Arnica Documentation

Team COOP

Nov 26, 2021

Contents

1 Introduction 3

2 Composition 5

3 arnica.phys package 7
- 3.1 Physical utilities 7
- 3.2 Submodules 7
- 3.3 arnica.phys.solid_material module 7
- 3.4 arnica.phys.thermodyn_properties module 8
- 3.5 arnica.phys.wall_thermal_equilibrium module 8
- 3.6 arnica.phys.yk_from_phi module 9

4 arnica.utils package 11
- 4.1 The utils 11
- 4.2 Submodules 12
- 4.3 arnica.utils.axishell module 12
- 4.4 arnica.utils.cloud2cloud module 14
- 4.5 arnica.utils.data_avbp_as_ptcloud module 14
- 4.6 arnica.utils.datadict2file module 14
- 4.7 arnica.utils.directed_projection module 16
- 4.8 arnica.utils.lay_and_temp_manager module 19
- 4.9 arnica.utils.mesh_tool module 19
- 4.10 arnica.utils.nparray2xmf module 20
- 4.11 arnica.utils.plot_ave_with_interval module 20
- 4.12 arnica.utils.plot_density_mesh module 21
- 4.13 arnica.utils.show_mat module 21
- 4.14 arnica.utils.showy module 21
- 4.15 arnica.utils.string_manager module 25
- 4.16 arnica.utils.temporal_analysis_tools module 25
- 4.17 arnica.utils.timer_decorator module 29
- 4.18 arnica.utils.unstructured_adjacency module 29
- 4.19 arnica.utils.vector_actions module 30

5 Indices and tables 33

Python Module Index 35

Index 37

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*`)`

```
    - - - - - - - - >
    HOT SIDE  hot_h, hot_t_ad

 ^ phi
_|___________________  t_ceram_hot
 |       Layer 2
_|___________________ t_eq
 |       Layer 1
_|___________________ t_metal_hot
 |
    COLD SIDE cold_h, cold_t_ad
    - - - - >
```

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.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

```
          ___________
 ___----             ----___       Cylindrical system
\                           /      Example angle = 0 deg
 \                         /
  \                       /        r - longi
   \                     /         ^
    \                   /          |
     \       ___       /           |
      \___---   ---___/            X-----> theta - azi
                                  x
```

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

4.4 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)

4.5 arnica.utils.data_avbp_as_ptcloud module

module loading avbp h5py into numpy arrays, limited to point cloud (connectivity sucks, mark my word, really)

class arnica.utils.data_avbp_as_ptcloud.**AVBPAsPointCloud**(*meshfile*)
Bases: object

class handling mesh and solutions as point cloud *no connectivity asked*

get_skinpts(*listpatch*)
return a dict of numpy array [x, y ,z] coordinates of a subset of patches

load_avgsol(*solavgfile*, **extra_vars*)
load a solution AVBP avg for the moment

load_mesh_bnd(*patchlist=None*)
load only the boundaries. Load all patches, unless a subset of patch is provided with opt keyword patchlist

load_mesh_bulk()
load the bulk of the mesh, withound the boundaries

4.6 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)

4.7 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
              \
               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                    .
            /                     .
           /                      .
          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.

4.8 arnica.utils.lay_and_temp_manager module

lay_and_temp_manager.py

Functions which deal with layouts and templates

`arnica.utils.lay_and_temp_manager.`**`fetch_avbp_layouts`**`()`

It returns all the avbp layouts in a dictionary

avbp_layouts : nested object

`arnica.utils.lay_and_temp_manager.`**`fetch_avbp_templates`**`()`

It returns all the avbp templates in a dictionary

avbp_templates : nested object

`arnica.utils.lay_and_temp_manager.`**`decompact_template`**`(`*template*, *data*`)`

It decompacts the provided template in function of the provided data which are in the form of a key-value object and returns it

template : nested object data : key-value object

layout : nested object

4.9 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

4.10 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

4.11 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

4.12 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

4.13 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

4.14 arnica.utils.showy module

showy.py

4.14.1 Showy

SHOWY in *arnica/utils* is a helper for matplotlib subplots. If the data is stored as a disct, the layout can be saved as a template in .yml format. If can use wildcards if the dictionary keys allows it.

Simple example

```python
import numpy as np
from arnica.utils.matplotlib_display import showy


def showy_demo_plain():
    data = dict()
    data["time"] = np.linspace(0, 0.1, num=256)

    data["sine_10"] = np.cos(data["time"] * 10 * 2 * np.pi)
    data["sine_30"] = np.cos(data["time"] * 30 * 2 * np.pi)
    data["sine_100"] = np.cos(data["time"] * 100 * 2 * np.pi)
    data["sine_100p1"] = 1. + np.cos(data["time"] * 100 * 2 * np.pi)

    # Creating a template
    layout = {
        "title": "Example",
        "graphs": [
            {
                "curves": [{"var": "sine_10"}],
                "x_var": "time",
                "y_label": "Fifi [mol/m³/s]",
                "x_label": "Time [s]",
                "title": "Sinus of frquency *"
            },
            {
                "curves": [{"var": "sine_30"}],
                "x_var": "time",
                "y_label": "Riri [Hz]",
                "x_label": "Time [s]",
                "title": "Second graph"
            },
            {
                "curves": [
                    {
                        "var": "sine_100",
                        "legend": "origin",
                    },
                    {
                        "var": "sine_100p1",
                        "legend": "shifted",
                    }
                ],
                "x_var": "time",
                "y_label": "Loulou [cow/mug]",
                "x_label": "Time [s]",
                "title": "Third graphg"
            }
        ],
        "figure_structure": [3, 1],
        "figure_dpi": 92.6
    }

# Displaying the data described in the new created layout
showy(layout, data)
```

Using wildcard ' * '

In showy you can show all the graphs with a same prefix putting a "*". For example if you have 3 variables like var_1, var_2, var_3 you can just write var_*. An example is shown below:

```
import numpy as np
from arnica.utils.matplotlib_display import display

def showy_demo_wildcards():
    data = dict()
    data["time"] = np.linspace(0, 0.1, num=256)

    freq = 10.
    for freq in np.linspace(10, 20, num=9):
        data["sine_" + str(freq)] = np.cos(data["time"]*freq*2*np.pi)

    # Creating a template
    template = {
        "title": "Example",
        "graphs": [{
            "curves": [{"var": "sine_*"}],
            "x_var": "time",
            "y_label": "Sine [mol/m³/s]",
            "x_label": "Time [s]",
            "title": "Sinus of frquency *"
        }],
        "figure_structure": [3, 3],
        "figure_dpi": 92.6
    }

    showy(template, data)
```

Options available

The scheme that showing all the options available is shown below.

```
title: Layout scheme
description: The structure that a layout has to respect in order to be used to plot
  data with Showy
type: object
properties:
  title:
    description: The title of the layout
    type: string
  graphs:
    description: The graphs of the layout
    type: array
    items:
      description: A graph
      type: object
      properties:
        curves:
          description: The curves of the graph
          type: array
          items:
            description: A curve
```

(continues on next page)

(continued from previous page)

```
            type: object
            properties:
              var:
                description: The name of the data for the Y-axis
                type: string
              legend:
                description: The legend of the curve
                type: string
            required:
            - var
            additionalProperties: false
          minItems: 1
        x_var:
          description: The name of the data for the X-axis
          type: string
        y_label:
          description: The label of the Y-axis
          type: string
        x_label:
          description: The label of the X-axis
          type: string
        title:
          description: The title of the graph
          type: string
      required:
      - curves
      - x_var
      additionalProperties: false
    minItems: 1
  figure_dpi:
    description: The number of dots per inch of the figure
    type: number
    exclusiveMinimum: 0
  figure_size:
    description: The size of the figure in inches
    type: array
    items:
      description: A length in inches
      type: number
      exclusiveMinimum: 0
    minItems: 2
    maxItems: 2
  figure_structure:
    description: The numbers of rows and columns of graphs
    type: array
    items:
      description: An integer for a number of rows or columns
      type: integer
      minimum: 1
    minItems: 2
    maxItems: 2
required:
- graphs
additionalProperties: false
```

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

It displays the desired graphs described by the provided layout thanks to the provided key-value object which contains the required data

data : key-value object layout : nested object

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

4.15 arnica.utils.string_manager module

string_manager.py

Functions which deal with strings

4.16 arnica.utils.temporal_analysis_tools module

arnica.utils.temporal_analysis_tools.**calc_autocorrelation_time**(*time*, *signal*, *threshold=0.2*)
Estimate the autocorrelation time at a given threshold.

Parameters

- **time** – Time vector of your signal
- **signal** – Signal vector
- **threshold** (*float*) – Threshold under which the signal is correlated

Returns

- **autocorrelation_time - Minimum time step to capture the signal at a** correltion under thethreshold

arnica.utils.temporal_analysis_tools.**calculate_std**(*time*, *signal*, *frequency*)
Give the standard deviation of a signal at a given frequency.

Parameters

- **time** – Time vector of your signal
- **signal** – Signal vector
- **frequency** – Frequency at which values of the recording are taken

Returns

- **std** - Standard deviation of the values taken from the recording

arnica.utils.temporal_analysis_tools.**convergence_cartography**(*time*, *signal*, ***kwargs*)
Create a cartography of the convergence of the confidence interval in a simulation.

Parameters

- **time** – Time vector of your signal
- **signal** – Signal vector

==**kwargs==

param max_time Maximal simulation duration

param interlen Maximal interval length

Returns

- **fig** - Figure of the cartography

`arnica.utils.temporal_analysis_tools.`**`duration_for_uncertainty`**(*time*, *signal*, *target=10*, *confidence=0.95*, *distribution='normal'*)

Give suggestion of simulation duration of a plan40 calculation.

Parameters

- **`time`** – Time vector of your signal
- **`signal`** – Signal vector
- **`target`** – Desired amplitude of the confidence interval
- **`confidence`** – Level of confidence of the interval
- **`distribution`** – Type of distribution of the signal to make the interval

Returns

- **duration** -Duration of the signal

`arnica.utils.temporal_analysis_tools.`**`ks_test_distrib`**(*data*, *distribution='normal'*)

Calculate the correlation score of the signal with the distribution

Parameters

- **`data`** – array of values
- **`distribution`** – kind of distribution the values follow to test

Returns

- **score** -Minimum score over the height of the ks test
- **position** -Index of the height at which the min. of the test is found
- **height** -Corresponding heiht where the min. is found
- **scale** -Scale parameter of the lognormal fitting

`arnica.utils.temporal_analysis_tools.`**`plot_distributions`**(*path='./data.txt'*)

`arnica.utils.temporal_analysis_tools.`**`power_representative_frequency`**(*time*, *signal*, *threshold=0.8*)

Calculate the frequency that captures a level of spectral power.

Parameters

- **`time`** – Time vector of your signal
- **`signal`** – Signal vector
- **`threshold`** – Level of representativity of the spectral power

Returns

- **representative_frequency** - Frequency above which the power threshold is reached

Note: It calculates the cumulative power spectral density and returns the frequency that reaches the threshold of spectral power.

`arnica.utils.temporal_analysis_tools.`**`power_spectral_density`**(*time*, *signal*)

Automate the computation of the Power Spectral Density of a signal.

Parameters

- **`time`** – Time vector of your signal
- **`signal`** – Signal vector

Returns

- **frequency** -Frequency vector of the signal's power spectral density
- **power_spectral_density** -Power spectral density of the signal

`arnica.utils.temporal_analysis_tools.`**`resample_signal`**(*time*, *signal*, *dtime=None*)

Resample the initial signal at a constant time interval.

Parameters

- **`time`** – Time vector of your signal
- **`signal`** – Signal vector

Dtime New time step

Returns

- **rescaled_time** - Uniformally rescaled time vector
- **rescaled_signal** - Rescaled signal

Note: If a dtime is given, the interpolation is made to have a signal with a time interval of dtime. Else, the dt is the smallest time interval between two values of the signal.

`arnica.utils.temporal_analysis_tools.`**`show_autocorrelation_time`**(*time*, *signal*, *threshold=0.2*)

Plot the autocorrelation function of the signal.

Parameters

- **`time`** – Time vector of your signal
- **`signal`** – Signal vector
- **`threshold`** – Autocorrelation threshold

Returns

- **fig** - Figure of the result

arnica.utils.temporal_analysis_tools.**show_power_representative_frequency**(*time*, *signal*, *threshold=0.8*)

Plot the power spectral density of the signal.

Parameters

- **time** – Time vector of your signal
- **signal** – Signal vector
- **threshold** – Power representative frequency threshold

Returns

- **fig** - Figure of the result

arnica.utils.temporal_analysis_tools.**show_temperature_distribution**(*temperature_recording*, *height*, *distribution='normal'*)

Plot the temperature distribution and the fitting curve

Parameters

- **temperature_recording** – Temperature as a function of height and time
- **height** – Height in the plan40
- **distribution** – Type of distribution for the fitting method

Returns

- **fig** - Figure of the result

arnica.utils.temporal_analysis_tools.**sort_spectral_power**(*time*, *signal*)

Determine the harmonic power contribution of the signal.

Parameters

- **time** – Time vector of your signal
- **signal** – Signal vector

returns:

- **harmonic_power** - Harmonic power of the signal
- **total_power** - Total spectral power of the signal

Note: It calculates the Power Spectral Density (PSD) of the complete signal and of a downsampled version of the signal. The difference of the two PSD contains only harmonic components.

arnica.utils.temporal_analysis_tools.**to_percent**(*y*, *position*)

Rescale the y-axis to per

arnica.utils.temporal_analysis_tools.**uncertainty_from_duration**(*dtime*, *sigma*, *duration*, *confidence=0.95*, *distribution='normal'*)

Give confidence interval length of a plan40 calculation.

Parameters

- **dtime** – Time step of your solutions
- **sigma** – Standard deviation of your signal
- **duration** – Desired duration of the signal
- **confidence** – Level of confidence of the interval
- **distribution** – Type of distribution of the signal to make the interval

Returns

- **length** - Length of the confidence interval in K

4.17 arnica.utils.timer_decorator module

small timer function

arnica.utils.timer_decorator.**timing**(*function*)

lazy method to time my function

4.18 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

4.19 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

arnica.phys, 7
arnica.phys.solid_material, 7
arnica.phys.thermodyn_properties, 8
arnica.phys.wall_thermal_equilibrium, 8
arnica.phys.yk_from_phi, 9
arnica.utils, 11
arnica.utils.axishell, 12
arnica.utils.cloud2cloud, 14
arnica.utils.data_avbp_as_ptcloud, 14
arnica.utils.datadict2file, 14
arnica.utils.directed_projection, 16
arnica.utils.lay_and_temp_manager, 19
arnica.utils.mesh_tool, 19
arnica.utils.nparray2xmf, 20
arnica.utils.plot_ave_with_interval, 20
arnica.utils.plot_density_mesh, 21
arnica.utils.show_mat, 21
arnica.utils.showy, 21
arnica.utils.string_manager, 25
arnica.utils.temporal_analysis_tools, 25
arnica.utils.timer_decorator, 29
arnica.utils.unstructured_adjacency, 29
arnica.utils.vector_actions, 30

Index

A

add_curviwidth() (*arnica.utils.axishell.AxiShell method*), 12
add_field() (*arnica.utils.nparray2xmf.NpArray2Xmf method*), 20
angle_btw_vects() (*in module arnica.utils.vector_actions*), 30
arnica.phys (*module*), 7
arnica.phys.solid_material (*module*), 7
arnica.phys.thermodyn_properties (*module*), 8
arnica.phys.wall_thermal_equilibrium (*module*), 8
arnica.phys.yk_from_phi (*module*), 9
arnica.utils (*module*), 11
arnica.utils.axishell (*module*), 12
arnica.utils.cloud2cloud (*module*), 14
arnica.utils.data_avbp_as_ptcloud (*module*), 14
arnica.utils.datadict2file (*module*), 14
arnica.utils.directed_projection (*module*), 16
arnica.utils.lay_and_temp_manager (*module*), 19
arnica.utils.mesh_tool (*module*), 19
arnica.utils.nparray2xmf (*module*), 20
arnica.utils.plot_ave_with_interval (*module*), 20
arnica.utils.plot_density_mesh (*module*), 21
arnica.utils.show_mat (*module*), 21
arnica.utils.showy (*module*), 21
arnica.utils.string_manager (*module*), 25
arnica.utils.temporal_analysis_tools (*module*), 25
arnica.utils.timer_decorator (*module*), 29
arnica.utils.unstructured_adjacency (*module*), 29
arnica.utils.vector_actions (*module*), 30
AVBPAsPointCloud (*class in arnica.utils.data_avbp_as_ptcloud*), 14
average_on_shell_over_dirs() (*arnica.utils.axishell.AxiShell method*), 13
AxiShell (*class in arnica.utils.axishell*), 12

B

bake_millefeuille() (*arnica.utils.axishell.AxiShell method*), 13
build_shell() (*arnica.utils.axishell.AxiShell method*), 13

C

calc_autocorrelation_time() (*in module arnica.utils.temporal_analysis_tools*), 25
calculate_std() (*in module arnica.utils.temporal_analysis_tools*), 25
cart_to_cyl() (*in module arnica.utils.vector_actions*), 30
clip_by_bounds() (*in module arnica.utils.vector_actions*), 30
cloud2cloud() (*in module arnica.utils.cloud2cloud*), 14
compute_dists() (*in module arnica.utils.directed_projection*), 16
compute_equilibrium() (*in module arnica.phys.wall_thermal_equilibrium*), 8
convergence_cartography() (*in module arnica.utils.temporal_analysis_tools*), 25
create_grid() (*arnica.utils.nparray2xmf.NpArray2Xmf method*), 20
create_time_collection_xmf() (*in module arnica.utils.nparray2xmf*), 20
cyl_to_cart() (*in module arnica.utils.vector_actions*), 31

D

decompact_template() (*in module arnica.utils.lay_and_temp_manager*), 19

dilate_center() (*in module arnica.utils.mesh_tool*), 19
dilate_vect_around_x() (*in module arnica.utils.vector_actions*), 31
display() (*in module arnica.utils.showy*), 25
dump() (*arnica.utils.axishell.AxiShell method*), 13
dump() (*arnica.utils.nparray2xmf.NpArray2Xmf method*), 20
dump_dico_0d() (*in module arnica.utils.datadict2file*), 14
dump_dico_1d_nparrays() (*in module arnica.utils.datadict2file*), 15
dump_dico_2d_nparrays() (*in module arnica.utils.datadict2file*), 15
dump_dico_2d_time_nparrays() (*in module arnica.utils.datadict2file*), 15
dump_dict2xmdf() (*in module arnica.utils.datadict2file*), 15
duration_for_uncertainty() (*in module arnica.utils.temporal_analysis_tools*), 26

F

fetch_avbp_layouts() (*in module arnica.utils.lay_and_temp_manager*), 19
fetch_avbp_templates() (*in module arnica.utils.lay_and_temp_manager*), 19
filter_stupid_characters() (*in module arnica.utils.show_mat*), 21
fluid_cp() (*in module arnica.phys.thermodyn_properties*), 8

G

gen_cart_grid_2d() (*in module arnica.utils.mesh_tool*), 19
gen_cyl_grid_2d() (*in module arnica.utils.mesh_tool*), 19
get_cell2node() (*arnica.utils.unstructured_adjacency.UnstructuredAdjacency method*), 29
get_filter() (*arnica.utils.unstructured_adjacency.UnstructuredAdjacency method*), 29
get_mesh() (*in module arnica.utils.mesh_tool*), 19
get_node2cell() (*arnica.utils.unstructured_adjacency.UnstructuredAdjacency method*), 29
get_skinpts() (*arnica.utils.data_avbp_as_ptcloud.AVBPAsPointCloud method*), 14

H

h_kader() (*in module arnica.phys.thermodyn_properties*), 8
heat_map_mesh() (*in module arnica.utils.plot_density_mesh*), 21

I

init_mockup() (*arnica.utils.axishell.AxiShell method*), 13
intersect_plan_line() (*in module arnica.utils.directed_projection*), 16

K

ks_test_distrib() (*in module arnica.utils.temporal_analysis_tools*), 26

L

lambda_cp_visco_fluid() (*in module arnica.phys.thermodyn_properties*), 8
lambda_th() (*arnica.phys.solid_material.SolidMaterial method*), 7
load() (*arnica.utils.axishell.AxiShell method*), 13
load_avgsol() (*arnica.utils.data_avbp_as_ptcloud.AVBPAsPointCloud method*), 14
load_mesh_bnd() (*arnica.utils.data_avbp_as_ptcloud.AVBPAsPointCloud method*), 14
load_mesh_bulk() (*arnica.utils.data_avbp_as_ptcloud.AVBPAsPointCloud method*), 14

M

make_radial_vect() (*in module arnica.utils.vector_actions*), 31
mask_cloud() (*in module arnica.utils.vector_actions*), 31

N

ncell (*arnica.utils.unstructured_adjacency.UnstructuredAdjacency attribute*), 30
ncell_per_nvert (*arnica.utils.unstructured_adjacency.UnstructuredAdjacency attribute*), 30
NpArray2Xmf (*class in arnica.utils.nparray2xmf*), 20

P

phi_from_far() (*in module arnica.phys.yk_from_phi*), 9
plot_ave_with_interval() (*in module arnica.utils.plot_ave_with_interval*), 20
plot_dict_data_as_file() (*in module arnica.utils.datadict2file*), 16
plot_distributions() (*in module arnica.utils.temporal_analysis_tools*), 26
power_representative_frequency() (*in module arnica.utils.temporal_analysis_tools*), 26
power_spectral_density() (*in module arnica.utils.temporal_analysis_tools*), 27

project_points() (*in module arnica.utils.directed_projection*), 17
projection_kdtree() (*in module arnica.utils.directed_projection*), 18

R

renormalize() (*in module arnica.utils.vector_actions*), 31
resample_signal() (*in module arnica.utils.temporal_analysis_tools*), 27
rotate_vect_around_axis() (*in module arnica.utils.vector_actions*), 31
rotate_vect_around_x() (*in module arnica.utils.vector_actions*), 31
rtheta2yz() (*in module arnica.utils.vector_actions*), 31

S

scatter_plot_mesh() (*in module arnica.utils.plot_density_mesh*), 21
set_mask_on_shell() (*arnica.utils.axishell.AxiShell method*), 13
show_autocorrelation_time() (*in module arnica.utils.temporal_analysis_tools*), 27
show_mat() (*in module arnica.utils.show_mat*), 21
show_power_representative_frequency() (*in module arnica.utils.temporal_analysis_tools*), 27
show_temperature_distribution() (*in module arnica.utils.temporal_analysis_tools*), 28
showy() (*in module arnica.utils.showy*), 24
SolidMaterial (*class in arnica.phys.solid_material*), 7
sort_spectral_power() (*in module arnica.utils.temporal_analysis_tools*), 28

T

thermal_constants() (*in module arnica.phys.thermodyn_properties*), 8
thermal_resistance() (*arnica.phys.solid_material.SolidMaterial method*), 7
timing() (*in module arnica.utils.timer_decorator*), 29
to_percent() (*in module arnica.utils.temporal_analysis_tools*), 28

U

uncertainty_from_duration() (*in module arnica.utils.temporal_analysis_tools*), 28
UnstructuredAdjacency (*class in arnica.utils.unstructured_adjacency*), 29

V

vect_to_quat() (*in module arnica.utils.vector_actions*), 32
viscosity_sutherland() (*in module arnica.phys.thermodyn_properties*), 8

W

width_mockup() (*in module arnica.utils.axishell*), 14

X

xmf_dump() (*arnica.utils.nparray2xmf.NpArray2Xmf method*), 20

Y

yk_from_phi() (*in module arnica.phys.yk_from_phi*), 9
yz_to_theta() (*in module arnica.utils.vector_actions*), 32