# Pleque Documentation

***Release 0.0.3***

**Lukas Kripner**

**Feb 07, 2020**

# Contents:

Code home: https://github.com/kripnerl/pleque/

CHAPTER 1

# Coordinates

## 1.1 Accepted coordinates types

**1D - coordinates**

| Coordinate | Code | Note |
|---|---|---|
| $\psi_{\mathrm{N}}$ | `psi_n` | Default 1D coordinate |
| $\psi$ | `psi` | |
| $\rho$ | `rho` | $\rho = \sqrt{\psi_n}$ |

**2D - coordinates**

| Coordinate | Code | Note |
|---|---|---|
| $(R, Z)$ | `R, Z` | Default 2D coordinate |
| $(r, \theta)$ | `r, theta` | Polar coordinates with respect to magnetic axis |

**3D - coordinates**

| Coordinate | Code | Note |
|---|---|---|
| $(R, Z, \phi)$ | `R, Z, phi` | Default 3D coordinate |
| $(X, Y, Z)$ | `X, Y, Z` | |

CHAPTER 2

# Examples

Nothing here yet.

Please see examples in `notebooks` and `examples` folder. The examples will be also improved soon.

# CHAPTER 3

# Flux expansion module

PLEQUE provides set of functions for mapping of upstream heat fluxes.

## 3.1 API Reference

pleque.utils.flux_expansions.**effective_poloidal_heat_flux_exp_coef**(*equilibrium: pleque.core.equilibrium.Equilibriu*
*coords: pleque.core.coordinates.Coordinat*

**Effective poloidal heat flux expansion coefficient**

**Definition:**

$$f_{\mathrm{pol,heat,eff}} = \frac{B_\theta^{\mathrm{u}}}{B_\theta^{\mathrm{t}}} \frac{1}{\sin\beta} = \frac{f_{\mathrm{pol}}}{\sin\beta}$$

Where $\beta$ is inclination angle of the poloidal magnetic field and the target plane.

**Typical usage:**

*Effective poloidal heat flux expansion coefficient* is typically used scale upstream poloidal heat flux to the target plane.

$$q_\perp^{\mathrm{t}} = \frac{q_\theta^{\mathrm{u}}}{f_{\mathrm{pol,heat,eff}}}$$

**Parameters**

- **equilibrium** – Instance of `Equilibrium`.
- **coords** – `Coordinates` where the coefficient is evaluated.

**Returns**

pleque.utils.flux_expansions.**effective_poloidal_mag_flux_exp_coef**(*equilibrium: pleque.core.equilibrium.Equilibrium*, *coords: pleque.core.coordinates.Coordinates*)

**Effective poloidal magnetic flux expansion coefficient**

**Definition:**

$$f_{\mathrm{pol,eff}} = \frac{B_\theta^{\mathrm{u}} R^{\mathrm{u}}}{B_\theta^{\mathrm{t}} R^{\mathrm{t}}} \frac{1}{\sin\beta} = \frac{f_{\mathrm{pol}}}{\sin\beta}$$

Where $\beta$ is inclination angle of the poloidal magnetic field and the target plane.

**Typical usage:**

*Effective magnetic flux expansion coefficient* is typically used for $\lambda$ scaling of the target $\lambda$ with respect to the upstream value.

$$\lambda^{\mathrm{t}} = \lambda_q^{\mathrm{u}} f_{\mathrm{pol,eff}}$$

This coefficient can be also used to calculate peak target heat flux from the total power through LCFS if the perpendicular diffusion is neglected. Then for the peak value stays

$$q_{\perp,\mathrm{peak}} = \frac{P_{\mathrm{div}}}{2\pi R^{\mathrm{t}} \lambda_q^{\mathrm{u}}} \frac{1}{f_{\mathrm{pol,eff}}}$$

Where $P_{\mathrm{div}}$ is total power to outer strike point and $lambda_q^mathrm{u}$ is e-folding length on the outer midplane.

**Parameters**

- **equilibrium** – Instance of `Equilibrium`.
- **coords** – `Coordinates` where the coefficient is evaluated.

**Returns**

pleque.utils.flux_expansions.**impact_angle_cos_pol_projection**(*coords: pleque.core.coordinates.Coordinates*)

Impact angle calculation - dot product of PFC norm and local magnetic field direction poloidal projection only. Internally uses *incidence_angle_sin* function where *vecs* are replaced by the vector of the poloidal magnetic field (Bphi = 0).

**Returns** `array` of impact angles

pleque.utils.flux_expansions.**impact_angle_sin**(*coords: pleque.core.coordinates.Coordinates*)

Impact angle calculation - dot product of PFC norm and local magnetic field direction. Internally uses *incidence_angle_sin* function where *vecs* are replaced by the vector of the magnetic field.

**Returns** `array` of impact angles cosines

pleque.utils.flux_expansions.**incidence_angle_sin**(*coords: pleque.core.coordinates.Coordinates*, *vecs*)

**Parameters**

- **coords** – `Coordinate` object (of length `N_vecs`)of a line in the space on which the incidence angle is evaluated.
- **vecs** – `array (3, N_vecs)` vectors in (R, Z, phi) space.

**Returns** array of sines of angles of incidence. I.e. cosine of the angle between the normal to the line (in the poloidal plane) and the corresponding vector.

pleque.utils.flux_expansions.**parallel_heat_flux_exp_coef**(*equilibrium: pleque.core.equilibrium.Equilibrium*, *coords: pleque.core.coordinates.Coordinates*)

**Parallel heat flux expansion coefficient**

**Definition:**

$$f_{\parallel} = \frac{B^{\mathrm{u}}}{B^{\mathrm{t}}}$$

**Typical usage:**

*Parallel heat flux expansion coefficient* is typically used to scale total upstream heat flux parallel to the magnetic field along the magnetic field lines.

$$q_{\parallel}^{\mathrm{t}} = \frac{q_{\parallel}^{\mathrm{u}}}{f_{\parallel}}$$

**Parameters**

- **equilibrium** – Instance of `Equilibrium`.
- **coords** – `Coordinates` where the coefficient is evaluated.

**Returns**

pleque.utils.flux_expansions.**poloidal_heat_flux_exp_coef**(*equilibrium: pleque.core.equilibrium.Equilibrium*, *coords: pleque.core.coordinates.Coordinates*)

**Poloidal heat flux expansion coefficient**

**Definition:**

$$f_{\mathrm{pol,heat}} = \frac{B_{\theta}^{\mathrm{u}}}{B_{\theta}^{\mathrm{t}}}$$

**Typical usage:** *Poloidal heat flux expansion coefficient* is typically used to scale poloidal heat flux (heat flux projected along poloidal magnetic field) along the magnetic field line.

$$q_{\theta}^{\mathrm{t}} = \frac{q_{\theta}^{\mathrm{u}}}{f_{\mathrm{pol,heat}}}$$

**Parameters**

- **equilibrium** – Instance of `Equilibrium`.
- **coords** – `Coordinates` where the coefficient is evaluated.

**Returns**

pleque.utils.flux_expansions.**poloidal_mag_flux_exp_coef**(*equilibrium: pleque.core.equilibrium.Equilibrium*, *coords: pleque.core.coordinates.Coordinates*)

**Poloidal magnetic flux expansion coefficient.**

**Definition:**

$$f_{\mathrm{pol}} = \frac{\Delta r^{\mathrm{t}}}{\Delta r^{\mathrm{u}}} = \frac{B_{\theta}^{\mathrm{u}} R^{\mathrm{u}}}{B_{\theta}^{\mathrm{t}} R^{\mathrm{t}}}$$

**Typical usage:**

*Poloidal magnetic flux expansion coefficient* is typically used for $\lambda$ scaling in plane perpendicular to the poloidal component of the magnetic field.

**Parameters**

- **equilibrium** – Instance of `Equilibrium`.
- **coords** – `Coordinates` where the coefficient is evaluated.

**Returns**

pleque.utils.flux_expansions.**total_heat_flux_exp_coef**(*equilibrium: pleque.core.equilibrium.Equilibrium*, *coords: pleque.core.coordinates.Coordinates*)

**Total heat flux expansion coefficient**

**Definition:**

$$f_{\mathrm{tot}} = \frac{B^{\mathrm{u}}}{B^{\mathrm{t}}} \frac{1}{\sin\alpha} = \frac{f_{\parallel}}{\sin\alpha}$$

Where $\alpha$ is an inclination angle of the total magnetic field and the target plane.

---

**Important:** $\alpha$ is an inclination angle of the total magnetic field to the target plate. Whereas $\beta$ is an inclination of poloidal components of the magnetic field to the target plate.

---

**Typical usage:**

*Total heat flux expansion coefficient* is typically used to project total upstream heat flux parallel to the magnetic field to the target plane.

$$q_{\perp}^{\mathrm{t}} = \frac{q_{\parallel}^{\mathrm{u}}}{f_{\mathrm{tot}}}$$

**Parameters**

- **equilibrium** – Instance of `Equilibrium`.
- **coords** – `Coordinates` where the coefficient is evaluated.

**Returns**

## 3.2 References

Theiler, C., et al.: *Results from recent detachment experiments in alternativee divertor cofigurations on TCV*, Nucl. Fusion **57** (2017) 072008 16pp

Vondracek, P.: *Plasma Heat Flux to Solid Structures in Tokamaks*, PhD thesis, Prague 2019

CHAPTER 4

# Naming convention used in PLEQUE

## 4.1 Coordinates

Here presented naming convention is used to read/create input/output dict/xarray files.

- **2D**
  - `R` (default): Radial cylindrical coordinates with zero on machine axis
  - `Z` (default): Vertical coordinate with zero on machine geometrical axis
- **1D**
  - `psi_n` (default): Normalized poloidal magnetic flux with zero on magnetic axis and one on the last closed flux surface

    $\psi_{\mathrm{N}} = \frac{\psi - \psi_{\mathrm{ax}}}{\psi_{\mathrm{LCFS}} - \psi_{\mathrm{ax}}}$
  - Fallowing input options are not implemented yet.
  - `rho`: $\rho = \sqrt{\psi_{\mathrm{N}}}$
  - `psi_1dprof` - poloidal magnetic flux; this coordinate axis is used only if `psi_n` is not found on the input. Output files uses implicitly `psi_n` axis.

## 4.2 2D profiles

- **Required on the input**
  - `psi` (Wb): poloidal magnetic flux
- **Calculated**
  - `B_R` (T): $R$ component of the magnetic field.
  - `B_Z` (T): $Z$ component of the magnetic field.

  - `B_pol` (T): Poloidal component of the magnetic field. $B_\theta = \text{sign}\,(I_p)\sqrt{B_R^2 + B_Z^2}$ **Todo** resolve the sign of B_pol and implement it!!!
  - `B_tor` (T): Toroidal component of the magnetic field.
  - `B_abs` (T): Absolute value of the magnetic field.
  - `j_R` (A/m2): $R$ component of the current density. **todo: Check the current unit**
  - `j_Z` (A/m2): $Z$ component of the current density.
  - `j_pol` (A/m2): Poloidal component of the current density.
  - `j_tor` (A/m2): Toroidal component of the current density.
  - `j_abs` (A/m2): Asolute value of the current density.

## 4.3 1D profiles

- **Required on the input**
  - `pressure` (Pa)
  - `pprime` (Pa/Wb)
  - `F`: $F = RB_\phi$
- **Calculated**
  - `pprime`: $p\partial_\psi$
  - `Fprime`: $F' = \partial_\psi F$
  - `FFprime`: $FF' = F\partial_\psi F$
  - `fprime`: $f' = \partial_\psi f$
  - `f`: $f = (1/\mu_0)RB_\phi$
  - `ffprime`: $ff' = f\partial_\psi f$
  - `rho`, `psi_n`
- **Deriver**
  - $q$: safety factor profile
  - *qprime*$q' = \partial_\psi q$
  - **Not yet implemented:**
    - *magnetic_shear*
    - ...

## 4.4 Attributes

- To be written.

## 4.5 FluxSurface quantities

CHAPTER 5

# Notebooks

CHAPTER 6

# API Reference

## 6.1 API Reference

### 6.1.1 Equilibrium

**class** pleque.core.equilibrium.**Equilibrium**(*basedata: xarray.core.dataset.Dataset*, *first_wall=None*, *mg_axis=None*, *psi_lcfs=None*, *x_points=None*, *strike_points=None*, *init_method='hints'*, *spline_order=3*, *spline_smooth=0*, *cocos=3*, *verbose=True*)

Bases: object

Equilibrium class ...

**B_R**(**coordinates*, *R=None*, *Z=None*, *coord_type=('R', 'Z')*, *grid=True*, ***coords*)

Poloidal value of magnetic field in Tesla.

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**B_Z**(**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)

Poloidal value of magnetic field in Tesla.

**Parameters**

- **grid** –
- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **coords** –

**Returns**

**B_abs** (**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)
Absolute value of magnetic field in Tesla.

**Parameters**

- **grid** –
- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **coords** –

**Returns** Absolute value of magnetic field in Tesla.

**B_pol** (**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)
Absolute value of magnetic field in Tesla.

**Parameters**

- **grid** –
- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **coords** –

**Returns**

**B_tor** (**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *grid=True*, ***coords*)
Toroidal value of magnetic field in Tesla.

**Parameters**

- **grid** –
- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **coords** –

**Returns**

**Bvec**(**coordinates*, *swap_order=False*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)
Magnetic field vector

**Parameters**

- **grid** –
- **coordinates** –
- **swap_order** – bool,
- **R** –
- **Z** –
- **coord_type** –
- **coords** –

**Returns** Magnetic field vector array (3, N) if swap_order is False.

**Bvec_norm**(**coordinates*, *swap_order=False*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)
Magnetic field vector, normalised

**Parameters**

- **grid** –
- **coordinates** –
- **swap_order** –
- **R** –
- **Z** –
- **coord_type** –
- **coords** –

**Returns** Normalised magnetic field vector array (3, N) if swap_order is False.

**F**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=True*, ***coords*)

**FFprime**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=True*, ***coords*)

**Fprime**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=True*, ***coords*)

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **psi_n** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**I_plasma**
Toroidal plasma current. Calculated as toroidal current through the LCFS.

**Returns** (float) Value of toroidal plasma current.

**__init__**(*basedata: xarray.core.dataset.Dataset*, *first_wall=None*, *mg_axis=None*, *psi_lcfs=None*, *x_points=None*, *strike_points=None*, *init_method='hints'*, *spline_order=3*, *spline_smooth=0*, *cocos=3*, *verbose=True*)

Equilibrium class instance should be obtained generally by functions in pleque.io package.

Optional arguments may help the initialization.

**Parameters**

- **basedata** – xarray.Dataset with psi(R, Z) on a rectangular R, Z grid, f(psi_norm), p(psi_norm) f = B_tor * R
- **first_wall** – array-like (Nwall, 2) required for initialization in case of limiter configuration.
- **mg_axis** – suspected position of the o-point
- **psi_lcfs** –
- **x_points** –
- **strike_points** –
- **init_method** – str On of ("full", "hints", "fast_forward"). If "full" no hints are taken and module tries to recognize all critical points itself. If "hints" module use given optional arguments as a help with initialization. If "fast-forward" module use given optional arguments as final and doesn't try to correct. *Note:* Only "hints" method is currently tested.
- **spline_order** –
- **spline_smooth** –
- **cocos** – At the moment module assume cocos to be 3 (no other option). The implemetnation is not fully working. Be aware of signs in the module!
- **verbose** –

**abs_q**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *grid=False*, ***coords*)

Absolute value of q.

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**cocos**

Number of internal COCOS representation.

**Returns** int

**connection_length**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *direction=1*, ***coords*)

Calculate connection length from given coordinates to first wall

**Todo: The field line is traced to min/max value of z of first wall, distance is calculated to the last** point before first wall.

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **direction** – if positive trace field line in/cons the direction of magnetic field.
- **stopper** – (None, 'poloidal', 'z-stopper) force to use stopper. If None stopper is automatically chosen based on psi_n coordinate.
- **coords** –

**Returns**

**contact_point**
Returns contact point as instance of coordinates for circular plasmas. Returns None otherwise. :return:

**coordinates**(**coordinates*, *coord_type=None*, *grid=False*, ***coords*)
Return instance of Coordinates. If instances of coordinates is already on the input, just pass it through.

**Parameters**

- **coordinates** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**diff_psi**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=False*, ***coords*)
Return the value of $\nabla\psi$. It is positive/negative if the $\psi$ is increasing/decreasing.

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **psi_n** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**diff_q**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=False*, ***coords*)

**Parameters**

- **self** –
- **coordinates** –

- **R** –
- **Z** –
- **psi_n** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns** Derivative of q with respect to psi.

**effective_poloidal_heat_flux_exp_coef**(**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)

**Effective poloidal heat flux expansion coefficient**

**Definition:**

$$f_{\mathrm{pol,heat,eff}} = \frac{B_\theta^{\mathrm{u}}}{B_\theta^{\mathrm{t}}} \frac{1}{\sin\beta} = \frac{f_{\mathrm{pol}}}{\sin\beta}$$

Where $\beta$ is inclination angle of the poloidal magnetic field and the target plane.

**Typical usage:**

*Effective poloidal heat flux expansion coefficient* is typically used scale upstream poloidal heat flux to the target plane.

$$q_\perp^{\mathrm{t}} = \frac{q_\theta^{\mathrm{u}}}{f_{\mathrm{pol,heat,eff}}}$$

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**effective_poloidal_mag_flux_exp_coef**(**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)

**Effective poloidal magnetic flux expansion coefficient**

**Definition:**

$$f_{\mathrm{pol,eff}} = \frac{B_\theta^{\mathrm{u}} R^{\mathrm{u}}}{B_\theta^{\mathrm{t}} R^{\mathrm{t}}} \frac{1}{\sin\beta} = \frac{f_{\mathrm{pol}}}{\sin\beta}$$

Where $\beta$ is inclination angle of the poloidal magnetic field and the target plane.

**Typical usage:**

*Effective magnetic flux expansion coefficient* is typically used for $\lambda$ scaling of the target $\lambda$ with respect to the upstream value.

$$\lambda^{\mathrm{t}} = \lambda^{\mathrm{u}} f_{\mathrm{pol,eff}}$$

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**f**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=True*, ***coords*)

**ffprime**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=True*, ***coords*)

**first_wall**
If the first wall polygon is composed of 3 and more points Surface instance is returned. If the wall contour is composed of less than 3 points, coordinate instance is returned, because Surface can't be constructed :return:

**flux_surface**(**coordinates*, *resolution=(0.001*, *0.001)*, *dim='step'*, *closed=True*, *inlcfs=True*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, ***coords*)

**fluxfuncs**

**get_precise_lcfs**()
Calculate plasma LCFS by field line tracing technique and save LCFS as instance property.

**Returns**

**grid**(*resolution=None*, *dim='step'*)
Function which returns 2d grid with requested step/dimensions generated over the reconstruction space.

**Parameters**

- **resolution** – Iterable of size 2 or a number. If a number is passed, R and Z dimensions will have the same size or step (depending on dim parameter). Different R and Z resolutions or dimension sizes can be required by passing an iterable of size 2. If None, default grid of size (1000, 2000) is returned.
- **dim** – iterable of size 2 or string ('step', 'size'). Default is "step", determines the meaning of the resolution. If "step" used, values in resolution are interpreted as step length in psi poloidal map. If "size" is used, values in resolution are interpreted as requested number of points in a dimension. If string is passed, same value is used for R and Z dimension. Different interpretation of resolution for R, Z dimensions can be achieved by passing an iterable of shape 2.

**Returns** Instance of *Coordinates* class with grid data

**in_first_wall**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *grid=True*, ***coords*)

**in_lcfs**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *grid=True*, ***coords*)

**is_limter_plasma**
Return true if the plasma is limited by point or some limiter point.

**Returns** bool

**is_xpoint_plasma**
Return true for x-point plasma.

**Returns** bool

**j_R**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *grid=True*, ***coords*)

**j_Z**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *grid=True*, ***coords*)

**j_pol**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *grid=False*, ***coords*)

Poloidal component of the current density. Calculated as

$$\frac{f'\nabla\psi}{R\mu_0}$$

[Wesson: Tokamaks, p. 105]

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**j_tor**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *grid=True*, ***coords*)

todo: to be tested

Toroidal component of the current denisity. Calculated as

$$Rp' + \frac{1}{\mu_0 R} f f'$$

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**lcfs**

**limiter_point**

The point which "limits" the LCFS of plasma. I.e. contact point in case of limiter plasma and x-point in case of x-point plasma.

**Returns** Coordinates

**magnetic_axis**

**outter_parallel_fl_expansion_coef**(**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)

WIP:Calculate parallel expansion coefitient of the given coordinates with respect to positon on the outer midplane.

**outter_poloidal_fl_expansion_coef**(**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)

WIP:Calculate parallel expansion coefitient of the given coordinates with respect to positon on the outer midplane.

**parallel_heat_flux_exp_coef**(**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)

**Parallel heat flux expansion coefficient**

**Definition:**

$$f_{\parallel} = \frac{B^{\mathrm{u}}}{B^{\mathrm{t}}}$$

**Typical usage:**

*Parallel heat flux expansion coefficient* is typically used to scale total upstream heat flux parallel to the magnetic field along the magnetic field lines.

$$q_{\parallel}^{\mathrm{t}} = \frac{q_{\parallel}^{\mathrm{u}}}{f_{\parallel}}$$

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**plot_geometry**(*axs=None*, ***kwargs*)

Plots the the directions of angles, current and magnetic field.

**Parameters**

- **axs** – None or tuple of axes. If None new figure with to axes is created.
- **kwargs** – parameters passed to the *plot* routine.

**Returns** tuple of axis (ax1, ax2)

**plot_overview**(*ax=None*, ***kwargs*)

Simple routine for plot of plasma overview :return:

**poloidal_heat_flux_exp_coef**(**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)

**Poloidal heat flux expansion coefficient**

**Definition:**

$$f_{\mathrm{pol,heat}} = \frac{B_{\theta}^{\mathrm{u}}}{B_{\theta}^{\mathrm{t}}}$$

**Typical usage:** *Poloidal heat flux expansion coefficient* is typically used to scale poloidal heat flux (heat flux projected along poloidal magnetic field) along the magnetic field line.

$$q_\theta^{\mathrm{t}} = \frac{q_\theta^{\mathrm{u}}}{f_{\mathrm{pol,heat}}}$$

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**poloidal_mag_flux_exp_coef**(**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)

**Poloidal magnetic flux expansion coefficient.**

**Definition:**

$$f_{\mathrm{pol}} = \frac{\Delta r^{\mathrm{t}}}{\Delta r^{\mathrm{u}}} = \frac{B_\theta^{\mathrm{u}} R^{\mathrm{u}}}{B_\theta^{\mathrm{t}} R^{\mathrm{t}}}$$

**Typical usage:**

*Poloidal magnetic flux expansion coefficient* is typically used for $\lambda$ scaling in plane perpendicular to the poloidal component of the magnetic field.

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**pprime**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=True*, ***coords*)

**pressure**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=True*, ***coords*)

**psi**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=True*, ***coords*)

Psi value

**Parameters**

- **psi_n** –
- **coordinates** –
- **R** –
- **Z** –

- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**psi_n**(**coordinates*, *R=None*, *Z=None*, *psi=None*, *coord_type=None*, *grid=True*, ***coords*)

**q**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *grid=False*, ***coords*)

**r_mid**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=True*, ***coords*)

**rho**(**coordinates*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, *grid=True*, ***coords*)

**separatrix**
If the equilibrium is limited, returns lcfs. If it is diverted it returns separatrix flux surface

**Returns**

**strike_points**
Returns contact point if the equilibrium is limited. If the equilibrium is diverted it returns strike points. :return:

**surfacefuncs**

**to_geqdsk**(*file*, *nx=64*, *ny=128*, *q_positive=True*)
Write a GEQDSK equilibrium file.

**Parameters**

- **file** – str, file name
- **nx** – int
- **ny** – int

**tor_flux**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *grid=False*, ***coords*)

Calculate toroidal magnetic flux $\Phi$ from:

$$q =$$

rac{mathrm{d Phi} }{mathrm{d psi}}

**param coordinates**

**param R**

**param Z**

**param coord_type**

**param grid**

**param coords**

**return**

**total_heat_flux_exp_coef**(**coordinates*, *R=None*, *Z=None*, *coord_type=None*, *grid=True*, ***coords*)
**Total heat flux expansion coefficient**

**Definition:**

$$f_{\mathrm{tot}} = \frac{B^{\mathrm{u}}}{B^{\mathrm{t}}} \frac{1}{\sin\alpha} = \frac{f_{\parallel}}{\sin\alpha}$$

Where $\alpha$ is inclination angle of the total magnetic field and the target plane.

**Typical usage:**

*Total heat flux expansion coefficient* is typically used to project total upstream heat flux parallel to the magnetic field to the target plane.

$$q_{\perp}^{\mathrm{t}} = \frac{q_{\parallel}^{\mathrm{u}}}{f_{\mathrm{tot}}}$$

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **grid** –
- **coords** –

**Returns**

**trace_field_line**(**coordinates*, *R: numpy.array = None*, *Z: numpy.array = None*, *coord_type=None*, *direction=1*, *stopper_method=None*, *in_first_wall=False*, ***coords*)

Return traced field lines starting from the given set of at least 2d coordinates. One poloidal turn is calculated for field lines inside the separatrix. Outter field lines are limited by z planes given be outermost z coordinates of the first wall.

**Parameters**

- **coordinates** –
- **R** –
- **Z** –
- **coord_type** –
- **direction** – if positive trace field line in/cons the direction of magnetic field.
- **stopper_method** – (None, 'poloidal', 'z-stopper) force to use stopper. If None stopper is automatically chosen based on psi_n coordinate.
- **in_first_wall** – if True the only inner part of field line is returned.
- **coords** –

**Returns**

**trace_flux_surface**(**coordinates*, *s_resolution=0.001*, *R=None*, *Z=None*, *psi_n=None*, *coord_type=None*, ***coords*)

Find a closed flux surface inside LCFS with requested values of psi or psi-normalized.

TODO support open and/or flux surfaces outise LCFS, needs different stopper

**Parameters**

- **R** –

- **Z** –
- **psi_n** –
- **coord_type** –
- **coordinates** – specifies flux surface to search for (by spatial point or values of psi or psi normalised). If coordinates is spatial point (dim=2) then the trace starts at the midplane. Coordinates.grid must be False.
- **s_resolution** – max_step in the distance along the flux surface contour

**Returns** FluxSurface

**x_point**
Return x-point closest in psi to mg-axis if presented on grid. None otherwise.

:return Coordinates

### 6.1.2 Fluxsurface

**class** pleque.core.fluxsurface.**FluxSurface**(*equilibrium*, **coordinates*, *coord_type=None*, *grid=False*, ***coords*)

Bases: *pleque.core.fluxsurface.Surface*

**__init__**(*equilibrium*, **coordinates*, *coord_type=None*, *grid=False*, ***coords*)
Calculates geometrical properties of the flux surface. To make the contour closed, the first and last points in the passed coordinates have to be the same. Instance is obtained by calling method *flux_surface* in instance of *Equilibrium*.

**Parameters** **coords** – Instance of coordinate class

**contains**(*coords: pleque.core.coordinates.Coordinates*)

**contour**
Depracated. Fluxsurface contour points. :return: numpy ndarray

**cumsum_surface_average**(*func*, *roll=0*)
Return the surface average (over single magnetic surface) value of *func*. Return the value of integration

$$< func > (\psi)_i = \oint_0^{\theta_i} \frac{\mathrm{d}lR}{|\nabla\psi|} a(R, Z)$$

**Parameters** **func** – func(X, Y), Union[ndarray, int, float]

**Returns** ndarray

**distance**(*coords: pleque.core.coordinates.Coordinates*)

**elongation**
Elongation :return:

**eval_q**

**geom_radius**
Geometrical radius a= (R_min + R_max)./2 :return:

**get_eval_q**(*method*)
Evaluete q usiong formula (5.35) from [Jardin, 2010: Computational methods in Plasma Physics]

**Parameters** **method** – str, [‘sum’, ‘trapz’, ‘simps’]

**Returns**

**max_radius**
maximum radius on the given flux surface :return:

**min_radius**
minimum radius on the given flux surface :return:

**minor_radius**
a= (R_min - R_max)./2 :return:

**straight_fieldline_theta**
Calculate straight field line $\theta^*$ coordinate.

**Returns**

**surface_average**(*func*, *method='sum'*)
Return the surface average (over single magnetic surface) value of *func*. Return the value of integration

$$< func > (\psi) = \oint \frac{\mathrm{d}lR}{|\nabla\psi|} a(R, Z)$$

**Parameters**

- **func** – func(X, Y), Union[ndarray, int, float]
- **method** – str, ['sum', 'trapz', 'simps']

**Returns**

**tor_current**
Return toroidal current through the closed flux surface

**Returns**

**triangul_low**
Lower triangularity :return:

**triangul_up**
Upper triangularity :return:

**triangularity**

**Returns**

**class** pleque.core.fluxsurface.**Surface**(*equilibrium*, **coordinates*, *coord_type=None*, *grid=False*, ***coords*)
Bases: *pleque.core.coordinates.Coordinates*

**__init__**(*equilibrium*, **coordinates*, *coord_type=None*, *grid=False*, ***coords*)
Calculates geometrical properties of a specified surface. To make the contour closed, the first and last points in the passed coordinates have to be the same. Instance is obtained by calling method *surface* in instance of *Equilibrium*.

**Parameters** **coords** – Instance of coordinate class

**area**
Area of the closed fluxsurface.

**Returns**

**centroid**

**closed**
True if the fluxsurface is closed.

**Returns**

**diff_volume**
Diferential volume $V' = dV/d\psi$ Jardin, S.: Computational Methods in Plasma Physics

**Returns**

**length**
Length of the fluxsurface contour

**Returns**

**surface**
Surface of fluxsurface calculated from the contour length using Pappus centroid theorem : https://en.wikipedia.org/wiki/Pappus%27s_centroid_theorem

**Returns** float

**volume**
Volume of the closed fluxsurface calculated from the area using Pappus centroid theorem : https://en.wikipedia.org/wiki/Pappus%27s_centroid_theorem

**Returns** float

### 6.1.3 Coordinates

**class** pleque.core.coordinates.**Coordinates**(*equilibrium*, **coordinates*, *coord_type=None*, *grid=False*, *cocos=None*, ***coords*)

Bases: object

**R**

**X**

**Y**

**Z**

**__init__**(*equilibrium*, **coordinates*, *coord_type=None*, *grid=False*, *cocos=None*, ***coords*)
Basic PLEQUE class to handle various coordinate systems in tokamak equilibrium.

**Parameters**

- **equilibrium** –
- ***coordinates** –
  - Can be skipped.
  - array (N, dim) - N points will be generated.
  - One, two are three comma separated one dimensional arrays.
- **coord_type** –
- **grid** –
- **cocos** – Define coordinate system cocos. Id *None* equilibrium default cocos is used. If *equilibrium is None* cocos = 3 (both systems cnt-clockwise) is used.
- ****coords** – Lorem ipsum.

- **1D**: $\psi_{\mathrm{N}}$,
- **2D**: $(R, Z)$,
- **3D**: $(R, Z, \phi)$.

**1D - coordinates**

| Coordinate | Code | Note |
|---|---|---|
| $\psi_{\mathrm{N}}$ | `psi_n` | Default 1D coordinate |
| $\psi$ | `psi` | |
| $\rho$ | `rho` | $\rho = \sqrt{\psi_n}$ |

**2D - coordintares**

| Coordinate | Code | Note |
|---|---|---|
| $(R, Z)$ | `R, Z` | Default 2D coordinate |
| $(r, \theta)$ | `r, theta` | Polar coordinates with respect to magnetic axis |

**3D - coordinates**

| Coordinate | Code | Note |
|---|---|---|
| $(R, Z, \phi)$ | `R, Z, phi` | Default 3D coordinate |
| $(X, Y, Z)$ | `(X, Y, Z)` | Polar coordinates with respect to magnetic axis |

**as_array**(*dim=None*, *coord_type=None*)

Return array of size (N, dim), where N is number of points and dim number of dimensions specified by coord_type

**Parameters**

- **dim** – reduce the number of dimensions to dim (todo)
- **coord_type** – not effected at the moment (TODO)

**Returns**

**cum_length**

Cumulative length along the coordinate points.

**Returns** array(N)

**dists**

distances between spatial steps along the tracked field line :return: self._dists

**impact_angle_cos**()

Impact angle calculation - dot product of PFC norm and local magnetic field direction. Internally uses *incidence_angle_sin* function where *vecs* are replaced by the vector of the magnetic field.

**Returns** array of impact angles cosines

**impact_angle_sin**()

Impact angle calculation - dot product of PFC norm and local magnetic field direction. Internally uses *incidence_angle_sin* function where *vecs* are replaced by the vector of the magnetic field.

**Returns** array of impact angles sines

**impact_angle_sin_pol_projection**()

Impact angle calculation - dot product of PFC norm and local magnetic field direction poloidal projection only. Internally uses *incidence_angle_sin* function where *vecs* are replaced by the vector of the poloidal magnetic field (Bphi = 0).

**Returns** array of impact angles cosines

**incidence_angle_cos**(*vecs*)

**Parameters** **vecs** – array (3, N_vecs)

**Returns** array of cosines of angles of incidence

**incidence_angle_sin**(*vecs*)

**Parameters** **vecs** – array (3, N_vecs)

**Returns** array of sines of angles of incidence

**intersection**(*coords2*, *dim=None*)
input: 2 sets of coordinates crossection of two lines (2 sets of coordinates)

**Parameters** **dim** – reduce number of dimension in which is the intersection searched

**Returns**

**length**
Total length along the coordinate points.

**Returns** length in meters

**line_integral**(*func*, *method='sum'*)
func = /oint F(x,y) dl :param func: self - func(X, Y), Union[ndarray, int, float] or function values or 2D spline :param method: str, [‘sum’, ‘trapz’, ‘simps’] :return:

**mesh**()

**normal_vector**()
Calculate limiter normal vector with fw input directly from eq class

**Parameters** **first_wall** – interpolated first wall

**Returns** array (3, N_vecs) of limiter elements normals of the same

**phi**

**plot**(*ax=None*, ***kwargs*)

**Parameters**

- **ax** – Axis to which will be plotted. Default is plt.gca()
- **kwargs** – Arguments forwarded to matplotlib plot function.

**Returns**

**pol_projection_impact_angle_cos**()
Impact angle calculation - dot product of PFC norm and local magnetic field direction poloidal projection only. Internally uses *incidence_angle_sin* function where *vecs* are replaced by the vector of the poloidal magnetic field (Bphi = 0).

**Returns** array of impact angles cosines

**psi**

**psi_n**

**r**

**r_mid**

**resample**(*multiple=None*)
Return new, resampled instance of *pleque.Coordinates*

**Parameters** **multiple** – int, use multiple to multiply number of points.

**Returns** pleque.Coordinates

**resample2**(*npoints*)
Implicit spline curve interpolation for the limiter, number of points must be specified

**Parameters**

- **coords** – instance of coordinates object
- **npoints** – int - number of points of the result

**rho**

**theta**

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## p

# Index

## Symbols



## A



## B



## C



## D



## E

## F



## G



## I



## J



## L



## M



## N

## O



## P



## Q



## R



## S



## T

## V



## X



## Y



## Z