



Post.Rotor Documentation

Release 3.5

/ELSA/MU-10019/V3.5

Nov 21, 2022

CONTENTS

- 1 List of functions** **3**
- 2 Contents** **5**
 - 2.1 Force extractions 5
 - 2.2 Accumulator export 9

Specific post-processing for rotors or propellers.

Those functions work with a solution tree “t” or a blade surface tree “teff”.

LIST OF FUNCTIONS

– Force extractions

<i>Post.Rotor.extractSlices</i> (teff, bladeName, ...)	Extract slices on blade and compute Kp,Cf,CnM2,CmM2.
<i>Post.Rotor.computeZb</i> (teff, psi, RoInf, ...)	Compute Zb.
<i>Post.Rotor.computeThrustAndTorque</i> (teff, psi, ...)	Compute thrust.

– Accumulator export

<i>Post.Rotor.exportAccumulatorPerPsi</i> (accumulator) given psi.	Export accumulator (psi,rad) in a zone for a given psi.
<i>Post.Rotor.exportAccumulatorPerRadius</i> (...[, ...])	Export accumulator (psi,rad) in a zone for a given radius.
<i>Post.Rotor.exportAccumulatorMap</i> (accumulator)	Export accumulator (psi,rad) in a map zone.

CONTENTS

2.1 Force extractions

`Post.Rotor.extractSlices` (*teff*, *bladeName*, *psi*, *radius*, *RoInf*, *PInf*, *ASOUND*, *Mtip*, *AR*, *CHORD*, *MU*, *relativeShaft=0.*, *localFrame=True*, *delta=0.05*, *accumulatorSlices=None*, *accumulatorCnM2=None*, *accumulatorCmM2=None*)

Extract slices on blades. Export Cp, Cf on those slices. Compute CnM2 and CmM2 on those slices.

Parameters

- **teff** (*[zone, list of zones, base, tree]*) – surface stress tree
- **bladeName** (*string*) – name of the blade base to work with
- **psi** (*float*) – angular angle of blade in teff (in degree)
- **radius** (*list of floats*) – list of radius to extract
- **RoInf** (*float*) – infinite flow density
- **PInf** (*float*) – infinite flow pressure
- **ASOUND** (*float*) – infinite flow sound speed
- **Mtip** (*float*) – blade mach tip
- **AR** (*float*) – blade length
- **CHORD** (*float*) – blade mean chord
- **MU** (*float*) – advance ratio
- **relativeShaft** (*float*) – relative shaft angle if the mesh is not in the wind frame

- **localFrame** (*boolean*) – if True, return CnM2 and CmM2 in relative (blade section) frame
- **delta** (*float*) – mean mesh step on blade in the span wise direction
- **accumulatorSlices** (*dictionary*) – if not None, accumulate slices
- **accumulatorCnM2** (*dictionary*) – if not None, accumulate CnM2
- **accumulatorCmM2** (*dictionary*) – if not None, accumulate CmM2

Returns list of slices, list of CnM2, list of CmM2 (one for each radius)

Return type list of zones, list of 3 floats, list of 3 floats

Example of use:

- Extract slices (pyTree):

```
# - extractSlices (pyTree) -
import Converter.PyTree as C
import Post.Rotor as PR
import math

Mtip = 0.6462; MU = 0.4
CHORD = 0.14; AR = 2.1
RoInf = 1.225; PInf = 101325.; AINF = 340.1
SIGMA = 4*CHORD / (math.pi*AR)
teff = C.convertFile2PyTree('stress_419.cgns')

accu = {}
psi = 419.; radius = [1.2,1.3,2.]
slices, CnM2, CmM2 = PR.extractSlices(teff, 'Blade7A_00', psi,
    ↪radius,
    ↪RoInf, PInf, AINF, Mtip, AR,
    ↪CHORD, MU,
    ↪accumulatorCnM2=accu,
    ↪localFrame=True,
    ↪relativeShaft=-12.12)

# export for Cp, Cf
C.convertPyTree2File(slices, 'slice.cgns')

# export for CnM2 maps
exp = PR.exportAccumulatorMap(accu, vars=['CnM2x', 'CnM2y', 'CnM2z'])
C.convertPyTree2File(exp, 'map.cgns')
```

`Post.Rotor.computeZb` (*teff*, *psi*, *RoInf*, *ASOUND*, *Mtip*, *AR*, *SIGMA*, *relativeShaft=0.*, *accumulatorZb=None*)

Compute Zb in the wind frame.

Parameters

- **teff** (*[zone, list of zones, base, tree]*) – surface stress tree
- **psi** (*float*) – angular angle of blade in teff (in degree)
- **RoInf** (*float*) – infinite flow density
- **ASOUND** (*float*) – infinite flow sound speed
- **Mtip** (*float*) – blade mach tip
- **AR** (*float*) – blade length in m
- **SIGMA** (*float*) – rotor solidity (= Nb*c / pi*AR)
- **relativeShaft** (*float*) – relative shaft angle if the mesh is not in the wind frame
- **accumulatorZb** (*dictionary*) – if not None, accumulate Zb

Returns [Xb,Yb,Zb]

Return type list of 3 floats

Example of use:

- Compute Zb (pyTree):

```
# - computeZb (pyTree) -
import Converter.PyTree as C
import Post.Rotor as PR
import math

Mtip = 0.6462; MU = 0.4
CHORD = 0.14; AR = 2.1
RoInf = 1.225; PInf = 101325.; AINF = 340.1
SIGMA = 4*CHORD / (math.pi*AR)
teff = C.convertFile2PyTree('stress_419.cgns')

zb = PR.computeZb(teff, 419., RoInf, AINF, Mtip, AR, SIGMA,
↳relativeShaft=-12.12); print(zb)
#> [-1.101453126880057, -0.11259440192933268, 10.18004327358801]

accu = {}
zb = PR.computeZb(teff, 419., RoInf, AINF, Mtip, AR, SIGMA,
↳relativeShaft=-12.12, accumulatorZb=accu)
```

(continues on next page)

(continued from previous page)

```
ret = PR.exportAccumulatorPerRadius(accum, vars=['Xb', 'Yb', 'Zb'])
C.convertPyTree2File(ret, 'Zb.cgns')
```

`Post.Rotor.computeThrustAndTorque` (*teff, psi, PInf, center=(0, 0, 0), relativeShaft=0., accumulatorThrust=None*)
 Compute Thrust in the rotor frame (that is orthogonal to rotor).

Parameters

- **teff** (*[zone, list of zones, base, tree]*) – surface stress tree
- **psi** (*float*) – angular angle of blade in teff (in degree)
- **PInf** (*float*) – infinite flow pressure
- **center** (*list of 3 floats*) – center for momentum computations
- **relativeShaft** (*float*) – relative shaft angle if the mesh is not in the rotor frame
- **accumulatorThrust** (*dictionary*) – if not None, accumulate thrust and torque

Returns thrust=[tx,ty,tz] and torque=[mx,my,mz]

Return type 2 lists of 3 floats

Example of use:

- Compute thrust and torque (pyTree):

```
# - computeThrustAndTorque (pyTree) -
import Converter.PyTree as C
import Post.Rotor as PR
import math

Mtip = 0.6462; MU = 0.4
CHORD = 0.14; AR = 2.1
RoInf = 1.225; PInf = 101325.; AINF = 340.1
SIGMA = 4*CHORD / (math.pi*AR)
teff = C.convertFile2PyTree('stress_419.cgns')

thrust,torque = PR.computeThrustAndTorque(teff, 419., PInf,
↳center=(0,0,0), relativeShaft=0.); print('thrust', thrust,
↳'torque',torque)
#> thrust [368.952736931205, -39.172151751517326, 3543.
↳2002154791667] torque [226.5518638611441, 950.9479780913017, -
↳935.2791149345967]
```

(continues on next page)

(continued from previous page)

```

accu = {}
thrust, torque = PR.computeThrustAndTorque(teff, 419., PInf,
    ↪center=(0,0,0), relativeShaft=0., accumulatorThrust=accu)
ret = PR.exportAccumulatorPerRadius(accu, vars=['ThrustX', 'ThrustY',
    ↪', 'ThrustZ', 'TorqueX', 'TorqueY', 'TorqueZ'])
C.convertPyTree2File(ret, 'Thrust.cgns')

```

2.2 Accumulator export

Post.Rotor.**exportAccumulatorPerPsi** (*accumulator*, *psi=0.*,
vars=['F1','F2'])

Export a given psi of an accumulator (psi,rad) in a 1D zone. For distributed computations, the exported zone is identical on all processors.

Parameters

- **accumulator** (*dictionary*) – (psi,rad) accumulator
- **psi** (*float*) – angular angle to be extracted (in degree)
- **vars** (*list of strings*) – the name of variables stored in accumulator

Returns a single Zone with vars corresponding to psi

Return type Zone

Example of use:

- Export accumulator for given psi (pyTree):

```

# - exportAccumulatorPerPsi (pyTree) -
import Post.Rotor as PR
import Converter.PyTree as C

accu = {}
for psi in range(0, 360, 10):
    for rad in range(0, 10):
        accu[(psi,rad)] = [psi+rad*0.1]

z = PR.exportAccumulatorPerPsi(accu, psi=10., vars=['F'])
C.convertPyTree2File(z, 'out.cgns')

```

`Post.Rotor.exportAccumulatorPerRadius` (*accumulator*, *rad=0.*,
vars=['F1','F2'])

Export a given radius of an accumulator (psi,rad) in a 1D zone. For distributed computations, the exported zone is identical on all processors.

Parameters

- **accumulator** (*dictionary*) – (psi,rad) accumulator
- **rad** (*float*) – radius to be extracted
- **vars** (*list of strings*) – the name of variables stored in accumulator

Returns a single Zone with vars corresponding to rad

Return type Zone

Example of use:

- Export accumulator for given rad (pyTree):

```
# - exportAccumulatorPerRadius (pyTree) -
import Post.Rotor as PR
import Converter.PyTree as C

accu = {}
for psi in range(0, 360, 10):
    for rad in range(0, 10):
        accu[(psi,rad)] = [psi+rad*0.1]

z = PR.exportAccumulatorPerRadius(accu, rad=4, vars=['F'])
C.convertPyTree2File(z, 'out.cgns')
```

`Post.Rotor.exportAccumulatorMap` (*accumulator*, *vars=['Fx','Fy','Fz']*)

Export accumulator (psi,rad) to a 2D zone. For distributed computations, the exported zone is identical on all processors.

Parameters

- **accumulator** (*dictionary*) – (psi,rad) accumulator
- **vars** (*list of strings*) – the name of variables stored in accumulator

Returns a single Zone with fields

Return type Zone

Example of use:

- Export accumulator to a map (pyTree):

```
# - exportAccumulatorMap (pyTree) -
import Post.Rotor as PR
import Converter.PyTree as C

accu = {}
for psi in range(0, 360, 10):
    for rad in range(0, 10):
        accu[(psi,rad)] = [psi+rad*0.1]

z = PR.exportAccumulatorMap(accu, vars=['F'])
C.convertPyTree2File(z, 'out.cgns')
```