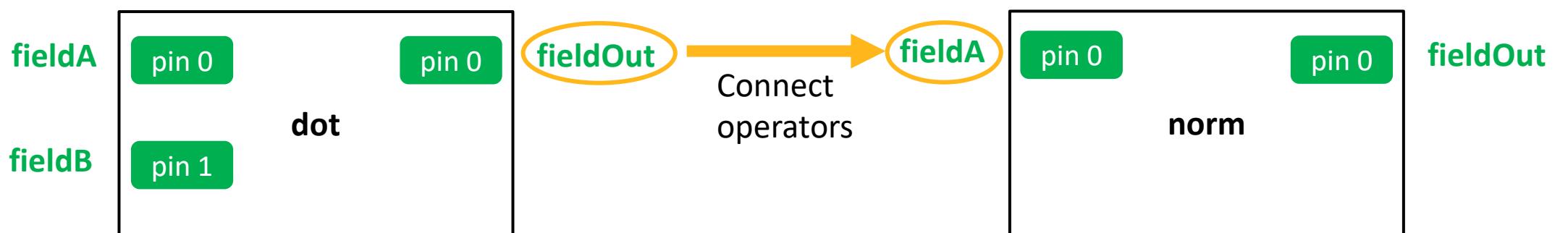
Operators

- The example of usage, whether IPython or Cpython, will help you to define the operator



Chaining Operators: Workflow Creation

- In most of the cases, the use of one operator will not be sufficient to bring the desired result as output.
- The use of multiple operators, acting like a workflow will be necessary in this case to get the desired result
- Workflows are created by connecting operators: output pins of one operator are connected to input pins of another operator
- Example: get the norm of a resulting vector from the dot product of two vectors

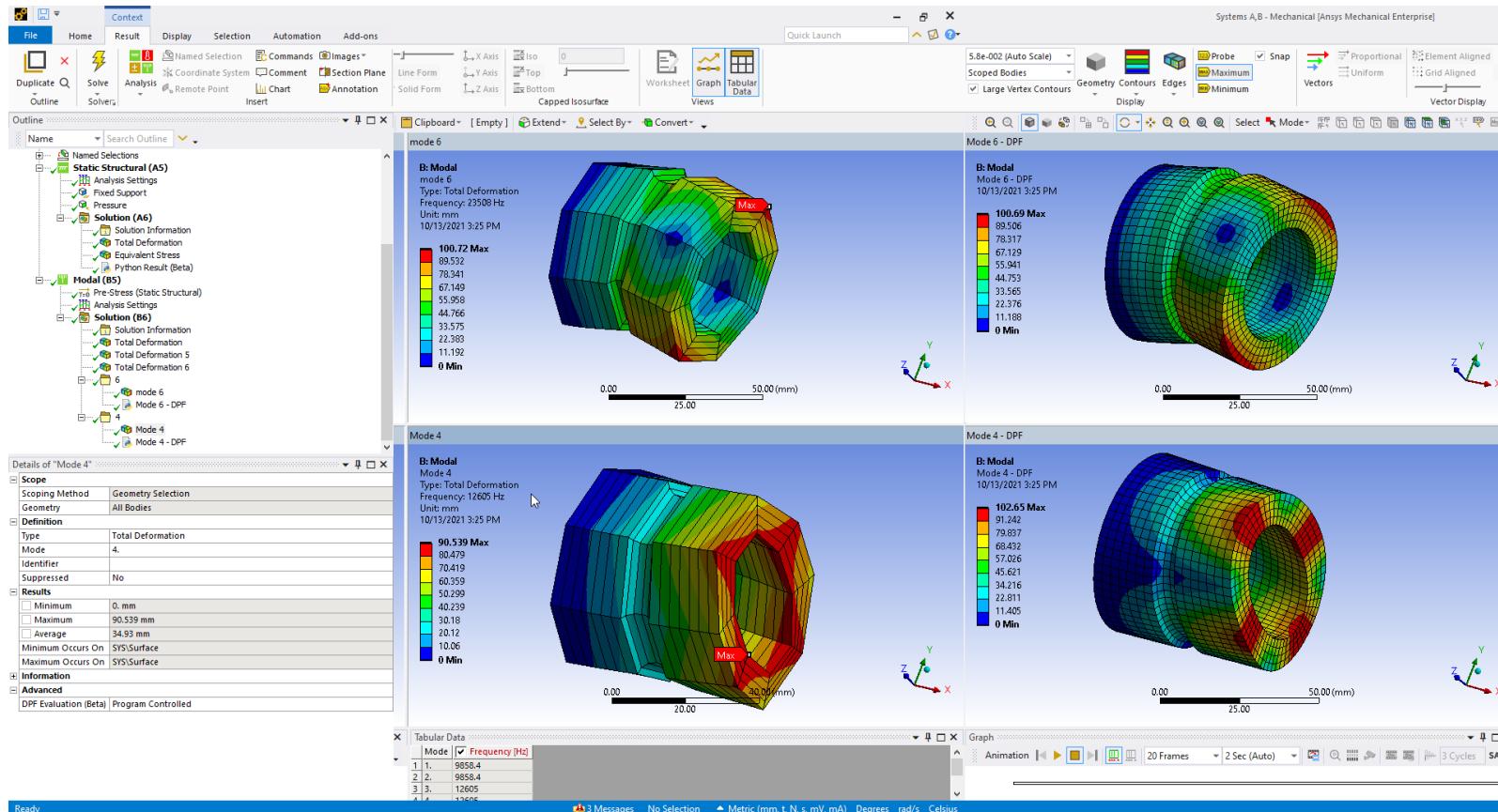


Use-Case: Exporting transient results to a CSV file – DPF in Mechanical

The screenshot displays the ANSYS Mechanical interface with several windows open:

- Graph**: Shows two plots: one for Temperature (green line) and another for Total Heat Flux (blue line). A large orange arrow points from the Graph window down towards the CSV export table.
- Mechanical Scripting**: An editor window titled "New Script 2 : Description" containing Python code for exporting DPF data to CSV. The code imports necessary modules, retrieves the solution object, and defines a temperature operator to export data to "exported_disp.csv".
- Tabular Data**: A table showing transient results for Temperature [C] over time [s]. The data includes rows for time steps 1 to 21, with values ranging from 21.996 to 106.18.
- CSV File Content**: A table showing the contents of the "exported_disp.csv" file, which contains the same transient temperature data as the Tabular Data table.

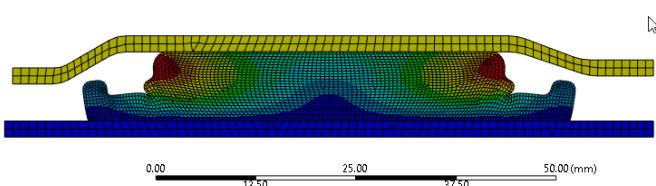
Use-Case: Expanding generalized Axisymmetric results



Standard results

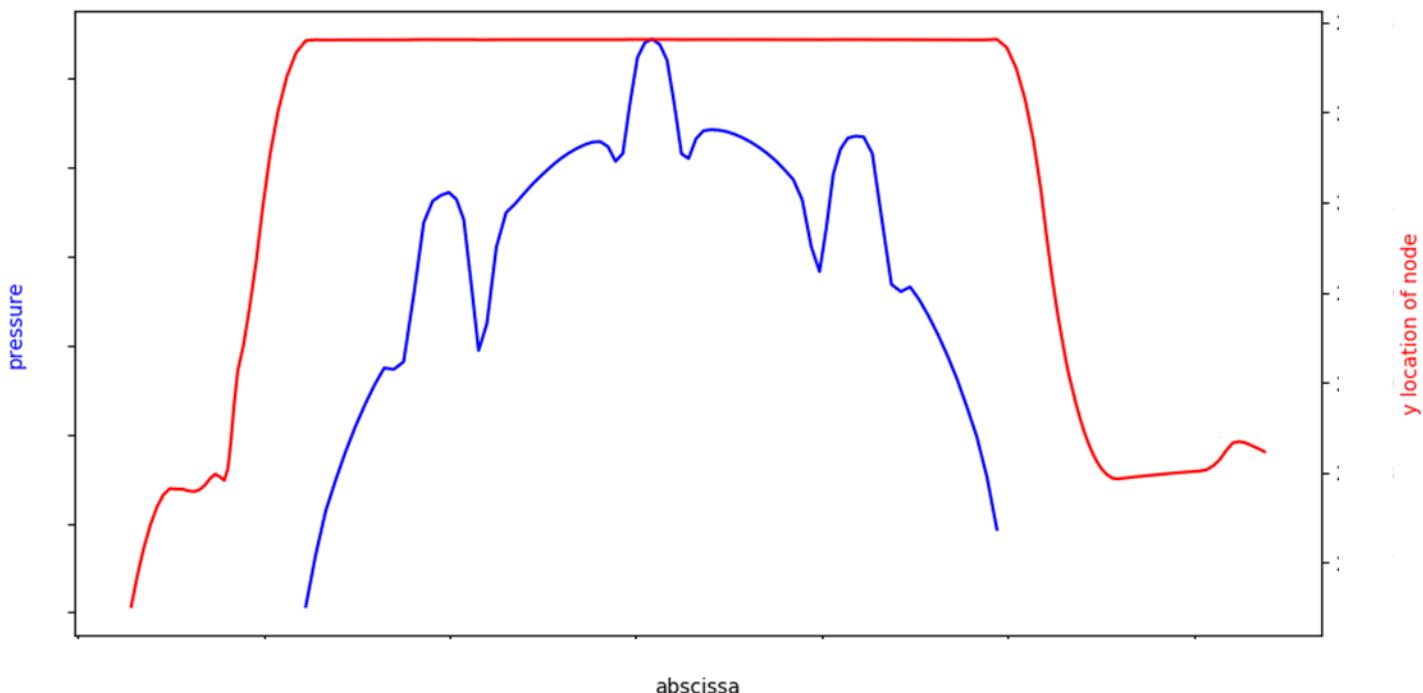
Expanded results thanks to
Mechanical DPF

Use-Case: Extracting a pressure profile



```
....# Compute curvilinear abscissa
....curv_abs=getCurvilinearAbcissa(node_coords)
.....
....# Export pressure vs. abscissa
....myPath=os.path.join(userFilesDir,'pressure_on_path_{0}.txt'.format(dp))
....with open(myPath,'w') as f:
....    f.write('Abscissa\tPressure\n')
....    for ca in curv_abs:
....        f.write('{0:12.6e}\t{1:12.6e}\n'.format(ca[1],value_by_node[ca[0]]))
....    # In same file, export node profile after deformation
....    f.write('$\n')
....    for nid,coord in node_coords.items():
....        f.write('{0:12.6e}\t{1:12.6e}\n'.format(coord[0],coord[1]))
.....
....# Create plot using CPython code
....pngPath=os.path.join(userFilesDir,'pressure_on_path_{0}.png'.format(dp))
.....
....c_python_script='''...
import os
import matplotlib.pyplot as plt
with open(r''''+myPath+'''',"r") as f:
    x_for_plot=[]
    y_for_plot=[]
    ...
    x_profile=[]
    y_profile=[]
    line=f.readline()
    line=f.readline()
    while not('$' in line):
        sp_line=line.strip("\n").split("\t")
        x_for_plot.append(float(sp_line[0]))
        ...
        x_for_plot.append(float(sp_line[1]))'''')
```

Pressure variation for nodes in contact with upper plate along with deformation profile



Use-Case: Extracting bolt reactions from a RST file - pyDPF

```
# Get reaction forces for pretension
# Create a scoping on pretension nodes
prets_nodes=dfpf.Scoping(location='Nodal')
prets_nodes.ids=list(mi.PretensionNodes.keys())

# Retrieve raw forces (seems to be in element CS)
forces=dfpf.operators.result.raw_reaction_force(data_sources=dataSource,mesh=mesh,mesh_scoping=prets_nodes)
reaction_fc=forces.outputs.fields_container()[0]

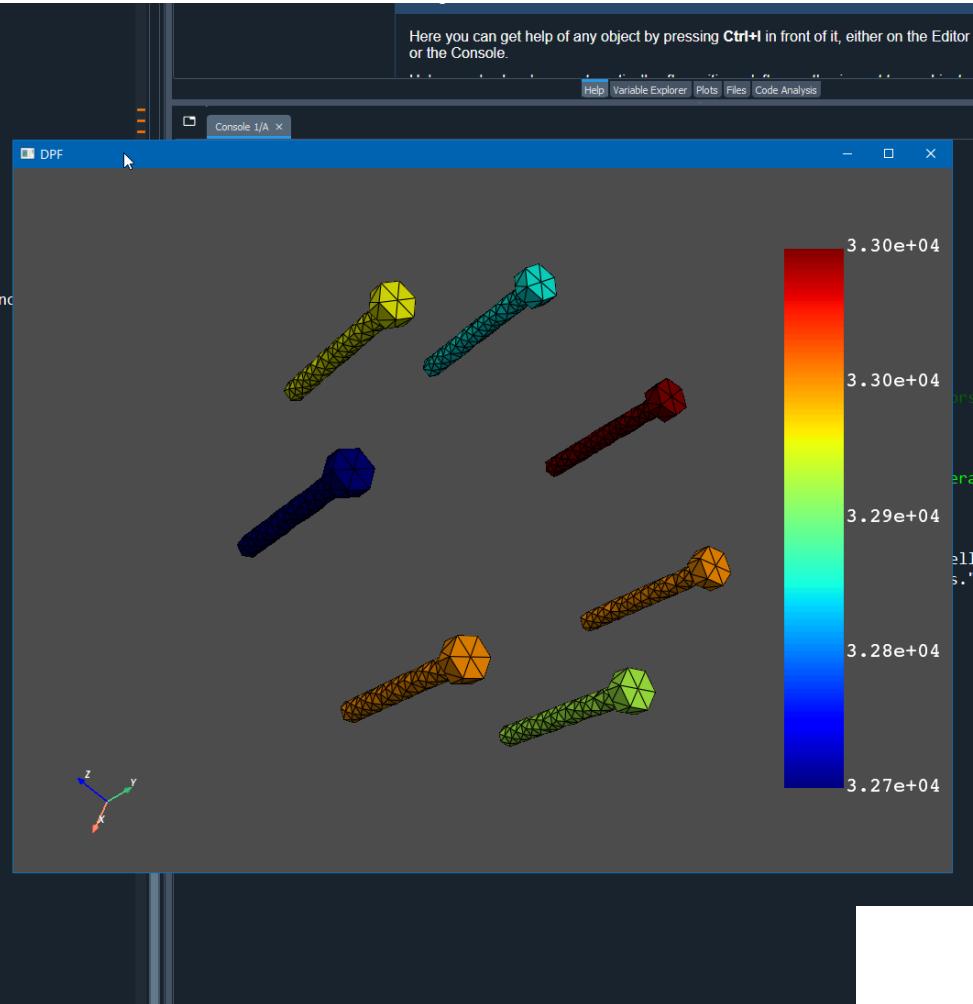
# Print reaction for each pretension

firstRound=True
reaction_field=dfpf.fields_factory.create_scalar_field(1,location='Nodal')
for node,pret_name in mi.PretensionNodes.items():
    print('Reaction for pretension {0} (node {1}) : {2}'.format(pret_name,node,reaction_fc.get_entity_data_by_id(node)))
    # Retrieve mesh associated to pretension
    bolt_nodes=dfpf.operators.scoping.on_named_selection(named_selection_name=mi.PretensionComponents[node],
                                                          requested_location='Nodal',data_sources=dataSource)
    bolt_mesh_op= dfpf.operators.mesh.from_scoping(scoping=bolt_nodes,mesh=mesh)
    bolt_mesh=bolt_mesh_op.outputs.mesh()
    # Create field with reaction value

    n_bolt_elems_ref=bolt_mesh.elements.n_elements
    loop=True
    while loop:
        escop=bolt_mesh.elements.scoping
        conv_to_nodes=dfpf.operators.scoping.transpose(mesh_scoping=escop,meshed_region=mesh)
        bolt_mesh_op= dfpf.operators.mesh.from_scoping(scoping=conv_to_nodes,mesh=mesh)
        bolt_mesh=bolt_mesh_op.outputs.mesh().get_data()
        n_test=bolt_mesh.elements.n_elements
        if n_test==n_bolt_elems_ref:
            loop=False
        else:
            n_bolt_elems_ref=n_test

    n_bolt_nodes=bolt_mesh.nodes.n_nodes
    nodes=bolt_mesh.nodes
    v1=reaction_fc.get_entity_data_by_id(node)[0][0]
    for i in range (0,n_bolt_nodes):
        nn=nodes.node_by_index(i).id
        reaction_field.append([v1],nn)
    if firstRound:
        full_mesh=bolt_mesh
        firstRound=False
    else:
        op = dfpf.operators.utility.merge_meshes(meshes1=full_mesh,meshes2=bolt_mesh)
        full_mesh=op.outputs.merges_mesh()

full_mesh.plot(reaction_field)
```



DPF Licensing

- Starting from 2023R2, the DPF Server is subject to licensing, as follows:

Server Context	License check	Available operators
Entry	Verifies that a license is available but does not check it out.	Standard capabilities.
Premium	Checks out a license	All Entry level operators + additional operators.

- More details on [this page](#).

Compatible Ansys license increments

The following Ansys licensing increments provide rights to use the licensed DPF capabilities:

- `prepost` available in the [Ansys Mechanical Enterprise PrepPost](#) product
- `meba` available in the [ANSYS Mechanical Enterprise Solver](#) product
- `mech_2` available in the [ANSYS Mechanical Premium](#) product
- `mech_1` available in the [ANSYS Mechanical Pro](#) product
- `ansys` available in the [ANSYS Mechanical Enterprise](#) product
- `dynapp` available in the [ANSYS LS-DYNA PrepPost](#) product
- `dyna` available in the [ANSYS LS-DYNA](#) product
- `vmotion` available in the [Ansys Motion](#) product
- `acprepost` available in the [Ansys Mechanical Enterprise](#) product
- `acdi_adprepost` available in the [Ansys AUTODYN](#) and [Ansys AUTODYN PrepPost](#) products
- `cfd_prepst` available in the [Ansys CFD Enterprise](#) product
- `cfd_prepst_pro` available in the [Ansys CFD Enterprise](#) product
- `vmotion_post` available in the [Ansys Motion Post](#) product
- `vmotion_pre` available in the [Ansys Motion Pre](#) product
- `advanced_meshing` available in the [Ansys CFD Enterprise](#) product
- `fluent_meshing_pro` available in the [Ansys CFD Enterprise](#) product
- `fluent_setup_post` available in the [Ansys CFD Enterprise](#) product
- `fluent_setup_post_pro` available in the [Ansys CFD Enterprise](#) product
- `acfx_pre` available in the [Ansys CFD Enterprise](#) product
- `cfid_base` available in the [Ansys CFD Enterprise](#) product
- `cfid_solve_level1` available in the [Ansys CFD Enterprise](#) product
- `cfid_solve_level2` available in the [Ansys CFD Enterprise](#) product
- `cfid_solve_level3` available in the [Ansys CFD Enterprise](#) product
- `fluent_meshing` available in the [Ansys CFD Enterprise](#) product
- `avrxp_snd_level1` available in the [Ansys Sound Pro](#) product
- `sherlock` available in the [Ansys Sherlock](#) product

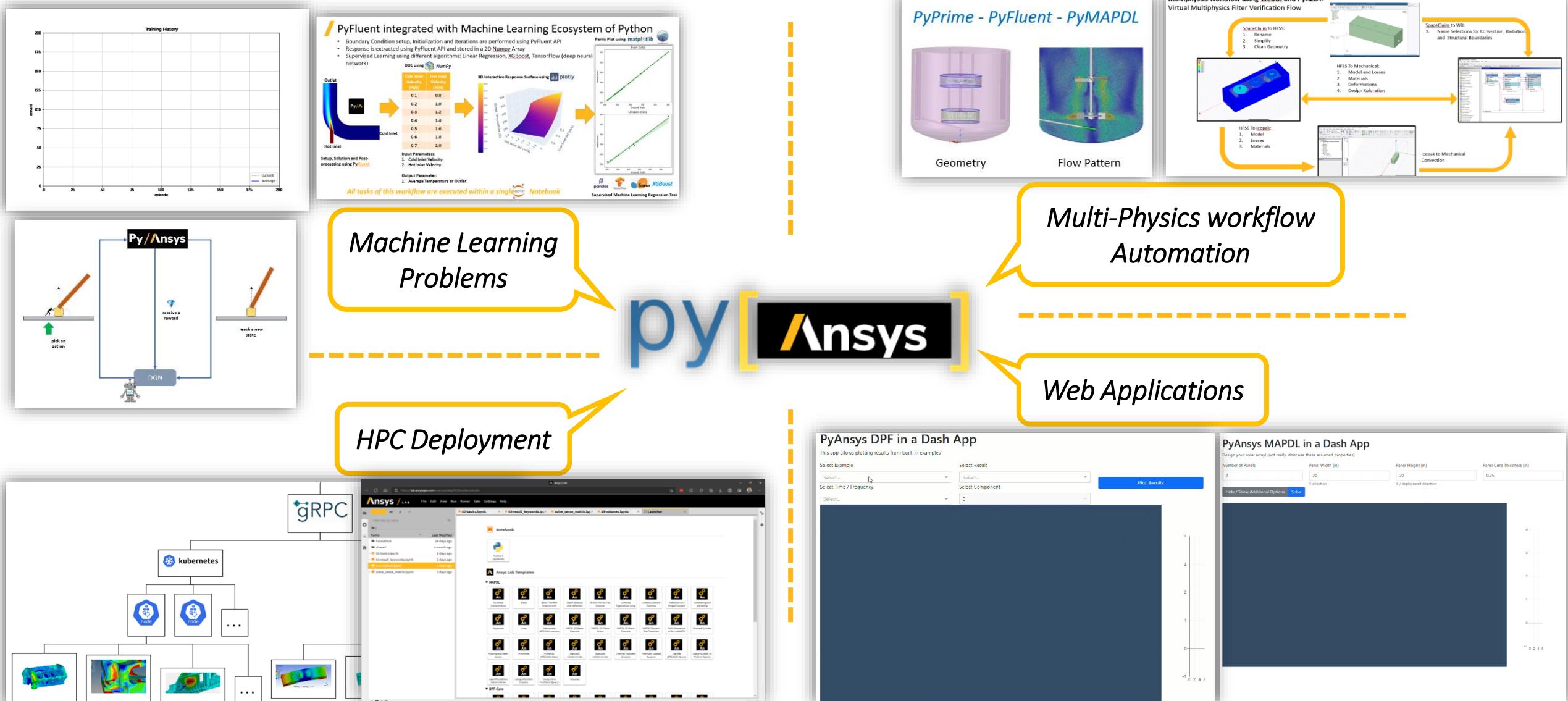


PyDPF Live Demo



PyDPF Workshop Challenge

Key workflows where Ansys customers are using PyAnsys



Angle Bracket Optimization



*Angle bracket has “dimple”
to increase stiffness*

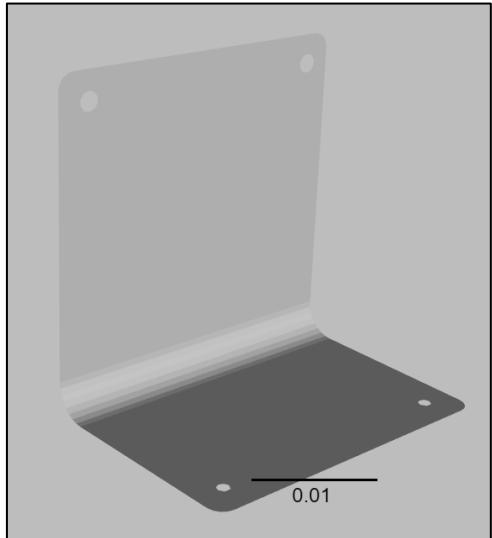


ML Example: Topographical Optimization using Reinforced Learning

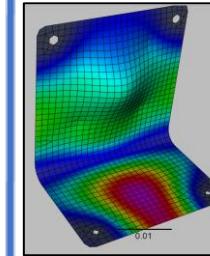


MDP Description

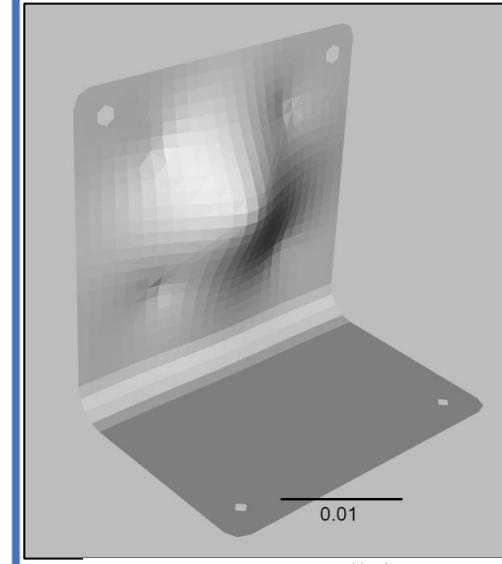
- Action:
 - morph point
 - morph direction
- State:
 - morph extent
- Reward:
 - Increase freq/stiff
 - penalty for distortion



morph solve



receive a
reward

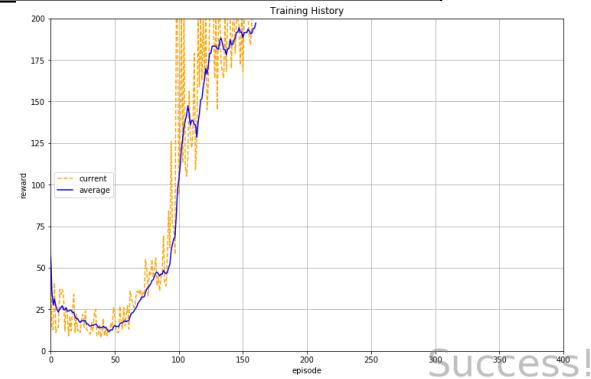


RL Algorithm

- Find sequences of morphing operations that maximize stiffness



Reinforcement
Learning



Success!

Spray Dryer WebApp



Documentation & Help?

Source Code: <https://github.com/ansys/pyansys>

The screenshot shows the GitHub organization page for PyAnsys. At the top, there's a logo and the text "PyAnsys - Ansys Python development organization". Below this, a pinned repository section displays five public repositories:

- pymapdl**: Pythonic interface to MAPDL. Python, 259 stars, 59 forks.
- pyaedt**: AEDT Python Client Package. Python, 32 stars, 7 forks.
- pydpf-core**: Data Processing Framework - Python Core. Python, 23 stars, 6 forks.
- pydpf-post**: Data Processing Framework - Post Processing Module. Python, 12 stars, 1 fork.
- pymapdl-reader**: Legacy binary interface to MAPDL binary files. Python, 15 stars, 9 forks.

Contribute/Post-Issues directly on Github:

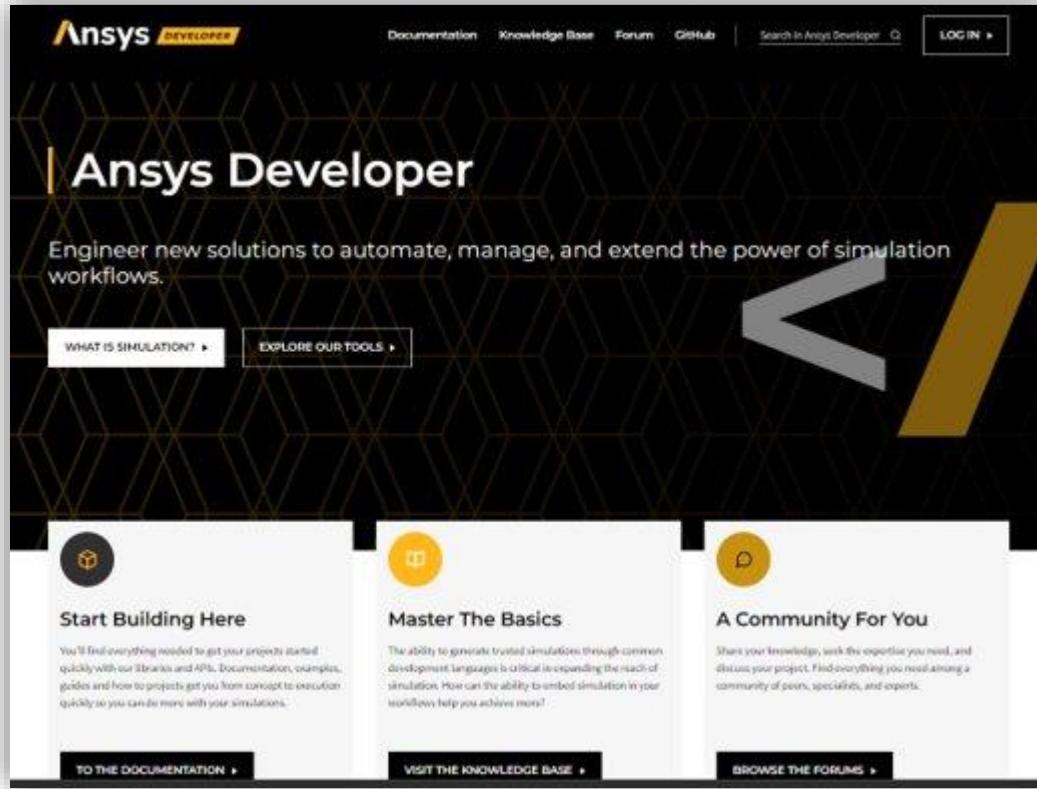
The screenshot shows two GitHub issue lists side-by-side:

- pyansys/pymapdl** (34 open issues):
 - MAPDL server connection terminated after error in do loop
 - Add Python 3.9 Support
 - Iplot and kplot slow and unresponsive within notebook environment when showing number labels
 - Hello, I want to know something about Solve Sparse Linear Systems
 - Autosummary for working plane is broken
 - Include the element Library in PyMAPDL's documentation
 - Implement the APDL "GET shortcuts in PyMAPDL"
 - title, return_plotter, theme ... are not valid parameters for plot_nodal...
 - Visualization of cross sections when plotting
- pyansys/PyAEDT** (22 open issues):
 - Add full doc build to the CI/CD
 - function Maxwell2d.assign_current does not work on Magnetostatic solver enhancement
 - Maxwell 2D example in Example gallery does not work in PyCharm
 - Assigning design variable to parameter "amplitude" Maxwell2d.assign_current() does not work.
 - Add doc coverage CI/CD
 - UDP Parametrization needs expanding to handle Version and NoOfParameters fields
 - The returned string is "10Watt". It should be "10W" or "10Watts"
 - Add Eye Diagrams methods to Circuit
 - Possibly redundant methods in Q2d.py

Documentation:

- Access from GitHub or use these Direct links:
 - <https://docs.pyansys.com/>
 - <https://mapdl.docs.pyansys.com/>
 - <https://dpf.docs.pyansys.com/>
 - <https://aedt.docs.pyansys.com/>

Need additional help? Use the Ansys Developer Portal and Developer Forum



Ansys Developer Portal:
<https://developer.ansys.com/>

The screenshot shows a forum page titled "HOME > ENGINEERING SIMULATION". It displays a list of posts with titles like "how to control contour in PyDPF plot", "Retrieve messages from Mechanical", and "How to get results with DPF for a time/frequency not available in my rst file?". Each post includes a user profile picture, the question title, a link to the post, view count, comment count, reaction count, start date, and most recent activity. The interface includes a navigation bar at the top with page numbers 1 through 6.

Ansys Developer Forum:
<https://discuss.ansys.com/>

Trainings (AIC & ALH)

The image shows two side-by-side video player interfaces. The top interface is titled "Getting Started with PyMAPDL" and features a man in a plaid shirt speaking. The bottom interface is titled "Overview of PyMAPDL" and features a man in a grey sweatshirt speaking. Both interfaces have a dark background and include a "LEARN MORE" button at the bottom right.

Getting Started with PyMAPDL

0:11 / 2:05

Getting Started With PyMAPDL - Course Overview

Overview of PyMAPDL

0:39 / 9:12

Overview of PyMAPDL - Lesson 1

Ansys Innovation Courses (AIC)

PYTHON

New!

python

LEARN SIMULATION

Intro to Python

STRUCTURES

New!

LEARN SIMULATION

Getting Started with PyMAPDL

LEARN MORE →

The image is a screenshot of a group page on the Ansys Learning Hub (ALH). The group is titled "Electronics | Automation" and has 41,506 members. It includes sections for "Learning Guide" and "Learn from the Experts". A banner at the top says "Click here to go back to Electronics Bu...". Below the banner is a search bar and a welcome message. A section titled "Ansys Electronics Desktop Automation Started" lists several bullet points about PyAEDT. There is also an "Upcoming Events" section which currently shows "No active events".

My Dashboard Ansys Campus Ansys Solutions New to Ansys?

Electronics | Automation

Private 41,506 Members

Overview

Learning Guide Learn from the Experts

Electronics | Automation Click here to go back to Electronics Bu...

Search this Group...

Welcome to the Ansys Electronics | Automation Learning Room

If you have completed Ansys Electronics Desktop 3D Modeling and have reached a point where you want to go to the next level in terms of productivity and customization, then you are in the right area. The course hosted here deals with the following topics:

- Efficient code generation to build advanced automation to pre/post processing models
- Connection between development environments and Ansys Electronics Desktop
- Quick extension of AEDT functionality by leveraging existing Python libraries

Upcoming Events

Ansys Electronics Desktop Automation Started

- In which situations can PyAEDT be useful?
- How can I take full advantage of Python functionality?
- How can I create and run scripts?
- How can PyAEDT be integrated in my si...

No active events Add a group event

Ansys
Learning
Hub (ALH)

PyAnsys: Module Cheat Sheets

Cheat sheet for PyMAPDL

/ Launching PyMAPDL

To launch PyMAPDL instance locally and exit it

```
# To launch an instance
# from ansys.mapdl.core import Launch_mapdl
# mapdl=Launch_mapdl()

# To exit the instance
mapdl.exit()
```

To specify a jobname, number of processors, and working directory.

```
jobname="user_jobname"
path="path\of\directory."
mapdl=Launch_mapdl(path=path,
                   numproc=2,
                   run_location=path,
                   jobname=jobname)
```

To connect to an existing instance of MAPDL at IP 192.168.1.30 and port 50001.

```
mapdl=Launch_mapdl(
    ip="192.168.1.30",
    port=50001)
```

To create and exit a pool of instances

```
# To create a pool of 10 instances
from ansys.mapdl.core import LocalMapdlPool
pool=LocalMapdlPool(10)

# To exit the pool
pool.exit()
```

/ PyMAPDL Language

PyMAPDL commands are Python statements that act as a wrapper for AMPL commands. For instance, ESEL, S, type, I, is treated as:

```
mapdl.esel(*args, **kwargs)
```

Commands that start with '/' or have those characters removed.

```
mapdl.prep1() # /PREP1
mapdl.get() # /GET
```

In cases where removing '/' or will cause conflict with other commands, a prefix 'slash' or 'star' is added.

```
mapdl.solu() # SOLU
mapdl.slashsolu() # /SOLU

mapdl.vget() # /GET
mapdl.stargett() # /GET
```

Converting an existing APDL script to PyMAPDL format

```
mapfile="myapdl_inputfile.inp"
pyscript="myscript.py"
mapdl.convert_apdl_script(mapfile, pycscript)
```

/ MAPDL Class

Load a table from Python to MAPDL

```
mapdl.load_table(name, array, var1="var1", var2="var2",
                 var3="var3", **kwargs)
```

To access from or write parameters to MAPDL database

```
# Sets a parameter from a Python array
mapdl.set_parametrized(*args, **kwargs)

# Create a parameter from a NumPy array
mapdl.parameters['myshape']=maparray

mapdl.parameters['myshape']=maparray
```

To accumulate using GET and /GETSET directly to NumPy arrays

```
# Runs /GET command and returns a Python value
mapdl.get_value(*args, **kwargs)
    # Example: mapdl.get_value('etnum')
    #           item1 = 'etnum'
    #           item2 = 't2num'
    #           **kwargs

# Runs /GETSET command and returns a Python array
mapdl.get_array(*args, **kwargs)
    # Example: mapdl.get_array('etnum')
    #           item1 = 'etnum'
    #           item2 = 't2num'
    #           kloop = 1
    #           **kwargs
```

/ Mesh Class

Start the finite element mesh as a VTK UnstructuredGrid object

```
grid=mapdl.mesh.grid
```

Save element & node numbers to Python arrays.

```
# Array of node coordinates
nodes=mapdl.mesh.nodes

# Save node numbers of selected nodes to array
node_num, mapdl.mesh.nnum
node_num=mapdl.mesh.nnum
node_num[mapdl.mesh.nnum]=array
node_num.all=mapdl.mesh.nnum_all
```

Element numbers of currently selected elements
elec_num=mapdl.mesh.elec

All element numbers incl. those not selected
elec_num.all=mapdl.mesh.elec_all

/ Post-Processing Class

This class is used for plotting and saving results to NumPy arrays

```
mapdl.post1()
mapdl.post1(1, 2)

# Plot the nodal equivalent stress
mapdl.post_processing.plot_nodal_equiv_stress()

# Save modal equ. stress to a Python array
mapdl.eqv_stress()

# Map post_processing.plot_nodal_equiv_stress()
mapdl.post_processing.plot_nodal_equiv_stress()

# A plot contour legend using dictionary
mapdl.post_processing.plot_contour()

# Set a colorbar
shar_kwargs={"colorbar": "black",
             "title": "Equivalent Stress (psi)"}
mapdl.post_processing.plot_nodal_equiv_stress(shar_kwargs=shar_kwargs)

# Map post_processing.plot_nodal_equiv_stress()
mapdl.post_processing.plot_nodal_equiv_stress(shar_kwargs=shar_kwargs, n_colors=16)

# Plotting Class


Plotting is interpolated with PyVista by saving the resulting stress and storing within the underlying UnstructuredGrid



```
plotipyvista.Plotter()
mapdl.post_processing.plot_nodal_stress()
mapdl.post_processing.plot_nodal_equiv_stress()
mapdl.post_processing.plot_nodal_equiv_stress(shar_kwargs=shar_kwargs)
mapdl.post_processing.plot_nodal_equiv_stress(shar_kwargs=shar_kwargs, n_colors=16)

Plot the currently selected elements
mapdl.plot(*args, node_numbering='vtk')
 # Example: mapdl.plot(1, 2, 3, 4, 5, 6, 7, 8)

mapdl.vplot(*args, vtk=True, mvc=True, degen=True, scale=True, ...)

Display the selected areas
mapdl.vplot(*args, vtk=True, mvc=True, degen=True, scale=True, ...)

Display the selected lines without
the endpoints
mapdl.lplot(*args, vtk=True, cpos="xy", linc_width=10)

Save fig file of line plot with MAPDL
mapdl.vplot(*args, vtk=True, mvc=True, degen=True, scale=True, ...)

mapdl.pyscript('CST_1')
mapdl.plot(vtk=False)
```



References from PyMAPDL Documentation



- Getting Started
- MAPDL Commands
- API Reference

```

Getting Started with PyMAPDL | PyMAPDL on GitHub | Visit mapdl.docs.pyansys.com

Cheat sheet for PyFluent

Solver Settings Object Interface

/ Launch Fluent locally

The following method is used to start Fluent from Python in gRPC mode.

This starts Fluent in the background so that commands can be sent to Fluent from the Python interpreter.

```
import ansys.fluent.core as pyfluent
solver = pyfluent.launch_fluent(mode = "solver",
                                solver_type = "Python")
```

The solver object contains attributes such as `file`, `solution`, and `results`, which are also instances of setting objects.

/ Import mesh in launched session

The following examples show how to read the available mesh file formats.

```
import_file_name = "example_file.msh.h5"
solver.read_file(file_type="case", file_name=import_file_name)
```

There are other specific APIs available for reading case files and reading case-data files.

```
solver.read_case(case_data=True)
```

```
import_file_name = "example_file.case.h5"
solver.read_file(file_type="case", file_name=import_file_name)
```

Reading case-data files

/ Enable heat transfer physics

The following examples show how to enable heat transfer by activating the energy equation.

```
solver.setup.models.energy.enabled = True
```

Accessing the object state with `print`

```
>>> from pprint import pprint
>>> pprint(solver.setup.models.energy())
{'enabled': True,
 'inlet_diffusion': True,
 'kinetic_energy': False,
 'pressure_work': False,
 'viscous_dissipation': False}
```

/ Define materials

This example shows how you use Solver settings objects to define materials.

```
solver.setup.materials.core.database.material_by_name[
    "water-air"] = "Water-Air"
solver.setup.cell_zone_conditions.fluid["value-fluid"][
    "material"] = "water-air"
```

/ Apply solution settings

PyFluent allows you to use Solver settings objects to apply solution settings, initialize, and solve.

```
solver.solution_INITIALIZATION.hybrid_initialize()
solver.solution_run_calculation.tolerance(
    number_of_iterations=100)
```

/ Define boundary conditions

The examples in this section show how you use Solver settings objects to define boundary conditions.

```
solver.setup.boundary_conditions.velocity_inlet["cold-inlet"] = "constant"
# or
# "option", "constant or expression",
# "constant": 0.4
solver.setup.boundary_conditions.velocity_inlet[
    "cold-inlet"]
    .label = "Inlet"
    .angle = 90
    .normal_and_tangential = "Diameter"
solver.setup.boundary_conditions.velocity_inlet["cold-inlet"].turbulence_intensity = 5
solver.setup.boundary_conditions.velocity_inlet["cold-inlet"]
    .turb_hybrid_ic.c_turb = 4e-05
solver.setup.boundary_conditions.velocity_inlet["cold-inlet"]
    .turb_hybrid_ic.turb_ic = "constant"
# or
# "option", "constant or expression",
# "constant": 200.0
    ]
```

/ Post-processing

PyFluent allows you to post-process data with `results` object. The following example shows how to create and display contours on a plane.

```
solver.results.graphics.contour["contour-type"] = {}
solver.results.graphics.contour["contour-type"].label = "contour"
solver.state.print_state()
solver.results.graphics.contour["contour-type"].Field = "Temperature"
solver.results.graphics.contour["contour-type"].surfaces_list = [
    "symmetry_xy-plane",
    ]
```

/ Temperature Contour



References from PyAnsys Documentation

- Getting Started
- PyFluent Solver Settings Objects
- PyFluent Examples

Getting Started with PyAnsys | PyAnsys on GitHub | Visit dev.docs.pyansys.com

PyAEDT EDB-API Cheatsheet	
/ Launching EDB-API using PyAEDT	Ansys
EDB manager manages the EDB Database. An EDB Manager is a folder that contains the database representing any part of a PCB that can be opened and edited using the EDB class.	
# To launch an instance	
import pyaedt from pyaedt import Edb edb=pyaedt.Edb("C:\Users\...\\2023.1", "edbpath=... edb.set_db(...)") # Save the edb file edb.close_db() # Exit the edb file	edb.setup.add_source.terminal.ts.ground("V1", 1) edb.solve =edb.solve.solve(ivave)
/ Stackup and Layers	
These classes are the containers of the layer and stackup manager of the EDB API.	
# Adding a stackup layer	
# To get the names of all the layers edb.stackup.stackup_layers.keys()	
/ Modeler and primitives	
These classes are the containers of primitives and all related methods. Primitives are planes, lines, rectangles, and polygons.	
# Creating a polygon by defining points	
points = [(0, 0, 1), (0, 10, -3), (10, 0, -3), (10, 10, -3), (0, 10, 1), (0, 0, 1)] edb.modeler.primitives.create_frontpoints(points, layer_name = "Base")	
/ Components	
Component class contains API reference for net management. The component object is created directly from main application using the property components.	
# To get the list of components	
edb.components.components.keys()	
# To get the connection of components	
edb.components.get_component.net.connections.info ("Q101")	
/ Nets	
The Net class contains API references for net management. The main net object is created directly from main application using the property nets.	
# List of all nets available in EDB file	
edb.nets.netlist. # To delete a net edb.nets.net["Net1"].delete()	
/ Vias and padstacks	
These containers contains the API references for padstack management. The main padstack object is created directly from property via and the property padstacks.	
# Creating a via	
edb.padstacks.place_via(position = [3,-3, -6,-3], "Via1") # To get the pad parameters edb.padstacks.get_pad_parameters()	
/ Sources and Excitation	
These classes are the containers of sources methods of the EDB Manager and Siwave	
# To get the dictionary of EDB excitations	
edb.excitations	
# To create different port types	
edb.create_port_type("positive_port", positive_primitive_id = trapezoid[0].id, positive_points_ns_odege = +1.pi.points, negative_points_ns_odege = -1.pi.points, negative_points_ndge = +1.pi.points, name = "wave_port", id=1)	
/ Simulation Setup	
These classes are the containers of setup classes in EDB for both HFSS and Siwave.	
# HFSS simulation setup	
setup = edb.create_aeiaff.setup(name = "my_aeiaff") setup.set_dc_dc.current = True setup.set_dc_dc.voltage = 10 setup.set_dc_dc.order_basis = "linear" setup.adaptive.settings .add_dc_adaptive_frequency_data("0.001", 8, 0.01)	
# Siwave Simulation setup	
setup = edb.create_siwave_dc.analysis(name = "my_siwave") setup.set_dc_dc.current = True setup.set_dc_dc.voltage = 10 setup.dc_slider.position = 0	
/ Simulation Configuration	
These classes are the containers of simulation configuration constructors for the EDB.	
# AC settings	
sim.setup.ac_settings.acsrc.freq = "100Hz" sim.setup.ac_settings.step.freq = "100Hz" sim.setup.ac_settings.step.freq = "10MHz"	
# Batch solve	
sim.setup.batch_solve.new_simulation.configuration(sim. simulation_type = sim.SolverType. SiwaveET2) sim.setup.batch_solve.settings .dc_circuit.circuit = "Circuit1" do_circuit.sundeign = True sim.setup.batch_solve.settings.step.freq = False sim.setup.batch_solve.settings.signal.basis = sim.siggen.list sim.siggen.list.components = component.list sim.siggen.list.settings.power.net = power.xls.list	
# Saving config file	
sim.setup.export_on_cm.push.json(project.push, "configuration.json") edb.build.simulation.project(sim.setup.eup)	
/ SiWave Manager	
Siwave is a specialized tool for power integrity, signal integrity, and thermal analysis of PCBs and PWBs. The tool solves power delivery systems and high-speed channels in electronic devices. It can be accessed from PyAEDT in Windows only. All setups can be implemented through EDB API.	
# To print the version of the Siwave	
print(pyaedt.siwave.siwave_version)	
# To start the Siwave session	
siwave.start(siwave.siwave_start("2023.1"))	
# To open a project	
siwave.open_project("C:\...\my_siwave.siw")	
# To close a project	
siwave.close_project()	
References from PyAEDT Documentation	
Getting Started	

Getting Started With AEDT / Ansys Innovation Courses /

Getting Started with PyDynamicReporting / PyAnsys on GitHub / Visit dynamicreporting.docs.pyansys.com

Getting Started With AEDT / Ansys Innovation Courses /

And many more on....
cheatsheets.docs.pyansys.com
or
developer.ansys.com

Testimonials – Research Community

Pyansys tremendously helped by providing an **easy** way to procedurally read and process data directly from Ansys **without any human intervention**.

Combined with Python's enormous data analysis and **plotting capability** this helped me automate extraction of knowledge from 100s of giant Ansys simulations and saved a lot of expensive human hours

Ph.D. Student

You can **easily** retrieve node stresses and temperatures. Then outside, there is a material database which you can retrieve yield strength based on temperatures, and finally you can calculate factor of safety. **And perhaps you can plot it.** Think about you have many analyses in different locations and you can automatize this process

Aerospace Engineer

I was trying to avoid exporting my results to csv and then importing back into python. [I] started to look also into the "controller" features, which I came to like a lot.

I also contributed to pyansys once, to support reading transient analyses with a reduced number of DOFs [...]. As far as I remember it was written in C, which I don't "speak".

Ph.D. Student

Quite **easy** to embed the simulation in different environments. I also have to say, how glad I am that the plotting functionality in pyansys is so much faster and intuitive than its APDL counterpart

Undergraduate Student

data extraction **data transforms** **plotting**

automation **solver embedding** **user contributions**

Conclusions

- PyAnsys introduces a **paradigm shift** in how Ansys simulation tools will be used going forward. Ansys is the '**first**' simulation software provider to introduce such **dynamic interaction** with its products.
- PyAnsys has been deployed on **GitHub** and is **controlled by ANSYS**.
- Ability to separate **Pre-processing**, **FE model** and **Post-processing** from outside the Ansys environment is a strength that PyAnsys has and will help us **deploy**, **Maintain** and **scale** for applications across various industries.
- **AI/ML Community** can easily integrate ANSYS physics capabilities into their processes.



Appendix

The needs

- More simulation: more complex, more easily, more automation
- Ansys provides simulation solutions for most applications, but there has a lot of file formats and data models to handle

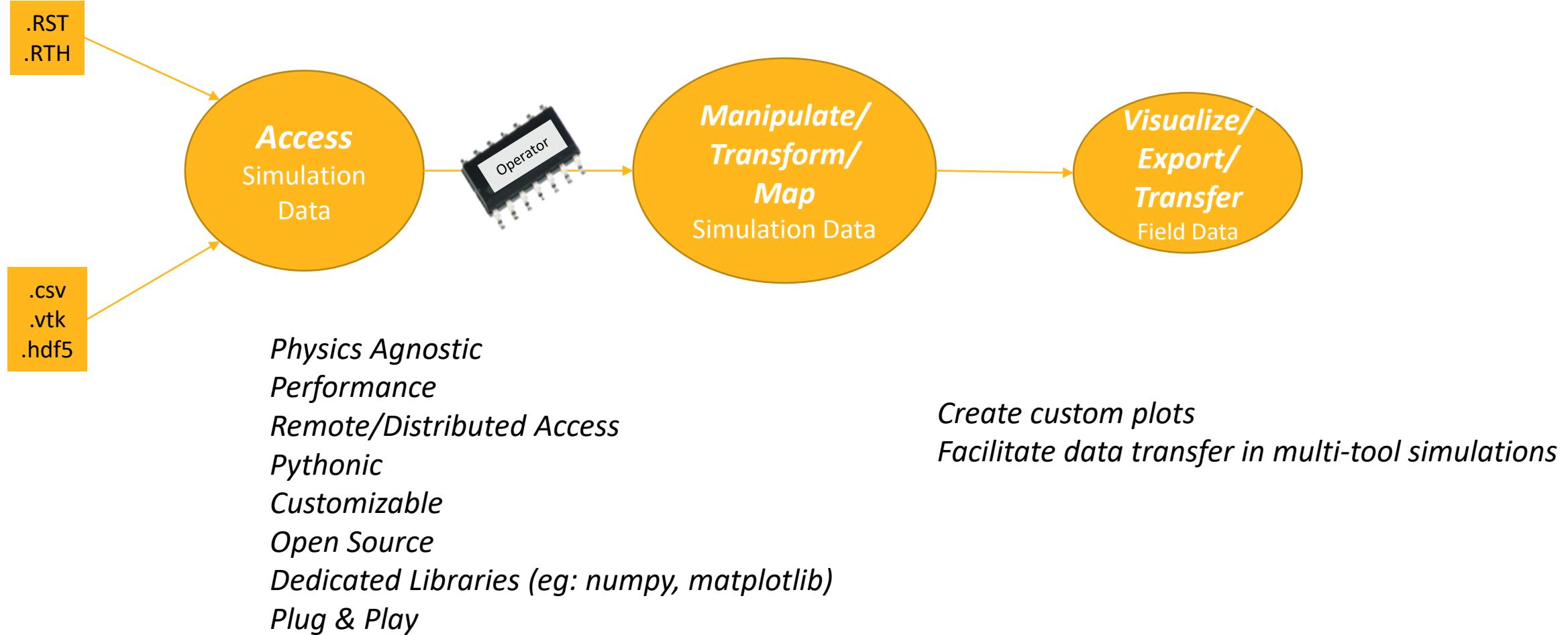
La solution

Ansys Data Processing Framework

- **Data** -> any kind of simulation data
- **Processing** -> efficient handling (read, write, send) or transformation of data
- **Framework** -> anyone can build upon
- Ansys -> optimized for Ansys data

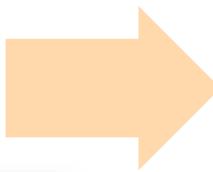
DPF: a powerful way to post-process and create custom results

Very likely the tool most needed by our customers



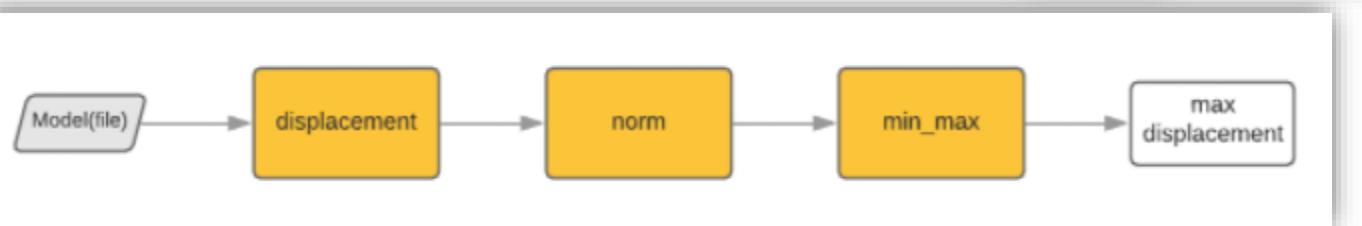
PyDPF-Core

Run simulation in ANSYS and save binary result file (.rst)



Use PyAnsys to:

- Manipulate and create new data easily thanks to DPF-Core



DPF can access data from **solver result files** as well as several neutral formats (**csv**, **hdf5**, **vtk**, etc.).

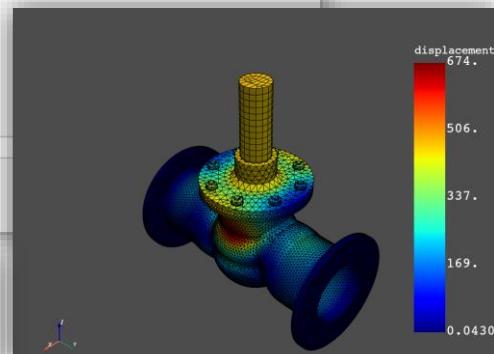
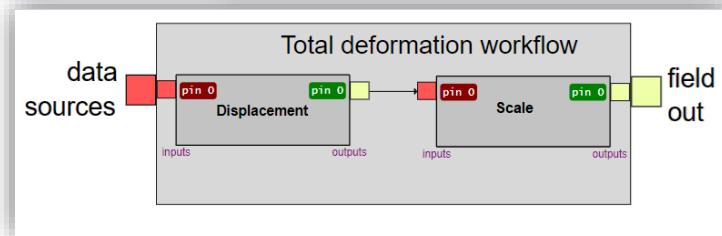
```
## Use DPF core to manipulate data
from ansys.dpf import core

model = core.Model(file_name)
U = model.results.displacement()
field_container = U.outputs.fields_container()
print(field_container[0].data)

op = core.Operator('scale') # operator instantiation
op.inputs.field.connect(fieldContainer[0])
scale_value = float(max_total_deformation/min_total_deformation)
op.inputs.ponderation.connect(scale_value)

my_field = op.outputs.field()
print(my_field)
print(my_field.data)

mesh = model.metadata.meshed_region
mesh.plot(my_field)
```



PyMAPDL

- ❑ With PyMAPDL, it's possible to:
 - Create geometry, mesh, model setup
 - Do interactive plotting
 - Post-process
 - Translate MAPDL scripts to PyMAPDL script

- ❑ Either use standard APDL commands through "Run" command, or call MAPDL Pythonically:

```
mapdl.run('/PREP7')
mapdl.run('K, 1, 0, 0, 0')
mapdl.run('K, 2, 1, 0, 0')
mapdl.run('K, 3, 1, 1, 0')
mapdl.run('K, 4, 0, 1, 0')
mapdl.run('L, 1, 2')
mapdl.run('L, 2, 3')
mapdl.run('L, 3, 4')
mapdl.run('L, 4, 1')
mapdl.run('AL, 1, 2, 3, 4')
```

```
mapdl.prep7()
mapdl.k(1, 0, 0, 0)
mapdl.k(2, 1, 0, 0)
mapdl.k(3, 1, 1, 0)
mapdl.k(4, 0, 1, 0)
mapdl.l(1, 2)
mapdl.l(2, 3)
mapdl.l(3, 4)
mapdl.l(4, 1)
mapdl.al(1, 2, 3, 4)
```

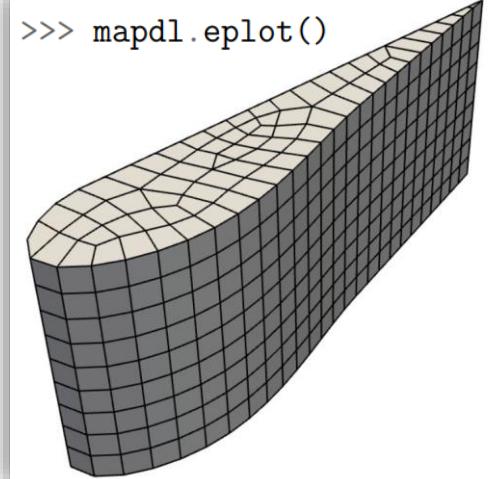
PyAnsys Code Example: Meshing

```
>>> mapdl.et(1, 'SOLID186')
>>> mapdl.vsweep('ALL')
>>> mapdl.esize(0.1)
>>> mapdl.mesh
```

ANSYS Mesh

Number of Nodes:	7217
Number of Elements:	2080
Number of Element Types:	2
Number of Node Components:	0

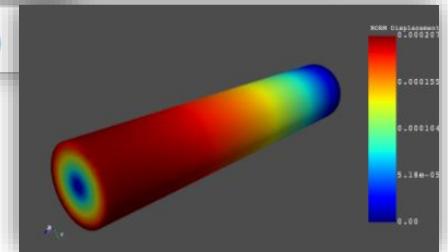
```
>>> mapdl.eplot()
```



```
POST1:
PRNSOL, U, X
PRINT U NODAL SOLUTION PER NODE
***** POST1 NODAL DEGREE OF FREEDOM LISTING *****
LOAD STEP= 1 SUBSTEP= 1
TIME= 1.0000 LOAD CASE= 0
THE FOLLOWING DEGREE OF FREEDOM RESULTS ARE IN THE GLOBAL COORDINATE SYSTEM
NODE UX
1 0.10751E-003
2 0.85914E-004
3 0.57069E-004
4 0.13913E-003
5 0.35621E-004
6 0.52186E-004
7 0.30417E-004
8 0.26110E-004
```

```
>>> mapdl.set(1, 1)
>>> disp_x = mapdl.post_processing.nodal_displacement('X')
array([1.07512979e-04, 8.59137773e-05, 5.70690047e-05, ...,
       5.70333124e-05, 8.58600402e-05, 1.07445726e-04])
```

```
>>> mapdl.post_processing.plot_nodal_displacement('X')
```



DEMO: PyMAPDL

PyANSYS MAPDL Demo

Create a simple cylinder from scratch within pyansys, solve it, and plot the results.

```
[1]: import numpy as np
from ansys.mapdl.core import launch_mapdl

mapdl = launch_mapdl(loglevel='WARNING')
```

Define parameters for a cylinder with an outside torque

```
[ ]: radius = 2
h_tip = 4
height = 40
elemsize = 0.35
force = 100/radius
pressure = force/(h_tip*2*np.pi*radius)
```

Preprocessing

Generate the cylinder and plot it

```
[ ]: mapdl.clear()
# Define higher-order SOLID186
# Define surface effect elements SURF154 to apply torque
# as a tangential pressure
mapdl.prep7()
mapdl.et(1, 186)
mapdl.et(2, 154)
mapdl.r(1)
mapdl.r(2)
```

DEMO: PyDPF (Core & Post)

The screenshot shows a Microsoft Edge browser window displaying the PyANSYS DPF-Core documentation at <https://dpfdocs.pyansys.com/index.html>. The page title is "PyANSYS DPF-Core". The main content area contains an introduction to the Data Processing Framework (DPF), explaining its purpose of providing a toolbox for accessing and transforming simulation data. It highlights that DPF is a workflow-based framework based on physics agnostic mathematical quantities, allowing for modular and easy-to-use capabilities. A brief demo section shows a Python code snippet for opening a result file and extracting results, followed by the resulting output text. The browser's address bar, navigation buttons, and a taskbar with various application icons are visible at the bottom.

PyANSYS DPF-Core

The Data Processing Framework (DPF) is designed to provide numerical simulation users/engineers with a toolbox for accessing and transforming simulation data. DPF can access data from solver result files as well as several neutral formats (csv, hdf5, vtk, etc.). Various operators are available allowing the manipulation and the transformation of this data.

DPF is a workflow-based framework which allows simple and/or complex evaluations by chaining operators. The data in DPF is defined based on physics agnostic mathematical quantities described in a self-sufficient entity called field. This allows DPF to be a modular and easy to use tool with a large range of capabilities. It's a product designed to handle large amount of data.

The Python `ansys.dpf.core` module provides a Python interface to the powerful DPF framework enabling rapid post-processing of a variety of Ansys file formats and physics solutions without ever leaving a Python environment.

Brief Demo

Opening a result file generated from MAPDL (or other of ANSYS solvers) and extracting results from it is as easy as:

```
from ansys.dpf.core import Model
from ansys.dpf.core import examples
model = Model(examples.simple_bar)
print(model)
```

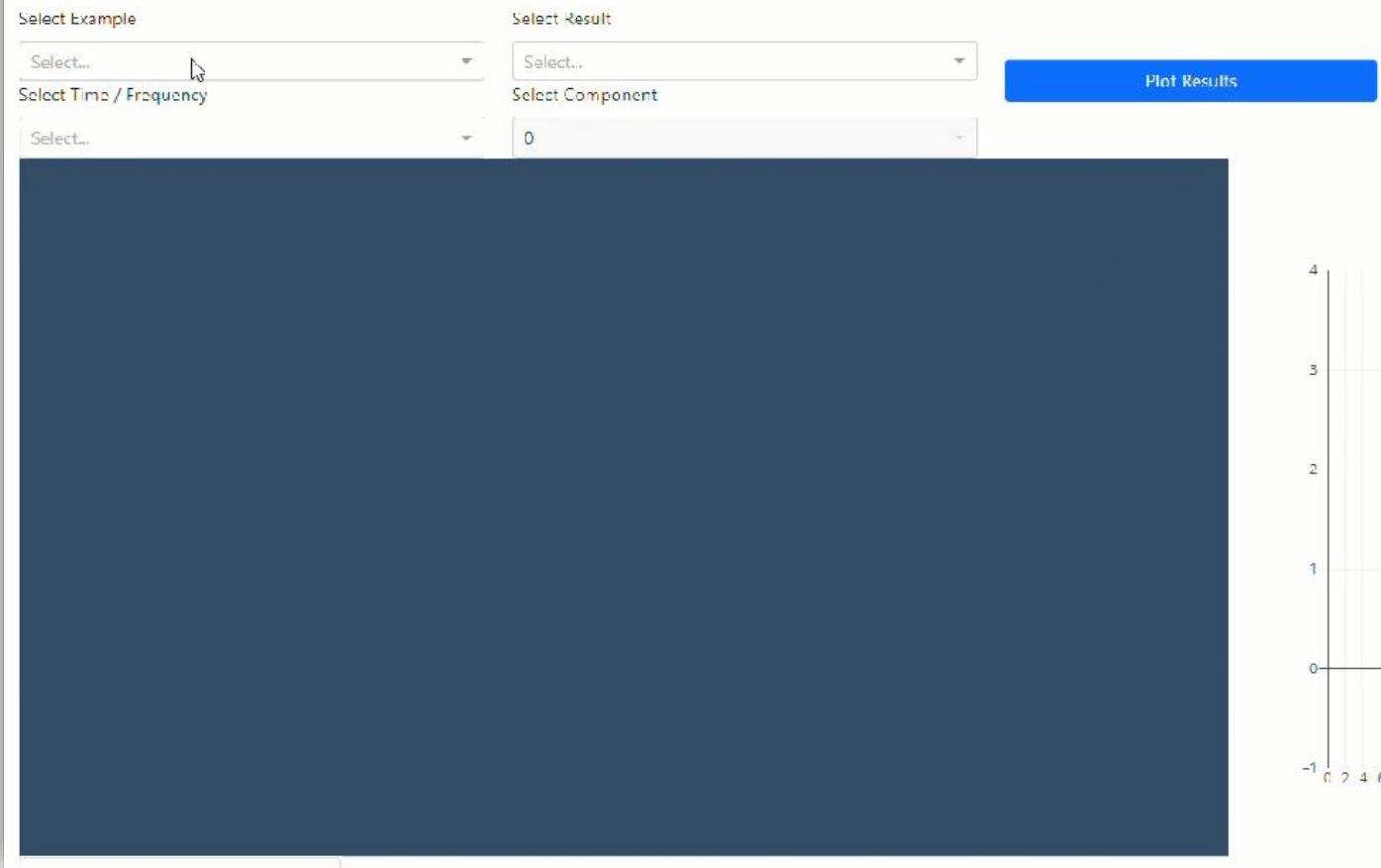
Out:

```
DPF Model
-----
DPF Result Info
Analysis: static
Physics Type: mechanic
Unit system: MKS: m, kg, N, s, V, A, degC
Available results:
U Displacement :nodal displacements
ENF Element nodal Forces :element nodal forces
ENG_VOL Volume :element volume
ENG_SE Energy-stiffness matrix :element energy associated with the stiffness matrix
ENG_AHO Hourglass Energy :artificial hourglass energy
ENG_TH thermal dissipation energy :thermal dissipation energy
ENG_KE Kinetic Energy :kinetic energy
ENG_CO co-energy :co-energy (magnetics)
ENG_TMC incremental energy :incremental energy (magnetics)
```

3rd Party Customers: PyDPF in a Dash App

PyAnsys DPF in a Dash App

This app allows plotting results from built-in examples



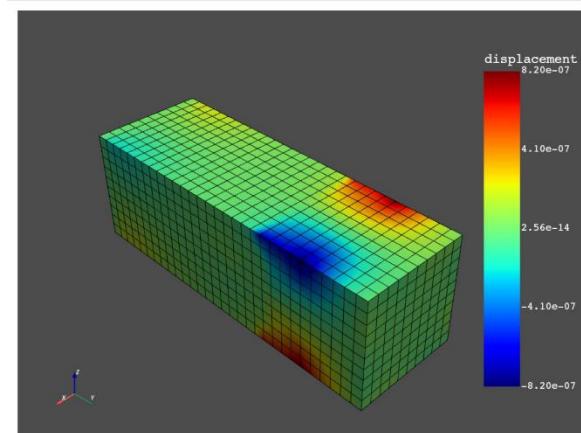
Steve Kiefer

Jan 4 · 9 min read ★

ANSYS in a Python Web App, Part 1: Post Processing with PyDPF

Integrating PyAnsys with Plotly's Dash and the Dash-VTK component to build an Ansys structural analysis post-processing web application

[ANSYS in a Python Web App, Part 1: Post Processing with PyDPF](#) | by Steve Kiefer | Jan, 2022 | Towards Data Science



3rd Party Customers: PyMAPDL in a Dash App

PyAnsys MAPDL in a Dash App

Design your solar array! (not really, dont use these assumed properties)

Number of Panels: 2 Panel Width (in): 20 Panel Height (in): 20 Panel Core Thickness (in): 0.25
Y direction X / deployment direction

[Hide / Show Additional Options](#) [Solve](#)



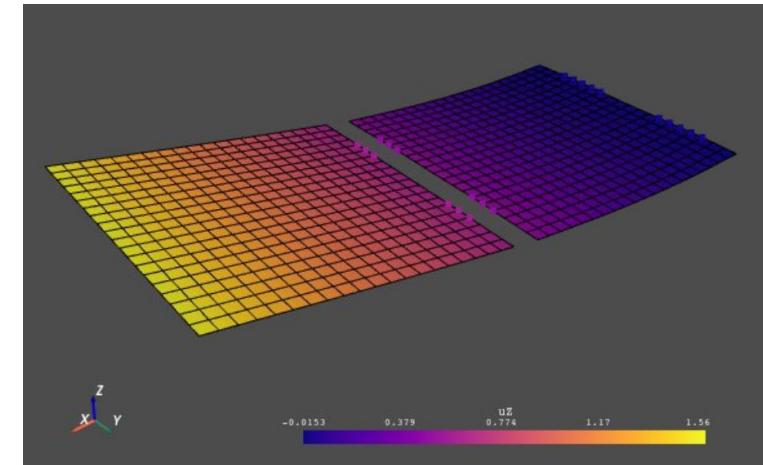
Steve Kiefer

Jan 20 · 9 min read ★

ANSYS in a Python Web App, Part 2: Pre Processing & Solving with PyMAPDL

Integrating PyAnsys with Plotly's Dash and the Dash-VTK component to build an Ansys structural analysis web application

[ANSYS in a Python Web App, Part 2: Pre Processing & Solving with PyMAPDL](#) | by Steve Kiefer | Jan, 2022



Mode	Frequency (Hz)	TX	TY	TZ	RX	RY	RZ	SENE Hinges	SENE Panels
1	0.674	0.00%	0.00%	58.70%	0.00%	96.14%	0.00%	10.42%	89.18%
2	3.18	0.00%	0.00%	0.00%	73.97%	0.00%	0.00%	1.79%	98.21%
3	3.72	0.00%	0.00%	21.28%	0.00%	3.52%	0.00%	22.89%	77.11%

Ansys

