pyCFS

A COMPANION LIBRARY FOR openCFS

PART 2: DATA MANIPULATION

Andreas Wurzinger

MOTIVATION

INTUITIVE DATA ACCESS

- Easy install
- Easy modification
- Easy addition of new features
- Flexibility
- Comprehensive documentation!

FOCUS

- Test case generation
  - Create `.cfs` files from scratch

FOCUS

- Test case generation
  - Create `.cfs` files from scratch
- Pre-Processing
  - Mesh preparation
  - Data processing

FOCUS

- Test case generation
  - Create `.cfs` files from scratch
- Pre-Processing
  - Mesh preparation
  - Data processing
- Post-Processing
  - Compare to analytic computations
  - Plot time series (faster than ParaView)

FOCUS

- Test case generation
  - Create `.cfs` files from scratch
- Pre-Processing
  - Mesh preparation
  - Data processing
- Post-Processing
  - Compare to analytic computations
  - Plot time series (faster than ParaView)
- Small to medium size problems!
  - Many parts are parallelized
  - Some Python operations are still slow for large problems

GETTING STARTED

INSTALLATION

- Install in `pip` environment

```
pip install pyCFS
```

INSTALLATION

- Install in `pip` environment

```
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 excluded from standard install
- Install dependencies for all functionality

```
pip install pyCFS[data]
```

DOCUMENTATION

- Documentation page
  - Installation Guide

DOCUMENTATION

- Documentation page
  - Installation Guide
  - Basic usage Guide
    - Contains only some features

DOCUMENTATION

- Documentation page
  - Installation Guide
  - Basic usage Guide
    - Contains only some features
  - API-Documenation

pyCFS 0.1.1

Search Ctrl + K

Contents:

API Documentation

Information on specific functions, classes, and methods.

Previous
pyCFS-data

FUNCTIONALITY

OVERVIEW

Structured into submodules

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

- `io`
  - I/O operations for CFS type HDF5 format

OVERVIEW

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

OVERVIEW

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*

OVERVIEW

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 compatibilty methods to other file formats
  - Additional functionality not directly related to *openCFS*

I/O (CFSReader)

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

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

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

I/O (CFSReader)

Usage

```
with CFSReader(filename="file.cfs") as reader:
    # Print file information
    print(reader)

    # Read the whole mesh
    mesh = reader.MeshData

    # Read coordinates, connectivity
    coordinates = reader.Coordinates
    connectivity = reader.Connectivity

    # Read node coordinates of a specific region
    reg_1 = reader.get_mesh_region_coordinates(region="S_CAPACITOR")

    # Read all result data for sequence step 2
    reader.set_multi_step(multi_step_id=2)
    results_2 = reader.MultiStepData

    # Read data for a specific quantity and region
    result_1 = reader.get_multi_step_data(multi_step_id=1, quantitie
```

I/O (CFSReader)

Usage

```
with CFSReader(filename="file.cfs") as reader:
    # Print file information
    print(reader)

    # Read the whole mesh
    mesh = reader.MeshData

    # Read coordinates, connectivity
    coordinates = reader.Coordinates
    connectivity = reader.Connectivity

    # Read node coordinates of a specific region
    reg_1 = reader.get_mesh_region_coordinates(region="S_CAPACITOR")

    # Read all result data for sequence step 2
    reader.set_multi_step(multi_step_id=2)
    results_2 = reader.MultiStepData

    # Read data for a specific quantity and region
    result_1 = reader.get_multi_step_data(multi_step_id=1, quantitie
```

I/O (CFSReader)

Usage

```
with CFSReader(filename="file.cfs") as reader:
    # Print file information
    print(reader)

    # Read the whole mesh
    mesh = reader.MeshData

    # Read coordinates, connectivity
    coordinates = reader.Coordinates
    connectivity = reader.Connectivity

    # Read node coordinates of a specific region
    reg_1 = reader.get_mesh_region_coordinates(region="S_CAPACITOR")

    # Read all result data for sequence step 2
    reader.set_multi_step(multi_step_id=2)
    results_2 = reader.MultiStepData

    # Read data for a specific quantity and region
    result_1 = reader.get_multi_step_data(multi_step_id=1, quantitie
```

I/O (`CFSWriter`)

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

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

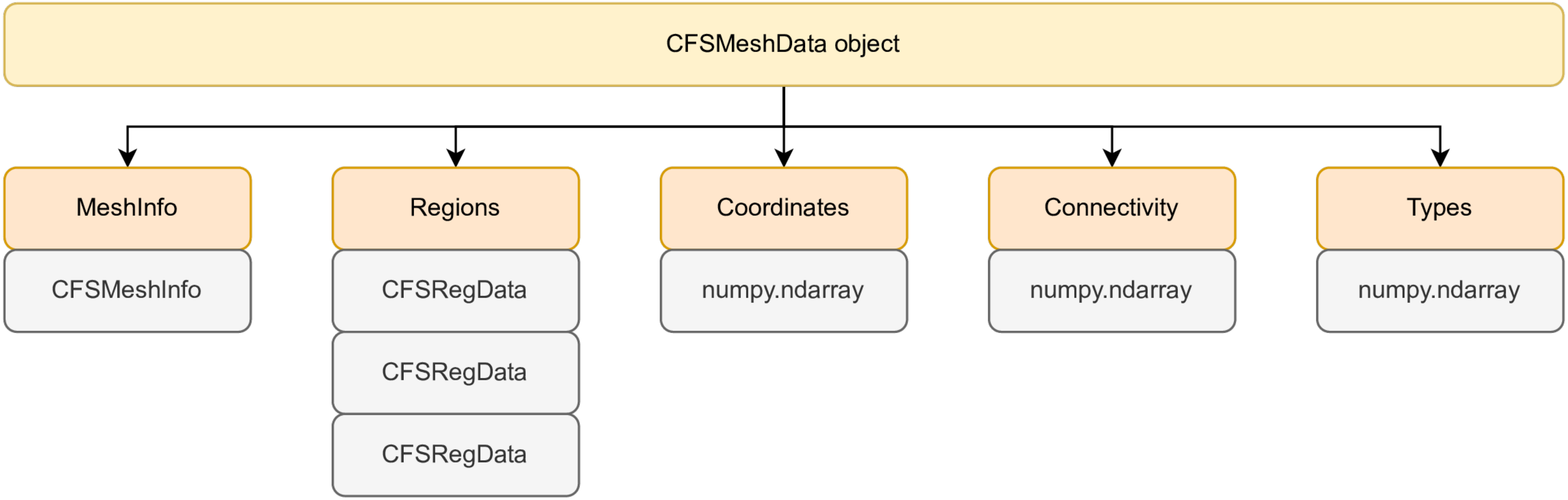
Usage

```
with CFSWriter(filename="file.cfs") as writer:
    # Create new file
    writer.create_file(mesh_data=mesh, result_data=result_1)

    # Write additional squence step
    writer.write_multistep(result_data=results_2, multi_step_id=2)
```

I/O (CFSMeshData)

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

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



I/O (CFSMeshData)

Usage examples

```
# Create mesh object of point cloud
mesh_points = CFSMeshData.from_coordinates_connectivity(coordinates=

# Create mesh object from coordinates and connectivity
mesh = CFSMeshData.from_coordinates_connectivity(
    coordinates=coordinates, connectivity=connectivity, element_dime
)

# Merge mesh objects
mesh = mesh + mesh_points

# Print information
print(mesh)

# Compute element normals for a region
mesh.get_region_centroids(region="S_plate")

# Get closest node/element to a coordinate
mesh.get_closest_node(coordinate=[0.1, 0.2, 0.3], region="S_plate")
mesh.get_closest_element(coordinate=[0.1, 0.2, 0.3], region="S_plate
# Split mesh into regions by element clusters
```

I/O (`CFSMeshData`)

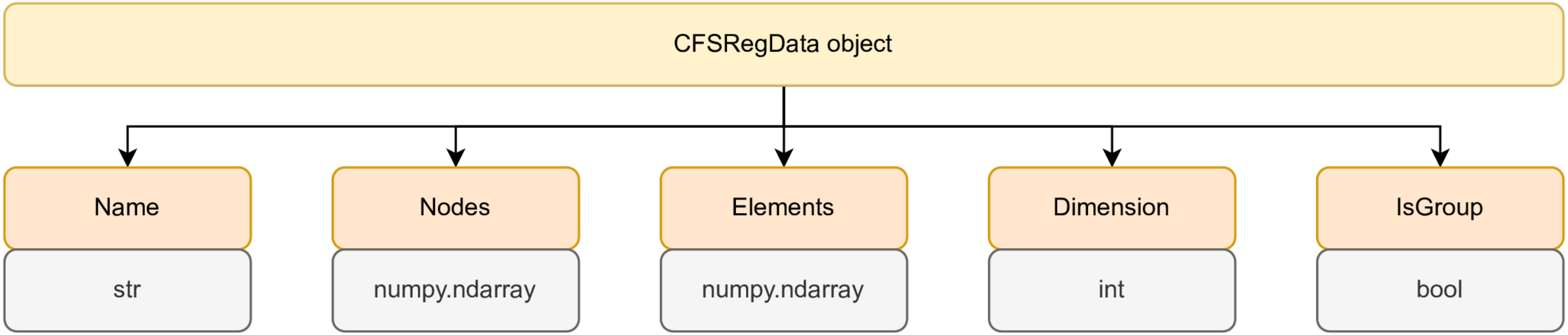
Usage examples

```python
# Create mesh object of point cloud
mesh_points = CFSMeshData.from_coordinates_connectivity(coordinates=

# Create mesh object from coordinates and connectivity
mesh = CFSMeshData.from_coordinates_connectivity(
    coordinates=coordinates, connectivity=connectivity, element_dime
)

# Merge mesh objects
mesh = mesh + mesh_points

# Print information
print(mesh)

# Compute element normals for a region
mesh.get_region_centroids(region="S_plate")

# Get closest node/element to a coordinate
mesh.get_closest_node(coordinate=[0.1, 0.2, 0.3], region="S_plate")
mesh.get_closest_element(coordinate=[0.1, 0.2, 0.3], region="S_plate

# Split mesh into regions by element clusters
```

I/O (`CFSRegData`)

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

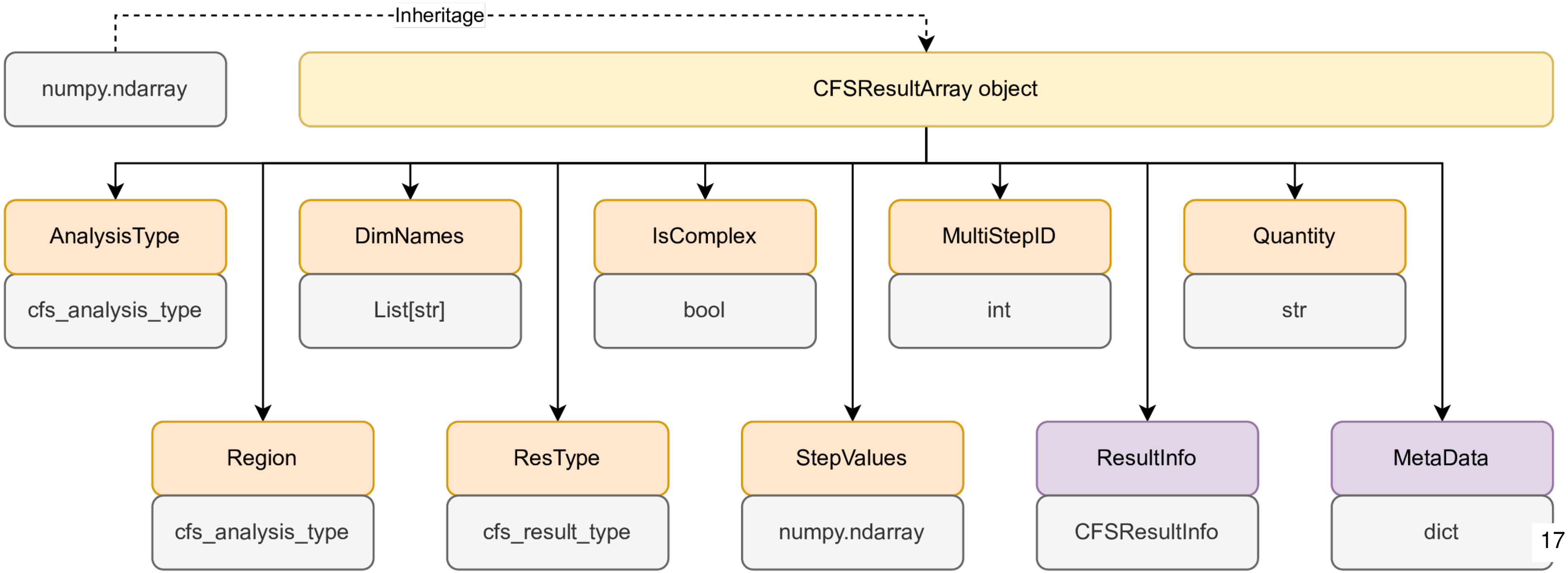
- Container object for all region related data



I/O (`CFSResultArray`)

```python
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`)

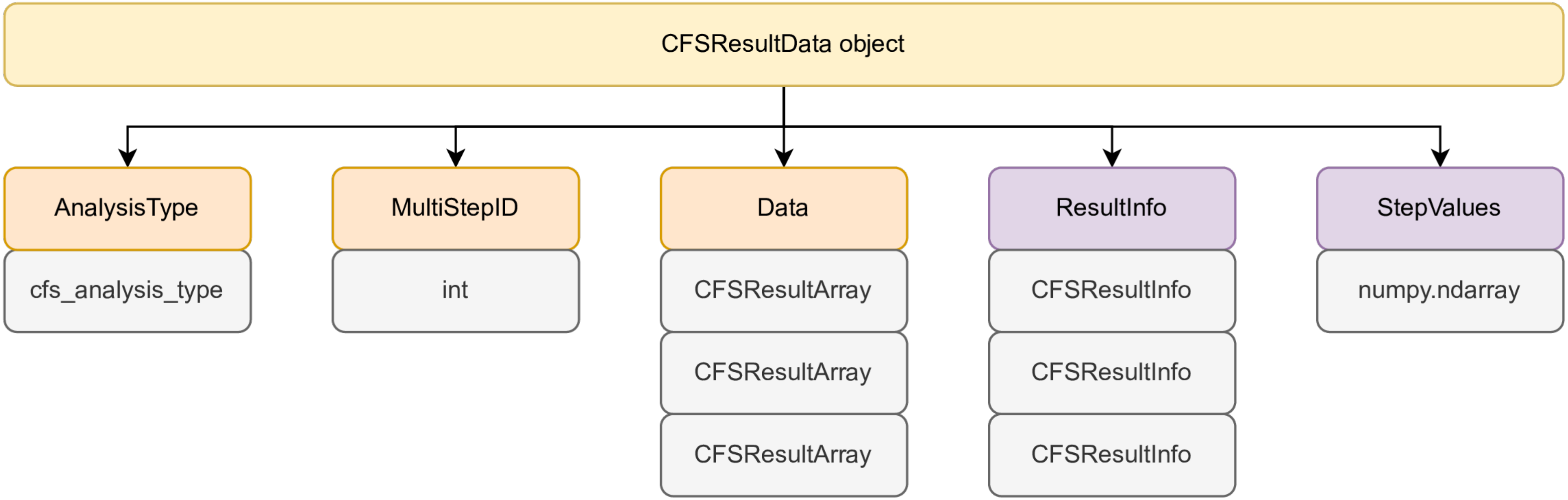
Usage examples

```
# 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`)

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

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



I/O (CFSResultData)

Usage examples

```
# Create a result container object
result = CFSResultData(analysis_type=cfs_analysis_type.TRANSIENT,
                       multi_step_id=2, data=[array_1, array_2])

# Print information
print(result)

# Extract certain time steps
result_1 = result[0:5]

# Extract certain region and quantity
result_2 = result.extract_quantity_region(quantity="elecPotential",

# Add data to result object (define different multi step ID)
result.add_data_array(data=cfs_array, multi_step_id=2)
```

I/O (CFSResultData)

Usage examples

```
# Create a result container object
result = CFSResultData(analysis_type=cfs_analysis_type.TRANSIENT,
                        multi_step_id=2, data=[array_1, array_2])

# Print information
print(result)

# Extract certain time steps
result_1 = result[0:5]

# Extract certain region and quantity
result_2 = result.extract_quantity_region(quantity="elecPotential",

# Add data to result object (define different multi step ID)
result.add_data_array(data=cfs_array, multi_step_id=2)
```

I/O (`CFSResultData`)

Usage examples

```
# Create a result container object
result = CFSResultData(analysis_type=cfs_analysis_type.TRANSIENT,
                       multi_step_id=2, data=[array_1, array_2])

# Print information
print(result)

# Extract certain time steps
result_1 = result[0:5]

# Extract certain region and quantity
result_2 = result.extract_quantity_region(quantity="elecPotential",

# Add data to result object (define different multi step ID)
result.add_data_array(data=cfs_array, multi_step_id=2)
```

I/O (OTHER)

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

- `cfs_types`
  - Enum definitions based on *openCFS* source code

I/O (OTHER)

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

- `cfs_types`
  - Enum definitions based on *openCFS* source code
- `cfs_util`
  - Functions to check object validity

OPERATORS

```python
from pyCFS.data.operators import (transformation, interpolators,
                                  projection_interpolation, sngr)
```

- `interpolators`
  - Basic interpolators
    - Node2Cell
    - Cell2Node
    - Nearest Neighbor (bidirectional)

OPERATORS

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

- `interpolators`
  - Basic interpolators
    - Node2Cell
    - Cell2Node
    - Nearest Neighbor (bidirectional)
- `projection_interpolation`
  - Projection-based interpolation

OPERATORS

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

- `transformation`
  - Translate / rotate / extrude / revolve mesh
  - Fit mesh onto target mesh

OPERATORS

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

- `transformation`
  - Translate / rotate / extrude / revolve mesh
  - Fit mesh onto target mesh
- `sngr`
  - Compute fluctuating flow field from stationary RANS solution

EXTRA FUNCTIONALITY

- Read mesh and data from various formats

EXTRA FUNCTIONALITY

- Read mesh and data from various formats
  - `ansys_io` (Ansys Mechanical: `.rst`)

EXTRA FUNCTIONALITY

- Read mesh and data from various formats
  - `ansys_io` (Ansys Mechanical: `.rst`)
  - `ensight_io` (various CFD software: `.case`)

EXTRA FUNCTIONALITY

- Read mesh and data from various formats
  - `ansys_io` (Ansys Mechanical: `.rst`)
  - `ensight_io` (various CFD software: `.case`)
  - `psv_io` (Polytec PSV export: `.unv`)

EXTRA FUNCTIONALITY

- Read mesh and data from various formats
  - `ansys_io` (Ansys Mechanical: `.rst`)
  - `ensight_io` (various CFD software: `.case`)
  - `psv_io` (Polytec PSV export: `.unv`)
  - `nihu_io` (NiHu simulation export: `.mat`)

EXTRA FUNCTIONALITY

- Read mesh and data from various formats
  - `ansys_io` (Ansys Mechanical: `.rst`)
  - `ensight_io` (various CFD software: `.case`)
  - `psv_io` (Polytec PSV export: `.unv`)
  - `nihu_io` (NiHu simulation export: `.mat`)
  - *Work in progress:* `exodus_io` (Cubit mesh export)

EXAMPLE WORKFLOW

I/O

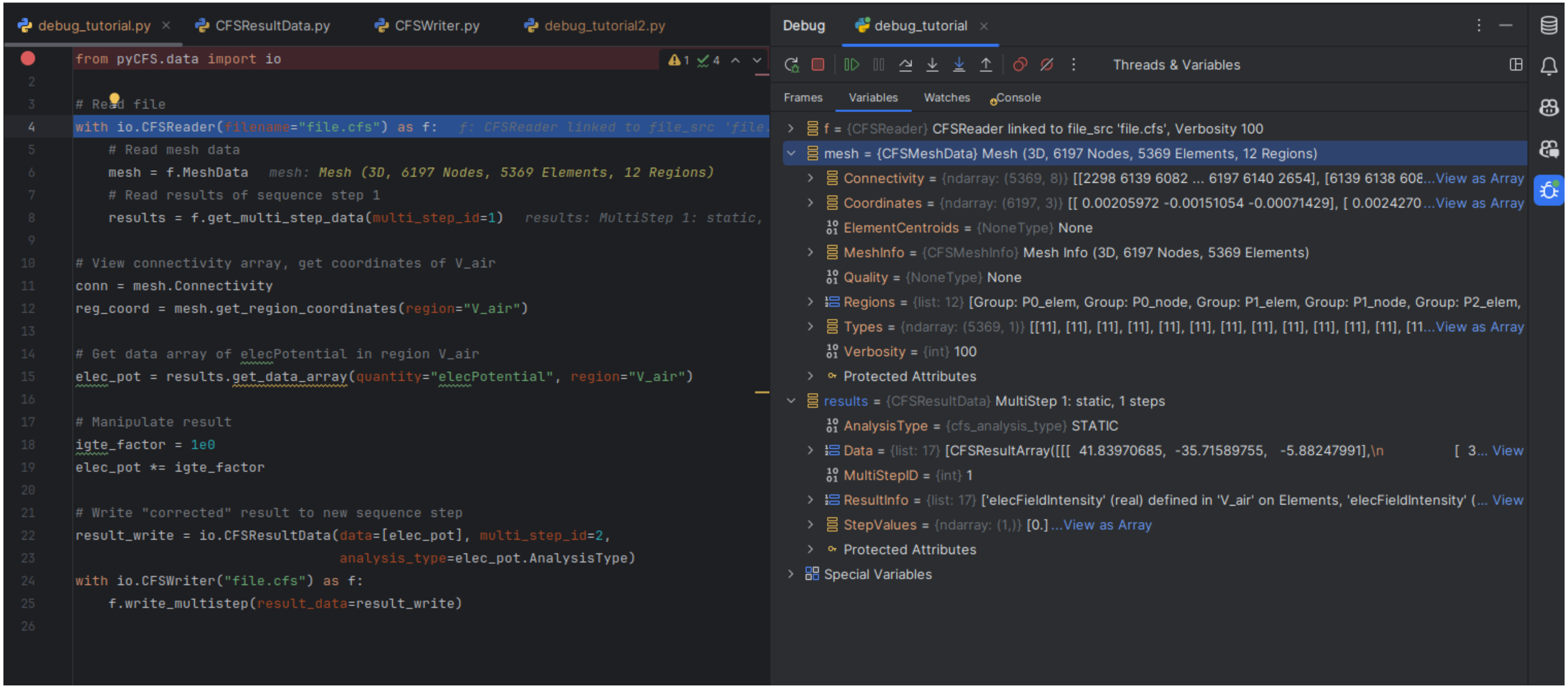
```python
# Import necessary modules
from pyCFS.data import io

# Read file
with io.CFSReader(filename="file.cfs") as f:
    # Read mesh data
    mesh = f.MeshData
    # Read results of sequence step 1
    results = f.get_multi_step_data(multi_step_id=1)

# View connectivity array, get coordinates of V_air
conn = print(mesh.Connectivity)
reg_coord = mesh.get_region_coordinates(region="V_air")

# Get data array of elecPotential in region V_air
elec_pot = results.get_data_array(quantity="elecPotential", region="

# Manipulate result
igte_factor = 1e0
elec_pot *= igte_factor

# Write "corrected" result to new sequence step
```

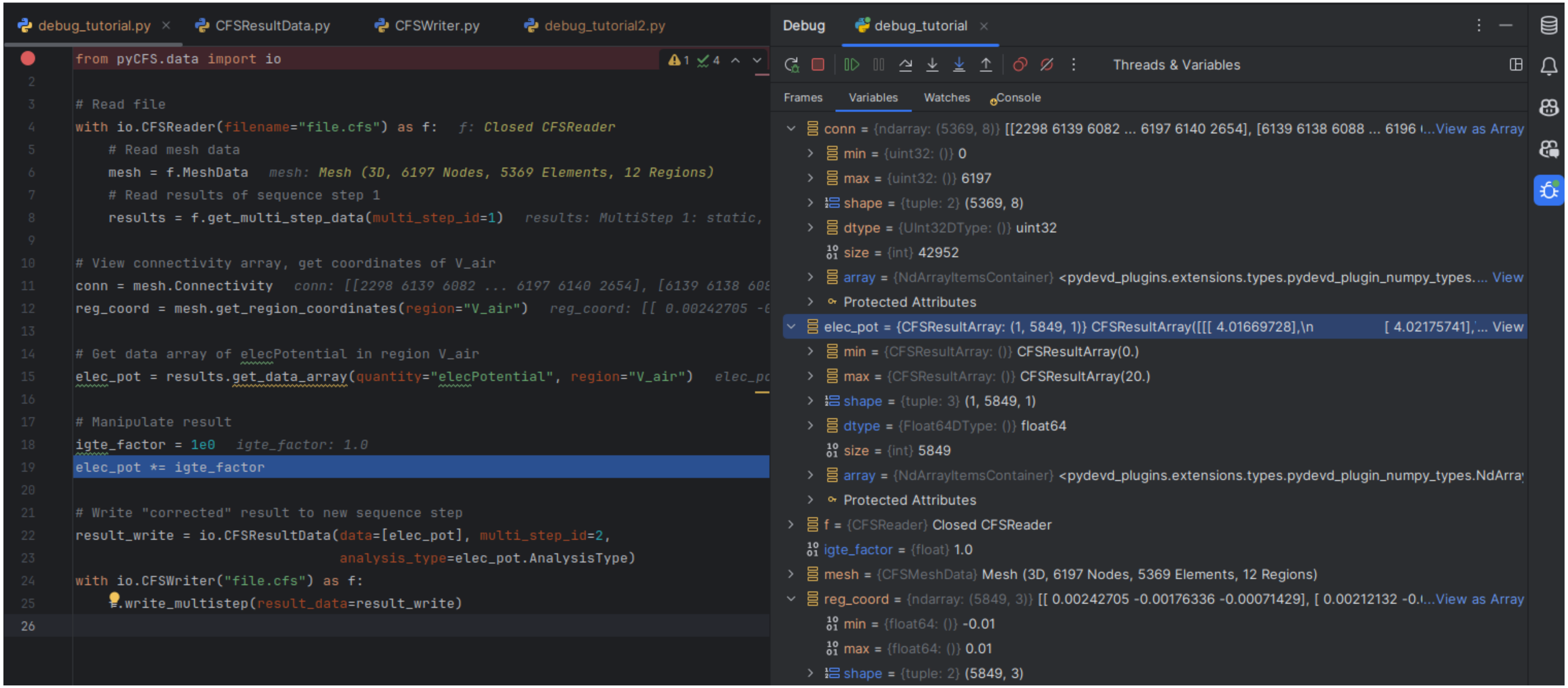
```python
# Import necessary modules
from pyCFS.data import io

# Read file
with io.CFSReader(filename="file.cfs") as f:
    # Read mesh data
    mesh = f.MeshData
    # Read results of sequence step 1
    results = f.get_multi_step_data(multi_step_id=1)

# View connectivity array, get coordinates of V_air
conn = print(mesh.Connectivity)
reg_coord = mesh.get_region_coordinates(region="V_air")

# Get data array of elecPotential in region V_air
elec_pot = results.get_data_array(quantity="elecPotential", region='

# Manipulate result
igte_factor = 1e0
elec_pot *= igte_factor

# Write "corrected" result to new sequence step
```

```python
# Import necessary modules
from pyCFS.data import io

# Read file
with io.CFSReader(filename="file.cfs") as f:
    # Read mesh data
    mesh = f.MeshData
    # Read results of sequence step 1
    results = f.get_multi_step_data(multi_step_id=1)

# View connectivity array, get coordinates of V_air
conn = print(mesh.Connectivity)
reg_coord = mesh.get_region_coordinates(region="V_air")

# Get data array of elecPotential in region V_air
elec_pot = results.get_data_array(quantity="elecPotential", region='

# Manipulate result
igte_factor = 1e0
elec_pot *= igte_factor

# Write "corrected" result to new sequence step
```

```python
# Import necessary modules
from pyCFS.data import io

# Read file
with io.CFSReader(filename="file.cfs") as f:
    # Read mesh data
    mesh = f.MeshData
    # Read results of sequence step 1
    results = f.get_multi_step_data(multi_step_id=1)

# View connectivity array, get coordinates of V_air
conn = print(mesh.Connectivity)
reg_coord = mesh.get_region_coordinates(region="V_air")

# Get data array of elecPotential in region V_air
elec_pot = results.get_data_array(quantity="elecPotential", region="

# Manipulate result
igte_factor = 1e0
elec_pot *= igte_factor

# Write "corrected" result to new sequence step
```

```python
# Import necessary modules
from pyCFS.data import io

# Read file
with io.CFSReader(filename="file.cfs") as f:
    # Read mesh data
    mesh = f.MeshData
    # Read results of sequence step 1
    results = f.get_multi_step_data(multi_step_id=1)

# View connectivity array, get coordinates of V_air
conn = print(mesh.Connectivity)
reg_coord = mesh.get_region_coordinates(region="V_air")

# Get data array of elecPotential in region V_air
elec_pot = results.get_data_array(quantity="elecPotential", region='

# Manipulate result
igte_factor = 1e0
elec_pot *= igte_factor

# Write "corrected" result to new sequence step
```


debug_tutorial.py
CFSResultData.py
CFSWriter.py
debug_tutorial2.py
from pyCFS.data import io
# Read file
with io.CFSReader(filename="file.cfs") as f:
# Read mesh data
mesh = f.MeshData
mesh: Mesh (3D, 6197 Nodes, 5369 Elements, 12 Regions)
# Read results of sequence step 1
results = f.get_multi_step_data(multi_step_id=1)
results: MultiStep 1: static,
# View connectivity array, get coordinates of V_air
conn = mesh.Connectivity
reg_coord = mesh.get_region_coordinates(region="V_air")
# Get data array of elecPotential in region V_air
elec_pot = results.get_data_array(quantity="elecPotential", region="V_air")
# Manipulate result
igte_factor = 1e0
elec_pot *= igte_factor
# Write "corrected" result to new sequence step
result_write = io.CFSResultData(data=[elec_pot], multi_step_id=2,
analysis_type=elec_pot.AnalysisType)
with io.CFSWriter("file.cfs") as f:
f.write_multistep(result_data=result_write)
Debug
debug_tutorial
Threads & Variables
Frames
Variables
Watches
Console
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 608...View as Array
Coordinates = {ndarray: (6197, 3)} [[0.00205972 -0.00151054 -0.00071429], [0.0024270...View as Array
ElementCentroids = {NoneType} None
MeshInfo = {CFSMeshInfo} Mesh Info (3D, 6197 Nodes, 5369 Elements)
Quality = {NoneType} None
Regions = {list: 12} [Group: P0_elem, Group: P0_node, Group: P1_elem, Group: P1_node, Group: P2_elem,
Types = {ndarray: (5369, 1)} [[11], [11], [11], [11], [11], [11], [11], [11], [11], [11], [11], [11], [11...View as Array
Verbosity = {int} 100
Protected Attributes
results = {CFSResultData} MultiStep 1: static, 1 steps
AnalysisType = {cfs_analysis_type} STATIC
Data = {list: 17} [CFSResultArray([[[41.83970685, -35.71589755, -5.88247991],\n [3... View
MultiStepID = {int} 1
ResultInfo = {list: 17} ['elecFieldIntensity' (real) defined in 'V_air' on Elements, 'elecFieldIntensity' (... View
StepValues = {ndarray: (1,)} [0.] ...View as Array
Protected Attributes
Special Variables


debug_tutorial.py
CFSResultData.py
CFSWriter.py
debug_tutorial2.py
from pyCFS.data import io
# Read file
with io.CFSReader(filename="file.cfs") as f: f: Closed CFSReader
# Read mesh data
mesh = f.MeshData   mesh: Mesh (3D, 6197 Nodes, 5369 Elements, 12 Regions)
# Read results of sequence step 1
results = f.get_multi_step_data(multi_step_id=1)   results: MultiStep 1: static,
# View connectivity array, get coordinates of V_air
conn = mesh.Connectivity   conn: [[2298 6139 6082 ... 6197 6140 2654], [6139 6138 608
reg_coord = mesh.get_region_coordinates(region="V_air")   reg_coord: [[0.00242705 -0
# Get data array of elecPotential in region V_air
elec_pot = results.get_data_array(quantity="elecPotential", region="V_air")
# Manipulate result
igte_factor = 1e0   igte_factor: 1.0
elec_pot *= igte_factor
# Write "corrected" result to new sequence step
result_write = io.CFSResultData(data=[elec_pot], multi_step_id=2,
analysis_type=elec_pot.AnalysisType)
with io.CFSWriter("file.cfs") as f:
f.write_multistep(result_data=result_write)
Debug
debug_tutorial
Threads & Variables
Frames
Variables
Watches
Console
conn = {ndarray: (5369, 8)} [[2298 6139 6082 ... 6197 6140 2654], [6139 6138 6088 ... 6196 ...View as Array
min = {uint32: ()} 0
max = {uint32: ()} 6197
shape = {tuple: 2} (5369, 8)
dtype = {UInt32DType: ()} uint32
size = {int} 42952
array = {NdArrayItemsContainer} <pydevd_plugins.extensions.types.pydevd_plugin_numpy_types.... View
Protected Attributes
elec_pot = {CFSResultArray: (1, 5849, 1)} CFSResultArray([[[4.01669728],\n [4.02175741],... View
min = {CFSResultArray: ()} CFSResultArray(0.)
max = {CFSResultArray: ()} CFSResultArray(20.)
shape = {tuple: 3} (1, 5849, 1)
dtype = {Float64DType: ()} float64
size = {int} 5849
array = {NdArrayItemsContainer} <pydevd_plugins.extensions.types.pydevd_plugin_numpy_types.NdArray
Protected Attributes
f = {CFSReader} Closed CFSReader
igte_factor = {float} 1.0
mesh = {CFSMeshData} Mesh (3D, 6197 Nodes, 5369 Elements, 12 Regions)
reg_coord = {ndarray: (5849, 3)} [[0.00242705 -0.00176336 -0.00071429], [0.00212132 -0....View as Array
min = {float64: ()} -0.01
max = {float64: ()} 0.01
shape = {tuple: 2} (5849, 3)

EXAMPLE WORKFLOW

Operators

```python
# Import necessary modules
from pyCFS.data import io
from pyCFS.data.operators import interpolators

# Read source file
with io.CFSReader(filename="file.cfs") as h5r:
    print(h5r)
    mesh = h5r.MeshData
    results = h5r.MultiStepData

# Perform interpolation
results_interpolated = interpolators.interpolate_node_to_cell(
    mesh_data=mesh,
    result_data=results,
    regions=["V_air"],
    quantity_names={"elecPotential": "interpolated_elecPotential"},
)

# Add interpolated result to results container
results.combine_with(results_interpolated)

# Check results container
```

EXAMPLE WORKFLOW

Operators

```
# Import necessary modules
from pyCFS.data import io
from pyCFS.data.operators import interpolators

# Read source file
with io.CFSReader(filename="file.cfs") as h5r:
    print(h5r)
    mesh = h5r.MeshData
    results = h5r.MultiStepData

# Perform interpolation
results_interpolated = interpolators.interpolate_node_to_cell(
    mesh_data=mesh,
    result_data=results,
    regions=["V_air"],
    quantity_names={"elecPotential": "interpolated_elecPotential"},
)

# Add interpolated result to results container
results.combine_with(results_interpolated)

# Check results container
```

EXAMPLE WORKFLOW

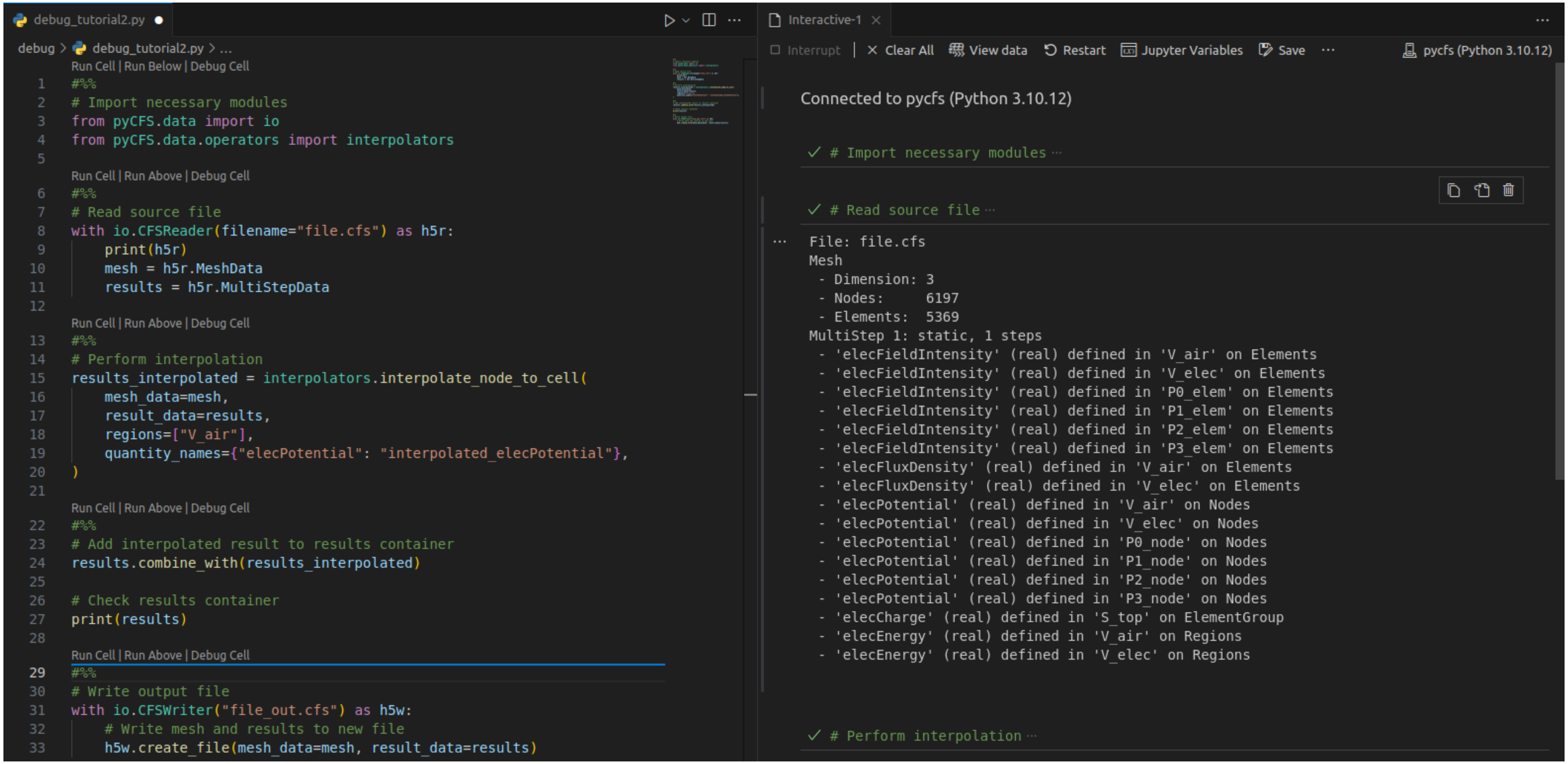
Operators

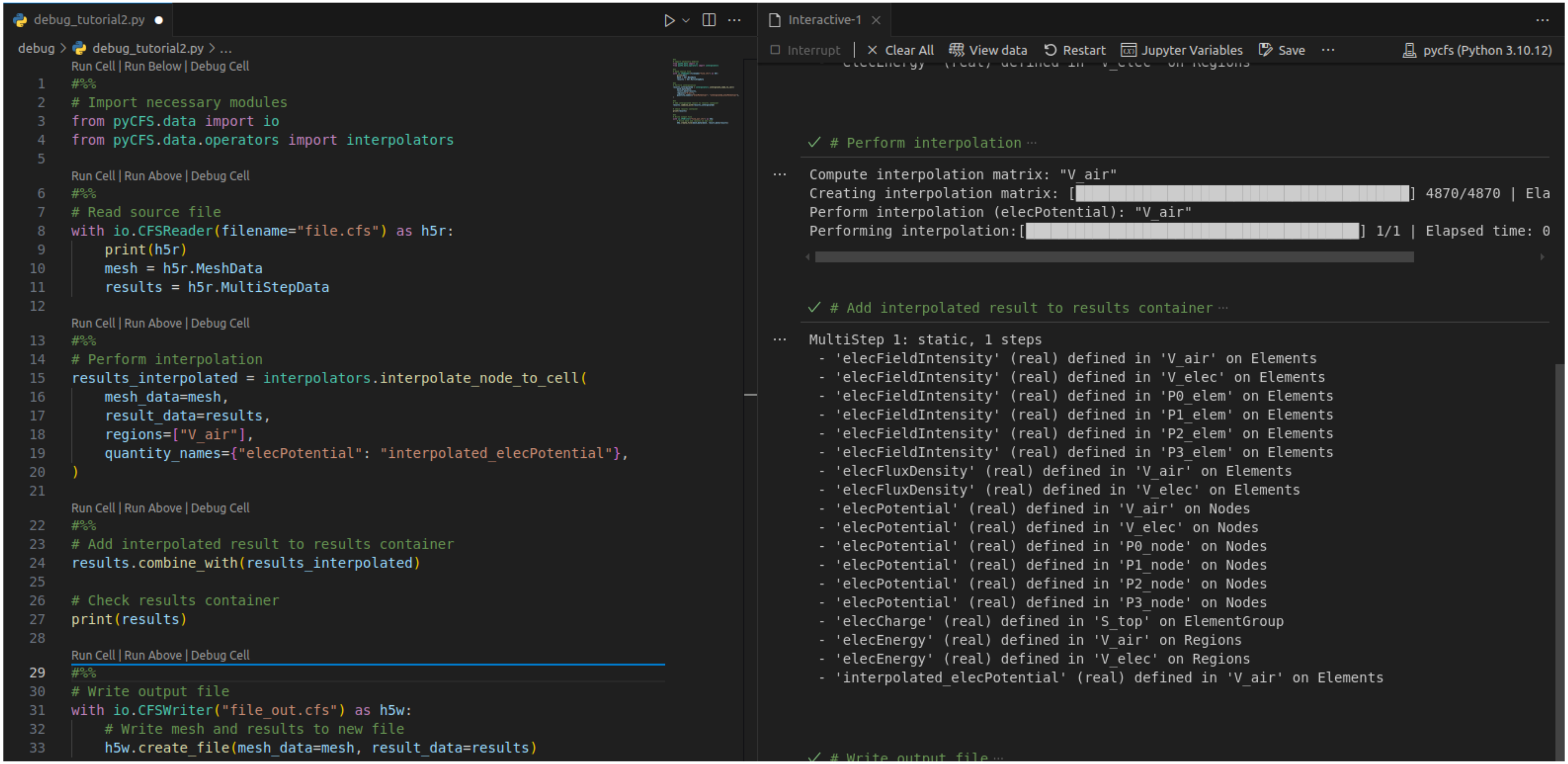
```python
# Import necessary modules
from pyCFS.data import io
from pyCFS.data.operators import interpolators

# Read source file
with io.CFSReader(filename="file.cfs") as h5r:
    print(h5r)
    mesh = h5r.MeshData
    results = h5r.MultiStepData

# Perform interpolation
results_interpolated = interpolators.interpolate_node_to_cell(
    mesh_data=mesh,
    result_data=results,
    regions=["V_air"],
    quantity_names={"elecPotential": "interpolated_elecPotential"},
)

# Add interpolated result to results container
results.combine_with(results_interpolated)

# Check results container
```

EXAMPLE WORKFLOW

Operators

```python
# Import necessary modules
from pyCFS.data import io
from pyCFS.data.operators import interpolators

# Read source file
with io.CFSReader(filename="file.cfs") as h5r:
    print(h5r)
    mesh = h5r.MeshData
    results = h5r.MultiStepData

# Perform interpolation
results_interpolated = interpolators.interpolate_node_to_cell(
    mesh_data=mesh,
    result_data=results,
    regions=["V_air"],
    quantity_names={"elecPotential": "interpolated_elecPotential"},
)

# Add interpolated result to results container
results.combine_with(results_interpolated)

# Check results container
```


debug_tutorial2.py
debug > debug_tutorial2.py > ...
Run Cell | Run Below | Debug Cell
#%%
# Import necessary modules
from pyCFS.data import io
from pyCFS.data.operators import interpolators
Run Cell | Run Above | Debug Cell
#%%
Read source file
with io.CFSReader(filename="file.cfs") as h5r:
    print(h5r)
    mesh = h5r.MeshData
    results = h5r.MultiStepData
Run Cell | Run Above | Debug Cell
#%%
# Perform interpolation
results_interpolated = interpolators.interpolate_node_to_cell(
    mesh_data=mesh,
    result_data=results,
    regions=["V_air"],
    quantity_names={"elecPotential": "interpolated_elecPotential"},
)
Run Cell | Run Above | Debug Cell
#%%
# Add interpolated result to results container
results.combine_with(results_interpolated)
# Check results container
print(results)
Run Cell | Run Above | Debug Cell
#%%
Write output file
with io.CFSWriter("file_out.cfs") as h5w:
    # Write mesh and results to new file
    h5w.create_file(mesh_data=mesh, result_data=results)
Interactive-1
Interrupt | Clear All | View data | Restart | Jupyter Variables | Save
pycfs (Python 3.10.12)
Connected to pycfs (Python 3.10.12)
Import necessary modules
Read source file
File: file.cfs
Mesh
 - Dimension: 3
 - Nodes: 6197
 - Elements: 5369
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
Perform interpolation


debug_tutorial2.py
debug > debug_tutorial2.py > ...
Run Cell | Run Below | Debug Cell
#%%
# Import necessary modules
from pyCFS.data import io
from pyCFS.data.operators import interpolators
Run Cell | Run Above | Debug Cell
#%%
Read source file
with io.CFSReader(filename="file.cfs") as h5r:
print(h5r)
mesh = h5r.MeshData
results = h5r.MultiStepData
Run Cell | Run Above | Debug Cell
#%%
# Perform interpolation
results_interpolated = interpolators.interpolate_node_to_cell(
mesh_data=mesh,
result_data=results,
regions=["V_air"],
quantity_names={"elecPotential": "interpolated_elecPotential"},
)
Run Cell | Run Above | Debug Cell
#%%
# Add interpolated result to results container
results.combine_with(results_interpolated)
# Check results container
print(results)
Run Cell | Run Above | Debug Cell
#%%
Write output file
with io.CFSWriter("file_out.cfs") as h5w:
# Write mesh and results to new file
h5w.create_file(mesh_data=mesh, result_data=results)
Interactive-1
Interrupt
Clear All
View data
Restart
Jupyter Variables
Save
pycfs (Python 3.10.12)
Perform interpolation
Compute interpolation matrix: "V_air"
Creating interpolation matrix: [] 4870/4870 | Ela
Perform interpolation (elecPotential): "V_air"
Performing interpolation:[] 1/1 | Elapsed time: 0
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

debug_tutorial2.py

debug > debug_tutorial2.py > ...

```python
#%%
# Import necessary modules
from pyCFS.data import io
from pyCFS.data.operators import interpolators

#%%
# Read source file
with io.CFSReader(filename="file.cfs") as h5r:
    print(h5r)
    mesh = h5r.MeshData
    results = h5r.MultiStepData

#%%
# Perform interpolation
results_interpolated = interpolators.interpolate_node_to_cell(
    mesh_data=mesh,
    result_data=results,
    regions=["V_air"],
    quantity_names={"elecPotential": "interpolated_elecPotential"},
)

#%%
# Add interpolated result to results container
results.combine_with(results_interpolated)

# Check results container
print(results)

#%%
# Write output file
with io.CFSWriter("file_out.cfs") as h5w:
    # Write mesh and results to new file
    h5w.create_file(mesh_data=mesh, result_data=results)
```

Interactive-1

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

✓ # 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
```