Arnica Documentation

Team COOP

Feb 22, 2022

Contents

This is the documentation for the *default* branch of ARNICA.

Contents:

CHAPTER 1

Introduction

Arnica is a package of open source python modules developped by CERFACS-Team COOP, as a toolkit for CFD. This package contains a solver of second partial derivative equations to treat heat conduction and heat radiation problem. The 2nd order finit difference scheme is used to solve the inside of a computational domain and that of first order for boundaries. Arnica is able to treat a 2D computational mesh at present.

CHAPTER 2

Composition

- phys: a few test cases for different type of phenomenon
- utils: some tools to facilitate developement/communication with other external applications
- solvers_2d (Deprecated) : modules to solve two dimensional heat conduction and radiation problem

CHAPTER 3

arnica.phys package

3.1 Physical utilities

These physical tools are helpers around CFD-related problems.

- **solid_material** is a class to store solid properties for CHT problems.
- **thermodyn_properties** is a set of tools for propertiesand correlations used in CHT problems
- **wall_thermal_equilibrium** compute the thermal equilibrium for a 2-layer wall (Metal/ceramic)
- **yk_from_phi** compute the mass fraction set according to equivalence ratio.

3.2 Submodules

3.3 arnica.phys.solid_material module

module to define a solid material for thermal computations

class arnica.phys.solid_material.**SolidMaterial**(*lambda_poly*, *lambda_range*)
Bases: object

define properties of a solid material object

lambda_th(*temperature*)
return the lambda of ceramics material [W/mK]

thermal_resistance(*width*, *t_est*)
return the thermal resistance [m2.K/W] width : width of the layer t_est : estimated temperature of the layer

3.4 arnica.phys.thermodyn_properties module

Module for computing thermodynamic properties

arnica.phys.thermodyn_properties.**thermal_constants**()
Generate a dictionnary of thermal constants

Returns: TYPE: Description

arnica.phys.thermodyn_properties.**h_kader**(*t_wall*, *rho_wall*, *y_wall*, *u_2*, *t_2*, *temp_adiab*)

compute h at the wall as in kader names taken equalt to loglaw_cwm.f90 AVBP

Args: t_wall (TYPE): Description rho_wall (TYPE): Description y_wall (TYPE): Description u_2 (TYPE): Description t_2 (TYPE): Description temp_adiab (TYPE): Description

Returns: TYPE: Description

arnica.phys.thermodyn_properties.**lambda_cp_visco_fluid**(*temperature*)
compute Fluid properties lambda , cp, visco

Args: temperature (TYPE): Description

Returns: TYPE: Description

arnica.phys.thermodyn_properties.**viscosity_sutherland**(*temp*)
compute visocity as in sutherland

Args: temp (TYPE): Description

Returns: TYPE: Description

arnica.phys.thermodyn_properties.**fluid_cp**(*temp*, *clipping=False*)
compute cp of fluid

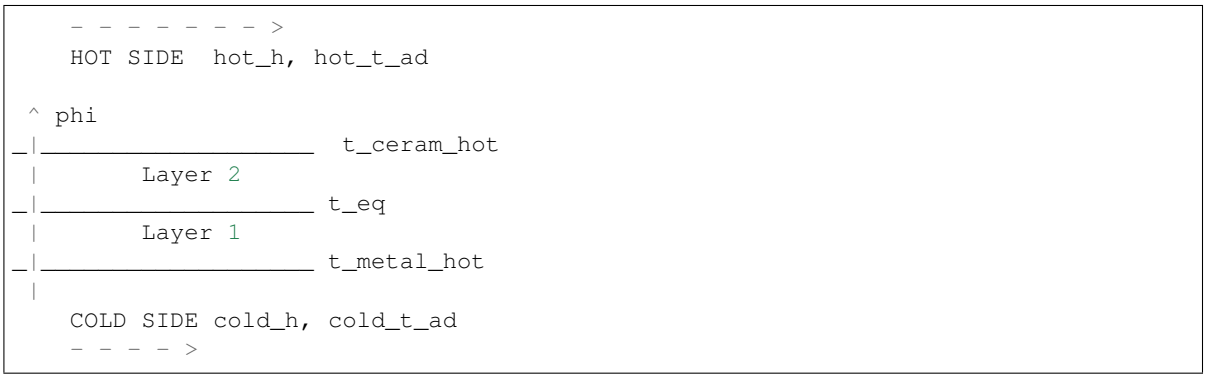
Args: temp (TYPE): Description clipping (bool, optional): Description

Returns: TYPE: Description

3.5 arnica.phys.wall_thermal_equilibrium module

module to compute wall equilibrium

arnica.phys.wall_thermal_equilibrium.**compute_equilibrium**(*hot_t_ad*, *cold_t_ad*, *hot_h*, *cold_h*, *metal*, *ceram*, *ep_metal*, *ep_ceram*)



3.6 arnica.phys.yk_from_phi module

This script calculate mass_fraction of species from a Phi

arnica.phys.yk_from_phi.**yk_from_phi**(*phi*, *c_x*, *h_y*)
Return the mass fraction of elements from a fuel aspect ratio and stoechio element coeff.

Parameters

- **phi** (*float*) – the air-fuel aspect ratio
- **c_x** (*float*) – stoechio coeff of Carbone
- **h_y** (*float*) – stoechio coeff of hydrogene

arnica.phys.yk_from_phi.**phi_from_far**(*far*, *c_x*, *h_y*)
Return phi coefficient with the fuel air ratior coeff + fuel composition.

Parameters

- **far** (*float*) – the air-fuel ratio
- **c_x** (*float*) – stoechio coeff of Carbone
- **h_y** (*float*) – stoechio coeff of hydrogene

CHAPTER 4

arnica.utils package

4.1 The utils

These utils are helpers around CFD-related problems.

- **showy** is a matplotlib helper for using subplots with re-usable templates.
- **show_mat** is a matplotlib helper for fast matrix plotting with legend and axis naming.
- **cloud2cloud** is an inverse distance interpolator without connectivity.
- **directed_projection** is a projection of vectors clouds along their directions.
- **vector_actions** is a set of vector transformation helpers.
- **plot_density_mesh** is a mesh rendering tool using matplotlib hist2d.
- **axi_shell** is a 2D i-j structured mesh mapping axycylindrical splaine-based surfaces.
- **nparray2xmf** is a 1-2-3D i-j-k structured numpy datastructure dumping facility to XDMF format.

4.1.1 Untested - to be deleted :

- **unstructured_adjacency** *unstested* is the beginning of mesh handling using connectivity.
- **mesh_tools** *unstested* is a 2D mesh generation in numpy for solvers
- **datadict2file** was a dumping facility for dictionnary-like data. To be replaced by *hdfdict* or h5py-wrapper* packages.
- **timer_decorator** is a lightweight timer for functions. Better to use cProfile…

4.2 Submodules

4.3 arnica.utils.axipointcloud module

class arnica.utils.axipointcloud.**AxiPointCloud**(*xcoor*, *ycoor*, *zcoor*, *name='Unnamed'*, *vars=None*, *theta_range=None*)

Bases: object

Handle Axysymmetric points clouds as 1D meshs with a dict of variables

dump(*filename*)
Dump an XDMF file of the pointcloud

dupli_rotate(*repeat=1*)
duplicate by rotation around x, in radians

Parameters **repeat** – number or repetitions integer

rad()
Return radius np array from Y and Z

recompute_theta_range_from_coords()
recompute theta range

rotate(*shift_angle*)
rotate around x

Parameters **shift_angle** – angle in radians float

theta()
Return theta np array (radians)

- **range -pi/pi**
  - 0 on the y+ axis (z=0, y>0)

  spanning -pi to pi 0pi=0deg Y ^ | |

-0.5pi=-90deg o——>Z 0.5pi=90deg

vars_stack()
Stacked version numpy of variables

shape: (n, k)

xyz()
Stacked version numpy of coordinates

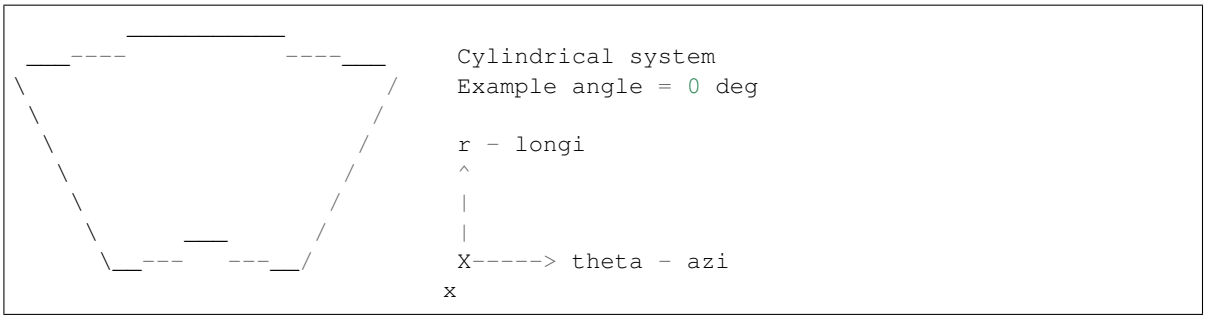
shape: (n, 3)

4.4 arnica.utils.axishell module

axishell module to create x-axisymmetric shells for scientific computations

class arnica.utils.axishell.**AxiShell**(*n_longi*, *n_azi*)
Bases: object

Base class for x-axisymmetric computationnal shells



Parameters

- **n_longi** (*int*) – Number of longitudinal cut points
- **n_azi** (*int*) – Number of azimuthal cut points
- **shape** (*tuple of int(dim 2)*) – Number of cut points n_longi and n_azi
- **geom** – dict() containing the geometrical parameters :
  - **angle** - float - Axi-symmetric angle range
  - **angle_min** - float - Minimum axi-symmetric angle
  - **ctrl_pts_x** - tuple of float - x-component of the spline control point
  - **ctrl_pts_r** - tuple of float - r-component of the spline control point
- **matrix** – dict() containing shell data :
  - **xyz** - np.array of dim (n_azi,n_longi,3) - Array of x,y,z-components
  - **r** - np.array of dim (n_azi,n_longi) - Array of r-component
  - **theta** - np.array of dim (n_azi,n_longi) - Array of theta-component
  - **n_x** - np.array of dim (n_azi,n_longi) - Array of normal x-component
  - **n_y** - np.array of dim (n_azi,n_longi) - Array of normal y-component
  - **n_z** - np.array of dim (n_azi,n_longi) - Array of normal z-component
  - **n_r** - np.array of dim (n_azi,n_longi) - Array of normal r-component
- **cake** – dict() containing 3D mesh from 2D shell extrusion :
  - *xyz* - np.array of dim (n_azi, n_longi, n_layers, 3) - Array of x,y,z-components
  - *dz* - np.array of dim (n_azi, n_longi, n_layers) - Array of width per layer

add_curviwidth (*label*, *points*)
Add a 2D width matrix of shell shape extruded from points spline

Parameters

- **label** (*str*) – Label of the width matrix
- **points** – Tuple (dim n) of tuple (dim 2) of float coordinates

average_on_shell_over_dirs (*variable*, *directions*, *scale=True*)
Performs an integration (averaging) over one or multiple directions

Parameters

- **variable** – A np.array to be averaged of dim (n_time, n_theta, n_r)

- **directions** – A list() of directions on which the average process is to be performed. Contains keywords from ['time','theta','r'].

Returns

- **averaged_variable** - A np.array of averaged data on given directions.

bake_millefeuille(*width_matrix_label*, *n_layers*, *shift=0.0*)
Create a millefeuille-like shell.

Extrude a 2D shell in the normal direction up pointwise height given by "width_matrix_label" matrix.

Parameters

- **width_matrix_label** (*str*) – Label of the width matrix
- **n_layers** (*int*) – Number of layer for extrusion
- **shift** (*float*) – Additional depth (optional)

Returns

- **cake** - A dict() containing shell data :
  - *xyz* - np.array of dim (n_longi, n_azi, n_layers, 3)
  - *dz* - np.array of dim (n_longi, n_azi, n_layers)

"Bon appetit!"

build_shell()
Build shell from geometric features

- Construct a spline used as base for extrusion from control points : tck
- Discretise the spline : shell_crest
- Compute normal vectors for the 1D shell_crest
- Compute r,n_x,n_r-components for 2D shell
- Compute theta-components for 2D shell
- Compute xyz,n_y,n_z-components for 2D shell

dump()
Dump AxiShell geometric features in JavaScript Object Notation

init_mockup()
Initialize with a mockup mesh

load()
Load AxiShell geometric features in JavaScript Object Notation

set_mask_on_shell(*point_cloud*, *tol*)
Create a mask on the shell from a point cloud

The mask value is 1 for shell points located near cloud points.

Parameters

- **point_cloud** (*numpy array*) – Array of dim (n,3) of coordinates of points.
- **tol** (*int*) – Tolerance of proximity

arnica.utils.axishell.**width_mockup**()
Create a mockup tuple of tuple for widths

4.5 arnica.utils.axishell_2 module

4.6 arnica.utils.cartshell_2 module

4.7 arnica.utils.cartshell_3 module

4.8 arnica.utils.cloud2cloud module

interpolate a cloud from an other cloud

arnica.utils.cloud2cloud.**cloud2cloud**(*source_xyz*, *source_val*, *target_xyz*, *stencil=3*, *limitsource=None*, *power=1.0*, *tol=None*)

Interpolate form a cloud to an other

source_xyz : numpy array shape (n_s, 3) either (1000, 3) or (10,10,10, 3) source_val : numpy array shape (n_s, k) of k variables target_xyz : numpy array shape (n_t, 3) stencil (int): nb of neigbors to compute (1 is closest point)

limitsource (int) : maximum nb of source points allowed (subsample beyond) power(float) : Description tol(float) : Description Returns : ———- target_val : numpy array shape (n_t, k)

4.9 arnica.utils.cloud2cloud2 module

interpolate a cloud from an other cloud

Alternate version with Rbf for testing purpose. NOT WORKING YET

arnica.utils.cloud2cloud2.**RbfWeights**(*source*, *target*, *power*, *eps=1e-16*)

arnica.utils.cloud2cloud2.**cloud2cloud2**(*source_xyz*, *source_val*, *target_xyz*, *power=1.0*)

Interpolate form a cloud to an other NOT WORKING YET

source_xyz : numpy array shape (:, 3) or (:,:, 3) or (:,:,:, 3) source_val : numpy array shape (:, 3) or (:,:, k) or (:,:,:, k) k variables , first dims equal to source mesh. target_xyz : numpy array shape (:, 3) or (:,:, 3) or (:,:,:, 3) stencil (int): nb of neigbors to compute (1 is closest point)

limitsource (int) : maximum nb of source points allowed (subsample beyond) power(float) : Description tol(float) : Description Returns : ———- target_val : numpy array shape (:, 3) or (:,:, 3) or (:,:,:, 3) , first dims equal to target mesh.

4.10 arnica.utils.curve_feat_extract module

Extract features from a curve

arnica.utils.curve_feat_extract.**amax_keepshape**(*arr*, *axis*)

expand “row” POS from axis AXIS of an array ARR everywhere else

Args: arr (np.array): source array axis (int): index of axis to expand

Returns: np.array: amax on the same shape

```
>>> a = np.array([[0, 1, 0, 0, 0],
                  [0, 1, 1, 1, 1],
                  [0, 1, 2, 3, 3]])
>>> amax_keepshape(a, axis=1, pos=3)
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [3, 3, 3, 3, 3]])
```

arnica.utils.curve_feat_extract.**amin_keepshape**(*arr*, *axis*)

expand "row" POS from axis AXIS of an array ARR everywhere else

Args: arr (np.array): source array axis (int): index of axis to expand

Returns: np.array: amin on the same shape

```
>>> a = np.array([[0, 1, 0, 0, 0],
                  [0, 1, -1, 1, 1],
                  [1, 1, 2, 3, 4]])
>>> amin_keepshape(a, axis=1, pos=3)
array([[0, 0, 0, 0, 0],
       [-1, -1, -1, -1, -1],
       [1, 1, 1, 1, 1]])
```

arnica.utils.curve_feat_extract.**mask_boundary_layer**(*source*, *eps=0.1*, *gain=1.0*, *axis=0*)

Identify the cells relative to a boundary layer in a numpy multi-d array. If no fluctuation detected, only the "wall" node is set to 1.

Args: source (np.array): source array eps (float):]0,1[relative fluctuation thresold (0.1 is 10% of max curve amplitude) gain (float): >0 increase of decrease the width of the layer found axis (int): index of axis to search

Returns: np.array: masked array equal to one on the fluctuation side.

```
>>> a = np.array(
                [[ 0,  1,  0,  0,  0]
                 [ 0,  1,  1,  1,  1]
                 [ 0,  3,  2,  3,  4]
                 [ 0,  1, -2,  3,  4]
                 [ 0,  0,  0,  0,  0]
                 [ 0,  1,  2,  3,  4]]
    )
>>> mask_boundary_layer(ource, eps=0.1, axis=1)
array(
    [[1, 1, 0, 0, 0],
    [1, 0, 0, 0, 0],
    [1, 1, 0, 0, 0],
    [1, 1, 1, 0, 0],
    [1, 0, 0, 0, 0],
    [1, 0, 0, 0, 0]]
)
```

arnica.utils.curve_feat_extract.**reptile**(*arr*, *axis*, *pos*)

expand "row" POS from axis AXIS of an array ARR everywhere else

Args: arr (np.array): source array axis (int): index of axis to expand pos (int): index of element to expand

Returns: np.array: Expanded array

```
>>> a = np.array([[0, 1, 0, 0, 0],
                  [0, 1, 1, 1, 1],
                  [0, 1, 2, 3, 4]])
>>> reptile(a, axis=1, pos=3)
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [3, 3, 3, 3, 3]])
```

4.11 arnica.utils.datadict2file module

module to data array-like dictionnary to files for visulaisation or storage pupopses

arnica.utils.datadict2file.**dump_dico_0d**(*filename*, *data_dict*)

Write statistics to file

filename [the file name to which array dictionnary are dumped]

possible extensions [- .xlsx (if pandas is found)]

- .csv (default format)

data_dict : a dictionnary holding the data arrays

None

arnica.utils.datadict2file.**dump_dico_1d_nparrays**(*filename*, *data_dict*)

Write statistics to file

filename [the file name to which array dictionnary are dumped]

possible extensions [- .xlsx (if pandas is found)]

- .csv (default format)

data_dict : a dictionnary holding the data arrays

None

arnica.utils.datadict2file.**dump_dico_2d_nparrays**(*data_dict*, *filename*, *x_coords*, *y_coords*, *z_coords*, ***kw_args*)

data_dict [dictionnary holding the 2d arrays, on the format] data_dict[key] = array(n1, n2) where (n1, n_2) is a subset of (n_x, n_y, n_z)

filename : the xmf filename

<x|y|z>_coords [2d numpy arrays for coordiantes over each axis] must be of shape (n_1, n_2)

time [physical time corresponding to the array] used in the xmf file as <Time Value="time"....

grid_name [the name of the grid to be used in the xmf file] as <Grid Name="grid_name"....

domain_name [the name of the domain to be used in the xmf file] as <Domain Name=domain_name....

None

arnica.utils.datadict2file.**dump_dico_2d_time_nparrays**(*data_dict*, *root_path*, *prefix*, *x_coords*, *y_coords*, *z_coords*, ***kw_args*)

Dumps a dictionnary of time series 2d arrays to xmf files

data_dict [dictionnary holding the times series 2d] arrays, on the format data_dict[key] = array(n_time, n1, n2) where:

- n_time is the number of time steps
- (n1, n_2) is a subset of (n_x, n_y, n_z)

prefix : the prefix to be used to generate xmf filenames

<x|y|z>_coords [2d numpy arrays for coordiantes over each axis] must be of shape (n_1, n_2)

steps [a list of integer time series steps] that will be used to generate xmf files on the format : <prefix>_<step>.xmf if None steps will be generated as the range of time dimension of data arrays

times [a list of float physical times that will] be used in xmf files to describe the time of each step. if None will be generated as the range of time dimension of data arrays

arnica.utils.datadict2file.**dump_dict2xmdf**(*filename*, *grid*, *data_dict*)

Dump 2D matrices into hdf5 file

Parameters

- **filename** (*str*) – Name of the hdf file
- **grid** – Array of xyz-coordinates of shape (n_v, n_u, 3)
- **data_dict** – Dict of field arrays of shape (n_v, n_u)

arnica.utils.datadict2file.**plot_dict_data_as_file**(*data_dict*, *filename*, *x_data*, *y_data*, ***kw_args*)

Generates and write XY-plot to file

filename [the file to which the plot is written it contains] the extension that defines the format e.g: 'plot_toto.png' Supported formats/extensions : png, pdf, ps, eps and svg If not provided, by default "pdf" extension is used.

data_dict : a dictionnary holding the data arrays x_data : the key to the array holding the abscissa data y_data : the key to the array holding the y data

x_label : label of the x axis, by default x_data is used (supports latex) y_label : label of the y axis, by default y_data is used (supports latex)

4.12 arnica.utils.directed_projection module

Module to compute the directed projection of a point to a surface along a direction

```
___
                                x---->
```

OST : Seven nation Army (Westworld), R. Djawadi

arnica.utils.directed_projection.**compute_dists**(*points*, *directions*, *points_surf*, *normals_surf*, *tol*)

Compute cylindrical distances

For a i-number of points coordinates, compute the cylindrical distances between the i,p-number of nodes with the i-number of axis.

The array is then clipped according to the node normals and the direction of the drills.

Parameters

- **points** (*np.array*) – Array of dim (i,3) of drill float coordinates
- **directions** (*np.array*) – Array of dim (i,3) of drill float axis components
- **points_surf** (*np.array*) – Array of dim (i,p,3) of nodes float coordinates
- **normals_surf** (*np.array*) – Array of dim (i,p,3) of nodes float normal components
- **params** – Dict of parameter

Returns

- **cyl_dists** - Array of dim (i,p) of float cylindrical distances

arnica.utils.directed_projection.**intersect_plan_line**(*xyz_line*, *vec_line*, *xyz_plan*, *nml_plan*)

Compute intersection coordinates of a line and a plan

- Line defined by a point *xyz_line* and a vector *vec_line*
- Plan defined by a point *xyz_plan* and a normal *nml_plan*

Arrays dimensions must be consistent together.

::

nml_plan xyz_line A x | / | / vec_line

___x_____I_____ xyz_plan

Intersection point

Parameters

- **xyz_line** – Array of coordinates of shape (3,) or (n, 3)
- **vec_line** – Array of components of shape (3,) or (n,3)
- **xyz_plan** – Array of coordinates of shape (3,) or (n,3)
- **nml_plan** – Array of complonents of shape (3,) or (n,3)

Returns **xyz_intersect** - Array of coordinates of shape (3,) or (n,3)

arnica.utils.directed_projection.**project_points**(*points_source*, *normals*, *points_target*)

Compute the projected points, radial dists and axial dists

Compute projection from source points S of dim (k,) or (i,k),on a plan defined by normals N of dim (k,), (i,k), (p,k) or (i,p,k),and points T of dim (k,), (i,k), (p,k), (i,p,k). With :

- **i** : Number of points to project
- **p** : Number of points defining plans
- **k** : Dimension of the domain

```
    S             T1           T2
     \           ⟋            ⟋
      \         ⟋            ⟋
axi_d1 \ ⟋\ ⟋ rad_d1     ⟋
        \ ⟋            ⟋ rad_d2
         \           ⟋
          \         ⟋
    axi_d2 \ ⟋\ ⟋
            \ ⟋
             X proj_pt2
```

(continues on next page)

(continued from previous page)

```
                \
                 N
```

S : Points source N : Normal T : Points target axi_d : Axial distance of the point T projected on the axis Ax rad_d : Cylindrical or Radial distance between T and the axis Ax

——> ->

axi_dist = (T - S) . N

————————> -> -> projected_point = S + N * axi_dist

-> ————————>

rad_dist = norm(T - projected_point)

Parameters

- **points_source** (*np.array*) – Array of source points coordinates
- **proj_axis** (*np.array*) – Array of normal components defining projection plans
- **points_target** (*np.array*) – Array of points coordinates defining projection plans

Returns

- **projected_points** - Array of shape points_target.shape of float coordinates
- **axi_dists** - Array of shape points_target.shape[:-1] of float distances
- **rad_dists** - Array of shape points_target.shape[:-1] of float distances

arnica.utils.directed_projection.**projection_kdtree**(*points*, *directions*, *point_surface*, *normal_surface*, ***kwargs*)

Project the n points following the direction on the m suface.

Parameters

- **points** – Array of [p] points of shape (p,3)
- **directions** – Array of [p] direction vectors of shape (p,3)
- **point_surface** – Array of [n] surface nodes of shape (n,3)
- **normal_surface** – Array of [n] surface node normals of shape (n,3)
- **neigbors** (*int*) – Number [k] of neigbors to take into account
- **tol** (*float*) – Maximum distance beyond cyl dist with big set to BIG
- **project** (*bool*) – If True, first project points along normal.

Returns

- **projected_pts** - “t” nparray of shape (n,3), projected on the surface
- **indexes** - neigborhood of the points (n,k)
- **cyl_dist** - cylindrical distance of p with each neighbor (n,k)

```
       < shp_dist >
______s___________t___________________
      |' .                        A
      |     '  .< cyl_dist >      .
      v                           .< surface_dist>
             4                    .
```

(continues on next page)

(continued from previous page)

```
          /                    .
         /                     .
        p                      V

    align : alignment (pscal of two unit vectors, in [-1,1])
```

Algorithm :

- If project bool is True, first compute [p] projection from [p] points along the [p,1] spherical closest node's normal. If False, projected_points = points.
- Reduce computation to the [p,k] spherical closest nodes of the [p] projected_points
- Compute the [p,k] cylindrical distances from the [p,k] closest nodes to the [p] lines defined by the [p] projected points and the [p] direction vectors.

4.13 arnica.utils.lay_and_temp_manager module

4.14 arnica.utils.mesh_tool module

This module contains function to create and modify meshes

arnica.utils.mesh_tool.**dilate_center**(*x_coor*, *y_coor*, *perturbation=0.1*)
perturb cartesian mesh dilatation in the center

x_coor : numpy array (n,m) , x_coordinates y_coor : numpy array (n,m) , y_coordinates perturbation : float, amplitude of the perturbation perturbation

with respect to the grid size

x_coor : numpy array (n,m) , x_coordinates shifted y_coor : numpy array (n,m) , y_coordinates shifted

arnica.utils.mesh_tool.**gen_cart_grid_2d**(*gridrange*, *gridpoints*)
Generate cartesian grid.

gridrange : tuple of floats, dimensions of the grid gridpoints : tuple of ints (n,m), sampling on the grid

x_coor, y_coor : numpy arrays (n,m) with coordinates

arnica.utils.mesh_tool.**gen_cyl_grid_2d**(*r_min*, *r_max*, *r_points*, *theta_min*, *theta_max*, *theta_points*)
Generate a cylindrical grid center on x = 0 and y = 0

r_min : inner radius r_max : outer radius theta_min : lower angle [0, 2 * pi] theta_max : upper angle [0, 2 * pi] r_points : number of points in the radial direction theta_points : number of points in the tangential direction

x_coor : x coordinates of the mesh y_coor : y coordinates of the mesh

arnica.utils.mesh_tool.**get_mesh**(*params_mesh*)
Call specific meshing functions from mesh parameters dict

params_mesh: dictionary containing mesh parameters

x_coor: x coordinates of the mesh y_coor: y coordinates of the mesh

4.15 arnica.utils.nparray2xmf module

module to create an ensight compatible file to visualize your data

class arnica.utils.nparray2xmf.**NpArray2Xmf**(*filename*, *domain_name=None*, *mesh_name=None*, *time=None*, *xmf_only=False*)

Bases: object

main class for data output in XDMF format

add_field(*nparray_field*, *variable_name*)
add a field, assuming same shape as nparray of coordiantes

create_grid(*nparray_x*, *nparray_y*, *nparray_z*)
create the grid according to numpy arrays x, y ,z if arrays are 1D, switch to cloud point if arrays are 2D, switch to quad connectivity if arrays are 3D, switch to hexaedrons connectivity

dump()
dump the final file

xmf_dump()
create XDMF descriptor file

arnica.utils.nparray2xmf.**create_time_collection_xmf**(*collection_filenames*, *xmf_filename*)

Creates xmf file holding time collection of xmf files

collection_filenames: a list of single time xmf filenames to collect xmf_filename : the name of the output file

None

4.16 arnica.utils.plot_ave_with_interval module

Plot graphs from 1D average array with or without its confidence interval. Rotate the graph from 90 deg.

arnica.utils.plot_ave_with_interval.**plot_ave_with_interval**(*x_arr*, *average*, *profile='average-interval'*, *upper=None*, *lower=None*, ***kw_args*)

Plot average profile with or without confidence interval

Parameters

- **x_arr** (*np.array*) – Array of float of x-axis
- **average** (*np.array*) – Array of float of average curve
- **profile** (*str*) – Plot type (average-interval, average, integral)
- **upper** (*np.array*) – Array of float of upper interval values
- **lower** – Array of float of lower interval values

Optional Keyword Args:

Parameters

- **x_label** (*str*) – Label for x-axe

- **y_label** (*str*) – Label for y-axe
- **style** (*str*) – Style of the axes - plain or sci

Returns

- **plt** - Matplotlib.pyplot object

4.17 arnica.utils.plot_density_mesh module

Plot density mesh module

arnica.utils.plot_density_mesh.**heat_map_mesh**(*x_crd*, *y_crd*, *z_crd*, *show=False*, *save=False*, *view_axes='xr'*)

heat map plot of skin

arnica.utils.plot_density_mesh.**scatter_plot_mesh**(*x_crd*, *y_crd*, *z_crd*, *axisym=False*, *show=False*)

scatter plot of skin

4.18 arnica.utils.plot_quad2tri module

Module to plot a field on cartesian mesh through a triangular mesh

arnica.utils.plot_quad2tri.**get_connectivity**(*i*, *j*, *shape*)

Generate connectivities

Parameters

- **i** (*int*) – Index of the cell on u-axis
- **j** (*int*) – Index of the cell on v-axis
- **shape** – Shape of the shell

Returns

- **connectivity** - Array of shape (t, 3) of triangle point indexes

arnica.utils.plot_quad2tri.**plot_quad2tri**(*grid*, *title*, *field*, *shading=True*)

Generate a plot of a field on a x,y shell cartesian grid

The cartesian grid is converted into a triangle grid with connectivities. The field is extended by interpolation on the new points created. The plot is generated from the points coordinates extended and the connectivity, using tripcolor of matplotlib.

If shading is True : using 'gouraud' from field value defined at nodes. If shading is False : using 'flat' from mean field value defined at the nodes or

field value defined at the cell.

Parameters

- **grid** – Array of shape (n_u, n_v, 2) of cartesian coordinates
- **title** (*str*) – Title of the plot
- **field** – Array of shape (n_u, n_v) of field values
- **shading** (*bool*) – Define if plot is shaded of not

Returns

- **plt** - Matplotlib object containing the graph

::

1 quad 4 tri

O———O O———O | | | / | | | | / | | | | / | | | —> | O | | | | / | | | | / | | | | / | O———O O———O

arnica.utils.plot_quad2tri.**quad2tri**(*grid*, *field*)
Divide a cartesian grid into an extended triangular grid

Parameters **grid** – Array of shape (n_u, n_v, 2) of cartesian coordinates

:param field Array of shape (n_u, n_v) of field values

Returns

- **triangulation** - Matplotlib.tri.Triangulation object containing extended grid coordinates and connectivity array
- **field_at_node** - Array of extended field values stored at the node

4.19 arnica.utils.plotcsv module

4.20 arnica.utils.same_nob module

arnica.utils.same_nob.**h5_same**(*source_file: str*, *target_file: str*, ***kwargs*)
Main call function to test two h5 files.

Parameters

- **source** – Path to the source file to compare
- **target** – Path to the target file to compare

Returns **True** if files are identical, **False** otherwise

arnica.utils.same_nob.**dict_same**(*source_dict: dict*, *target_dict: dict*, ***kwargs*)
Main call function to test two dictionaries.

Parameters

- **source_dict** – first dict to compare
- **target_dict** – second dict to compare

Returns **True** if files are identical, raises an error otherwise

arnica.utils.same_nob.**h5dict_safe**(*file: str*) → dict
h5 files safe loader with hdfdict. :param file: str ou os.Path menant au h5 à lire

4.21 arnica.utils.sample_curve_from_cloud module

SORT AND SAMPLE POINTS FROM CURVE

arnica.utils.sample_curve_from_cloud.**get_neighbor**(*kdtree*, *point*, *list_points*)
Find the closiest neighbor that has not already been found

From the kdtree, find the two closiest points of 'point', the 1st being itself. If the second closiest point has already been found, then the number of researched points is increased until finding a point that has not already been found.

Parameters

- **kdtree** – KDTree of all points.
- **point** (*np.array*) – Array of dim (k,) of point float coordinates.
- **list_points** – List of integer indexes

Returns

- **index** - Index (int) of the unfound closiest point.
- **dist** - Distance (float) of the closiest point from point.

arnica.utils.sample_curve_from_cloud.**sample_arrays_by_dist_interval**(*dists*, *samples_res*, **args*)

Sample data and optional args at each sample_res along data

From an array containing the distance between sorted points, and from a sample resolution defined by the distance betweentwo samples, the function picks up the indexes correspondingto a sample, and returns an array of sampled distances andarrays of sampled arguments from those indexes.

::

. . . . . .__.__.__.__.__.

. . . . .__ ..

. . | . . . . . ===> | .__.__.__.__.__. . . . .

. . . . .__.__.__.__.__.__. .. ..

. . . . . 9 .__.__.__.__.__.__.__. 9 321 3 2 1

Parameters

- **dists** (*np.array*) – Array of dim (n,) of floats
- **samples_res** (*float*) – Sample resolution for sampling
- **args** – Tuple de data of dim (n,)

arnica.utils.sample_curve_from_cloud.**sample_points_from_cloud**(*points_coor*, *starting_pt*, *n_samples=None*, *samples_res=None*)

Sort and sample unsorted curve

First the point coordinates are sorted by distances. Secondly compute sample resolution if not provided. Finally sample the ordered indexes points by distance. Returns the array of coordinates ordered and sampled.

Parameters

- **points_coor** – Array of dim (n,k) of points coordinates
- **starting_pt** (*np.array*) – Array of dim (k,) of starting point coordinates
- **n_samples** (*int*) – Number of samples to extract

- **sample_res** (*float*) – Resolution of the sampling

Returns

- **skin_cyl** - Array of dim (n_samples, 3) of coordinates

arnica.utils.sample_curve_from_cloud.**sort_points_by_dist**(*points_coor*, *starting_pt*)

Reorder a point cloud by distances

.

From a starting fictive point, the first point 1 of the splineiget_neighbors found, as the closiest one. . From that point, the following points are obtained with get_neighbor() until having sorted all points.

::

. . . . . r/y

. . . 2 8 .. . . ^

. . . . | x . . . . . ===> . . . . . o—–> . . . .

. . . .

. . . . . .

1 3 X 8 321 starting_pt

Parameters

- **points_coor** (*np.array*) – Array of dim (n,k) of points coordinates
- **starting_pt** (*np.array*) – Array of dim (k,) of starting pt coordinates

Returns

- **ordered_indexes** - Array of dim (n,) of indexes
- **ordered_dists** - Array of dim (n,) of floats

4.22 arnica.utils.shell module

4.23 arnica.utils.show_mat module

This script contains function to properly visualize matrices

arnica.utils.show_mat.**filter_stupid_characters**(*string*)

Delete and replace stupid characters to save the figure

string: title of the plot to be changed into the filename

cleaned string

arnica.utils.show_mat.**show_mat**(*matrix*, *title*, *show=True*, *save=False*)

Show and/or save a matrix visualization.

matrix: 2d matrix title: Title of the plot show: Boolean to show the plot or not save: Boolean to save the plot or nor (automatic name from title)

None

4.24 arnica.utils.showy module

arnica.utils.showy.**showy**(*layout*, *data*, *data_c=None*, *show=True*)

arnica.utils.showy.**display**(***kwargs*)
Retro compatibility

4.25 arnica.utils.string_manager module

string_manager.py

Functions which deal with strings

4.26 arnica.utils.timer_decorator module

small timer function

arnica.utils.timer_decorator.**timing**(*function*)
lazy method to time my function

4.27 arnica.utils.unstructured_adjacency module

Efficient implementation of unstructured mesh operations

Created Feb 8th, 2018 by C. Lapeyre (lapeyre@cerfacs.fr)

class arnica.utils.unstructured_adjacency.**UnstructuredAdjacency**(*connectivity*)
Bases: object

Efficient scipy implementation of unstructured mesh adjacency ops

The connectivity is stored in a sparse adjencency matrix A of shape (nnode, ncell). The gather operation on vector X (nnode) yields the scattered vector Y (ncell), and the scatter operation yields the filtered vector X' (nnode). This writes:

Y = 1/nvert . A . X X' = 1/bincount . A^t . Y

where ^t is the transpose operation.

The gatter-scatter operation resulting in filtering X can be performed efficiently by storing:

F = 1/bincount . A^t . 1/nvert . A X' = F . X

get_cell2node()
Return the cell2node function

get_filter(*times=1*)
Return the full gather + scatter filter operation

If you need to perform the operation N times, you can use the times attribute.

get_node2cell()
Return the node2cell function

ncell
Total number of cells

ncell_per_nvert
Dictionary of {nvert: ncell}

For each type of element with nvert vertices, stores the nubmer of cells ncell

4.28 arnica.utils.vector_actions module

Module concerning some 3D vector manipulations in numpy

OST :Mercy in Darkness, by Two Steps From Hell

arnica.utils.vector_actions.**angle_btw_vects**(*np_ar_vect1*, *np_ar_vect2*, *convert_to_degree=False*)
compute the angle in deg btw two UNIT vectors

arnica.utils.vector_actions.**cart_to_cyl**(*vects_xyz*)
Transform vects from xyz-system to xrtheta-system

x -> x : x = x y -> r : r = sqrt(y^2 + z^2) z -> theta : theta = arctan2(z,y)

Parameters **vects_xyz** (*np.array*) – Array of dim (n,3) of xyz components

Returns

- **vects_cyl** - Array of dim (n,3) of xrtheta components

arnica.utils.vector_actions.**clip_by_bounds**(*points_coord*, *bounds_dict*, *keep='in'*, *return_mask=False*)
Clip a cloud by keeping only or removing a bounded region

The dict to provide must be filled as follow : bounds_dict = {component_1 : (1_min, 1_max),

compoment_2 : (2_min, 2_max), . . . }

component_1 = [“x”, “y”, “z”, “r”, “theta”]

The bounded region can either be :

- A 1D slice if only 1 component is provided ;
- A 2D box if 2 components are provided ;
- A 3D box if 3 components are provided.

If keep=”in”, returns the point coordinates inside the bounds. If keep=”out”, returns the point coordinates outside the bounds.

If returns=True, returns the coordinates clipped If returns=False, returns the mask of boolean than can be applied on other arrays

Parameters

- **point_cloud** (*np.array*) – Array of dim (n,k) of coordinates
- **bounds_dict** – Dict of MAX lengh k of tuple of floats
- **keep** (*str*) – Either keeps what is inside or outside

Returns

- **points_coord_clipped** - Array of dim (m,k) with m<=n

OR - **mask** - Array of dim (n,) of booleans

arnica.utils.vector_actions.**cyl_to_cart**(*vects_cyl*)
Transform vects from xrtheta-system to xyz-system

x -> x : x = x r -> y : y = r * cos(theta) theta -> z : z = r * sin(theta)

Parameters **vects_cyl** (*np.array*) – Array of dim (n,3) of xrtheta components

Returns

- **vects_xyz** - Array of dim (n,3) of xyz components

arnica.utils.vector_actions.**dilate_vect_around_x**(*azimuth*, *np_ar_vect*, *angle_deg_init=None*, *angle_deg_targ=360*)

dilate vectors around axis x from a specified initial range angle to a target range angle.

np_ar_vect : numpy array of dim (n,3) angle_deg_targ : tuple or float

numpy array of dim (n,3)

arnica.utils.vector_actions.**make_radial_vect**(*coord*, *vects*)
Recalibrate vectors to make them radial.

The vectors are readjusted to cross x-axis. It is mainly done for nodes on the limit of the boundary for axi-cylindrical geometries.

Parameters

- **coord** (*np.array*) – Array of dim (n,3) of float coordinates
- **vects** (*np.array*) – Array of dim (n,3) of float components

Returns

- **radial_vect** - Array of dim (n,3) of float components

arnica.utils.vector_actions.**mask_cloud**(*np_ar_xyz*, *axis*, *support*)
mask a cloud of n 3D points in in xyz axis among x,y,z,theta,r support a 2 value tuple : (0,3), (-12,float(inf))

x and (0,3) reads as 0 <= x < 3 (lower bound inclusive) z and (-12,float(inf)) reads as -12 <= z theta in degree, cyl. coordinate around x axis - range -180,180 - 0 on the y+ axis (z=0, y>0)

arnica.utils.vector_actions.**renormalize**(*np_ar_vect*)
renormalize a numpy array of vectors considering the last axis

arnica.utils.vector_actions.**rotate_vect_around_axis**(*xyz*, **tuples_rot*)
Rotate vector around vector or series of vector

Parameters

- **xyz** – Array of xyz-coordinates of shape (n,3)
- **tuples_rot** – List of tuple with rotation data : Axis array of shape (3,) axis, Float angle in degree

Returns Array of rotated xyz-coordinates of shape (n,3)

arnica.utils.vector_actions.**rotate_vect_around_x**(*np_ar_vect*, *angle_deg*)
rotate vector around axis x in degree

arnica.utils.vector_actions.**rtheta2yz**(*rrr*, *theta*)

return yz fror rtheta , theta in radians measure of ange in the yz plane, - range -pi/pi - 0 on the y+ axis (z=0, y>0) spanning -pi to pi

```
                    0pi=0deg
                  Y
                  ^
                  |
                  |
-0.5pi=-90deg     o------>Z   0.5pi=90deg
```

arnica.utils.vector_actions.**vect_to_quat**(*vect_targ*, *vect_source*)

Generate a quaternion from two vectors

A quaternion is a rotation object. From two vectors, the rotation angle and the rotation axis are computed. The rotation vector generates then a quaternion for each serie of vectors.

Parameters

- **vect_targ** (*np.array*) – Array of dim (n,3) of vect components
- **vect_source** (*np.array*) – Array of dim (n,3) of vect components

Returns

- **quat** - Array of quaternion of dim (n,)

arnica.utils.vector_actions.**yz_to_theta**(*np_ar_vect*)

return theta , a radians measure of ange in the yz plane,

- range -pi/pi
- 0 on the y+ axis (z=0, y>0)

spanning -pi to pi

0pi=0deg

-0.5pi=-90deg o——>Z 0.5pi=90deg

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

R

S

T

U

V

W

X

Y