Post.IBM Documentation

Release 3.4

/ELSA/MU-10019/V3.4

Jun 29, 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.extractConvectiveTerms(tc)	Computes the convective terms required for the thin boundary layers equations (TBLE) and stores them in the tc.
Post.IBM.extractIBMInfo(tc_in[, filename_out])	Extracts the geometrical information required for the IBM (i.e.
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.loads(t_case[, tc_in, tc2_in, ...])	Computes the viscous and pressure forces on the IB.

CHAPTER

TWO

CONTENTS

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])

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), [],
↪'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'])
        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'])
```

(continues on next page)

(continued from previous page)

```
        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.**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):
```

(continues on next page)

(continued from previous page)

```
    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(['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.**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
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'])
```

(continues on next page)

(continued from previous page)

```
        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'])

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')
```

(continues on next page)

(continued from previous page)

```
        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'])
        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'])
```

(continues on next page)

(continued from previous page)

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

        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')
```

Post.IBM.**loads**(*t_case*, *tc_in=None*, *tc2_in=None*, *wall_out=None*, *alpha=0.*, *beta=0.*, *gradP=False*, *order=1*, *Sref=None*, *famZones=[]*)

Computes the viscous and pressure forces on the IB. If tc_in=None, t_case must also contain the projection of the flow field solution onto the IB.

Parameters

- **t_case** (*[zone, list of zones, base, tree]*) – geometry tree
- **tc_in** (*[zone, list of zones, base, tree, or None]*) – connectivity tree
- **tc2_in** (*[zone, list of zones, base, tree, or None]*) – connectivity tree of second image point (if present)
- **wall_out** (*string or None*) – file name for the output of the forces at the wall and at the cell centers
- **alpha** (*float*) – Angle with respect to (0,Z) axe (in degrees)
- **beta** (*float*) – Angle with respect to (0,Y) axe (in degrees)
- **gradP** (*boolean*) – calculate the pressure gradient?
- **order** (*integer*) – pressure extrapolation order
- **Sref** (*float or None*) – reference surface area
- **famZones** (*list of strings or None*) – name of families of immersed boundaries on whih loads are computed

Returns tree with the solution at the IB and the viscous and pressure loads

Example of use:

- Computes the viscous and pressure forces on an IB (pyTree):

Post.IBM.**extractMassFlowThroughSurface**(*tb*, *t*, *famZones=[]*)

Returns massflow through a surface defined by tb and returns tb. If famZones is a list of families, then only the zones of tb where the Currently: only sequential mode!

Parameters

- **tb** (*[zone, list of zones, base, tree]*) – geometry tree
- **t** (*pyTree*) – solution tree with (Density,VelocityX, VelocityY, VelocityZ) stored at cell centers.
- **famZones** (*list of strings*) – list of names of families of zones of tb where the massflow must be computed.

Example of use:

- Computes the massflow through an inlet surface (pyTree):

```
import Converter.PyTree as C
import Post.IBM as P_IBM
import Geom.IBM as D_IBM
import Geom.PyTree as D
import Generator.PyTree as G
import Transform.PyTree as T
import Converter.Internal as Internal

a = D.cylinder((0.,0.,0.),1., 5, N=20,ntype='TRI')
a = T.splitSharpEdges(a)
tb = C.newPyTree(['Body'])
C._addFamily2Base(tb[2][1],'inlet')

for z in a:
    if C.getMaxValue(z,'CoordinateZ')==0:
        Internal.newFamilyName(name='FamilyName', value='inlet',␣
↪parent=z)
tb[2][1][2]=a

h = 0.1
a = G.cart((-1.5,-1.5,-1.5),(h,h,h),(int(3/h)+1,int(3/h)+1,int(6.5/
↪h)+1))
C._initVars(a,'centers:Density',1.)
C._initVars(a,'centers:VelocityX',0.)
C._initVars(a,'centers:VelocityY',0.)
C._initVars(a,'centers:VelocityZ',0.2)
```

(continues on next page)

(continued from previous page)

```
massflow, tmf = P_IBM.extractMassFlowThroughSurface(tb, a,␣
↪famZones=['inlet'])
print(massflow)
C.convertPyTree2File(tmf,"out.cgns")
```