Post 2.4

Stephanie Peron, Christophe Benoit, Gaelle Jeanfaivre, Pascal Raud
- Onera -

1 Post: solution post-processing module

1.1 Preamble

This module provides post-processing tools for CFD simulations. It manipulates arrays (as defined in Converter documentation) or CGNS/Python trees (pyTrees) as data structures.

This module is part of Cassiopee, a free open-source pre- and post-processor for CFD simulations.

When using the Converter array interface, a (or b) denotes an array, and A (or B) denotes a list of arrays. Then, Post module must be imported:

```
import Post as P
```

When using the pyTree interface, import the module:

```
import Post.PyTree as P
```

In that case, a is a zone node. A is a list of zone nodes or a complete pyTree.

1.2 Changing variable names

P.renameVars: Rename a list of variables with new variable names:

```
t = P.renameVars(t, oldVarNameList, newVarNameList)
```

(See: renameVars.py) (See: renameVarsPT.py)

1.3 Variables import

P.importVariables: variables located at nodes and/or centers can be imported from a pyTree t1 to a pyTree t2. If one variable already exists in t2, it is replaced by the same variable from t1. If method=0, zone are matched from names, if method=1, zones are matched from coordinates with a tolerance eps, if method=2, zones are taken in the given order of t1 and t2 (must match one by one). If addExtra=1, unmatched zones are added to a EXTRA base:

```
t = P.importVariables(t1, t2, method=0, eps=1.e-6, addExtra=1)
```

(See: importVariablesPT.py)

1.4 Variables computation

P.computeVariables: new variables can be computed from conservative variables. The list of the names of the variables to compute must be provided.
The computation of some variables (e.g. viscosity) require some constants as input data. In the pyTree version, if a reference state node is defined in the pyTree, then the corresponding reference constants are used. Otherwise, they must be specified as an argument of the function. These constants are:
- 'gamma' for the specific heat ratio,
- 'rgp' for the perfect gas constant (rgp = (gamma-1) * cv),
- 'betas' and 'Cs' (Sutherland's law constants), or 'Cs','Ts' and 'mus'
- 's0' for a constant entropy, defined by:

```
s0 = sref - rgp gamma/(gamma-1) ln(Tref) + rgp ln(Pref),
where sref, Tref and Pref are defined for a reference state.
```

Computed variables are defined by their CGNS names:

```
- 'VelocityX', 'VelocityY', 'VelocityZ' for components of the absolute velocity,
- 'VelocityMagnitude' for the absolute velocity magnitude,
- 'Pressure' for the static pressure (requires: gamma),
- 'Temperature' for the static temperature (requires: gamma, rgp),
- 'Enthalpy' for the enthalpy (requires: gamma),
- 'Entropy' for the entropy (requires: gamma, rgp, s0),
- 'Mach' for the Mach number (requires: gamma),
- 'ViscosityMolecular' for the fluid molecular viscosity (requires: gamma, rgp, Ts, mus, Cs),
- 'PressureStagnation' for stagnation pressure(requires: gamma),
- 'TemperatureStagnation' for stagnation temperature (requires: gamma, rgp),
- 'PressureDynamic' for dynamic pressure (requires: gamma).
```

```
b = P.computeVariables(a, ['varname1', 'varname2'], gamma=1.4, rgp=287.053, s0=0., betas=1.458e-6, Cs=110.4, mus=1.76e-5, Ts=273.15) .or. B = P.computeVariables(A, ...)
```

In the pyTree version, if the variable name is prefixed by 'centers:' then the variable is computed at centers only (e.g. 'centers:Pressure'), and if it is not prefixed, then the variable is computed at nodes.
(See: computeVariables.py) (See: computeVariablesPT.py)

P.computeExtraVariable: compute more advanced variables from conservative variables. 'varName' can be 'Vorticity', 'VorticityMagnitude', 'QCriterion', 'ShearStress' (requires: gamma, rgp, Ts, mus, Cs), 'SkinFriction', 'SkinFrictionTangential': The computation of the shear stress requires gamma, rgp, Ts, mus, Cs as input data. In the pyTree version, if a reference state node is defined in the pyTree, then the corresponding reference constants are used. Otherwise, they must be specified as an argument of the function.

b = P.computeExtraVariable(a, 'varName', gamma=1.4, rgp=287.053, Cs=110.4, mus=1.76e-5, Ts=273.15) *.or.* B = P.computeExtraVariable(A, 'varName', gamma=1.4, rgp=287.053, Cs=110.4, mus=1.76e-5, Ts=273.15)

(See: computeExtraVariable.py) (See: computeExtraVariablePT.py)

P.computeWallShearStress: compute the shear stress at wall boundaries provided the velocity gradient is already computed. The problem dimension and the reference state must be provided in t, defining the skin mesh. The function is only available in the pyTree version.

t = P.computeWallShearStress(t) *.or.* P._computeWallShearStress(t) *for the in place version.*

(See: computeWallShearStress.py)

P.computeGrad: compute the gradient (gradx,grady,gradz) of a field of name varname defined in a. The returned field is located at cell centers:

b = P.computeGrad(a, varname) *.or.* B = P.computeGrad(A, varname)

(See: computeGrad.py) (See: computeGradPT.py)

P.computeGrad2: compute the gradient (gradx,grady,gradz) at cell centers for a field located at cell centers. Using Converter.array interface, a(A) denotes the mesh, ac(AC) denotes the fields

located at centers. indices is a numpy 1D-array of face list, BCField is the corresponding numpy array of face fields. They are used to force a value at some faces before computing the gradient.

b = P.computeGrad2(a, ac, indices=None, BCField=None) *.or.* B = P.computeGrad2(A, AC, indices=None, BCField=None)

(See: computeGrad2.py)

Using the pyTree version, the variable name must be located at cell centers. Indices and BCFields are automatically extracted from BCDataSet nodes: if a BCDataSet node is defined for a BC of the pyTree, the corresponding face fields are imposed when computing the gradient:

b = P.computeGrad2(a,varname) *.or.* B = P.computeGrad2(A, varname)

(See: computeGrad2.py)

P.computeNormGrad: compute the norm of gradient (gradx,grady,gradz) of a field of name varname defined in a. The returned field 'grad'+varname and is located at cell centers:

b = P.computeNormGrad(a, varname) *.or.* B = P.computeNormGrad(A, varname)

(See: computeNormGrad.py) (See: computeNormGradPT.py)

P.computeCurl: compute curl of a 3D vector defined by its variable names ['vectx','vecty','vectz'] in a. The returned field is defined at cell centers for structured grids and elements centers for unstructured grids:

```
b = P.computeCurl(a, ['vectx','vecty','vectz']) .or. B = P.computeCurl(A, ['vectx','vecty','vectz'] )
```
(See: computeCurl.py) (See: computeCurlPT.py)

P.computeNormCurl: compute the norm of the curl of a 3D vector defined by its variable names ['vectx','vecty','vectz'] in a:

```
b = P.computeNormCurl(a, ['vectx','vecty','vectz']) .or. B = P.computeNormCurl(A, ['vectx','vecty','vectz'] )
```
(See: computeNormCurl.py) (See: computeNormCurlPT.py)

P.computeDiff: compute the difference between neighbouring cells of a scalar field defined by its variable varname in a. The maximum of the absolute difference among all directions is kept:

```
b = P.computeDiff(a, varname) .or. B = P.computeDiff(A,varname) ¡/a¿
```
(See: computeDiff.py) (See: computeDiffPT.py)

1.5 Solution selection

P.selectCells: select cells with respect to a given criterion. If strict=0, the cell is selected if at least one of the cell vertices satisfies the criterion. If strict=1, the cell is selected if all the cell vertices satisfy the criterion. The criterion can be defined as a python function returning True (=selected) or False (=not selected):

```
b = P.selectCells(a, F, ['var1', 'var2'], strict=0) .or. B = P.selectCells(A, F, ['var1', 'var2'], strict=0)
```
or by a formula:

```
b = P.selectCells(a, 'x+y¿2', strict=0) .or. B = P.selectCells(A, 'x+y¿2', strict=0)
```
(See: selectCells.py) (See: selectCellsPT.py)

P.selectCells2: select cells according to a field defined by a variable 'tag' (=1 if selected, =0 if not selected). If 'tag' is located at centers, only cells of tag=1 are selected. If 'tag' is located at nodes and 'strict'=0, the cell is selected if at least one of the cell vertices is tag=1. If 'tag' is located at nodes and 'strict'=1, the cell is selected if all the cell vertices is tag=1. In the array version, the tag is an array. In the pyTree version, the tag must be defined in a 'FlowSolution_t' type node located at cell centers or nodes.

```
b = P.selectCells2(a, tag, strict=0) .or. B = P.selectCells2(A, TAG, strict=0)
```
(See: selectCells2.py) (See: selectCells2PT.py)

P.interiorFaces: select the interior faces of a mesh. Interior faces are faces with two neighbours. If 'strict' is set to 1, select the interior faces that have only interior nodes:

```
b = P.interiorFaces(a, strict=0)
```
(See: interiorFaces.py) (See: interiorFacesPT.py)

P.exteriorFaces: Select the exterior faces of a mesh, and return them in a single unstructured zone. If indices=[], the indices of the original exterior faces are stored.

```
b = P.exteriorFaces(a,indices=None)
```

(See: exteriorFaces.py) (See: exteriorFacesPT.py)

P.exteriorFacesStructured: Select the exterior faces of a structured mesh as a list of structured meshes:

```
b = P.exteriorFacesStructured(a)
```

(See: exteriorFacesStructured.py) (See: exteriorFacesStructuredPT.py)

P.exteriorElts: select the exterior elements of a mesh, that is the first border fringe of cells:

```
b = P.exteriorElts(a)
```

(See: exteriorElts.py) (See: exteriorEltsPT.py)

P.frontFaces: select faces that are located at the boundary where a tag indicator change from 0 to 1:

```
b = P.frontFaces(a, tag)
```

(See: frontFaces.py) (See: frontFacesPT.py)

P.sharpEdges: return sharp edges arrays starting from surfaces or contours. Adjacent cells having an angle deviating from more than alphaRef to 180 degrees are considered as sharp:

```
res = P.sharpEdges(A, alphaRef=30.)
```

(See: sharpEdges.py) (See: sharpEdgesPT.py)

P.silhouette: return silhouette arrays starting from surfaces or contours, according to a direction vector.

```
res = P.silhouette(A, vector=[1.,0.,0.])
```

(See: silhouette.py) (See: silhouettePT.py)

P.coarsen: coarsen a triangle mesh by providing a coarsening indicator, which is 1 if the element must be coarsened, 0 elsewhere. Triangles are merged by edge contraction, if tagged to be coarsened by indic and if new triangles deviate less than tol to the original triangle. Required mesh quality is controled by argqual: argqual equal to 0.5 corresponds to an equilateral triangle, whereas a value near zero corresponds to a bad triangle shape.
Array version: an indic i-array must be provided, whose dimension ni is equal to the number of elements in the initial triangulation:

```
b = P.coarsen(a, indic, argqual=0.1, tol=1.e6)
```

(See: coarsen.py)
PyTree version: indic is stored as a solution located at centers:

```
b = P.coarsen(a, indicName='indic', argqual=0.25, tol=1.e-6)
```

(See: coarsenPT.py)

P.refine: refine a triangle mesh by providing a refinement indicator, which is 1 if the element must be refined, 0 elsewhere. Array version: an indic i-array must be provided, whose dimension ni is equal to the number of elements in the initial triangulation:

```
b = P.refine(a, indic)
```

(See: refine.py)

PyTree version: indic is stored as a solution located at centers:

```
b = P.refine(a, indicName='indic')
```

(See: refinePT.py)

P.refine: refine a triangle mesh every where using butterfly interpolation with coefficient w:

```
b = P.refine(a, w=1./64.)
```

(See: refine2.py) (See: refine2PT.py)

P.computeIndicatorValue: compute the indicator value on the unstructured octree mesh a based on the absolute maximum value of a varName field defined in the corresponding structured octree t. In the array version, t is a list of zones, and in the pyTree version, it can be a tree or a base or a list of bases or a zone or a list of zones. Variable varName can be located at nodes or centers. The resulting projected field is stored at centers in the octree mesh:

```
b = P.computeIndicatorValue(a, t, varName)
```

(See: computeIndicatorValue.py) (See: computeIndicatorValuePT.py)

P.computeIndicatorField: compute an indicator field to adapt an octree mesh with respect to the required number of points nbTargetPts, a field, and bodies. If refineFinestLevel=1, the finest level of the octree o is refined. If coarsenCoarsestLevel=1, the coarsest level of the octree o is coarsened provided the balancing is respected.
This function computes epsInf, epsSup, indicator such that when indicVal ¡ valInf, the octree is coarsened (indicator=-1), when indicVal ¿ valSup, the octree is refined (indicator=1).
For an octree defined in an array o, and the field in indicVal:

```
indicator, valInf, valSup = P.computeIndicatorField(o, indicVal, nbTargetPts=-1, bodies=[], refineFinestLevel=1, coarsenCoarsestLevel=1)
```

For the pyTree version, the name varname of the field on which is based the indicator must be specified:

```
o, valInf, valSup = P.computeIndicatorField(o, varname, nbTargetPts=-1, bodies=[], refineFinestLevel=1, coarsenCoarsestLevel=1)
```

(See: computeIndicatorField.py) (See: computeIndicatorFieldPT.py)

1.6 Solution extraction

P.extractPoint: extract the field in one or several points, given a solution defined by A. The extracted field(s) is returned as a list of values for each point. If the point (x,y,z) is not interpolable from a grid, then 0 for all fields is returned.
In the pyTree version, extractPoint returns the extracted solution from solutions located at nodes followed by the solution extracted from solutions at centers.

If 'cellN', 'ichim', 'cellnf', 'status', or 'cellNF' variable is defined, it is returned in the last position in the output array. The interpolation order can be 2, 3, or 5.
constraint is a thresold for extrapolation to occur. To enable more extrapolation, rise this value.
If some blocks in A define surfaces, a tolerance 'tol' for interpolation cell search can be defined.
A hook can be defined in order to keep in memory the ADT on the interpolation cell search. It can be built and deleted by createHook and freeHook functions in Converter module, using 'extractMesh' function:

```
field = P.extractPoint(A, (x,y,z), order=2, constraint=40., tol=1.e-6, hook=None).or. F = P.extractPoint(A, [(x1,y1,z1),(x2,y2,z2)], order=2, constraint=40., tol=1.e-6, hook=None)
```

(See: extractPoint.py) (See: extractPointPT.py)

P.extractPlane: slice a solution A with a plane. The extracted solution is interpolated from A. Interpolation order can be 2, 3, or 5 (but the 5th order is very time-consuming for the moment). The best solution is kept. Plane is defined by **c1 x + c2 y + c3 z + c4 = 0**:

```
b = P.extractPlane(A, (c1, c2, c3, c4), order=2, tol=1.e-6)
```

(See: extractPlane.py) (See: extractPlanePT.py)

P.extractMesh: Interpolate a solution from a set of donor zones defined by A to an extraction zone a. Parameter order can be 2, 3 or 5, meaning that 2nd, 3rd and 5th order interpolations are performed.
Parameter constraint¿0 enables to extrapolate from A if interpolation is not possible for some points. Extrapolation order can be 0 or 1 and is defined by extrapOrder.
If mode='robust', extract from the node mesh (solution in centers is first put to nodes, resulting interpolated solution is located in nodes).
If mode='accurate', extract node solution from node mesh and center solution from center mesh (variables don't change location).
A preconditioning tree for the interpolation cell search can be built prior to extractMesh (if is used several times for instance) and is stored in a hook. It can be created and deleted by C.createHook and C.freeHook (see Converter module userguide):

```
b = P.extractMesh(A, a, order=2, extrapOrder=1, constraint=40., tol=1.e-6, mode='robust', hook=None)
```

(See: extractMesh.py) (See: extractMeshPT.py)

P.projectCloudSolution: Project the solution by a Least-Square Interpolation defined on a set of points pts defined as a 'NODE' zone to a body defined by a 'TRI' mesh in 3D and 'BAR' mesh in 2D.:

```
b = P.projectCloudSolution(pts, t, dim=3)
```

(See: projectCloudSolution.py) (See: projectCloudSolutionPT.py)

P.zipper: build an unstructured unique surface mesh, given a list of structured overlapping surface grids A. Cell nature field is used to find blanked (0) and interpolated (2) cells:

```
a = P.zipper(A, options=[])
```
The options argument is a list of arguments such as [”argName”, argValue]. Option names can be:
- ’overlapTol’ for tolerance required between two overlapping grids : if the projection distance between them is under this value then the grids are considered to be overset. Default value is 1.e-5.
- For some cases, ’matchTol’ can be set to modify the matching boundaries tolerance. Default value is set 1e-6. In most cases, one needn’t modify this parameter.
(See: zipper.py) (See: zipperPT.py)

P.usurp*: an alternative to ”zipper” is ”usurp”. Result is a ratio field located at cell centers. In case of no overset, ratio are set to 1, otherwise ratio represents the percentage of overlap of a cell by another mesh.

When using the array interface, the input arrays are a list of grid arrays A, defining nodes coordinates and a corresponding list of arrays defining the chimera nature of cells at cell centers B. Blanked cells must be flagged by a null value. Other values are equally considered as computed or interpolated cells:
```
C = P.usurp(A, B)
```
When using the pyTree interface, chimera cell nature field must be defined as a center field in A:
```
B = P.usurp(A)
```
Warning: normal of surfaces grids defined by A must be oriented in the same direction.
(See: usurp.py) (See: usurpPT.py)

1.7 Streams

P.streamLine: compute the stream line with N points starting from point (x0,y0,z0), given a solution A and a vector defined by 3 variables [’v1’,’v2,’v3’]. Parameter ’dir’ can be set to 1 (streamline follows velocity), -1 (streamline follows -velocity), or 2 (streamline expands in both directions). The output yields the set of N extracted points on the streamline, and the input fields at these points. The streamline computation stops when the current point is not interpolable from the input grids:
```
b = P.streamLine(A, (x0,y0,z0), [’v1’,’v2,’v3’], N=2000, dir=2)
```
(See: streamLine.py) (See: streamLinePT.py)

P.streamRibbon: compute the stream ribbon starting from point (x0,y0,z0), of width and direction given by the vector (nx,ny,nz). This vector must be roughly orthogonal to the vector [’v1’, ’v2’, ’v3’] at point (x0,y0,z0). The output yields the set of N extracted points on the stream ribbon, and the input fields at these points. The stream ribbon computation stops when the current point is not interpolable from the input grids:
```
b = P.streamRibbon(A, (x0,y0,z0), (nx,ny,nz), [’v1’, ’v2’, ’v3’], N=2000, dir=2)
```
(See: streamRibbon.py) (See: streamRibbonPT.py)

P.streamSurf: compute the stream surface starting from a BAR array c:

```
b = P.streamSurf(A, c, ['v1','v2,'v3'], N=2000, dir=1)
```
(See: streamSurf.py) (See: streamSurfPT.py)

1.8 Isos

P.isoLine: compute an isoline correponding to value val of field:

```
b = P.isoLine(A, field, val)
```
(See: isoLine.py) (See: isoLinePT.py)

P.isoSurf: compute an isosurface correponding to value val of field (using marching tetrahedra). Resulting solution is always located in nodes. Return a list of two zones (one TRI and one BAR, if relevant):

```
B = P.isoSurf(A, field, val)
```
(See: isoSurf.py) (See: isoSurfPT.py)

P.isoSurfMC: compute an isosurface correponding to value val of field (using marching cubes). Resulting solution is always located in nodes:

```
b = P.isoSurfMC(A, field, val)
```
(See: isoSurfMC.py) (See: isoSurfMCPT.py)

1.9 Solution integration

For all integration functions, the interface is different when using Converter arrays interface or pyTree interface. For arrays, fields must be input separately, for pyTree, they must be defined in each zone.

P.integ: compute the integral of a scalar field (whose name is varString) over the geometry defined by arrays containing the coordinates + field (+ an optional ratio). Solution and ratio can be located at nodes or at centers. For array interface:

```
res = P.integ([coord], [field], [ratio]=[])
```
For pyTree interface, the variable to be integrated can be specified. If no variable is specified, all the fields located at nodes and centers are integrated:

```
res = P.integ(A, var='')
```
(See: integ.py) (See: integPT.py)

P.integNorm: compute the integral of each scalar field times the surface normal over the geometry defined by coord. For array interface:

```
res = P.integNorm([coord], [field], [ratio]=[])
```
For pyTree interface, the variable to be integrated can be specified. If no variable is specified, all the fields located at nodes and centers are integrated:

```
res = P.integNorm(A, var='')
```
(See: integNorm.py) (See: integNormPT.py)

P.integNormProduct: compute the integral of a vector field times the surface normal over the geometry defined by coord. The input field must have 3 variables. For array interface, field must be a vector field:

```
res = P.integNormProduct([coord], [field], [ratio]=[])
```
For pyTree interface, the vector field to be integrated must be specified:
```
res = P.integNormProduct(A, vector=[])
```
(See: integNormProduct.py) (See: integNormProductPT.py)

P.integMoment: compute the integral of a moment over the geometry defined by coord. The input field must have 3 variables. (cx,cy,cz) are the center coordinates. For array interface:

```
res = P.integMoment([coord], [field], [ratio]=[], center=(0.,0.,0.))
```
For pyTree interface, the vector of variables to be integrated must be specified:
```
res = P.integMoment(A, center=(0.,0.,0.), vector=[])
```
(See: integMoment.py) (See: integMomentPT.py)

P.integMomentNorm: compute the integral of a moment over the geometry defined by coord, taking into account the surface normal. The input field is a scalar. For array interface:

```
res = P.integMomentNorm([coord], [field], [ratio]=[], center=(cx,cy,cz))
```
For pyTree interface, the variable to be integrated can be specified. If no variable is specified, all the fields located at nodes and centers are integrated:
```
res = P.integMomentNorm(A, center=(cx,cy,cz), var='')
```
(See: integMomentNorm.py) (See: integMomentNormPT.py)

1.10 Example files

Example file: renameVars.py

```
# - renameVars (array) -
import Converter as C
import Post as P
import Generator as G

ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,2))
m = C.initVars(m, 'ro', 1.)
m = C.initVars(m, 'rou', 1.)

# Rename a list of variables
m2 = P.renameVars(m, ['ro','rou'], ['Density','MomentumX'])
C.convertArrays2File([m2], "out.plt")
```

Example file: renameVarsPT.py

```
# - renameVars (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,2))
m = C.addVars(m, ['Density', 'centers:MomentumX'])

# Rename a list of variables
m2 = P.renameVars(m, ['Density', 'centers:MomentumX'], ['Density_M', 'centers:MomentumX_M'])

C.convertPyTree2File([m2], 'out.cgns')
```

Example file: importVariablesPT.py

```
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

t1 = C.newPyTree(['Base']); t2 = C.newPyTree(['Base'])
z1 = G.cart((0.,0.,0.),(0.1,0.1,0.1),(10,10,10))
t1[2][1][2].append(z1); t2[2][1][2].append(z1)
t1 = C.initVars(t1,'centers:cellN',1.)
t2 = C.initVars(t2,'centers:cellN',0.)
t1 = C.initVars(t1,'centers:Density',1.)
t1 = C.initVars(t1,'Pressure',10.)

t2 = P.importVariables(t1, t2)
C.convertPyTree2File(t2, 'out.cgns')
```

Example file: computeVariables.py

```
# - computeVariables (array) -
import Converter as C
import Post as P
import Generator as G

ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,2))
c = C.array('ro,rou, rov,row,roE', ni, nj, 2)
c = C.initVars(c, 'ro', 1.)
c = C.initVars(c, 'rou', 1.)
c = C.initVars(c, 'rov', 0.)
c = C.initVars(c, 'row', 0.)
c = C.initVars(c, 'roE', 1.)
m = C.addVars([m, c])

#---------------------------------------------
# Pressure and Mach number extraction
# default values of rgp and gamma are used
#---------------------------------------------
p = P.computeVariables(m, ['Mach', 'Pressure'])
m = C.addVars([m, p])
C.convertArrays2File([m], "out.plt")
```

Example file: computeVariablesPT.py

```
# - computeVariables (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G
```

```
ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,2))
vars = ['Density','MomentumX', 'MomentumY', 'MomentumZ', \
        'EnergyStagnationDensity']
for v in vars: m = C.addVars(m, v)

# Pressure and Mach number extraction
m = P.computeVariables(m, ['Mach', 'Pressure'])
C.convertPyTree2File(m, 'out.cgns')
```

Example file: computeExtraVariable.py

```
# - computeExtraVariable (array) -
import Generator as G
import Converter as C
import Post as P
import Transform as T

a = G.cart( (0,0,0), (1,1,1), (50,50,50) )
a = C.initVars(a, 'Density', 1.)
a = C.initVars(a, 'MomentumX', 1.)
a = C.initVars(a, 'MomentumY', 0.)
a = C.initVars(a, 'MomentumZ', 0.)
a = C.initVars(a, 'EnergyStagnationDensity', 100000.)
m = P.computeExtraVariable(a, 'VorticityMagnitude')
q = P.computeExtraVariable(a, 'QCriterion')
tau = P.computeExtraVariable(a, 'ShearStress')
a = C.node2Center(a)
a = C.addVars([a, m, q, tau])

# Skin friction requires a surface array with shear stress already computed
wall = T.subzone(a, (1,1,1), (49,49,1))
skinFriction = P.computeExtraVariable(wall, 'SkinFriction')
skinFrictionTangential = P.computeExtraVariable(wall, 'SkinFrictionTangential')
wall = C.addVars([wall, skinFriction, skinFrictionTangential])
C.convertArrays2File([wall], 'out.plt')
```

Example file: computeExtraVariablePT.py

```
# - computeExtraVariable (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Transform.PyTree as T
import Post.PyTree as P

def F(x,y): return x*x + y*y

a = G.cart((0,0,0), (1,1,1), (50,50,50))
a = C.initVars(a, 'Density', 1.)
a = C.initVars(a, 'MomentumX', F, ['CoordinateX', 'CoordinateY'])
a = C.initVars(a, 'MomentumY', 0.)
a = C.initVars(a, 'MomentumZ', 0.)
a = C.initVars(a, 'EnergyStagnationDensity', 100000.)
a = P.computeExtraVariable(a, 'centers:VorticityMagnitude')
a = P.computeExtraVariable(a, 'centers:QCriterion')
a = P.computeExtraVariable(a, 'centers:ShearStress')

b = T.subzone(a, (1,1,1), (50,50,1))
b = P.computeExtraVariable(b, 'centers:SkinFriction')
```

```
b = P.computeExtraVariable(b, 'centers:SkinFrictionTangential')

C.convertPyTree2File(a, 'out.cgns')
```

Example file: computeGrad.py

```
# - computeGrad (array) -
import Converter as C
import Post as P
import Generator as G

ni = 1001; nj = 1001; nk = 1
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m, '{ro}= 2*{x}+{x}*{y}')
p = P.computeGrad(m,'ro') # p is defined on centers
p = C.center2Node(p) # back on initial mesh
p = C.addVars([m, p])
C.convertArrays2File([p], 'out.plt')
```

Example file: computeGradPT.py

```
# - computeGrad (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

ni = 30; nj = 40; nk = 10
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m, 'Density=2*{CoordinateX}+{CoordinateX}*{CoordinateY}')
m = P.computeGrad(m, 'Density')
C.convertPyTree2File(m, 'out.cgns')
```

Example file: computeGrad2.py

```
# - computeGrad2 (array) -
import Converter as C
import Post as P
import Generator as G

m = G.cartNGon((0,0,0), (1,1,1), (4,4,4))

mc = C.node2Center(m)
mc = C.initVars(mc, '{ro}= 2*{x}+{x}*{y}')
#mc = C.initVars(mc, '{ro}=1.')
mv = C.extractVars(mc, ['ro'])

p = P.computeGrad2(m, mv) # p is defined on centers
p = C.center2Node(p) # back on initial mesh
p = C.addVars([m, p])
C.convertArrays2File([p], 'out.plt')
```

Example file: computeGrad2.py

```
# - computeGrad2 (array) -
import Converter as C
import Post as P
import Generator as G

m = G.cartNGon((0,0,0), (1,1,1), (4,4,4))

mc = C.node2Center(m)
mc = C.initVars(mc, '{ro}= 2*{x}+{x}*{y}')
```

```
#mc = C.initVars(mc, '{ro}=1.')
mv = C.extractVars(mc, ['ro'])

p = P.computeGrad2(m, mv) # p is defined on centers
p = C.center2Node(p) # back on initial mesh
p = C.addVars([m, p])
C.convertArrays2File([p], 'out.plt')
```

Example file: computeNormGrad.py

```
# - computeNormGrad (array) -
import Converter as C
import Post as P
import Generator as G

ni = 11; nj = 11; nk = 1
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m, '{ro}= 2*{x}+{x}*{y}')
p = P.computeNormGrad(m,'ro') # p is defined on centers
p = C.center2Node(p) # back on initial mesh
p = C.addVars([m, p])
C.convertArrays2File([p], 'out.plt')
```

Example file: computeNormGradPT.py

```
# - computeNormGrad (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

ni = 30; nj = 40; nk = 10
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m, 'Density=2*{CoordinateX}+{CoordinateX}*{CoordinateY}')
m = P.computeNormGrad(m, 'Density')
C.convertPyTree2File(m, 'out.cgns')
```

Example file: computeCurl.py

```
# - computeCurl (array) -
import Converter as C
import Post as P
import Generator as G

def F(x,y,z):
    return 12*y*y + 4

ni = 30; nj = 40; nk = 3
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m,'F1',F,['x','y','z'])
m = C.initVars(m,'F2',0.); m = C.initVars(m,'F3',0.)

varname = ['F1','F2','F3']
p = P.computeCurl(m, varname) # defined on centers
p = C.center2Node(p) # back on init grid
p = C.addVars([m,p])
C.convertArrays2File([p], "out.plt")
```

Example file: computeCurlPT.py

```
# - computeCurl (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

ni = 30; nj = 40; nk = 3
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m,'F1=12*{CoordinateY}*{CoordinateY}+4')
m = C.addVars(m,'F2'); m = C.addVars(m,'F3')

varname = ['F1','F2','F3']
m = P.computeCurl(m, varname)
C.convertPyTree2File(m, 'out.cgns')
```

Example file: computeNormCurl.py

```
# - computeNormCurl (array) -
import Converter as C
import Post as P
import Generator as G

def F(x,y,z):
    return 12*y*y + 4

ni = 30; nj = 40; nk = 3
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m,'F1',F,['x','y','z'])
m = C.initVars(m,'F2',0.); m = C.initVars(m,'F3',0.)

varname = ['F1','F2','F3']
p = P.computeNormCurl(m, varname) # defined on centers
p = C.center2Node(p) # back on init grid
p = C.addVars([m,p])
C.convertArrays2File([p], "out.plt")
```

Example file: computeNormCurlPT.py

```
# - computeNormCurl (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

def F(x,y,z): return 12*y*y + 4

ni = 30; nj = 40; nk = 3
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m,'F1',F,['CoordinateX','CoordinateY','CoordinateZ'])
m = C.addVars(m,'F2'); m = C.addVars(m,'F3')

varname = ['F1','F2','F3']
m = P.computeNormCurl(m, varname)
C.convertPyTree2File(m, 'out.cgns')
```

Example file: computeDiff.py

```
# - computeDiff (array) -
import Converter as C
import Post as P
import Generator as G

def F(x):
    if ( x > 5. ): return True
    else : return False
```

```
ni = 30; nj = 40; nk = 1
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1.), (ni,nj,nk))
m = C.initVars(m, 'ro', F, ['x'])
p = P.computeDiff(m,'ro')
p = C.addVars([m, p])
C.convertArrays2File([p], 'out.plt')
```

Example file: computeDiffPT.py

```
# - computeDiff (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

ni = 30; nj = 40; nk = 1
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m, 'Density=({CoordinateX}>5)*1.')
m = P.computeDiff(m, 'Density')
C.convertPyTree2File(m, 'out.cgns')
```

Example file: selectCells.py

```
# - selectCells (array) -
import Converter as C
import Generator as G
import Post as P

a = G.cart( (0,0,0), (1,1,1), (11,11,11) )
def F(x, y, z):
    if (x + 2*y + z > 20.): return True
    else: return False

b = P.selectCells(a, F, ['x', 'y', 'z'])
c = P.selectCells(a, F, ['x', 'y', 'z'], strict=1)
d = P.selectCells(a, '{x}+2*{y}+{z}>20')
e = P.selectCells(a, '({x}>2) & ({y}>2)')
C.convertArrays2File([b,c,d,e], 'out.plt')
```

Example file: selectCellsPT.py

```
# - selectCells (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

def F(x, y, z):
    if (x + 2*y + z > 20.): return True
    else: return False

a = G.cart( (0,0,0), (1,1,1), (11,11,11) )
a = P.selectCells(a, F, ['CoordinateX', 'CoordinateY', 'CoordinateZ'])
C.convertPyTree2File(a, 'out.cgns')
```

Example file: selectCells2.py

```
# - selectCells2 (array) -
import Converter as C
import Generator as G
import Post as P

a = G.cart((0,0,0), (1,1,1), (11,11,11))
tag = C.array('tag', 10, 10, 10); tag = C.initVars(tag, 'tag', 1.)
b = P.selectCells2(a, tag)
C.convertArrays2File([b], 'out.plt')
```

Example file: selectCells2PT.py

```
# - selectCells2 (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

a = G.cart((0,0,0), (1,1,1), (11,11,11))
a = C.initVars(a, 'tag', 1.)
b = P.selectCells2(a, 'tag')
C.convertPyTree2File(b, 'out.cgns')
```

Example file: interiorFaces.py

```
# - interiorFaces (array) -
import Converter as C
import Post as P
import Generator as G

# Get interior faces in broad sense :
# faces with 2 neighbours
a = G.cartTetra((0,0,0), (1,1.,1), (20,2,1))
b = P.interiorFaces(a)
C.convertArrays2File([a,b], 'out1.plt')

# Get interior faces in strict sense :
# faces having only interior nodes
a = G.cartTetra((0,0,0), (1,1.,1), (20,3,1))
b = P.interiorFaces(a,1)
C.convertArrays2File([a,b], 'out2.plt')
```

Example file: interiorFacesPT.py

```
# - interiorFaces (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

a = G.cartTetra((0,0,0), (1,1.,1), (20,20,1))
b = P.interiorFaces(a)
t = C.newPyTree(['Base',1]); t[2][1][2].append(b)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: exteriorFaces.py

```
# - exteriorFaces (array) -
import Converter as C
import Post as P
import Generator as G

a = G.cartTetra((0,0,0), (1,1,1), (20,20,20))
indices = []
b = P.exteriorFaces(a, indices=indices)
print indices
C.convertArrays2File([b], 'out.plt')
```

Example file: exteriorFacesPT.py

```
# - exteriorFaces (pyTree)-
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

a = G.cartTetra((0,0,0), (1,1,1), (4,4,6))
b = P.exteriorFaces(a)
C.convertPyTree2File(b, 'out.cgns')
```

Example file: exteriorFacesStructured.py

```
# - exteriorFacesStructured (array) -
import Converter as C
import Post as P
import Generator as G

a = G.cart((0,0,0), (1,1,1), (4,4,6))
A = P.exteriorFacesStructured(a)
C.convertArrays2File(A, 'out.plt')
```

Example file: exteriorFacesStructuredPT.py

```
# - exteriorFacesStructured (pyTree)-
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (4,4,6))
zones = P.exteriorFacesStructured(a)
C.convertPyTree2File(zones, 'out.cgns')
```

Example file: exteriorElts.py

```
# - exteriorElts (array) -
import Converter as C
import Post as P
import Generator as G

a = G.cartTetra((0,0,0), (1,1,1), (10,10,10))
b = P.exteriorElts(a)
C.convertArrays2File([b], 'out.plt')
```

Example file: exteriorEltsPT.py

```
# - exteriorElts (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

a = G.cartTetra((0,0,0), (1,1,1), (10,10,10))
b = P.exteriorElts(a)
C.convertPyTree2File(b, 'out.cgns')
```

Example file: frontFaces.py

```
# - frontFaces (array) -
import Converter as C
import Generator as G
import Post as P

a = G.cart( (0,0,0), (1,1,1), (11,11,11) )
def F(x, y, z):
    if (x + 2*y + z   > 20.): return 1
    else: return 0
a = C.initVars(a, 'tag', F, ['x', 'y', 'z'])
t = C.extractVars(a, ['tag'])
f = P.frontFaces(a, t)
C.convertArrays2File([a,f], 'out.plt')
```

Example file: frontFacesPT.py

```
# - frontFaces (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

a = G.cart( (0,0,0), (1,1,1), (11,11,11) )
def F(x, y, z):
    if (x + 2*y + z > 20.): return 1
    else: return 0
a = C.initVars(a, 'tag', F, ['CoordinateX', 'CoordinateY', 'CoordinateZ'])
f = P.frontFaces(a, 'tag'); f[0] = 'front'
t = C.newPyTree(['Base']); t[2][1][2] += [a, f]
C.convertPyTree2File(t, 'out.cgns')
```

Example file: sharpEdges.py

```
# - sharpEdges ( array) -
import Converter as C
import Generator as G
import Post as P
import Transform as T
a1 = G.cart((0.,0.,0.),(1.5,1.,1.),(2,2,1))
a2 = T.rotate(a1,(0.,0.,0.),(0.,1.,0.),100.)
res = P.sharpEdges([a1,a2],alphaRef=45.)
C.convertArrays2File([a1,a2]+res,"out.plt")
```

Example file: sharpEdgesPT.py

```
# - sharpEdges (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P
import Transform.PyTree as T
a1 = G.cart((0.,0.,0.),(1.5,1.,1.),(2,2,1))
a2 = T.rotate(a1,(0.,0.,0.),(0.,1.,0.),100.);a2[0] = 'cart2'
res = P.sharpEdges([a1,a2],alphaRef=45.)
t = C.newPyTree(['Edge',1,'Base',2]); t[2][1][2] += res; t[2][2][2]+=[a1,a2]
C.convertPyTree2File(t,"out.cgns")
```

Example file: silhouette.py

```
# - silhouette ( array) -
import Generator as G
import Converter as C
import Post as P

a = G.cylinder((0.,0.,0.), 0.5, 1., 360., 0., 10., (50,1,30))

vector=[1.,0.,0.]
res = P.silhouette([a], vector)

l = [a]+res
C.convertArrays2File(l, 'out.plt')
```

Example file: silhouettePT.py

```
# - silhouette (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Post.PyTree as P
```

```
a = G.cylinder((0.,0.,0.), 0.5, 1., 360., 0., 10., (50,1,30))
t = C.newPyTree(['Base']); t[2][1][2].append(a)

vector=[1.,0.,0.]
res = P.silhouette(a, vector)
t[2][1][2] += res
C.convertPyTree2File(t, 'out.cgns')
```

Example file: coarsen.py

```
# - coarsen (array) -
import Post as P
import Converter as C
import Generator as G
import Transform as T

# coarsen all cells of a square
ni = 21; nj = 21; nk = 11
hi = 2./(ni-1); hj = 2./(nj-1); hk = 1./(nk-1)
m = G.cart((0.,0.,0.),(hi,hj,hk), (ni,nj,nk))

hi = hi/2; hj = hj/2; hk = hk/2
m2 = G.cart((0.,0.,0.),(hi,hj,hk), (ni,nj,nk))
m2 = T.subzone(m2,(3,3,6),(m[2]-2,m[3]-2,6))
m2 = T.translate(m2, (0.75,0.75,0.25))
m2 = T.perturbate(m2, 0.51)
tri = G.delaunay(m2)

npts = tri[2].shape[1]
indic = C.array('indic', npts, 1, 1)
indic = C.initVars(indic, 'indic', 1)

sol = P.coarsen(tri, indic, argqual=0.25, tol=1.e6)
C.convertArrays2File([tri, sol], 'out.plt')
```

Example file: coarsenPT.py

```
# - coarsen (pyTree)-
import Post.PyTree as P
import Converter.PyTree as C
import Generator.PyTree as G
import Transform.PyTree as T

ni = 21; nj = 21; nk = 1
hi = 2./(ni-1); hj = 2./(nj-1)
m = G.cart((0.,0.,0.),(hi,hj,1.), (ni,nj,nk)); m = T.perturbate(m, 0.51)
tri = G.delaunay(m); tri = C.initVars(tri, 'centers:indic', 1.)

sol = P.coarsen(tri, 'indic'); sol[0] = 'coarse'

C.convertPyTree2File([sol,tri], 'out.cgns')
```

Example file: refine.py

```
# - refine (array) -
import Post as P
import Converter as C
import Generator as G

# Using indic (linear)
```

```
a = G.cartTetra((0,0,0), (1,1,1), (10,10,1))
indic = C.array('indic', a[2].shape[1], 1, 1)
indic = C.initVars(indic, 'indic', 0)
C.setValue(indic, 50, [1])
C.setValue(indic, 49, [1])
a = P.refine(a, indic)
C.convertArrays2File(a, 'out.plt')

# Using butterfly
a = G.cartTetra((0,0,0), (2,1,1), (3,3,3))
a = P.exteriorFaces(a)
#a = C.initVars(a, "z = 0.1*{x}*{x}+0.2*{y}")
for i in xrange(6):
    a = P.refine(a, w=1./64.)
C.convertArrays2File(a, 'out.plt')
```

Example file: refinePT.py

```
# - refine (pyTree) -
import Post.PyTree as P
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D

# Linear with indicator field
a = G.cartTetra((0,0,0), (1,1,1), (10,10,1))
a = C.initVars(a, 'centers:indic', 1.)
a = P.refine(a, 'indic')
C.convertPyTree2File(a, 'out.cgns')
```

Example file: refine2.py

```
# - refine (array) -
import Post as P
import Converter as C
import Generator as G

# Refine using butterfly
a = G.cartTetra((0,0,0), (2,1,1), (3,3,3))
a = P.exteriorFaces(a)
for i in xrange(5):
    a = P.refine(a, w=1./64.)
C.convertArrays2File(a, 'out.plt')
```

Example file: refine2PT.py

```
# - refine (pyTree) -
import Post.PyTree as P
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D

# Refine with butterfly interpolation
a = G.cartTetra((0,0,0), (1,1,1), (10,10,1))
a = P.refine(a, w=1./64.)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: computeIndicatorValue.py

```
# - compIndicatorValue(array) -
import Generator as G
import Converter as C
import Geom as D
import Post as P
import KCore.test as test

s = D.circle((0,0,0),1.)
snear = 0.1
o = G.octree([s], [snear], dfar=10., balancing=0)
res = G.octree2Struct(o, vmin=11,merged=0)
vol = G.getVolumeMap(res); res = C.node2Center(res)
val = P.computeIndicatorValue(o,res,vol)
o = C.addVars([o,val])
C.convertArrays2File([o], "out.plt")
```

Example file: computeIndicatorValuePT.py

```
# - compIndicatorValue(pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D
import Post.PyTree as P

s = D.circle((0,0,0),1.)
snear = 0.1
o = G.octree([s], [snear], dfar=10., balancing=1)
res = G.octree2Struct(o, vmin=11,merged=1)
res = G.getVolumeMap(res)
o = P.computeIndicatorValue(o,res,'centers:vol')
t = C.newPyTree(['Base']); t[2][1][2] +=[o]
C.convertPyTree2File(t,"out.cgns")
```

Example file: computeIndicatorField.py

```
# - compIndicatorField (array) -
import Generator as G
import Converter as C
import Geom as D
import Post as P
import KCore.test as test

s = D.circle((0,0,0), 1., N=100); snear = 0.1
o = G.octree([s], [snear], dfar=10., balancing=1)
npts = len(o[1][0])
indicVal = G.getVolumeMap(o)
indicator, valInf, valSup = P.computeIndicatorField(
    o, indicVal, nbTargetPts=2.*npts, bodies=[s])
indicator = C.center2Node(indicator)
o = C.addVars([o, indicator])
C.convertArrays2File([o], "out.plt")
```

Example file: computeIndicatorFieldPT.py

```
# - compIndicatorField (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D
import Post.PyTree as P

#----------------------
# indicateur en centres
#----------------------
s = D.circle((0,0,0), 1., N=100); snear = 0.1
```

```
o = G.octree([s], [snear], dfar=10., balancing=1)
npts = o[1][0][0]
o = G.getVolumeMap(o)
o, valInf, valSup = P.computeIndicatorField(o, 'centers:vol',
                                            nbTargetPts=2*npts, bodies=[s])
C.convertPyTree2File(o, 'out.cgns')
```

Example file: extractPoint.py

```
# - extractPoint (array) -
import Converter as C
import Generator as G
import Post as P

ni = 10; nj = 10; nk = 10;
a = G.cart((0,0,0), (1./(ni-1),1./(nj-1),1./(nk-1)), (ni,nj,nk))
def F(x,y,z): return x*x*x*x + 2.*y + z*z
a = C.initVars(a, 'F', F, ['x','y','z'])

# Utilisation directe
val = P.extractPoint([a], (0.55, 0.38, 0.12), 2); print val

# Utilisation avec un hook
hook = C.createHook([a], function='extractMesh')
val = P.extractPoint([a], (0.55, 0.38, 0.12), 2, hook=hook); print val
```

Example file: extractPointPT.py

```
# - extractPoint (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

ni = 10; nj = 10; nk = 10;
a = G.cart((0,0,0), (1./(ni-1),1./(nj-1),1./(nk-1)), (ni,nj,nk))
def F(x,y,z): return x + 2.*y + 3.*z
a = C.initVars(a, 'F', F, ['CoordinateX','CoordinateY','CoordinateZ'])

# Utilisation directe
val = P.extractPoint(a, (0.55, 0.38, 0.12), 2); print val

# Utilisation avec un hook
hook = C.createHook(a, function='extractMesh')
val = P.extractPoint(a, (0.55, 0.38, 0.12), 2, hook=hook); print val
```

Example file: extractPlane.py

```
# - extractPlane (array) -
import Converter as C
import Post as P
import Transform as T
import Generator as G

m = G.cylinder((0,0,0), 1, 5, 0., 360., 10., (50,50,50))
m = T.rotate(m , (0,0,0), (1,0,0), 35.)
a = P.extractPlane([m], (0.5, 1., 0., 1), 2)
C.convertArrays2File([m,a], "out.plt")
```

Example file: extractPlanePT.py

```
# - extractPlane (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Transform.PyTree as T
import Generator.PyTree as G

m = G.cylinder((0,0,0), 1, 5, 0., 360., 10., (50,50,50))
m = T.rotate(m, (0,0,0), (1,0,0), 35.)
m = C.initVars(m,'Density',1); m = C.initVars(m,'centers:cellN',1)
z = P.extractPlane(m, (0.5, 1., 0., 1), 2)
C.convertPyTree2File(z, 'out.cgns')
```

Example file: extractMesh.py

```
# - extractMesh (array) -
import Converter as C
import Post as P
import Generator as G

ni = 30; nj = 40; nk = 10
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m, 'ro', 1.)
# Create extraction mesh
a = G.cart((0.,0.,0.), (1., 0.1, 0.1), (20, 20, 1))
# Extract solution on extraction mesh
a2 = P.extractMesh([m], a)
C.convertArrays2File([m,a2], 'out.plt')
```

Example file: extractMeshPT.py

```
# - extractMesh (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G

ni = 30; nj = 40; nk = 10
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk))
m = C.initVars(m, 'Density', 2.)
m = C.initVars(m, 'centers:cellN', 1)

# Extraction mesh
a = G.cart((0.,0.,0.5), (1., 0.1, 1.), (20, 20, 1)); a[0] = 'extraction'

# Extract solution on extraction mesh
a = P.extractMesh(m, a)
t = C.newPyTree(['Solution', 3, 'Extraction', 2])
t[2][1][2].append(m); t[2][2][2] += [a]
C.convertPyTree2File(a, 'out.cgns')
```

Example file: projectCloudSolution.py

```
# -projectCloudSolution (array)
import Converter as C
import Geom as D
import Post as P
import Transform as T
import Generator as G

a = D.sphere((0,0,0),1.,N=20)
a = C.convertArray2Tetra(a); a = G.close(a)
b = D.sphere6((0,0,0),1.,N=15)
```

```
b = C.convertArray2Tetra(b); b = T.join(b)
pts = C.convertArray2Node(b)
pts = C.initVars(pts,'F={x}*{y}')
a = C.initVars(a,'F=0.')
a = P.projectCloudSolution(pts,a)
C.convertArrays2File(a,"out.plt")
```

Example file: projectCloudSolutionPT.py

```
# -projectCloudSolution (pyTree)
import Converter.PyTree as C
import Geom.PyTree as D
import Post.PyTree as P
import Transform.PyTree as T
import Generator.PyTree as G
import KCore.test as test

a = D.sphere((0,0,0),1.,N=20)
a = C.convertArray2Tetra(a); a = G.close(a)
b = D.sphere6((0,0,0),1.,N=15)
b = C.convertArray2Tetra(b); b = T.join(b)
pts = C.convertArray2Node(b)
C._initVars(pts,'F={CoordinateX}*{CoordinateY}')
C._initVars(a,'F=0.')
a = P.projectCloudSolution(pts,a)
C.convertPyTree2File(a,"out.cgns")
```

Example file: zipper.py

```
# - zipper (array) -
import Converter as C
import Post as P
import Generator as G
import Transform as T

m1 = G.cylinder((0,0,0), 1, 5, 0., 360., 10., (50,50,1))
m1 = C.initVars(m1, 'cellN', 1.)

# Set cellN = 2 (interpolated points) to boundary
s = T.subzone(m1, (1,m1[3],1),(m1[2],m1[3],m1[4]))
s = C.initVars(s, 'cellN', 2)
m1 = T.patch(m1, s, (1,m1[3],1))
s = T.subzone(m1, (1,1,1),(m1[2],1,m1[4]))
s = C.initVars(s, 'cellN', 2)
m1 = T.patch(m1, s, (1,1,1))

ni = 30; nj = 40
m2 = G.cart((0,0,0), (10./(ni-1),10./(nj-1),-1), (ni,nj,1))
m2 = C.initVars(m2, 'cellN', 1.)

array = P.zipper([m1,m2],[])
C.convertArrays2File([array], 'out.plt')
```

Example file: zipperPT.py

```
# - zipper (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G
import Transform.PyTree as T
```

```
# cylindre
ni = 30; nj = 40; nk = 1
m1 = G.cylinder((0,0,0), 1, 5, 0., 360., 10., (ni,nj,nk))
m1 = C.addVars(m1, 'Density'); m1 = C.initVars(m1,'cellN',1)

# Set cellN = 2 (interpolated points) to boundary
s = T.subzone(m1, (1,nj,1),(ni,nj,nk))
s = C.initVars(s, 'cellN', 2)
m1 = T.patch(m1, s, (1,nj,1))
s = T.subzone(m1, (1,1,1),(ni,1,nk))
s = C.initVars(s, 'cellN', 2)
m1 = T.patch(m1, s, (1,1,1))

# carre
ni = 30; nj = 40
m2 = G.cart((0,0,0), (10./(ni-1),10./(nj-1),-1), (ni,nj,1))
m2 = C.initVars(m2, 'Density', 1.2); m2 = C.initVars(m2, 'cellN', 1.)

t = C.newPyTree(['Base',2]); t[2][1][2] += [m1, m2]
z = P.zipper(t); z[0] = 'zipper'; t[2][1][2].append(z)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: usurp.py

```
# - usurp (array) -
import Post as P
import Converter as C
import Generator as G
import Transform as T

cyln = []
a1 = G.cylinder((0,0,0), 0, 2, 360, 0, 1., (100,2,10))
a1 = T.subzone(a1, (1,2,1), (a1[2],2,a1[4]))
cyln.append(a1)
#
a2 = G.cylinder((0,0,0), 0, 2, 90, 0, 0.5, (10,2,10))
a2 = T.translate(a2, (0,0,0.2))
a2 = T.subzone(a2, (1,2,1), (a2[2],2,a2[4]))
cyln.append(a2)
#
c1 = cyln[0]
ib1 = C.array('cellN', c1[2]-1, c1[3], c1[4]-1)
ib1 = C.initVars(ib1,'cellN', 1)
ib1[1][0,586] = 0.
#
c2 = cyln[1]
ib2 = C.array('cellN', c2[2]-1, c2[3], c2[4]-1)
ib2 = C.initVars(ib2, 'cellN', 1)

ibc = [ib1, ib2]; r = P.usurp(cyln, ibc)
cylc = C.node2Center(cyln)
out = []
l = len(cylc)
for i in range(l) :
    out.append(C.addVars([cylc[i], ibc[i]]))

C.convertArrays2File(out, 'outc.plt')
```

Example file: usurpPT.py

```
# - usurp (pyTree)-
```

```
import Post.PyTree as P
import Converter.PyTree as C
import Generator.PyTree as G
import Transform.PyTree as T

a1 = G.cylinder((0,0,0), 0, 2, 360, 0, 1., (100,2,10))
a1 = T.subzone(a1, (1,2,1), (100,2,10)); a1[0]='cyl1'

a2 = G.cylinder((0,0,0), 0, 2, 90, 0, 0.5, (10,2,10))
a2 = T.translate(a2, (0,0,0.2))
a2 = T.subzone(a2, (1,2,1),(10,2,10)); a2[0]='cyl2'

a1 = C.initVars(a1, 'centers:cellN',1.)
a2 = C.initVars(a2, 'centers:cellN',1.)
t = C.newPyTree(['Base',2])
t[2][1][2] += [a1, a2]
t = P.usurp(t)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: streamLine.py

```
# - streamLine (array) -
import Converter as C
import Post as P
import Generator as G
import math as M

ni = 30; nj = 40
m1 = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,2))
m2 = G.cart((5.5,0,0), (9./(ni-1),9./(nj-1),1), (ni,nj,2))

def F(x): return M.cos(x)

m = [m1,m2]
m = C.initVars(m, 'rou', 1.)
m = C.initVars(m, 'rov', F, ['x'])
m = C.initVars(m, 'row', 0.)
x0=0.1; y0=5.; z0=0.5; p = P.streamLine(m, (x0,y0,z0),['rou','rov','row'])
C.convertArrays2File(m+[p], 'out.plt')
```

Example file: streamLinePT.py

```
# - streamLine (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G
import math as M


def F(x): return M.cos(x)
ni = 30; nj = 40
m1 = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,2))
m2 = G.cart((5.5,0,0), (9./(ni-1),9./(nj-1),1), (ni,nj,2))
t = C.newPyTree(['Base','StreamR']); t[2][1][2] = [m1,m2]
t = C.initVars(t, 'vx', 1.)
t = C.initVars(t, 'vy', F, ['CoordinateX'])
t = C.initVars(t, 'vz', 0.)
x0=0.1; y0=5.; z0=0.5
p = P.streamLine(t, (x0,y0,z0),['vx','vy','vz'])
C.convertPyTree2File(p, "out.cgns")
```

Example file: streamRibbon.py

```
# - streamRibbon (array) -
import Converter as C
import Post as P
import Generator as G
import math as M

ni = 30; nj = 40
def F(x): return M.cos(x)

m1 = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,2))
m2 = G.cart((5.5,0,0), (9./(ni-1),9./(nj-1),1), (ni,nj,2))
m = [m1,m2]
m = C.initVars(m, 'rou', 1.)
m = C.initVars(m, 'rov', F, ['x'])
m = C.initVars(m, 'row', 0.)
x0=0.1; y0=5.; z0=0.5; p = P.streamRibbon(m, (x0,y0,z0),(0.,0.2,0.),['rou','rov','row'])
C.convertArrays2File(m+[p], 'out.plt')
```

Example file: streamRibbonPT.py

```
# - streamRibbon (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G
import math as M

def F(x): return M.cos(x)
ni = 30; nj = 40
m1 = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,2))
m2 = G.cart((5.5,0,0), (9./(ni-1),9./(nj-1),1), (ni,nj,2))
t = C.newPyTree(['Base','StreamR']); t[2][1][2] = [m1,m2]
t = C.initVars(t, 'vx', 1.)
t = C.initVars(t, 'vy', F, ['CoordinateX'])
t = C.initVars(t, 'vz', 0.)
x0=0.1; y0=5.; z0=0.5
p = P.streamRibbon(t, (x0,y0,z0),(0.,0.2,0.),['vx','vy','vz'])
C.convertPyTree2File(p, "out.cgns")
```

Example file: streamSurf.py

```
# - streamSurf (array) -
import Converter as C
import Post as P
import Generator as G
import Geom as D
import math as M

ni = 30; nj = 40

# Node mesh
m1 = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,5))
m2 = G.cart((5.5,0,0), (9./(ni-1),9./(nj-1),1), (ni,nj,5))
b = D.line((0.1,5.,0.1), (0.1,5.,3.9), N=10)
b = C.convertArray2Tetra(b)
m = [m1,m2]
def F(x): return M.cos(x)

m = C.initVars(m, 'rou', 1.)
m = C.initVars(m, 'rov', F, ['x'])
m = C.initVars(m, 'row', 0.)
p = P.streamSurf(m, b,['rou','rov','row'])
C.convertArrays2File(m+[p], 'out.plt')
```

Example file: streamSurfPT.py

```
# - streamSurf (pyTree) -
import Converter.PyTree as C
import Post.PyTree as P
import Generator.PyTree as G
import Geom.PyTree as D
import math as M

ni = 30; nj = 40; nk = 5
m1 = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,nk)); m1[0] = 'cart1'
m2 = G.cart((5.5,0,0), (9./(ni-1),9./(nj-1),1), (ni,nj,nk)); m2[0] = 'cart2'
b = D.line((0.1,5.,0.1), (0.1,5.,3.9), N=10)
b = C.convertArray2Tetra(b)
def F(x): return M.cos(x)

t = C.newPyTree(['Base','StreamR']); t[2][1][2] = [m1,m2]
t = C.initVars(t, 'vx', 1.)
t = C.initVars(t, 'vy', F, ['CoordinateX'])
t = C.initVars(t, 'vz', 0.)
x0=0.1; y0=5.; z0=0.5
p = P.streamSurf(t, b,['vx','vy','vz'])
C.convertPyTree2File(p, "out.cgns")
```

Example file: isoLine.py

```
# - isoLine (array) -
import Post as P
import Converter as C
import Generator as G
import Geom as D

def F(x, y): return x*x+y*y

a = G.cartTetra( (0,0,0), (1,1,1), (10,10,1))
a = C.initVars(a, 'field', F, ['x','y'])

isos = []
min = C.getMinValue(a, 'field')
max = C.getMaxValue(a, 'field')
for v in xrange(20):
    value = min + (max-min)/18.*v
    i = P.isoLine(a, 'field', value)
    if (i != []): isos.append(i)
C.convertArrays2File([a]+isos, 'out.plt')
```

Example file: isoLinePT.py

```
# - isoLine (pyTree) -
import Post.PyTree as P
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D

def F(x, y): return x*x+y*y

a = G.cartTetra( (0,0,0), (1,1,1), (10,10,1))
a = C.initVars(a, 'field', F, ['CoordinateX','CoordinateY'])

isos = []
min = C.getMinValue(a, 'field')
```

```
max = C.getMaxValue(a, 'field')
for v in xrange(20):
    value = min + (max-min)/18.*v
    try:
        i = P.isoLine(a, 'field', value)
        isos.append(i)
    except: pass
t = C.newPyTree(['Base',3,'ISOS',1])
t[2][1][2].append(a)
t[2][2][2] += isos
C.convertPyTree2File(t, 'out.cgns')
```

Example file: isoSurf.py

```
# - isoSurf (array) -
import Post as P
import Converter as C
import Generator as G
import Geom as D

a = G.cartTetra( (-20,-20,-20), (0.25,0.25,0.5), (100,100,50))
a = C.initVars(a, 'field={x}*{x}+{y}*{y}+{z}')

iso = P.isoSurf(a, 'field', value=5.)
C.convertArrays2File(iso, 'out.plt')
```

Example file: isoSurfPT.py

```
# - isoSurf (pyTree) -
import Post.PyTree as P
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cartTetra( (-20,-20,-20), (0.5,0.5,0.5), (50,50,50))
a = C.initVars(a, 'field={CoordinateX}*{CoordinateX}+{CoordinateY}*{CoordinateY}+{CoordinateZ}')

iso = P.isoSurf(a, 'field', value=5.)
C.convertPyTree2File(iso, 'out.cgns')
```

Example file: isoSurfMC.py

```
# - isoSurfMC (array) -
import Post as P
import Converter as C
import Generator as G
import Geom as D

a = G.cartHexa( (-20,-20,-20), (0.25,0.25,0.5), (100,100,50))
a = C.initVars(a, 'field={x}*{x}+{y}*{y}+{z}')

iso = P.isoSurfMC(a, 'field', value=5.)
C.convertArrays2File(iso, 'out.plt')
```

Example file: isoSurfMCPT.py

```
# - isoSurfMC (pyTree) -
import Post.PyTree as P
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cartHexa( (-20,-20,-20), (0.5,0.5,0.5), (50,50,50))
a = C.initVars(a, 'field={CoordinateX}*{CoordinateX}+{CoordinateY}*{CoordinateY}+{CoordinateZ}')

iso = P.isoSurfMC(a, 'field', value=5.)
C.convertPyTree2File(iso, 'out.cgns')
```

Example file: integ.py

```
# - integ (array) -
import Converter as C
import Generator as G
import Post as P

# Node mesh
ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,1))

# Field in centers
c = C.array('vx', ni-1, nj-1, 1); c = C.initVars(c, 'vx', 1.)
resc = P.integ([m], [c], [])[0]; print resc

# Field in nodes
cn = C.array('vx', ni, nj, 1); cn = C.initVars(cn, 'vx', 1.)
resn = P.integ([m], [cn], [])[0]; print resn
```

Example file: integPT.py

```
# - integ (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,1))
m = C.initVars(m, 'vx', 1.); m = C.initVars(m, 'ratio', 1.)
resn = P.integ(m, 'vx'); print resn
```

Example file: integNorm.py

```
# - integNorm (array) -
import Converter as C
import Generator as G
import Post as P

# Node mesh and field
m = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
c1 = C.array('ro', 100, 162, 'TRI')
c = C.initVars(c1, 'ro', 1.)
res = P.integNorm([m], [c], []); print 'res1 = ',res

# Node mesh
ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,1))

# Centers field
c1 = C.array('vx', ni-1, nj-1, 1)
c = C.initVars(c1, 'vx', 1.)

# Integration
res = P.integNorm([m], [c], []); print 'res2 = ',res

# Node field
c1 = C.array('vx, vy', ni, nj, 1)
cn = C.initVars(c1, 'vx', 1.)
cn = C.initVars(c1, 'vy', 1.)
resn = P.integNorm([m], [cn], []); print 'res3 = ',resn
```

Example file: integNormPT.py

```
# - integNorm (pyTree)-
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

# Node mesh
m = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
m = C.initVars(m, 'Density', 1.)
t = C.newPyTree(['Base',2]); t[2][1][2].append(m)
res = P.integNorm(t, 'Density'); print res
```

Example file: integNormProduct.py

```
# - integNormProduct (array) -
import Converter as C
import Generator as G
import Post as P

# Maillage et champs non structure, en noeuds
m = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
c = C.array('vx,vy,vz', m[1].shape[1], m[2].shape[1], 'TRI')
c = C.initVars(c, 'vx,vy,vz', 1.)
res = P.integNormProduct([m], [c], []); print res

# Maillage en noeuds
ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,1))

# Champ a integrer en centres
c = C.array('vx,vy,vz', ni-1, nj-1, 1)
c = C.initVars(c, 'vx,vy,vz', 1.)

# Integration de chaque champ
res = P.integNormProduct([m], [c], []); print res

# Champ a integrer en noeuds
cn = C.array('vx,vy,vz', ni, nj, 1)
cn = C.initVars(cn, 'vx,vy,vz', 1.)
resn = P.integNormProduct([m], [cn], []); print resn
```

Example file: integNormProductPT.py

```
# - integNormProduct (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,1))
m = C.initVars(m,'MomentumX',1.)
m = C.initVars(m,'MomentumY',1.)
m = C.initVars(m,'MomentumZ',1.)
res = P.integNormProduct(m,['MomentumX','MomentumY','MomentumZ']); print res
```

Example file: integMoment.py

```
# - integMoment (array) -
import Converter as C
import Generator as G
import Post as P

# Maillage et champs non structure, en noeuds
m = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
c = C.array('vx,vy,vz', 100, 162, 'TRI')
c = C.initVars(c, 'vx,vy,vz', 1.)
res = P.integMoment([m], [c], [], (5.,5., 0.)); print res

# Maillage en noeuds
ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,1))
C.convertArrays2File([m], "new.plt", "bin_tp")

# Champ a integrer en centres
c = C.array('vx,vy,vz', ni-1, nj-1, 1)
c = C.initVars(c, 'vx,vy,vz', 1.)

# Integration de chaque champ
res = P.integMoment([m], [c], [], (5.,5., 0.) ); print res

# Champ a integrer en noeuds
cn = C.array('vx,vy,vz', ni, nj, 1)
cn = C.initVars(cn, 'vx,vy,vz', 1.)
resn = P.integMoment([m], [cn], [], (5.,5., 0.)); print resn
```

Example file: integMomentPT.py

```
# - integMoment (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

m = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
m = C.initVars(m,'vx',1.); m = C.initVars(m,'vy',0.); m = C.initVars(m,'vz',0.)
res = P.integMoment(m, center=(5.,5., 0.),vector=['vx','vy','vz']); print res
```

Example file: integMomentNorm.py

```
# - integMomentNorm (array) -
import Converter as C
import Generator as G
import Post as P

# Maillage et champs non structure, en noeuds
m = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
c = C.array('ro', 100, 162, 'TRI')
c = C.initVars(c, 'ro', 1.)
res = P.integMomentNorm([m], [c], [], (5.,5., 0.)); print res

# Maillage en noeuds
ni = 30; nj = 40
m = G.cart((0,0,0), (10./(ni-1),10./(nj-1),1), (ni,nj,1))

# Champ a integrer en centres
c = C.array('v', ni-1, nj-1, 1)
c = C.initVars(c, 'v', 1.)

# Integration de chaque champ
res = P.integMomentNorm([m], [c], [], (5.,5., 0.) ); print res

# Champ a integrer en noeuds
```

```
cn = C.array('v', ni, nj, 1)
cn = C.initVars(cn, 'v', 1.)
resn = P.integMomentNorm([m], [cn], [], (5.,5., 0.))
print resn
```

Example file: integMomentNormPT.py

```
# - integMomentNorm (pyTree)-
import Converter.PyTree as C
import Generator.PyTree as G
import Post.PyTree as P

m = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
m = C.initVars(m, 'Density',1.)
res = P.integMomentNorm(m, var='Density',center=(5.,5.,0.)); print res
```