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

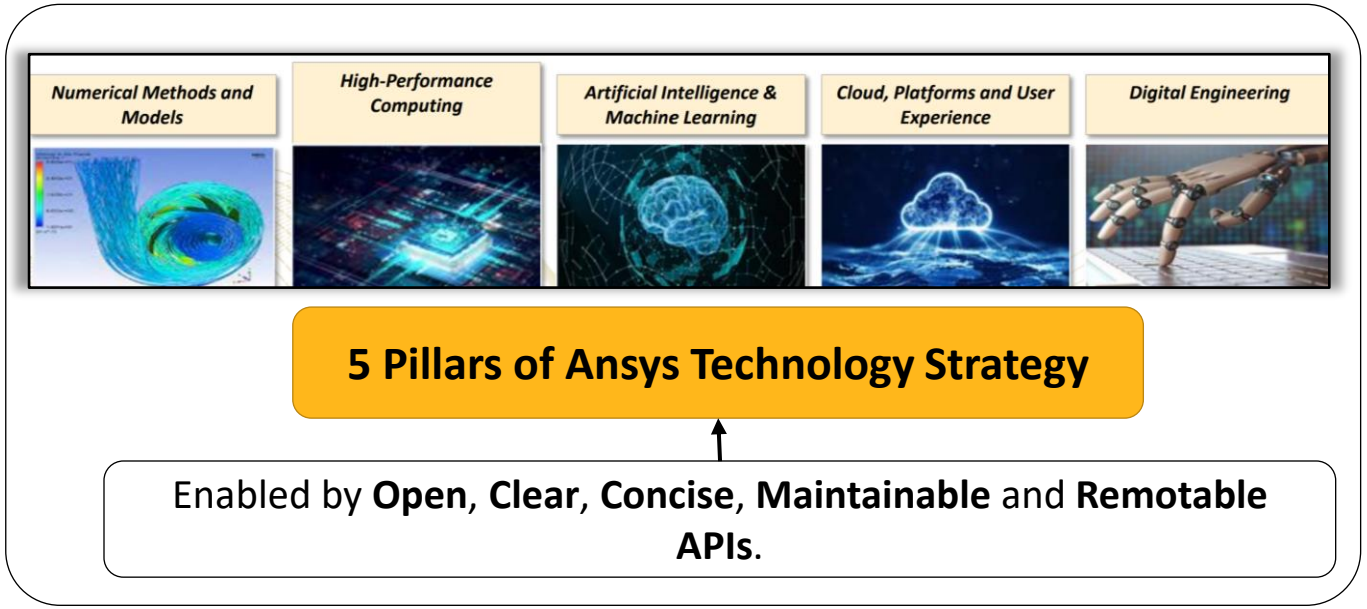
Need for APIs?

Evolving Ansys Customer Needs:

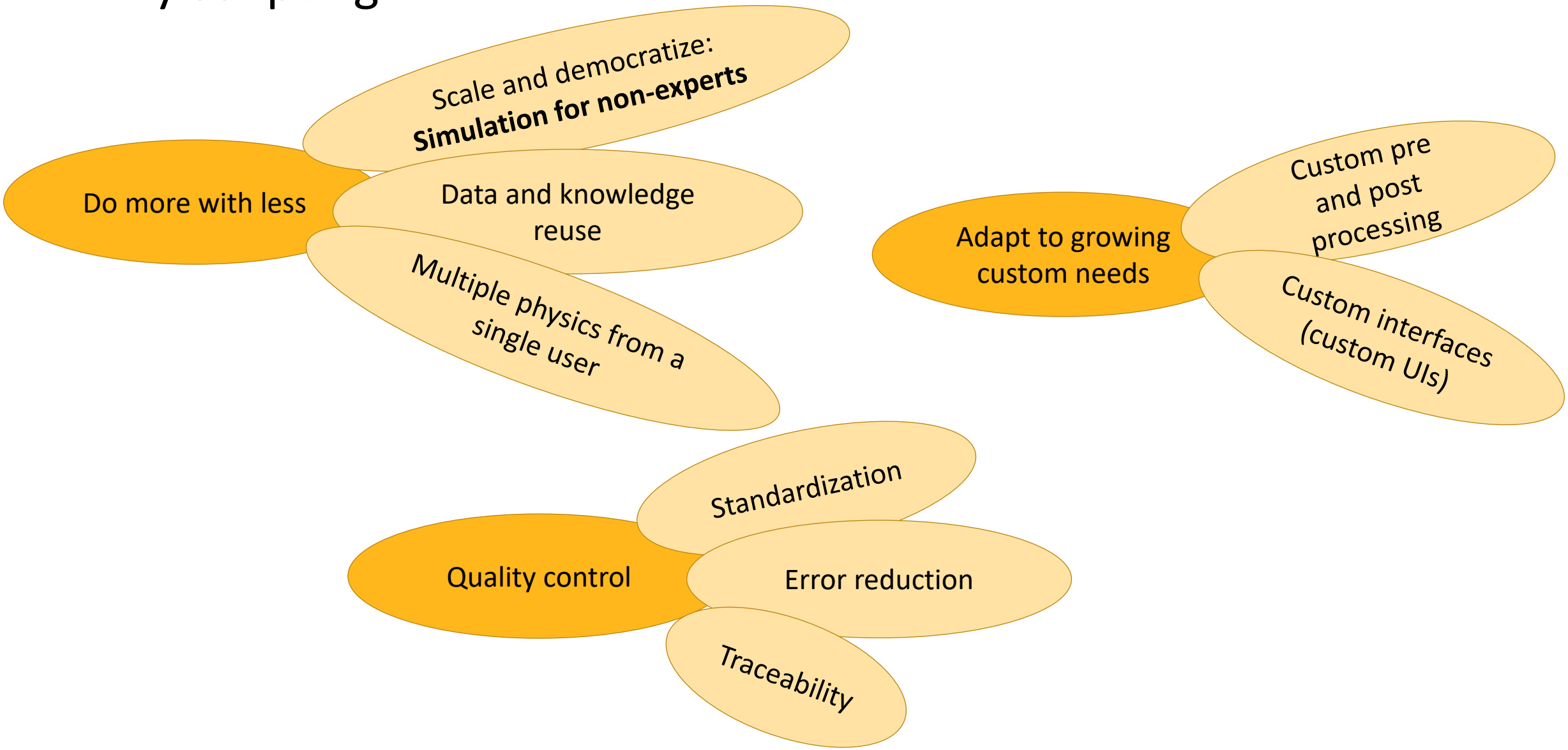
- Democratization of simulation (including non-engineers)
- Accelerate predictable insights
- Connect Ansys technologies to wider enabling technologies such as AI/ML

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?



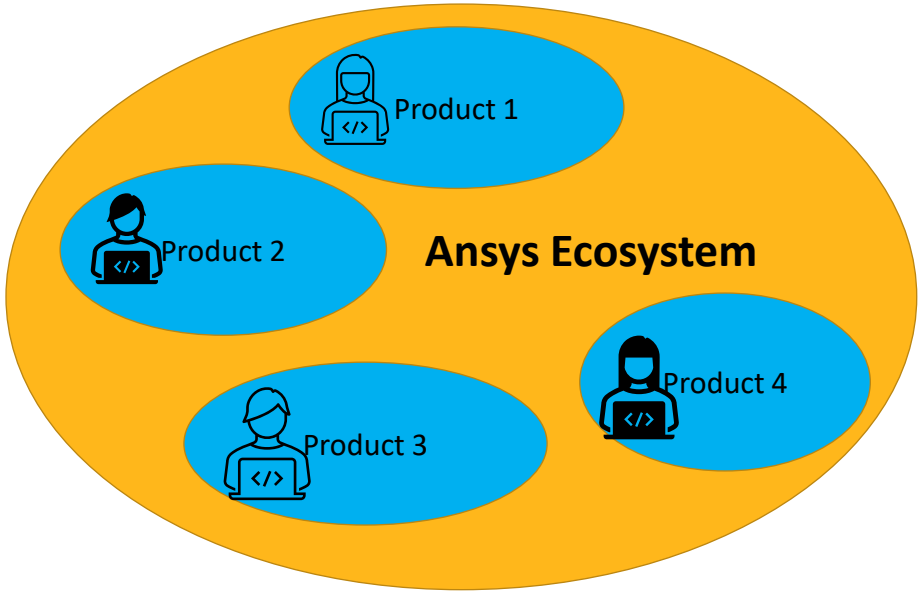
Why scripting?

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

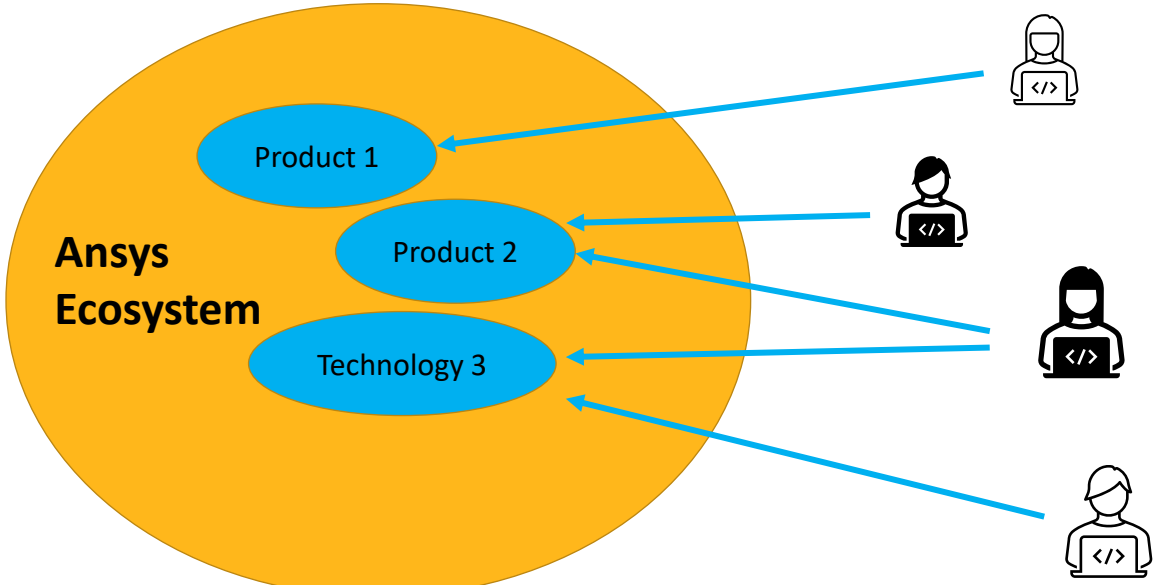
(*) and saving ourselves time as well ☺

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

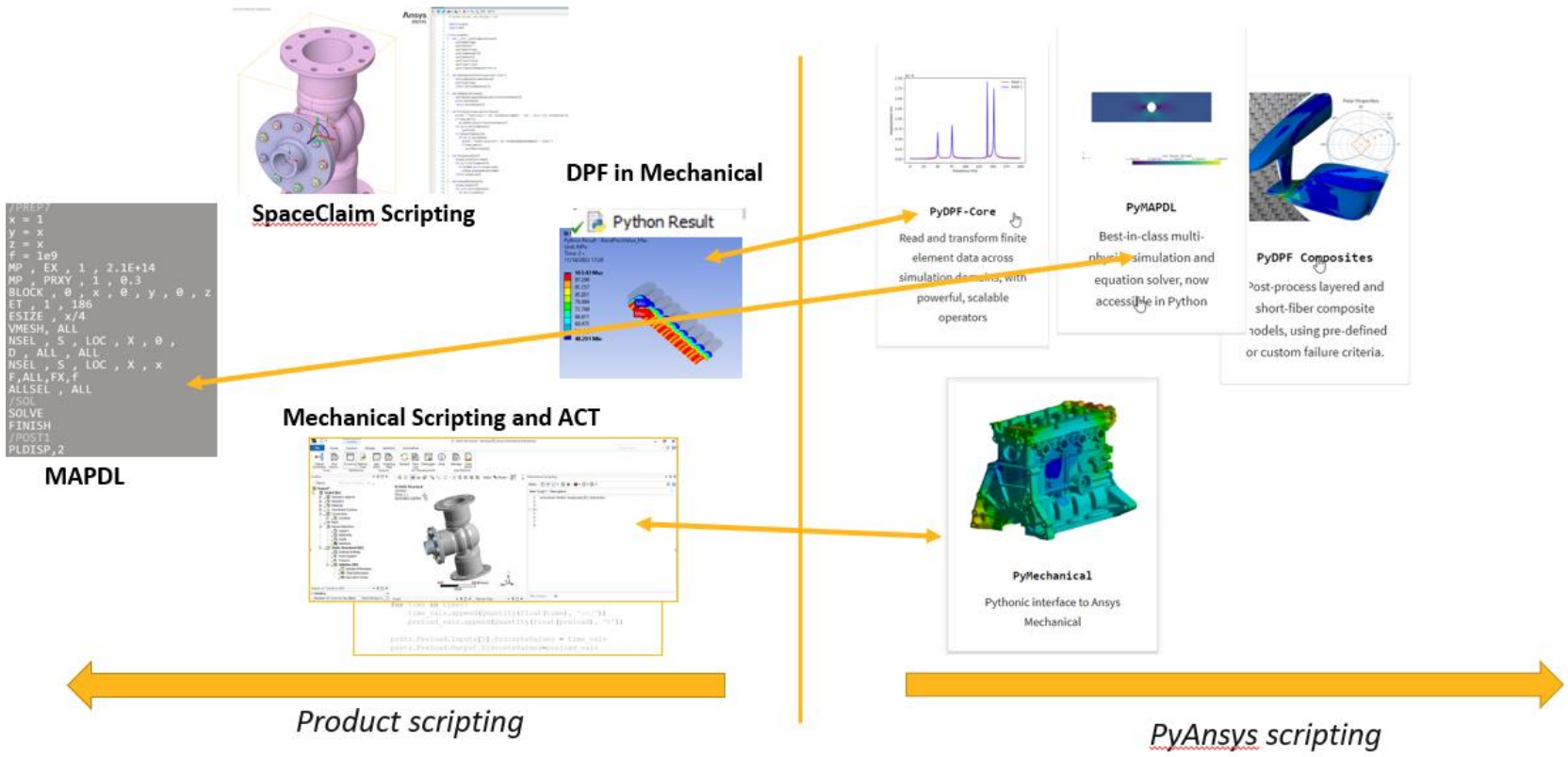


Product scripting



PyAnsys scripting

With a lot of similarities:



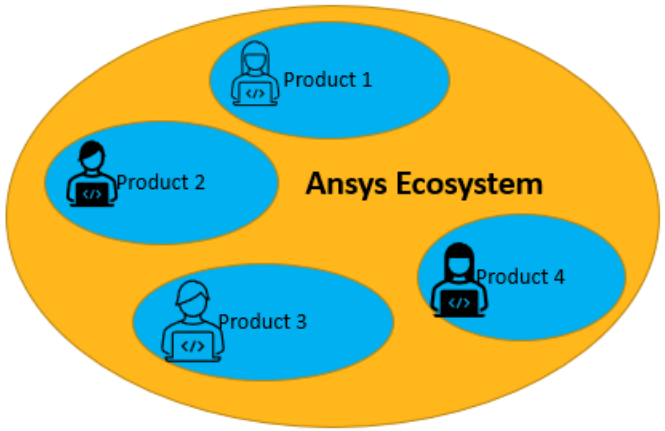
PyAnsys

Ansys

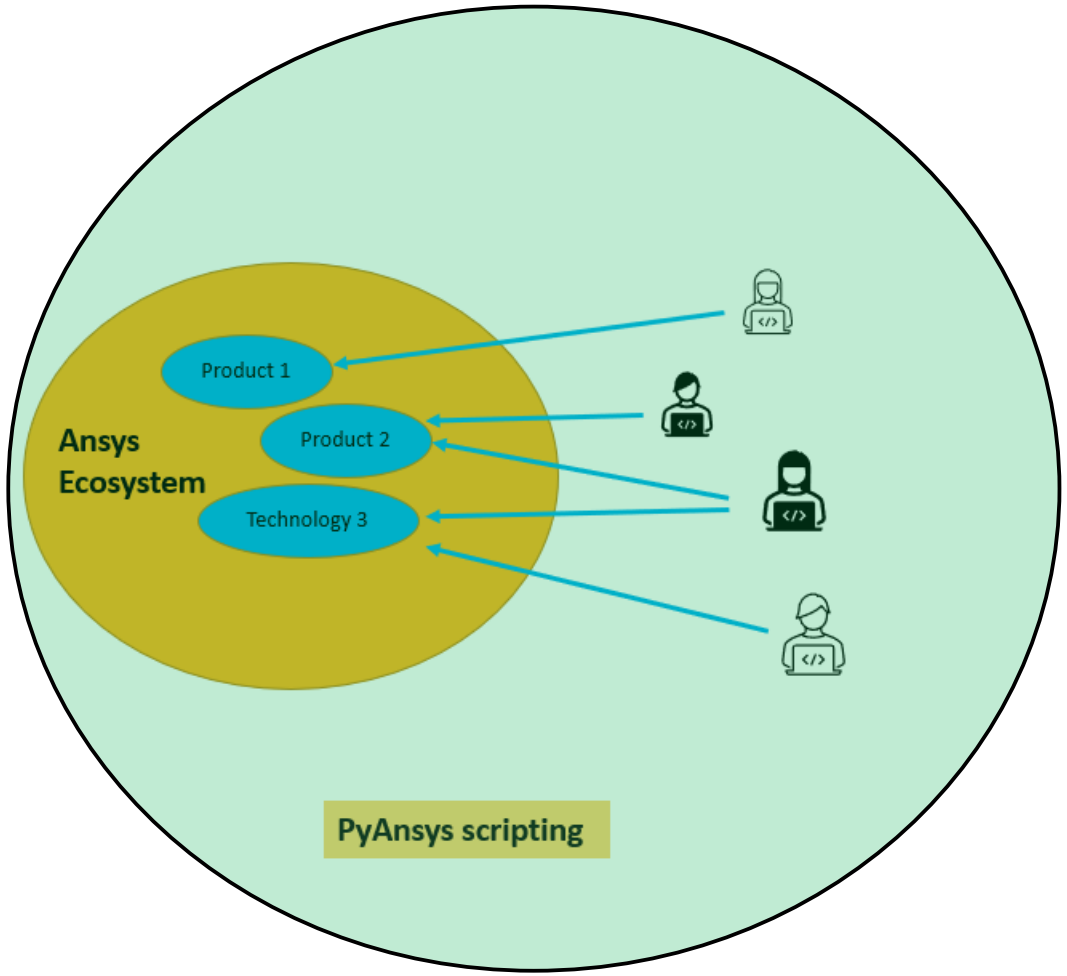
What is PyAnsys?

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

PyAnsys scripting



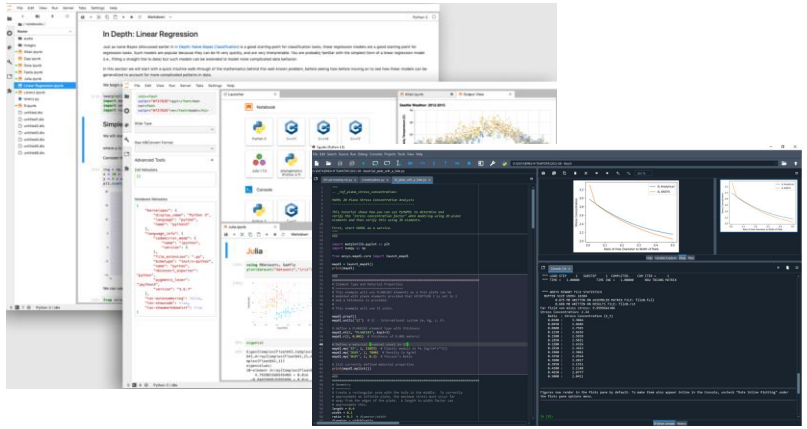
Product scripting



PyAnsys scripting

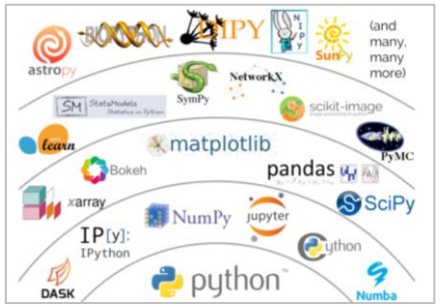
Scripting from within **YOUR** tools

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



Customer tools

Possibility to use other useful Python tools



PyAnsys technology

Products

API — Ansys Product A

API — Ansys Product B

API — Ansys MAPDL

Technology components

Meshing

Data Processing Framework

Ansys Technology

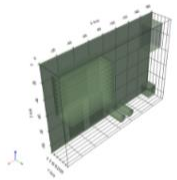
What Packages Are Available?

Check out all available packages in the PyAnsys doc

Simulation Libraries

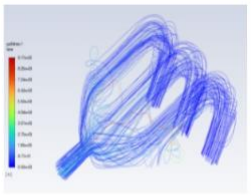
PyMAPDL

Pythonic interface to Ansys MAPDL (Mechanical APDL)



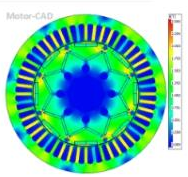
PyAEDT

Pythonic interface to AEDT (Ansys Electronic Desktop)



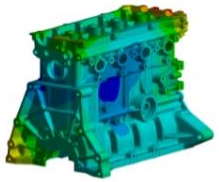
PyFluent

Pythonic interface to Ansys Fluent



PyMotorCAD

Pythonic interface to Ansys Motor-CAD

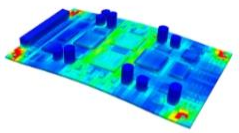


PyMechanical

Pythonic interface to Ansys Mechanical

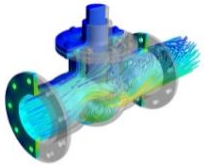
PyOptislang

Pythonic interface to Ansys Optislang



PySherlock

Pythonic interface to communicate with Ansys Sherlock



PySystem Coupling

Pythonic interface to communicate with Ansys System Coupling

PyDyna

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

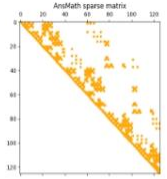
Utility Libraries

PyPrimeMesh

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

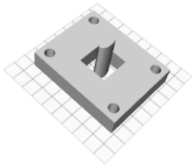
PyDynamicReporting

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



PyAnsys Math

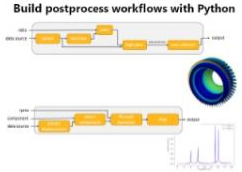
Pythonic interface to PyAnsys Math libraries



PyAnsys Geometry

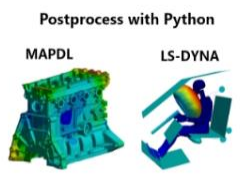
Pythonic interface to the Ansys Geometry service

Postprocessing Libraries



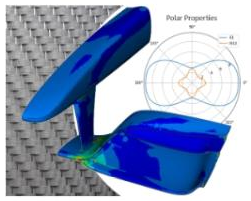
PyDPF - Core

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



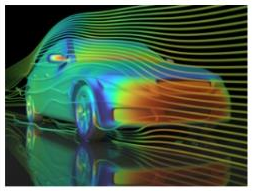
PyDPF - Post

Pythonic interface to access and post process Ansys solver result files



PyDPF - Composites

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

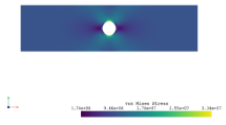


PyEnSight

Pythonic interface to EnSight, the Ansys simulation postprocessor

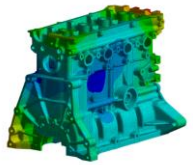
PyAnsys: What's available?

Simulation libraries



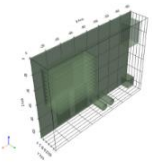
PyMAPDL

Pythonic interface to Ansys MAPDL (Mechanical APLD)



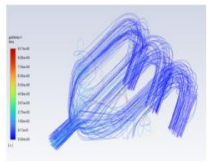
PyMechanical

Pythonic interface to Ansys Mechanical



PyAEDT

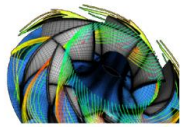
Pythonic interface to AEDT (Ansys Electronic Desktop)



PyFluent

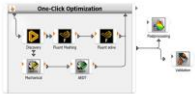
Pythonic interface to Ansys Fluent

Utility libraries



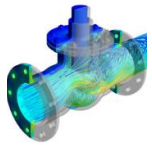
PyPrimeMesh

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



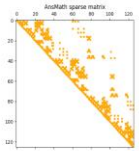
PyOptislang

Pythonic interface to Ansys Optislang



PySystem Coupling

Pythonic interface to communicate with Ansys System Coupling



PyAnsys Math

Pythonic interface to PyAnsys Math libraries

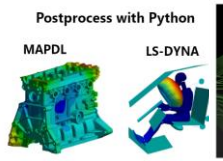
PyDynamicReporting

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

Post-processing libraries

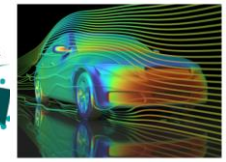
PyDPF - Core

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



PyDPF - Post

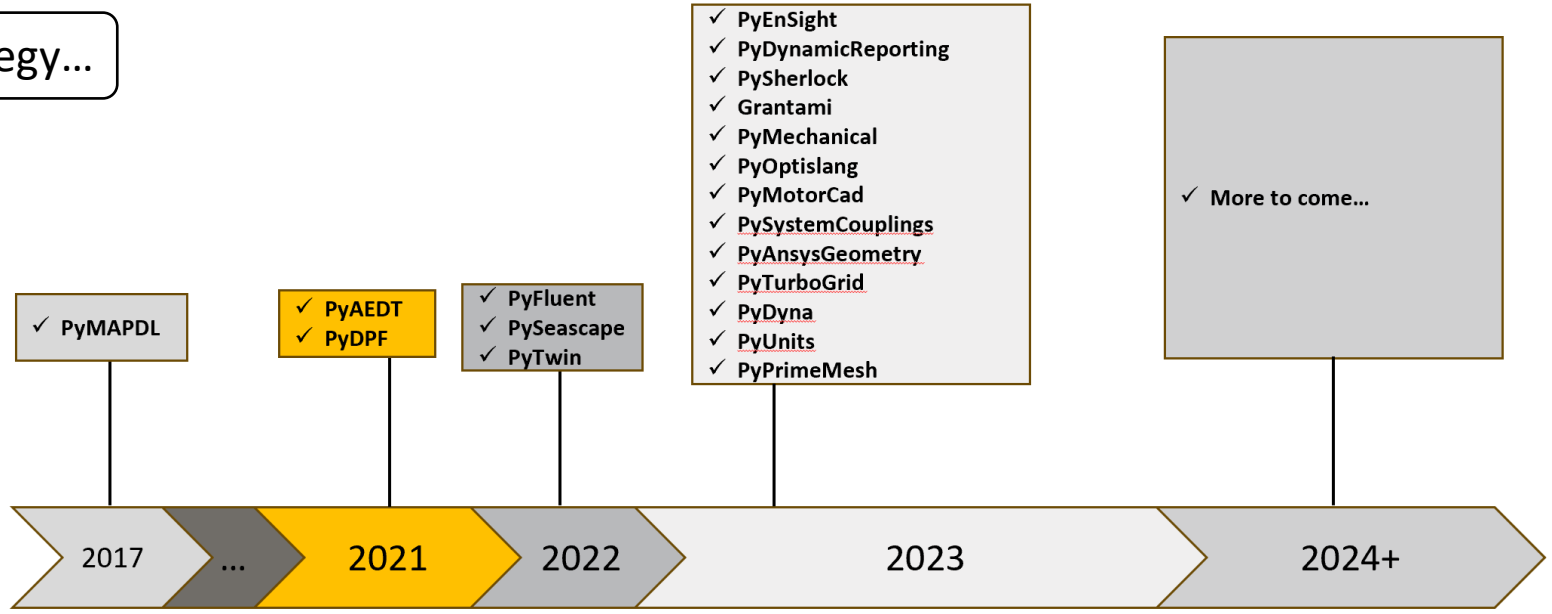
Pythonic interface to access and post process Ansys solver result files



PyEnSight

Pythonic interface to EnSight, the Ansys simulation postprocessor

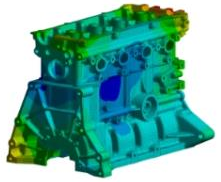
5-year strategy…



Keep an eye on…
docs.pyansys.com

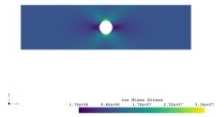
And If I’m a Structural Engineer?

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

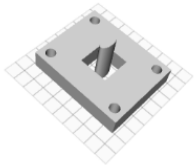
Utility Libraries

PyPrimeMesh

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

PyDynamicReporting

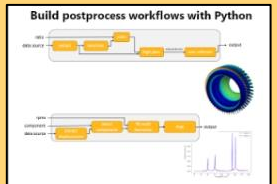
Pythonic interface to Ansys Dynamic Reporting for service and control of its database and reports



PyAnsys Geometry

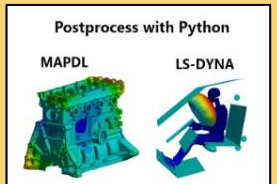
Pythonic interface to the Ansys Geometry service

Postprocessing Libraries



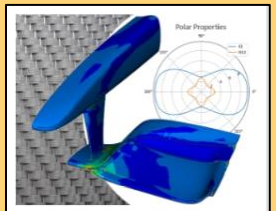
PyDPF - Core

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



PyDPF - Post

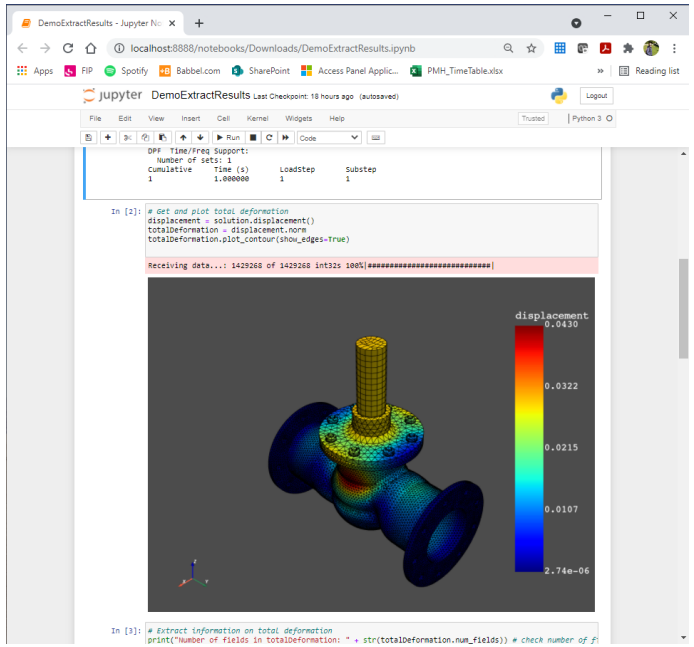
Pythonic interface to access and post process Ansys solver result files



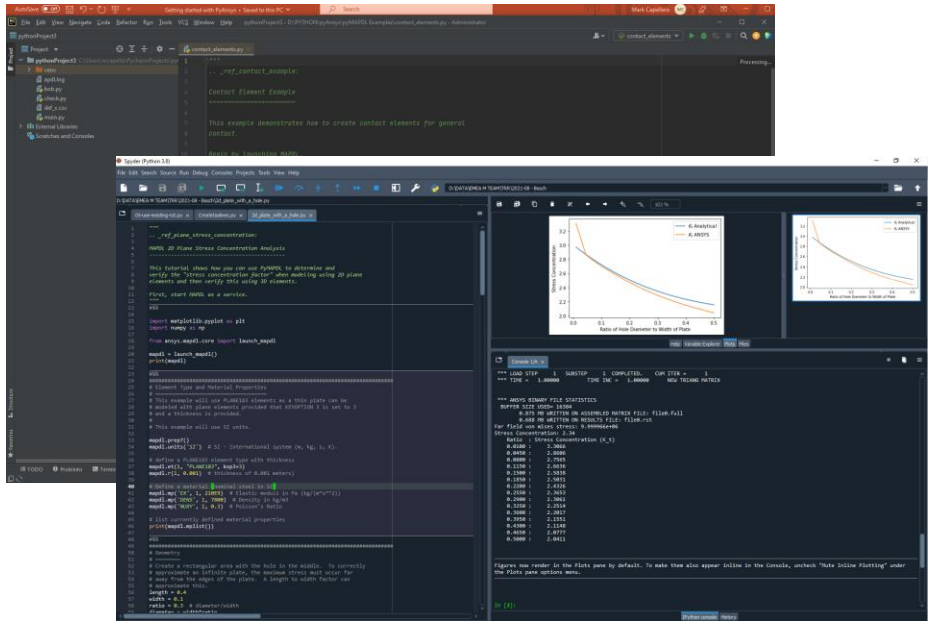
PyDPF - Composites

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

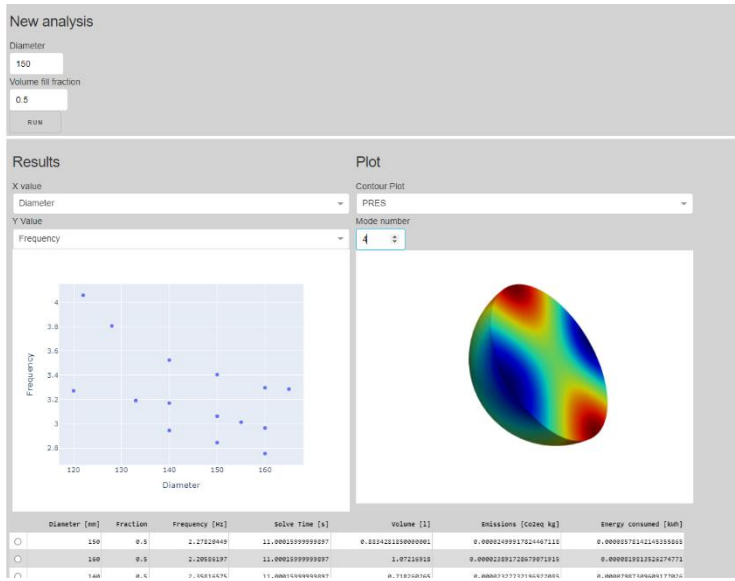
Is there a user interface (UI)?



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

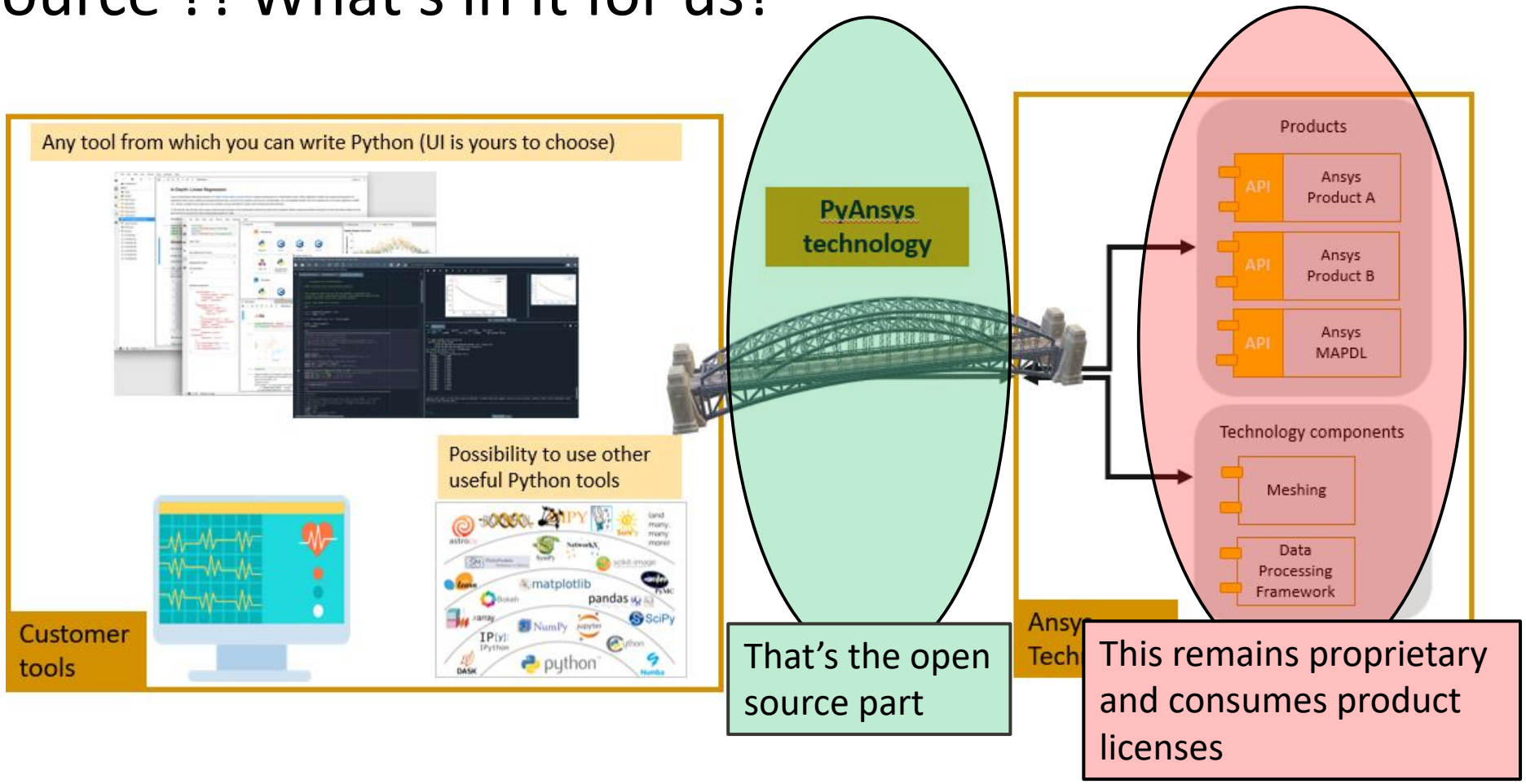


An IDE can be used (PyCharm, Spyder, …)



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

Ansys

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

Need:

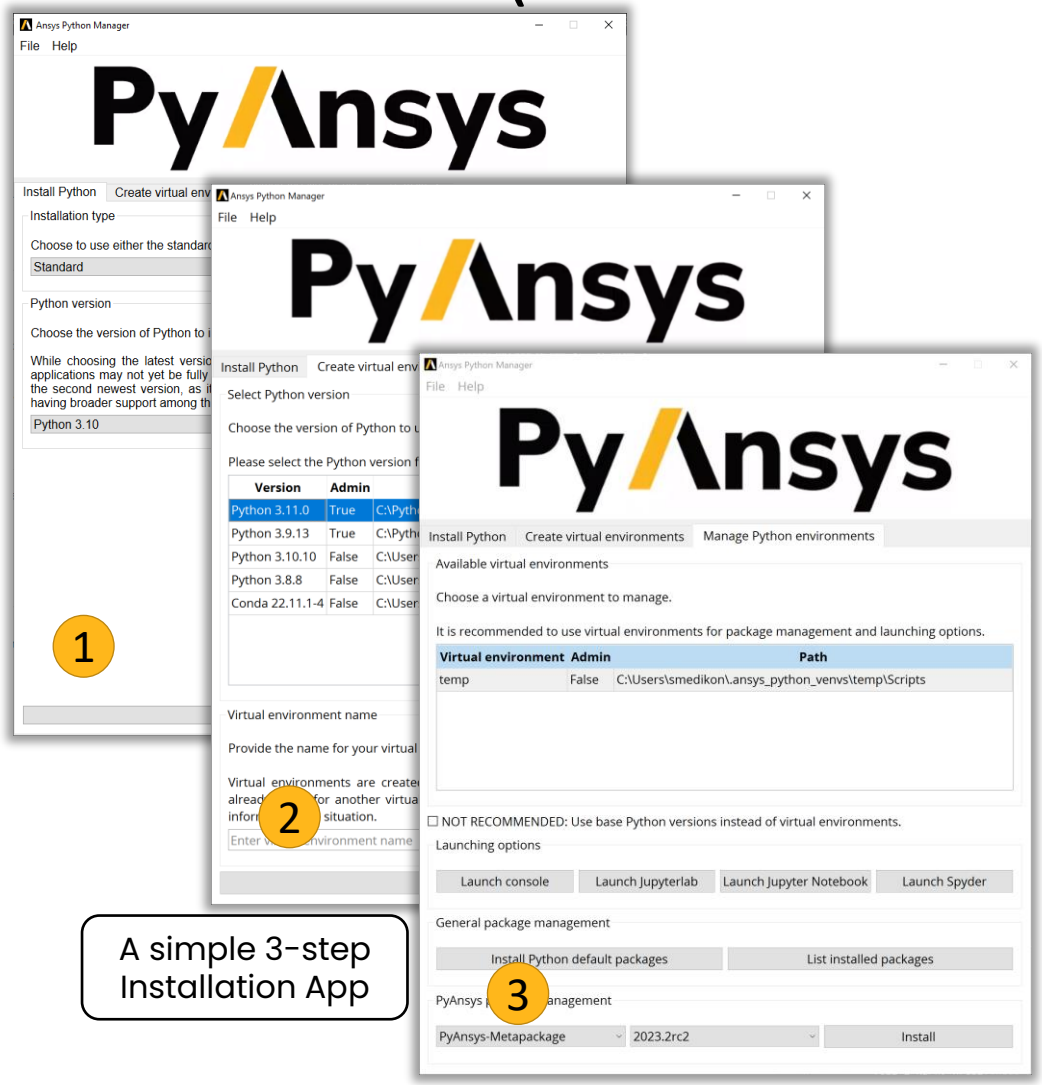
- Similar to the **Anaconda GUI installer**: Transition to a **Commercial Business Model** was a major roadblock for Ansys Customers.
- Reduce Impact for Beginner & Intermediate Python users.

Solution/Tool Created:

- A simple 3-step installer app.

- Using Ansys Python Manager:
  Download it from:
  [Releases · ansys/python-installer-qt-gui (github.com)]

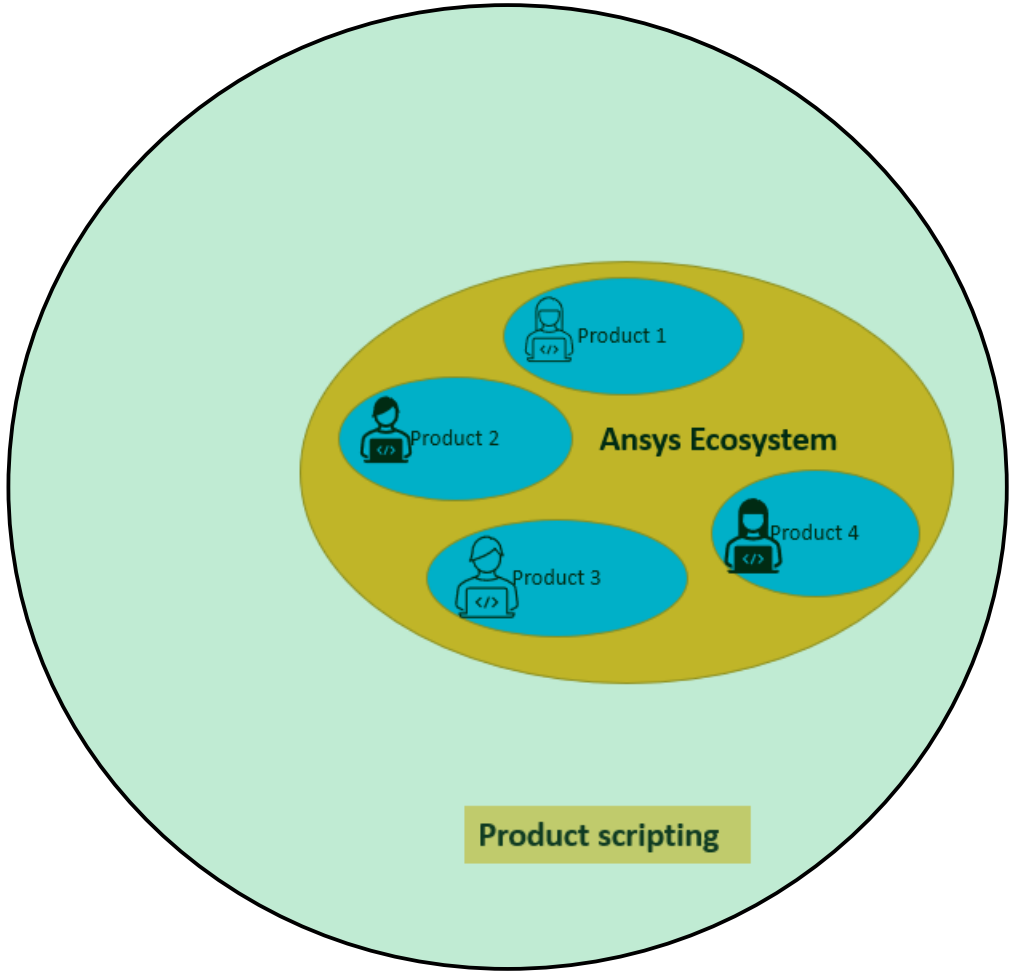
https://developer.ansys.com/ansys-python-manager

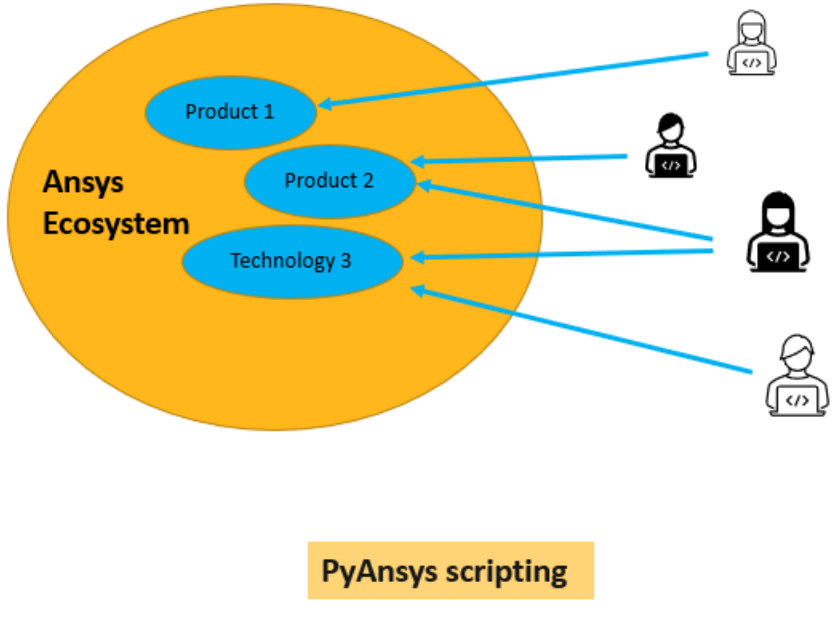


A simple 3-step Installation App

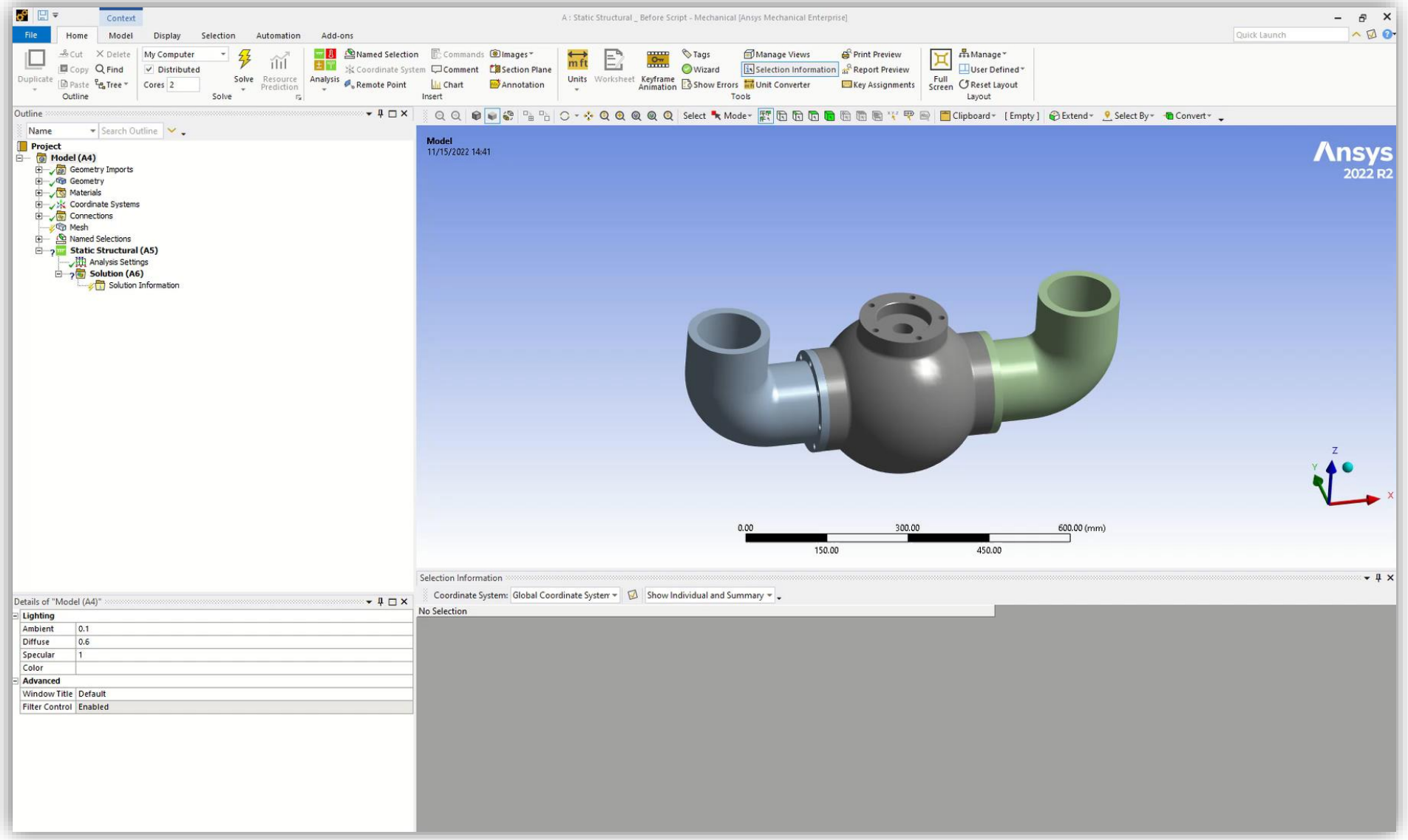
Mechanical scripting (in-product)

Product scripting





Mechanical scripting demo: complete model setup

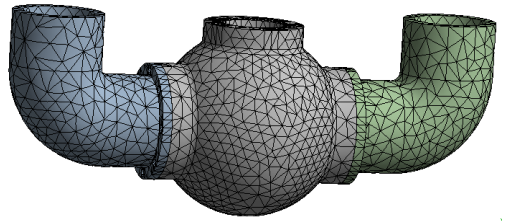


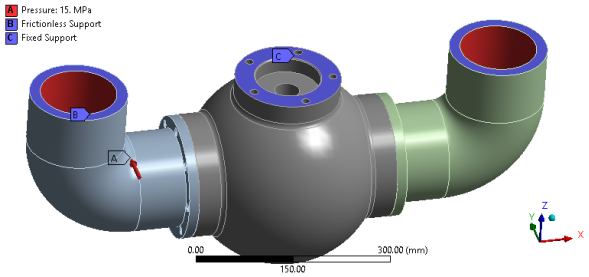
Mechanical scripting demo: complete model setup

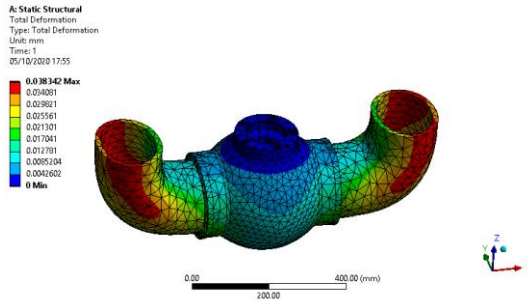
AutomateModelSetup.py

- Gray Cast Iron Assignment
- Mesh
- Fixed Support
- Frictionless Support
- Pressure
- Total Deformation
- Equivalent Stress

ResultEquivalent Stress.txt
ResultTotal Deformation.txt



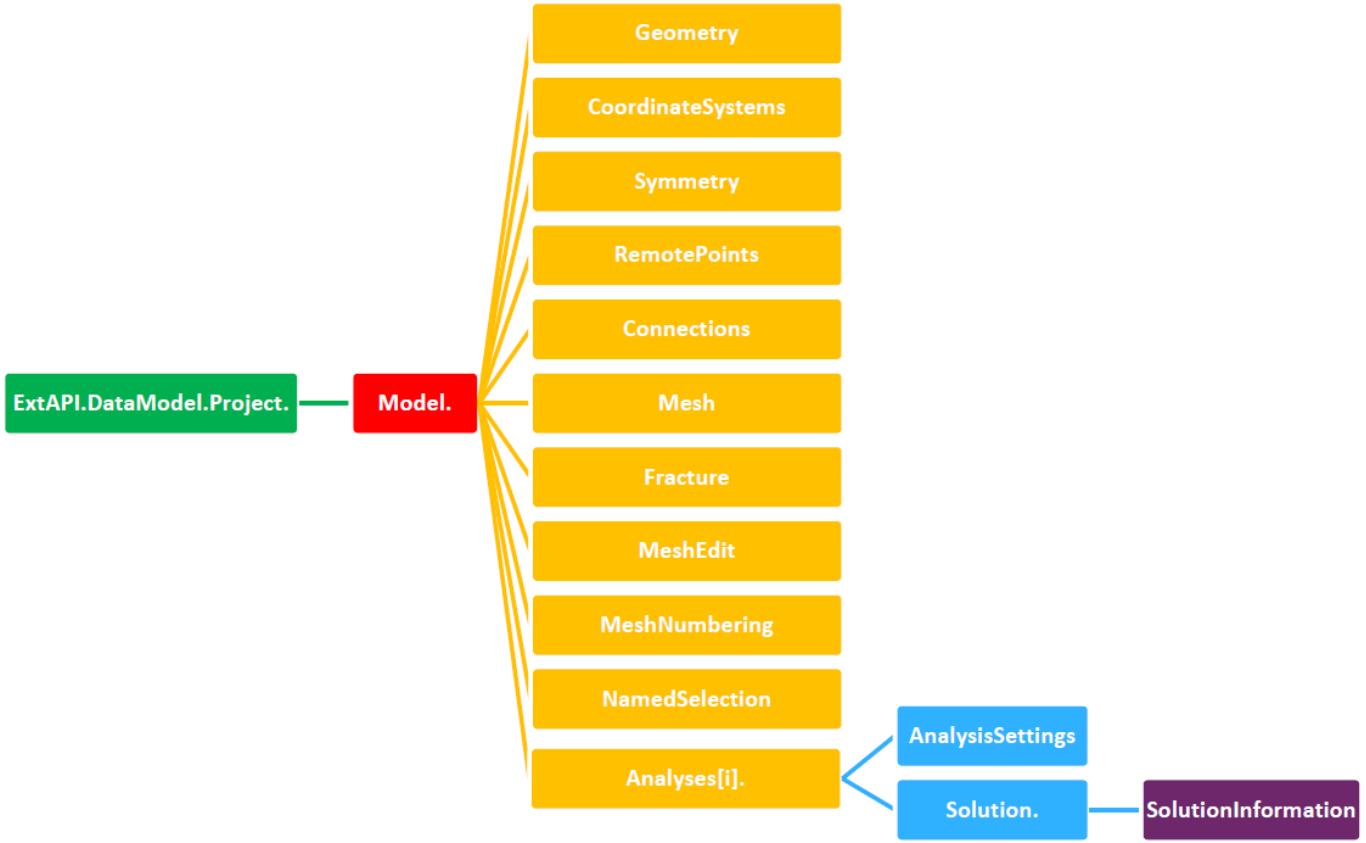


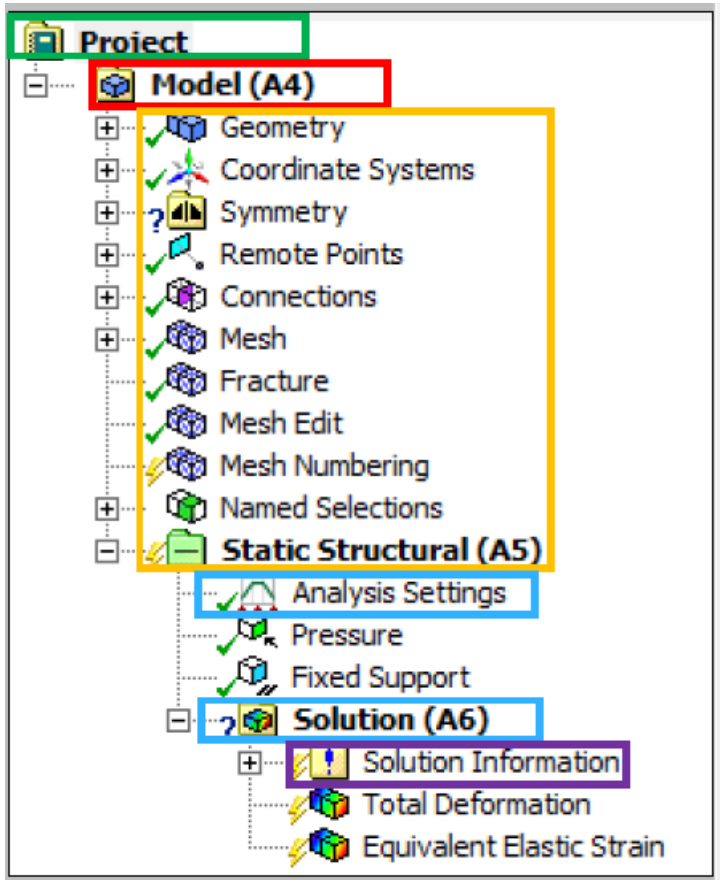


Node Number	Equivalent (von-Mises) Stress (MPa)
11136	4.9244
11137	5.2469
11138	4.3853
11139	5.9214
11140	5.5261
11141	4.1676

Mechanical Automation API

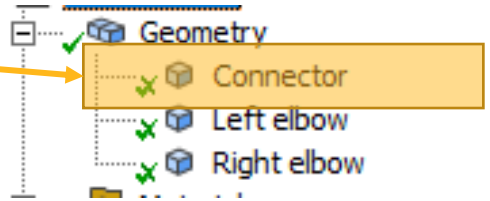
```
# Reference model
model = ExtAPI.DataModel.Project.Model
```

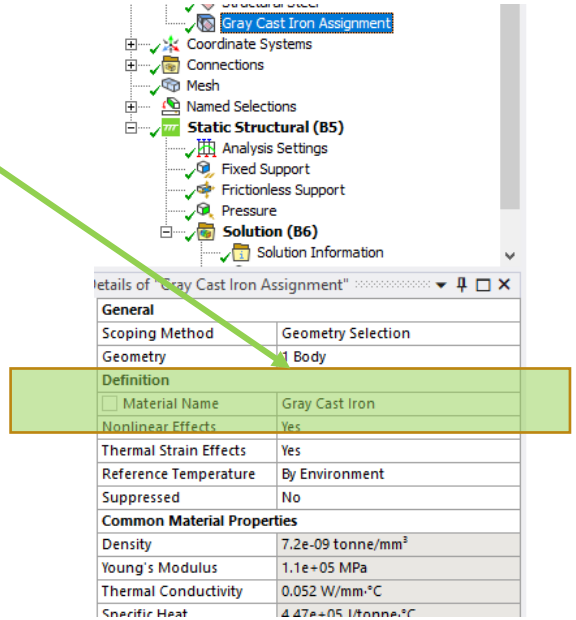


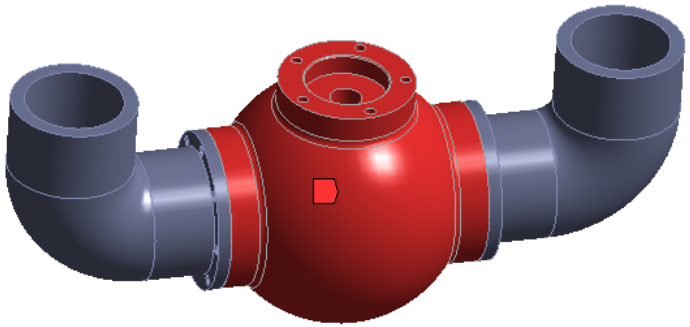


Let’s dive into the script: define material assignment

```
# Change material assignement 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()
```

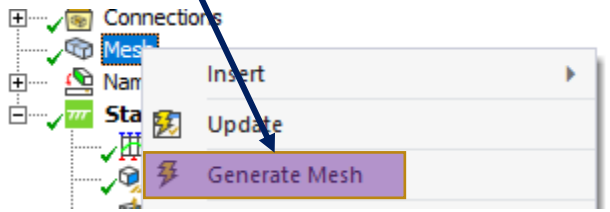


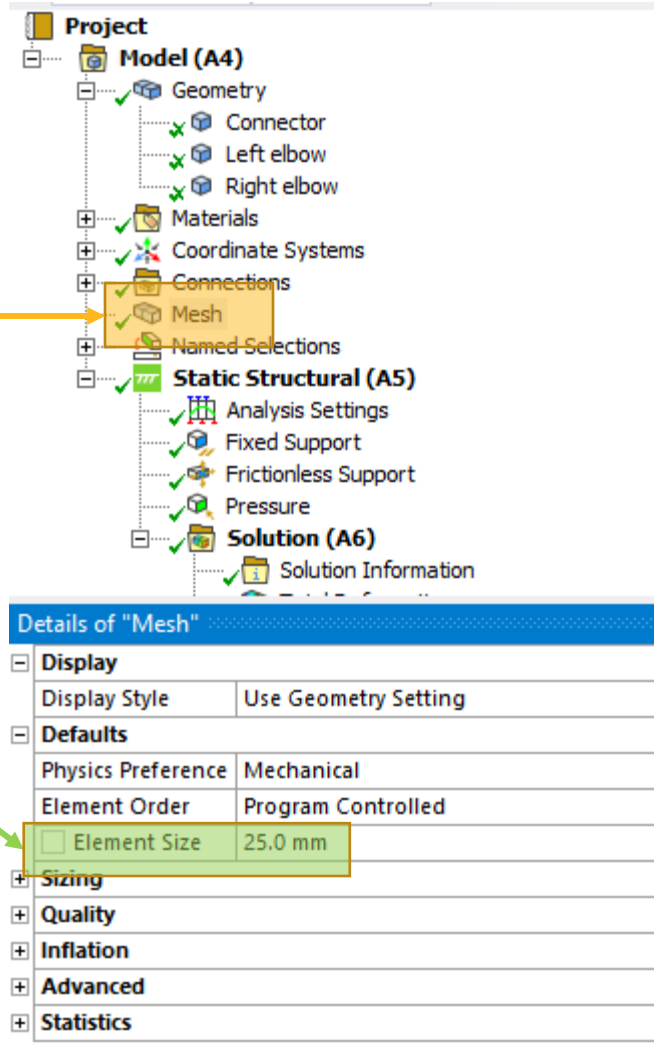




Let’s dive into the script: define mesh settings

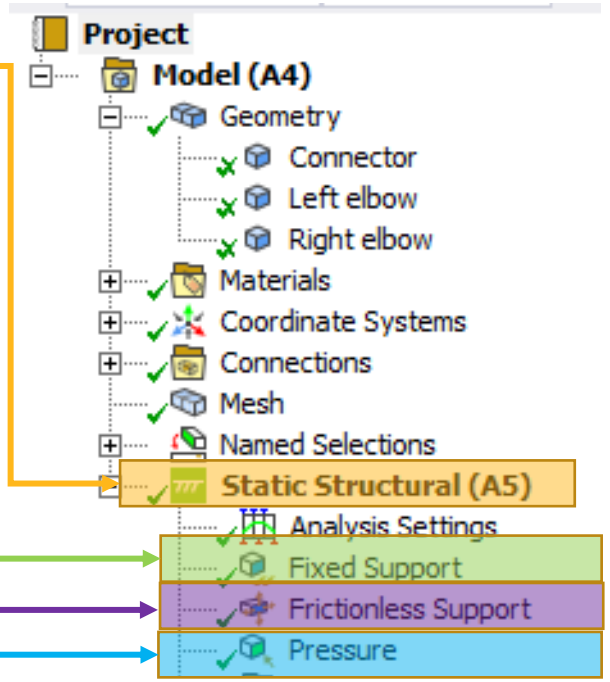
```
# Define mesh settings
mesh = model.Mesh
mesh.ElementSize = Quantity('25 [mm]')
mesh.GenerateMesh()
```

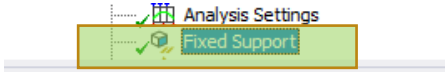




Let’s dive into the script: define boundary conditions

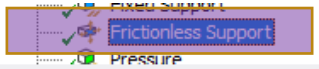
```
# Define boundary conditions:
analysis = model.Analyses[0]

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]")]
```





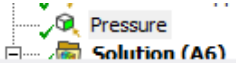
Details of "Fixed Support"

Scope	
Scoping Method	Named Selection
Named Selection	NSFixedSupportFaces
Definition	
ID (Beta)	92
Type	Fixed Support
Suppressed	No



Details of "Frictionless Support"

Scope	
Scoping Method	Named Selection
Named Selection	NSFrictionlessSupportFaces
Definition	
ID (Beta)	94
Type	Frictionless Support
Suppressed	No



Details of "Pressure"

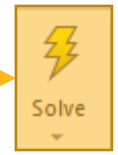
Scope	
Scoping Method	Named Selection
Named Selection	NSInsideFaces
Definition	
ID (Beta)	96
Type	Pressure
Define By	Normal To
Applied By	Surface Effect
Loaded Area	Deformed
Magnitude	Tabular Data
Suppressed	No
Tabular Data	
Independent Variable	Time

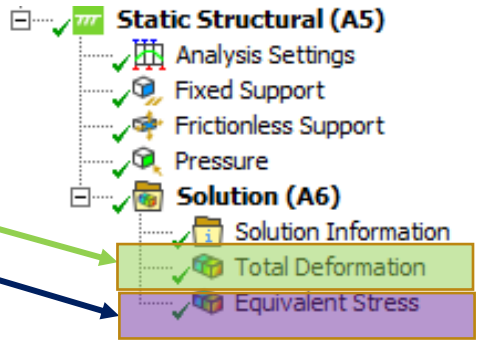
Tabular Data

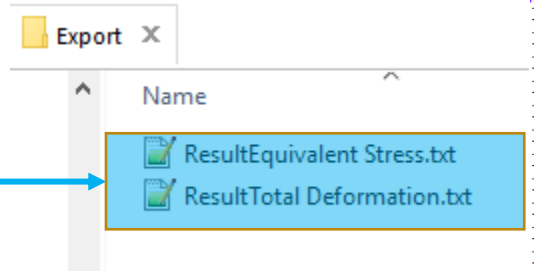
	Steps	Time [s]	Pressure [MPa]
1	1	0.	0.
2	1	1.	15.
*			

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)
```



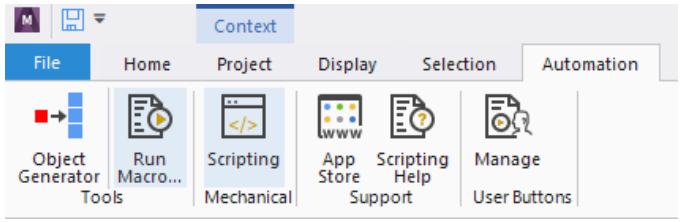


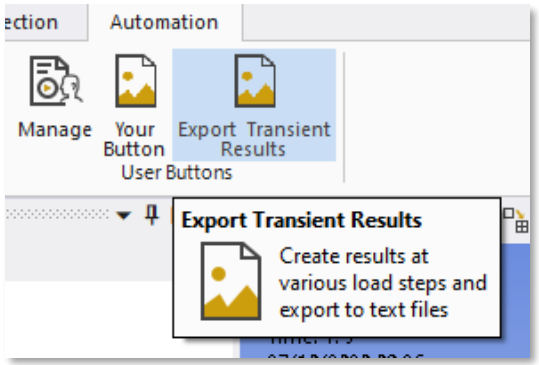



```
Node Number Equivalent (von-Mises) Stress (MPa)
11136    4.9244
11137    5.2469
11138    4.3853
11139    5.9214
11140    5.5261
11141    4.1676
11142    2.8028
11143    3.7985
11144    20.445
11145    22.654
11146    27.851
11147    36.228
```

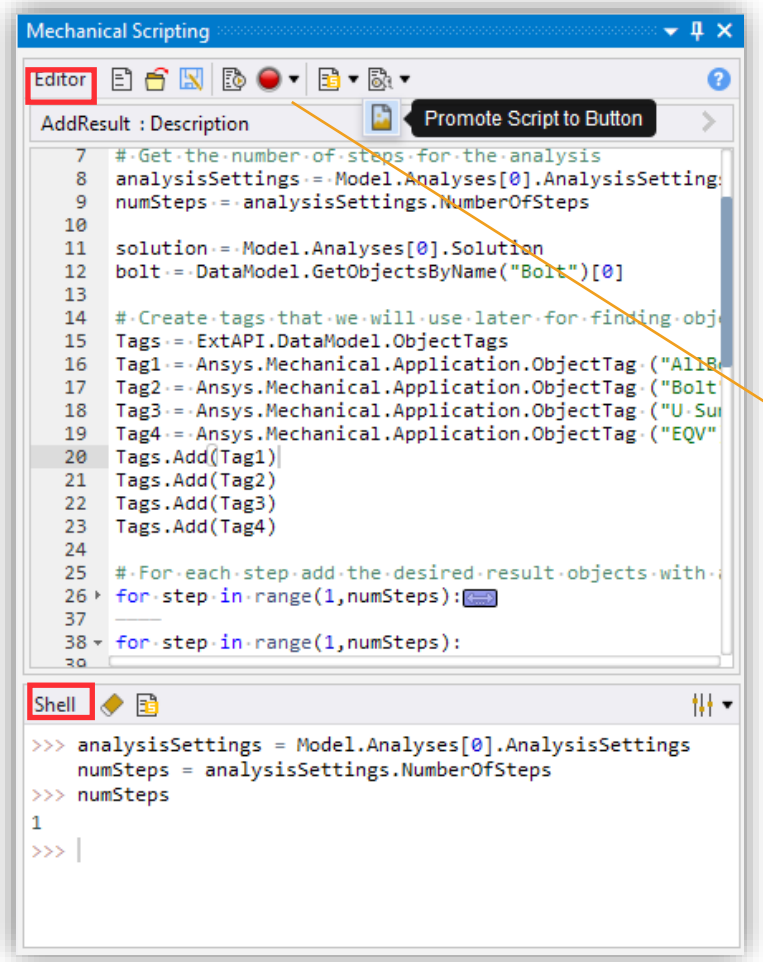
Mechanical Automation Scripting

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

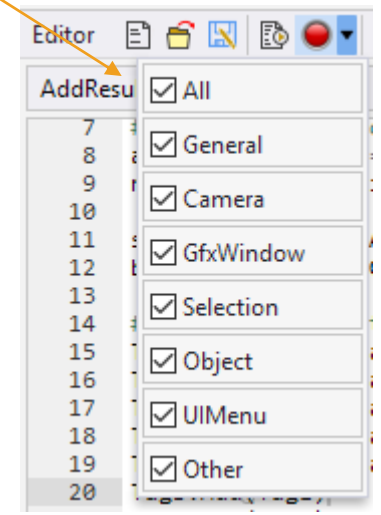




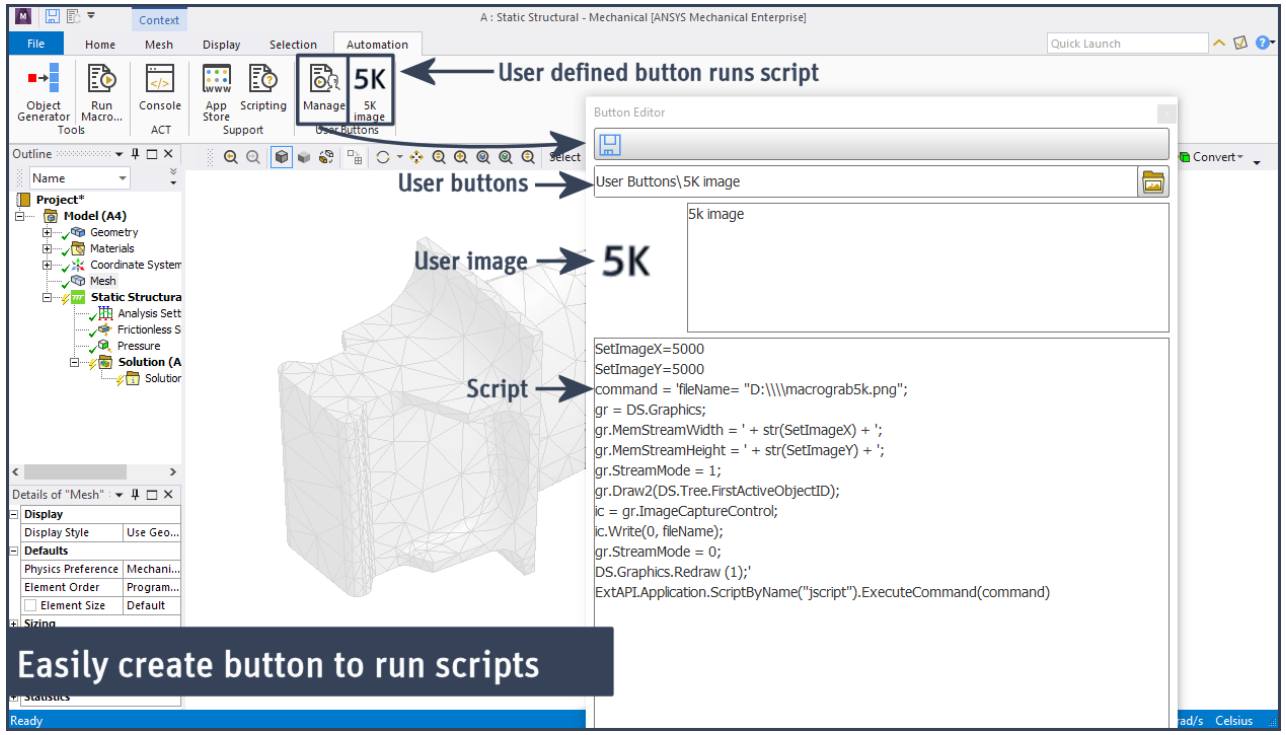
User buttons



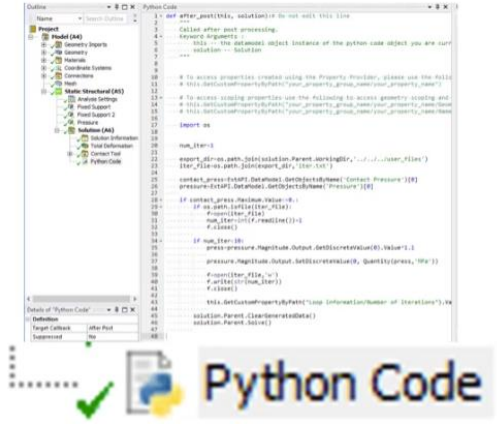
Record button with control of how much to journal

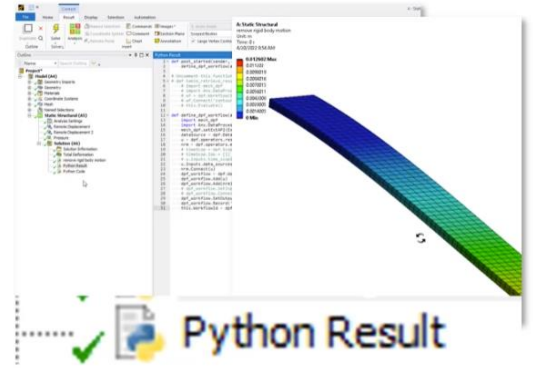


More advanced usage: 1-time execution, Python Code & Python Result Objects



Save script to button
(Script executed when clicked on)

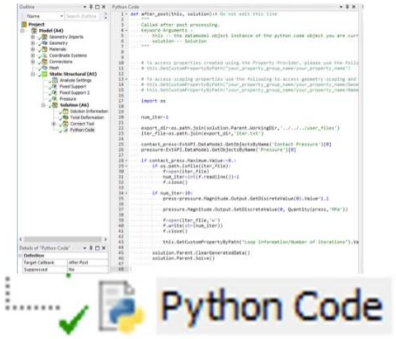


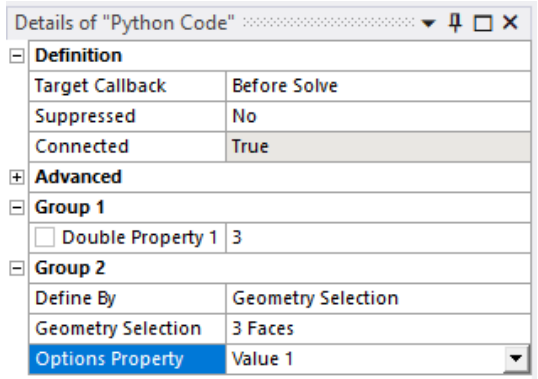


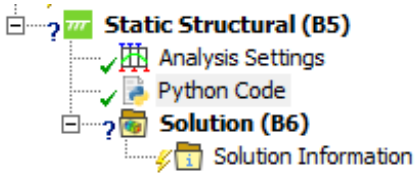
Execute code automatically during simulation

Within the simulation tree to execute with each simulation

Python Code Object

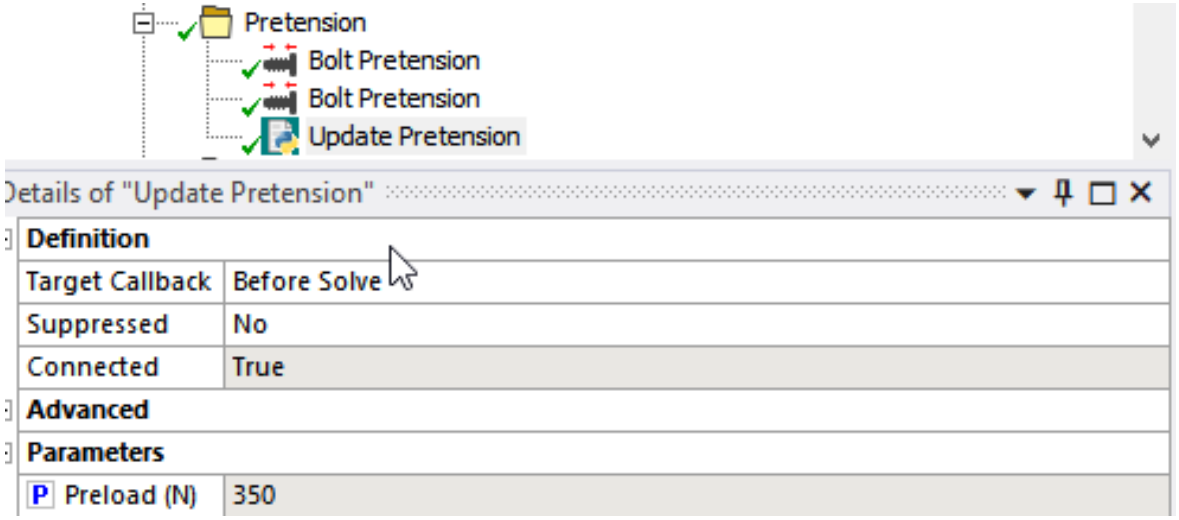






- Execute Python code in response to events occurring in Mechanical (“target callback”)
- Use Python code to inject MAPDL commands into the solver input
- Define custom properties

Python Code example: Create new parameters for design variations



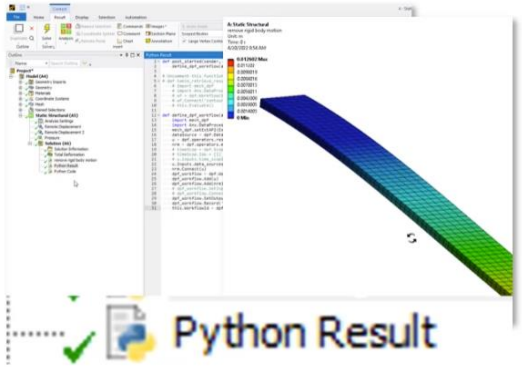
Preload value available as parameter

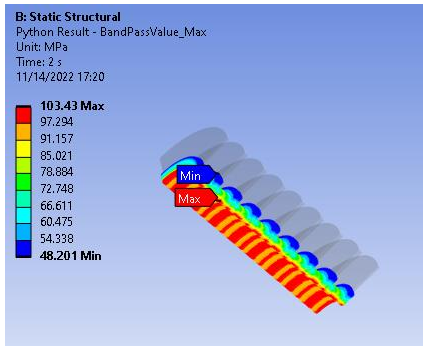
- *Update can be applied to one or more pretension objects*
- *Called before solving to update value sent to the solver*

```
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



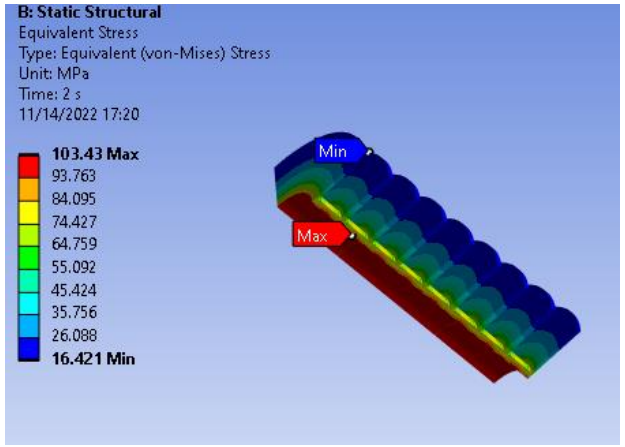


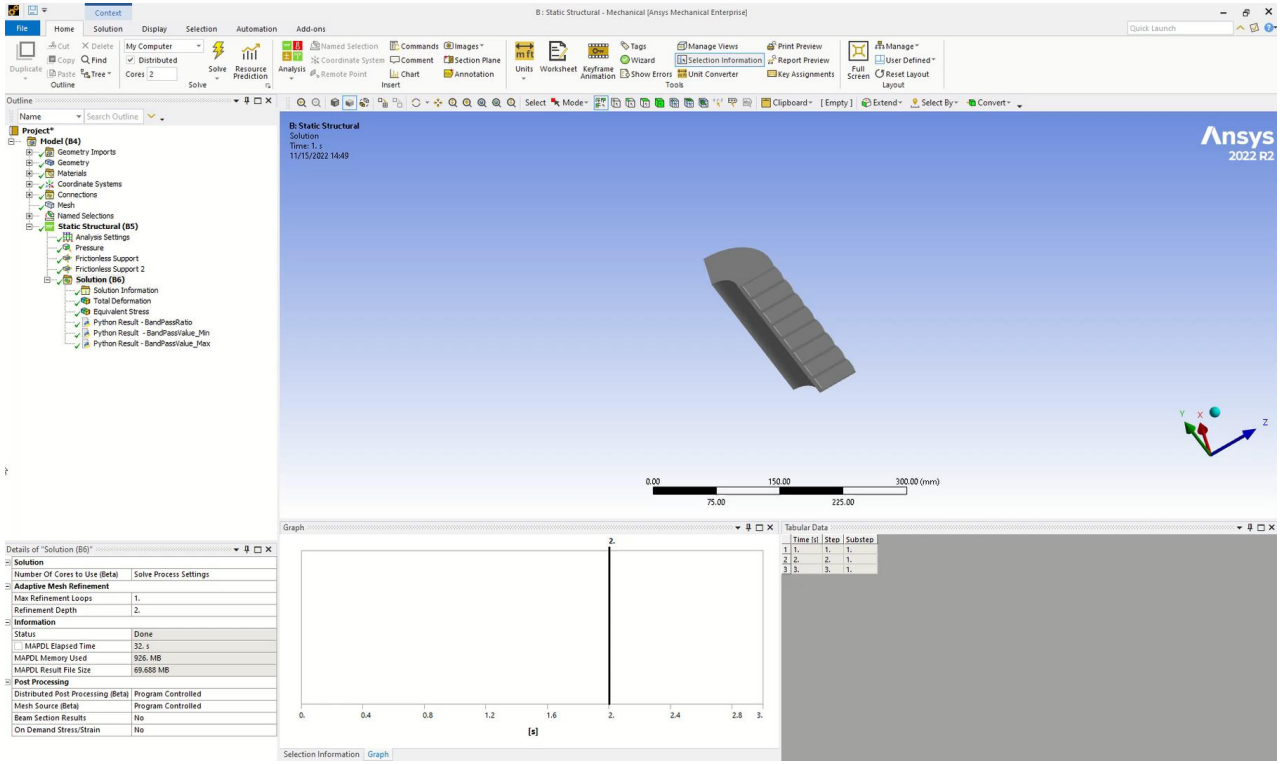
Details of "Python Result - BandPassValue_Max"	
Definition	
Suppressed	No
Connected	True
Advanced	
Script Execution Scope	__python_code_89__
Engine Type	Iron Python
Group 1	
Stress Min Value	48

- 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)

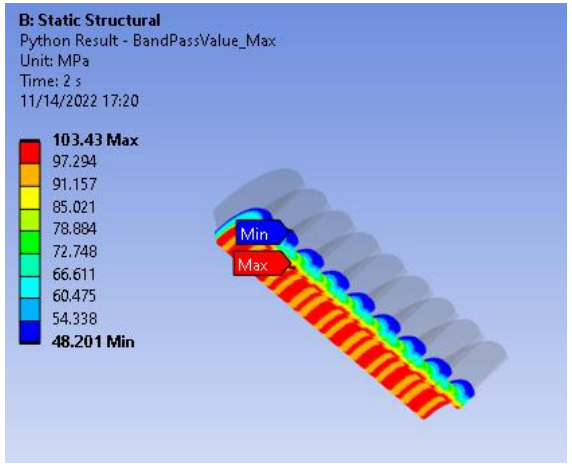
Python Result example: stress threshold

Standard result: plot stress on complete model





Python result: plot stress above a specified value



Details of "Python Result - BandPassValue_Max"

Definition	
Suppressed	No
Connected	True
Advanced	
Script Execution Scope	__python_code_89__
Engine Type	Iron Python
Group 1	
Stress Min Value	48

Introduction to PyMechanical

Ansys

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

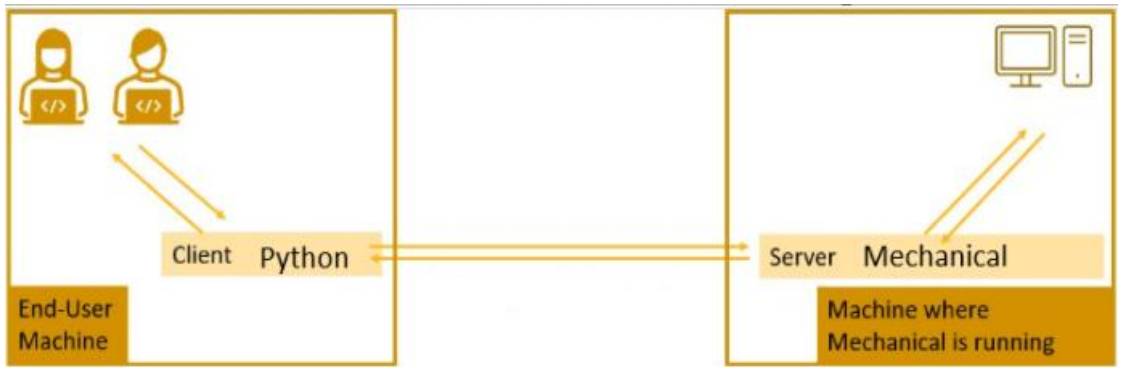
Embedded instance

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

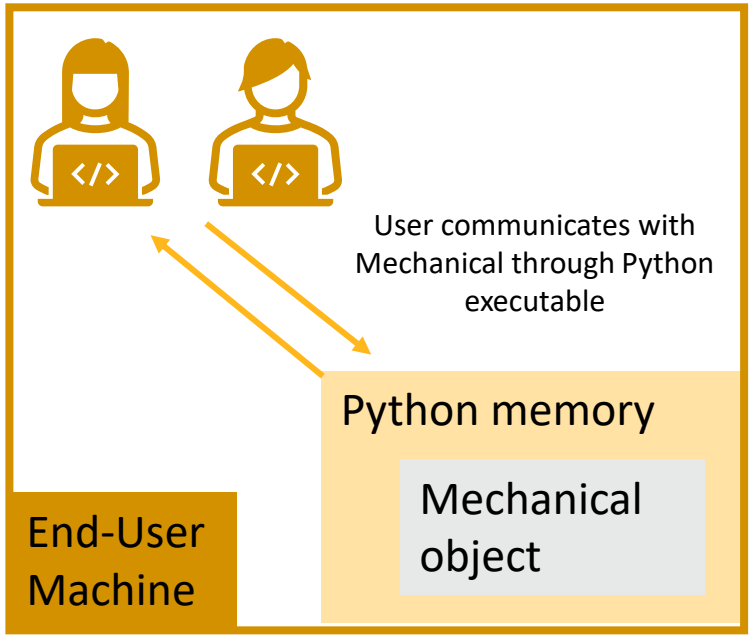
Go here for an explanation of the Mechanical application architecture and why both interfaces are offered

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:

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

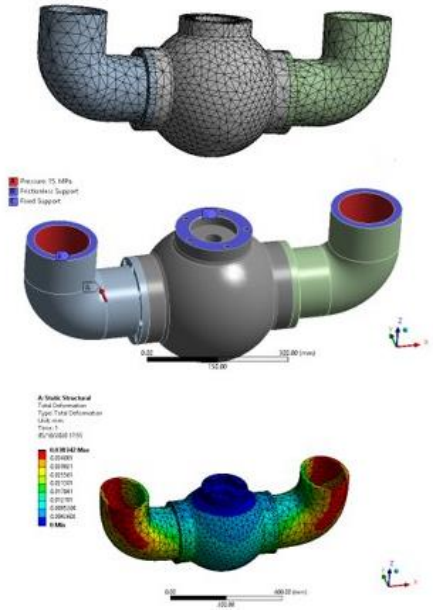
```python
import os
from ansys.mechanical.core import App, global_variables

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

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

Type commands as if in the Mechanical console

Remote session

```python
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

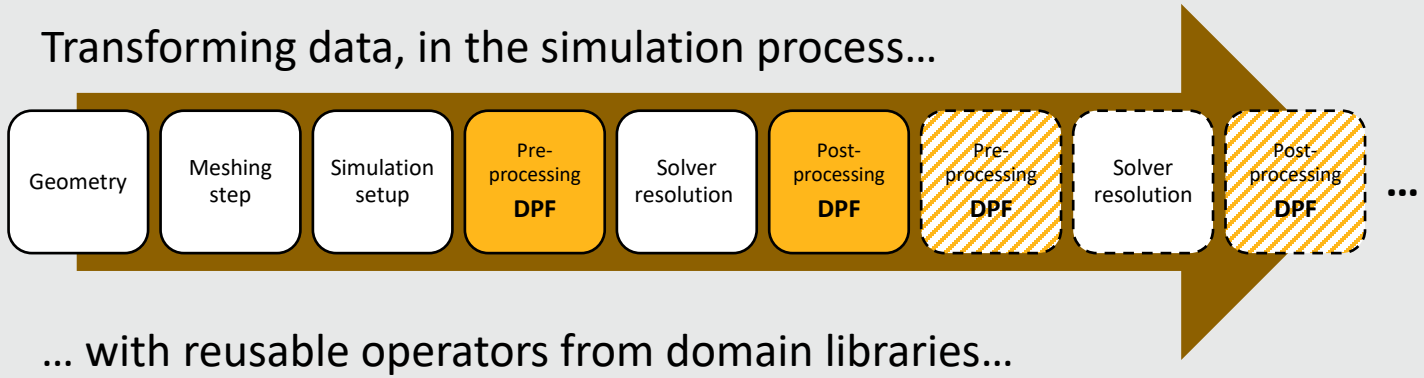
Introduction to PyDPF

Ansys

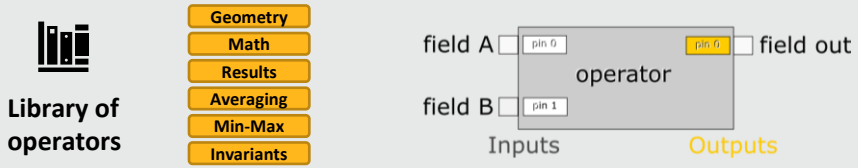
Ansys Data Processing Framework (DPF)

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

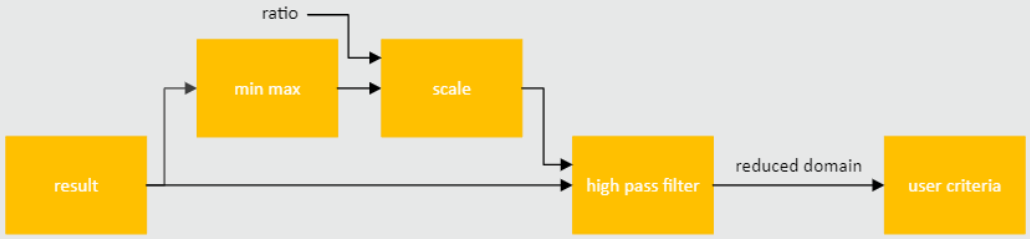
Transforming data, in the simulation process…



… 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
```

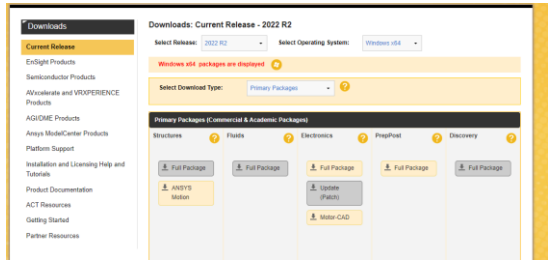
```
pip install ansys-dpf-post
```

- 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

.RST
.RTH
.d3plot

.cas
.dat
.vtk
.hdf5

Access
Simulation Data

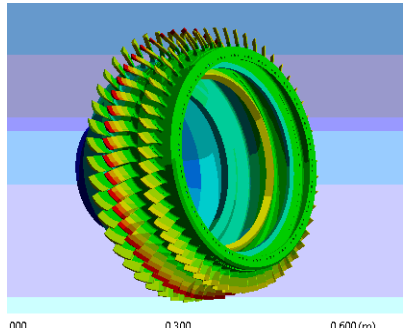
Manipulate/ Transform
Simulation Data

Visualize/ Export
Output **Field**

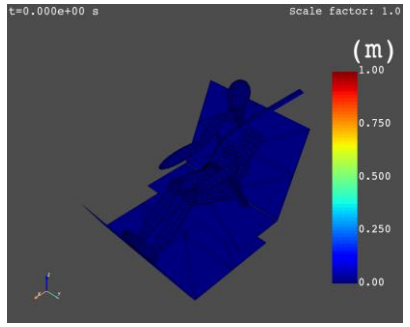
Physics Agnostic
Performance
Remote/Distributed Access
Pythonic
Customizable
Open Source
Dedicated Libraries (eg: numpy, matplotlib)
Plug & Play

Create custom plots
Facilitate data transfer in multi-tool simulations

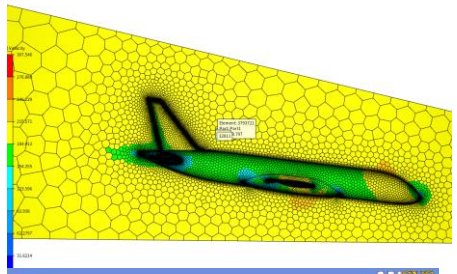
MAPDL



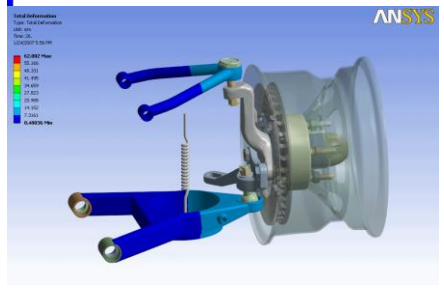
LSDyna



Fluids



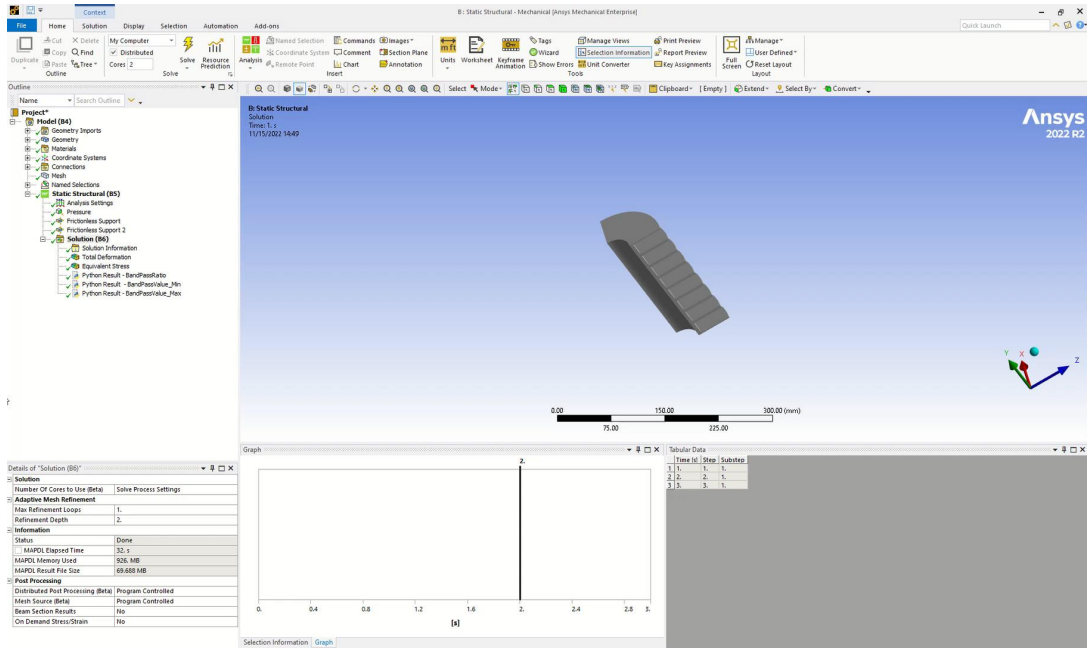
Motion



Where can DPF be used ?

In Mechanical

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

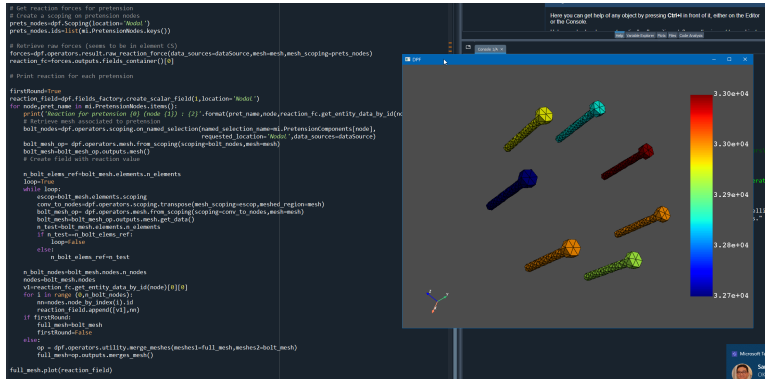


With PyAnsys

Generic client API

PyDPF-Core

Highly customizable, physics agnostic workflow-based API

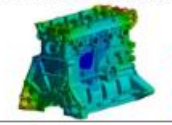


Dedicated libraries

PyDPF-Composite

PyDPF-Post

Post-processing physics-based API



PyDPF-Sound

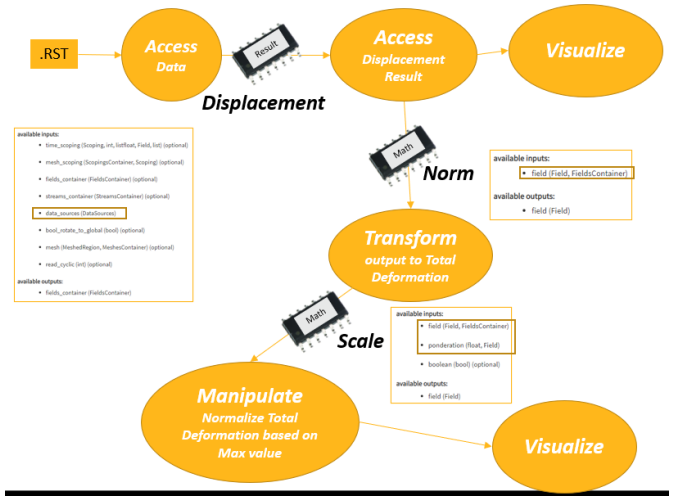
PyDyna

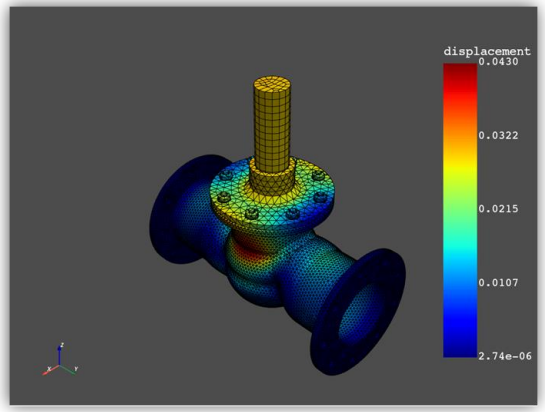
Using DPF in PyAnsys

- 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

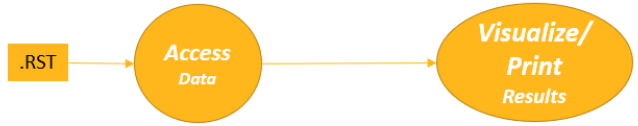
Exposure of complete DPF technology: chain operators together to create a customized postprocessing workflow





DPF Post

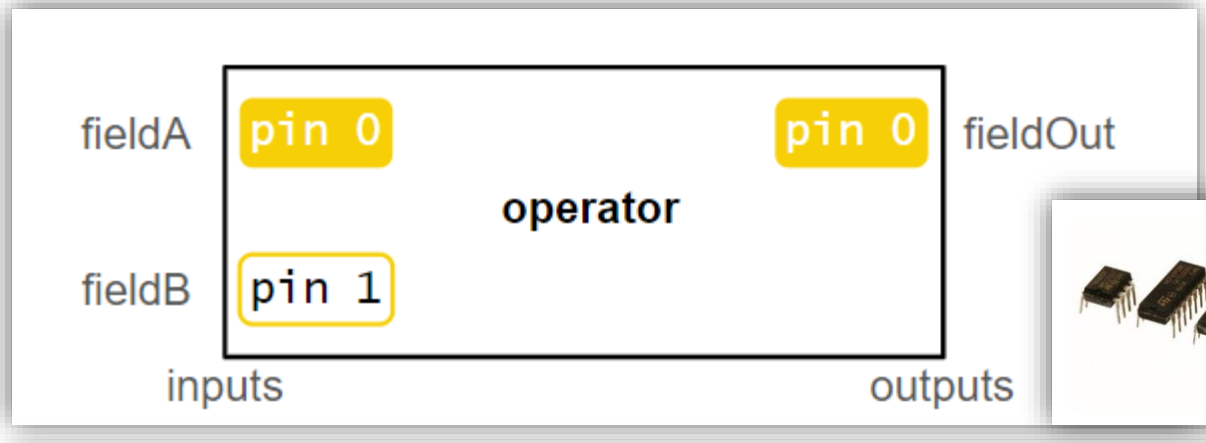
Simple wrapper functions to main DPF capabilities

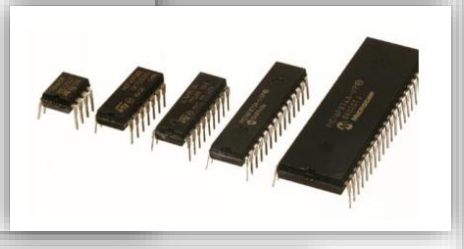


PyDPF-Core

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

Use PyAnsys to:
- Manipulate and create new data easily thanks to DPF-Core

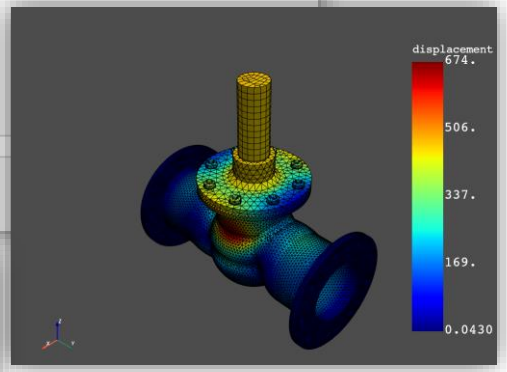


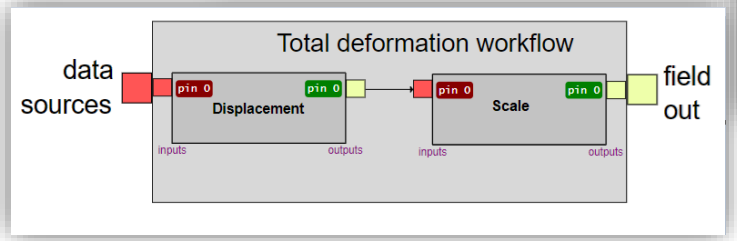


```python
## Use DPF core to manipulate data
from ansys.dpf import core

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

```python
op = core.Operator('scale') # operator instanciation
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)
```

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





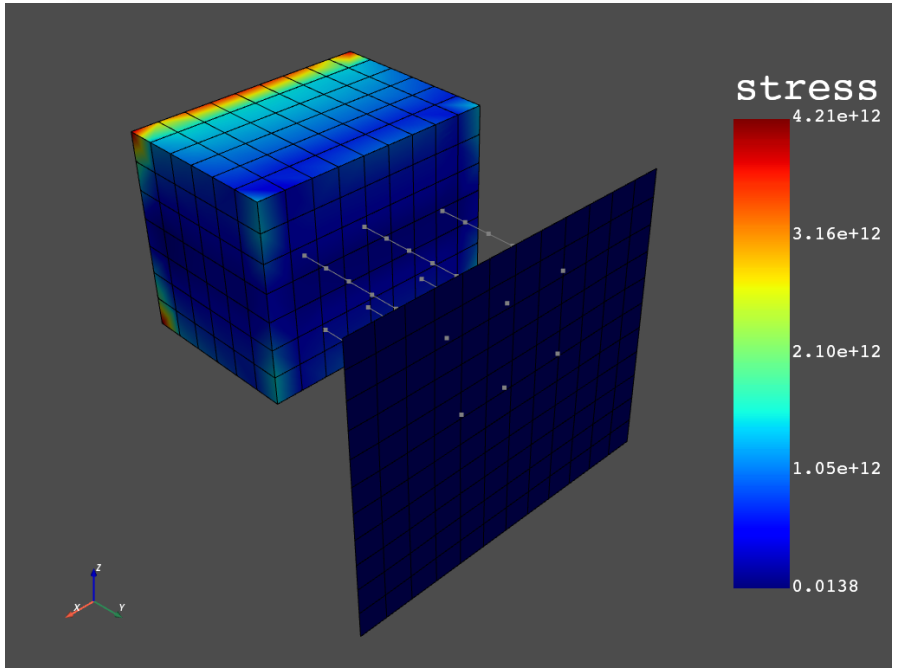
DPF can access data from **solver result files** as well as several neutral formats (**csv**, **hdf5**, **vtk**, etc.).

PyDPF-Post

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

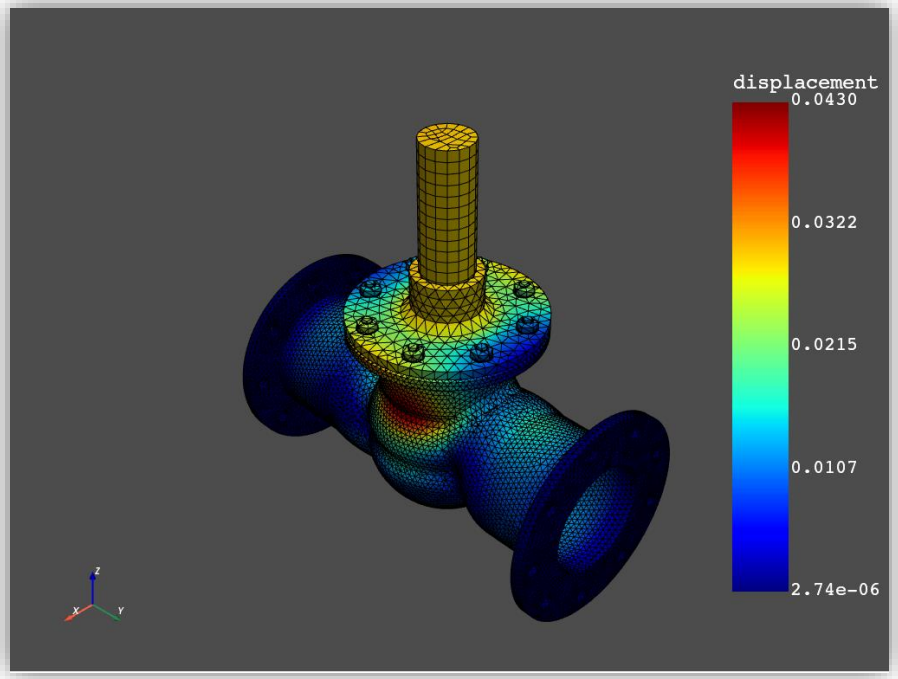
Plot the amplitude contour

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



Use PyAnsys to:
- Plot data in .rst file thanks to DPF-Post

```
# 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

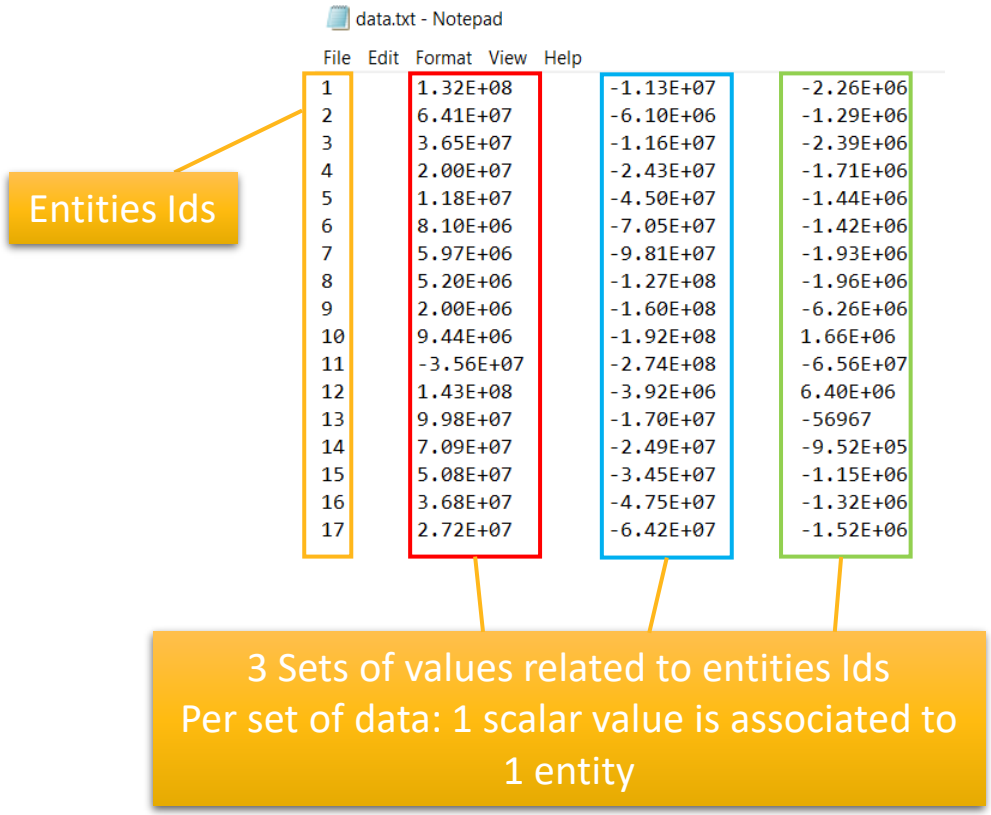
- **Data source:** One or more files containing analysis results.
- **Field:** Main simulation data container.
- **Field container:** For a transient, harmonic, modal, or multi-step static analysis, a set of fields, with one field for each time step or frequency.
- **Location:** Type of topology associated 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.

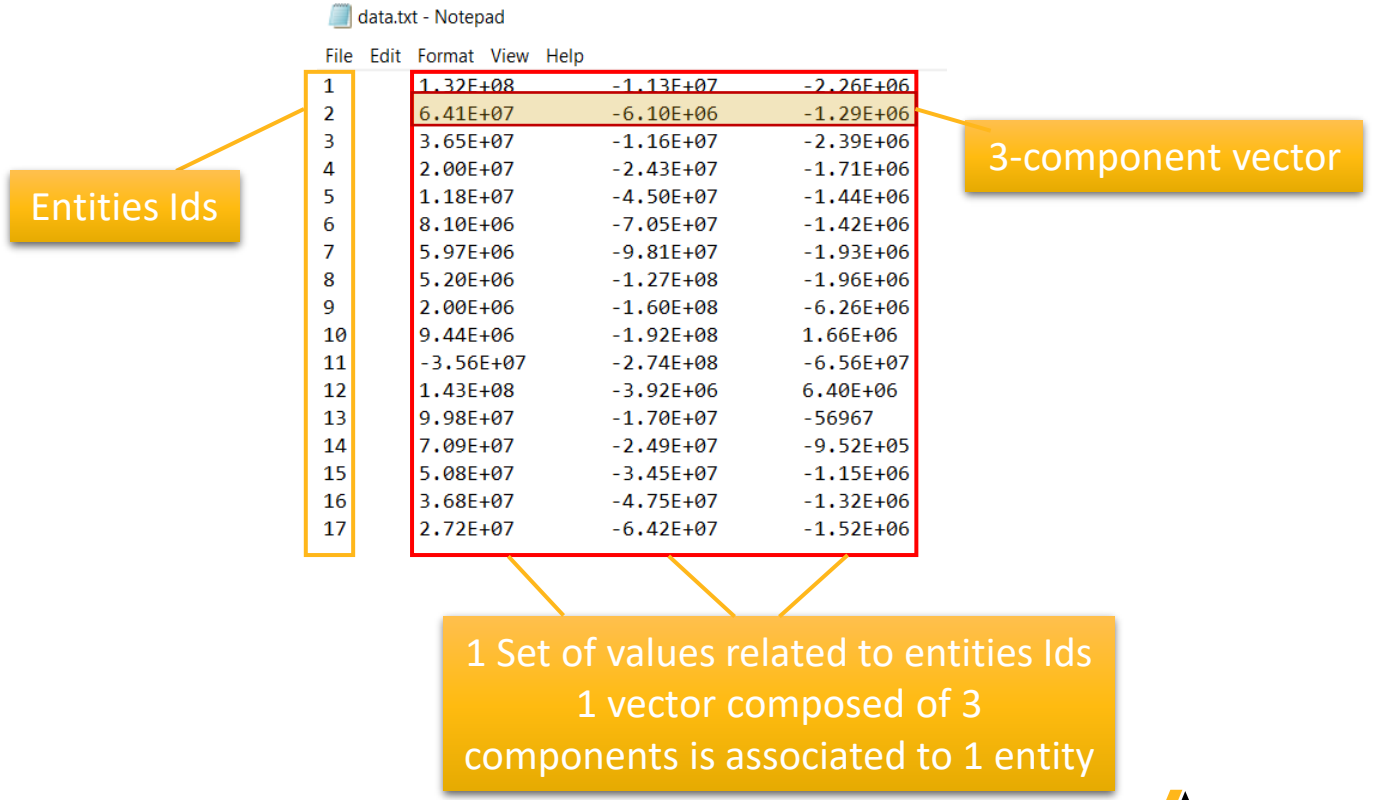
Simulation Data

- 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:



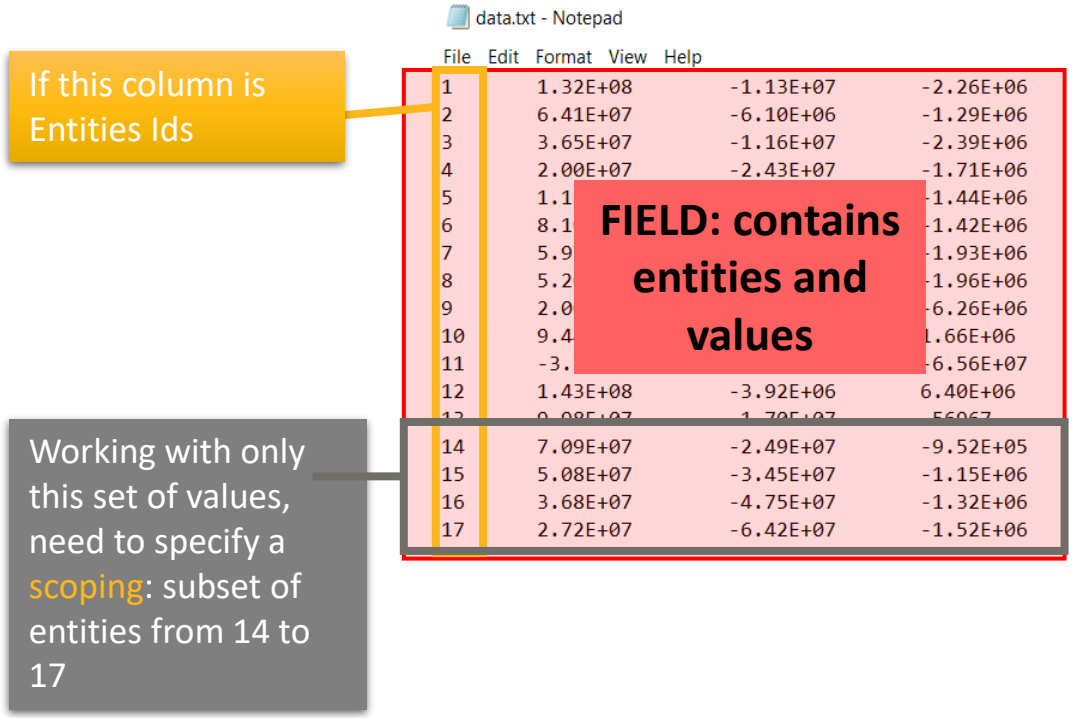


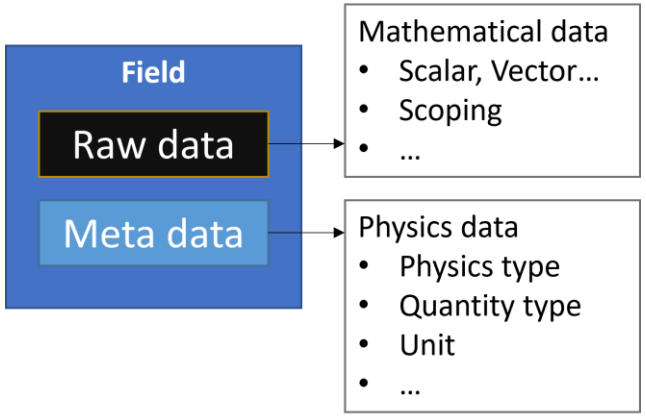
Simulation Data in DPF: Fields & Scopings

- 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

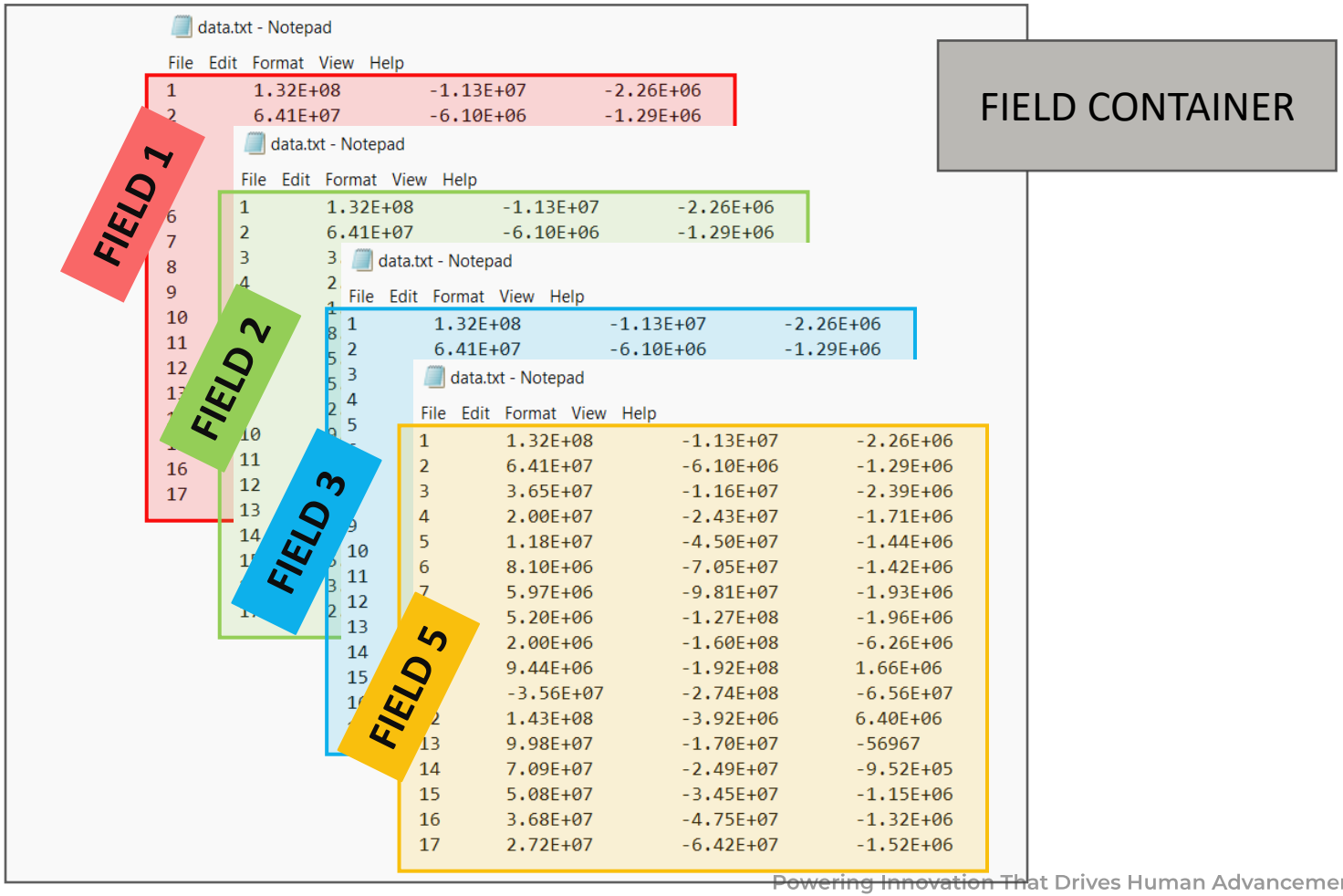
- **Scoping** can be any type, for example, a subset of:
  - Mesh Ids
  - Geometric entities Ids
  - Time domain
  - Frequency domain





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.

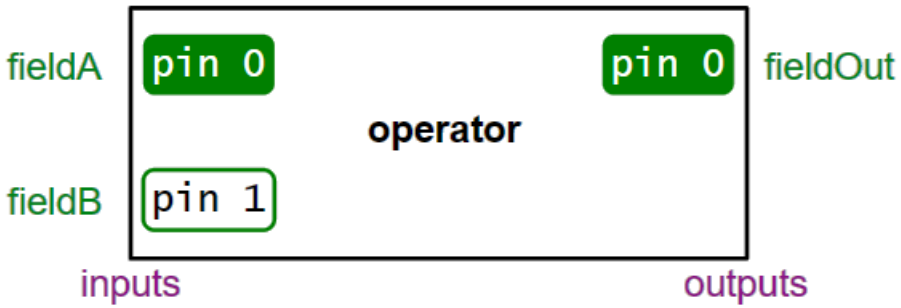
- **Streams** is an entity containing the data sources.
  - Defining **Streams** opens the data files. They stay opened and they keep some data in cache to make the next evaluation faster.
  - Using Streams is convenient when using large files. It saves time when opening/closing files.
  - The streams should be released to close the files.

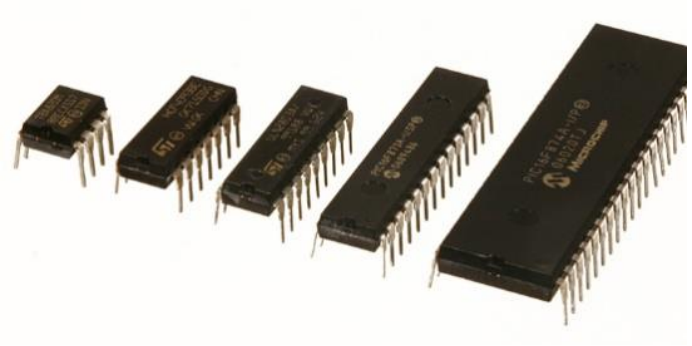
- **Streams** are associated to a **DataSource**

Transform Data: Operators

Operators

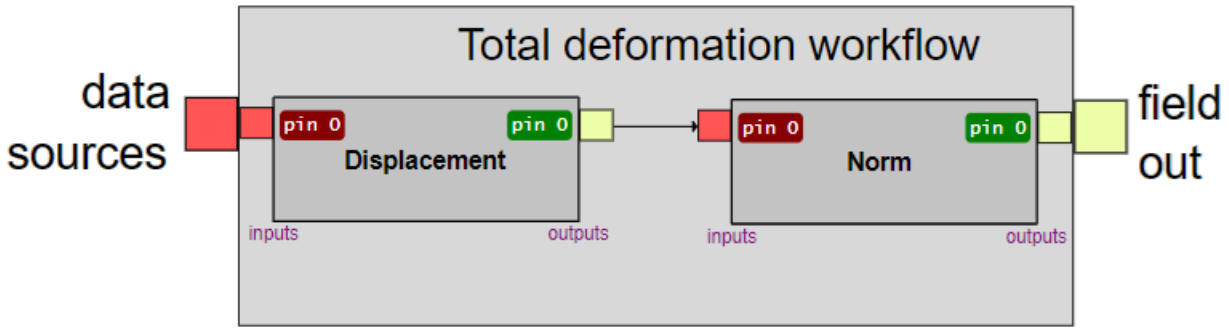
- The Operator is the only object used to **create and transform the data**.
- It can be seen as an integrated circuit in electronics with a range of pins in input and in output.
- Those pins allow the user to pass on his information to each operator. When the operator is evaluated, it will process the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

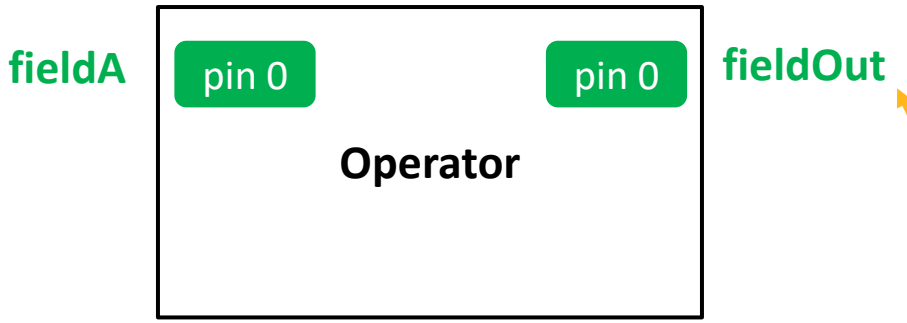
utility

- 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.



Operator's output(s) are fields or fields containers.

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

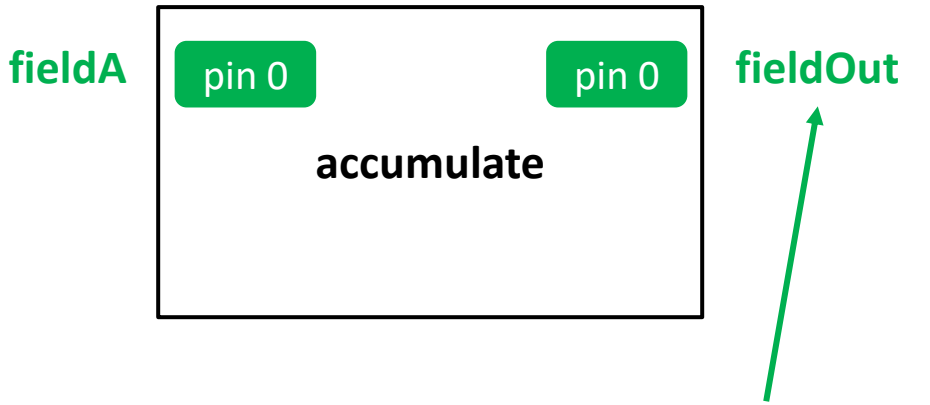
pin 0 **field** (field)

Configurations

mutex (bool) **false**

Scripting

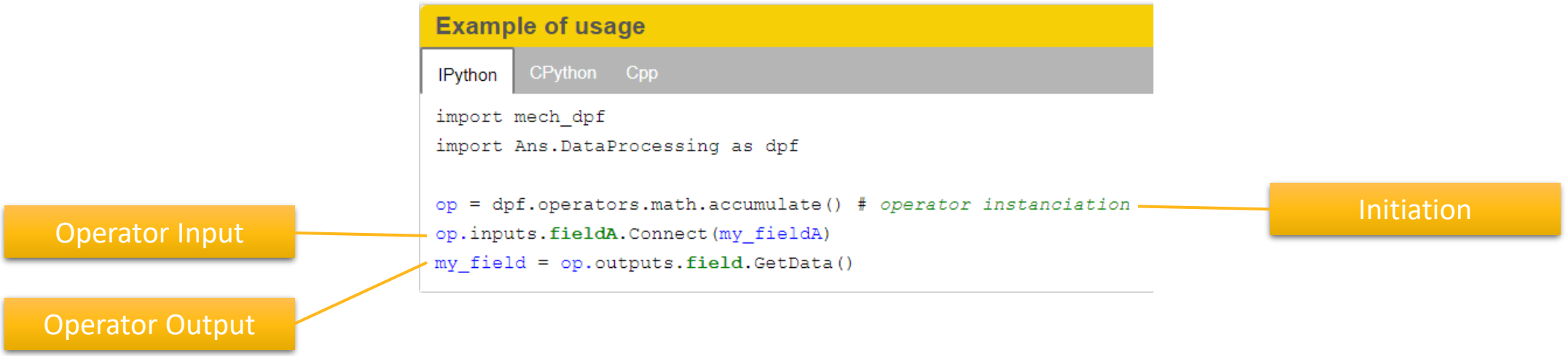
category: math
plugin: core
scripting name: accumulate



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

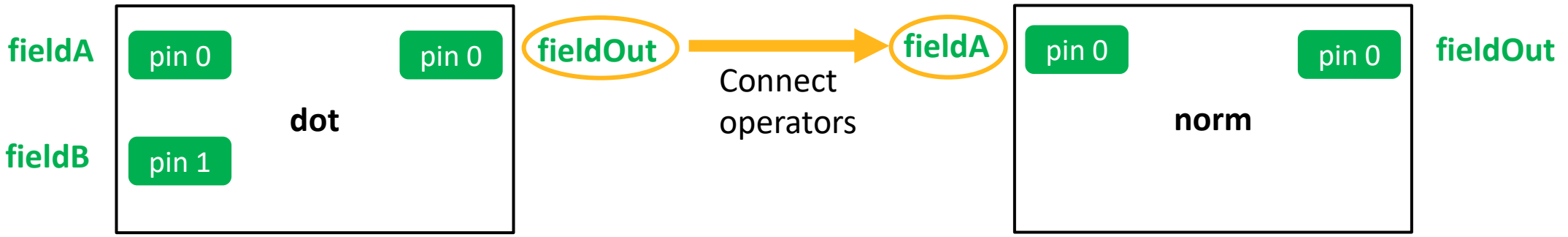
Operators

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

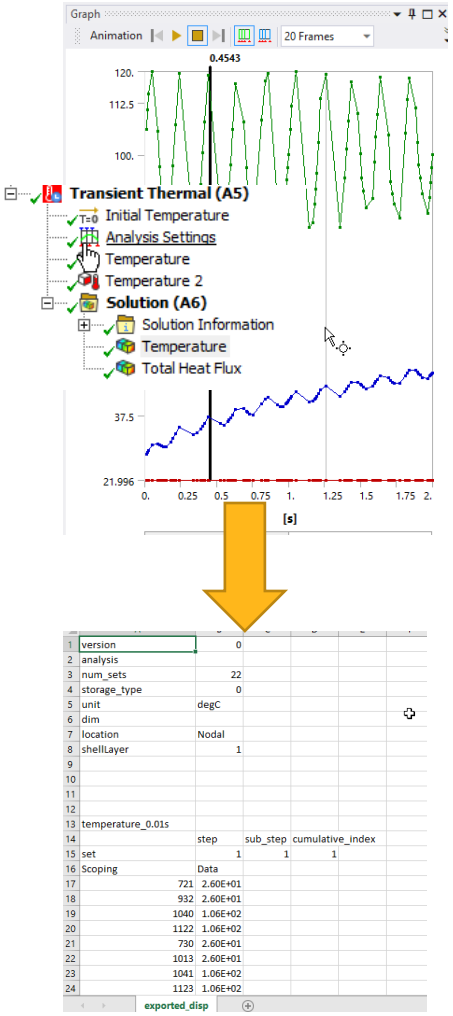


Chaining Operators: Workflow Creation

- 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

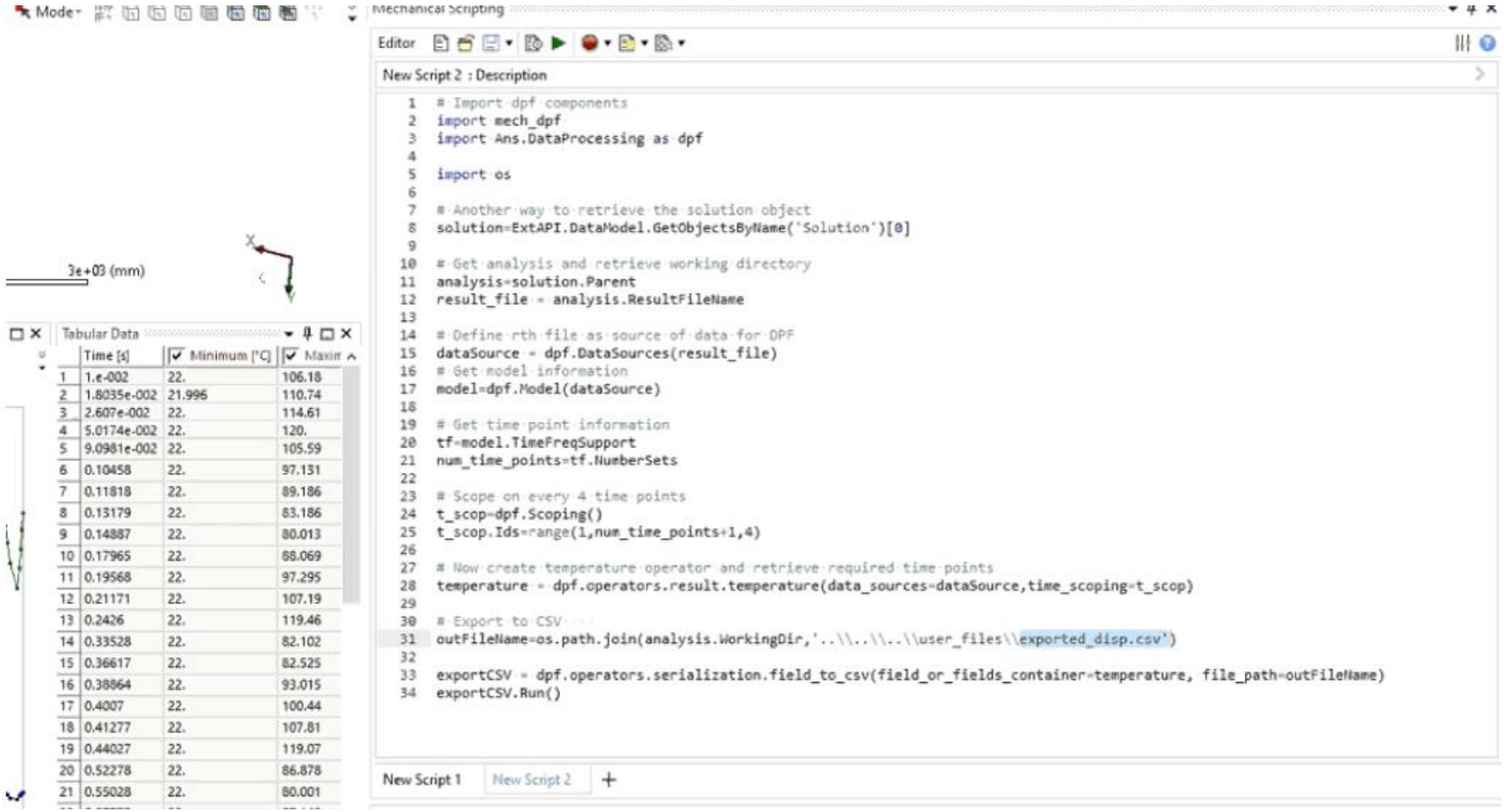


Use-Case: Exporting transient results to a CSV file – DPF in Mechanical

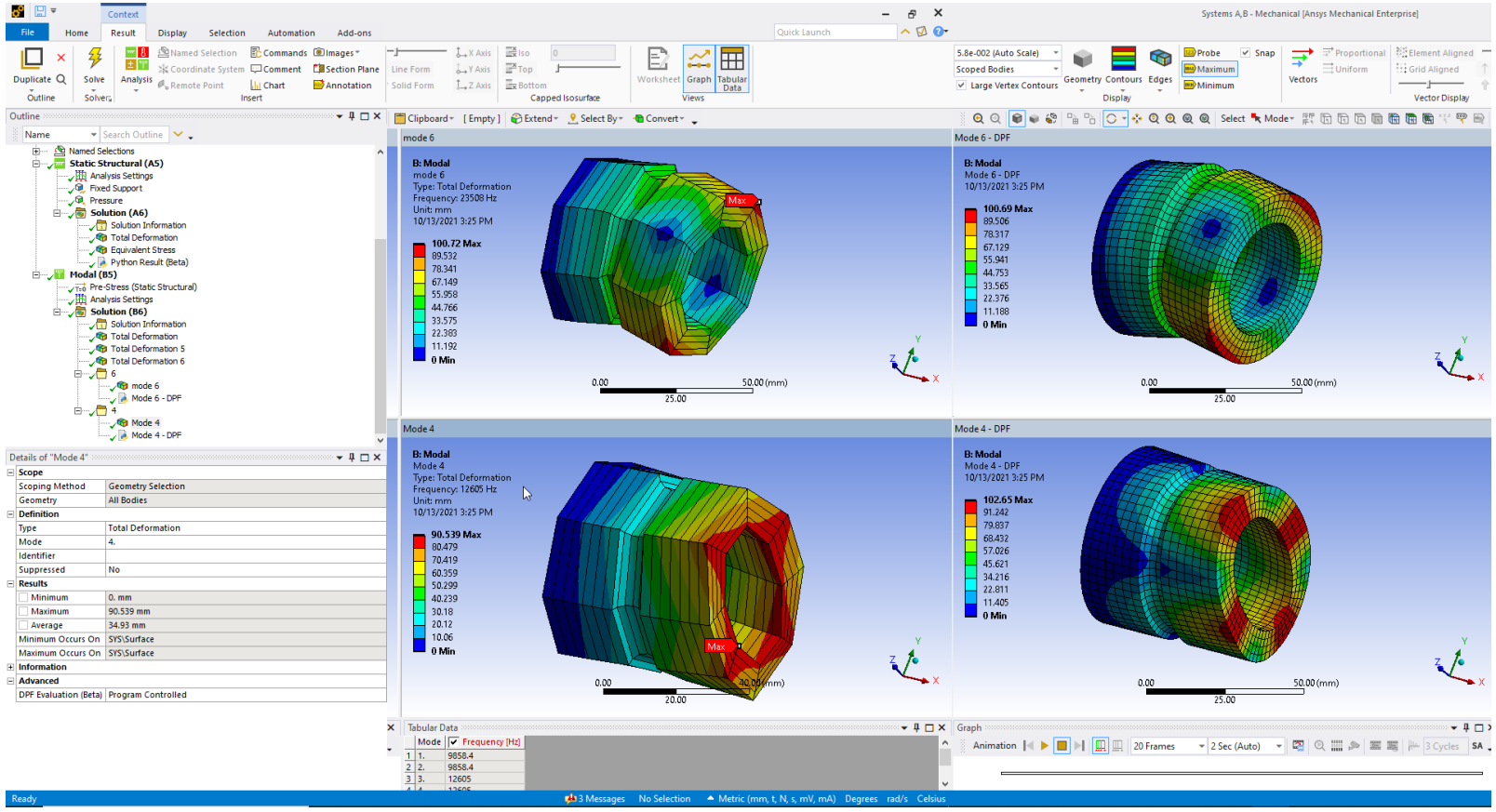


1	version	0		
2	analysis			
3	num_sets	22		
4	storage_type	0		
5	unit	degC		
6	dim			
7	location	Nodal		
8	shellLayer	1		
9				
10				
11				
12				
13	temperature_0.01s			
14		step	sub_step	cumulative_index
15	set	1	1	1
16	Scoping	Data		
17	721	2.60E+01		
18	932	2.60E+01		
19	1040	1.06E+02		
20	1122	1.06E+02		
21	730	2.60E+01		
22	1013	2.60E+01		
23	1041	1.06E+02		
24	1123	1.06E+02		

exported_disp



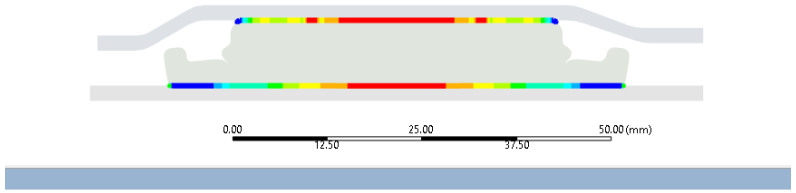
Use-Case: Expanding generalized Axisymmetric results

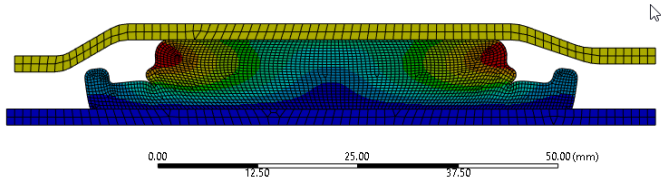


Standard results

Expanded results thanks to Mechanical DPF

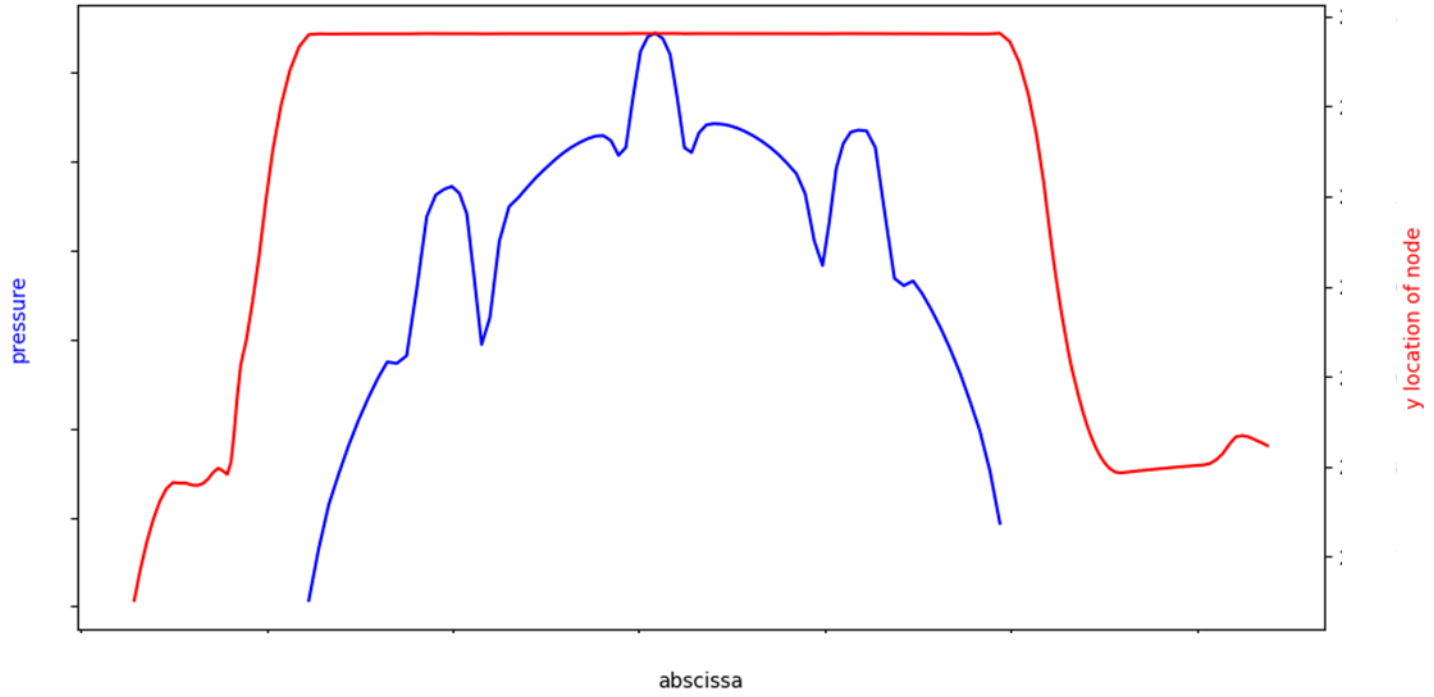
Use-Case: Extracting a pressure profile





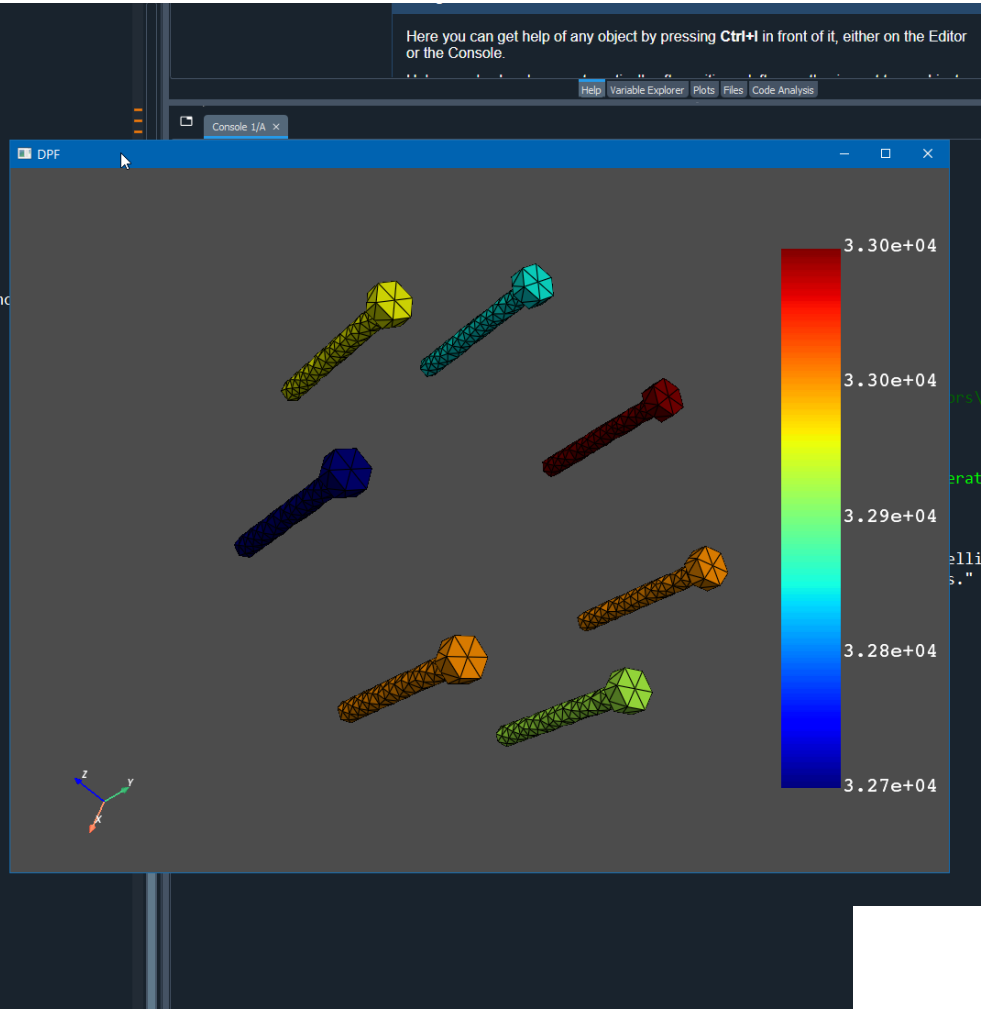

```
    #·Compute·curvilinear·abscissa
    curv_abs=getCurvilinearAbscissa(node_coords)
    
    #·Export·pressure·vs·abscissa
    myPath·=·os.path.join(userFilesDir,'pressure_on_path_{0}.txt'.format(dp))
    with·open(myPath,'w')·as·f:
        f.write('Abscissa\tPressure}\n')
        for·ca·in·curv_abs:
            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]))
```

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



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

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

# Retrieve raw forces (seems to be in element CS)
forces=dpf.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=dpf.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(nc
    # Retrieve mesh associated to pretension
    bolt_nodes=dpf.operators.scoping.on_named_selection(named_selection_name=mi.PretensionComponents[node],
                                                        requested_location='Nodal',data_sources=dataSource)
    bolt_mesh_op= dpf.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=dpf.operators.scoping.transpose(mesh_scoping=escop,meshed_region=mesh)
        bolt_mesh_op= dpf.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 = dpf.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

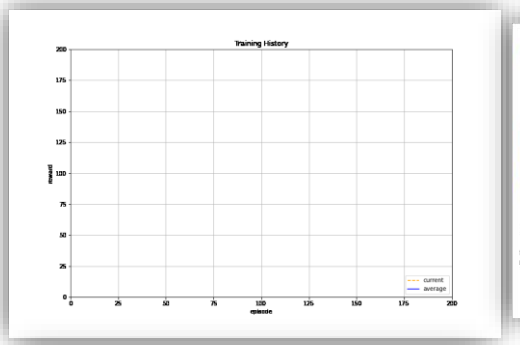
The following Ansys licensing increments provide rights to use the licensed DPF capabilities:

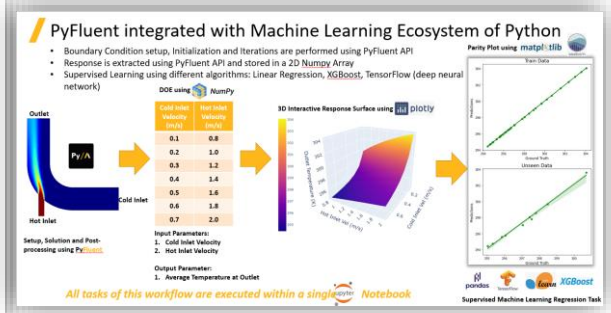
- `preppost` available in the `Ansys Mechanical Enterprise PrepPost` product
- `meba` available in the `ANSYS Mechanical Enterprise Solver` product
- `mech_2` available in the `ANSYS Mechanical Premium` product
- `mech_1` available in the `ANSYS Mechanical Pro` product
- `ansys` available in the `ANSYS 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
- `acpreppost` available in the `Ansys Mechanical Enterprise` product
- `acdi_adprepost` available in the `Ansys AUTODYN` and `Ansys AUTODYN PrepPost` products
- `cfd_preppost` available in the `Ansys CFD Enterprise` product
- `cfd_preppost_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
- `cfd_base` available in the `Ansys CFD Enterprise` product
- `cfd_solve_level1` available in the `Ansys CFD Enterprise` product
- `cfd_solve_level2` available in the `Ansys CFD Enterprise` product
- `cfd_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

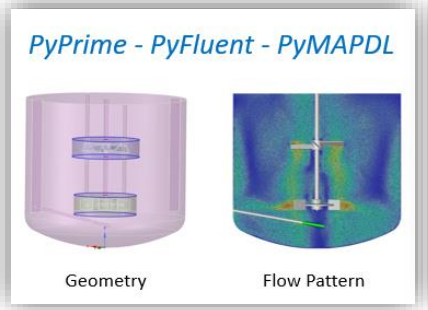
PyDPF Live Demo

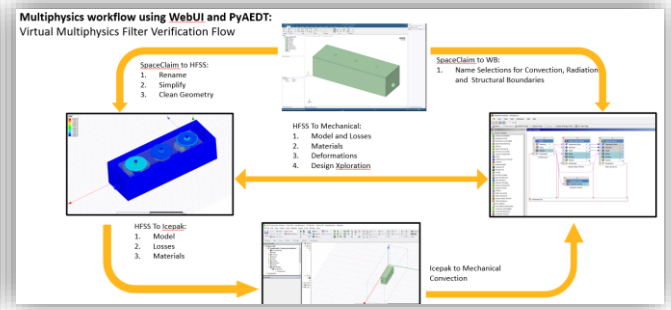
PyDPF Workshop Challenge

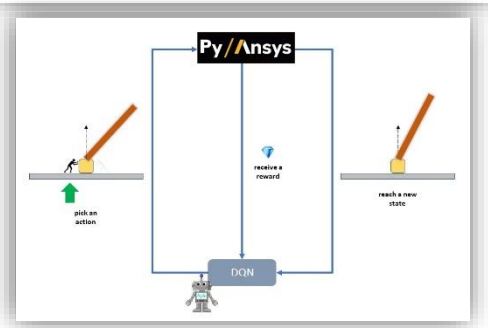
Key workflows where Ansys customers are using PyAnsys











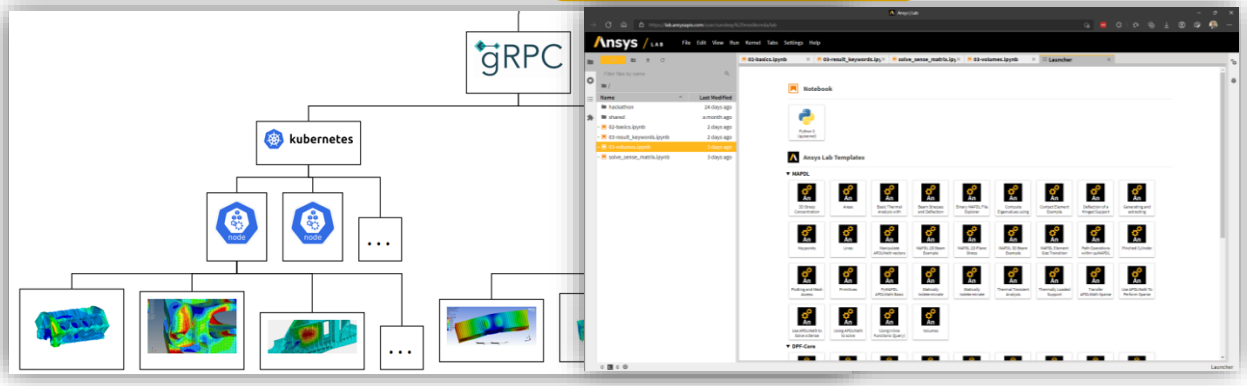
Machine Learning Problems

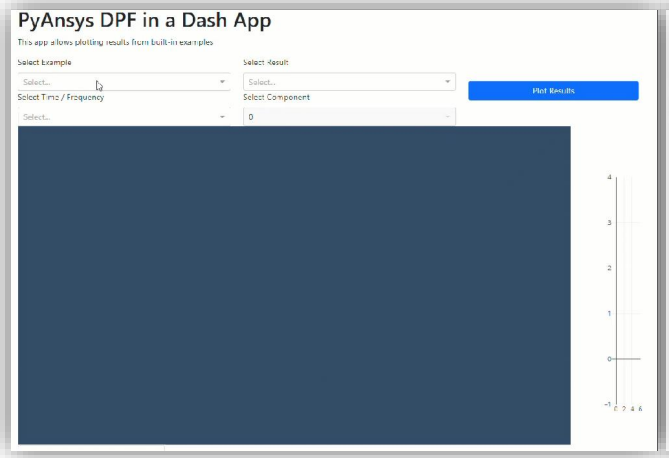
Multi-Physics workflow Automation

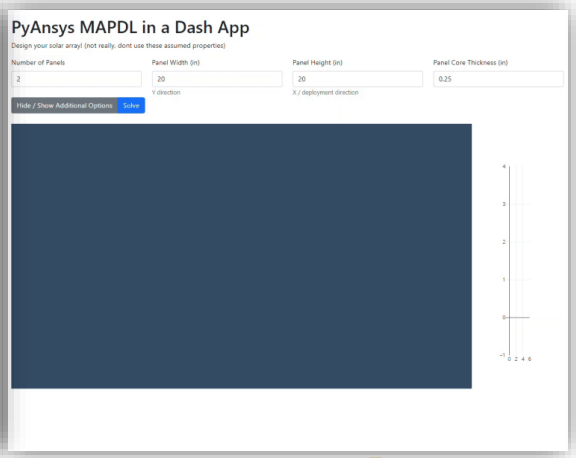
py[Ansys]

HPC Deployment

Web Applications







Angle Bracket Optimization

Angle bracket has “dimple” to increase stiffness

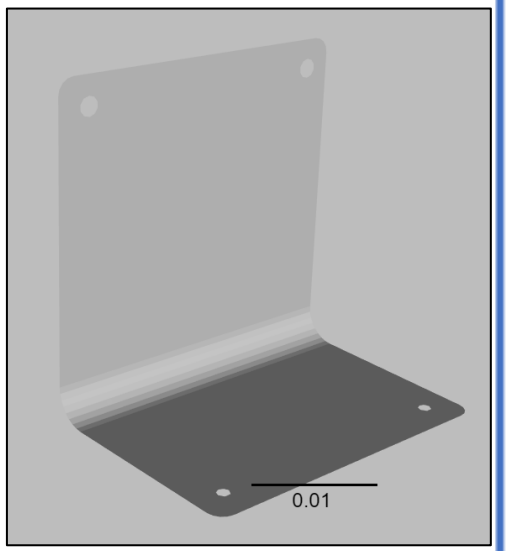
ML Example: Topographical Optimization using Reinforced Learning

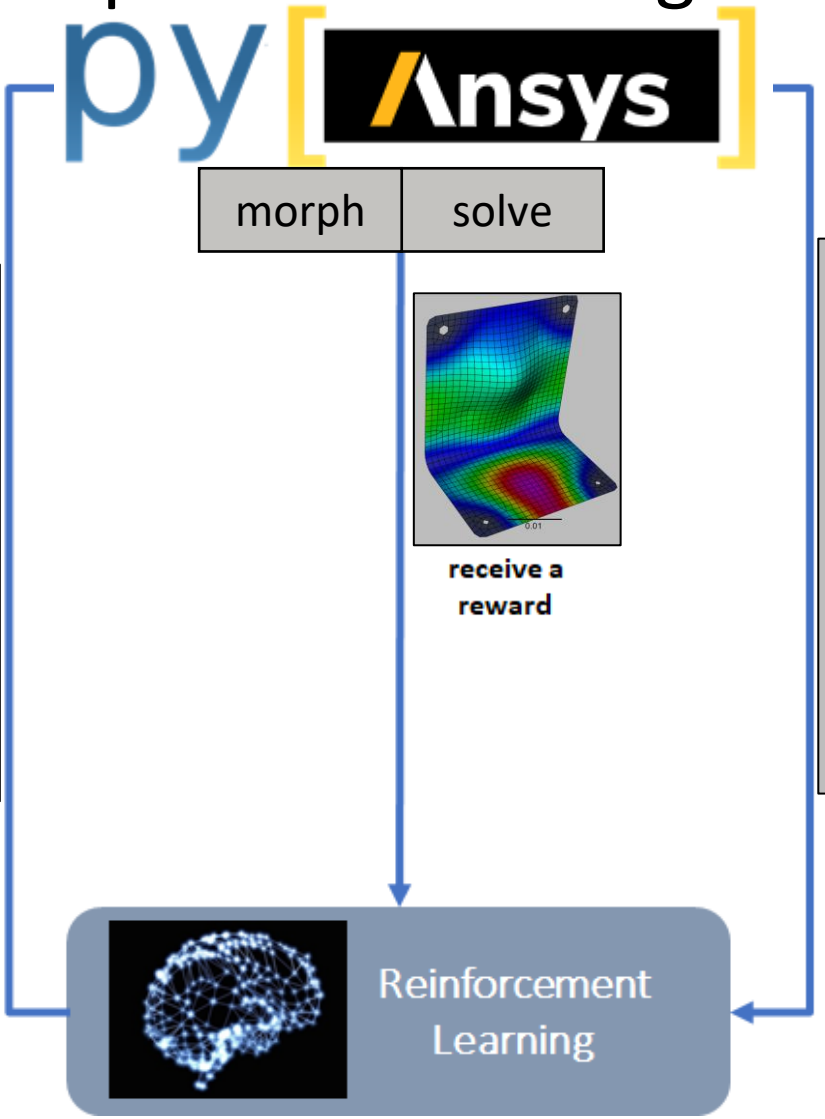
MDP Description

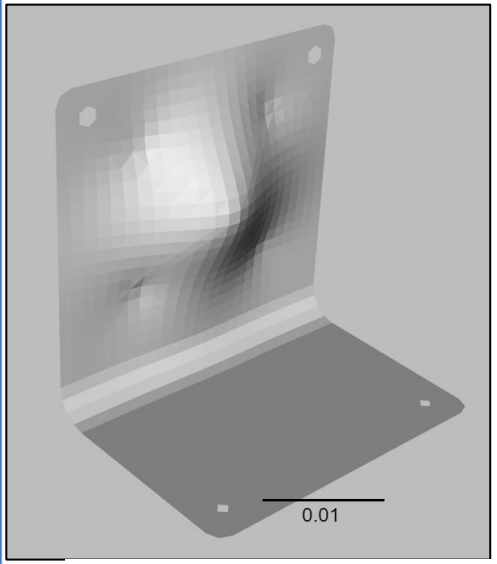
- Action:
  - morph point
  - morph direction
- State:
  - morph extent
- Reward:
  - Increase freq/stiff
  - penalty for distortion

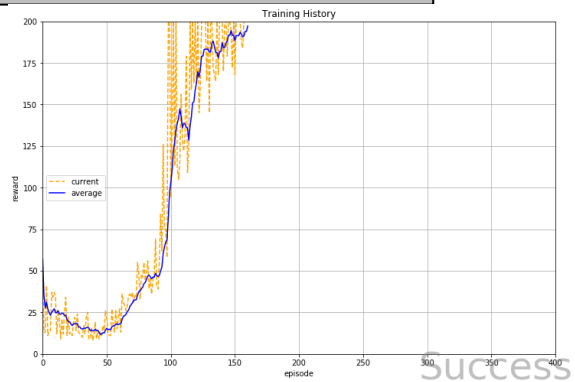
RL Algorithm

- Find sequences of morphing operations that maximize stiffness









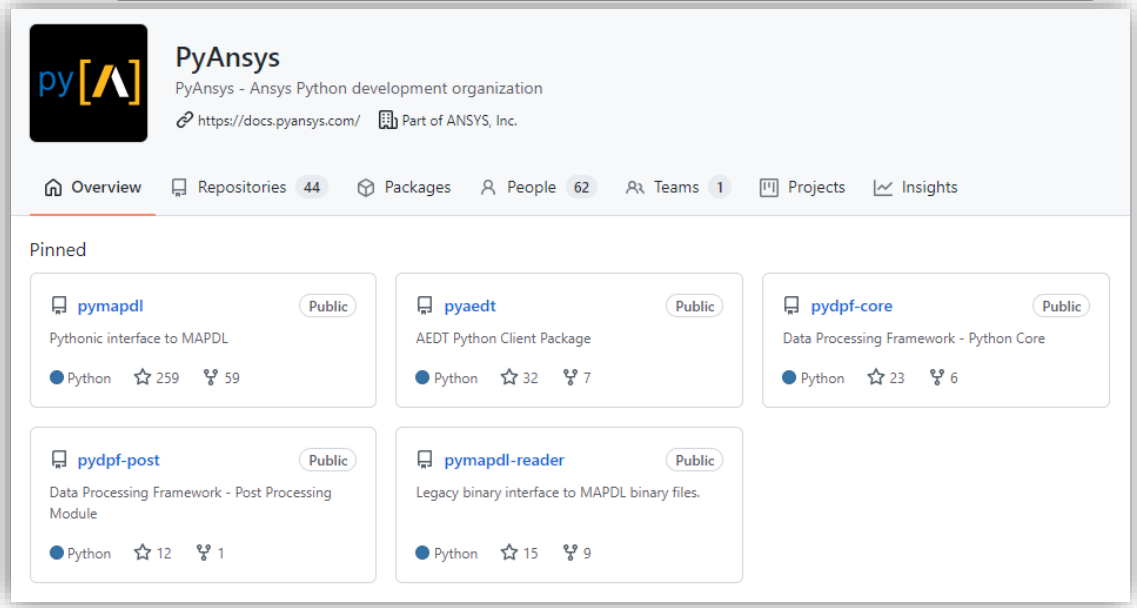
Success!

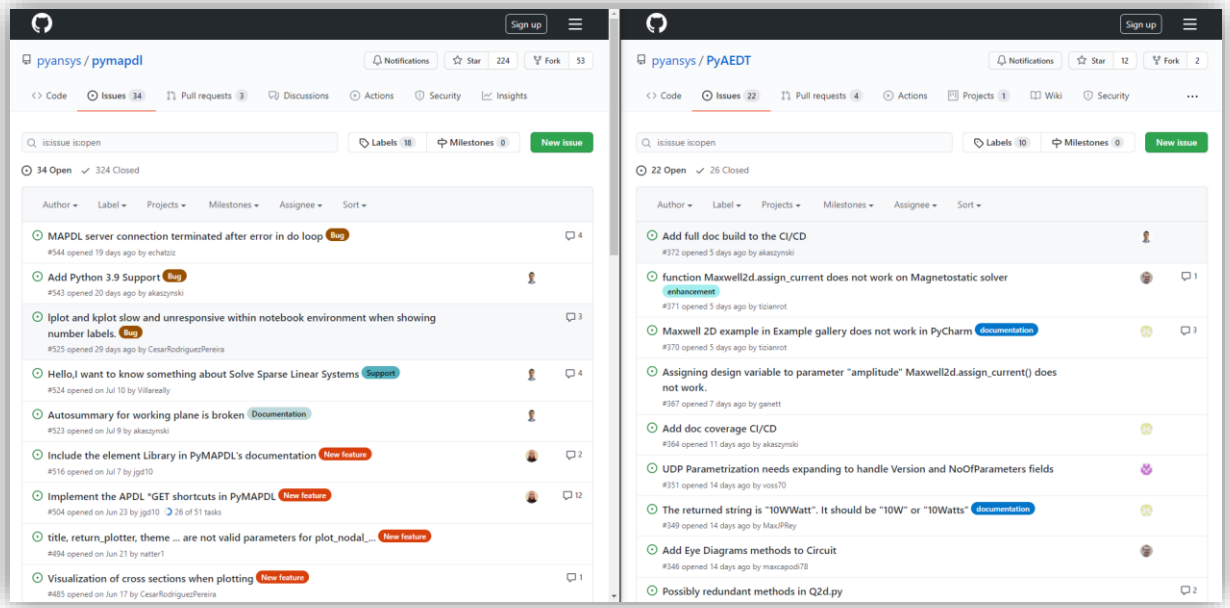
Spray Dryer WebApp

Documentation & Help?

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

Contribute/Post-Issues directly on Github:

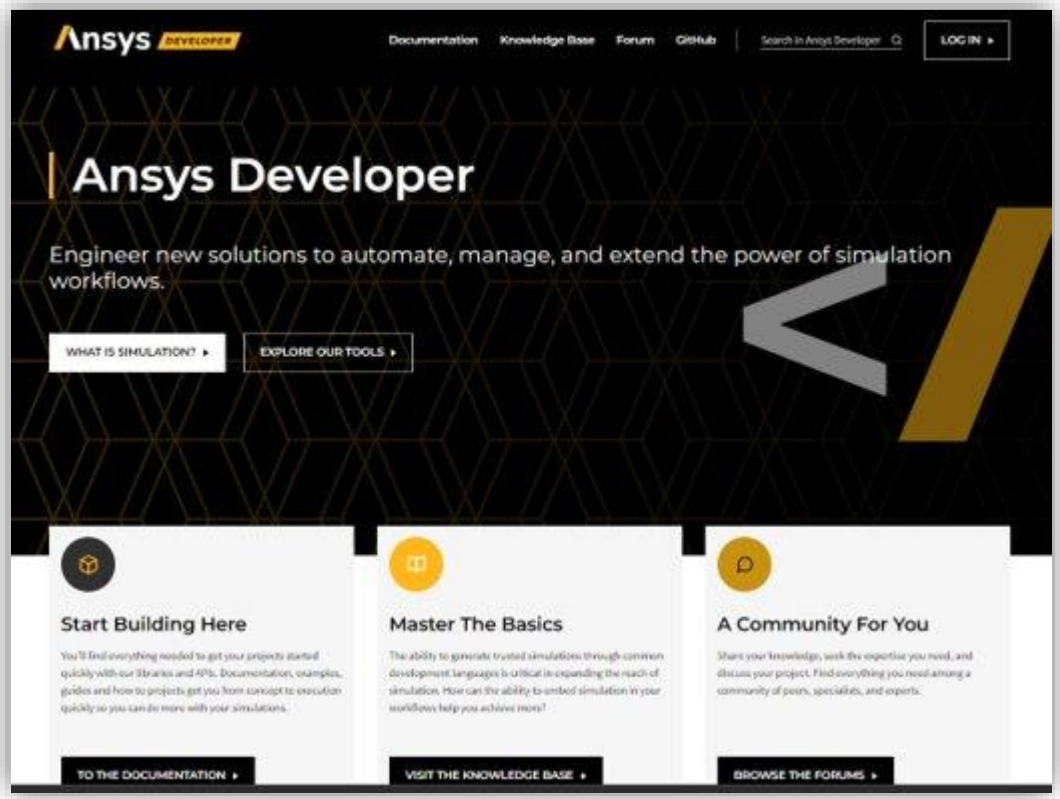


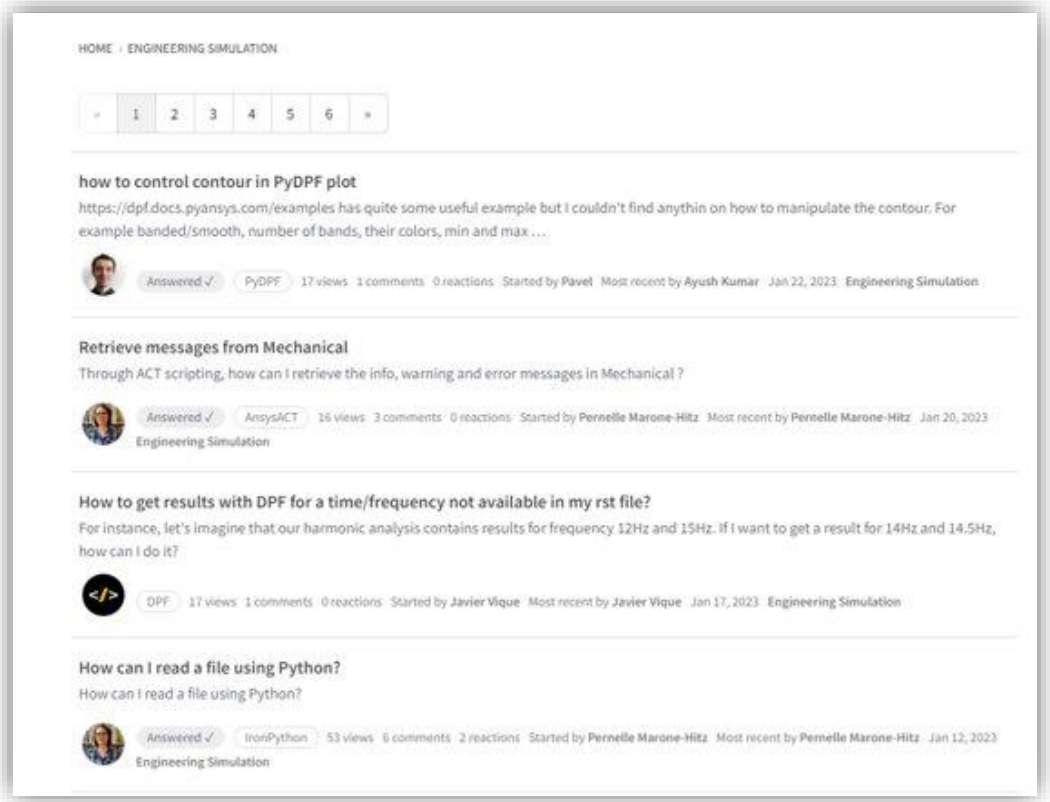


Documentation:

- 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

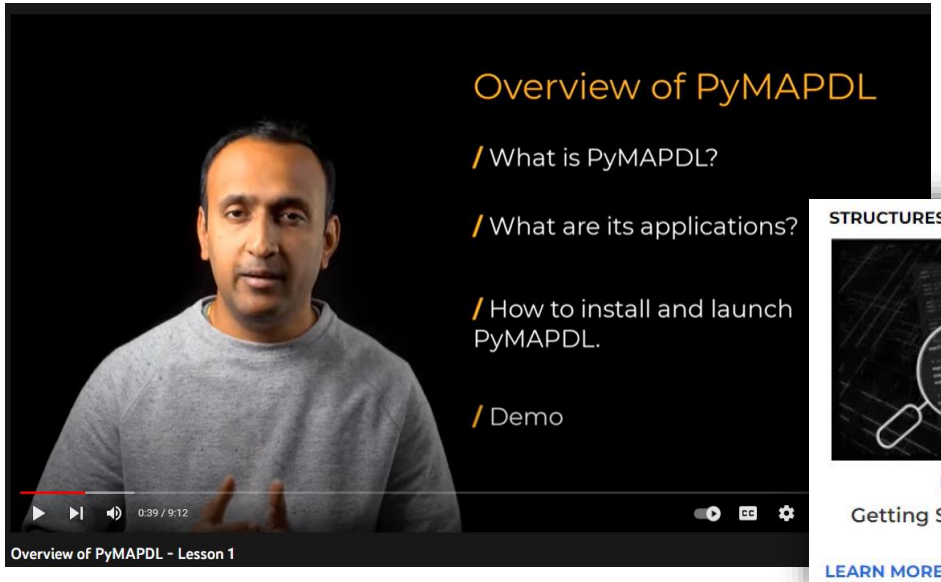




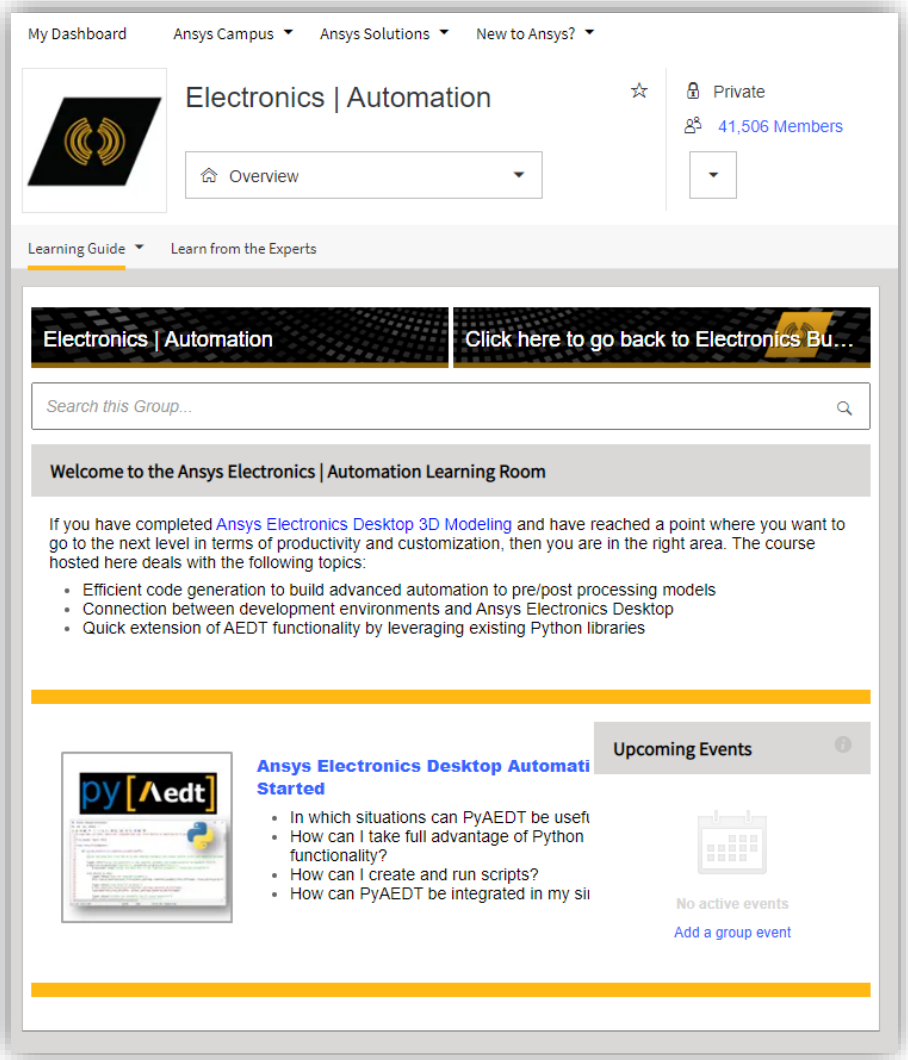
Ansys Developer Portal:
https://developer.ansys.com/

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

Trainings (AIC & ALH)

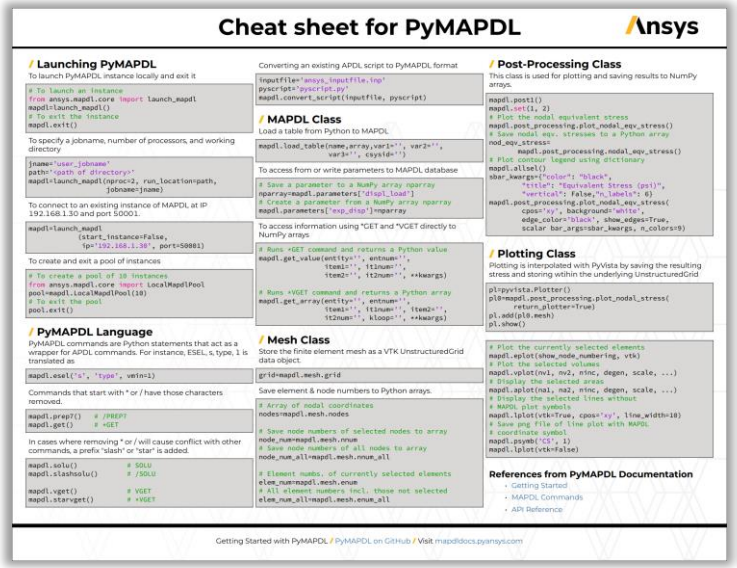


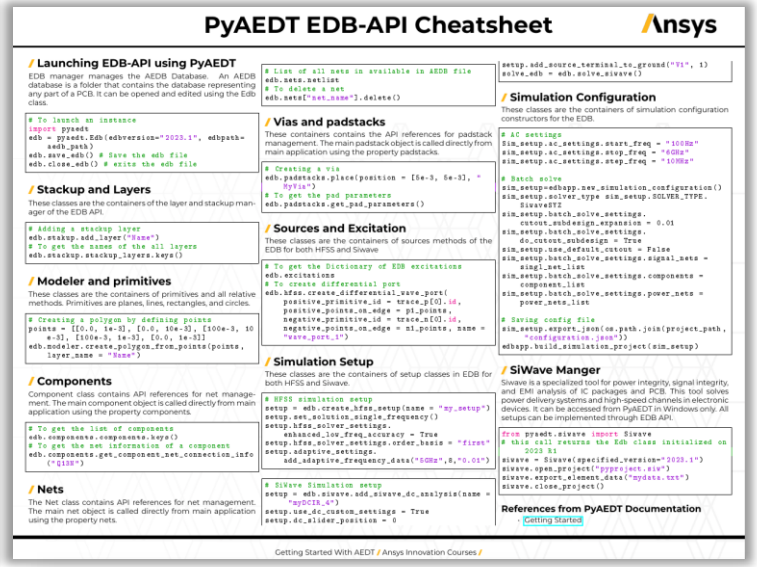
Ansys Innovation Courses (AIC)

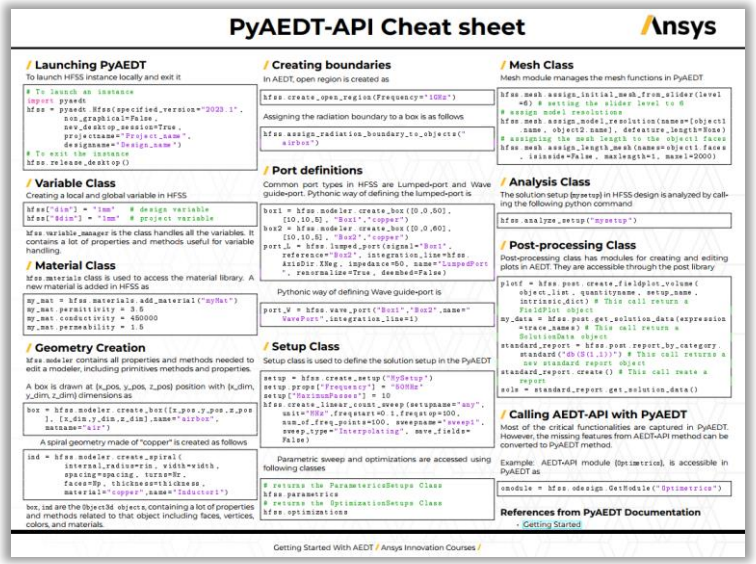


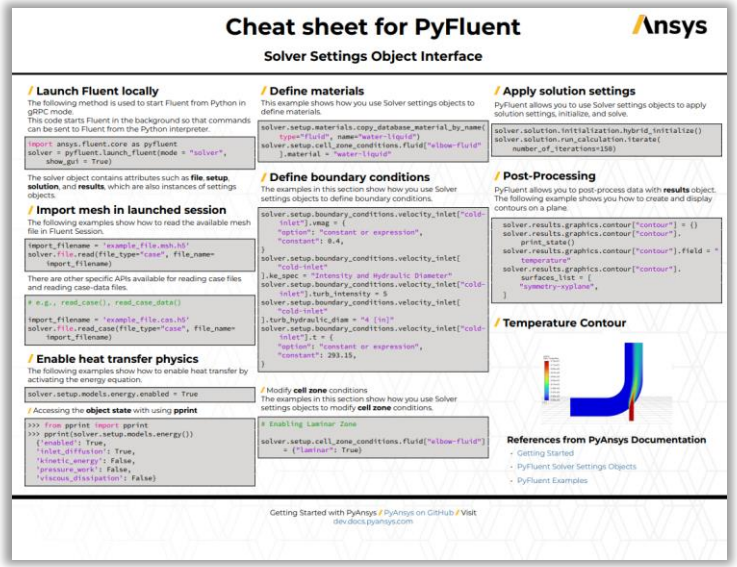
Ansys Learning Hub (ALH)

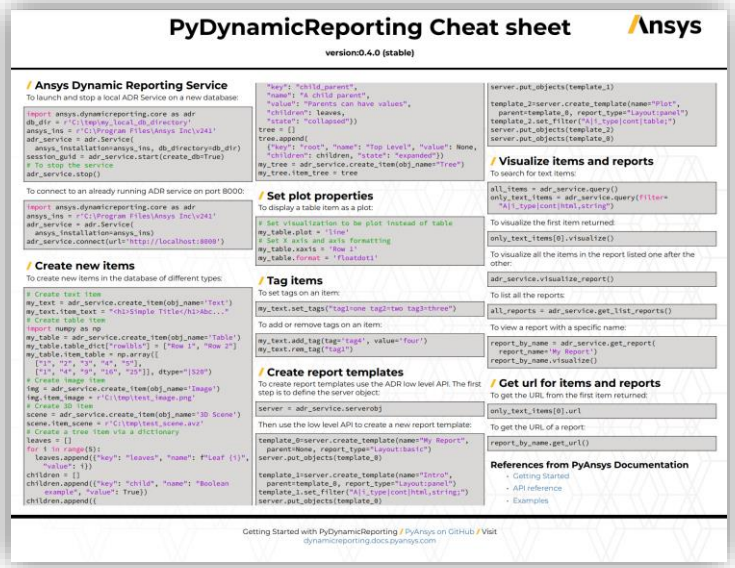
PyAnsys: Module Cheat Sheets











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

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

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

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

DPF

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

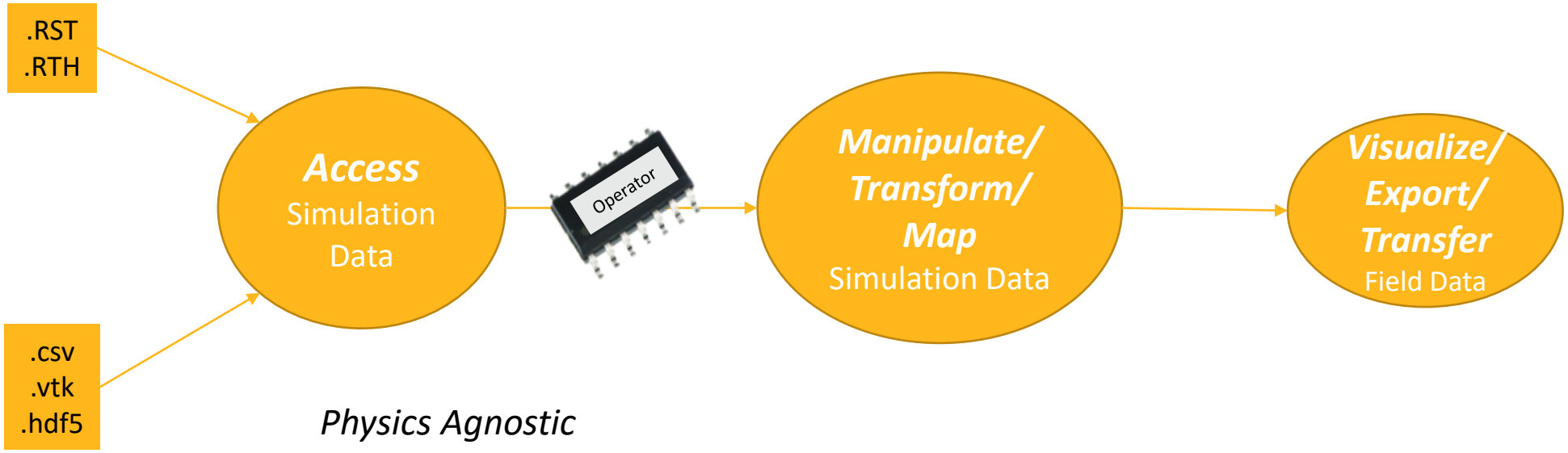
La solution

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



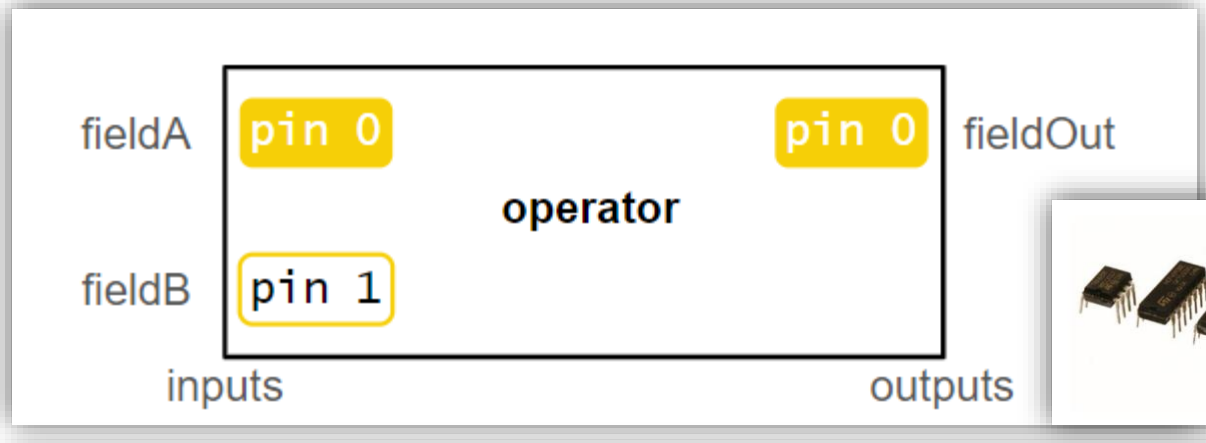
Physics Agnostic
Performance
Remote/Distributed Access
Pythonic
Customizable
Open Source
Dedicated Libraries (eg: numpy, matplotlib)
Plug & Play

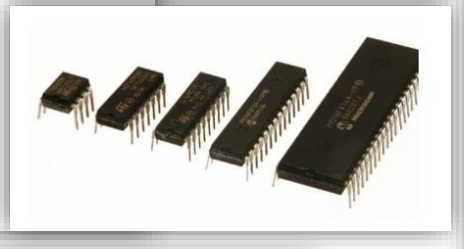
Create custom plots
Facilitate data transfer in multi-tool simulations

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

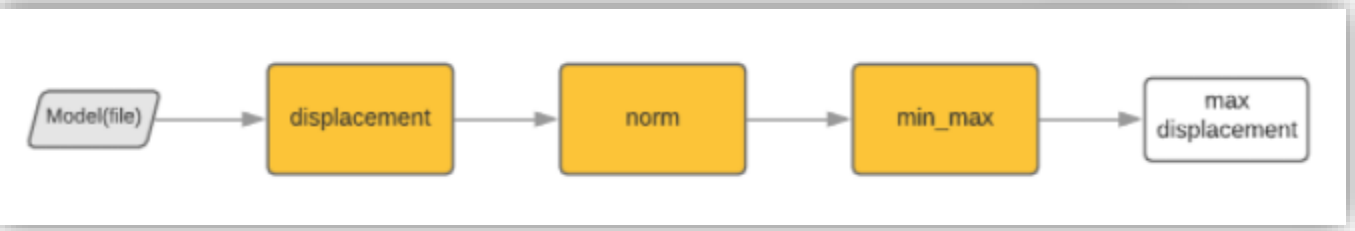


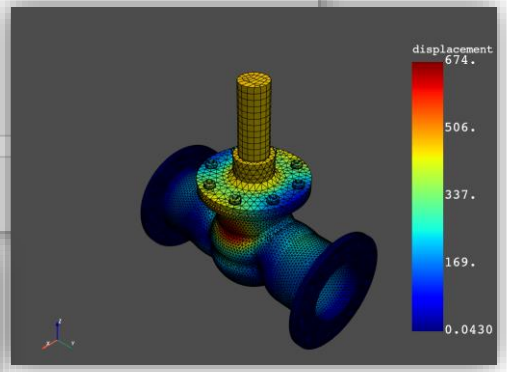


```python
## Use DPF core to manipulate data
from ansys.dpf import core

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

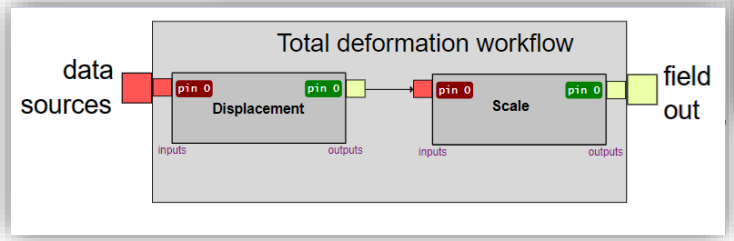
```python
op = core.Operator('scale') # operator instanciation
op.inputs.field.connect(fieldContainer[0])
scale_value = float(max_total_deformation/min_total_deformation)
op.inputs.ponderation.connect(scale_value)

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

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





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



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:

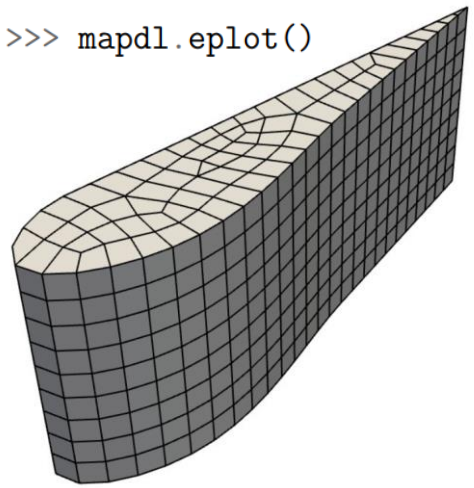
```
mapdl.run('/PREP7')
mapdl.run('K, 1, 0, 0, 0')
mapdl.run('K, 2, 1, 0, 0')
mapdl.run('K, 3, 1, 1, 0')
mapdl.run('K, 4, 0, 1, 0')
mapdl.run('L, 1, 2')
mapdl.run('L, 2, 3')
mapdl.run('L, 3, 4')
mapdl.run('L, 4, 1')
mapdl.run('AL, 1, 2, 3, 4')
```

```
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
```

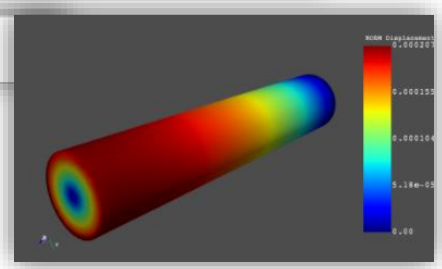
```
>>> mapdl.eplot()
```



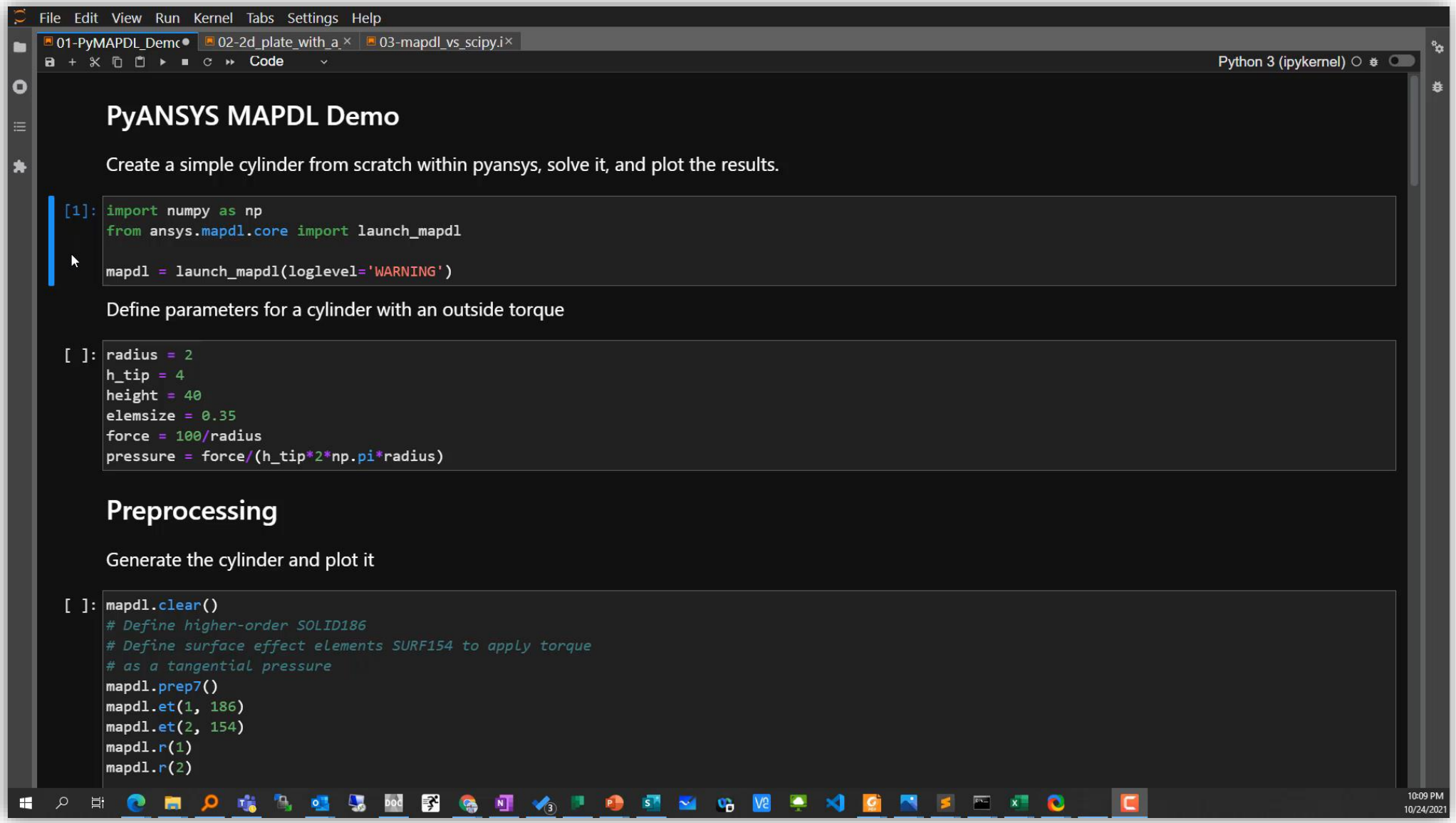
```
POST1:
PRNSOL, U, X

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
```

```
>>> 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])
```

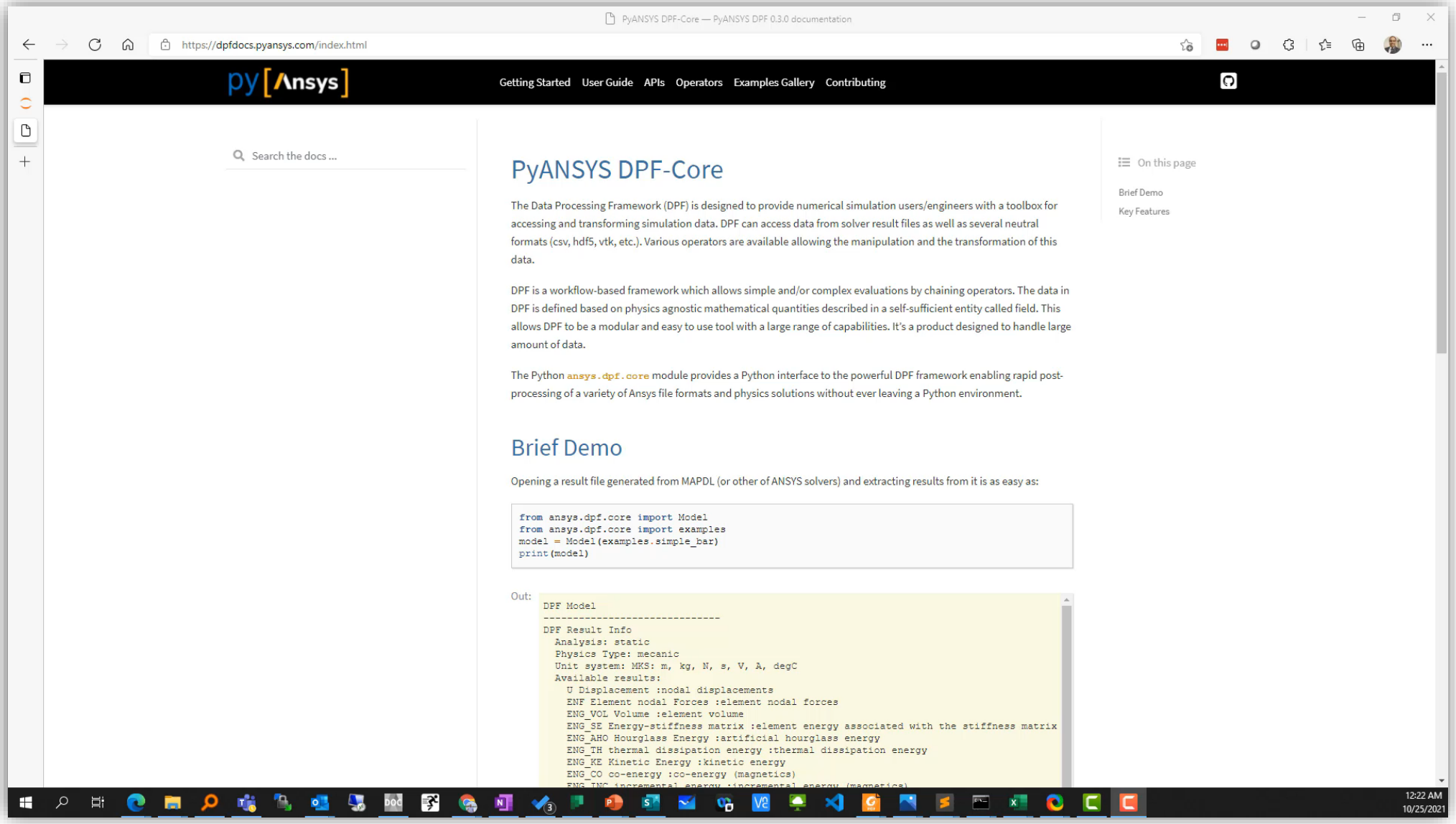
```
>>> mapdl.post_processing.plot_nodal_displacement('X')
```



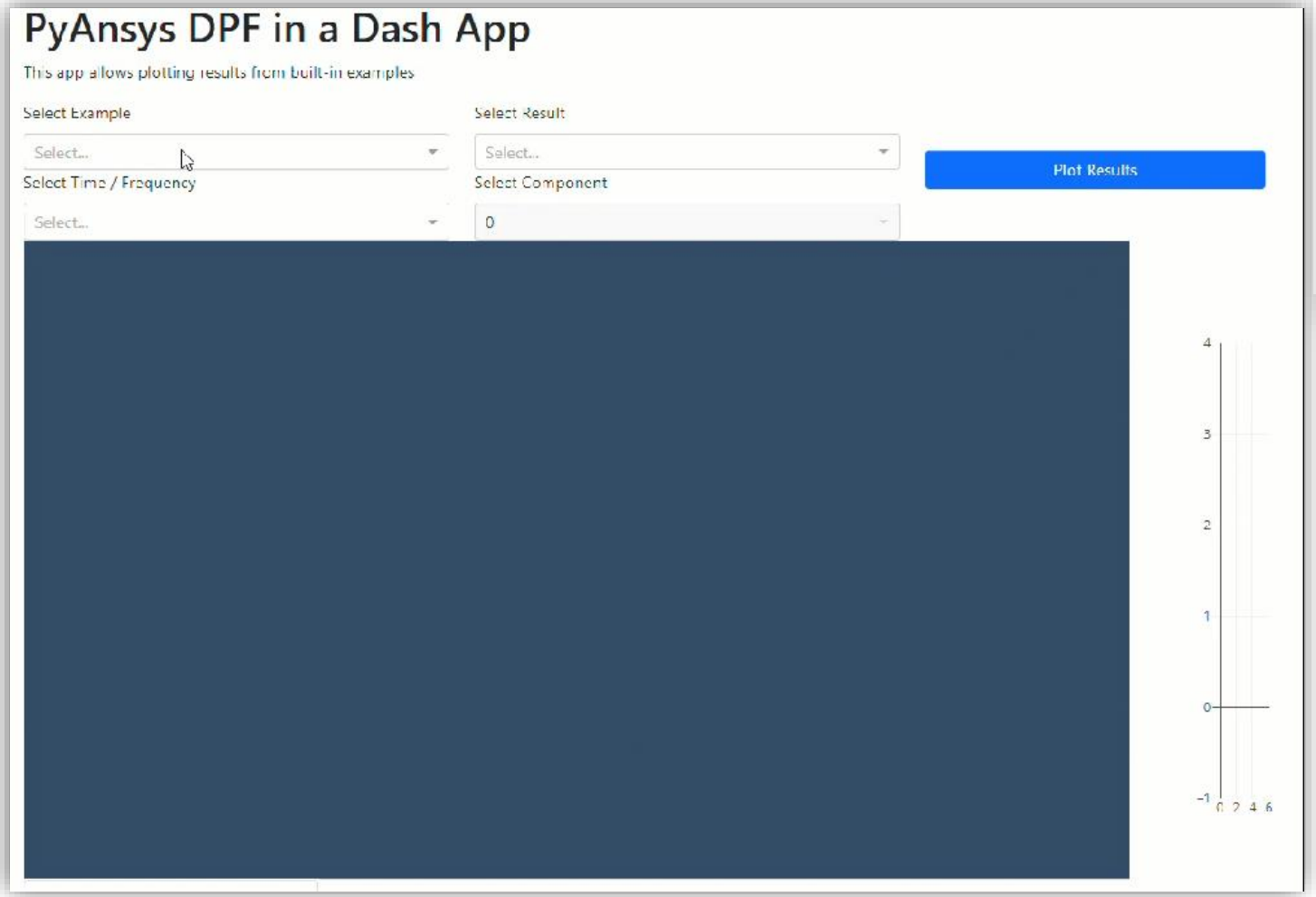
DEMO: PyMAPDL

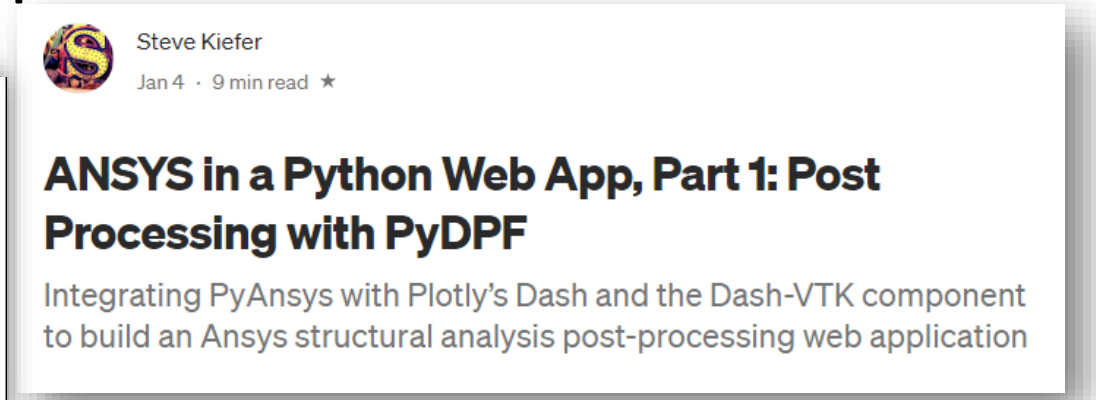


DEMO: PyDPF (Core & Post)

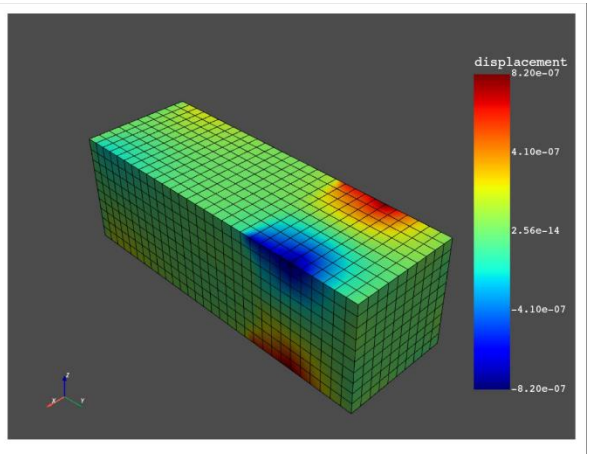


3rd Party Customers: PyDPF in a Dash App

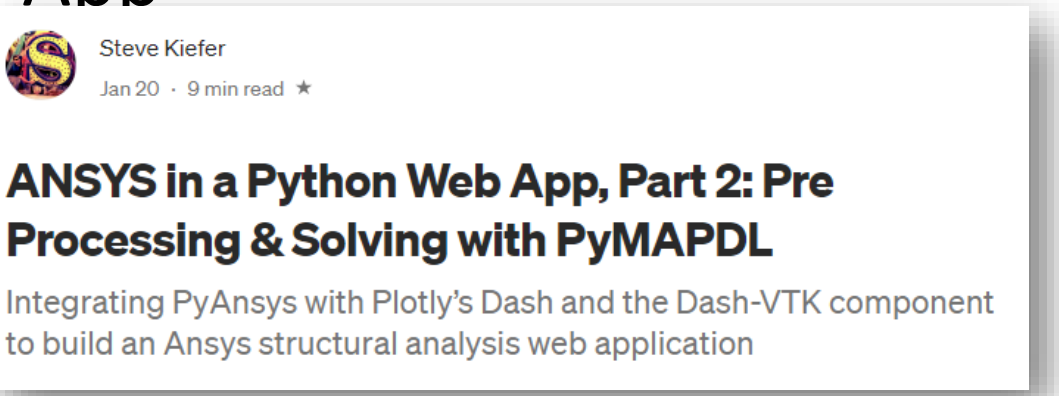




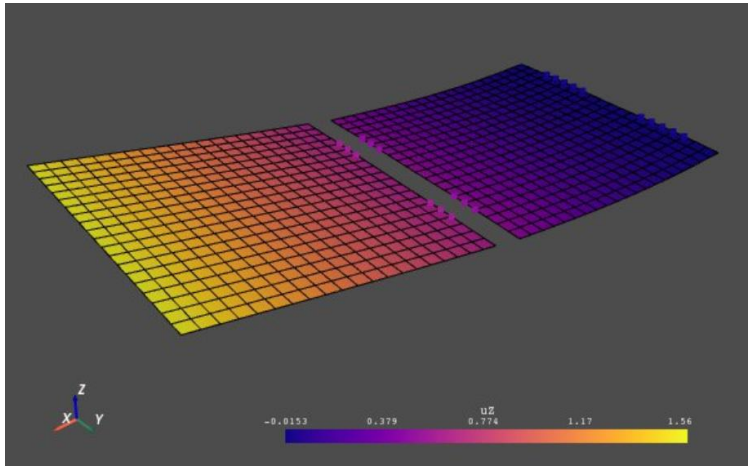
[ANSYS in a Python Web App, Part 1: Post Processing with PyDPF | by Steve Kiefer | Jan, 2022 | Towards Data Science]



3rd Party Customers: PyMAPDL in a Dash App



[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%	89.58%
2	3.18	0.00%	0.00%	0.00%	73.37%	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