

pyCFS

A COMPANION LIBRARY FOR **openCFS**

PART 2: DATA MANIPULATION

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
 - Basic usage Guide
 - Contains only some features
 - [API-Documentation](#)

FUNCTIONALITY

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

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

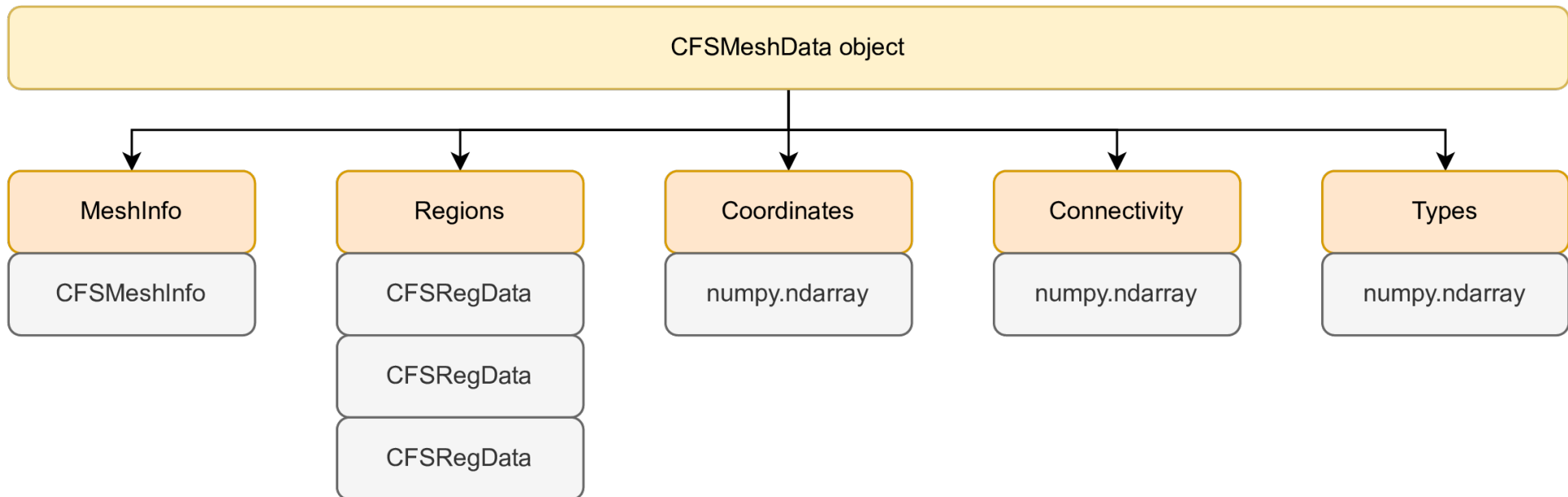
Usage

```
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)

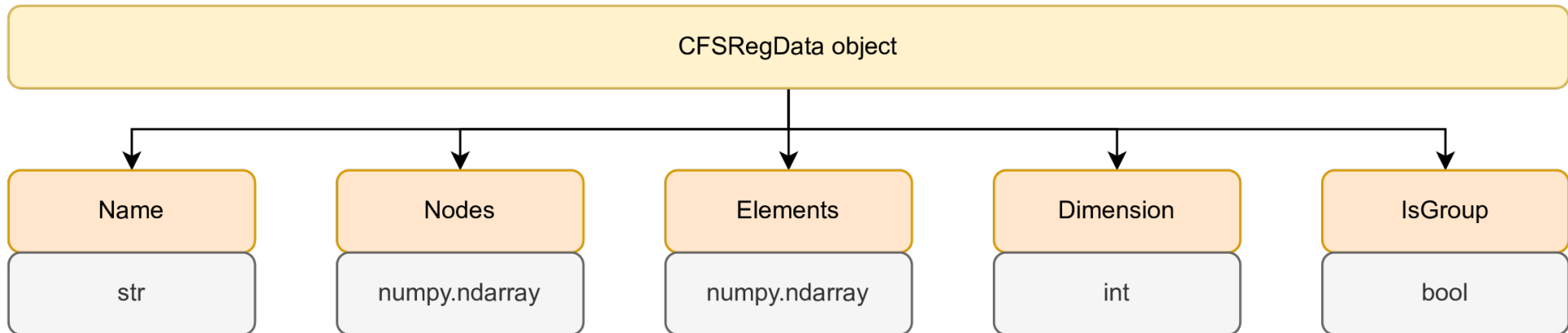
Usage examples

```
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
```

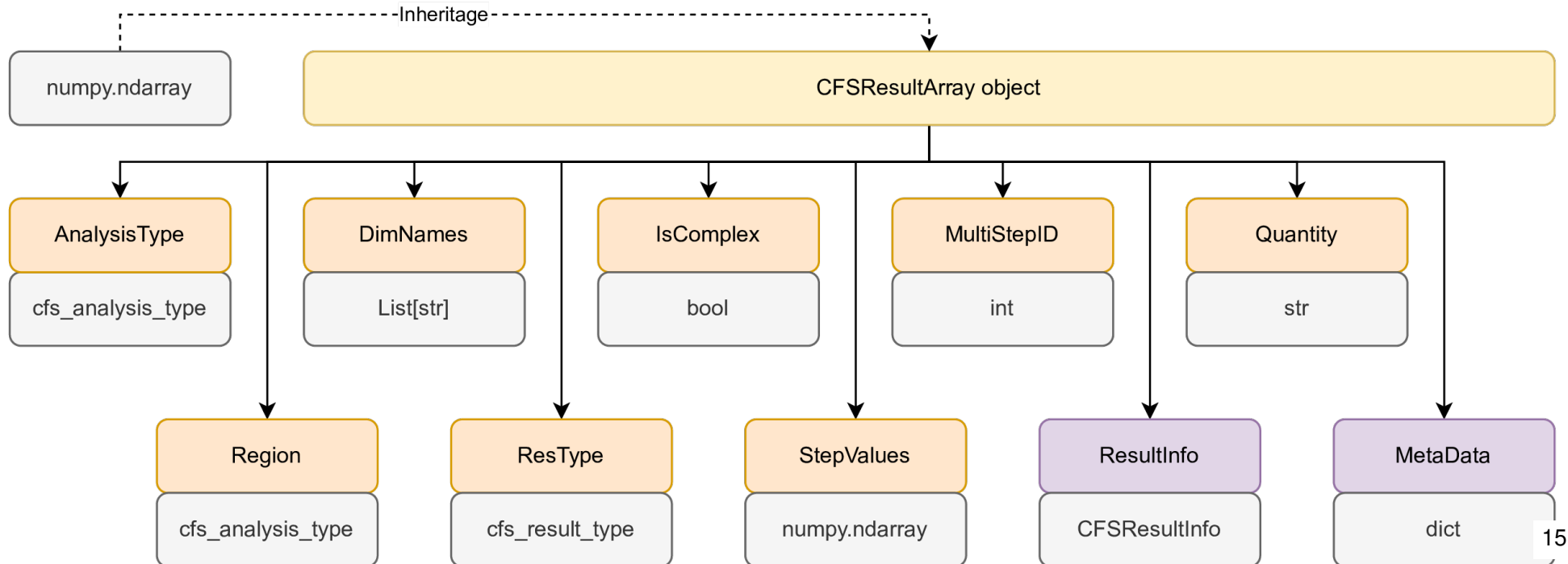
- Container object for all region related data



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

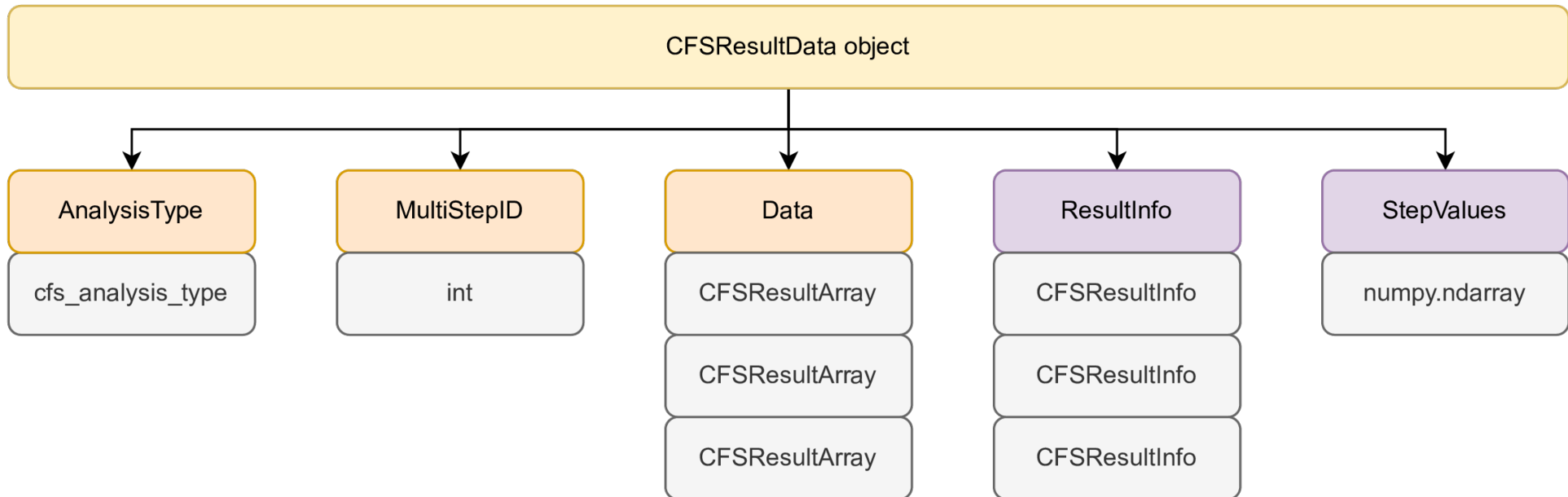
```
# Create a result array object
np_array = np.ones((5, 10, 3))
cfs_array = CFSResultArray(np_array)

# 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_util
```

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

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, sng)
```

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

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`)
 - *Planned:* `exodus_io` (Cubit mesh export)

EXAMPLE WORKFLOW

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
```

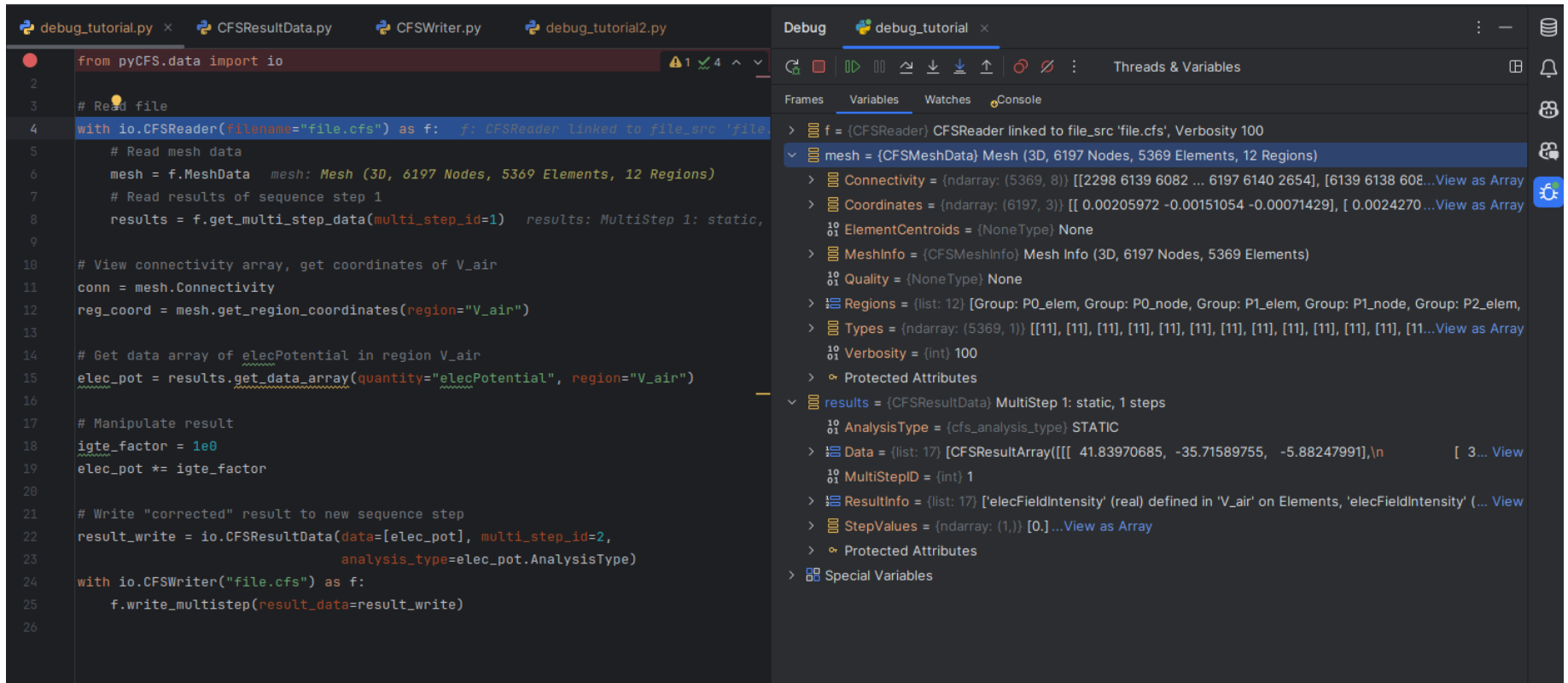

CODE

```
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 interface during a debugging session. The left pane shows the source code of `debug_tutorial.py`, and the right pane shows the debug console with the current state of the program.

Source Code (debug_tutorial.py):

```
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'
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,
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.CFSResultData(data=[elec_pot], multi_step_id=2,
23                                analysis_type=elec_pot.AnalysisType)
24 with io.CFSWriter("file.cfs") as f:
25     f.write_multistep(result_data=result_write)
26
```

Debug Console (debug_tutorial):

The debug console shows the current state of the program, with the following variables and values:

- `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]...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

DEBUGGING IN PYCHARM (2)

The screenshot displays the PyCharm IDE interface during a debugging session. The left pane shows the source code of `debug_tutorial.py`, and the right pane shows the debug console for the `debug_tutorial` process.

Source Code (debug_tutorial.py):

```
1 from pyCFS.data import io
2
3 # Read file
4 with io.CFSReader(filename="file.cfs") as f: f: Closed CFSReader
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,
9
10 # View connectivity array, get coordinates of V_air
11 conn = mesh.Connectivity conn: [[2298 6139 6082 ... 6197 6140 2654], [6139 6138 6088 ... 6196 ...View as Array
12 reg_coord = mesh.get_region_coordinates(region="V_air") reg_coord: [[ 0.00242705 -0.00176336 -0.00071429], [ 0.00212132 -0.01...View as Array
13
14 # Get data array of elecPotential in region V_air
15 elec_pot = results.get_data_array(quantity="elecPotential", region="V_air") elec_pot: CFSResultArray(1, 5849, 1) CFSResultArray([[[ 4.01669728], [ 4.02175741], ...View as Array
16
17 # Manipulate result
18 igte_factor = 1e0 igte_factor: 1.0
19 elec_pot *= igte_factor
20
21 # Write "corrected" result to new sequence step
22 result_write = io.CFSResultData(data=[elec_pot], multi_step_id=2,
23                                analysis_type=elec_pot.AnalysisType)
24 with io.CFSWriter("file.cfs") as f:
25     f.write_multistep(result_data=result_write)
26
```

Debug Console (debug_tutorial):

The debug console shows the state of variables during execution. The current frame is `debug_tutorial`.

- `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], [4.02175741], ...View as Array`
- `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.NdArra`
- `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 -0.01...View as Array`
- `min`: `{float64: ()} -0.01`
- `max`: `{float64: ()} 0.01`
- `shape`: `{tuple: 2} (5849, 3)`

EXAMPLE WORKFLOW

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 with io.CFSReader(filename="file.cfs") as h5r:
7     print(h5r)
8     mesh = h5r.MeshData
9     results = h5r.MultiStepData
10
11 # Perform interpolation
12 results_interpolated = interpolators.interpolate_node_to_cell(
13     mesh_data=mesh,
14     result_data=results,
15     regions=["V_air"],
16     quantity_names={"elecPotential": "interpolated_elecPotential"},
17 )
18
19 # Add interpolated result to results container
20 results.combine_with(results_interpolated)
21
22 # Check results container
```

CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3 from pyCFS.data.operators import interpolators
4
5 # Read source file
6 with io.CFSReader(filename="file.cfs") as h5r:
7     print(h5r)
8     mesh = h5r.MeshData
9     results = h5r.MultiStepData
10
11 # Perform interpolation
12 results_interpolated = interpolators.interpolate_node_to_cell(
13     mesh_data=mesh,
14     result_data=results,
15     regions=["V_air"],
16     quantity_names={"elecPotential": "interpolated_elecPotential"},
17 )
18
19 # Add interpolated result to results container
20 results.combine_with(results_interpolated)
21
22 # Check results container
```


CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3 from pyCFS.data.operators import interpolators
4
5 # Read source file
6 with io.CFSReader(filename="file.cfs") as h5r:
7     print(h5r)
8     mesh = h5r.MeshData
9     results = h5r.MultiStepData
10
11 # Perform interpolation
12 results_interpolated = interpolators.interpolate_node_to_cell(
13     mesh_data=mesh,
14     result_data=results,
15     regions=["V_air"],
16     quantity_names={"elecPotential": "interpolated_elecPotential"},
17 )
18
19 # Add interpolated result to results container
20 results.combine_with(results_interpolated)
21
22 # Check results container
```

CODE

```
1 # Import necessary modules
2 from pyCFS.data import io
3 from pyCFS.data.operators import interpolators
4
5 # Read source file
6 with io.CFSReader(filename="file.cfs") as h5r:
7     print(h5r)
8     mesh = h5r.MeshData
9     results = h5r.MultiStepData
10
11 # Perform interpolation
12 results_interpolated = interpolators.interpolate_node_to_cell(
13     mesh_data=mesh,
14     result_data=results,
15     regions=["V_air"],
16     quantity_names={"elecPotential": "interpolated_elecPotential"},
17 )
18
19 # Add interpolated result to results container
20 results.combine_with(results_interpolated)
21
22 # Check results container
```

INTERACTIVE MODE IN VS CODE (1)

```
debug_tutorial2.py
debug > debug_tutorial2.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  with io.CFSReader(filename="file.cfs") as h5r:
9      print(h5r)
10     mesh = h5r.MeshData
11     results = h5r.MultiStepData
12
Run Cell | Run Above | Debug Cell
13  #%%
14  # Perform interpolation
15  results_interpolated = interpolators.interpolate_node_to_cell(
16      mesh_data=mesh,
17      result_data=results,
18      regions=["V_air"],
19      quantity_names={"elecPotential": "interpolated_elecPotential"},
20  )
21
Run Cell | Run Above | Debug Cell
22  #%%
23  # Add interpolated result to results container
24  results.combine_with(results_interpolated)
25
26  # Check results container
27  print(results)
28
Run Cell | Run Above | Debug Cell
29  #%%
30  # Write output file
31  with io.CFSWriter("file_out.cfs") as h5w:
32      # Write mesh and results to new file
33      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 ...

INTERACTIVE MODE IN VS CODE (2)

```
debug_tutorial2.py
debug > debug_tutorial2.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  with io.CFSReader(filename="file.cfs") as h5r:
10     print(h5r)
11     mesh = h5r.MeshData
12     results = h5r.MultiStepData
13
14  Run Cell | Run Above | Debug Cell
15  #%%
16  # Perform interpolation
17  results_interpolated = interpolators.interpolate_node_to_cell(
18     mesh_data=mesh,
19     result_data=results,
20     regions=["V_air"],
21     quantity_names={"elecPotential": "interpolated_elecPotential"},
22 )
23
24  Run Cell | Run Above | Debug Cell
25  #%%
26  # Add interpolated result to results container
27  results.combine_with(results_interpolated)
28
29  # Check results container
30  print(results)
31
32  Run Cell | Run Above | Debug Cell
33  #%%
34  # Write output file
35  with io.CFSWriter("file_out.cfs") as h5w:
36     # Write mesh and results to new file
37     h5w.create_file(mesh_data=mesh, result_data=results)
38
```

Interactive-1

Interrupt | X 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
...
✓ # Write output file ...
```

INTERACTIVE MODE IN VS CODE (3)

```
debug_tutorial2.py
debug > debug_tutorial2.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  with io.CFSReader(filename="file.cfs") as h5r:
9      print(h5r)
10     mesh = h5r.MeshData
11     results = h5r.MultiStepData
12
Run Cell | Run Above | Debug Cell
13  """
14  # Perform interpolation
15  results_interpolated = interpolators.interpolate_node_to_cell(
16      mesh_data=mesh,
17      result_data=results,
18      regions=["V_air"],
19      quantity_names={"elecPotential": "interpolated_elecPotential"},
20  )
21
Run Cell | Run Above | Debug Cell
22  """
23  # Add interpolated result to results container
24  results.combine_with(results_interpolated)
25
26  # Check results container
27  print(results)
28
Run Cell | Run Above | Debug Cell
29  """
30  # Write output file
31  with io.CFSWriter("file_out.cfs") as h5w:
32      # Write mesh and results to new file
33      h5w.create_file(mesh_data=mesh, result_data=results)
34
```

```
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
```