# Post.IBM Documentation

*Release 3.5*

## /ELSA/MU-10019/V3.5

Nov 21, 2022

# CONTENTS

Specific post-processing for immersed boundaries (IB).

These functions work with a solution tree “t”, a geometry tree “tb”, and/or a connectivity tree “tc”.

# CHAPTER ONE

# LIST OF FUNCTIONS

**– Post-processing of IBs**

| | |
|---|---|
| *Post.IBM.extractIBMWallFields*(tc[, tb, ...]) | Extract the value of the flow field at the IBM target points onto the surface. |
| *Post.IBM.extractIBMInfo*(tc_in[, filename_out]) | Extracts the geometrical information required for the IBM (i e wall points, target points, and image points). |
| *Post.IBM.extractShearStress*(ts) | Compute the shear stress on the IB surface. |
| *Post.IBM.computeExtraVariables*(ts, PInf, QInf) | Computes additional variables required for IBM post-processing.. |
| *Post.IBM.extractPressureHO*(tc) | 1st order extrapolation of the pressure at the immersed boundary (IB). |
| *Post.IBM.extractPressureHO2*(tc) | 2nd order extrapolation of the pressure at the immersed boundary (IB). |
| *Post.IBM.extractConvectiveTerms*(tc) | Computes the convective terms required for the thin boundary layers equations (TBLE) and stores them in the tc. |

CHAPTER

# TWO

# CONTENTS

Post.IBM.**extractIBMInfo**(*tc*, *filename='IBMInfo.cgns'*)

Extracts the geometrical information required for the IBM (i.e. wall points, target points, and image points).

**Parameters** **tc** (*[zone, list of zones, base, tree]*) – connectivity tree

**Returns** tree with geometrical information required for the IBM

*Example of use:*

- Extract the IBM geometrical information (pyTree):

```
# - extractConvectiveTerms (pyTree) -
import Converter.Internal as Internal
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D
import KCore.test as test
import Post.IBM as P_IBM
import copy
import numpy

a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,11,12))
a = C.node2Center(a)
for z in Internal.getZones(a):
    Internal._createChild(z, 'IBCD_2_'+z[0] , 'ZoneSubRegion_t',
↪value=z[0])

Nlength   = numpy.zeros((10),numpy.float64)
for z in Internal.getZones(a):
    subRegions = Internal.getNodesFromType1(z, 'ZoneSubRegion_t')
    for zsr in subRegions:
        Internal._createChild(zsr, 'ZoneRole', 'DataArray_t',
↪value='Donor')
```

(continues on next page)

(continued from previous page)

```
        Internal._createChild(zsr, 'GridLocation', 'GridLocation_t
↪', value='CellCenter')

        zsr[2].append(['CoordinateX_PW', copy.copy(Nlength)+13, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateY_PW', copy.copy(Nlength)+13, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateZ_PW', copy.copy(Nlength)+13, [],
↪ 'DataArray_t'])

        zsr[2].append(['CoordinateX_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateY_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateZ_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])

        zsr[2].append(['CoordinateX_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateY_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateZ_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])

a=P_IBM.extractIBMInfo(a)
C.convertPyTree2File(a,'out.cgns')
```

---

`Post.IBM.`**`extractIBMWallFields`**(*tc*, *tb=None*, *coordRef='wall'*, *famZones=[]*, *front=1*)

Project the solution at IBM wall points onto the vertices of the surface. If tb is None, returns the cloud of points, else interpolate by Moving Least Squares interpolation on tb. Returns density, pressure, utau, yplus, velocity components.

**Parameters**

- **tc** (*[zone, list of zones, base, tree]*) – connectivity tree
- **tb** (*[zone, list of zones, base, tree]*) – surface mesh (TRI-type)
- **coordRef** (*string*) – coordinates of IBM points to be projected (default is IBM wall points) :'wall','cible','image'
- **famZones** (*list of names of families*) – list of families of IBCs to be projected

**Returns** surface tree with solution (density, pressure, friction velocity, yplus)

*Example of use:*

- Project the solution at IBM wall points onto the vertices of the surface (pyTree):

---

`Post.IBM.`**`extractShearStress`**`(`*tb*`)`

Computes the shear stress using utau values at vertices of the surface mesh

**Parameters** **`tb`** (*`[zone, list of zones, base, tree]`*) – surface mesh (TRI-type) with density, velocity, utau variable

**Returns** surface tree with shear stress variables "ShearStressXX", "ShearStressYY","ShearStressZZ","ShearStressXY", "ShearStressXZ", "ShearStressYZ"

*Example of use:*

- Computes the shear stress using utau values at vertices of the surface mesh (pyTree):

---

`Post.IBM.`**`computeExtraVariables`**`(`*tb, PInf, QInf, variables=['Cp','Cf','frictionX','frictionY','frictionZ', 'frictionMagnitude','ShearStress']*`)`

Computes variables using variables density, pressure, utau, and velocity at vertices of tb. Solution is located at cell centers.

**Parameters**

- **`tb`** (*`[zone, list of zones, base, tree]`*) – surface mesh (TRI-type) with density, velocity, utau variable
- **`PInf`** (*`real`*) – reference pressure to compute Cp
- **`QInf`** (*`real`*) – reference dynamic pressure
- **`variables`** (*`list of strings`*) – list of variables to be computed.

**Returns** surface tree with additional variables.

*Example of use:*

- Computes variables using variables density, pressure, utau, and velocity at vertices of tb (pyTree):

```
#compute shear stress for IBM
import Post.IBM as P_IBM
import Converter.PyTree as C
import Converter.Internal as Internal
import Geom.PyTree as D
import Generator.PyTree as G
```

(continues on next page)

(continued from previous page)

```
a=D.sphere((0,0,0),0.5,N=30)
a = C.convertArray2Tetra(a); a = G.close(a)
C._initVars(a,'{centers:utau}={centers:CoordinateX}**2')
C._initVars(a,'{centers:VelocityX}={centers:CoordinateZ}*
↪{centers:CoordinateY}')
C._initVars(a,'{centers:VelocityY}={centers:CoordinateX}*
↪{centers:CoordinateZ}')
C._initVars(a,'{centers:VelocityZ}={centers:CoordinateX}*
↪{centers:CoordinateY}')
C._initVars(a,'{centers:Density}=1')
C._initVars(a,'{centers:Pressure}=0.71')

P_IBM._computeExtraVariables(a,PInf=0.71, QInf=0.005, variables=[
↪'Cp','Cf','ShearStress'])
C.convertPyTree2File(a,"out.cgns")
```

Post.IBM.**extractConvectiveTerms**(*tc*)

Computes the convective terms required for the thin boundary layers equations (TBLE) and stores them in the tc.

**Parameters** **tc** (*[zone, list of zones, base, tree]*) – connectivity tree

**Returns** same as input

*Example of use:*

- Compute the convective terms (pyTree):

```
# - extractConvectiveTerms (pyTree) -
import Converter.Internal as Internal
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D
import KCore.test as test
import Post.IBM as P_IBM
import copy
import numpy

a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,11,12))
a = C.node2Center(a)
for z in Internal.getZones(a):
    Internal._createChild(z, 'IBCD_2_'+z[0] , 'ZoneSubRegion_t',␣
↪value=z[0])
```

(continues on next page)

(continued from previous page)

```
Nlength  = numpy.zeros((10),numpy.float64)
for z in Internal.getZones(a):
    subRegions = Internal.getNodesFromType1(z, 'ZoneSubRegion_t')
    for zsr in subRegions:
        Internal._createChild(zsr, 'ZoneRole', 'DataArray_t',␣
↪value='Donor')
        Internal._createChild(zsr, 'GridLocation', 'GridLocation_t
↪', value='CellCenter')

        zsr[2].append(['CoordinateX_PW', copy.copy(Nlength), [],
↪'DataArray_t'])
        zsr[2].append(['CoordinateY_PW', copy.copy(Nlength), [],
↪'DataArray_t'])
        zsr[2].append(['CoordinateZ_PW', copy.copy(Nlength), [],
↪'DataArray_t'])

        zsr[2].append(['CoordinateX_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateY_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateZ_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])

        zsr[2].append(['CoordinateX_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateY_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateZ_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])

        zsr[2].append(['Pressure', Nlength+3, [], 'DataArray_t'])
        zsr[2].append(['Density' , copy.copy(Nlength)+1, [],
↪'DataArray_t'])

        zsr[2].append(['gradxPressure', copy.copy(Nlength)+2, [],
↪'DataArray_t'])
        zsr[2].append(['gradyPressure', copy.copy(Nlength)+2, [],
↪'DataArray_t'])
        zsr[2].append(['gradzPressure', copy.copy(Nlength)+2, [],
↪'DataArray_t'])

        zsr[2].append(['gradxVelocityX', copy.copy(Nlength)+3, [],
↪'DataArray_t'])
        zsr[2].append(['gradyVelocityX', copy.copy(Nlength)+3, [],
↪'DataArray_t'])
```

(continues on next page)

(continued from previous page)

```
        zsr[2].append(['gradzVelocityX', copy.copy(Nlength)+3, [],
↪'DataArray_t'])

        zsr[2].append(['gradxVelocityY', copy.copy(Nlength)+4, [],
↪'DataArray_t'])
        zsr[2].append(['gradyVelocityY', copy.copy(Nlength)+4, [],
↪'DataArray_t'])
        zsr[2].append(['gradzVelocityY', copy.copy(Nlength)+4, [],
↪'DataArray_t'])

        zsr[2].append(['gradxVelocityZ', copy.copy(Nlength)+5, [],
↪'DataArray_t'])
        zsr[2].append(['gradyVelocityZ', copy.copy(Nlength)+5, [],
↪'DataArray_t'])
        zsr[2].append(['gradzVelocityZ', copy.copy(Nlength)+5, [],
↪'DataArray_t'])

        zsr[2].append(['VelocityX', copy.copy(Nlength)+6, [],
↪'DataArray_t'])
        zsr[2].append(['VelocityY', copy.copy(Nlength)+7, [],
↪'DataArray_t'])
        zsr[2].append(['VelocityZ', copy.copy(Nlength)+8, [],
↪'DataArray_t'])

a=P_IBM.extractConvectiveTerms(a)
C.convertPyTree2File(a,'out.cgns')
```

Post.IBM.**extractPressureHO**(*tc*)

1st order extrapolation of the pressure at the IB.

**Parameters** **tc** (*[zone, list of zones, base, tree]*) – connectivity tree

**Returns** same as input

*Example of use:*

- 1st order extrapolation of the pressure at the IB (pyTree):

```
# - extractPressureHO (pyTree) -
import Converter.Internal as Internal
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D
import KCore.test as test
import Post.IBM as P_IBM
```

(continues on next page)

(continued from previous page)

```
import copy
import numpy

a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,11,12))
a = C.node2Center(a)
for z in Internal.getZones(a):
    Internal._createChild(z, 'IBCD_2_'+z[0] , 'ZoneSubRegion_t',␣
↪value=z[0])

Nlength  = numpy.zeros((10),numpy.float64)
for z in Internal.getZones(a):
    subRegions = Internal.getNodesFromType1(z, 'ZoneSubRegion_t')
    for zsr in subRegions:
        Internal._createChild(zsr, 'ZoneRole', 'DataArray_t',␣
↪value='Donor')
        Internal._createChild(zsr, 'GridLocation', 'GridLocation_t
↪', value='CellCenter')

        zsr[2].append(['Pressure', Nlength+3, [], 'DataArray_t'])
        zsr[2].append(['Density' , copy.copy(Nlength)+1, [],
↪'DataArray_t'])

        zsr[2].append(['CoordinateX_PW', copy.copy(Nlength), [],
↪'DataArray_t'])
        zsr[2].append(['CoordinateY_PW', copy.copy(Nlength), [],
↪'DataArray_t'])
        zsr[2].append(['CoordinateZ_PW', copy.copy(Nlength), [],
↪'DataArray_t'])

        zsr[2].append(['CoordinateX_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateY_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateZ_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])

        zsr[2].append(['CoordinateX_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateY_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateZ_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])

        zsr[2].append(['gradxPressure', copy.copy(Nlength)+2, [],
↪'DataArray_t'])
```

(continues on next page)

(continued from previous page)

```
        zsr[2].append(['gradyPressure', copy.copy(Nlength)+2, [],
↪'DataArray_t'])
        zsr[2].append(['gradzPressure', copy.copy(Nlength)+2, [],
↪'DataArray_t'])

a=P_IBM.extractPressureHO(a)
C.convertPyTree2File(a,'out.cgns')
```

Post.IBM.**extractPressureHO2**(*tc*)

2nd order extrapolation of the pressure at the IB.

**Parameters** **tc** (*[zone, list of zones, base, tree]*) – connectivity tree

**Returns** same as input

*Example of use:*

- 2nd order extrapolation of the pressure at the IB (pyTree):

```
# - extractPressureHO2 (pyTree) -
import Converter.Internal as Internal
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D
import KCore.test as test
import Post.IBM as P_IBM
import copy
import numpy

a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,11,12))
a = C.node2Center(a)
for z in Internal.getZones(a):
    Internal._createChild(z, 'IBCD_2_'+z[0] , 'ZoneSubRegion_t',␣
↪value=z[0])

Nlength  = numpy.zeros((10),numpy.float64)
for z in Internal.getZones(a):
    subRegions = Internal.getNodesFromType1(z, 'ZoneSubRegion_t')
    for zsr in subRegions:
        Internal._createChild(zsr, 'ZoneRole', 'DataArray_t',␣
↪value='Donor')
        Internal._createChild(zsr, 'GridLocation', 'GridLocation_t
↪', value='CellCenter')

        zsr[2].append(['Pressure', Nlength+3, [], 'DataArray_t'])
```

(continues on next page)

(continued from previous page)

```
        zsr[2].append(['Density' , copy.copy(Nlength)+1, [],
↪'DataArray_t'])

        zsr[2].append(['CoordinateX_PW', copy.copy(Nlength), [],
↪'DataArray_t'])
        zsr[2].append(['CoordinateY_PW', copy.copy(Nlength), [],
↪'DataArray_t'])
        zsr[2].append(['CoordinateZ_PW', copy.copy(Nlength), [],
↪'DataArray_t'])

        zsr[2].append(['CoordinateX_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateY_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateZ_PC', copy.copy(Nlength)+14, [],
↪ 'DataArray_t'])

        zsr[2].append(['CoordinateX_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateY_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])
        zsr[2].append(['CoordinateZ_PI', copy.copy(Nlength)+15, [],
↪ 'DataArray_t'])

        zsr[2].append(['gradxPressure', copy.copy(Nlength)+2, [],
↪'DataArray_t'])
        zsr[2].append(['gradyPressure', copy.copy(Nlength)+2, [],
↪'DataArray_t'])
        zsr[2].append(['gradzPressure', copy.copy(Nlength)+2, [],
↪'DataArray_t'])

        zsr[2].append(['gradxxPressure', copy.copy(Nlength)+3, [],
↪'DataArray_t'])
        zsr[2].append(['gradxyPressure', copy.copy(Nlength)+3, [],
↪'DataArray_t'])
        zsr[2].append(['gradxzPressure', copy.copy(Nlength)+3, [],
↪'DataArray_t'])

        zsr[2].append(['gradyxPressure', copy.copy(Nlength)+4, [],
↪'DataArray_t'])
        zsr[2].append(['gradyyPressure', copy.copy(Nlength)+4, [],
↪'DataArray_t'])
        zsr[2].append(['gradyzPressure', copy.copy(Nlength)+4, [],
↪'DataArray_t'])
```

(continues on next page)

(continued from previous page)

```
        zsr[2].append(['gradzxPressure', copy.copy(Nlength)+5, [],
↪'DataArray_t'])
        zsr[2].append(['gradzyPressure', copy.copy(Nlength)+5, [],
↪'DataArray_t'])
        zsr[2].append(['gradzzPressure', copy.copy(Nlength)+5, [],
↪'DataArray_t'])

a=P_IBM.extractPressureHO2(a)
C.convertPyTree2File(a,'out.cgns')
```