Geom Documentation

Release 3.0

/ELSA/MU-09021/V3.0

Dec 05, 2019

CONTENTS

CHAPTER

ONE

PREAMBLE

In this module, a geometry is defined discretely with a great number of points. The discrete geometry is stored in a Converter array (as defined in Converter documentation) and in a CGNS/Python tree zone or set of zones, depending on the selected interface.

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

To use the module with the Converter array interface:

```
import Geom as D
```

To use the module with the CGNS/Python interface:

```
import Geom.PyTree as D
```

CHAPTER

TWO

LIST OF FUNCTIONS

– Geometry creation

Geom.point(P)	Create a point.
Geom.line(P1, P2[, N])	Create a line of N points.
Geom.polyline(Pts)	Create a polyline of N points.
Geom.circle(C, R[, tetas, tetae, N])	Create a portion of circle of N points and of center C, radius R, between angle tetas and tetae.
Geom.naca(e[, N, sharpte])	Create a NACA00xx profile of N points and thickness e.
Geom.spline(Pts[, order, N, M, density])	Create a spline of N points.
Geom.nurbs(Pts[, weight, order, N, M, density])	Create a nurbs of N points.
Geom.bezier(controlPts[, N, M, density])	Create a a Bezier curve defined by an array of control points controlPts.
Geom.curve(f[, N])	Create a curve from a user defined parametric function or a formula.
Geom.surface(f[, N])	Create a surface from a user defined parametric function or a formula.
Geom.cone(C, Rb, Rv, H[, N])	Create a cone of NxN points and of center C, basis radius Rb, vertex radius Rv and height H.
Geom.torus(C, R, r[, alphas, alphae, betas, ...])	Create a surface mesh of a torus made by NRxNr points.
Geom.sphere(C, R[, N])	Create a sphere of Nx2N points and of center C and radius R.
Geom.sphere6(C, R[, N, ntype])	Create a sphere of 6NxN points and of center C and radius R, made of 6 parts.
Geom.sphereYinYang(C, R[, N, ntype])	Create a sphere of center C and radius R made of two overset parts.

Continued on next page

Table 2.1 – continued from previous page

Geom.disc(C, R[, N, ntype])	Create a disc of center C and radius R made of 5 parts.
Geom.triangle(P0, P1, P2[, N, ntype])	Create a triangle made of 3 parts.
Geom.quadrangle(P1, P2, P3, P4[, N, ntype])	Create a single quadrangle with points P1, P2, P3, P4.
Geom.box(Pmin, Pmax[, N, ntype])	Create a box passing by Pmin and Pmax (axis aligned).
Geom.cylinder(C, R, H[, N, ntype])	Create a cylinder of center C, radius R and height H.

– Typing text using meshes

Geom.text1D(string[, font, smooth, offset])	Create a 1D text.
Geom.text2D(string[, font, smooth, offset])	Create a 2D text.
Geom.text3D(string[, font, smooth, offset, ...])	Create a 3D text.

– Geometry modification

Geom.uniformize(a[, N, h, factor, density, ...])	Unformize the distribution of points on a 1D-mesh.
Geom.refine(a[, N, factor, sharpAngle])	Refine the point distribution on a 1D-mesh.
Geom.enforceh(a[, N, h])	Enforce mesh size in a 1D-mesh.
Geom.lineDrive(a, d)	Generate a surface mesh starting from a curve and a driving curve defined by d.
Geom.orthoDrive(a, d[, mode])	Generate a surface mesh starting from a curve and a driving orthogonally to curve defined by d.
Geom.axisym(a, C, axis[, angle, Ntheta, rmod])	Create an axisymetrical mesh given an azimuthal 1D or 2D mesh.
Geom.connect1D(curves[, sharpness, N, ...])	Connect 1D curves in a single curve.

– Information about geometries

Geom.getLength(a)	Return the length of 1D-mesh.
Geom.getDistantIndex(a, ind, l)	Return the index of 1D-mesh located at a distance l of ind.
Geom.getNearestPointIndex(a, pointList)	Return the nearest index of points in array.
Geom.getCurvatureRadius(a)	Return the curvature radius for each point.
Geom.getCurvatureAngle(a)	Return the curvature angle for each point.

Continued on next page

Table 2.4 – continued from previous page

`Geom.getCurvatureHeight`(a)	Return the curvature height for each node in a 2D or 1D mesh.
`Geom.getSharpestAngle`(a)	Return the sharpest angle for each point of a surface based on the sharpest angle between adjacent element to which the point belongs to.
`Geom.getCurvilinearAbscissa`(a)	Return the curvilinear abscissa for each point.
`Geom.getDistribution`(a)	Return the curvilinear abscissa for each point as coordinates.
`Geom.getTangent`(a)	Return the unit tangent vector of a 1D curve as coordinates.

CHAPTER

THREE

CONTENTS

3.1 Geometry creation

A geometry can be defined by either structured (i-arrays in 1D, (i,j) arrays in 2D) or unstructured (BAR arrays in 1D and QUAD or TRI arrays in 2D) grids.

A polyline is defined as a C0 i-array which contains only the polyline points (with no extra discretization points).

Geom.**point**(*P*)

Create a point of coordinates P=(x,y,z).

Parameters **P** (3-tuple of floats) – (x,y,z) of point

Returns a point

Return type one array/zone (NODE)

Example of use:

- Creation of a point (array):

```
# - point (array) -
import Geom as D
import Converter as C

a = D.point((0,0,0))
C.convertArrays2File(a, "out.plt")
```

- Creation of a point (pyTree):

```
# - point (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C
```

```
a = D.point((0,0,0))
C.convertPyTree2File(a, "out.cgns")
```

Geom.**line**(*P1*, *P2*, *N=100*)

Create a line from point P1 to point P2, uniformly discretized with N points.

Parameters

- **P1** (3-tuple of floats) – (x,y,z) of the starting point
- **P2** (3-tuple of floats) – (x,y,z) of the end point
- **N** (integer) – number of points discretizing the line

Returns a line

Return type one array/zone (1D STRUCT)

Example of use:

- Creation of a line (array):

```
# - line (array) -
import Geom as D
import Converter as C

a = D.line((0,0,0), (1,0,0))
C.convertArrays2File(a, 'out.plt')
```

- Creation of a line (pyTree):

```
# - line (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.line((0,0,0), (1,0,0))
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**polyline**(*Pts*)

Create a polyline made of points Pts=[P1, P2,. . . ,Pn].

Parameters **Pts** (list of 3-tuple of floats) – list of (x,y,z) of points defining the polyline

Returns a polyline

Return type one array/zone (1D STRUCT)

Example of use:

- Creation of a polyline (array):

```
# - polyline (array) -
import Geom as D
import Converter as C

a = D.polyline([(0.,0.,0.),(1.,1.,0.),(2.,0.,0.)])
C.convertArrays2File(a, "out.plt")
```

- Creation of a polyline (pyTree):

```
# - polyline (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.polyline([(0.,0.,0.),(1.,1.,0.),(2.,0.,0.)])
C.convertPyTree2File(a, 'out.cgns')
```

`Geom.`**`circle`**(*C, R, tetas=0., tetae=360., N=100*)

Create a circle or a circle arc of center C and radius R.

Parameters

- **C** (`3-tuple of floats`) – (x,y,z) of circle center
- **R** (`float`) – radius of the circle
- **tetas** (`float`) – initial azimuth of the circle arc
- **tetae** (`float`) – end azimuth of circle arc
- **N** (`integer`) – number of points discretizing the circle arc

Returns a circle

Return type one array/zone (1D STRUCT)

Example of use:

- Creation of a circle (array):

```
# - circle (array) -
import Geom as D
import Converter as C

a = D.circle((0,0,0), 1. , 0., 360.)
C.convertArrays2File(a, "out.plt")
```

- Creation of a circle (pyTree):

```
# - circle (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.circle((0,0,0), 1. , 0., 360.)
C.convertPyTree2File(a, 'out.cgns')
```

`Geom.`**naca**(*e*, *N=101*, *sharpte=True*)

Create a NACA profile. e can be the thickness of the profile (e=15. for NACA0015 for instance) or a string of digit such as "0012" for serie 4, "23012" for serie 5, "0008-45" for modified serie 4 of NACA profiles.

Parameters

- **e** (`float or string`) – thickness of the NACA00xx profile or digit string
- **N** (`integer`) – number of points discretizing the profile
- **sharpte** (boolean) – true if sharp trailing edge

Returns a NACAxx profile

Return type one array/zone (1D STRUCT)

Example of use:

- Creation of a NACA0012 (array):

```
# - naca (array) -
import Geom as D
import Converter as C

# Naca serie 4 defined by height
a = D.naca(12.)

# Naca serie 4 by name
b = D.naca('0012', N=301, sharpte=1)

# Naca serie 5 by name
c = D.naca('23012', N=301, sharpte=1)

# Naca serie 4 modified by name
d = D.naca('0008-45', N=301, sharpte=1)

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

- Creation of a NACA0012 (pyTree):

```
# - naca (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.naca(12.)
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**spline**(*Pts, order=3, N=100, M=100, density=-1*)
Create a Spline curve/surface using control points defined by Pts.

Parameters

- **Pts** (array or zone of control points) – i-mesh (resp. (i,j)-mesh) of control points for a Spline curve (resp. surface)
- **order** (integer) – order of the Spline
- **N** (integer) – number of points in the i-direction in the resulting discretized Spline
- **M** (integer) – number of points in the j-direction in the resulting discretized Spline
- **density** (float) – density of points in the discretized Spline (instead of specifying N and M)

Returns a Spline curve/surface

Return type one array/zone (1D STRUCT or 2D STRUCT)

Example of use:

- Creation of a Spline (array):

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

# Spline 1D
c = D.polyline([(0.,0.,0.), (1.,1.,0.), (2.,1.,0.), \
                (3.,0.,0.), (4.,-1.,0.), (5.,6.,0.), \
                (6.,1.,0.), (7.,2.,0.), (8.,1.,0.), \
                (9.,-1.,0.), (10.,1.,0.), (11.,-1.,0.)])
# With a specified number of points
d = D.spline(c, 3, N=100)
# With a specified density of points
```

```
e = D.spline(c, 3, density=10.)
C.convertArrays2File([c, d, e], 'out.plt')

# Spline 2D
ni = 4; nj = 4
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.,5.])
C.setValue(a, (1,4,1), [1.,4.,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.,5.])
C.setValue(a, (2,4,1), [2.,4.,2.])
C.setValue(a, (3,1,1), [3.,1.,2.])
C.setValue(a, (3,2,1), [3.,2.,5.])
C.setValue(a, (3,3,1), [3.,3.,5.])
C.setValue(a, (3,4,1), [3.,4.,2.])
C.setValue(a, (4,1,1), [4.,1.,2.])
C.setValue(a, (4,2,1), [4.,2.,5.])
C.setValue(a, (4,3,1), [4.,3.,5.])
C.setValue(a, (4,4,1), [4.,4.,2.])

b = D.spline(a, 4, N=30, M=30)
c = D.spline(a, 4, density=10.)
C.convertArrays2File([a, b, c], 'out2.plt')
```

- Creation of a Spline (pyTree):

```
# - spline (pyTree) -
import Converter.PyTree as C
import Geom.PyTree as D

# Spline 1D
c = D.polyline([(0.,0.,0.), (1.,1.,0.), (2.,1.,0.), \
                (3.,0.,0.), (4.,-1.,0.), (5.,6.,0.), \
                (6.,1.,0.), (7.,2.,0.), (8.,1.,0.), \
                (9.,-1.,0.), (10.,1.,0.), (11.,-1.,0.)])
d = D.spline(c,3,100)
C.convertPyTree2File(d, 'out.cgns')
```

Geom.**nurbs**(*Pts*, *W*, *order=3*, *N=100*, *M=100*, *density=-1.*)

Create a NURBS curve/surface using control points and weights defined by Pts and

W.

Parameters

- **Pts** (array or zone) – i-mesh (resp. (i,j)-mesh) of control points for a NRUBS curve (resp. surface)
- **W** (string) – weight for each control point defined in Pts
- **order** (integer) – order of the NURBS
- **N** (integer) – number of points in the i-direction in the resulting discretized NURBS
- **M** (integer) – number of points in the j-direction in the resulting discretized NURBS
- **density** (float) – density of points in the discretized NURBS (instead of specifying N and M)

Returns a NURBS curve/surface

Return type one array/zone (1D STRUCT or 2D STRUCT)

Example of use:

- Creation of a NURBS (array):

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

a = D.polyline ([(4.1,0.1,1.1), (1.1,0.2,1.2), (1.1,1.3,1.3),
                 (1.1,1.5,1.4), (4.5,2.5,1.5), (5.6,1.5,1.6),
                 (6.7,1.7,1.7), (7.8,0.8,1.8), (8.9,-1.9,1.9), (9,0,1)])
a = C.initVars(a,'W',1.)
C.convertArrays2File([a],'in.plt')
b = D.nurbs(a,"W", 4, N=100)
c = D.nurbs(a,"W", 4, density=10.)
C.convertArrays2File([b,c], 'out.plt')

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

```
C.setValue(a, (3,1,1), [3.,1.,2.])
C.setValue(a, (3,2,1), [3.,2.,5.])
C.setValue(a, (3,3,1), [3.,3.,12.])
C.setValue(a, (3,4,1), [3.,4.,2.])
C.setValue(a, (4,1,1), [4.,1.,2.])
C.setValue(a, (4,2,1), [4.,2.,5.])
C.setValue(a, (4,3,1), [4.,3.,5.])
C.setValue(a, (4,4,1), [4.,4.,2.])
C.setValue(a, (6,8,1), [4.,6.,14.])
C.setValue(a, (8,6,1), [4.,6.,-4.])
a = C.initVars(a,"W",1.)
a[1][3,6]=7; a[1][3,14]=9.
d = D.nurbs(a, "W", 4, N=100, M=100)
e = D.nurbs(a, "W", 4, density=20.)
C.convertArrays2File([a],'in2.plt')
C.convertArrays2File([d,e],'out2.plt')
```

- Creation of a NURBS (pyTree):

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

ni = 10; nj = 10
a = G.cart((0,0,0), (1,1,1), (ni,nj,1));
C._initVars(a,'weight',1.)
C.setValue(a,'weight',(7,1,1), 7.)
C.setValue(a,'weight',(9,5,1), 9.)
d = D.nurbs(a,'weight',4,100,100)
C.convertPyTree2File(d, 'out.cgns')

a = D.polyline ([(4.1,0.1,1.1),(1.1,0.2,1.2),(1.1,1.3,1.3),(1.1,1.5,1.4),(4.5,2.
↪5,1.5),(5.6,1.5,1.6),(6.7,1.7,1.7),(7.8,0.8,1.8),(8.9,-1.9,1.9),(9,0,1)])
a = C.initVars(a,'weight',1.)
C.setValue(a, 'weight', (7,1,1), 7.)
C.setValue(a, 'weight', (9,1,1), 9.)
b = D.nurbs(a,'weight',4,2000)
C.convertPyTree2File(b, 'out2.cgns')
```

Geom.**bezier**(*Pts, N=100, M=100, density=-1.*)

Create a Bezier curve/surface using control points defined by Pts.

Parameters

- **Pts** (array or zone) – i-mesh (resp. (i,j)-mesh) of control points for a spline curve (resp. surface)
- **N** (integer) – number of points in the i-direction in the resulting discretized Bezier
- **M** (integer) – number of points in the j-direction in the resulting discretized Bezier
- **density** (float) – density of points in the discretized Bezier (instead of specifying N and M)

Returns a Bezier curve/surface

Return type one array/zone (1D STRUCT or 2D STRUCT)

Example of use:

- Creation of a Bezier curve (array):

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

# Bezier 1D
pts = D.polyline([(0.,0.,0.), (0.,1.,0.), (2.,1.,0.), (2.,0.,0.),\
                  (4.,-1.,0.), (5.,6.,0.)])
# With a specified number of points
a = D.bezier(pts, N=100)
# With a specified point density
b = D.bezier(pts, density=10.)
C.convertArrays2File([pts, a, b], 'out.plt')

# Bezier 2D
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.)
C.convertArrays2File([a]+[b], 'out2.plt')
```

- Creation of a Bezier curve (pyTree):

```
# - bezier (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

# Bezier 1D
pts = D.polyline([(0.,0.,0.), (0.,1.,0.), (2.,1.,0.), (2.,0.,0.),
                  (4.,-1.,0.), (5.,6.,0.),])
a = D.bezier(pts, 100); a[0] = 'bezier'
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**curve**(*f*, *N=100*)

Create a curve defined by a parametric function or an expression.

Parameters

- **f** (Python function or string) – Python function or set of expressions separated by ";"“
- **N** (integer) – number of discretization points per direction

Returns a parametric curve

Return type one array/zone (1D STRUCT)

Example of use:

- Creation of a parametric curve (array):

```
# - curve (array) -
import Converter as C
import Geom as D

# Definition of parametric curve by a function
def f(t):
    x = t; y = t*t+1; z = 0.
    return (x,y,z)
a = D.curve(f)

# Definition by equation
b = D.curve('{x}=cos(2*pi*{t}); {y}=sin(2*pi*{t}); {z} = 0.')

# Definition from data base
from Geom.Parametrics import base
c = D.curve(base['circle'])
C.convertArrays2File([a,b], "out.plt")
```

- Creation of a parametric curve (pyTree):

```
# - curve (pyTree) -
import Converter.PyTree as C
import Geom.PyTree as D

# User definition of parametric curve
def f(t):
    x = t; y = t*t+1; z = 0.
    return (x,y,z)

a = D.curve(f)
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**surface**(*f*, *N=100*)

Create a surface defined by an parametric function or an expression.

Parameters

- **f** (Python function or string) – Python function or set of expressions separated by ";"“
- **N** (integer) – number of discretization points per direction

Returns a parametric surface

Return type one array/zone (2D STRUCT)

Example of use:

- Creation of a parametric surface (array):

```
# - surface (array) -
import Converter as C
import Geom as D

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

a = D.surface(f)

# Definition by formula
b = D.surface('{x} = cos(pi*{t}); {y} = sin(pi*{u}); {z} = {t}*{u}')
C.convertArrays2File([a, b], 'out.plt')
```

- Creation of a parametric surface (pyTree):

```
# - surface (PyTree) -
import Converter.PyTree as C
import Geom.PyTree as D

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

a = D.surface(f)
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**cone**(*C, Rb, Rt, H, N=100*)

Create a cone discretized by NxN points.

Parameters

- **C** (3-tuple of floats) – center coordinates
- **Rb** (float) – radius of the basis of the cone
- **Rt** (float) – radius of the top of the cone
- **H** (float) – height of the cone
- **N** (integer) – number of discretization points per direction

Returns a cone

Return type one array/zone (2D STRUCT)

Example of use:

- Creation of a cone(array):

```
# - cone (array) -
import Geom as D
import Converter as C

a = D.cone((0,0,0), 1. , 0.5, 1.)
C.convertArrays2File(a, "out.plt")
```

- Creation of a cone (pyTree):

```
# - cone (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C
```

```
a = D.cone((0,0,0), 1. , 0.5, 1.)
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**torus**(*C*, *R*, *r*, *alphas=0.*, *alphae=360.*, *betas=0.*, *betae=360.*, *NR=100*, *Nr=100*)

Create a portion of torus discretized by NRxNr points, of center C, axis Z and radii R (main radius) and r (tube radius) between angles alphas and alphae (in the XY-plane) and between betas and betae (in the RZ-plane).

Parameters

- **C** (3-tuple of floats) – center coordinates
- **R** (float) – main radius
- **r** (float) – tube radius
- **alphas** (float) – minimum azimuth in the XY-plane
- **alphae** (float) – maximum azimuth in the XY-plane
- **betas** (float) – minimum azimuth in the RZ-plane
- **betae** (float) – maximum azimuth in the RZ-plane
- **NR** (integer) – number of discretization points in azimuth
- **Nr** (integer) – number of discretization points in the axial direction

Returns a torus

Return type one array/zone (2D STRUCT)

Example of use:

- Creation of a torus (array):

```
# - torus (array) -
import Geom as D
import Converter as C

a = D.torus((0,0,0), 5., 2.)
C.convertArrays2File(a, "out.plt")
```

- Creation of a torus (pyTree):

```
# - torus (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C
```

```
a = D.torus((0.,0.,0.), 5., 2.)
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**sphere**(*C*, *R*, *N=100*)

Create a structured mesh defining a sphere of radius R with N points in the longitudinal direction and NxN along the latitude.

Parameters

- **C** (3-tuple of floats) – sphere center coordinates
- **R** (float) – sphere radius
- **N** (integer) – number of discretization points in the longitudinal direction

Returns a structured mesh of a sphere degenerated at poles

Return type one array/zone (2D STRUCT)

Example of use:

- Creation of a sphere (array):

```
# - sphere (array) -
import Geom as D
import Converter as C

a = D.sphere((0,0,0), 1., 20)
C.convertArrays2File(a, "out.plt")
```

- Creation of a sphere (pyTree):

```
# - sphere (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.sphere((0,0,0), 1., 20)
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**sphere6**(*C*, *R*, *N=100*, *ntype='STRUCT'*)

Create a mesh made of 6 parts defining a sphere of radius R with N points per direction. This mesh is not degenerated at poles in consequence.

Parameters

- **C** (3-tuple of floats) – sphere center coordinates
- **R** (float) – sphere radius
- **N** (integer) – number of discretization points in the longitudinal direction
- **ntype** (string) – type of output mesh ('STRUCT', 'QUAD', 'TRI')

Returns a mesh of a sphere

Return type a list of 6 arrays/zones (STRUCT) or 1 array/zone (QUAD and TRI)

Example of use:

- Non-degenerated surface mesh of a sphere (array):

```
# - sphere6 (array) -
import Geom as D
import Converter as C

a = D.sphere6((0,0,0), 1., N=20)
b = D.sphere6((3,3,0), 1.2, N=20, ntype='QUAD')
C.convertArrays2File(a+[b], "out.plt")
```

- Non-degenerated surface mesh of a sphere (pyTree):

```
# - sphere6 (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

A = D.sphere6((0,0,0), 1., 20)
b = D.sphere6((3,0,0), 1.2, N=20, ntype='QUAD')
C.convertPyTree2File(A+[b], 'out.cgns')
```

Geom.**sphereYinYang**(*C, R, N=100, ntype='STRUCT'*)

Create an overset mesh of 2 parts defining a sphere of radius R with N points per direction.

Parameters

- **C** (3-tuple of floats) – sphere center coordinates
- **R** (float) – sphere radius
- **N** (integer) – number of discretization points in the longitudinal direction
- **ntype** (string) – type of output mesh ('STRUCT', 'QUAD', 'TRI')

Returns a mesh of a sphere

Return type a list of 2 arrays/zones (STRUCT) or 1 array/zone (QUAD and TRI)

Example of use:

- Creation of a Yin-Yang sphere (array):

```
# - sphereYinYang (array) -
import Geom as D
import Converter as C

a = D.sphereYinYang((0,0,0), 1., 50)
C.convertArrays2File(a, "out.plt")
```

- Creation of a Yin-Yang sphere (pyTree):

```
# - sphereYinYang (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.sphereYinYang((0,0,0), 1., 50)
C.convertPyTree2File(a, "out.cgns")
```

Geom.**disc**(*C, R, N=100, ntype='STRUCT'*)

Create a mesh of 5 parts defining a disc of radius R with NxN grid points.

Parameters

- **C** (3-tuple of floats) – sphere center coordinates
- **R** (float) – disc radius
- **N** (integer) – number of discretization points for each grid
- **ntype** (string) – type of output mesh ('STRUCT', 'QUAD', 'TRI')

Returns a mesh of a disc

Return type a list of 5 arrays/zones (STRUCT) or 1 array/zone (QUAD and TRI)

Example of use:

- Creation of a disc (array):

```
# - disc (array) -
import Geom as D
import Converter as C
```

```
a = D.disc((0,0,0), 1.)
C.convertArrays2File(a, 'out.plt')
```

- Creation of a disc (pyTree):

```
# - disc (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.disc((0,0,0), 1.)
b = D.disc((3,0,0), 1., N=20, ntype='QUAD')
C.convertPyTree2File(a+[b], 'out.cgns')
```

Geom.**triangle**(*P1, P2, P3, N=0, ntype='TRI'*)

Create a triangle mesh defined by 3 vertices P1, P2, P3.

Parameters

- **P1** (3-tuple of floats) – (x,y,z) of first vertex
- **P2** (3-tuple of floats) – (x,y,z) of second vertex
- **P3** (3-tuple of floats) – (x,y,z) of third vertex
- **N** (integer) – number of discretization points
- **ntype** (string) – type of output mesh ('STRUCT', 'QUAD', 'TRI')

Returns a mesh of a triangle

Return type a list of 3 arrays/zones (STRUCT) or 1 array/zone (QUAD and TRI)

Example of use:

- Creation of a triangle (array):

```
# - triangle (array) -
import Geom as D
import Converter as C

a = D.triangle((0,0,0), (0.1,0.,0.1), (0.05, 0.08, 0.1))
C.convertArrays2File(a, "out.plt")
```

- Creation of a triangle (pyTree):

```
# - triangle (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.triangle((0,0,0), (0.1,0.,0.1), (0.05, 0.08, 0.1))
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**quadrangle**(*P1, P2, P3, P4, N=0, ntype='QUAD'*)

Create a quadrangle of vertices P1, P2, P3, P4.

Parameters

- **P1** (3-tuple of floats) – (x,y,z) of first vertex
- **P2** (3-tuple of floats) – (x,y,z) of second vertex
- **P3** (3-tuple of floats) – (x,y,z) of third vertex
- **P4** (3-tuple of floats) – (x,y,z) of fourth vertex
- **N** (integer) – number of discretization points
- **ntype** (string) – type of output mesh ('STRUCT', 'QUAD', 'TRI')

Returns a mesh of a quadrangle

Return type a list of 1 array/zone (STRUCT) or 1 array/zone (QUAD and TRI)

Example of use:

- Creation of a quadrangle (array):

```
# - quadrangle (array) -
import Geom as D
import Converter as C

a = D.quadrangle((0,0,0.1), (0.1,0.,0.1), (0.05, 0.08, 0.1), (0.02,0.05,0.1))
C.convertArrays2File(a, "out.plt")
```

- Creation of a quadrangle (pyTree):

```
# - quadrangle (PyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.quadrangle((0,0,0.1), (0.1,0.,0.1), (0.05, 0.08, 0.1), (0.02,0.05,0.1))
```

```
b = D.quadrangle((0,0,0.1), (0.1,0.,0.1), (0.05, 0.08, 0.1), (0.02,0.05,0.1),␣
↪N=20, ntype='QUAD')
C.convertPyTree2File(a+[b], 'out.cgns')
```

Geom.**box**(*P1, P2, N=100, ntype='STRUCT'*)

Create an axis aligned box passing by points P1 and P2.

Parameters

- **P1** (3-tuple of floats) – (x,y,z) of first vertex
- **P2** (3-tuple of floats) – (x,y,z) of second vertex
- **N** (integer) – number of discretization points
- **ntype** (string) – type of output mesh ('STRUCT', 'QUAD', 'TRI')

Returns a mesh of a box

Return type a list of 6 arrays/zones (STRUCT) or 1 array/zone (QUAD and TRI)

Example of use:

- Creation of a box (array):

```
# - box (array) -
import Geom as D
import Converter as C

a = D.box((0,0,0), (1,1,1))
C.convertArrays2File(a, 'out.plt')
```

- Creation of a box (pyTree):

```
# - box (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.box((0,0,0), (1,1,1))
b = D.box((2,0,0), (3,1,1), N=30, ntype='QUAD')
C.convertPyTree2File(a+[b], 'out.cgns')
```

Geom.**cylinder**(*C, R, H, N=100, ntype='STRUCT'*)

Create cylinder mesh made of two discs of center C and radius R and of height H.

Parameters

- **C** (3-tuple of floats) – bottom disc center
- **R** (float) – Radius of discs
- **H** (float) – Height of cylinder
- **N** (integer) – number of discretization points
- **ntype** (string) – type of output mesh ('STRUCT', 'QUAD', 'TRI')

Returns a mesh of a cylinder

Return type a list of 11 arrays/zones (STRUCT) or 1 array/zone (QUAD and TRI)

Example of use:

- Creation of a cylinder (array):

```
# - cylinder (array) -
import Geom as D
import Converter as C

a = D.cylinder((0,0,0), 1., 10.)
C.convertArrays2File(a, 'out.plt')
```

- Creation of a cylinder (pyTree):

```
# - cylinder (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.cylinder((0,0,0), 1., 10.)
b = D.cylinder((3,0,0), 1., 5., N=20, ntype='QUAD')
C.convertPyTree2File(a+[b], 'out.cgns')
```

3.2 Typing text using meshes

Geom.**text1D**(*text*, *font='text1'*, *smooth=0*, *offset=0.5*)

Create 1D meshes of given text.

Parameters

- **text** (string) – text with separated characters
- **font** (string) – chosen font name (can be 'vera','chancery','courier','text1','nimbus')
- **smooth** (integer) – letter smoothness (0-4)

- **offset** (float) – distance between two letters

Returns a mesh for each character of the text

Return type a list of arrays or zones

Example of use:

- Text defined by a set of 1D meshes (array):

```
# - text1D (array) -
import Geom as D
import Converter as C
import Transform as T

a = D.text1D("Cassiopee - text1")
b = D.text1D("Cassiopee - text1 smoothed", smooth=4, offset=1.)
b = T.translate(b, (0,-12,0))
c = D.text1D("Cassiopee - vera", font='vera')
c = T.translate(c, (0,-24,0))
d = D.text1D("Cassiopee - chancery", font='chancery')
d = T.translate(d, (0,-36,0))
e = D.text1D("Cassiopee - courier", font='courier')
e = T.translate(e, (0,-48,0))
f = D.text1D("Cassiopee - nimbus", font='nimbus')
f = T.translate(f, (0,-60,0))

C.convertArrays2File(a+b+c+d+e+f, 'out.plt')
```

- Text defined by a set of 1D meshes (pyTree):

```
# - text1D (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.text1D("CASSIOPEE")
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**text2D**(*text*, *font='text1'*, *smooth=0*, *offset=0.5*)
Create a triangular mesh of a text (letters are filled with triangles).

Parameters

- **text** (string) – text with separated characters
- **font** (string) – chosen font name (can be 'vera','chancery','courier','text1','nimbus')
- **smooth** (integer) – letter smoothness (0-4)

- **offset** (float) – distance between two letters

Returns a single mesh for the text string

Return type an array or a zone

Example of use:

- Text defined by a set of 2D meshes (array):

```
# - text2D (array) -
import Geom as D
import Converter as C

a = D.text2D("ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789", smooth=0, offset=1.)
C.convertArrays2File([a], 'out.plt')
```

- Text defined by a set of 2D meshes (pyTree):

```
# - text2D (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.text2D("Cassiopee")
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**text3D**(*text*, *font='text1'*, *smooth=0*, *offset=0.5*, *thickness=8.*)
Create a 3D mesh of a text.

Parameters

- **text** (string) – text with separated characters
- **font** (string) – chosen font name (can be 'vera','chancery','courier','text1','nimbus')
- **smooth** (integer) – letter smoothness (0-4)
- **offset** (float) – distance between two letters
- **thickness** (float) – thickness of letters

Returns a single mesh of text

Return type an array or a zone

Example of use:

- Text defined by a set of 3D meshes (array):

```
# - text3D (array) -
import Geom as D
import Converter as C

a = D.text3D("Cassiopee", smooth=1, thickness=2.)
C.convertArrays2File([a], 'out.plt')
```

- Text defined by a set of 3D meshes (pyTree):

```
# - text3D (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.text3D("CASSIOPEE")
C.convertPyTree2File(a, 'out.cgns')
```

3.3 Geometry modification

`Geom.`**`uniformize`**(*a, N=100, h=-1, factor=-1, density=-1, sharpAngle=30.*)

Remesh a 1D curve with a regular mesh step. You can specify one of N or factor or density or h.

Parameters

- **a** (`[array, list of arrays] or [pyTree, base, zone, list of zones]`) – original curve to be remeshed (i-STRUCT or BAR)
- **N** (`int`) – the final number of points
- **h** (`float`) – the final mesh step
- **factor** (`float`) – factor for the number of points regarding initial number of points of curve.
- **density** (`float`) – point density
- **sharpAngle** (`float`) – point where the curve has a local angle greated than sharpAngle are enforced.

Return type identical to a

Example of use:

- Uniformizing steps of a 1D curve (array):

```
# - uniformize (array) -
import Geom as D
```

```
import Converter as C

a = D.polyline([(0,0,0), (1,1,0), (2,0,0), (3,1,0), (4,0,0)])
a = D.uniformize(a, N=100)

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

- Uniformizing steps of a 1D curve (pyTree):

```
# - uniformize (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.polyline([(0,0,0), (1,1,0), (2,0,0), (3,1,0), (4,0,0)])
a = D.uniformize(a, N=100)

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

Note: new in version 2.7.

`Geom.`**`refine`**(*a, N=10, factor=-1, sharpAngle=30.*)

Remesh a 1D curve keeping the original point distribution but densifying or coarsening it. You can specify N or factor.

Parameters

- **a** (`[array, list of arrays] or [pyTree, base, zone, list of zones]`) – original curve to be refined (i-STRUCT or BAR)
- **N** (`int`) – the final number of points
- **factor** (`float`) – factor for the number of points regarding initial number of points of curve.
- **sharpAngle** (`float`) – point where the curve has a local angle greated than sharpAngle are enforced.

Return type identical to a

Example of use:

- Refining/coarsening a 1D curve (array):

```
# - refine (array) -
import Geom as D
import Transform as T
```

```
import Converter as C

a = D.line((0,0,0), (1,0,0), N=10)
b = D.line((1,0,0), (2,1,0), N=30)
a = T.join([a,b])
a = D.refine(a, N=30)
C.convertArrays2File(a, 'out.plt')
```

- Refining/coarsening a 1D curve (pyTree):

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

a = D.line((0,0,0), (1,0,0), N=10)
b = D.line((1,0,0), (2,1,0), N=30)
a = T.join([a,b])
a = D.refine(a, N=30)
C.convertPyTree2File(a, 'out.cgns')
```

Note: new in version 2.7.

Geom.**enforceh**(*a*, *N=100*, *h=-1*)

Enforce some mesh steps or mesh factors in a 1D curve. To enforce a step use D.setH(a, ind, h), to enforce a factor, use D.setF(a, ind, f). If you want to enforce h, you must specify N, the final number of points. If you want to enforce f, you must specify h, the mesh size for f=1.

Parameters

- **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – original curve to be remeshed (i-STRUCT or BAR)
- **N** (int) – the final number of points
- **h** (float) – the final mesh step

Return type identical to a

Example of use:

- Enforce steps in a 1D curve (array):

```
# - enforceh (array) -
import Geom as D
import Converter as C

a = D.line((0,0,0), (1,0,0), N=30)
D.setH(a, 0, 0.01); D.setH(a, -1, 0.1)

b = D.enforceh(a, N=40)
C.convertArrays2File(b, 'out.plt')
```

- Enforce steps in a 1D curve (pyTree):

```
# - enforceh (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.line((0,0,0), (1,0,0), N=30)
D.setH(a, 0, 0.01); D.setH(a, -1, 0.1)

b = D.enforceh(a, N=40)
C.convertPyTree2File(b, 'out.cgns')
```

Note: new in version 2.7.

Geom.**lineDrive**(*a*, *d*)

Generate a surface mesh starting from a curve a and a single or a set of driving curves. The first point of the driving curves must match with one point of the original curve a.

Parameters

- **a** (`array or zone`) – original curve to be extruded wrt the driving curve d
- **d** (`[array, list of arrays] or [pyTree, base, zone, list of zones]`) – driving curve or set of driving curves

Returns a surface structured mesh

Return type an array or a zone

Example of use:

- Extrusion of a NACA0012 (array):

```
# - lineDrive (array) -
import Geom as D
import Converter as C

# With one driving curve
a = D.naca(12.)
b = D.line((0,0,0), (0.,0.,1.))
c = D.lineDrive(a, b)
C.convertArrays2File([c], 'out.plt')

# With a set of driving curves
a = D.naca(12.)
d1 = D.line((0,0,0), (0.,0.,1.))
d2 = D.line((1,0,0), (2,0,1))
c = D.lineDrive(a, [d1,d2])
C.convertArrays2File([c,d1,d2,a], 'out.plt')
```

- Extrusion of a NACA0012 (pyTree):

```
# - lineDrive (pyTree)-
import Geom.PyTree as D
import Converter.PyTree as C

# With one driving curve
a = D.naca(12.)
l = D.line((0,0,0), (0,0.,1.))
a = D.lineDrive(a, l)
C.convertPyTree2File(a, 'out.cgns')

# With a set of driving curves
a = D.naca(12.)
d1 = D.line((0,0,0), (0.,0.,1.))
d2 = D.line((1,0,0), (2,0,1))
a = D.lineDrive(a, [d1,d2])
C.convertPyTree2File(a, 'out.cgns')
```

`Geom.`**`orthoDrive`**(*a*, *d*, *mode=0*)

Generate a surface mesh starting from a curve a and a single driving curve. The initial mesh is driven orthogonally to the driving curve. The first point of the driving curves must match with one point of the original curve a.

Parameters

- **a** (`array or zone`) – original curve to be extruded orthogonally wrt the driving curve d

- **d** ([array] or [zone]) – driving curve
- **mode** – if mode=0, return one single zone, if mode=1, duplicate a and return a list of zones

Returns a surface structured mesh or a list of meshes

Example of use:

- Extrusion of a NACA0012 (array):

```
# - orthoDrive (array) -
import Geom as D
import Converter as C

a = D.circle((0,0,0),1.)
c = D.polyline([(0.,1.,0.), (0.,1.,1.), (2.,1.,2.)])
d = D.spline(c, 3, N=100)
o = D.orthoDrive(a, d, mode=0)
C.convertArrays2File(o, 'out.plt')
```

- Extrusion of a NACA0012 (pyTree):

```
# - orthoDrive (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

a = D.circle((0,0,0),1.)
c = D.polyline([(0.,1.,0.), (0.,1.,1.), (2.,1.,2.)])
d = D.spline(c, 3, N=100)
o = D.orthoDrive(a, d, mode=0)
C.convertPyTree2File(o, 'out.cgns')
```

Geom.**axisym**(*a, C, axis, angle=360., Ntheta=100, rmod=None*)

Create an axisymmetrical mesh given one of its borders following axis.

Exists also as in place version (_axisym) that modifies a and returns None.

Parameters

- **a** ([array, list of arrays] or [zone, list of zones, base, pyTree]) – axis-aligned border of the axisymmetrical mesh (either structured-1D or 2D or BAR or TRI or QUAD)
- **C** (3-tuple of floats) – center of rotation of the mesh
- **axis** (3-tuple of floats) – rotation axis

- **angle** (float) – azimuthal sector angle
- **Ntheta** (integer) – number of points in the azimuthal direction
- **rmod** (identical to a) – optional curve defining r=f(theta) instead of defining theta and Ntheta

Returns a 2D or 3D mesh (either structured or QUAD or PENTA or HEXA)

Return type Identical to a

Example of use:

- Creation of a surface mesh by axisymmetry (array):

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

# Axisym a curve
a0 = D.line((0.5,0,0), (0.6,0,1))
a = D.axisym(a0,(0.,0.,0.),(0.,0.,1.),360.,360)
C.convertArrays2File(a, "out.plt")

# Axisym a curve with varying r
a0 = D.line((1.0,0,0), (0.,0,1))
a1 = D.circle((0,0,0), 2.)
import Modeler.Models as Models
a1 = Models.circle2(1, 0.8)
a = D.axisym(a0, (0.,0.,0.), (0.,0.,1.), rmod=a1)
C.convertArrays2File([a,a0,a1], "out1.plt")

# Axisym a 2D cart grid
a0 = G.cart((0.,0.,0.), (0.1,0.1,0.2),(10,10,1))
a = D.axisym(a0,(1.,0.,0.),(0.,1.,0.),30.,4)
C.convertArrays2File(a, "out2.plt")
```

- Creation of a surface mesh by axisymmetry (pyTree):

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

# Axisym a curve
a0 = D.line((0.5,0,0), (0.6,0,1))
a = D.axisym(a0,(0.,0.,0.),(0.,0.,1.),360.,360)
C.convertPyTree2File(a, "out.cgns")
```

```
# Axisym a curve with varying r
a0 = D.line((1.0,0,0), (0.,0,1))
a1 = D.circle((0,0,0), 2.)
a = D.axisym(a0, (0.,0.,0.), (0.,0.,1.), rmod=a1)
C.convertPyTree2File([a,a0,a1], "out1.cgns")

# Axisym a 2D cart grid
a = G.cart((0.,0.,0.), (0.1,0.1,0.2),(10,10,1))
a = D.axisym(a,(1.,0.,0.),(0.,1.,0.),30.,4)
C.convertPyTree2File(a, 'out2.cgns')
```

Geom.**connect1D**(*curves*, *sharpness=0*, *N=10*, *lengthFactor=1.*)

Connect non-matching curves by a line or by a Spline with N points.

Parameters

- **curves** (list of arrays or list of zones) – two curves to be connected
- **sharpness** (integer) – 0: connected by a line; 1: connected by a Spline
- **N** (integer) – number of points in the connection
- **lengthFactor** (float) – the connection is bounded by lengthFactor x the length of the initial curves.

Returns a single curve connecting both curves

Return type array or zone

Example of use:

- Connect two lines (array):

```
# - connect1D (array) -
import Geom as D
import Converter as C
# input
P1 = [-0.5,0,0]; P1b = [0.5,0,0]
P2 = [1,-1.5,0]; P2b = [1,-0.5,0]
l1 = D.line(P1,P1b)
l2 = D.line(P2,P2b)
out = D.connect1D([l1,l2], sharpness=1, lengthFactor=10.)
C.convertArrays2File(out, 'out.plt')
```

- Connect two lines (pyTree):

```
# - connect1D (pyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

P1 = [-0.5,0,0]; P1b = [0.5,0,0]
P2 = [1,-1.5,0]; P2b = [1,-0.5,0]
l1 = D.line(P1,P1b)
l2 = D.line(P2,P2b)

out = D.connect1D([l1,l2], sharpness=0)
C.convertPyTree2File(out, 'out.cgns')
```

3.4 Information about geometries

For pyTrees, the information is stored as a son node of 'FlowSolution' if it is defined for all the points of the geometry.

Geom.**getLength**(*a*)

Return the length of a discretized curve or a set of curves.

Parameters **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – curve or list of curves

Returns the length of curves

Return type float

Example of use:

- Length of a line (array):

```
# - getLength (array) -
import Geom as D

a = D.line((0,0,0), (1,0,0)); print(D.getLength(a))
```

- Length of a line (pyTree):

```
# - getLength (pyTree) -
import Geom.PyTree as D

a = D.line((0,0,0), (1,0,0)); print(D.getLength(a))
```

Geom.**getDistantIndex**(*a, ind, l*)

Return the point index in a that is distant of l from a point of index ind in a.

Parameters

- **a** (`array or zone`) – 1D mesh
- **ind** (`integer`) – index of starting point
- **l** (`float`) – distance of the end point to the starting point of index ind

Returns the index in a of the end point at distance l of ind

Return type integer

Example of use:

- Index of point distant to another one (array):

```
# - getDistantIndex (array) -
import Geom as D

a = D.line((0.,0.,0.), (1.,0.,0), 100)
print('distant Index: %d.'%D.getDistantIndex(a, 25, 0.2))
print('distant Index: %d.'%D.getDistantIndex(a, 25, -0.2))
```

- Index of point distant to another one (pyTree):

```
# - getDistantIndex (pyTree)-
import Geom.PyTree as D

a = D.line((0.,0.,0.), (1.,0.,0), 100)
print('distant Index: %d.'%D.getDistantIndex(a, 25, 0.2))
```

`Geom.`**`getNearestPointIndex`**(*a*, *P*)

Return the index and the squared distance of the nearest point of P(x,y,z) in a If a is a list of meshes, the minimum distance for all meshes in a from P is returned.

Parameters

- **a** (`[array, list of arrays] or [pyTree,base, list of zones, zone]`) – 1D mesh
- **P** (`(float,float,float) or [(float,float,float),...,(float, float,float)]`) – coordinates of the point P or point list P

Returns the index and squared distance of the nearest point(s) of a to point(s) P

Return type [(integer,float) or list of (integer,float)]

Example of use:

- Index of nearest point to P (array):

```
# - getNearestPointIndex (array) -
import Generator as G
import Geom as D

a = G.cart((0.,0.,0.), (0.1,0.1,0.2),(10,10,1))
inds = D.getNearestPointIndex(a, (0.55,0.34,0)); print(inds)
inds = D.getNearestPointIndex(a, [(0.55,0.34,0), (0.56,0.32,0)]); print(inds)
```

- Index of nearest point to P (pyTree):

```
# - getNearestPointIndex (pyTree) -
import Generator.PyTree as G
import Geom.PyTree as D

a = G.cart((0.,0.,0.), (0.1,0.1,0.2),(10,10,1))
inds = D.getNearestPointIndex(a, (0.55,0.34,0)); print(inds)
```

Geom.**getCurvatureRadius**(*a*)

Return the curvature radius of a curve a.

Parameters **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – 1D mesh

Returns the curvature radius named 'radius'.

Return type Identical to a

Example of use:

- Curvature radius of a curve (array):

```
# - getCurvatureRadius (array) -
import Geom as D
import Converter as C
pts = D.polyline([(6,0.01,1), (5.4,0.036,1), (4.8,0.064,1), (2.5,0.21,1),
                  (0.3,0.26,1),(0,0.047,1),(0,0,0)])
a = D.bezier(pts, 100)
rad = D.getCurvatureRadius(a)
C.convertArrays2File(a, 'out.plt')
```

- Curvature radius of a curve (pyTree):

```
# - getCurvatureRadius (pyTree) -
import Geom.PyTree as D
```

```
import Converter.PyTree as C

a = D.circle((0,0,0), 1, 10, 0, 10)
a = D.getCurvatureRadius(a)
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**getCurvatureAngle**(*a*)

Return the curvature angle of a curve a.

Parameters **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – 1D mesh

Returns the curvature angle named 'angle'.

Return type Identical to a

Example of use:

- Curvature angle of a curve (array):

```
# - getCurvatureAngle (array) -
import Converter as C
import Geom as D
import Transform as T

a1 = D.line((0.,0.,0.), (1.,0.,0), 100)
a2 = D.line((1.,0.,0.), (1.,1,0), 100)
a = T.join (a1, a2)
a3 = D.getCurvatureAngle(a)
a = C.addVars([a, a3])
C.convertArrays2File(a, 'out.plt')
```

- Curvature angle of a curve (pyTree):

```
# - getCurvatureAngle (pyTree) -
import Converter.PyTree as C
import Geom.PyTree as D

a = D.polyline([(0.,0.,0.),(1.,1.,0.),(2.,0.,0.)])
a = D.getCurvatureAngle(a)
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**getCurvatureHeight**(*a*)

Return the curvature height of a curve a.

Parameters **a** (`[array, list of arrays] or [pyTree, base, zone, list of zones]`) – 1D mesh

Returns the curvature height named 'hmax' as an array or as a flow solution at nodes.

Return type Identical to a

Example of use:

- Curvature height of a curve (array):

```
# - getCurvatureHeight (array) -
import Converter as C
import Geom as D
import Transform as T

a1 = D.line((0.,0.,0.), (1.,0.,0), 100)
a2 = D.line((1.,0.,0.), (1.,1,0), 100)
a = T.join (a1, a2)
hmax = D.getCurvatureHeight( a )
a = C.addVars([a,hmax])
C.convertArrays2File(a, 'out.plt')
```

- Curvature height of a curve (pyTree):

```
# - getCurvatureHeight(pyTree) -
import Converter.PyTree as C
import Geom.PyTree as D

a = D.polyline([(0.,0.,0.),(1.,1.,0.),(2.,0.,0.)])
a = D.getCurvatureHeight(a)
C.convertPyTree2File(a, 'out.cgns')
```

`Geom.`**`getSharpestAngle`**(*a*)

Return the sharpest angle (in degrees) of a curve. Sharpest angle is defined at each node of input curve.

Parameters **a** (`[array, list of arrays] or [pyTree, base, zone, list of zones]`) – 1D mesh

Returns the sharpest angle named 'alpha' as an array or as a flow solution at nodes.

Return type Identical to a

Example of use:

- Sharpest angle of a curve (array):

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

N = 10
d1 = G.cart((0.,0.,0.), (0.05,1,1),(N,1,4))
d2 = G.cart((0.,0.,0.), (1.,0.001,1),(1,10*N,4))
d2 = T.rotate(d2,(0.,0.,0.),(0.,0.,1.),30.)
s = T.join(d1,d2)
s = C.convertArray2Hexa(s)
s = T.reorder(s,(-1,))
r = D.getSharpestAngle(s)
s = C.addVars([s,r])
C.convertArrays2File(s, "out.plt")
```

- Sharpest angle of a curve (pyTree):

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

N = 10
d1 = G.cart((0.,0.,0.), (0.05,1,1),(N,1,4))
d2 = G.cart((0.,0.,0.), (1.,0.001,1),(1,10*N,4))
d2 = T.rotate(d2,(0.,0.,0.),(0.,0.,1.),30.)
s = T.join(d1,d2)
s = C.convertArray2Hexa(s)
s = T.reorder(s,(-1,))
s = D.getSharpestAngle(s)
C.convertPyTree2File(s, "out.cgns")
```

`Geom.`**`getCurvilinearAbscissa`**(*a*)

Return the curvilinear abscissa of a curve a (scalar in range [0.,1.]).

Parameters **a** (`[array, list of arrays] or [pyTree, base, zone, list of zones]`) – 1D mesh

Returns the curvilinear abscissa named 's' as an array or as a flow solution at nodes.

Return type Identical to a

Example of use:

- Curvilinear abscissa of a curve (array):

```
# - getCurvilinearAbscissa (array) -
import Converter as C
import Geom as D
import Transform as T

a = D.line((0.,0.,0.), (1.,0.,0), 100)
a2 = D.line((1.,0.,0.), (1.,1,0), 100)
a = T.join (a, a2)
a3 = D.getCurvilinearAbscissa(a)
a = C.addVars([a, a3])
C.convertArrays2File(a, "out.plt")
```

- Curvilinear abscissa of a curve (pyTree):

```
# - getCurvilinearAbscissa (pyTree)-
import Converter.PyTree as C
import Geom.PyTree as D

a = D.line((0.,0.,0.), (1.,0.,0), 100)
a = D.getCurvilinearAbscissa(a)
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**getDistribution**(*a*)

Return the distribution (curvilinear abscissa) of a curve as a mesh coordinates.

Parameters **a** (`[array, list of arrays] or [pyTree, base, zone, list of zones]`) – 1D mesh

Returns the distribution of the curve as mesh coordinates

Return type Identical to a

Example of use:

- Distribution of a uniform NACA0012 profile (array):

```
# - getDistribution (array) -
import Geom as D
import Converter as C

Foil = D.naca(12., N=49)
a = D.getDistribution(Foil)
C.convertArrays2File(a, 'out.plt')
```

- Distribution of a uniform NACA0012 profile (pyTree):

```
# - getDistribution (PyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

Foil = D.naca(12., N=49)
a = D.getDistribution(Foil)
C.convertPyTree2File(a, 'out.cgns')
```

Geom.**getTangent**(*a*)

Return the unit tangent vector of all nodes of a 1D array (only structured) as a mesh coordinates.

Parameters **a** ([array, list of arrays] or [pyTree, base, zone, list of zones]) – 1D structured mesh

Returns the unit tangent vector of the curve as mesh coordinates

Return type Identical to a

Example of use:

- Unit tangent vector of a spline (array):

```
# - getTangent (array) -
import Geom as D
import Converter as C
c = D.polyline([(0,0,0),(1,1,0),(2,-1,0)])
a = D.spline(c, order=3, density=10.)
b = D.getTangent(a)
C.convertArrays2File(b, "out.plt")
```

- Unit tangent vector of a spline (pyTree):

```
# - getTangent (PyTree) -
import Geom.PyTree as D
import Converter.PyTree as C

c = D.polyline([(0,0,0),(1,1,0),(2,-1,0)])
a = D.spline(c, order=3, density=10.)
b = D.getTangent(a)
C.convertPyTree2File(b, "out.cgns")
```

CHAPTER FOUR

INDEX

- genindex
- modindex
- search