

# pyCFS

*v0.1.8*

## A COMPANION LIBRARY FOR **openCFS**

Andreas Wurzinger, Eniz Mušeljić

# WHAT IS pyCFS

pyCFS is a companion project to openCFS written in Python, accumulating useful tools for incorporating openCFS in Python-based workflows.

- Run simulations from python script / notebook for
  - parameter optimization,
  - sensitivity analysis,
  - convergence studies, etc.
- Pre- and postprocessing of simulation data
  - Manipulate mesh and data
- Interface to other software packages

# WHY USE pyCFS

- Easy install via [PyPI](#)
- Easy modification / addition of new features
- Flexibility, interface to other Python packages
  - Make use of numpy, scipy, pytorch, etc.
- Comprehensive [documentation](#)!

# GETTING STARTED

# INSTALLATION

- Install latest version in pip environment

```
python -m pip install pyCFS
```

---

# INSTALLATION

- Install latest version in pip environment

```
python -m pip install pyCFS
```

---

> Update from current main branch `` ` pip pip install  
git+https://gitlab.com/openCFS/pycfs@main --upgrade --  
force-reinstall `` `

# ADDITIONAL DEPENDENCIES

- Large dependencies are excluded from the standard install
- Install dependencies for individual modules

```
pip install pyCFS[ansys]  
pip install pyCFS[vtk]
```

- Install all dependencies (not recommended)

```
pip install pyCFS[all]
```

# DOCUMENTATION

- [Documentation](#) page
  - Installation Guide
  - Basic usage Guide
    - Covers only selected features
  - [API-Documenation](#)



pyCFS 0.1.1



Q Search

ctrl + K

## Contents:

Installation

Getting started

Developer notes

pyCFS-data

**API Documentation**

data

extras

io

operators

util

v\_def



# API Documentation

Information on specific functions, classes, and methods.

[data](#)

[pyCFS.data](#)

[extras](#)

[io](#)

[operators](#)

[util](#)

[v\\_def](#)

[pyCFS](#)

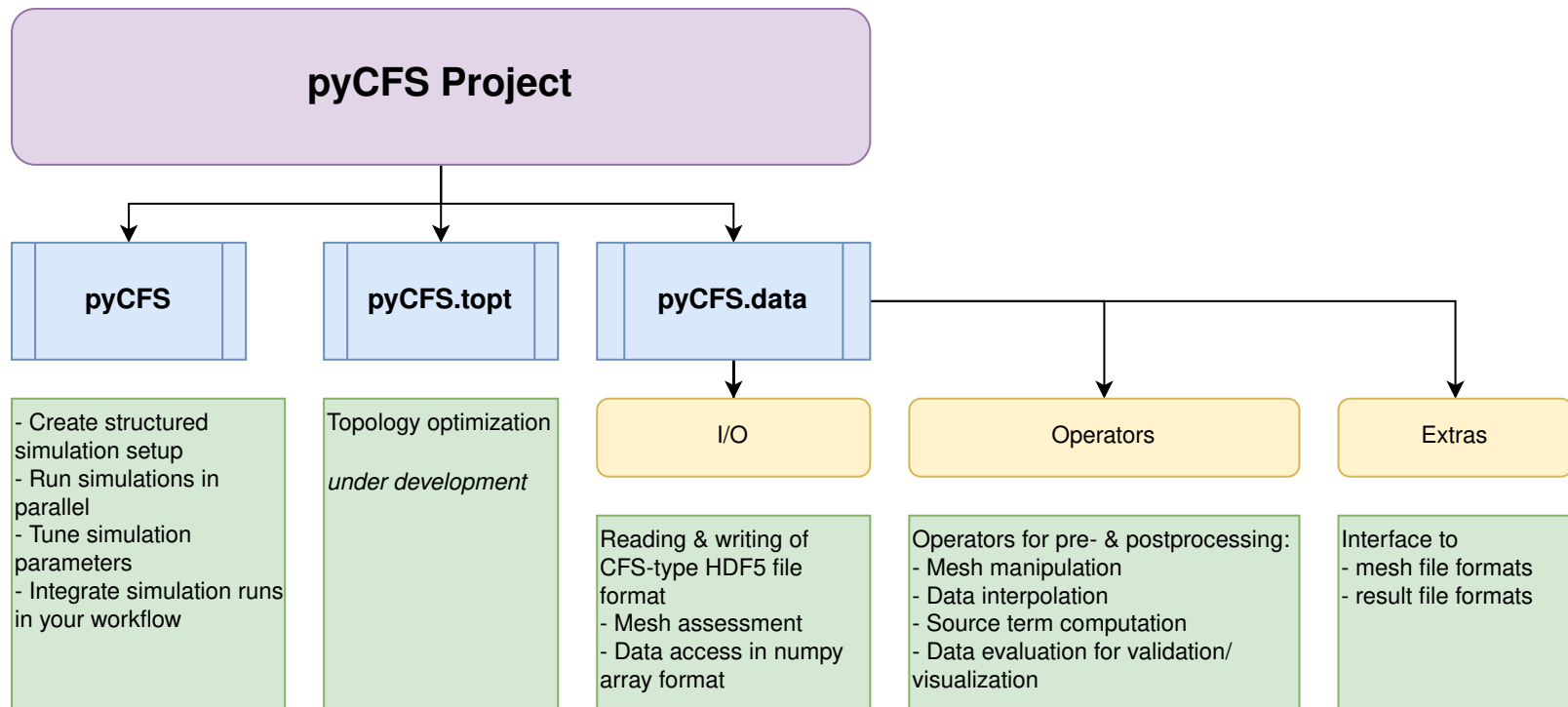
[pyCFS.pyCFS](#)

**pyCFS**

Previous

[pyCFS-data](#)

# OVERVIEW



# pyCFS

```
import pyCFS
```

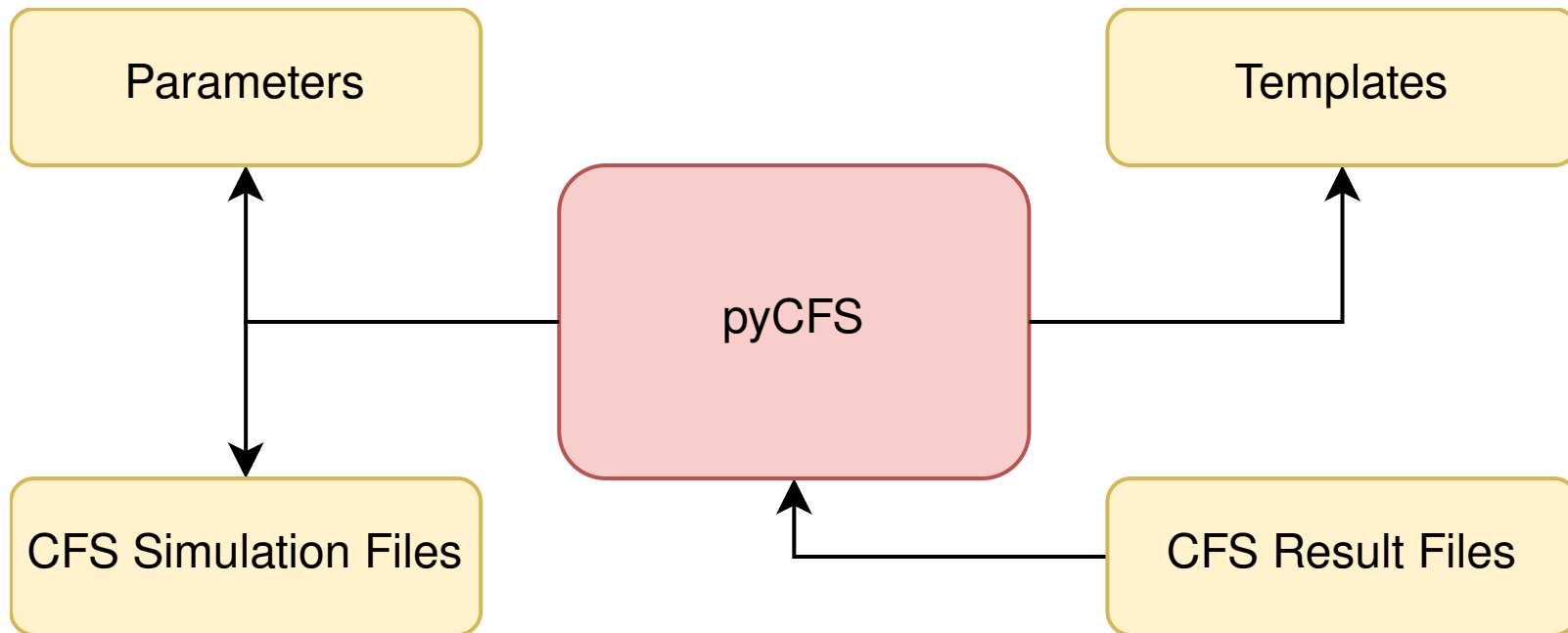
- Data management for automated simulation setup and execution
- openCFS simulation abstracted as function evaluation

# Initialize project

```
pycfs newsim -n capacitor2d
```

## Root directory structure

```
capacitor2d
|  -- sim_setup.py
|  -- run_sim.py
|  -- templates
|    -- capacitor2d_init.xml
|    -- mat_init.xml
|    -- capacitor2d_init.jou
```

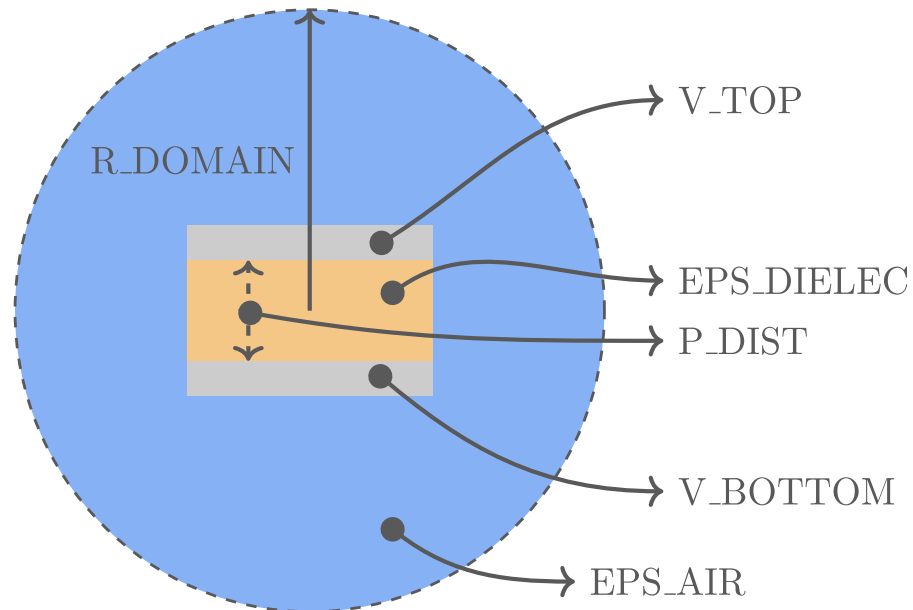


# Define variable names

```
cfs_params = ["V_TOP", "V_BOTTOM"]  
mat_params = ["MAT_AIR", "MAT_DIELEC"]  
geo_params = ["R_DOMAIN", "P_DIST"]
```

## Set variables in tempate files

```
<bcsAndLoads>  
  <potential name="S_top" value="V_TOP"/>  
  <potential name="S_bottom" value="V_BOTTOM"/>  
</bcsAndLoads>
```



# Construct pyCFS object

```
1 import pyCFS
2
3 # Set project name and cfs path :
4 project_name = "capacitor2d"
5 cfs_path = "/home/Devel/CFS/bin/cfs"
6
7 # Define variable names
8 cfs_params = ["V_TOP", "V_BOTTOM"]
9 mat_params = ["MAT_AIR", "MAT_DIELEC"]
10 geo_params = ["R_DOMAIN", "P_DIST"]
11
12 # Construct pyCFS object :
13 cap_sim = pyCFS(
14     project_name,
15     cfs_path,
16     cfs_params_names=cfs_params,
17     mat_params_names=mat_params,
18     trellis_params_names=geo_params,
19 )
20
21 # Generate parameter vector :
22 p = np.array([[10.0, 0.0, 1.0,
```

# Construct pyCFS object

```
1 import pyCFS
2
3 # Set project name and cfs path :
4 project_name = "capacitor2d"
5 cfs_path = "/home/Devel/CFS/bin/cfs"
6
7 # Define variable names
8 cfs_params = ["V_TOP", "V_BOTTOM"]
9 mat_params = ["MAT_AIR", "MAT_DIELEC"]
10 geo_params = ["R_DOMAIN", "P_DIST"]
11
12 # Construct pyCFS object :
13 cap_sim = pyCFS(
14     project_name,
15     cfs_path,
16     cfs_params_names=cfs_params,
17     mat_params_names=mat_params,
18     trellis_params_names=geo_params,
19 )
20
21 # Generate parameter vector :
22 p = np.array([[10.0, 0.0, 1.0,
```

## Construct pyCFS object

```
1 import pyCFS
2
3 # Set project name and cfs path :
4 project_name = "capacitor2d"
5 cfs_path = "/home/Devel/CFS/bin/cfs"
6
7 # Define variable names
8 cfs_params = ["V_TOP", "V_BOTTOM"]
9 mat_params = ["MAT_AIR", "MAT_DIELEC"]
10 geo_params = ["R_DOMAIN", "P_DIST"]
11
12 # Construct pyCFS object :
13 cap_sim = pyCFS(
14     project_name,
15     cfs_path,
16     cfs_params_names=cfs_params,
17     mat_params_names=mat_params,
18     trellis_params_names=geo_params,
19 )
20
21 # Generate parameter vector :
22 p = np.array([[10.0, 0.0, 1.0,
```

# ADDITIONAL FEATURES

- Result handling
  - Track certain results
  - Create sensor arrays (openCFS sensor arrays are currently not supported!)
- Parallelization (run concurrent openCFS simulations)

# pyCFS.data

```
from pyCFS.data import io, operators, util, extras
```

- Pre-Processing
  - Create .cfs files from scratch
  - Mesh and data preparation
- Post-Processing
  - Post-processing routines, e.g. FFT, MAC, MSE, etc.
  - Plot time series (significantly faster than ParaView)
- Interface to external packages
  - numpy, scipy, pyTorch, etc.

# pyCFS.data

```
from pyCFS.data import io, operators, util, extras
```

- Pre-Processing
  - Create .cfs files from scratch
  - Mesh and data preparation
- Post-Processing
  - Post-processing routines, e.g. FFT, MAC, MSE, etc.
  - Plot time series (significantly faster than ParaView)
- Interface to external packages
  - numpy, scipy, pyTorch, etc.

*Intended for small to medium size problems. Partly vectorized and parallelized, but not always RAM efficient.*

## Structured into submodules

```
from pyCFS.data import io, operators, util, extras
```

- `io`
  - I/O operations for *CFS type HDF5* format
- `operators`
  - Basic mesh/data operations
- `util`
  - Various useful functions when working with pyCFS
- `extras`
  - I/O compatibility methods to other file formats
  - Additional functionality not directly related to openCFS

# I/O

```
from pyCFS.data import io
```

- Top level functions

```
file = "file.cfs"

# Print information about the file content
print(io.file_info(file))

# Read the file
mesh, data = io.read_file(file)
mesh = io.read_mesh(file)
data = io.read_data(file, quantities=['acouPressure'])

# Create a new file
io.write_file(file, mesh=mesh, result=data)
```

- Reader and writer classes
- Data structures for mesh, and data

# I/O (CFSReader)

```
from pyCFS.data.io import CFSReader
```

- Reading CFS-type HDF5 files
  - Mesh
  - Data (on Nodes/Elements, History data)

```
<surfRegionResult type="acouPower">  
  <surfRegionList>  
    <surfRegion name="S_body" outputIds="hdf5" writeAsHistResult="ye  
  </surfRegionList>  
</surfRegionResult>
```

# I/O (CFSReader)

## Usage

```
1 with CFSReader(filename="file.cfs") as reader:
2     # Print file information
3     print(reader)
4
5     # Read the whole mesh
6     mesh = reader.MeshData
7
8     # Read coordinates, connectivity
9     coordinates = reader.Coordinates
10    connectivity = reader.Connectivity
11
12    # Read node coordinates of a specific region
13    reg_1 = reader.get_mesh_region_coordinates(region="S_CAPACITOR")
14
15    # Read all result data for sequence step 2
16    reader.set_multi_step(multi_step_id=2)
17    results_2 = reader.MultiStepData
18
19    # Read data for a specific quantity and region
20    result_1 = reader.get_multi_step_data(multi_step_id=1,
21                                         quantities=["elecPotential"],
22                                         regions=["S_CAPACITOR"])
```

# I/O (CFSReader)

## Usage

```
1 with CFSReader(filename="file.cfs") as reader:
2     # Print file information
3     print(reader)
4
5     # Read the whole mesh
6     mesh = reader.MeshData
7
8     # Read coordinates, connectivity
9     coordinates = reader.Coordinates
10    connectivity = reader.Connectivity
11
12    # Read node coordinates of a specific region
13    reg_1 = reader.get_mesh_region_coordinates(region="S_CAPACITOR")
14
15    # Read all result data for sequence step 2
16    reader.set_multi_step(multi_step_id=2)
17    results_2 = reader.MultiStepData
18
19    # Read data for a specific quantity and region
20    result_1 = reader.get_multi_step_data(multi_step_id=1,
21                                         quantities=["elecPotential"]
22                                         regions=["S_CAPACITOR"])
```

# I/O (CFSWriter)

```
from pyCFS.data.io import CFSWriter
```

- Creating new CFS-type HDF5 files
- Writing to existing CFS-type HDF5 files

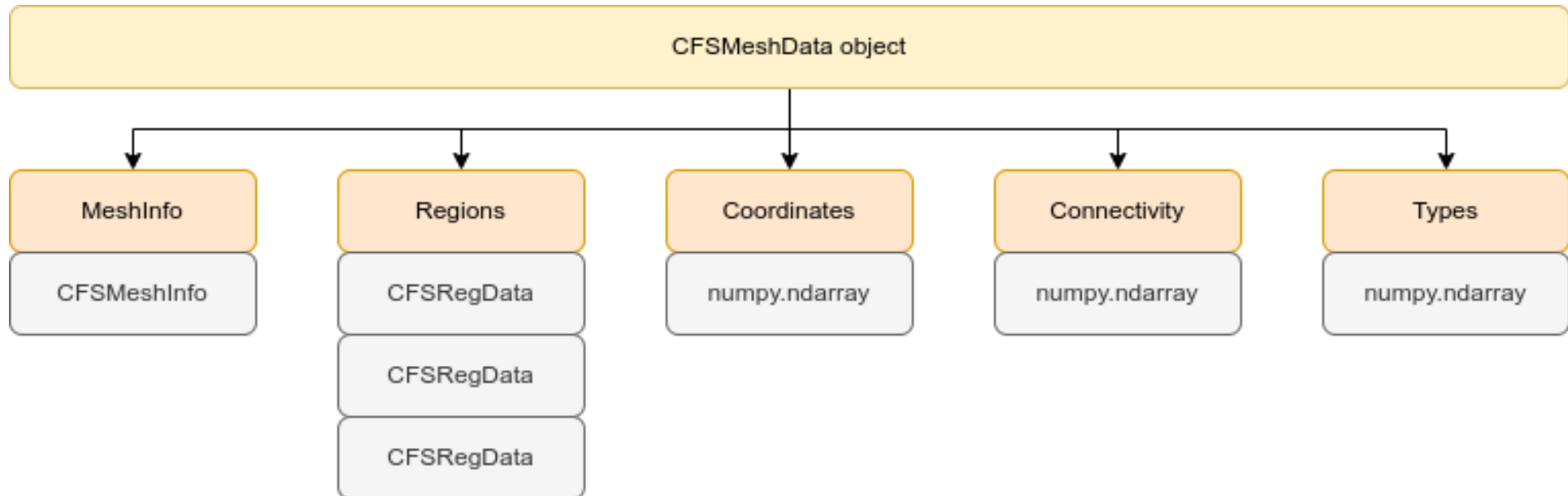
## Usage

```
with CFSWriter(filename="file.cfs") as writer:  
    # Create new file  
    writer.create_file(mesh_data=mesh, result_data=result_1)  
  
    # Write additional sequence step  
    writer.write_multistep(result_data=results_2, multi_step_id=2)
```

# I/O (CFSMeshData)

```
from pyCFS.data.io import CFSMeshData
```

- Container object for all mesh related data
- Various mesh operations



# I/O (CFSMeshData)

## Usage examples

```
1 # Create mesh object of point cloud
2 mesh_points = CFSMeshData.from_coordinates_connectivity(
3     coordinates=coordinates,
4     region_name="P_measurement"
5 )
6
7 # Create mesh object from coordinates and connectivity
8 mesh = CFSMeshData.from_coordinates_connectivity(
9     coordinates=coordinates,
10    connectivity=connectivity,
11    element_dimension=2,
12    region_name="S_plate"
13 )
14
15 # Merge mesh objects
16 mesh = mesh + mesh_points
17
18 # Print information
19 print(mesh)
20
21 # Compute element normals for a region
22 mesh.get_region_centroids(region="S_plate")
23
```

# I/O (CFSMeshData)

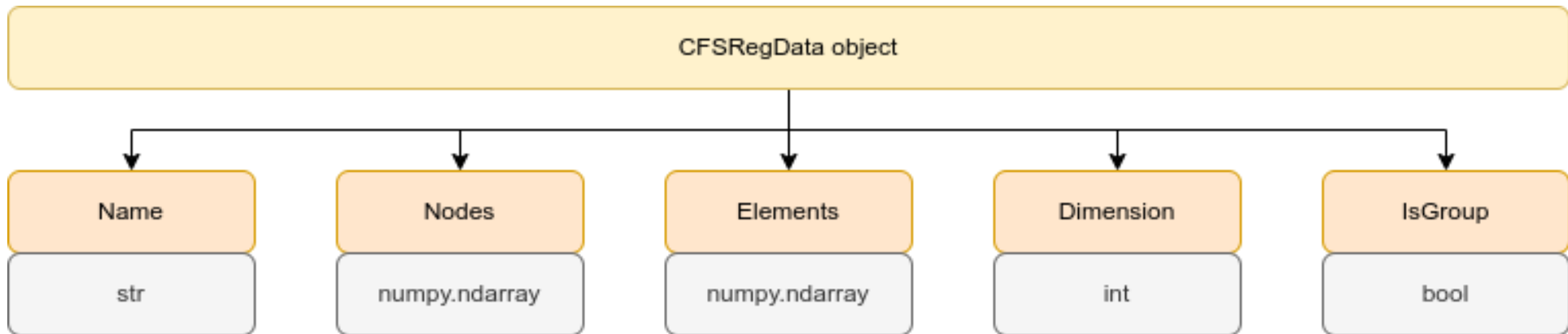
## Usage examples

```
1 # Create mesh object of point cloud
2 mesh_points = CFSMeshData.from_coordinates_connectivity(
3     coordinates=coordinates,
4     region_name="P_measurement"
5 )
6
7 # Create mesh object from coordinates and connectivity
8 mesh = CFSMeshData.from_coordinates_connectivity(
9     coordinates=coordinates,
10    connectivity=connectivity,
11    element_dimension=2,
12    region_name="S_plate"
13 )
14
15 # Merge mesh objects
16 mesh = mesh + mesh_points
17
18 # Print information
19 print(mesh)
20
21 # Compute element normals for a region
22 mesh.get_region_centroids(region="S_plate")
23
```

# I/O (CFSRegData)

```
from pyCFS.data.io import CFSRegData
```

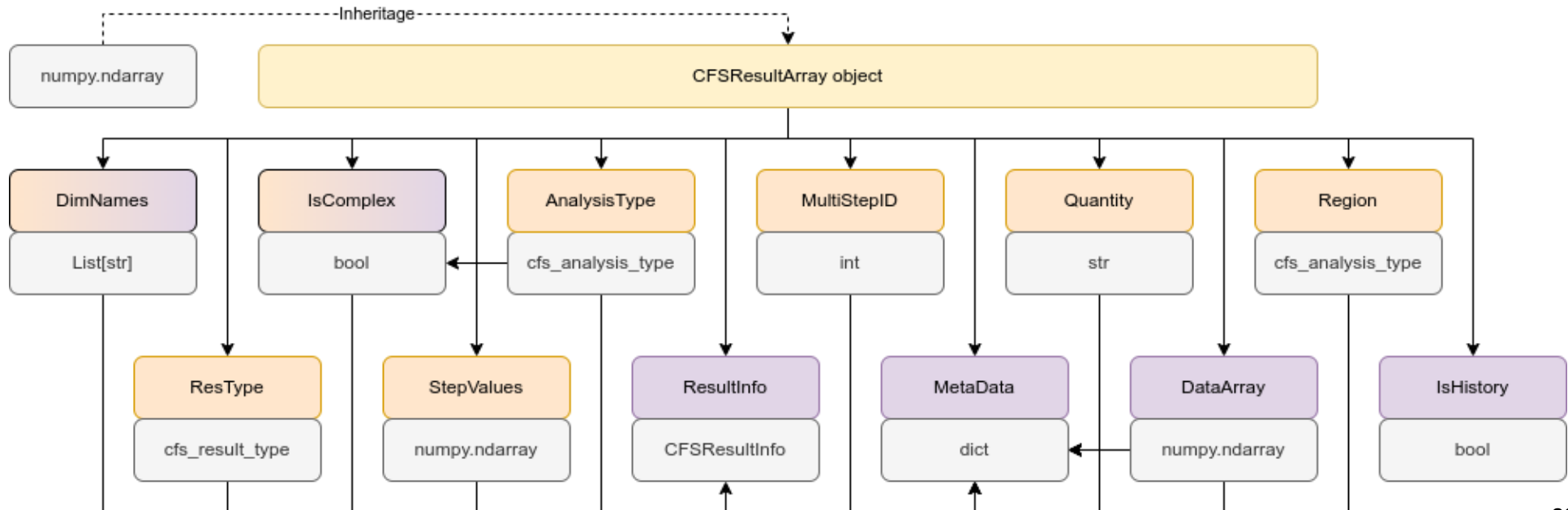
- Container object for all region related data



# I/O (CFSResultArray)

```
from pyCFS.data.io import CFSResultArray
```

- Custom numpy array type  
(compatible with all operations numpy.ndarray is compatible!)
- Including all meta data for write operations



# I/O (CFSResultArray)

## Usage examples

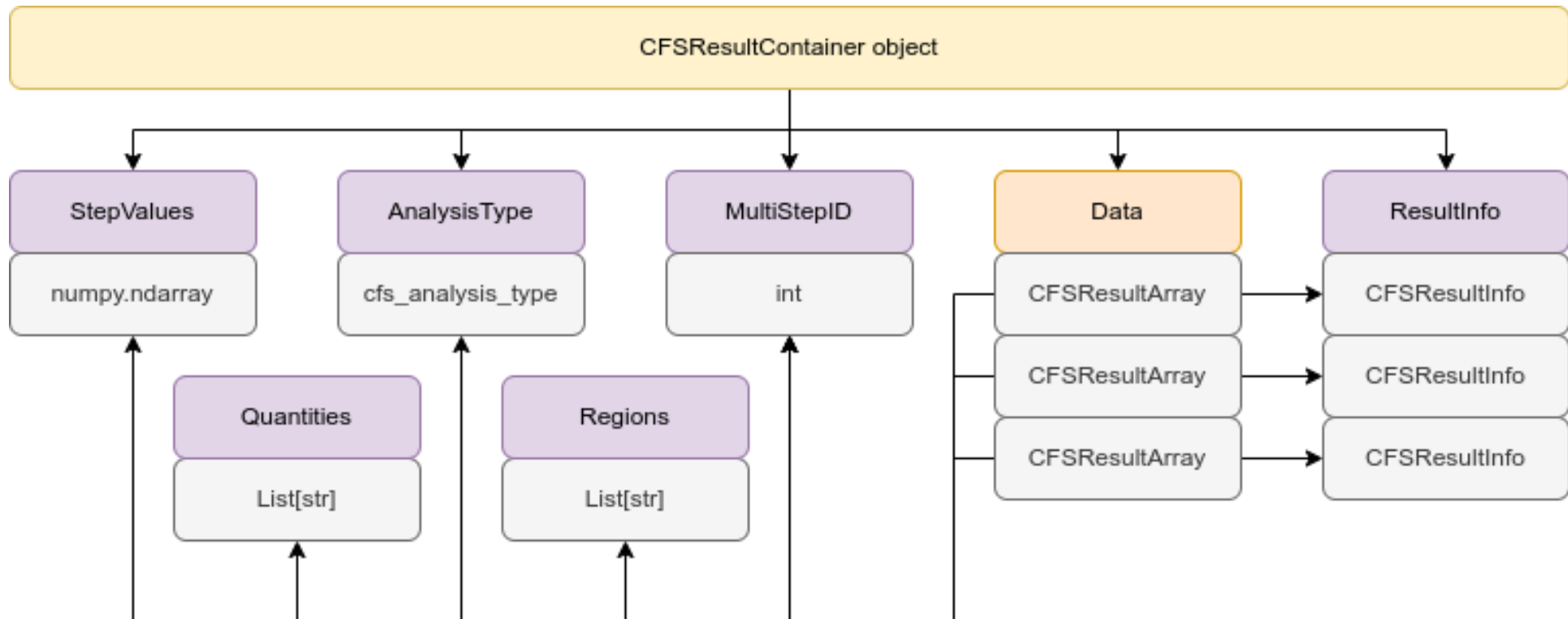
```
# Create a result array object
np_array = np.ones((5, 10, 3))
cfs_array = CFSResultArray(np_array)

# Set meta data for the result array
cfs_array.set_meta_data(
    quantity="elecPotential",
    region="S_CAPACITOR",
    step_values=np.array([0, 1, 2, 3]),
    # dim_names=["-"],
    res_type=cfs_result_type.NODE,
    # is_complex=False,
    # multi_step_id=1,
    analysis_type=cfs_analysis_type.TRANSIENT,
)
```

# I/O (CFSResultData)

```
from pyCFS.data.io import CFSResultData
```

- Container object for data of a single multistep / sequence step



# I/O (CFSResultData)

## Usage examples

```
1 # Create a result container object
2 result = CFSResultData(analysis_type=cfs_analysis_type.TRANSIENT,
3                          multi_step_id=2, data=[array_1, array_2])
4
5 # Print information
6 print(result)
7
8 # Extract certain time steps
9 result_1 = result[0:5]
10
11 # Extract certain region and quantity
12 result_2 = result.extract_quantity_region(quantity="elecPotential",
13
14 # Add data to result object (define different multi step ID)
15 result.add_data_array(data=cfs_array, multi_step_id=2)
```

# I/O (CFSResultData)

## Usage examples

```
1 # Create a result container object
2 result = CFSResultData(analysis_type=cfs_analysis_type.TRANSIENT,
3                         multi_step_id=2, data=[array_1, array_2])
4
5 # Print information
6 print(result)
7
8 # Extract certain time steps
9 result_1 = result[0:5]
10
11 # Extract certain region and quantity
12 result_2 = result.extract_quantity_region(quantity="elecPotential",
13
14 # Add data to result object (define different multi step ID)
15 result.add_data_array(data=cfs_array, multi_step_id=2)
```

# I/O (CFSResultData)

## Usage examples

```
1 # Create a result container object
2 result = CFSResultData(analysis_type=cfs_analysis_type.TRANSIENT,
3                          multi_step_id=2, data=[array_1, array_2])
4
5 # Print information
6 print(result)
7
8 # Extract certain time steps
9 result_1 = result[0:5]
10
11 # Extract certain region and quantity
12 result_2 = result.extract_quantity_region(quantity="elecPotential",
13
14 # Add data to result object (define different multi step ID)
15 result.add_data_array(data=cfs_array, multi_step_id=2)
```

# I/O (OTHER)

```
from pyCFS.data.io import cfs_types
```

- cfs\_types
  - Enum definitions based on openCFS source code

# OPERATORS

```
from pyCFS.data.operators import (interpolators, projection_interpolati  
modal_analysis, sngr, transformation)
```

- interpolators
  - Basic interpolators
    - Node2Cell
    - Cell2Node
    - Nearest Neighbor (bidirectional)
- projection\_interpolation
  - Projection-based interpolation

# OPERATORS

```
from pyCFS.data.operators import (interpolators, projection_interpolati  
modal_analysis, sng, transformation)
```

- `modal_analysis`
  - Dynamic mode decomposition
  - Field FFT
  - Various metrics used in experimental modal analysis
- `sng`
  - Compute fluctuating flow field from stationary RANS solution
- `transformation`
  - Translate / rotate / extrude / revolve mesh
  - Fit mesh onto target mesh

# EXTRA FUNCTIONALITY

- Read mesh and data from various formats
  - `ansys_io` (*Ansys Mechanical: .rst*) → `pyCFS[ansys]`
  - `cgns_io` (*various CFD/FEM software: .cgns*)
  - `ensight_io` (*various CFD software: .case*) → `pyCFS[vtk]`
  - `exodus_io` (*Cubit mesh export: .e*)
  - `nihu_io` (*NiHu simulation export: .mat*)
  - `psv_io` (*Polytec PSV export: .unv*)
  - `stl_io` (*Surface mesh: .stl*)

# EXAMPLE WORKFLOWS

# I/O

## Tasks

1. Read mesh and result data
2. View connectivity array and node coordinates of a specific region
3. Multiply result with factor
4. Add result to existing file as a new sequence step (multi step)

# CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3
4 # Read file
5 with io.CFSReader(filename="file.cfs") as f:
6     # Read mesh data
7     mesh = f.MeshData
8     # Read results of sequence step 1
9     results = f.get_multi_step_data(multi_step_id=1)
10
11 # View connectivity array, get coordinates of V_air
12 conn = print(mesh.Connectivity)
13 reg_coord = mesh.get_region_coordinates(region="V_air")
14
15 # Get data array of elecPotential in region V_air
16 elec_pot = results.get_data_array(quantity="elecPotential", region='
17
18 # Manipulate result
19 igte_factor = 1e0
20 elec_pot *= igte_factor
21
22 # Write "corrected" result to new sequence step
```

# CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3
4 # Read file
5 with io.CFSReader(filename="file.cfs") as f:
6     # Read mesh data
7     mesh = f.MeshData
8     # Read results of sequence step 1
9     results = f.get_multi_step_data(multi_step_id=1)
10
11 # View connectivity array, get coordinates of V_air
12 conn = print(mesh.Connectivity)
13 reg_coord = mesh.get_region_coordinates(region="V_air")
14
15 # Get data array of elecPotential in region V_air
16 elec_pot = results.get_data_array(quantity="elecPotential", region='
17
18 # Manipulate result
19 igte_factor = 1e0
20 elec_pot *= igte_factor
21
22 # Write "corrected" result to new sequence step
```

# CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3
4 # Read file
5 with io.CFSReader(filename="file.cfs") as f:
6     # Read mesh data
7     mesh = f.MeshData
8     # Read results of sequence step 1
9     results = f.get_multi_step_data(multi_step_id=1)
10
11 # View connectivity array, get coordinates of V_air
12 conn = print(mesh.Connectivity)
13 reg_coord = mesh.get_region_coordinates(region="V_air")
14
15 # Get data array of elecPotential in region V_air
16 elec_pot = results.get_data_array(quantity="elecPotential", region='
17
18 # Manipulate result
19 igte_factor = 1e0
20 elec_pot *= igte_factor
21
22 # Write "corrected" result to new sequence step
```

# CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3
4 # Read file
5 with io.CFSReader(filename="file.cfs") as f:
6     # Read mesh data
7     mesh = f.MeshData
8     # Read results of sequence step 1
9     results = f.get_multi_step_data(multi_step_id=1)
10
11 # View connectivity array, get coordinates of V_air
12 conn = print(mesh.Connectivity)
13 reg_coord = mesh.get_region_coordinates(region="V_air")
14
15 # Get data array of elecPotential in region V_air
16 elec_pot = results.get_data_array(quantity="elecPotential", region='
17
18 # Manipulate result
19 igte_factor = 1e0
20 elec_pot *= igte_factor
21
22 # Write "corrected" result to new sequence step
```

# CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3
4 # Read file
5 with io.CFSReader(filename="file.cfs") as f:
6     # Read mesh data
7     mesh = f.MeshData
8     # Read results of sequence step 1
9     results = f.get_multi_step_data(multi_step_id=1)
10
11 # View connectivity array, get coordinates of V_air
12 conn = print(mesh.Connectivity)
13 reg_coord = mesh.get_region_coordinates(region="V_air")
14
15 # Get data array of elecPotential in region V_air
16 elec_pot = results.get_data_array(quantity="elecPotential", region='
17
18 # Manipulate result
19 igte_factor = 1e0
20 elec_pot *= igte_factor
21
22 # Write "corrected" result to new sequence step
```

# DEBUGGING IN PYCHARM (1)

The screenshot displays the PyCharm IDE with a Python script named `tutorial_io.py` and its debug console.

**Script Content:**

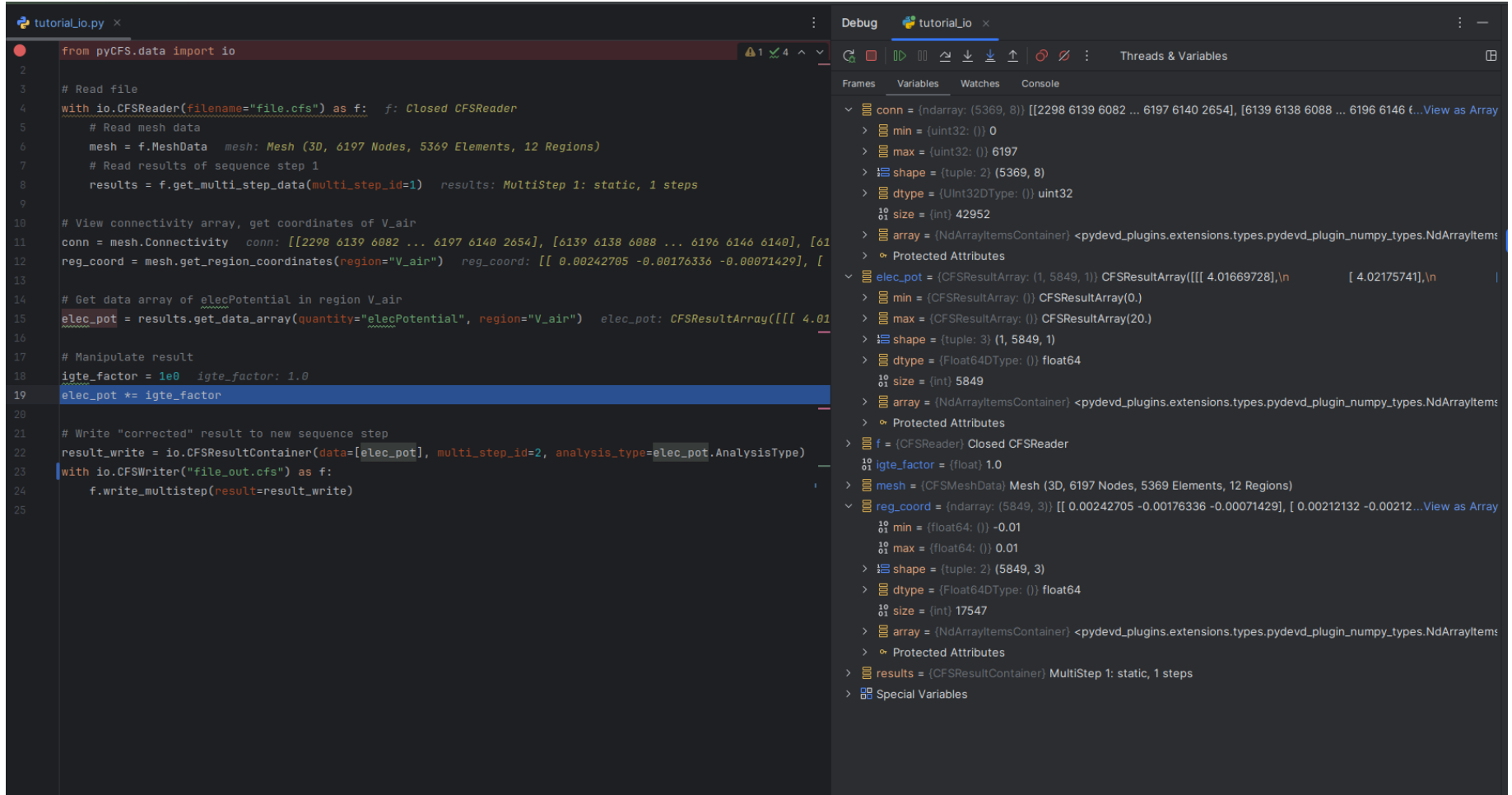
```
1 from pyCFS.data import io
2
3 # Read file
4 with io.CFSReader(filename="file.cfs") as f: f: CFSReader linked to file_src 'file.cfs', Verbosity 100
5     # Read mesh data
6     mesh = f.MeshData mesh: Mesh (3D, 6197 Nodes, 5369 Elements, 12 Regions)
7     # Read results of sequence step 1
8     results = f.get_multi_step_data(multi_step_id=1) results: MultiStep 1: static, 1 steps
9
10 # View connectivity array, get coordinates of V_air
11 conn = mesh.Connectivity
12 reg_coord = mesh.get_region_coordinates(region="V_air")
13
14 # Get data array of elecPotential in region V_air
15 elec_pot = results.get_data_array(quantity="elecPotential", region="V_air")
16
17 # Manipulate result
18 igte_factor = 1e0
19 elec_pot *= igte_factor
20
21 # Write "corrected" result to new sequence step
22 result_write = io.CFSResultContainer(data=[elec_pot], multi_step_id=2, analysis_type=elec_pot.AnalysisType)
23 with io.CFSWriter("file_out.cfs") as f:
24     f.write_multistep(result=result_write)
25
```

**Debug Console:**

The debug console shows the state of variables during execution:

- `f = {CFSReader} CFSReader linked to file_src 'file.cfs', Verbosity 100`
- `mesh = {CFSMeshData} Mesh (3D, 6197 Nodes, 5369 Elements, 12 Regions)`
  - `Connectivity = (ndarray: (5369, 8)) [[2298 6139 6082 ... 6197 6140 2654], [6139 6138 6088 ... 6...View as Array`
  - `Coordinates = (ndarray: (6197, 3)) [[ 0.00205972 -0.00151054 -0.00071429], [ 0.00242705 -0.0...View as Array`
  - `MeshInfo = {CFSMeshInfo} Mesh Info (3D, 6197 Nodes, 5369 Elements)`
  - `Regions = (list: 12) [Group: P0_elem, Group: P0_node, Group: P1_elem, Group: P1_node, Group: P2_elem, Group: P2_node, Group: P3_elem, Group: P3_node, Group: P4_elem, Group: P4_node, Group: P5_elem, Group: P5_node]`
  - `Types = (ndarray: (5369, 1)) [[11], [11], [11], [11], [11], [11], [11], [11], [11], [11], [11], [11]]...View as Array`
  - `Protected Attributes`
- `results = {CFSResultContainer} MultiStep 1: static, 1 steps`
  - `AnalysisType = {cfs_analysis_type} STATIC`
  - `Data = (list: 17) [CFSResultArray([[[ 41.83970685, -35.71589755, -5.88247991],\n [ 36.444... View`
  - `MultiStepID = (int) 1`
  - `Quantities = (list: 5) ['elecPotential', 'elecEnergy', 'elecFluxDensity', 'elecFieldIntensity', 'elecCharge']`
  - `Regions = (list: 11) ['P2_node', 'P2_elem', 'P1_node', 'P3_elem', 'P0_node', 'S_top', 'P3_node', 'V_elec', 'V_air', 'P0...`
  - `ResultInfo = (list: 17) ['elecFieldIntensity' (real) defined in 'V_air' on Elements, 'elecFieldIntensity' (real) c... View`
  - `StepValues = (ndarray: (1,)) [0.]...View as Array`
  - `Protected Attributes`
- `Special Variables`

# DEBUGGING IN PYCHARM (2)



# Operators

## Tasks

1. Read mesh and result data
2. Perform Node-to-Cell interpolation
3. Add interpolated data to existing results
4. Write mesh and results to a new file

# CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3 from pyCFS.data.operators import interpolators
4
5 # Read source file
6 print(io.file_info("file.cfs"))
7 mesh = io.read_mesh("file.cfs")
8 results = io.read_data("file.cfs")
9
10 # Perform interpolation
11 results_interpolated = interpolators.interpolate_node_to_cell(
12     mesh=mesh,
13     result=results,
14     regions=["V_air"],
15     quantity_names={"elecPotential": "interpolated_elecPotential"},
16 )
17
18 # Add interpolated result to results container
19 results.combine_with(results_interpolated)
20
21 # Check results container
22 print(results)
```

# CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3 from pyCFS.data.operators import interpolators
4
5 # Read source file
6 print(io.file_info("file.cfs"))
7 mesh = io.read_mesh("file.cfs")
8 results = io.read_data("file.cfs")
9
10 # Perform interpolation
11 results_interpolated = interpolators.interpolate_node_to_cell(
12     mesh=mesh,
13     result=results,
14     regions=["V_air"],
15     quantity_names={"elecPotential": "interpolated_elecPotential"},
16 )
17
18 # Add interpolated result to results container
19 results.combine_with(results_interpolated)
20
21 # Check results container
22 print(results)
```

# CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3 from pyCFS.data.operators import interpolators
4
5 # Read source file
6 print(io.file_info("file.cfs"))
7 mesh = io.read_mesh("file.cfs")
8 results = io.read_data("file.cfs")
9
10 # Perform interpolation
11 results_interpolated = interpolators.interpolate_node_to_cell(
12     mesh=mesh,
13     result=results,
14     regions=["V_air"],
15     quantity_names={"elecPotential": "interpolated_elecPotential"},
16 )
17
18 # Add interpolated result to results container
19 results.combine_with(results_interpolated)
20
21 # Check results container
22 print(results)
```

# CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3 from pyCFS.data.operators import interpolators
4
5 # Read source file
6 print(io.file_info("file.cfs"))
7 mesh = io.read_mesh("file.cfs")
8 results = io.read_data("file.cfs")
9
10 # Perform interpolation
11 results_interpolated = interpolators.interpolate_node_to_cell(
12     mesh=mesh,
13     result=results,
14     regions=["V_air"],
15     quantity_names={"elecPotential": "interpolated_elecPotential"},
16 )
17
18 # Add interpolated result to results container
19 results.combine_with(results_interpolated)
20
21 # Check results container
22 print(results)
```

# INTERACTIVE MODE IN VS CODE (1)

```
tutorial_operators.py x
tutorial > tutorial_operators.py > ...
Run Cell | Run Below | Debug Cell
1 # %%
2 # Import necessary modules
3 from pyCFS.data import io
4 from pyCFS.data.operators import interpolators
5
6 Run Cell | Run Above | Debug Cell
7 # %%
8 # Read source file
9 print(io.file_info("file.cfs"))
10 mesh = io.read_mesh("file.cfs")
11 results = io.read_data("file.cfs")
12
13 Run Cell | Run Above | Debug Cell
14 # %%
15 # Perform interpolation
16 results_interpolated = interpolators.interpolate_node_to_cell(
17     mesh=mesh,
18     result=results,
19     regions=["V_air"],
20     quantity_names={"elecPotential": "interpolated_elecPotential"},
21 )
22
23 Run Cell | Run Above | Debug Cell
24 # %%
25 # Add interpolated result to results container
26 results.combine_with(results_interpolated)
27
28 Run Cell | Run Above | Debug Cell
29 # %%
30 # Check results container
31 print(results)
32
33 Run Cell | Run Above | Debug Cell
34 # %%
35 # Write output file
36 io.write_file("file_out.cfs", mesh=mesh, result=results)
37
38 Interactive-1 x
Interrupt | Clear All | View data | Restart | Jupyter Variables | Save | Export | ...
pycfs (3.10.12) (Python 3.10.12)

✓ # Import necessary modules ...
✓ # Read source file ...
...
Mesh
- Dimension: 3
- Nodes: 6197
- Elements: 5369
- Regions: 12
  - Group : P0_elem (3D, 8 nodes, 1 elements)
  - Group : P0_node (0D, 1 nodes, 1 elements)
  - Group : P1_elem (3D, 8 nodes, 1 elements)
  - Group : P1_node (0D, 1 nodes, 1 elements)
  - Group : P2_elem (3D, 8 nodes, 1 elements)
  - Group : P2_node (0D, 1 nodes, 1 elements)
  - Group : P3_elem (3D, 8 nodes, 1 elements)
  - Group : P3_node (0D, 1 nodes, 1 elements)
  - Region: S_bottom (2D, 69 nodes, 55 elements)
  - Region: S_top (2D, 69 nodes, 55 elements)
  - Region: V_air (3D, 5849 nodes, 4870 elements)
  - Region: V_elec (3D, 552 nodes, 385 elements)
MultiStep 1: static, 1 steps
- 'elecFieldIntensity' (real) defined in 'V_air' on Elements
- 'elecFieldIntensity' (real) defined in 'V_elec' on Elements
- 'elecFieldIntensity' (real) defined in 'P0_elem' on Elements
- 'elecFieldIntensity' (real) defined in 'P1_elem' on Elements
- 'elecFieldIntensity' (real) defined in 'P2_elem' on Elements
- 'elecFieldIntensity' (real) defined in 'P3_elem' on Elements
- 'elecFluxDensity' (real) defined in 'V_air' on Elements
- 'elecFluxDensity' (real) defined in 'V_elec' on Elements
- 'elecPotential' (real) defined in 'V_air' on Nodes
- 'elecPotential' (real) defined in 'V_elec' on Nodes
- 'elecPotential' (real) defined in 'P0_node' on Nodes
- 'elecPotential' (real) defined in 'P1_node' on Nodes
- 'elecPotential' (real) defined in 'P2_node' on Nodes
- 'elecPotential' (real) defined in 'P3_node' on Nodes
- 'elecCharge' (real) defined in 'S_top' on ElementGroup
- 'elecEnergy' (real) defined in 'V_air' on Regions
- 'elecEnergy' (real) defined in 'V_elec' on Regions
```

## INTERACTIVE MODE IN VS CODE (2)

```
tutorial_operators.py x
tutorial > tutorial_operators.py > ...
Run Cell | Run Below | Debug Cell
1 # %%
2 # Import necessary modules
3 from pyCFS.data import io
4 from pyCFS.data.operators import interpolators
5
Run Cell | Run Above | Debug Cell
6 # %%
7 # Read source file
8 print(io.file_info("file.cfs"))
9 mesh = io.read_mesh("file.cfs")
10 results = io.read_data("file.cfs")
11
Run Cell | Run Above | Debug Cell
12 # %%
13 # Perform interpolation
14 results_interpolated = interpolators.interpolate_node_to_cell(
15     mesh=mesh,
16     result=results,
17     regions=["V_air"],
18     quantity_names={"elecPotential": "interpolated_elecPotential"},
19 )
20
Run Cell | Run Above | Debug Cell
21 # %%
22 # Add interpolated result to results container
23 results.combine_with(results_interpolated)
24
25 # Check results container
26 print(results)
27
Run Cell | Run Above | Debug Cell
28 # %%
29 # Write output file
30 io.write_file("file_out.cfs", mesh=mesh, result=results)
31
```

```
Interactive-1 x
Interrupt | x Clear All View data Restart Jupyter Variables Save Export ... pycfs (3.10.12) (Python 3.10.12)
- 'elecFluxDensity' (real) defined in 'V_elec' on Elements
- 'elecPotential' (real) defined in 'V_air' on Nodes
- 'elecPotential' (real) defined in 'V_elec' on Nodes
- 'elecPotential' (real) defined in 'P0_node' on Nodes
- 'elecPotential' (real) defined in 'P1_node' on Nodes
- 'elecPotential' (real) defined in 'P2_node' on Nodes
- 'elecPotential' (real) defined in 'P3_node' on Nodes
- 'elecCharge' (real) defined in 'S_top' on ElementGroup
- 'elecEnergy' (real) defined in 'V_air' on Regions
- 'elecEnergy' (real) defined in 'V_elec' on Regions

✓ # Perform interpolation ...
... Compute interpolation matrix: "V air"
Creating interpolation matrix: [ ] 4870/4870 | Elapsed time: 0:00:01
Perform interpolation (elecPotential): "V air"
Warning: Array shape (1, 5849, 1) is not unique! This can lead to unexpected results.
Performing interpolation: [ ] 1/1 | Elapsed time: 0:00:00

✓ # Add interpolated result to results container ...
... MultiStep 1: static, 1 steps
- 'elecFieldIntensity' (real) defined in 'V_air' on Elements
- 'elecFieldIntensity' (real) defined in 'V_elec' on Elements
- 'elecFieldIntensity' (real) defined in 'P0_elem' on Elements
- 'elecFieldIntensity' (real) defined in 'P1_elem' on Elements
- 'elecFieldIntensity' (real) defined in 'P2_elem' on Elements
- 'elecFieldIntensity' (real) defined in 'P3_elem' on Elements
- 'elecFluxDensity' (real) defined in 'V_air' on Elements
- 'elecFluxDensity' (real) defined in 'V_elec' on Elements
- 'elecPotential' (real) defined in 'V_air' on Nodes
- 'elecPotential' (real) defined in 'V_elec' on Nodes
- 'elecPotential' (real) defined in 'P0_node' on Nodes
- 'elecPotential' (real) defined in 'P1_node' on Nodes
- 'elecPotential' (real) defined in 'P2_node' on Nodes
- 'elecPotential' (real) defined in 'P3_node' on Nodes
- 'elecCharge' (real) defined in 'S_top' on ElementGroup
- 'elecEnergy' (real) defined in 'V_air' on Regions
- 'elecEnergy' (real) defined in 'V_elec' on Regions
- 'interpolated_elecPotential' (real) defined in 'V_air' on Elements
```

# INTERACTIVE MODE IN VS CODE (3)

```
tutorial_operators.py x
tutorial > tutorial_operators.py > ...
Run Cell | Run Below | Debug Cell
1 # %%
2 # Import necessary modules
3 from pyCFS.data import io
4 from pyCFS.data.operators import interpolators
5
6 Run Cell | Run Above | Debug Cell
7 # %%
8 # Read source file
9 print(io.file_info("file.cfs"))
10 mesh = io.read_mesh("file.cfs")
11 results = io.read_data("file.cfs")
12
13 Run Cell | Run Above | Debug Cell
14 # %%
15 # Perform interpolation
16 results_interpolated = interpolators.interpolate_node_to_cell(
17     mesh=mesh,
18     result=results,
19     regions=["V_air"],
20     quantity_names={"elecPotential": "interpolated_elecPotential"},
21 )
22
23 Run Cell | Run Above | Debug Cell
24 # %%
25 # Add interpolated result to results container
26 results.combine_with(results_interpolated)
27
28 Run Cell | Run Above | Debug Cell
29 # %%
30 # Check results container
31 print(results)
32
33 Run Cell | Run Above | Debug Cell
34 # %%
35 # Write output file
36 io.write_file("file_out.cfs", mesh=mesh, result=results)
37
```

Interactive-1 x

Interrupt | Clear All | View data | Restart | Jupyter Variables | Save | Export | ... | pycfs (3.10.12) (Python 3.10.12)

Performing interpolation: [ ] 1/1 | Elapsed time: 0:00:00

✓ # Add interpolated result to results container...

... MultiStep 1: static, 1 steps

- 'elecFieldIntensity' (real) defined in 'V\_air' on Elements
- 'elecFieldIntensity' (real) defined in 'V\_elec' on Elements
- 'elecFieldIntensity' (real) defined in 'P0\_elem' on Elements
- 'elecFieldIntensity' (real) defined in 'P1\_elem' on Elements
- 'elecFieldIntensity' (real) defined in 'P2\_elem' on Elements
- 'elecFieldIntensity' (real) defined in 'P3\_elem' on Elements
- 'elecFluxDensity' (real) defined in 'V\_air' on Elements
- 'elecFluxDensity' (real) defined in 'V\_elec' on Elements
- 'elecPotential' (real) defined in 'V\_air' on Nodes
- 'elecPotential' (real) defined in 'V\_elec' on Nodes
- 'elecPotential' (real) defined in 'P0\_node' on Nodes
- 'elecPotential' (real) defined in 'P1\_node' on Nodes
- 'elecPotential' (real) defined in 'P2\_node' on Nodes
- 'elecPotential' (real) defined in 'P3\_node' on Nodes
- 'elecCharge' (real) defined in 'S\_top' on ElementGroup
- 'elecEnergy' (real) defined in 'V\_air' on Regions
- 'elecEnergy' (real) defined in 'V\_elec' on Regions
- 'interpolated\_elecPotential' (real) defined in 'V\_air' on Elements

✓ # Write output file...

... Creating file file\_out.cfs

Writing Mesh Data

- Writing Group: P0\_elem
- Writing Group: P0\_node
- Writing Group: P1\_elem
- Writing Group: P1\_node
- Writing Group: P2\_elem
- Writing Group: P2\_node
- Writing Group: P3\_elem
- Writing Group: P3\_node
- Writing Region: S\_bottom
- Writing Region: S\_top
- Writing Region: V\_air
- Writing Region: V\_elec

Writing Step: [ ] 1/1 | Elapsed time: 0:00:00