pleiades

Release 0.2.0-dev

Jul 13, 2020

Contents

The Pleiades code is designed for computing axisymmetric magnetic fields from combinations of electromagnets and permanent magnets with a focus on modeling plasma physics experiments. The project originated at the Wisconsin Plasma Physics Laboratory during experiments on the novel confinement device known as the Big Red Ball. Pleiades is written entirely in Python for now and is intended to be an easily accessible toolkit for computing and visualizing magnetic confinement geometries and data simultaneously.

Aside from simple computations of magnetic fields, there is a plasma MHD equilibrium solver incorporated into the package as well, which has been used to generate magnetic mirror equilibria and input files suitable for being run with the GENRAY and CQL3D codes for the design of the new Wisconsin HTS Axisymmetric Mirror (WHAM) and its eventual upgrade to a neutron source (WHAM-NS).

The Pleiades code is available on github. Any and all feedback and contributions are welcome!

CHAPTER 1

Quick Install Guide

To install the Pleiades code. Simply clone the github repository and install with `pip`. An example for how to do this on linux is shown below.

1.1 Installing on Linux

```
git clone https://github.com/eepeterson/pleiades
cd pleiades
pip install -e .
```

CHAPTER 2

Examples

Here are some (1 right now) example Jupyter Notebooks for showing users how to get started with the Pleiades package.

2.1 General Usage

2.1.1 Introduction

IPython notebooks must be viewed in the online HTML documentation.

2.1.2 Big Red Ball (BRB) Basic Usage

IPython notebooks must be viewed in the online HTML documentation.

2.1.3 Wisconsin HTS Axisymmetric Mirror (WHAM) Basic Usage

IPython notebooks must be viewed in the online HTML documentation.

2.1.4 Equilibrium Solver Basics

IPython notebooks must be viewed in the online HTML documentation.

CHAPTER 3

Python API

This page contains documentation on the Python modules, classes, and functions that comprise the pleiades package.

Modules

3.1 `pleiades` – Basic Functionality

3.1.1 Building Magnetic Configurations

pleiades.RectangularCoil	A rectangular cross section coil in the R-Z plane
pleiades.Device	A container for a full configuration of magnets for an experiment
pleiades.CurrentFilamentSet	Set of locations that have the same current value.
pleiades.FieldsOperator	Mixin class for computing fields on meshes
pleiades.compute_greens	Compute axisymmetric Green's functions for magnetic fields

pleiades.RectangularCoil

class pleiades.**RectangularCoil**(*r0=1.0*, *z0=0.0*, *nr=1*, *nz=1*, *dr=0.1*, *dz=0.1*, *angle=0.0*, ***kwargs*)

A rectangular cross section coil in the R-Z plane

Parameters

- **r0** (*float*) – The R location of the centroid of the coil
- **z0** (*float*) – The Z location of the centroid of the coil
- **nr** (*float, optional*) – The number of current filaments in the R direction. Defaults to 10.

- **nz** (*float, optional*) – The number of current filaments in the Z direction. Defaults to 10.
- **dr** (*float, optional*) – The distance between current filaments in the R direction. Defaults to 0.01 m
- **dz** (*float, optional*) – The distance between current filaments in the Z direction. Defaults to 0.01 m
- **nhat** (*iterable of float, optional*) – A vector of (dr, dz) representing the orientation of the coil and the 'local z direction'. This is the direction which applies to nz and dz when constructing current filament locations. The 'r' direction is found by the relation nhat x phi_hat = rhat. Defaults to (0, 1) meaning the local z axis is aligned with the global z axis and likewise for the r axis.
- ****kwargs** – Any valid keyword arguments for CurrentFilamentSet.

Variables

- **r0** (*float*) – The R location of the centroid of the Coil
- **z0** (*float*) – The Z location of the centroid of the Coil
- **centroid** (*np.array*) – Helper attribue for the R, Z location of the centroid of the Coil
- **nr** (*float*) – The number of current filaments in the R direction. Defaults to 10.
- **nz** (*float*) – The number of current filaments in the Z direction. Defaults to 10.
- **dr** (*float*) – The distance between current filaments in the R direction. Defaults to 0.1 m
- **dz** (*float*) – The distance between current filaments in the Z direction. Defaults to 0.1 m
- **angle** (*float*) – An angle in degrees representing the rotation of the coil and the 'local z direction' with respect to the global z axis. This is the direction which applies to nz and dz when constructing current filament locations. Defaults to 0 meaning the local z axis is aligned with the global z axis.
- **verts** (*np.ndarray*) – A 4x2 np.array representing the 4 vertices of the coil (read-only).
- **area** (*float*) – The area of the coil in m^2 (read-only).
- **current_density** (*float*) – The current density in the coil. This is equal to the total current divided by the area (read-only).

BR(*current=None*, *mesh=None*)

Compute the radial component of the magnetic field, BR.

Parameters **current** (*float, optional*) – Specify a current value to override the current attribute for calculating the field. Defaults to None, which causes the current attribute to be used for the calculation

Returns **BR**

Return type np.array

BZ(*current=None*, *mesh=None*)

Compute the z component of the magnetic field, BZ.

Parameters **current** (*float, optional*) – Specify a current value to override the current attribute for calculating the field. Defaults to None, which causes the current attribute to be used for the calculation

Returns **BZ**

Return type np.array

clone()
Create and return a copy of this coil

gBR(*mesh=None*)
Compute the Green's function for the radial magnetic field, BR

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate BR. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gBR** – 1D array representing the Green's function for BR and whose size is equal to the number of mesh.

Return type ndarray

gBZ(*mesh=None*)
Compute the Green's function for the vertical magnetic field, BZ

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate BZ. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gBZ** – 1D array representing the Green's function for BZ and whose size is equal to the number of mesh.

Return type ndarray

gpsi(*mesh=None*)
Compute the Green's function for magnetic flux, psi.

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate the magnetic flux. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gpsi** – 1D array representing the Green's function for flux and whose size is equal to the number of mesh.

Return type ndarray

plot(*ax*, *plot_patch=True*, ***kwargs*)
Plot the current locations for the CurrentGroup

Parameters

- **ax** (*matplotlib.Axes object*) – The axes object for plotting the current locations
- **plot_patch** (*bool*) – Whether to add the patch for this CurrentFilament to the axes.
- ****kwargs** (*dict, optional*) – Keyword arguments to pass to Current.plot method

psi(*current=None*, *mesh=None*)
Compute the magnetic flux, psi.

Parameters

- **current** (*float, optional*) – Specify a current value in amps to use instead of CurrentFilamentSet.current. Defaults to None, in which case the current attribute is used to calculate the flux.
- **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate the magnetic flux. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **psi**

Return type ndarray

rotate(*angle*, *pivot=(0.0, 0.0)*)
Rotate the current group by a given angle around a specified pivot

Parameters

- **angle** (*float*) – The angle of the rotation in degrees as measured from the z axis
- **pivot** (*iterable of float, optional*) – The (R, Z) location of the pivot. Defaults to (0., 0.).

simplify()
Create a single coil object from weighted sum of current centroids.

translate(*vector*)
Translate the current group by the vector (dr, dz)

Parameters **vector** (*iterable of float*) – The displacement vector for the translation

pleiades.Device

class pleiades.**Device**(***kwargs*)
A container for a full configuration of magnets for an experiment

Parameters ****kwargs** – Any valid argument for FieldsOperator

Variables

- **current_sets** (*iterable*) – List of CurrentFilamentSet objects
- **R** (*np.array*) – The R locations of the mesh
- **Z** (*np.array*) – The Z locations of the mesh
- **patches** (*list*) – A list of patch objects for the configuration
- **patch_coll** (*matplotlib.patches.PatchCollection*) – A patch collection for easier adding to matplotlib axes

BR(*current=None*, *mesh=None*)
Compute the radial component of the magnetic field, BR.

Parameters **current** (*float, optional*) – Specify a current value to override the current attribute for calculating the field. Defaults to None, which causes the current attribute to be used for the calculation

Returns **BR**

Return type np.array

BZ(*current=None*, *mesh=None*)
Compute the z component of the magnetic field, BZ.

Parameters **current** (*float, optional*) – Specify a current value to override the current attribute for calculating the field. Defaults to None, which causes the current attribute to be used for the calculation

Returns **BZ**

Return type np.array

gBR(*mesh=None*)
Compute the Green's function for the radial magnetic field, BR

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate BR. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gBR** – 1D array representing the Green's function for BR and whose size is equal to the number of mesh.

Return type ndarray

gBZ(*mesh=None*)
Compute the Green's function for the vertical magnetic field, BZ

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate BZ. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gBZ** – 1D array representing the Green's function for BZ and whose size is equal to the number of mesh.

Return type ndarray

gpsi(*mesh=None*)
Compute the Green's function for magnetic flux, psi.

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate the magnetic flux. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gpsi** – 1D array representing the Green's function for flux and whose size is equal to the number of mesh.

Return type ndarray

psi(*current=None*, *mesh=None*)
Compute the magnetic flux, psi.

Parameters

- **current** (*float, optional*) – Specify a current value in amps to use instead of CurrentFilamentSet.current. Defaults to None, in which case the current attribute is used to calculate the flux.
- **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate the magnetic flux. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **psi**

Return type ndarray

pleiades.CurrentFilamentSet

class pleiades.**CurrentFilamentSet**(*current=1.0*, *weights=None*, *patch_kw=None*, ***kwargs*)
Set of locations that have the same current value.

A CurrentFilamentSet represents a set of axisymmetric current centroids with associated current weights to describe the current ratios between all the centroids. In addition, a CurrentFilamentSet implements the Green's function functionality for computing magnetic fields and flux on an R-Z mesh. A CurrentFilamentSet is not intended to be instatiated directly, but serves as the base class for all concrete current set classes and defines the minimum functional interface and protocols of a current set. Lastly a matplotlib.patches.PathPatch object is associated with each CurrentFilamentSet for ease in plotting and verifying device geometry.

Parameters

- **current** (*float, optional*) – The current to be used for calculating fields from the Green's functions. Defaults to 1 amp.
- **weights** (*iterable, optional*) – The weights for all the current locations. The current weight is effectively a current multiplier for a given position that is incorporated into the Green's function. This enables having both positive and negative currents in an object at the same time as well as current profile shaping in the case of permanent magnets. Defaults to 1 for every location.
- **patch_kw** (*dict*) – Dictionary of any matplotlib.patches.Patch keyword arguments. No type checking is performed on these inputs they are simply assigned to the patch_kw attribute.
- ****kwargs** – Any keyword arguments intended to be passed to FieldsOperator object using cooperative inheritance.

Variables

- **current** (*float*) – The current to be used for calculating fields from the Green's functions. Defaults to 1 amp.
- **weights** (*iterable*) – The weights for all the current locations. The current weight is effectively a current multiplier for a given position that is incorporated into the Green's function. This enables having both positive and negative currents in an object at the same time as well as current profile shaping in the case of permanent magnets. Defaults to 1 for every location.
- **npts** (*int*) – Integer for the number of current filaments in this CurrentFilamentSet (read-only).
- **rz_pts** (*ndarray*) – An Nx2 array representing (R, Z) coordinates for current centroids. Units are meters and the coordinate system is cylindrical (read-only)
- **rzw** (*np.ndarray*) – An Nx3 array whos columns describe the current centroid radial coordinate (R), vertical coordinate (Z), and current weight (W) for each filament in the CurrentFilamentSet (read-only).
- **total_current** (*float*) – The total current being carried in the filament set. This is equal to the current times the sum of the weights.
- **patch_kw** (*dict*) – A dictionary of valid matplotlib.patches.Patch keyword arguments

clone()
Create and return a copy of this coil

plot(*ax*, *plot_patch=True*, ***kwargs*)
Plot the current locations for the CurrentGroup

Parameters

- **ax** (*matplotlib.Axes object*) – The axes object for plotting the current locations
- **plot_patch** (*bool*) – Whether to add the patch for this CurrentFilament to the axes.
- ****kwargs** (*dict, optional*) – Keyword arguments to pass to Current.plot method

rotate(*angle*, *pivot=(0.0, 0.0)*)
Rotate the current group by a given angle around a specified pivot

Parameters

- **angle** (*float*) – The angle of the rotation in degrees as measured from the z axis

- **pivot** (*iterable of float, optional*) – The (R, Z) location of the pivot. Defaults to (0., 0.).

simplify()
Create a single coil object from weighted sum of current centroids.

translate(*vector*)
Translate the current group by the vector (dr, dz)

Parameters **vector** (*iterable of float*) – The displacement vector for the translation

pleiades.FieldsOperator

class pleiades.**FieldsOperator**(*mesh=None*, *rank=1*, ***kwargs*)
Mixin class for computing fields on meshes

Parameters

- **mesh** (*pleiades.Mesh object*) – The mesh to use for calculating fields
- **rank** (*int (1 or 2)*) – Indicator of whether the current attribute is a scalar or vector
- **Variables** –
- **---------** –
- **current** (*float or ndarray*) – Current values in this object
- **rzw** (*ndarray or list of ndarray*) – Nx3 arrays of centroid positions and weights
- **mesh** – The mesh to use for calculating fields

BR(*current=None*, *mesh=None*)
Compute the radial component of the magnetic field, BR.

Parameters **current** (*float, optional*) – Specify a current value to override the current attribute for calculating the field. Defaults to None, which causes the current attribute to be used for the calculation

Returns **BR**

Return type np.array

BZ(*current=None*, *mesh=None*)
Compute the z component of the magnetic field, BZ.

Parameters **current** (*float, optional*) – Specify a current value to override the current attribute for calculating the field. Defaults to None, which causes the current attribute to be used for the calculation

Returns **BZ**

Return type np.array

gBR(*mesh=None*)
Compute the Green's function for the radial magnetic field, BR

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate BR. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gBR** – 1D array representing the Green's function for BR and whose size is equal to the number of mesh.

Return type ndarray

gBZ(*mesh=None*)
Compute the Green's function for the vertical magnetic field, BZ

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate BZ. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gBZ** – 1D array representing the Green's function for BZ and whose size is equal to the number of mesh.

Return type ndarray

gpsi(*mesh=None*)
Compute the Green's function for magnetic flux, psi.

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate the magnetic flux. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gpsi** – 1D array representing the Green's function for flux and whose size is equal to the number of mesh.

Return type ndarray

psi(*current=None*, *mesh=None*)
Compute the magnetic flux, psi.

Parameters

- **current** (*float, optional*) – Specify a current value in amps to use instead of CurrentFilamentSet.current. Defaults to None, in which case the current attribute is used to calculate the flux.
- **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate the magnetic flux. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **psi**

Return type ndarray

pleiades.compute_greens

pleiades.**compute_greens**(*rzw*, *rz_pts*)
Compute axisymmetric Green's functions for magnetic fields

Parameters

- **rzw** (*ndarray or iterable of ndarray*) – An Nx3 array whose columns are r locations, z locations, and current weights respectively for the current filaments.
- **rz_pts** (*Nx2 np.array*) – An Nx2 array whose columns are r locations and z locations for the mesh points where we want to calculate the Green's functions.

Returns 3-tuple of 1D np.array representing the Green's function for psi, BR, and Bz respectively.

Return type tuple

3.1.2 Building Meshes

pleiades.Mesh	A mesh in the R-Z plane for calculating magnetic fields and flux.
pleiades.RectMesh	A regular linearly spaced 2D R-Z mesh.
pleiades.PointsMesh	An unstructured mesh specified by two lists of coordinate points.
pleiades.RChord	A 1D chord of cylindrical radii at fixed height z.
pleiades.ZChord	A 1D chord of Z values at fixed cylindrical radius.
pleiades.SphericalRChord	A 1D chord of spherical radius at fixed polar angle theta.
pleiades.ThetaChord	A 1D chord of polar angles at fixed spherical radius r.

pleiades.Mesh

class pleiades.**Mesh**

A mesh in the R-Z plane for calculating magnetic fields and flux.

The Mesh class is a base class for concrete representations of meshs that may be used in mesh based calculations. These meshs can be 1D or 2D structured meshs as well as a list of arbitrary coordinate pairs as in the case of unstructured meshs. This class outlines the interface and protocols that define a mesh in the Pleiades package.

Variables

- **R** (*np.ndarray*) – An N-dimensional array (typically 1 or 2) whose values represent the radial coordinates for the mesh in a cylindrical coordinate frame.
- **Z** (*np.ndarray*) – An N-dimensional array (typically 1 or 2) whose values represent the z coordinates for the mesh in a cylindrical coordinate frame.
- **r** (*np.ndarray*) – An N-dimensional array (typically 1 or 2) whose values represent the r coordinates for the mesh in a spherical coordinate frame.
- **theta** (*np.ndarray*) – An N-dimensional array (typically 1 or 2) whose values represent the theta coordinates for the mesh in a spherical coordinate frame.

classmethod to_points(*mesh*)

Take a mesh or numpy array, return Nx2 points array

pleiades.RectMesh

class pleiades.**RectMesh**(*rmin=0.0*, *rmax=1.0*, *nr=101*, *zmin=-0.5*, *zmax=0.5*, *nz=101*)

A regular linearly spaced 2D R-Z mesh.

Parameters

- **rmin** (*float, optional*) – The minimum cylindrical radius for the mesh in meters. Defaults to 0 m.
- **rmax** (*float, optional*) – The maximum cylindrical radius for the mesh in meters. Defaults to 0 m.
- **nr** (*float, optional*) – The number of radial mesh points. Defaults to 101.
- **zmin** (*float, optional*) – The minimum z height for the mesh in meters. Defaults to 0 m.
- **zmax** (*float, optional*) – The maximum z height for the mesh in meters. Defaults to 0 m.

- **nz** (*float, optional*) – The number of z mesh points. Defaults to 101.

Variables

- **rmin** (*float*) – The minimum cylindrical radius for the mesh in meters.
- **rmax** (*float*) – The maximum cylindrical radius for the mesh in meters.
- **nr** (*float*) – The number of radial mesh points.
- **zmin** (*float*) – The minimum z height for the mesh in meters.
- **zmax** (*float*) – The maximum z height for the mesh in meters.
- **nz** (*float*) – The number of z mesh points.

classmethod to_points(*mesh*)
Take a mesh or numpy array, return Nx2 points array

pleiades.PointsMesh

class pleiades.**PointsMesh**(*rpts*, *zpts*)
An unstructured mesh specified by two lists of coordinate points.

Parameters

- **rpts** (*iterable of float*) – List of cylindrical r values for the unstructured mesh in meters.
- **zpts** (*iterable of float*) – List of z values for the unstructured mesh in meters.

classmethod to_points(*mesh*)
Take a mesh or numpy array, return Nx2 points array

pleiades.RChord

class pleiades.**RChord**(*rpts*, *z0=0.0*)
A 1D chord of cylindrical radii at fixed height z.

Parameters

- **rpts** (*iterable of float*) – List of cylindrical r values for the chord in meters.
- **z0** (*float, optional*) – Height of the cylindrical chord in meters. Defaults to 0 m.

classmethod to_points(*mesh*)
Take a mesh or numpy array, return Nx2 points array

pleiades.ZChord

class pleiades.**ZChord**(*zpts*, *r0=0.0*)
A 1D chord of Z values at fixed cylindrical radius.

Parameters

- **zpts** (*iterable of float*) – List of z values in the chord in meters.
- **r0** (*float, optional*) – Cylindrical radius for the z chord. Defaults to 0 m.

classmethod to_points(*mesh*)
Take a mesh or numpy array, return Nx2 points array

pleiades.SphericalRChord

class pleiades.**SphericalRChord**(*rpts*, *theta0=0.0*)
A 1D chord of spherical radius at fixed polar angle theta.

Parameters

- **rpts** (*iterable of float*) – List of theta points in the chord in degrees.
- **theta0** (*float, optional*) – Polar angle for the radial chord in degrees. Defaults to 0.

classmethod to_points(*mesh*)
Take a mesh or numpy array, return Nx2 points array

pleiades.ThetaChord

class pleiades.**ThetaChord**(*tpts*, *r0=1.0*)
A 1D chord of polar angles at fixed spherical radius r.

Parameters

- **tpts** (*iterable of float*) – List of theta points in the chord in degrees.
- **r0** (*float, optional*) – Spherical radius for the chord. Defaults to 1.0 m

classmethod to_points(*mesh*)
Take a mesh or numpy array, return Nx2 points array

3.1.3 WIPPL Magnetic Configurations

pleiades.configurations.BRB	The Device object representing the Big Red Ball at UW-Madison.
pleiades.configurations.PCX_magCage	
pleiades.configurations.PCX_HH	

pleiades.configurations.BRB

class pleiades.configurations.**BRB**
The Device object representing the Big Red Ball at UW-Madison.

Variables

- **hh_n** (*TREXCoil object*) – A coil object for the north helmholtz coil
- **hh_s** (*TREXCoil object*) – A coil object for the south helmholtz coil
- **ltrx_n** (*RectangularCoil object*) – A coil for the north LTRX mirror coil
- **ltrx_s** (*RectangularCoil object*) – A coil for the south LTRX mirror coil
- **mr1** (*MagnetRing object*) – The magnet ring at 87.5 degrees N
- **mr2** (*MagnetRing object*) – The magnet ring at 82.5 degrees N
- **mr3** (*MagnetRing object*) – The magnet ring at 77.5 degrees N
- **mr4** (*MagnetRing object*) – The magnet ring at 72.5 degrees N
- **mr5** (*MagnetRing object*) – The magnet ring at 67.5 degrees N

- **mr6** (*MagnetRing object*) – The magnet ring at 62.5 degrees N
- **mr7** (*MagnetRing object*) – The magnet ring at 57.5 degrees N
- **mr8** (*MagnetRing object*) – The magnet ring at 52.5 degrees N
- **mr9** (*MagnetRing object*) – The magnet ring at 47.5 degrees N
- **mr10** (*MagnetRing object*) – The magnet ring at 42.5 degrees N
- **mr11** (*MagnetRing object*) – The magnet ring at 37.5 degrees N
- **mr12** (*MagnetRing object*) – The magnet ring at 32.5 degrees N
- **mr13** (*MagnetRing object*) – The magnet ring at 27.5 degrees N
- **mr14** (*MagnetRing object*) – The magnet ring at 22.5 degrees N
- **mr15** (*MagnetRing object*) – The magnet ring at 17.5 degrees N
- **mr16** (*MagnetRing object*) – The magnet ring at 12.5 degrees N
- **mr17** (*MagnetRing object*) – The magnet ring at 7.5 degrees N
- **mr18** (*MagnetRing object*) – The magnet ring at 2.5 degrees N
- **mr19** (*MagnetRing object*) – The magnet ring at 2.5 degrees S
- **mr20** (*MagnetRing object*) – The magnet ring at 7.5 degrees S
- **mr21** (*MagnetRing object*) – The magnet ring at 12.5 degrees S
- **mr22** (*MagnetRing object*) – The magnet ring at 17.5 degrees S
- **mr23** (*MagnetRing object*) – The magnet ring at 22.5 degrees S
- **mr24** (*MagnetRing object*) – The magnet ring at 27.5 degrees S
- **mr25** (*MagnetRing object*) – The magnet ring at 32.5 degrees S
- **mr26** (*MagnetRing object*) – The magnet ring at 37.5 degrees S
- **mr27** (*MagnetRing object*) – The magnet ring at 42.5 degrees S
- **mr28** (*MagnetRing object*) – The magnet ring at 47.5 degrees S
- **mr29** (*MagnetRing object*) – The magnet ring at 52.5 degrees S
- **mr30** (*MagnetRing object*) – The magnet ring at 57.5 degrees S
- **mr31** (*MagnetRing object*) – The magnet ring at 62.5 degrees S
- **mr32** (*MagnetRing object*) – The magnet ring at 67.5 degrees S
- **mr33** (*MagnetRing object*) – The magnet ring at 72.5 degrees S
- **mr34** (*MagnetRing object*) – The magnet ring at 77.5 degrees S
- **mr35** (*MagnetRing object*) – The magnet ring at 82.5 degrees S
- **mr36** (*MagnetRing object*) – The magnet ring at 87.5 degrees S

BR(*current=None*, *mesh=None*)

Compute the radial component of the magnetic field, BR.

Parameters **current** (*float, optional*) – Specify a current value to override the current attribute for calculating the field. Defaults to None, which causes the current attribute to be used for the calculation

Returns BR

Return type np.array

BZ(*current=None*, *mesh=None*)
Compute the z component of the magnetic field, BZ.

Parameters **current** (*float, optional*) – Specify a current value to override the current attribute for calculating the field. Defaults to None, which causes the current attribute to be used for the calculation

Returns **BZ**

Return type np.array

gBR(*mesh=None*)
Compute the Green's function for the radial magnetic field, BR

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate BR. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gBR** – 1D array representing the Green's function for BR and whose size is equal to the number of mesh.

Return type ndarray

gBZ(*mesh=None*)
Compute the Green's function for the vertical magnetic field, BZ

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate BZ. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gBZ** – 1D array representing the Green's function for BZ and whose size is equal to the number of mesh.

Return type ndarray

gpsi(*mesh=None*)
Compute the Green's function for magnetic flux, psi.

Parameters **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate the magnetic flux. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **gpsi** – 1D array representing the Green's function for flux and whose size is equal to the number of mesh.

Return type ndarray

psi(*current=None*, *mesh=None*)
Compute the magnetic flux, psi.

Parameters

- **current** (*float, optional*) – Specify a current value in amps to use instead of CurrentFilamentSet.current. Defaults to None, in which case the current attribute is used to calculate the flux.
- **mesh** (*ndarray, optional*) – An Nx2 array of points representing (R, Z) coordinates at which to calculate the magnetic flux. Defaults to None, in which case the CurrentFilamentSet.mesh attribute is used.

Returns **psi**

Return type ndarray

3.1.4 Computing Equilibria

pleiades.compute_equilibrium	Compute jxB=gradP equilibrium for given P as function of R
pleiades.mirror_equilibrium	Compute jxB=gradP equilibrium for given P as function of psi/psi_lim given by P(psi) = p0*(1-(psi/psi_lim)**alpha1)**alpha2

pleiades.compute_equilibrium

pleiades.**compute_equilibrium**(*R*, *Z*, *Pfunc*, *psi_vac*, *g_psi*, *tol=1e-10*, *maxiter=100*, *relax=0.0*, *plas_clip=None*, *plotit=False*)

Compute jxB=gradP equilibrium for given P as function of R

pleiades.mirror_equilibrium

pleiades.**mirror_equilibrium**(*brb*, *gplas*, *p0*, *alpha1*, *alpha2*, *tol=1e-10*, *maxiter=100*, *relax=0.0*, *plotit=False*)

Compute jxB=gradP equilibrium for given P as function of psi/psi_lim given by P(psi) = p0*(1-(psi/psi_lim)**alpha1)**alpha2

3.1.5 IO Functionality

pleiades.write_eqdsk

pleiades.write_eqdsk

pleiades.**write_eqdsk**(*Rho*, *Z*, *psi*, *plas_currents*, *fname*, *title*)

3.2 pleiades.configurations – Existing Experimental Facility Models

3.2.1 WIPPL Magnetic Configurations

pleiades.configurations.BRB	The Device object representing the Big Red Ball at UW-Madison.
pleiades.configurations.PCX_magCage	
pleiades.configurations.PCX_HH	

3.3 pleiades.analysis – Analysis Functions

3.3.1 Mathematical Functions

pleiades.analysis.get_gpsi	

pleiades.analysis.get_gpsi

pleiades.analysis.**get_gpsi**(*R*, *Z*)

3.3.2 Plotting Helper Functions

pleiades.analysis.get_mirror_patches	Get mirror image patches across desired axis.
pleiades.analysis.scale_patches	scale patches by desired factor.
pleiades.analysis.transpose_patches	Transpose patches (reflect across line y=x).

pleiades.analysis.get_mirror_patches

pleiades.analysis.**get_mirror_patches**(*patches*, *axis=0*, *scale=1*)
Get mirror image patches across desired axis. axis = 0 means reflect across x axis, axis = 1 means reflect across y axis. Optional scale argument can be used as well.

pleiades.analysis.scale_patches

pleiades.analysis.**scale_patches**(*patches*, *scale*)
scale patches by desired factor.

pleiades.analysis.transpose_patches

pleiades.analysis.**transpose_patches**(*patches*)
Transpose patches (reflect across line y=x).

3.3.3 Interpolation Helper Functions

pleiades.analysis.contour_points	
pleiades.analysis.reflect_and_hstack	Reflect and concatenate grid and quantities in args to plot both half planes (rho>=0 and rho<=0).
pleiades.analysis.regular_grid	
pleiades.analysis.poly_fit	
pleiades.analysis.transform_to_rtheta	Transform Rho Z grid and rho,z components of vector field to polar coordinates
pleiades.analysis.transform_to_rhoz	Transform R Theta grid and r theta components of vector field to cylindrical coordinates
pleiades.analysis.locs_to_vals	Picks values of field Q at desired coordinates.
pleiades.analysis.locs_to_vals_griddata	Picks values of field Q at desired coordinates.
pleiades.analysis.locs_to_vals1D	Picks values of field Q at desired coordinates.

pleiades.analysis.contour_points

pleiades.analysis.**contour_points**(*contourset*)

pleiades.analysis.reflect_and_hstack

pleiades.analysis.**reflect_and_hstack**(*Rho*, *Z*, **args*)

Reflect and concatenate grid and quantities in args to plot both half planes (rho>=0 and rho<=0). Currently this function only reflects across the z axis since that is the symmetry convention we've taken for the machine.

Parameters

- **Rho** (*np.array*) – 2D array for the R coordinates of the grid
- **Z** (*np.array*) – 2D array for the Z coordinates of the grid
- **args** (*tuple*) – 2D arrays of any quantity on this grid you wish to plot in both half planes

pleiades.analysis.regular_grid

pleiades.analysis.**regular_grid**(*xx*, *yy*, **args*, ***kwargs*)

pleiades.analysis.poly_fit

pleiades.analysis.**poly_fit**(*x*, *y*, *order=3*)

pleiades.analysis.transform_to_rtheta

pleiades.analysis.**transform_to_rtheta**(*Rho*, *Z*, *rho_component*, *z_component*)

Transform Rho Z grid and rho,z components of vector field to polar coordinates

pleiades.analysis.transform_to_rhoz

pleiades.analysis.**transform_to_rhoz**(*R*, *Theta*, *r_component*, *theta_component*)

Transform R Theta grid and r theta components of vector field to cylindrical coordinates

pleiades.analysis.locs_to_vals

pleiades.analysis.**locs_to_vals**(*X*, *Y*, *Q*, *coord_list*)

Picks values of field Q at desired coordinates.

Parameters

- **X** (*2D array*) – grid representing column coordinates of Q
- **Y** (*2D array*) – grid representing row coordinates of Q
- **Q** (*2D array*) – value of Q on grid
- **coord_list** (*list*) – list of tuples (x,y) for desired coordinates

pleiades.analysis.locs_to_vals_griddata

pleiades.analysis.**locs_to_vals_griddata**(*X*, *Y*, *Q*, *coord_list*)

Picks values of field Q at desired coordinates.

Parameters

- **X** (*2D array*) – grid representing column coordinates of Q

- **Y** (*2D array*) – grid representing row coordinates of Q
- **Q** (*2D array*) – value of Q on grid
- **coord_list** (*list*) – list of tuples (x,y) for desired coordinates

pleiades.analysis.locs_to_vals1D

pleiades.analysis.**locs_to_vals1D**(*X*, *Y*, *Q*, *coord_list*)

Picks values of field Q at desired coordinates.

Parameters

- **X** (*2D array*) – grid representing column coordinates of Q
- **Y** (*2D array*) – grid representing row coordinates of Q
- **Q** (*2D array*) – value of Q on grid
- **coord_list** (*list*) – list of tuples (x,y) for desired coordinates

3.3.4 Fieldline Analysis Functions

pleiades.analysis.get_fieldlines	Return coordinates for segments comprising a flux surface (Nx2 array).
pleiades.analysis.parse_segment	
pleiades.analysis.get_fieldline_distance	Return cumulative field line distance vector
pleiades.analysis.interp	interpolate quantity Q on Rho, Z grid onto flpoints (Nx2 array of x,y pairs).
pleiades.analysis.flux_surface_avg	Compute flux surface average of quantity Q or return dVdpsi (dl_B)
pleiades.analysis.diff_central	

pleiades.analysis.get_fieldlines

pleiades.analysis.**get_fieldlines**(*contourset*, *level*, *start_coord=None*, *end_coord=None*, *clockwise=True*, *idx_check=[]*)

Return coordinates for segments comprising a flux surface (Nx2 array).

Parameters

- **contourset** (*matplotlib.contour.QuadContourSet instance*) – i.e.
- **call** (*ax.contour*) –
- **level** (*float*) – desired contour level
- **start_coord** (*tuple*) – coordinate (x,y) at which to start the field line
- **end_coord** (*tuple*) – coordinate (x,y) at which to end the field line
- **clockwise** (*bool*) – whether to order the field line coordinates clockwise or
- **counterclockwise** –

pleiades.analysis.parse_segment

pleiades.analysis.**parse_segment**(*flpoints*, *start_coord=None*, *end_coord=None*, *clockwise=True*)

pleiades.analysis.get_fieldline_distance

pleiades.analysis.**get_fieldline_distance**(*flpoints*)
Return cumulative field line distance vector

pleiades.analysis.interp

pleiades.analysis.**interp**(*Rho*, *Z*, *Q*, *flpoints*)
interpolate quantity Q on Rho, Z grid onto flpoints (Nx2 array of x,y pairs).

pleiades.analysis.flux_surface_avg

pleiades.analysis.**flux_surface_avg**(*Rho*, *Z*, *B*, *flpoints*, *Q=None*)
Compute flux surface average of quantity Q or return dVdpsi (dl_B)

pleiades.analysis.diff_central

pleiades.analysis.**diff_central**(*x*, *y*)

3.3.5 Miscellaneous Helper Functions

pleiades.analysis.get_deltapsi	Returns contribution to psi from fast ion currents.
pleiades.analysis.get_concave_hull	

pleiades.analysis.get_deltapsi

pleiades.analysis.**get_deltapsi**(*data*, *Req*, *Zeq*)
Returns contribution to psi from fast ion currents.

Parameters

- **data** (*netcdf4 Dataset object*) –
- **Req** (*2D R grid from eqdsk*) –
- **Zeq** (*2D Z grid from eqdsk*) –

Returns deltapsi (psi from fast ion currents on eqdsk grid)

pleiades.analysis.get_concave_hull

pleiades.analysis.**get_concave_hull**(*Rho*, *Z*, *Q*)

CHAPTER 4

Gallery

This section will be filled in later with example plots of experiments as well as plots for publications that used Pleiades

CHAPTER 5

Publications

5.1 Used Pleiades in Analysis

- Ethan E. Peterson, Douglass A. Endrizzi, Matthew Beidler, Kyle J. Bunkers, Michael Clark, Jan Egedal, Ken Flanagan, Karsten J. McCollam, Jason Milhone, Joseph Olson, Carl R. Sovinec, Roger Waleffe, John Wallace, and Cary B. Forest, "A laboratory model for the Parker spiral and magnetized stellar winds," *Nature Physics*, **15**, 1095–1100 (2019).

CHAPTER 6

Resources and Reference Material

Below is a collection of resources for specific computing and software development resources we've curated in an attempt to provide a quick reference page for anybody starting out on a project like this.

6.1 Linux

Just for reference, the Linux Kernel Documentation uses Read the Docs to host their official documentation. So, if it's good enough for the kernel developers, it's good enough for us.

The Bash Reference Manual is super helpful for learning how to write bash scripts for manipulating files, running programs, and in general for executing things on the commandline.

The Linux Filesystem Hierarchy Standard is a helpful reference for how files are organized and common conventional uses of each directory. Understanding the purpose of the handful of directories under the root directory can go a long way towards helping you get comfortable with Linux and give you more control over your system. Also, because what the hell is . . . AppDataRoaming

6.2 Python

Python is the most popular programming language and is still growing. There are many reasons for this including general purpose capabilites as well as visualization toolkits and scientific computing libraries that are all user-friendly (more or less). The current supported version of Python is Python 3 as Python 2 support was ended in January 2020.

6.3 Scientific Computing with Python

The recommended Python distribution for scientific computing and data analysis is Anaconda 3 as it provides a very simple way to access many of most popular packages like numpy, scipy, pandas, matplotlib, scikit-learn and tensorflow. In addition many third-party packages are available for install through the conda package manager, including OpenMC.

6.4 Data Visualization

6.5 Git and Software Development

Git is a powerful tool for software development and version control for projects of all sizes, big or small. There are a number of helpful tutorials out there on how to use git,

Often times using git at the command line can be daunting for beginners. However, having a good understanding of proper git workflows can go a long way towards easing that anxiety. A very helpful reference that clearly explains a good git workflow including branching, merging, and releases can be found here

6.6 Program Design

CHAPTER 7

License Agreement

Index

S

T

W

Z