Generator 2.5

Stephanie Peron, Sam Landier, Christophe Benoit, Gaelle Jeanfaivre,
Pascal Raud, Luis Bernardos
- Onera -

1 Generator: Mesh generation module

1.1 Preamble

This module can be used to generate meshes from geometries. A mesh can be stored as an array (as defined in the Converter documentation) or in a zone node of a CGNS/python tree (pyTree).

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

When using the array interface, import the Generator module:

```
import Generator as G
```

Then, a is an array and A is a list of arrays.

When using the pyTree interface, import the module:

```
import Generator.PyTree as G
```

Then, a is a zone node and A is a list of zone nodes or a pyTree.

1.2 Basic grid generation

G.cart: create a structured Cartesian mesh with ni x nj x nk points starting from point (xo,yo,zo) and of step (hi,hj,hk):

```
a = G.cart((xo,yo,zo), (hi,hj,hk), (ni,nj,nk))
```

(See: cart.py) (See: cartPT.py)

G.cartHexa: create an unstructured hexahedral mesh defined from a Cartesian grid of ni x nj x nk points starting from point (xo,yo,zo) and of step (hi,hj,hk). Type of elements are 'QUAD' for 2D arrays and 'HEXA' for 3D arrays:

```
a = G.cartHexa((xo,yo,zo), (hi,hj,hk), (ni,nj,nk))
```

(See: cartHexa.py) (See: cartHexaPT.py)

G.cartTetra: create an unstructured tetrahedral mesh defined from a Cartesian grid of ni x nj x nk points starting from point (xo,yo,zo) and of step (hi,hj,hk). Type of elements are 'TRI' for 2D arrays and 'TETRA' for 3D arrays:

```
a = G.cartTetra((xo,yo,zo), (hi,hj,hk), (ni,nj,nk))
```

(See: cartTetra.py) (See: cartTetraPT.py)

G.cartPenta: create an unstructured prismatic mesh defined from a regular Cartesian mesh. The initial Cartesian mesh is defined by ni x nj x nk points starting from point (xo,yo,zo) and of step (hi,hj,hk). Type of elements is 'PENTA':

```
a = G.cartPenta((xo,yo,zo), (hi,hj,hk), (ni,nj,nk))
```

(See: cartPenta.py) (See: cartPentaPT.py)

G.cartPyra: create an unstructured pyramidal mesh defined from a regular Cartesian mesh. The initial Cartesian mesh is defined by ni x nj x nk points starting from point (xo,yo,zo) and of step (hi,hj,hk). Type of elements is 'PYRA':

```
a = G.cartPyra((xo,yo,zo), (hi,hj,hk), (ni,nj,nk))
```

(See: cartPyra.py) (See: cartPyraPT.py)

G.cartNGon: create a NGON mesh defined from a regular Cartesian mesh. The initial Cartesian mesh is defined by ni x nj x nk points starting from point (xo,yo,zo) and of step (hi,hj,hk). Type of elements is 'NGON':

```
a = G.cartNGon((xo,yo,zo), (hi,hj,hk), (ni,nj,nk))
```

(See: cartNGon.py) (See: cartNGonPT.py)

G.cylinder: create a regular cylindrical grid (or a portion of cylinder between tetas and tetae) with ni x nj x nk points, of center-bottom point (xo,yo,zo), of inner radius R1, outer radius R2 and height H. For a direct mesh, use tetae ¡ tetas:

```
a = G.cylinder((xo,yo,zo), R1, R2, tetas, tetae, H, (ni,nj,nk))
```

(See: cylinder.py) (See: cylinderPT.py)

G.cylinder2: create an irregular cylindrical grid (or a portion of cylinder between tetas and tetae) with ni x nj x nk points, of center-bottom point (xo,yo,zo), of inner radius R1, outer radius R2, height H and with distributions in r, teta, z. Distributions are arrays defining 1D meshes (x and i varying) giving a distribution in [0,1]. Their number of points gives ni, nj, nk:

```
a = G.cylinder2((xo,yo,zo), R1, R2, tetas, tetae, H, arrayR, arrayTeta, arrayZ)
```

(See: cylinder2.py) (See: cylinder2PT.py)

G.cylinder3: create an irregular cylindrical grid (or a portion of cylinder between tetas and tetae) from a xz plane mesh defined by a and a teta distribution defined by arrayTeta:

```
b = G.cylinder3(a, tetas, tetae, arrayTeta)
```

(See: cylinder3.py) (See: cylinder3PT.py)

1.3 General purpose grid generators

G.delaunay: create a 2D Delaunay type mesh from an array. The array can be a 2D structured array, or an unstructured array of type 'NODE', 'TRI' or 'QUAD'. Tol is a geometric tolerance. Points nearer than tol are merged. If keepBB is set to 1, the bounding box is kept in the final triangulation:

```
b = G.delaunay(a, tol=1.e-10, keepBB=0)
```

(See: delaunay.py) (See: delaunayPT.py)

G.constrainedDelaunay: create a constrained Delaunay triangulation of the convex hull of a contour c. Contour must be a BAR-array and must be in the plane (x,y). Tol is a geometric tolerance. Points nearer than tol are merged. If keepBB is set to 1, the bounding box is kept in the final triangulation:

```
b = G.constrainedDelaunay(c, tol=1.e-10, keepBB=0)
```

(See: constrainedDelaunay.py) (See: constrainedDelaunayPT.py)

G.checkDelaunay: check if the Delaunay triangulation defined in tri is inside the contour c:

```
c2 = G.checkDelaunay(c, tri)
```

(See: checkDelaunay.py) (See: checkDelaunayPT.py)

G.T3mesher2D: create a 2D Delaunay mesh given a BAR defined in a. If triangulateOnly=1 then only points of a are triangulated, if triangulateOnly=0, then interior points are inserted:

```
b = G.T3mesher2D(a, triangulateOnly=0)
```

(See: T3mesher2D.py) (See: T3mesher2DPT.py)

G.TetraMesher: create a 3D tetra mesh given a TRI surface defined in a. If the TRI surface has external normals, tetras are filled inside the surface. If algo=0, netgen is used, if algo=1, tetgen is used:

```
b = G.tetraMesher(a, algo=1)
```

(See: tetraMesher.py) (See: tetraMesherPT.py)

G.TFI: generate a mesh by transfinite interpolation (TFI). Generated mesh can be 2D or 3D structured, or unstructured TRI or PENTA mesh. Warning: the boundaries can be in a different order from the examples below, except for the PENTA TFI meshes.
2D structured mesh is built from imin, imax, jmin, jmax boundaries.
3D structured mesh is built from imin, imax, jmin, jmax, kmin, kmax boundaries.
Dimensions must be equal for each pair (imin,imax), (jmin,jmax)...
TRI mesh is built from imin, jmin, diag boundaries. Each boundary is a structured array with the same dimension. PENTA mesh is built from Tmin, Tmax triangles boundary and imin, imax, diag boundaries. Tmin, Tmax must be structured triangles of dimension nxn. imin, jmin, diag must be structured n*p arrays:

```
2D-struct: a = G.TFI([imin, imax, jmin, jmax])
3D-struct: a = G.TFI([imin, imax, jmin, jmax, kmin, kmax])
TRI: a = G.TFI([imin, jmin, diag])
PENTA: a = G.TFI([Tmin, Tmax, imin, imax, diag])
```

(See: TFI.py) (See: TFIPT.py)

G.TFITri: generate three meshes by transfinite interpolation around three given curves a1, a2, a3. N3-N2+N1 must be odd:

```
A = G.TFITri(a1, a2, a3)
```

(See: TFITri.py) (See: TFITriPT.py)

G.TFIO: generate five meshes by transfinite interpolation around one given curves a. The number of points of a must be odd:

```
A = G.TFIO(a)
```

(See: TFIO.py) (See: TFIOPT.py)

G.TFIHalfO: generate four meshes by transfinite interpolation around two given curves a1 and a2 forming a half-O. N1 and N2 must be odd:

```
A = G.TFIHalfO(a)
```

(See: TFIHalfO.py) (See: TFIHalfOPT.py)

G.TFIMono: generate one mesh by transfinite interpolation around two given curves a1 and a2 forming a half-O. N1-N2 must be even:

```
A = G.TFIMono(a)
```

(See: TFIMono.py) (See: TFIMonoPT.py)

G.hyper2D: generate an hyperbolic mesh (2D) of ”C” or ”O” type from a from a line defined by line a and from a distribution defined by distrib. The resulting mesh is nearly orthogonal:

```
b = G.hyper2D(line, distrib, ”C”)
```

(See: hyper2d.py) (See: hyper2dPT.py)

G.PolyLine.polyLineMesher: generate a 2D mesh around a 2D polyline:

```
B = G.PolyLine.polyLineMesher(a, h, hf, density)
```

where a is the input polyline (BAR-array), h is the height of the mesh, hf is the height of the first cell and density is the number of points per unity of length.
In the ’array’ version, it returns a list where B[0] is the list of generated meshes, B[1] is the list of wall boundaries, B[2] is the list of overlap boundaries, B[3] is h, B[4] is density (eventually modified by the mesher).
In the pyTree version, it returns a list [zones,hs,densities], where zones is a list of zones of a CGNS python tree, containing the blocks, wall boundaries, match and overlap boundaries; hs is the list of heights (modified if necessary), and densities the list of densities (also modified if necessary).
(See: polyLineMesher.py) (See: polyLineMesherPT.py)

G.PolyC1.polyC1Mesher: generate a 2D mesh around a 2D polyC1 curve:

```
B = G.PolyC1.polyC1Mesher(A, h, hf, density, splitCrit=10.)
```

where A is a list of i-arrays each representing a C1 curve. All i-arrays put together must represent a polyC1 curve. SplitCrit is a curvature radius triggering split. Other arguments are similar to polyLineMesher. The function return is also similar to polyLineMesher.
(See: polyC1Mesher.py) (See: polyC1MesherPT.py)

G.pointedHat: create a structured mesh from a curve defined by a i-array and a point. For the pyTree version: if a contains a solution, it is not taken into account in b:

```
b = G.pointedHat(a, (x,y,z))
```

(See: pointedHat.py) (See: pointedHatPT.py)

G.stitchedHat: create a stitched mesh from a curve defined by a i-array. The surface is stitched in the middle. Tol is the accuracy of the search, tol2 is a merging tolerance and offx, offy, off z an optional offset. For the pyTree version: if a contains a solution, it is not taken into account in b:

```
b = G.stitchedHat(a, (offx,offy,offz), tol=1.e-6, tol2=1.e-5)
```

(See: stitchedHat.py)

G.surfaceWalk: surface extrusion starting from a curve, resulting into a surface mesh. dj is the distribution of points in the extrusion direction starting from c, niter the number of smoothing iterations. check=1 means that the extrusion stops at the layer before cells intersect alphaRef is the deviation angle wrt 180 degrees enabling to stop the extrusion before it crosses a sharp edge on the surface. toldist is a tolerance below which points are considered matching. Constraints can be set as 1D zones:

```
walk = G.surfaceWalk(surfaces, c, dj, constraints=[], niter=0, alphaRef=180., check=0,
toldist=1.e-6)
```

(See: surfaceWalk.py) (See: surfaceWalkPT.py)

G.collarMesh*: create a collar mesh at junction(s) between two surfaces s1 and s2 in union or difference assembly, using a distribution along the surface dj and a distribution in the normal direction to the wall dk. niterj and niterk are the number of smoothing iterations for j and k directions. ext is the extension of the collar mesh for difference assembly. type is the assembly type, and can be 'union' or 'difference'. alphaRef is the deviation angle wrt 180 degrees above which the walk is stopped. contour is the starting contour to create the collar grids, constraints1 and constraints2 are 1D zones defining the curves the collar grid must follow on surfaces s1 and s2 respectively. toldist is the matching point tolerance. Parameter 'topology' can be 'overset' or 'extruded', only useful in case of difference. Topology set to 'overset' results in two overlapping collar grids, whereas it results in a collar grid extruded from the surface grid in the other case:

```
A = G.collarMesh(s1, s2, dj,dk, niterj=100, niterk=100, ext=5, alphaRef=30., type='union', con-
tour=[], constraints1=[], constraints2=[], toldist=1.e-10, topology='overset')
```

(See: collarMesh.py) (See: collarMeshPT.py)

1.4 Cartesian grid generators

G.gencartmb: simple Cartesian generator. Create a set of Cartesian grids (B) around a list of body grids (A). Those grids are patched with a ratio of 2. The user controls the number of levels, and the number of points for each level of grid. h is the spatial step on the finest level. Dfar is the maximal distance to the body. nlvl is a list that provides the number of points per level (nlvl[0]: finest grid), except for the finest level:

```
B = G.gencartmb(A, h, Dfar, nlvl)
```

(See: gencartmb.py) (See: gencartmbPT.py)

G.octree: create a QUAD quadtree mesh in 2D or an HEXA octree mesh in 3D starting from a list of bodies and snears. Each parameter snear is the required spatial step of the octree near the corresponding body; dfar is the extension of the octree mesh in all the directions; balancing=1 means that the octree is balanced, i.e. adjacent elements are at worst twice as big/small; levelMax is the maximum number of levels required. If ratio=2, then a classical octree mesh is built. If ratio=3, a 27-tree mesh is built, in which case the spacing ratio is 3 (and not 2) between two adjacent elements:

```
b = G.octree(surfs, snears, dfar=5., balancing=0, levelMax=1000, ratio=2)
```

(See: octree.py) (See: octreePT.py)

G.octree2Struct: convert an octree or a quadtree mesh into a set of Cartesian grids. Parameter ext is the extension of Cartesian grids in all the directions; vmin can be an integer defining the number of points in each Cartesian grid, or a list of integers, defining the number of points per refinement level. In that case, the first element of the list of vmin defines the finest level. Specifying all the levels is not mandatory. If optimized=1, the ext value is reduced by -1 at overlap borders for the coarsest grid for minimum overlapping. If merged=1, Cartesian grids are merged in order to reduce the number of created grids. If AMR=1, a set of AMR zones are generated. Parameter sizeMax can be used when merging is applied: in that case, the number of points per grid does not exceed sizeMax. Warning: to obtain multigrid blocks, optimized must be set to 0:

```
b = G.octree2Struct(octree, vmin=15, ext=0, optimized=1, merged=1, AMR=0, sizeMax=1000000)
```

(See: octree2Struct.py) (See: octree2StructPT.py)

G.adaptOctree: adapt an unstructured octree with respect to an indicator field located at element centers. If 'indicator' is strictly positive for an element, then the element must be refined as many times as required by the indicator number. If 'indicator' is strictly negative, the element is coarsened if possible as many times as required by the indicator number. If 'indicator' is 0., the element remains unchanged. balancing=1 means that the octree is balanced after adaptation. If ratio=2, then a classical octree mesh is built. If ratio=3, a 27-tree mesh is built, in which case the spacing ratio is 3 (and not 2) between two adjacent elements. For array interface indicator is an array, for pyTree version, indicator is the name of field stored as a solution located at centers:

```
res = G.adaptOctree(octree, indicator, balancing=1, ratio=2)
```

(See: adaptOctree.py) (See: adaptOctreePT.py)

G.expandLayer: Expand the layer of given level for an octree unstructured mesh. If corners=1, expand also in corners directions:

```
res = G.expandLayer(octree, level=0, corners=0, balancing=0)
```

(See: expandLayer.py) (See: expandLayerPT.py)

1.5 Operations on meshes

G.close: close a mesh defined by array a. Points that are distant of tol maximum to one another are merged:

```
b = G.close(a, tol=1.e-12) .or. B = G.close(A, tol=1.e-12)
```

(See: close.py) (See: closePT.py)

G.selectInsideElts: select elements of a TRI-array, whose centers are inside the given list of curves, defined by BAR-arrays:

```
b = G.selectInsideElts(a, curves)
```

(See: selectInsideElts.py)

G.map: map a distribution on a curve or on a structured surface:

```
b = G.map(a, distrib)
```

Map a i-array distribution in a direction (dir=1,2,3) in a surface or volume mesh:

```
b = G.map(a, distrib, dir)
```

(See: map.py) (See: mapPT.py)

G.mapSplit: split a i-array and map a distribution on the splitted i-array. SplitCrit is the curvature radius triggering split:

```
b = G.mapSplit(a, distrib, splitCrit=100)
```

(See: mapSplit.py) (See: mapSplitPT.py)

G.refine: refine a structured array. The original distribution is kept but the number of points is multiplied by power. Dir is the direction of refinement (1, 2, 3). If dir=0, refine in all directions:

```
b = G.refine(a, power, dir)
```

(See: refine.py) (See: refinePT.py)

G.mapCurvature: map a structured array following the curvature. N is the final number of points. Dir is the direction of remeshing (1, 2, 3):

```
b = G.mapCurvature(a, N, power, dir)
```

(See: mapCurvature.py) (See: mapCurvaturePT.py)

G.densify: densify a i-array or a BAR-array with a new discretization step h. Discretization points from the original array are kept:

```
b = G.densify(a, h)
```

(See: densify.py) (See: densifyPT.py)

G.grow: grow a surface array of one layer. Vector is the node displacement. For the array version, vector is defined by an array. For the PyTree version, vector = ['v1','v2','v3'] where variables 'v1', 'v2', 'v3' are defined as solutions in a, located at nodes:

```
b = G.grow(a, vector)
```

(See: grow.py) (See: growPT.py)

G.stack: stack two structured meshes (with the same nixnj) into a single mesh:

```
c = G.stack(a, b)
```

(See: stack.py) (See: stackPT.py)

G.addNormalLayers: normal extrusion from a surface mesh. d is a 1D distribution providing the height of each layer. If check=1, the extrusion stops before negative volume cells are created. Niter specifies the number of iterations for normals smoothing:

```
b = G.addNormalLayers(a, d, check=0, niter=0)
```

(See: addNormalLayers.py) (See: addNormalLayersPT.py)

G.TTM: smooth a mesh using elliptic generator:

```
b = G.TTM(a, niter=100)
```

(See: TTM.py) (See: TTMPT.py)

G.snapFront: snap a mesh to a surface S. A front must be defined in a by a cellN field. Points of this front are snapped to the surface. If optimized=0, the exterior front cellN=1 is snapped, else if optimized=1 optimized front cellN=1 is snapped, else if optimized=2, front cellN=0 is snapped:

```
b = G.snapFront(a, S, optimized=1)
```

(See: snapFront.py) (See: snapFrontPT.py)

G.snapSharpEdges: snap a mesh to a surface S, constrained by sharp edges and corners. if step != None, sharp edges are refined with this step. Sharp Edges are calculated depending on angle:

```
b = G.snapSharpEdges(a, S, step=None, angle=30.)
```

(See: snapSharpEdges.py) (See: snapSharpEdgesPT.py)

1.6 Operations on surface meshes

G.fittingPlaster: fit a surface structured patch to a curve a. BumpFactor controls the curvature of the patch:

```
b = G.fittingPlaster(a, bumpFactor=0.)
```

(See: fittingPlaster.py) (See: fittingPlasterPT.py)

G.gapfixer: fill a gap defined by a BAR contour a drawn on a surface c. You can force the generated mesh to pass through HardPoints (NODES). If refine=0, no inside points are added:

```
b = G.gapfixer(a, c, hardPoints=None, refine=1)
```

(See: gapfixer.py) (See: gapfixerPT.py)

G.gapsmanager: fill multiple gaps in a set of surface components A. Also, eliminate overlap regions between components if any. Normals for all patches must be pointed outwards. Set mode=0 for nodal mesh, 1 for center mesh, and 2 otherwise. Set coplanar=1 if all components are lying on a same plane:

```
B = G.gapsmanager(A, mode=0, coplanar=0)
```

(See: gapsmanager.py) (See: gapsmanagerPT.py)

1.7 Information on generated meshes

G.check: check regularity, orthogonality for a mesh defined by an array:

```
G.check(a)
```

(See: check.py) (See: checkPT.py)

G.barycenter: return the barycenter of a, with optional weight:

```
bary = G.barycenter(a, weight=None) .or. bary = G.barycenter(A, weight=None)
```

(See: barycenter.py) (See: barycenterPT.py)

G.bbox: return the bounding box [xmin, ymin, zmin, xmax, ymax, zmax] of a or A:

```
bb = G.bbox(a) .or. bb = G.bbox(A)
```

(See: bbox.py) (See: bboxPT.py)

G.bboxOfCells: return the bounding box of each cell of a. The bounding box field is located at centers of cells:

```
bb = G.bboxOfCells(a) .or. BB = G.bboxOfCells(A)
```

(See: bboxOfCells.py) (See: bboxOfCellsPT.py)

G.BB: return the bounding box of a as an array or a zone. If method is 'AABB', then it computes the Axis-Aligned Bounding-Box, if method is 'OBB' then it computes the Oriented Bounding-Box. The argument weighting may be 0, and the OBB is computed using a Cloud-Point approach, or 1, and it is computed using a Surface-Weighting approach. If weighting=1, then the provided array must be a surface composed of triangles:

```
b = G.BB(a, method='AABB', weighting=0) .or. B = G.BB(A, method='AABB', weighting=0)
```

(See: BB.py) (See: BBPT.py)

G.CEBBIntersection: test the Cartesian Elements Bounding Box (CEBB) intersection between a1 and a2. Tolerance is a float given by tol. Return 0 if no intersection, 1 otherwise:

```
intersect = G.CEBBIntersection(a1, a2, tol=1.e-10)
```

(See: CEBBIntersection.py) (See: CEBBIntersectionPT.py)

G.bboxIntersection: test if a1 and a2 intersects. Three options are available: method='AABB' (intersection between two Axis-Aligned Bounding Boxes, by default); method='OBB' (intersection between two Oriented Bounding Boxes, the most general case); method='AABBOBB' (intersection between an AABB -a1- and an OBB -a2-).; If a1 and a2 are directly the corresponding bounding boxes, the user may switch isBB=True in order to avoid recalculating them. Return 0 if no intersection, 1 otherwise:

```
intersect = G.bboxIntersection(a1, a2, tol=1.e-6, isBB=False, method='AABB')
```

(See: bboxIntersection.py) (See: bboxIntersectionPT.py)

G.checkPointInCEBB: test if a given point is in the CEBB of a:

```
inside = G.checkPointInCEBB(a, (x,y,z))
```

(See: checkPointInCEBB.py) (See: checkPointInCEBBPT.py)

G.getVolumeMap: return the volume field of an array. Volume is located at centers of cells:

```
b = G.getVolumeMap(a) .or. B = G.getVolumeMap(A)
```

(See: getVolumeMap.py) (See: getVolumeMapPT.py)

G.getNormalMap: return the surface normals field of a surface array. It is located at centers of cells:

```
b = G.getNormalMap(a) .or. B = G.getNormalMap(A)
```

(See: getNormalMap.py) (See: getNormalMapPT.py)

G.getSmoothNormalMap: return the smoothed surface normals field of a surface array, located at nodes. niter is the number of smoothing operations, and eps is a smoothing weight :

```
b = G.getSmoothNormalMap(a, niter=2,eps=0.4) .or. B = G.getNormalMap(A, niter=2,eps=0.4)
```

(See: getSmoothNormalMap.py) (See: getSmoothNormalMapPT.py)

G.getOrthogonalityMap: return the orthogonality map of an array. The orthogonality map corresponds to the maximum deviation of all dihedral angles of an element. The orthogonality map is expressed in degree and located at centers:

```
b = G.getOrthogonalityMap(a) .or. B = G.getOrthogonalityMap(A)
```

(See: getOrthogonalityMap.py) (See: getOrthogonalityMapPT.py)

G.getRegularityMap: return the regularity map of an array. The regularity map corresponds to the maximum deviation of the volume ratio of an element and all its neigbouring cells. The regularity map is located at centers:

```
b = G.getRegularityMap(a) .or. B = G.getRegularityMap(A)
```

(See: getRegularityMap.py) (See: getRegularityMapPT.py)

G.getTriQualityMap: return the quality map of a TRI array. The triangle quality is a value between 0. (degenerated triangle) and 1. (equilateral triangle). The quality map is located at centers:

```
b = G.getTriQualityMap(a) .or. B = G.getTriQualityMap(A)
```

(See: getTriQualityMap.py) (See: getTriQualityMapPT.py)

G.getCellPlanarity: return a measure of cell planarity for each cell. It is located at centers of cells:

```
b = G.getCellPlanarity(a) .or. B = G.getCellPlanarity(A)
```

(See: getCellPlanarity.py) (See: getCellPlanarityPT.py)

G.getCircumCircleMap: return the map of circum circle radius of any cell of a 'TRI' array:

```
b = G.getCircumCircleMap(a) .or. B = G.getCircumCircleMap(A)
```

(See: getCircumCircleMap.py) (See: getCircumCircleMapPT.py)

G.getInCircleMap: return the map of inscribed circle radius of any cell of a 'TRI' array:

```
b = G.getInCircleMap(a) .or. B = G.getInCircleMap(A)
```

(See: getInCircleMap.py) (See: getInCircleMapPT.py)

G.getEdgeRatio: return the ratio between the longest and the smallest edges of a cell:

```
r = G.getEdgeRatio(a) .or. B = G.getEdgeRatio(A)
```

(See: getEdgeRatio.py) (See: getEdgeRatioPT.py)

G.getMaxEdgeLength: return the length of the longer edge of each cell:

```
r = G.getMaxLength(a) .or. B = G.getMaxLength(A)
```

(See: getMaxLength.py) (See: getMaxLengthPT.py)

1.8 Operations on distributions

Distributions are Cartesian meshes that can be used to be mapped on profiles to make curvilinear meshes for instance. Distributions in 2D (x,y) distribution represent the length and height of each cell. Distributions in 3D (x,y,z) represents the lengths in the three topological directions of each cell. Distributions can be modified by the enforce functions.
The three following functions are also available in Y and Z directions: replace suffix 'X' by 'Y' or 'Z' in enforceX, enforcePlusX or enforceMoinsX functions.

G.enforceX: enforce a region around a line x=x0. The size of the cell around the line is enforcedh. "supp" points are suppressed from the starting distribution on the left and right side. "add" points are added on the left and add points are added on the right. Add exactely add points:

```
b = G.enforceX(a, x0, enforcedh, (supp,add))
```

Adjust add in order to have a monotonic distribution:

```
b = G.enforceX(a, x0, enforcedh, supp, add)
```

(See: enforceX.py) (See: enforceXPT.py) (See: enforce.py) (See: enforcePT.py)

G.enforceMoinsX: same as before but with a one sided distribution (left). This can be usefull to create a boundary layer distribution in an Euler mesh.

```
b = G.enforceMoinsX(a, enforcedh, (supp,add)) .or. b = G.enforceMoinsX(a, enforcedh, supp,
add)
```

(See: enforceMoinsX.py) (See: enforceMoinsXPT.py)

G.enforcePlusX: same as before but with a one sided distribution (right):

b = G.enforcePlusX(a, enforcedh, (supp,add)) *.or.* b = G.enforcePlusX(a, enforcedh, supp, add)

(See: enforcePlusX.py) (See: enforcePlusXPT.py)

G.enforceLine: enforce a curvilinear line defined by the array line in a distribution defined by the array a:

b = G.enforceLine(a, line, enforcedh, (supp,add))

(See: enforceLine.py) (See: enforceLinePT.py)

G.enforcePoint: enforce a point in the distribution. The index of enforced point is returned:

ind = G.enforcePoint(a, x0)

(See: enforcePoint.py) (See: enforcePointPT.py)

G.enforceCurvature: enforce the curvature of an i-curve in a distribution defined by a. Power reflects the power of stretching:

b = G.enforceCurvature(a, curve, power=0.5)

(See: enforceCurvature.py) (See: enforceCurvaturePT.py)

G.addPointInDistribution: add a point in a distribution at index ind:

b = G.addPointInDistribution(a, ind)

(See: addPointInDistribution.py) (See: addPointInDistributionPT.py)

1.9 More general examples of use

- See: Examples/Generator/naca.py

1.10 Example files

Example file: cart.py

```
# - cart (array) -
import Converter as C
import Generator as G
a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
C.convertArrays2File(a, 'out.plt')
```

Example file: cartPT.py

```
# - cart (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,11,12))
C.convertPyTree2File(a, 'out.cgns')
```

Example file: cartHexa.py

```
# - cartHexa (array) -
import Generator as G
import Converter as C

a = G.cartHexa((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
C.convertArrays2File([a], 'out.plt')
```

Example file: cartHexaPT.py

```
# - cartHexa (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cartHexa((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
C.convertPyTree2File(a, 'out.cgns')
```

Example file: cartTetra.py

```
# - cartTetra (array) -
import Generator as G
import Converter as C
a = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
C.convertArrays2File(a, 'out.plt')
```

Example file: cartTetraPT.py

```
# - cartTetra (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cartTetra((0.,0.,0.), (0.1,0.1,0.2), (10,10,1))
C.convertPyTree2File(a, 'out.cgns')
```

Example file: cartPenta.py

```
# - cartPenta (array) -
import Generator as G
import Converter as C
a = G.cartPenta((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
C.convertArrays2File([a], "out.plt")
```

Example file: cartPentaPT.py

```
# - cartPenta (pyTree)-
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cartPenta((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
C.convertPyTree2File(a, 'out.cgns')
```

Example file: cartPyra.py

```
# - cartHexa (array) -
import Generator as G
import Converter as C

a = G.cartPyra((0.,0.,0.), (1,1,1), (20,20,20))
C.convertArrays2File(a, 'out.tp')
```

Example file: cartPyraPT.py

```
# - cartPyra (pyTree)-
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cartPyra((0.,0.,0.), (0.1,0.1,0.2), (10,10,10))
C.convertPyTree2File(a, 'out.cgns')
```

Example file: cartNGon.py

```
# - cartNGon (array) -
import Generator as G
import Converter as C
import CPlot

a = G.cartNGon((0.,0.,0.), (0.1,0.1,0.2), (20,20,20))
CPlot.display([a])
```

Example file: cartNGonPT.py

```
# - cartNGon (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cartNGon((0.,0.,0.), (0.1,0.1,0.2), (2,2,2))
C.convertPyTree2File(a, 'out.cgns')
```

Example file: cylinder.py

```
# - cylinder (array) -
import Generator as G
import Converter as C
a = G.cylinder((0.,0.,0.), 0.5, 1., 360., 0., 10., (50,50,30))
C.convertArrays2File([a], "out.plt")
```

Example file: cylinderPT.py

```
# - cylinder (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
a = G.cylinder((0.,0.,0.), 0.5, 1., 360., 0., 10., (50,50,30))
C.convertPyTree2File(a, 'out.cgns')
```

Example file: cylinder2.py

```
# - cylinder2 (array) -
import Converter as C
import Generator as G
r = G.cart((0.,0.,0.), (0.1, 1., 1.), (11, 1, 1))
teta = G.cart((0.,0.,0.), (0.1, 1., 1.), (11, 1, 1))
z = G.cart((0.,0.,0.), (0.1, 1., 1.), (11, 1, 1))
cyl = G.cylinder2( (0.,0.,0.), 0.5, 1., 360., 0., 10., r, teta, z)
C.convertArrays2File([cyl], "out.plt")
```

Example file: cylinder2PT.py

```
# - cylinder2 (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

r = G.cart((0.,0.,0.), (0.1, 1., 1.), (11, 1, 1))
teta = G.cart((0.,0.,0.), (0.1, 1., 1.), (11, 1, 1))
z = G.cart((0.,0.,0.), (0.1, 1., 1.), (11, 1, 1))

cyl = G.cylinder2( (0.,0.,0.), 0.5, 1., 360., 0., 10., r, teta, z)
C.convertPyTree2File(cyl, 'out.cgns')
```

Example file: cylinder3.py

```
# - cylinder3 (array) -
import Generator as G
import Converter as C
teta = G.cart((0.,0.,0.), (0.1, 1., 1.), (11, 1, 1))
xz = G.cart((0.1,0.,0.), (0.1,1.,0.2), (20, 1, 30))
cyl = G.cylinder3( xz, 0., 90., teta)
C.convertArrays2File([cyl], 'out.plt')
```

Example file: cylinder3PT.py

```
# - cylinder3 (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
teta = G.cart((0.,0.,0.), (0.1, 1., 1.), (11, 1, 1))
xz = G.cart((0.1,0.,0.), (0.1,1.,0.2), (20, 1, 30))
cyl = G.cylinder3(xz, 0., 90., teta)
C.convertPyTree2File(cyl, 'out.cgns')
```

Example file: delaunay.py

```
# - delaunay (array) -
import Generator as G
import Converter as C
ni = 11; nj = 11; nk = 1
hi = 1./(ni-1); hj = 1./(nj-1); hk = 1.
a = G.cart((0.,0.,0.), (hi,hj,hk), (ni,nj,nk))
b = G.delaunay(a)
C.convertArrays2File([a,b], "out.plt")
```

Example file: delaunayPT.py

```
# - delaunay (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

ni = 11; nj = 11; nk = 1
hi = 1./(ni-1); hj = 1./(nj-1); hk = 1.
a = G.cart((0.,0.,0.), (hi,hj,hk), (ni,nj,nk))
b = G.delaunay(a); b[0] = 'delaunay'
t = C.newPyTree(['Base', 2, a,b])
C.convertPyTree2File(t, 'out.cgns')
```

Example file: constrainedDelaunay.py

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

A = D.text1D('STEPHANIE')
A = C.convertArray2Tetra(A); a = T.join(A)
# Triangulation respecting given contour
tri = G.constrainedDelaunay(a)
C.convertArrays2File([a,tri], "out.plt")
```

Example file: constrainedDelaunayPT.py

```
# - constrainedDelaunay (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D
import Transform.PyTree as T

A = D.text1D('STEPHANIE')
A = C.convertArray2Tetra(A); a = T.join(A)
# Triangulation respecting given contour
tri = G.constrainedDelaunay(a)
C.convertPyTree2File(tri, 'out.cgns')
```

Example file: checkDelaunay.py

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

A = D.text1D('STEPHANIE')
A = C.convertArray2Tetra(A); a = T.join(A)
# Triangulation respecting given contour
tri = G.constrainedDelaunay(a)
res = G.checkDelaunay(a, tri)
C.convertArrays2File([res], "out.plt")
```

Example file: checkDelaunayPT.py

```
# - checkDelaunay (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D
import Transform.PyTree as T

A = D.text1D('STEPHANIE')
A = C.convertArray2Tetra(A); a = T.join(A)
# Triangulation respecting given contour
tri = G.constrainedDelaunay(a)
res = G.checkDelaunay(a, tri)
C.convertPyTree2File(res, "out.cgns")
```

Example file: T3mesher2D.py

```
# - T3mesher2D (array) -
import Generator as G
import Converter as C
import Geom as D

a = D.circle( (0,0,0), 1, N=50 )
a = C.convertArray2Tetra(a)
a = G.close(a)
b = G.T3mesher2D(a, triangulateOnly=0)
C.convertArrays2File([a,b], 'out.plt')
```

Example file: T3mesher2DPT.py

```
# - T3mesher2D (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D
```

```
a = D.circle( (0,0,0), 1, N=50 )
a = C.convertArray2Tetra(a); a = G.close(a)
b = G.T3mesher2D(a, triangulateOnly=0)
t = C.newPyTree(['Base',2]); t[2][1][2].append(b)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: tetraMesher.py

```
# - tetraMesher (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), (3,3,3) )
ext = P.exteriorFaces(a)
ext = C.convertArray2Tetra(ext)
ext = G.close(ext)
ext = T.reorder(ext, (-1,))
m = G.tetraMesher(ext, algo=1)
C.convertArrays2File([m], 'out.plt')
```

Example file: tetraMesherPT.py

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

a = G.cart((0,0,0), (1,1,1), (3,3,3))
ext = P.exteriorFaces(a)
ext = C.convertArray2Tetra(ext)
ext = G.close(ext)
ext = T.reorder(ext, (-1,))
m = G.tetraMesher(ext, algo=1)
C.convertPyTree2File(m, 'out.cgns')
```

Example file: TFI.py

```
# - TFI 2D structured (array)
# - TFI 3D structured (array)
# - TFI TRI (array)
import Converter as C
import Generator as G
import Geom as D
import numpy as N
import Transform as T

#------------------
# TFI 2D structured
#------------------
P0 = (0,0,0); P1 = (5,0,0); P2 = (0,7,0); P3 = (5,7,0)

# Geometrie
d1 = D.line(P0, P1); d2 = D.line(P2, P3)

pts = C.array('x,y,z',5,1,1)
x = pts[1][0]; y = pts[1][1]; z = pts[1][2]
x[0] = 0.; y[0] = 0.; z[0] = 0.
x[1] =-2.; y[1] = 2.; z[1] = 0.
```

```
x[2] =-3.; y[2] = 3.; z[2] = 0.
x[3] = 2.; y[3] = 5.; z[3] = 0.
x[4] = 0.; y[4] = 7.; z[4] = 0.
b1 = D.bezier(pts)

pts = C.array('x,y,z',5,1,1)
x = pts[1][0]; y = pts[1][1]; z = pts[1][2]
x[0] = 5.; y[0] = 0.; z[0] = 0.
x[1] = 3.; y[1] = 2.; z[1] = 0.
x[2] = 2.; y[2] = 3.; z[2] = 0.
x[3] = 6.; y[3] = 5.; z[3] = 0.
x[4] = 5.; y[4] = 7.; z[4] = 0.
b2 = D.bezier( pts )

C.convertArrays2File([d1, d2, b1, b2], "geom.plt")

# Regular discretisation of each line
Ni = 20
Nj = 10
r = G.cart((0,0,0), (1./(Ni-1),1,1), (Ni,1,1))
q = G.cart((0,0,0), (1./(Nj-1),1,1), (Nj,1,1))
r1 = G.map(d1, r)
r2 = G.map(d2, r)
r3 = G.map(b1, q)
r4 = G.map(b2, q)

# TFI 2D
m = G.TFI([r1, r2, r3, r4])
C.convertArrays2File([r1,r2,r3,r4,m], 'tfi2d.plt')

#-------------------
# TFI 3D structured
#-------------------
xo = 0.; yo = 0.; zo = 0.
nx = 21; ny = 21; nz = 21
hx = 1./(nx-1); hy = 1./(ny-1); hz = 1./(nz-1)

# z = cste
fzmin = G.cart((xo,yo,zo), (hx,hy,1.), (nx,ny,1))
fzmax = T.translate(fzmin, (0.,0.,1.))

# x = cste
fxmin = G.cart((xo,yo,zo),(1,hy,hz),(1,ny,nz))
fxmin = T.reorder(fxmin,(3,1,2))
fxmax = T.translate(fxmin, (1.,0.,0.))

# y = cste
fymin = G.cart((xo,yo,zo),(hx,1.,hz),(nx,1,nz))
fymin = T.reorder(fymin,(1,3,2))
fymax = T.translate(fymin, (0.,1.,0.))

r = [fxmin,fxmax,fymin,fymax,fzmin,fzmax]
m = G.TFI(r)
C.convertArrays2File(r+[m], 'tfi3d.plt')

#---------
# TFI TRI
#---------
l1 = D.line((0,0,0),(0,1,0), 15)
l2 = D.line((0,0,0),(1,0,0), 15)
l3 = D.line((1,0,0),(0,1,0), 15)
```

```
tri = G.TFI([l1,l2,l3])
C.convertArrays2File([tri], 'tfitri.plt')
```

Example file: TFIPT.py

```
# - TFI (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Geom.PyTree as D
import numpy as N

# Geometry
P0 = (0,0,0); P1 = (5,0,0); P2 = (0,7,0); P3 = (5,7,0)
Ni = 20; Nj = 10
d1 = D.line(P0, P1,Ni); d2 = D.line(P2, P3,Ni)
d3 = D.line(P0, P2,Nj); d4 = D.line(P1, P3,Nj)
m = G.TFI([d1, d2, d3, d4])
C.convertPyTree2File(m, 'out.cgns')
```

Example file: TFITri.py

```
# - TFITri (array)
import Converter as C
import Generator as G
import Geom as D

P0 = (0,0,0); P1 = (5,0,0); P2 = (1,7,0)

# 3 curves (dont need to be lines)
d1 = D.line(P0, P1, N=11)
d2 = D.line(P1, P2, N=11)
d3 = D.line(P0, P2, N=11)
r = G.TFITri(d1, d2, d3)
C.convertArrays2File(r, 'out.plt')
```

Example file: TFITriPT.py

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

P0 = (0,0,0); P1 = (5,0,0); P2 = (1,7,0)

# 3 curves (dont need to be lines)
d1 = D.line(P0, P1, N=11)
d2 = D.line(P1, P2, N=11)
d3 = D.line(P0, P2, N=11)
r = G.TFITri(d1, d2, d3)
C.convertPyTree2File(r, 'out.cgns')
```

Example file: TFIO.py

```
# - TFIO (array) -
import Converter as C
import Generator as G
import Geom as D

a = D.circle((0,0,0), 1., N=41)
r = G.TFIO(a)
C.convertArrays2File(r, 'out.plt')
```

Example file: TFIOPT.py

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

a = D.circle((0,0,0), 1., N=41)
r = G.TFIO(a)
C.convertPyTree2File(r, 'out.cgns')
```

Example file: TFIHalfO.py

```
# - TFIHalfO (array) -
import Converter as C
import Generator as G
import Geom as D

a1 = D.circle((0,0,0), 1., tetas=0, tetae=180., N=41)
a2 = D.line((-1,0,0),(1,0,0), N=21)
r = G.TFIHalfO(a1, a2)
C.convertArrays2File(r, 'out.plt')
```

Example file: TFIHalfOPT.py

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

a1 = D.circle((0,0,0), 1., tetas=0, tetae=180., N=41)
a2 = D.line((-1,0,0),(1,0,0), N=21)
r = G.TFIHalfO(a1, a2)
C.convertPyTree2File(r, 'out.cgns')
```

Example file: TFIMono.py

```
# - TFIMono (array) -
import Converter as C
import Generator as G
import Geom as D

a1 = D.circle((0,0,0), 1., tetas=0, tetae=180., N=41)
a2 = D.line((-1,0,0),(1,0,0), N=21)
r = G.TFIMono(a1, a2)
C.convertArrays2File(r, 'out.plt')
```

Example file: TFIMonoPT.py

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

a1 = D.circle((0,0,0), 1., tetas=0, tetae=180., N=41)
a2 = D.line((-1,0,0),(1,0,0), N=21)
r = G.TFIMono(a1, a2)
C.convertPyTree2File(r, 'out.cgns')
```

Example file: hyper2d.py

```
# - hyper2D (array) -
import Geom as D
import Generator as G
import Converter as C

msh = D.naca(12., 5001)

# Distribution
Ni = 300; Nj = 50
distrib = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,1))
a = G.hyper2D(msh, distrib, "C")
C.convertArrays2File([a], 'out.plt')
```

Example file: hyper2dPT.py

```
# - hyper2D (pyTree) -
import Geom.PyTree as D
import Generator.PyTree as G
import Converter.PyTree as C

line = D.naca(12., 5001)
# Distribution
Ni = 300; Nj = 50
distrib = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,1))

a = G.hyper2D(line, distrib, "C")
C.convertPyTree2File(a, 'out.cgns')
```

Example file: polyLineMesher.py

```
# - polyLineMesher (array) -
import Converter as C
import Generator.PolyLine as GP
import Generator as G
import Post as P
import Geom as D
import Transform as T

# Read a 2D geometry created with tecplot
a = C.convertFile2Arrays('fusee.plt')
a = G.close(a,1e-2); a = T.reorder(a,(-1,2,3))
C.convertArrays2File(a, 'input.plt')

# Data
h = 0.02; hf = 0.0001; density = 500

# Per families
coords = []; walls = []
for i in a:
    b = GP.polyLineMesher(i, h, hf, density)
    coords.append(b[0])
    walls.append(b[1])

# Flat
meshes = []
for i in coords: meshes = meshes + i

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

Example file: polyLineMesherPT.py

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

tb = C.convertFile2PyTree('fusee.plt')
tb = G.close(tb,1e-2); tb = T.reorder(tb,(-1,2,3))
h = 0.02; hf = 0.0001; density = 500

res = G.polyLineMesher(tb[2][1][2][0], h, hf, density)
zones = res[0]; h = res[1]; density = res[2]
t = C.newPyTree(['PolyC1']); t[2][1][2] += zones
C.convertPyTree2File(t, 'out.cgns')
```

Example file: polyC1Mesher.py

```
# - polyC1Mesher (array) -
import Converter as C
import Generator.PolyC1 as GP
import Generator as G
import Post as P
import Geom as D
import Transform as T

# Read geometry from svg file
a = C.convertFile2Arrays('Data/curve.svg', density=1)[0]
a = T.homothety(a,(0,0,0),0.01)
a = T.reorder(a, (1,2,3))

h = 0.2; hp = 0.001; density = 10.; splitCrit = 2.
m = GP.polyC1Mesher(a, h, hp, density, splitCrit)

for i in m[0]:
    v = G.getVolumeMap(i)
    min = C.getMinValue(v, 'vol')
    if (min <= 0):
        print 'negative volume detected.'

C.convertArrays2File(m[0], 'out.plt')
```

Example file: polyC1MesherPT.py

```
# - polyC1Mesher (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Transform.PyTree as T

tb = C.convertFile2PyTree('curve1.svg', nptsCurve=100, nptsLine=400)
z = T.homothety(tb[2][1][2][0], (0.,0.,0.), 0.01)
z = T.reorder(z, (-1,2,3))
h = 0.1; hf = 0.001; density = 100; splitCrit = 10.
res = G.polyC1Mesher(z, h, hf, density, splitCrit)
zones = res[0]; h = res[1]; density = res[2]
t = C.newPyTree(['Base']); t[2][1][2] += zones
C.convertPyTree2File(t, 'out.cgns')
```

Example file: pointedHat.py

```
# - pointedHat (array) -
import Geom as D
import Generator as G
import Converter as C
c = D.circle( (0,0,0), 1., 360., 0., 100)
surf = G.pointedHat(c,(0.,0.,1.))
C.convertArrays2File([surf], 'out.plt')
```

Example file: pointedHatPT.py

```
# - pointedHat (pyTree) -
import Geom.PyTree as D
import Generator.PyTree as G
import Converter.PyTree as C
c = D.circle( (0,0,0), 1., 360., 0., 100)
surf = G.pointedHat(c,(0.,0.,1.))
t = C.newPyTree(['Base']); t[2][1][2].append(surf)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: stitchedHat.py

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

c = D.circle( (0,0,0), 1., 360., 0., 100)
c = T.contract(c, (0,0,0), (0,1,0), (0,0,1), 0.1)
c = G.stitchedHat(c, (0,0,0), 1.e-3)
C.convertArrays2File([c], 'out.plt')
```

Example file: surfaceWalk.py

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

# User definition of parametric curve
def f(t,u):
    x = t+u; y = t*t+1+u*u; z = u
    return (x,y,z)

# Array definition of geometry
a = D.surface(f)

c = D.circle((1.2,1.7,0.6), 0.1,N=100)
c = T.rotate(c, (1.2,1.7,0.6), (0,1,0), 90.)
c = T.reorder(c,(-1,2,3))
c = T.projectOrtho(c,[a])

h = G.cart((0.,0.,0.),(0.01,1,1),(30,1,1))
r = G.surfaceWalk([a], c, h, niter=100)
C.convertArrays2File([a,c,r], "out.plt")
```

Example file: surfaceWalkPT.py

```
# - surfaceWalk (pyTree)
import Converter.PyTree as C
import Geom.PyTree  as D
import Transform.PyTree  as T
import Generator.PyTree  as G
```

```
# User definition of parametric curve
def f(t,u):
    x = t+u; y = t*t+1+u*u; z = u
    return (x,y,z)

# Array definition of geometry
a = D.surface(f)

c = D.circle((1.2,1.7,0.6), 0.1)
c = T.rotate(c, (1.2,1.7,0.6), (0,1,0), 90.)
c = T.reorder(c,(-1,2,3))
c = T.projectOrtho(c,[a])

h = G.cart((0.,0.,0.),(0.01,1,1),(15,1,1))
r = G.surfaceWalk([a], c, h, niter=100)
C.convertPyTree2File(r, "out.cgns")
```

Example file: collarMesh.py

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

s1 = D.sphere((0.,0.,0.),1,20)
s2 = T.translate(s1,(1.2,0.,0.)); s2 = T.homothety(s2,(0,0,0),0.5)
dhj = G.cart((0.,0.,0.),(1.e-2,1,1),(21,1,1))
dhk = G.cart((0.,0.,0.),(1.e-2,1,1),(11,1,1))
a = G.collarMesh(s1,s2, dhj,dhk,niterj=100,niterk=100,type='union')
C.convertArrays2File(a,"out.plt")
```

Example file: collarMeshPT.py

```
# - collarMesh (pyTree)
import Converter.PyTree as C
import Geom.PyTree as D
import Transform.PyTree as T
import Generator.PyTree as G

s1 = D.sphere((0.,0.,0.),1,20)
s2 = T.translate(s1,(1.2,0.,0.)); s2 = T.homothety(s2,(0,0,0),0.5)
dhj = G.cart((0.,0.,0.),(1.e-2,1,1),(21,1,1))
dhk = G.cart((0.,0.,0.),(1.e-2,1,1),(11,1,1))
a = G.collarMesh(s1,s2, dhj,dhk,niterj=100,niterk=100,type='union')
C.convertPyTree2File(a,"out.cgns")
```

Example file: gencartmb.py

```
# - gencartmb (array) -
import Generator as G
import Converter as C

# body grid
a = G.cylinder((0.,0.,0.), 0.5, 1., 360., 0., 10., (50,50,30))
h = 1.e-1# Step of finest Cartesian grid
Dfar = 10.# Extension of far boundaries
nlvl = [5,5,5] # Nb of points per level, except the 4th level (automatic)
cartGrids = G.gencartmb([a], h, Dfar, nlvl)
C.convertArrays2File(cartGrids, 'out.plt')
```

Example file: gencartmbPT.py

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

# body mesh
a = G.cylinder((0.,0.,0.), 0.5, 1., 360., 0., 10., (50,50,30))
h = 1.e-1  # Step of finest Cartesian grid
Dfar = 20. # Distance to far boundaries

# Nb of points per level:
# Here 4 levels, but last one is computed automatically
nlvl = [10,10,5] # nlvl[0]: coarse grid

t = C.newPyTree(['Bodies', 'CARTESIAN']); t[2][1][2].append(a)
zones = G.gencartmb(t[2][1], h, Dfar, nlvl)
t[2][2][2] += zones
C.convertPyTree2File(t, 'out.cgns')
```

Example file: octree.py

```
# - octree (array) -
import Generator as G
import Converter as C
import Geom as D

s = D.circle((0,0,0), 1., N=100); snear = 0.01
res = G.octree([s], [snear], dfar=5.,balancing=1)
C.convertArrays2File([res], 'out.plt')
```

Example file: octreePT.py

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

s = D.circle((0,0,0), 1., N=100); snear = 0.1
res = G.octree([s], [snear], dfar=5.)
C.convertPyTree2File(res, 'out.cgns')
```

Example file: octree2Struct.py

```
# - octree2Struct (array) -
import Generator as G
import Converter as C
import Geom as D

s = D.circle((0,0,0), 1., N=100); snear = 0.1
res = G.octree([s],[snear], dfar=5., balancing=1)
res = G.octree2Struct(res, vmin=5, ext=2, optimized=1)
C.convertArrays2File([s]+res, "out.plt")
```

Example file: octree2StructPT.py

```
# - octree2Struct (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D

s = D.circle((0,0,0), 1., N=100); snear = 0.1
res = G.octree([s],[snear], dfar=5., balancing=1)
res = G.octree2Struct(res, vmin=5, ext=2,merged=1)
C.convertPyTree2File(res, 'out.cgns')
```

Example file: adaptOctree.py

```
# - adaptOctree (array) -
import Generator as G
import Converter as C
import Geom as D

s = D.circle((0,0,0), 1., N=100); snear = 0.1
o = G.octree([s], [snear], dfar=5., balancing=1)
indic = C.node2Center(o)
indic = C.initVars(indic, 'indicator', 1.)
res = G.adaptOctree(o, indic)
C.convertArrays2File([o,res], "out.plt")
```

Example file: adaptOctreePT.py

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

s = D.circle((0,0,0), 1., N=100); snear = 0.1
o = G.octree([s], [snear], dfar=5.,balancing=1)
o = C.initVars(o, 'centers:indicator', 1.)
res = G.adaptOctree(o)
C.convertPyTree2File(res, 'out.cgns')
```

Example file: expandLayer.py

```
# - expandLayer (array) -
import Generator as G
import Converter as C
import Geom as D

s = D.circle((0.,0.,0.), 1., N=100)
o = G.octree([s], [0.1], dfar=1., balancing=1)
o2 = G.expandLayer(o, level=0)
C.convertArrays2File([o, o2], "out.plt")
```

Example file: expandLayerPT.py

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

s = D.circle((0.,0.,0.),1.,N=100)
o = G.octree([s], [0.1], dfar=1., balancing=1)
o2 = G.expandLayer(o, level=0)
C.convertPyTree2File(o2, 'out.cgns')
```

Example file: close.py

```
# - close (array) -
import Converter as C
import Generator as G
import Transform as T

a = G.cylinder((0.,0.,0.), 0.5, 1., 360., 0.01, 10., (20,20,10))
a = C.convertArray2Tetra(a)
a = G.close(a, 1.e-3)
C.convertArrays2File(a, 'out.plt')
```

Example file: closePT.py

```
# - close (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a1 = G.cart((0,0,0), (1,1,1), (10,10,1))
a2 = G.cart((9+1.e-2,0,0), (1,1,1), (10,10,1))
a3 = G.cart((0,-5.01,0),(1,1,1),(19,6,1))
a4 = G.cart((0,9.0001,0),(1,1,1),(10,6,1))
a5 = G.cart((9.01,9.0002,0),(1,1,1),(10,6,1))
t = C.newPyTree(['Base',2,a1,a2,a3,a4,a5])
t = G.close(t, 1.e-1)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: selectInsideElts.py

```
# - selectInsideElts (array) -
import Converter as C
import Generator as G
import Geom as D

a = G.cart( (0,0,0), (1,1,1), (10,10,1)); a = C.convertArray2Tetra(a)
b = D.circle( (5,5,0), 3.); b = C.convertArray2Tetra(b)
a = G.selectInsideElts(a, [b])
C.convertArrays2File([a,b], 'out.plt')
```

Example file: map.py

```
# - map (array) -
import Geom as D
import Generator as G
import Converter as C

# Map on a curve
l = D.line( (0,0,0), (1,1,0) )
Ni = 10
d = G.cart( (0,0,0), (1./(Ni-1),1.,1.), (Ni,1,1) )
m = G.map(l, d)
C.convertArrays2File([m], "out1.plt")

# Map on a structured surface
ni = 2; nj = 3
a = G.cart((0,0,0), (1,1,1), (ni,nj,1))
C.setValue(a, (1,1,1), [1.,1.,2.])
C.setValue(a, (1,2,1), [1.,2.,5.])
C.setValue(a, (1,3,1), [1.,3.,2.])
C.setValue(a, (2,1,1), [2.,1.,2.])
C.setValue(a, (2,2,1), [2.,2.,5.])
C.setValue(a, (2,3,1), [2.,3.,2.])
b = D.bezier(a, 10, 10)
Ni = 50; Nj = 30
d = G.cart( (0,0,0), (1./(Ni-1),1./(Nj-1),1.), (Ni,Nj,1) )
d = G.enforceX(d, 0.5, 0.01, (10,20))
d = G.enforceY(d, 0.5, 0.01, (10,20))
b = G.map(b, d)
C.convertArrays2File([b], "out2.plt")

# Map in a direction
a = G.cylinder((0,0,0), 0.5, 2., 0, 60, 1., (20,20,1))
Ni = 10
```

```
d = G.cart( (0,0,0), (1./(Ni-1),1.,1.), (Ni,1,1) )
d = G.enforcePlusX(d, 0.01, (10,20))
a = G.map(a, d, 2)
C.convertArrays2File([a], "out3.plt")
```

Example file: mapPT.py

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

l = D.line( (0,0,0), (1,1,0) )
Ni = 11; dist = G.cart( (0,0,0), (1./(Ni-1),1.,1.), (Ni,1,1) )
l = G.map(l, dist)
t = C.newPyTree(['Base',1]); t[2][1][2].append(l)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: mapSplit.py

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

# polyline
a = D.polyline([(0,0,0),(1,0,0),(1,1,0),(2,3,0),(1.5,3,0),(1,1.5,0),(0,0,0)])
# distribution
Ni = 41
dist = G.cart((0,0,0),(1./(Ni-1),1,1),(Ni,1,1))
dist = G.enforceX(dist, 15.5/(Ni-1), 0.005, 2,5)
dist = G.enforceX(dist, 27.5/(Ni-1), 0.005, 2,5)
a = G.mapSplit(a,dist,0.25)
C.convertArrays2File(a, 'out.plt')
```

Example file: mapSplitPT.py

```
# - mapSplit (pyTree) -
import Generator.PyTree as G
import Transform.PyTree as T
import Converter.PyTree as C
import Geom.PyTree as D

# polyline
a = D.polyline([(0,0,0),(1,0,0),(1,1,0),(2,3,0),(1.5,3,0),(1,1.5,0),(0,0,0)])
# distribution
Ni = 41
dist = G.cart((0,0,0),(1./(Ni-1),1,1),(Ni,1,1))
dist = G.enforceX(dist, 15.5/(Ni-1), 0.005, 2,5)
dist = G.enforceX(dist, 27.5/(Ni-1), 0.005, 2,5)
zones = G.mapSplit(a,dist,0.25)
t = C.newPyTree(['Base',1]); t[2][1][2] += zones
C.convertPyTree2File(t, 'out.cgns')
```

Example file: refine.py

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

```
a = G.cart( (0,0,0), (0.1,0.1,0.1), (20,20,1) )

a = G.refine(a, 1.5, 1)
C.convertArrays2File([a], 'out.plt')
```

Example file: refinePT.py

```
# - refine (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cart( (0,0,0), (0.1,0.1,0.1), (20,20,1) )
a = G.refine(a, 1.5, 1)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: mapCurvature.py

```
# - mapCurvature (array) -
import Generator as G
import Converter as C
import Geom as D

ni = 2; nj = 3
a = G.cart((0,0,0), (1,1,1), (ni,nj,1))
C.setValue(a, (1,1,1), [1.,1.,2.])
C.setValue(a, (1,2,1), [1.,2.,4.])
C.setValue(a, (1,3,1), [1.,3.,2.])
C.setValue(a, (2,1,1), [2.,1.,2.])
C.setValue(a, (2,2,1), [2.,2.,5.])
C.setValue(a, (2,3,1), [2.,3.,2.])
b = D.bezier(a, density=10.)

b = G.mapCurvature(b, N=100, power=0.5, dir=1)
C.convertArrays2File([b], 'out.plt')
```

Example file: mapCurvaturePT.py

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

ni = 2; nj = 3
a = G.cart((0,0,0), (1,1,1), (ni,nj,1))
C.setValue(a,'GridCoordinates', (1,1,1), [1.,1.,2.])
C.setValue(a,'GridCoordinates', (1,2,1), [1.,2.,5.])
C.setValue(a,'GridCoordinates', (1,3,1), [1.,3.,2.])
C.setValue(a,'GridCoordinates', (2,1,1), [2.,1.,2.])
C.setValue(a,'GridCoordinates', (2,2,1), [2.,2.,5.])
C.setValue(a,'GridCoordinates', (2,3,1), [2.,3.,2.])
b = D.bezier(a, density=10.)

b = G.mapCurvature(b, N=100, power=0.5, dir=1)

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

Example file: densify.py

```
# - densify (array) -
import Generator as G
import Converter as C
import Geom as D
```

```
a = D.circle( (0,0,0), 1., 10 )
b = G.densify(a, 0.01)
C.convertArrays2File([b], 'out.plt')
```

Example file: densifyPT.py

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

a = D.circle((0,0,0), 1., 10)
b = G.densify(a, 0.01)
C.convertPyTree2File(b, 'out.cgns')
```

Example file: grow.py

```
# - grow (array) -
import Converter as C
import Generator as G
import Geom as D

a = D.sphere( (0,0,0), 1., 50 )
n = G.getNormalMap(a)
n = C.center2Node(n); n[1] = n[1]*100.
b = G.grow(a, n)
C.convertArrays2File([b], 'out.plt')
```

Example file: growPT.py

```
# - grow (pyTree)-
import Converter.PyTree as C
import Converter.Internal as Internal
import Generator.PyTree as G
import Geom.PyTree as D

a = D.sphere((0,0,0), 1., 50)
a = G.getNormalMap(a)
a = C.center2Node(a, Internal.__FlowSolutionCenters__)
a = C.rmVars(a, Internal.__FlowSolutionCenters__)
b = G.grow(a, ['sx','sy','sz'])
t = C.newPyTree(['Base1',2,'Base2',3])
t[2][1][2].append(a); t[2][2][2].append(b)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: stack.py

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

# Concatenate 2 structured grids
a = G.cylinder((0,0,0), 1, 1.3, 360, 0, 1., (50,10,1))
b = T.rotate(a, (0,0,0), (1,0,0), 5.)
b = T.translate(b, (0,0,0.5))
c = G.stack(a, b)

# Conacatenate n structured grids
a = []
```

```
for i in xrange(10):
    a.append(D.circle((0,0,i), 1.))
c = G.stack(a)

C.convertArrays2File([c], 'out.plt')
```

Example file: stackPT.py

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

a = G.cylinder((0,0,0), 1, 1.3, 360, 0, 1., (50,10,1))
b = T.rotate(a, (0,0,0), (1,0,0), 5.)
b = T.translate(b, (0,0,0.5))
c = G.stack(a, b)
C.convertPyTree2File(c, 'out.cgns')
```

Example file: addNormalLayers.py

```
# - addNormalLayers (array) -
import Generator as G
import Converter as C
import Geom as D

d = C.array('d', 3, 1, 1)
d[1][0,0] = 0.1; d[1][0,1] = 0.2; d[1][0,2] = 0.3
a = D.sphere( (0,0,0), 1, 50 )
a = G.addNormalLayers(a, d)
C.convertArrays2File([a], 'out.plt')
```

Example file: addNormalLayersPT.py

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

d = G.cart((0.1,0.,0.), (0.1,1,1),(2,1,1))
a = D.sphere((0,0,0), 1, 50)
a = D.line((0,0,0),(1,1,0),3)
b = G.addNormalLayers(a, d)
d = G.cart((0.1,0.,0.), (-0.1,1,1),(2,1,1))
c =  G.addNormalLayers(a, d)
import Transform.PyTree as T
a = T.join(b,c)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: TTM.py

```
# - TTM (array) -
import Converter as C
import Generator as G
import Geom as D

P0 = (0,0,0); P1 = (5,0,0); P2 = (0,7,0); P3 = (5,7,0)

# Geometry
d1 = D.line(P0, P1); d2 = D.line(P2, P3)
pts = C.array('x,y,z', 5, 1, 1)
x = pts[1][0]; y = pts[1][1]; z = pts[1][2]
```

```
x[0] = 0. ; y[0] = 0.; z[0] = 0.
x[1] =-2. ; y[ 1 ] = 2.; z[1] = 0.
x[2] =-3. ; y[ 2 ] = 3.; z[2] = 0.
x[3] = 2. ; y[ 3 ] = 5.; z[3] = 0.
x[4] = 0. ; y[ 4 ] = 7.; z[4] = 0.
b1 = D.bezier(pts)

x[0] = 5.; y[ 0 ] = 0.; z[ 0 ] = 0.
x[1] = 3.; y[ 1 ] = 2.; z[ 1 ] = 0.
x[2] = 2.; y[ 2 ] = 3.; z[ 2 ] = 0.
x[3] = 6.; y[ 3 ] = 5.; z[ 3 ] = 0.
x[4] = 5.; y[ 4 ] = 7.; z[ 4 ] = 0.
b2 = D.bezier( pts )
C.convertArrays2File([d1, d2, b1, b2], 'geom.plt')

# Regular discretision of each line
Ni = 20; Nj = 10
r = G.cart((0,0,0), (1./(Ni-1),1,1), (Ni,1,1))
q = G.cart((0,0,0), (1./(Nj-1),1,1), (Nj,1,1))
r1 = G.map(d1, r)
r2 = G.map(d2, r)
r3 = G.map(b1, q)
r4 = G.map(b2, q)

# TTM
m = G.TFI([r1, r2, r3, r4])
m2 = G.TTM(m, 2000)
C.convertArrays2File([m,m2], 'out.plt')
```

Example file: TTMPT.py

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

P0 = (0,0,0); P1 = (5,0,0); P2 = (0,7,0); P3 = (5,7,0)

# Geometry
d1 = D.line(P0,P1); d2 = D.line(P2,P3)
d3 = D.line(P0,P2); d4 = D.line(P1,P3)

# Regular discretisation of each line
Ni = 20; Nj = 10
r = G.cart((0,0,0), (1./(Ni-1),1,1), (Ni,1,1))
q = G.cart((0,0,0), (1./(Nj-1),1,1), (Nj,1,1))
r1 = G.map(d1, r); r2 = G.map(d2, r)
r3 = G.map(d3, q); r4 = G.map(d4, q)

# TTM
m = G.TFI([r1, r2, r3, r4])
m = G.TTM(m)
t = C.newPyTree(['Base',2]); t[2][1][2].append(m)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: snapFront.py

```
# - snapFront (array) -
import Generator as G
import Converter as C
import Geom as D
import Connector as X
import Transform as T

s = D.circle((0,0,0), 1., N=100)
s = T.addkplane(s)

# Grille cartesienne (reguliere)
BB = G.bbox([s])
ni = 100; nj = 100; nk = 3
xmin = BB[0]; ymin = BB[1]; zmin = BB[2]-0.5
xmax = BB[3]; ymax = BB[4]; zmax = BB[5]+0.5
hi = (xmax-xmin)/(ni-1); hj = (ymax-ymin)/(nj-1)
h = min(hi, hj)
ni = int((xmax-xmin)/h)+7; nj = int((ymax-ymin)/h)+7
b = G.cart((xmin-3*h, ymin-3*h, zmin), (h, h, 1.), (ni,nj,nk))
celln = C.array('cellN', ni, nj, nk)
celln = C.initVars(celln, 'cellN', 1.)

# Masquage
cellno = X.blankCells([b], [celln], [s], blankingType=0, delta=0., dim=2)
a = C.initVars(s, 'cellN', 1)
b = C.addVars([b, cellno[0]])

# Adapte le front de la grille a la surface
b = T.subzone(b, (1,1,2), (b[2],b[3],2))
b = G.snapFront(b, [s])

C.convertArrays2File([a,b], 'out.plt')
```

Example file: snapFrontPT.py

```
# - snapFront (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D
import Connector.PyTree as X
import Transform.PyTree as T
import Converter.Internal as Internal

s = D.circle((0,0,0), 1., N=100)
s2 = T.addkplane(s)

# Grille cartesienne (reguliere)
BB = G.bbox([s])
ni = 100; nj = 100; nk = 3
xmin = BB[0]; ymin = BB[1]; zmin = BB[2]-0.5
xmax = BB[3]; ymax = BB[4]; zmax = BB[5]+0.5
hi = (xmax-xmin)/(ni-1); hj = (ymax-ymin)/(nj-1)
h = min(hi, hj)
ni = int((xmax-xmin)/h)+7; nj = int((ymax-ymin)/h)+7
b = G.cart( (xmin-3*h, ymin-3*h, zmin), (h, h, 1.), (ni,nj,nk) )
t = C.newPyTree(['Cart'])
t[2][1][2].append(b)

# Masquage
t = C.initVars(t,'cellN',1)
import numpy
BM = numpy.array([[1]])
t = X.blankCells(t,[[s2]],BM,blankingType='node_in',dim=2)
```

```
# Adapte le front de la grille a la surface
dim = Internal.getZoneDim(b)
t = T.subzone(t, (1,1,2), (dim[1],dim[2],2))
t = G.snapFront(t, [s2])

t = C.addBase2PyTree(t, 'Surface', cellDim=2)
s2 = C.initVars(s2,'cellN',1)
t[2][2][2].append(s2)

C.convertPyTree2File(t, 'out.cgns')
```

Example file: snapSharpEdges.py

```
# - snapSharpEdges (array) -
import Generator as G
import Converter as C
import Geom as D

# polylines with sharp angles
s = D.polyline([(0.2,0,0),(1,1,0),(2.5,1,0),(0.2,0,0)])
# Regular cartesian grid
h = 0.1
ni = 30; nj = 20; nk=1
b = G.cart((-0.5, -0.5, 0), (h, h, 1.), (ni,nj,nk))
b = G.snapSharpEdges(b, [s], h)
C.convertArrays2File([b,s], 'out.plt')
```

Example file: snapSharpEdgesPT.py

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

# polylignes avec angles vifs
s = D.polyline([(0.02,0,0),(1,1,0),(2,1,0),(0.02,0,0)])
# Grille cartesienne (reguliere)
h = 0.1
ni = 30; nj = 20; nk=1
b = G.cart((-0.5, -0.5, 0), (h, h, 1.), (ni,nj,nk))
b = G.snapSharpEdges(b, [s], h)
t = C.newPyTree(['Cart','Surface'])
t[2][1][2].append(b); t[2][2][2].append(s)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: fittingPlaster.py

```
# - fittingPlaster (array) -
import Generator as G
import Converter as C
import Geom as D

a = D.circle( (0,0,0), 1, N=50 )
a = C.convertArray2Tetra(a)
a = G.close(a)
b = G.fittingPlaster(a, bumpFactor=0.5)
C.convertArrays2File([a,b], 'out.plt')
```

Example file: fittingPlasterPT.py

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

a = D.circle( (0,0,0), 1, N=50 )
a = C.convertArray2Tetra(a)
a = G.close(a)
b = G.fittingPlaster(a, bumpFactor=0.5)
C.convertPyTree2File(b, 'out.cgns')
```

Example file: gapfixer.py

```
# - gapfixer (array) -
import Generator as G
import Converter as C
import Geom as D
import numpy as np

# Fix the gap inside a circle drawn on a plane
a = D.circle( (0,0,0), 1, N=100)
a = C.convertArray2Tetra(a); a = G.close(a)
b = G.cart((-2.,-2.,0.), (0.1,0.1,1.), (50,50,1))
a1 = G.gapfixer(a, b)
C.convertArrays2File([a1], 'out.plt')

# Fill the gap in the circle, using one defined point
hp = D.point((0.5, 0.5, 0.))
a2 = G.gapfixer(a, b, hp, refine=0)
C.convertArrays2File([a2], 'outHP.plt')
```

Example file: gapfixerPT.py

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

# Fix the gap inside a circle drawn on a plane
a = D.circle((0,0,0), 1, N=100)
a = C.convertArray2Tetra(a); a = G.close(a)
b = G.cart((-2.,-2.,0.), (0.1,0.1,1.), (50,50,1))
a1 = G.gapfixer(a, b)
C.convertPyTree2File(a1, 'out.cgns')

# Fill the gap in the circle, using one defined point
hp = D.point((0.5, 0.5, 0.))
a2 = G.gapfixer(a, b, hp, refine=0)
C.convertPyTree2File(a2, 'outHP.cgns')
```

Example file: gapsmanager.py

```
# - gapsmanager (array) -
import Geom as D
import Converter as C
import Generator as G

a = D.sphere6((0,0,0), 1, N=10)
a = C.node2Center(a)
a = C.convertArray2Tetra(a)
b = G.gapsmanager(a, mode=2)
C.convertArrays2File(b, 'out.plt')
```

Example file: gapsmanagerPT.py

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

a = D.sphere6((0,0,0), 1, N=10)
a = C.node2Center(a)
a = C.convertArray2Tetra(a)
b = G.gapsmanager(a, mode=2)
C.convertPyTree2File(b, 'out.cgns')
```

Example file: check.py

```
# - check (array) -
import Generator as G
a = G.cart((0,0,0), (1,1,1), (10,10,10))
G.check(a)
```

Example file: checkPT.py

```
# - check (pyTree) -
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
G.check(a)
```

Example file: barycenter.py

```
# - barycenter (array) -
import Generator as G
import Converter as C
a = G.cart((0.,0.,0.), (0.1,0.1,1.), (20,20,20))
print G.barycenter(a)
w = C.initVars(a, 'weight', 1.); w = C.extractVars(w,['weight'])
print G.barycenter(a, w)
```

Example file: barycenterPT.py

```
# - barycenter (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
a = G.cart((0.,0.,0.), (0.1,0.1,1.), (20,20,20))
print G.barycenter(a)
a = C.initVars(a, 'weight', 1.)
print G.barycenter(a, 'weight')
```

Example file: bbox.py

```
# - bbox (array) -
import Generator as G
a = G.cart((0.,0.,0.),(0.1,0.1,1.),(20,20,20))
b = G.cart((12.,0.,0.),(0.1,0.1,1.),(20,20,20))
print G.bbox(a)
print G.bbox([a, b])
```

Example file: bboxPT.py

```
# - bbox (pyTree) -
import Generator.PyTree as G
a = G.cart((0.,0.,0.),(0.1,0.1,1.),(20,20,20))
print G.bbox(a)
```

Example file: bboxOfCells.py

```
# - bboxOfCells (array) -
import Generator as G
a = G.cart((0.,0.,0.),(0.1,0.1,1.),(20,20,20))
b = G.bboxOfCells(a)
print b
```

Example file: bboxOfCellsPT.py

```
# - bboxOfCells (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
a = G.cart((0.,0.,0.),(0.1,0.1,1.),(20,20,20))
a = G.bboxOfCells(a)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: BB.py

```
# - BB (array) -
import Generator as G
import Converter as C
import Geom as D

s = D.circle((0,0,0), 1., N=100)
a = G.BB(s)
C.convertArrays2File([a,s], 'out.plt')
```

Example file: BBPT.py

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

s = D.circle((0,0,0), 1., N=100)
a = G.BB(s); a[0] = 'bbox'
C.convertPyTree2File([s,a], 'out.cgns')
```

Example file: CEBBIntersection.py

```
# - CEBBIntersection (array) -
import Generator as G
import Converter as C
import Transform as T
ni = 11; nj = 3; nk = 11
a1 = G.cart((0.,0.,0.), (0.1,0.1,0.2),(ni, nj,nk))
a2 = G.cart((1.,0.,0.), (0.1,0.1,0.2),(ni, nj,nk))
a2 = T.rotate(a2, (0,0,0), (0,0,1), 12.)
print G.CEBBIntersection(a1, a2)
```

Example file: CEBBIntersectionPT.py

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

ni = 11; nj = 3; nk = 11
a1 = G.cart((0.,0.,0.), (0.1,0.1,0.2),(ni, nj,nk))
a2 = G.cart((1.,0.,0.), (0.1,0.1,0.2),(ni, nj,nk))
a2 = T.rotate(a2, (0,0,0), (0,0,1), 12.)
print G.CEBBIntersection(a1, a2)
```

Example file: bboxIntersection.py

```
# - bboxIntersection (array) -
import Generator as G
ni = 11; nj = 3; nk = 11
a1 = G.cart((0.,0.,0.), (0.1,0.1,0.2),(ni, nj,nk))
a2 = G.cart((0.5,0.05,0.01), (0.1,0.1,0.2),(ni, nj,nk))
intersect = G.bboxIntersection(a1,a2); print intersect
```

Example file: bboxIntersectionPT.py

```
# - bboxIntersection (pyTree) -
import Generator.PyTree as G
ni = 11; nj = 3; nk = 11
a1 = G.cart((0.,0.,0.), (0.1,0.1,0.2),(ni, nj,nk))
a2 = G.cart((0.5,0.05,0.01), (0.1,0.1,0.2),(ni, nj,nk))
intersect = G.bboxIntersection(a1, a2); print intersect
```

Example file: checkPointInCEBB.py

```
# - checkPointInCEBB (array) -
import Generator as G
import Transform as T
import Converter as C

Ni = 20; Nj = 20
a1 = G.cart((0,0,0),(1./Ni,0.5/Nj,1),(Ni,Nj,2))
a2 = G.cart((-0.1,0,0),(0.5/Ni, 0.5/Nj, 1), (Ni,Nj,2))
a2 = T.rotate(a2, (-0.1,0,0), (0,0,1), 0.22)

# Check if point is in CEBB of mesh2
val = G.checkPointInCEBB(a2, (0.04839, 0.03873, 0.5)); print val
```

Example file: checkPointInCEBBPT.py

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

Ni = 20; Nj = 20
a2 = G.cart((-0.1,0,0),(0.5/Ni, 0.5/Nj, 1), (Ni,Nj,2))
a2 = T.rotate(a2, (-0.1,0,0), (0,0,1), 0.22)

# Check if point is in CEBB of a2
val = G.checkPointInCEBB(a2, (0.04839, 0.03873, 0.5)); print val
```

Example file: getVolumeMap.py

```
# - getVolumeMap (array) -
import Generator as G
import Converter as C

a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,3))
vol = G.getVolumeMap(a)
vol = C.center2Node(vol); vol = C.addVars([a,  vol])
C.convertArrays2File([vol], "out.plt")
```

Example file: getVolumeMapPT.py

```
# - getVolumeMap (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cart((0.,0.,0.), (0.1,0.1,0.2), (10,10,3))
a = G.getVolumeMap(a)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: getNormalMap.py

```
# - getNormalMap (array) -
import Geom as D
import Generator as G
import Converter as C

# 2D structured
a = D.sphere((0,0,0), 1, 50)
n = G.getNormalMap(a)
n = C.center2Node(n);n = C.addVars([a, n])
C.convertArrays2File([n], 'out1.plt')

# 2D unstructured
a = D.sphere((0,0,0), 1, 50)
a = C.convertArray2Tetra(a)
n = G.getNormalMap(a)
n = C.center2Node(n);n = C.addVars([a, n])
C.convertArrays2File([n], 'out2.plt')
```

Example file: getNormalMapPT.py

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

a = D.sphere((0,0,0), 1, 50)
a = G.getNormalMap(a)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: getSmoothNormalMap.py

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

a = G.cart((0.,0.,0.),(1.,1.,1.),(10,10,1))
b = G.cart((0.,0.,0.),(1.,1.,1.),(1,10,10))
b = T.rotate(b,(0.,0.,0.),(0.,1.,0.),45.)
c = C.convertArray2Hexa([a,b])
```

```
c = T.join(c); c = T.reorder(c,(1,))
c = T.rotate(c,(0.,0.,0.),(0.,1.,0.),15.)
s = G.getSmoothNormalMap(c,niter=4)
c = C.addVars([c,s])
C.convertArrays2File([c],"out.plt")
```

Example file: getSmoothNormalMapPT.py

```
# - getSmoothNormalMap (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Transform.PyTree as T
import Geom.PyTree as D

a = G.cart((0.,0.,0.),(1.,1.,1.),(10,10,1))
b = G.cart((0.,0.,0.),(1.,1.,1.),(1,10,10))
b = T.rotate(b,(0.,0.,0.),(0.,1.,0.),45.)
c = C.convertArray2Hexa([a,b])
c = T.join(c); c = T.reorder(c,(1,))
c = T.rotate(c,(0.,0.,0.),(0.,1.,0.),15.)
c = G.getSmoothNormalMap(c,niter=4)
C.convertPyTree2File(c, "out.cgns")
```

Example file: getOrthogonalityMap.py

```
# - getOrthogonalityMap (array) -
import Generator as G
import Converter as C

a = G.cylinder((0.,0.,0.), 0.5, 1., 360., 0., 10., (50,50,10))
ac = C.node2Center(a)
ortho = G.getOrthogonalityMap(a)
ortho = C.addVars([ac,  ortho])
C.convertArrays2File([ortho], "out.plt")
```

Example file: getOrthogonalityMapPT.py

```
# - getOrthogonalityMap (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cylinder((0.,0.,0.), 0.5, 1., 360., 0., 10., (50,50,10))
a = G.getOrthogonalityMap(a)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: getRegularityMap.py

```
# - getRegularityMap (array) -
import Generator as G
import Converter as C

a = G.cart( (0,0,0), (1,1,1), (50,50,1))
a = G.enforceX(a, 25, 0.1, 10, 10)
ac = C.node2Center(a)
reg = G.getRegularityMap(a)
reg = C.addVars([ac,  reg])
C.convertArrays2File([reg], "out.plt")
```

Example file: getRegularityMapPT.py

```
# - getRegularityMapPT (array) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cart((0,0,0), (1,1,1), (50,50,1))
a = G.enforceX(a, 25, 0.1, 10, 10)
a = G.getRegularityMap(a)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: getTriQualityMap.py

```
# - getTriQualitylityMap (array) -
import Generator as G
import Converter as C
import Geom as D

a = D.sphere( (0,0,0), 1, N=10 )
a = C.convertArray2Tetra(a); a = G.close(a)
n = G.getTriQualityMap(a)
n = C.center2Node(n); n = C.addVars([a, n])
C.convertArrays2File([n], "out.plt")
```

Example file: getTriQualityMapPT.py

```
# - getTriQualitylityMap (PyTree) -
import Generator.PyTree as G
import Converter.PyTree as C
import Geom.PyTree as D

a = D.sphere((0,0,0), 1, N=10)
a = C.convertArray2Tetra(a)
a = G.close(a)
t = C.newPyTree(['Base',2,a])
t = G.getTriQualityMap(t)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: getCellPlanarity.py

```
# - getCellPlanarity (array) -
import Converter as C
import Generator as G
import Geom as D

a = D.sphere( (0,0,0), 1., 10)
p = G.getCellPlanarity(a)
p = C.center2Node(p); a = C.addVars([a, p])
C.convertArrays2File([a], 'out.plt')
```

Example file: getCellPlanarityPT.py

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

a = D.sphere((0,0,0), 1., 10)
a = G.getCellPlanarity(a)
C.convertPyTree2File(a, 'out.cgns')
```

Example file: getCircumCircleMap.py

```
# - getCircumCircleMap (array) -
import Geom as D
import Generator as G
import Converter as C

a = D.sphere((0,0,0), 1, 50)
a = C.convertArray2Tetra(a)
n = G.getCircumCircleMap(a)
n = C.center2Node(n); n = C.addVars([a, n])
C.convertArrays2File([n], "out.plt")
```

Example file: getCircumCircleMapPT.py

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

a = D.sphere((0,0,0), 1, 50)
a = C.convertArray2Tetra(a)
t = C.newPyTree(['Base',2,a])
t = G.getCircumCircleMap(t)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: getInCircleMap.py

```
# - getInCircleMap (array) -
import Geom as D
import Generator as G
import Converter as C

a = D.sphere((0,0,0), 1, 50)
a = C.convertArray2Tetra(a)
n = G.getInCircleMap(a)
n = C.center2Node(n); n = C.addVars([a, n])
C.convertArrays2File([n], "out.plt")
```

Example file: getInCircleMapPT.py

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

a = D.sphere((0,0,0), 1, 50)
a = C.convertArray2Tetra(a)
t = C.newPyTree(['Base',2,a])
t = G.getInCircleMap(t)
C.convertPyTree2File(t, 'out.cgns')
```

Example file: getEdgeRatio.py

```
# - getEdgeRatio(array) -
import Generator as G
import Converter as C

a = G.cart((0.,0.,0.), (0.1,0.1,0.1), (11,11,11))
a = G.enforcePlusX(a,1e-6,(5,50))
r = G.getEdgeRatio(a)
r = C.center2Node(r); r = C.addVars([a,r])
C.convertArrays2File([r], "out.plt")
```

Example file: getEdgeRatioPT.py

```
# - getEdgeRatio(pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cart((0.,0.,0.), (0.1,0.1,0.1), (11,11,11))
a = G.enforcePlusX(a,1e-6,(5,50))
a = G.getEdgeRatio(a)
C.convertPyTree2File(a, "out.cgns")
```

Example file: getMaxLength.py

```
# - getMaxLength(array) -
import Generator as G
import Converter as C

a = G.cart((0.,0.,0.), (0.1,0.1,0.1), (11,11,11))
a = G.enforcePlusX(a,1e-6,(5,50))
r = G.getMaxLength(a)
r = C.center2Node(r); r = C.addVars([a,r])
C.convertArrays2File([r], "out.plt")
```

Example file: getMaxLengthPT.py

```
# - getMaxLength(pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cart((0.,0.,0.), (0.1,0.1,0.1), (11,11,11))
a = G.enforcePlusX(a,1e-6,(5,50))
a = G.getMaxLength(a)
C.convertPyTree2File(a, "out.cgns")
```

Example file: enforceX.py

```
# - enforceX (array) -
import Generator as G
import Converter as C

Ni = 50; Nj = 50
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,1))
# Monotonic distribution
b = G.enforceX(a, 0.3, 0.001, (13,25))
C.convertArrays2File([b], "out.plt")
```

Example file: enforceXPT.py

```
# - enforceX (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

Ni = 50; Nj = 50; Nk = 2
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,Nk))
a = G.enforceX(a, 0.3, 0.001, (13,25))
C.convertPyTree2File(a, 'out.cgns')
```

Example file: enforce.py

```
# - enforceX, enforceY, ... monotonic (array) -
import Converter as C
import Generator as G

a = G.cart( (0,0,0), (1,1,1), (20,20,10) )
b = G.enforceX(a, 5., 0.2, 10, 5 )
b = G.enforceX(b, 15., 0.2, 10, 5 )
b = G.enforceY(b, 10., 0.1, 5, 5 )
b = G.enforcePlusY(b, 0.01, 20, 5 )
C.convertArrays2File([a,b], 'out.plt')
```

Example file: enforcePT.py

```
# - enforceX, enforceY, ... monotonic (pyTree) -
import Generator.PyTree as G
import Converter.PyTree as C

a = G.cart((0,0,0), (1,1,1), (20,20,10))
b = G.enforceX(a, 5., 0.2, 10, 5 )
b = G.enforceX(b, 15., 0.2, 10, 5 )
b = G.enforceY(b, 10., 0.1, 5, 5 )
C.convertPyTree2File(b, 'out.cgns')
```

Example file: enforceMoinsX.py

```
# - enforceMoinsX (array) -
import Generator as G
import Converter as C

Ni = 50; Nj = 50
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,1))
b = G.enforceMoinsX(a, 1.e-3, (10,15))
C.convertArrays2File([b], "out.plt")
```

Example file: enforceMoinsXPT.py

```
# - enforceMoinsX (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

Ni = 50; Nj = 50; Nk = 1
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,Nk))
b = G.enforceMoinsX(a, 1.e-3, (10,15))
C.convertPyTree2File(b, 'out.cgns')
```

Example file: enforcePlusX.py

```
# - enforcePlusX (array) -
import Generator as G
import Converter as C

# Distribution
Ni = 50; Nj = 50
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,1))
b = G.enforcePlusX(a, 1.e-3, (10,20))
C.convertArrays2File([b], "out.plt")
```

Example file: enforcePlusXPT.py

```
# - enforcePlusX (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

# Distribution
Ni = 50; Nj = 50
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,1))
b = G.enforcePlusX(a, 1.e-3, (10,20))
C.convertPyTree2File(b, 'out.cgns')
```

Example file: enforceLine.py

```
# - enforceLine (array) -
import Generator as G
import Converter as C
import Geom as D

Ni = 50; Nj = 50
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,1))
b = D.line((0.,0.2,0.), (1.,0.2,0.), 20)
c = G.enforceLine(a, b, 0.01, (5,3))
C.convertArrays2File([c], 'out.plt')
```

Example file: enforceLinePT.py

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

Ni = 50; Nj = 50
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,1))
b = D.line((0.,0.2,0.), (1.,0.2,0.), 20)
a = G.enforceLine(a, b, 0.01, (5,3))
C.convertPyTree2File(a, 'out.cgns')
```

Example file: enforcePoint.py

```
# - enforcePoint (array) -
import Converter as C
import Generator as G

# distribution
Ni = 20; Nj = 20
a = G.cart((0,0,0), (1./(Ni-1),5./(Nj-1),1), (Ni,Nj,1))
b = G.enforcePoint(a, 0.5)
C.convertArrays2File([b], "out.plt")
```

Example file: enforcePointPT.py

```
# - enforcePoint (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

Ni = 20; Nj = 20
a = G.cart((0,0,0), (1./(Ni-1),5./(Nj-1),1), (Ni,Nj,1))
b = G.enforcePoint(a, 0.5)
C.convertPyTree2File(b, 'out.cgns')
```

Example file: enforceCurvature.py

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

# Naca profile with lines
a = D.naca(12., 501)
l1 = D.getLength(a)
a2 = D.line((1.,0.,0.),(2.,0.,0.), 500)
l2 = D.getLength(a2)
b = T.join(a, a2)
c = D.line((2.,0.,0.),(1.,0.,0.), 500)
res = T.join(c, b)

# Distribution on the profile
l = l1+2*l2
Ni = 100; Nj = 100
p1 = l2/l; p2 = (l2+l1)/l
h = (p2-p1)/(Ni-1)
distrib = G.cart((p1,0,0), (h, 0.25/Nj,1), (Ni,Nj,1))
distrib = G.enforceCurvature(distrib, res, 0.6)
C.convertArrays2File([distrib], "out.plt")
```

Example file: enforceCurvaturePT.py

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

a = D.naca(12., 501)

# Distribution on the profile
Ni = 20; Nj = 20; Nk = 1; h = 1./(Ni-1)
b = G.cart((0,0,0), (h, 0.25/Nj,1), (Ni,Nj,Nk))
b = G.enforceCurvature(b, a, 0.6)
C.convertPyTree2File(b, 'out.cgns')
```

Example file: addPointInDistribution.py

```
# - addPointInDistribution (array) -
import Generator as G
import Converter as C

# Distribution
Ni = 50; Nj = 50
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,1))
b = G.addPointInDistribution( a, Ni )
C.convertArrays2File([b], 'out.plt')
```

Example file: addPointInDistributionPT.py

```
# - addPointInDistribution (pyTree)-
import Generator.PyTree as G
import Converter.PyTree as C

# Distribution
Ni = 50; Nj = 50
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1), (Ni,Nj,2))
b = G.addPointInDistribution(a, Ni)
C.convertPyTree2File(b, 'out.cgns')
```

Example file: naca.py

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

# Put a naca profile in msh
msh = D.naca(12., 5001); l1 = D.getLength(msh)

# Put a line in msh2
msh2 = D.line((1.,0.,0.),(20.,0.,0.), 5001); l2 = D.getLength(msh2)

# Join the two meshes
msh = T.join(msh, msh2)

# Add another line
msh2 = D.line((20.,0.,0.),(1.,0.,0.), 5001); l3 = D.getLength(msh2)
msh = T.join(msh2, msh)

C.convertArrays2File([msh], 'naca.plt')

# distribution
Ni = 200; Nj = 100
distrib = G.cart((0,0,0), (1./(Ni-1), 20./(Nj-1),1), (Ni,Nj,1))

distrib = G.enforcePlusY(distrib, 2.e-3, 50, 30)
distrib = G.enforceMoinsY(distrib, 2., 80, -60)
distrib = G.enforcePlusX(distrib, 1.e-4, 30, 50)
distrib = G.enforceMoinsX(distrib, 0.1, 80, -30)
distrib = G.enforceX(distrib, (l1*0.5)/(0.5*l1+l2), 1.25e-3, 30, 20)

distrib = G.enforcePoint(distrib, (l1*0.5)/(0.5*l1+l2))

distrib2 = T.symetrize(distrib, (0.,0.,0.), (0,1,0), (0,0,1))
distrib2 = T.reorder(distrib2, (-1,2,3))
distrib = T.join(distrib2, distrib)
distrib = T.contract(distrib, (0,0,0), (0,1,0), (0,0,1), 0.5)
distrib = T.translate(distrib, (0.5,0,0))
C.convertArrays2File([distrib], 'distrib.plt')
msh = T.reorder(msh, (1,2,3))

msh = G.hyper2D(msh, distrib, "C")

C.convertArrays2File([msh], 'out.plt')
```