eqtools Documentation

Release 0.0

Mark Chilenski, Ian Faust and John Walk

November 21, 2013

CONTENTS

CHAPTER

ONE

OVERVIEW

Useful description and details go here.

CHAPTER

TWO

PACKAGE REFERENCE

2.1 eqtools Package

2.1.1 `eqtools` Package

Provides classes for interacting with magnetic equilibrium data in a variety of formats.

2.1.2 `CModEFIT` Module

This module provides classes for working with C-Mod EFIT data.

class `eqtools.CModEFIT.`**`CModEFITTree`**(*shot*, *tree='ANALYSIS'*, *length_unit='m'*, *gfile='g_eqdsk'*, *afile='a_eqdsk'*, *tspline=False*, *monotonic=False*)

Bases: `eqtools.EFIT.EFITTree`

Inherits `EFITTree` class. Machine-specific data handling class for Alcator C-Mod. Pulls EFIT data from selected MDS tree and shot, stores as object attributes. Each EFIT variable or set of variables is recovered with a corresponding getter method. Essential data for EFIT mapping are pulled on initialization (e.g. psirz grid). Additional data are pulled at the first request and stored for subsequent usage.

Intializes C-Mod version of EFITTree object. Pulls data from MDS tree for storage in instance attributes. Core attributes are populated from the MDS tree on initialization. Additional attributes are initialized as None, filled on the first request to the object.

Parameters **shot** – (long) int C-Mod shot index (long)

Keyword Arguments

- **tree** – str Optional input for EFIT tree, defaults to 'ANALYSIS' (i.e., EFIT data are under analysis::top.efit.results). For any string TREE (such as 'EFIT20') other than 'ANALYSIS', data are taken from TREE::top.results.
- **length_unit** – str Sets the base unit used for any quantity whose dimensions are length to any power. Valid options are:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'de-fault'	whatever the default in the tree is (no conversion is performed, units may be inconsistent)

Default is 'm' (all units taken and returned in meters).

- **gfile** – str Optional input for EFIT geqdsk location name, defaults to 'g_eqdsk' (i.e., EFIT data are under tree::top.results.G_EQDSK)
- **afile** – str Optional input for EFIT aeqdsk location name, defaults to 'a_eqdsk' (i.e., EFIT data are under tree::top.results.A_EQDSK)
- **tspline** – Boolean Sets whether or not interpolation in time is performed using a tricubic spline or nearest-neighbor interpolation. Tricubic spline interpolation requires at least four complete equilibria at different times. It is also assumed that they are functionally correlated, and that parameters do not vary out of their boundaries (derivative = 0 boundary condition). Default is False (use nearest neighbor interpolation).
- **monotonic** – Boolean Sets whether or not the "monotonic" form of time window finding is used. If True, the timebase must be monotonically increasing. Default is False (use slower, safer method).

`getMachineCrossSectionFull`()

Pulls C-Mod cross-section data from tree, converts to plottable vector format for use in other plotting routines

Parameters **shot** – (long) int C-Mod shot index (used for tree access)

Returns *(x, y)* – The coordinates of the machine cross-section.

class `eqtools.CModEFIT.`**`CModEFITTreeProp`**(*shot*, *tree='ANALYSIS'*, *length_unit='m'*, *gfile='g_eqdsk'*, *afile='a_eqdsk'*, *tspline=False*, *monotonic=False*)

Bases: `eqtools.CModEFIT.CModEFITTree`, `eqtools.core.PropertyAccessMixin`

CModEFITTree with the PropertyAccessMixin added to enable property-style access. This is good for interactive use, but may drag the performance down.

2.1.3 `EFIT` Module

class `eqtools.EFIT.`**`EFITTree`**(*shot*, *tree*, *root*, *length_unit='m'*, *gfile='g_eqdsk'*, *afile='a_eqdsk'*, *tspline=False*, *monotonic=False*)

Bases: `eqtools.core.Equilibrium`

Inherits Equilibrium class. EFIT-specific data handling class for machines using standard EFIT tag names/tree structure with MDSplus. Constructor and/or data loading may need overriding in a machine-specific implementation. Pulls EFIT data from selected MDS tree and shot, stores as object attributes. Each EFIT variable or set of variables is recovered with a corresponding getter method. Essential data for EFIT mapping are pulled on initialization (e.g. psirz grid). Additional data are pulled at the first request and stored for subsequent usage.

Intializes EFITTree object. Pulls data from MDS tree for storage in instance attributes. Core attributes are populated from the MDS tree on initialization. Additional attributes are initialized as None, filled on the first request to the object.

Parameters

- **shot** – int shot number
- **tree** – tree MDSplus tree to open to fetch EFIT data.
- **root** – str Root path for EFIT data in MDSplus tree.

Keyword Arguments

- **length_unit** – String. Sets the base unit used for any quantity whose dimensions are length to any power. Valid options are:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	whatever the default in the tree is (no conversion is performed, units may be inconsistent)

  Default is 'm' (all units taken and returned in meters).
- **tspline** – Boolean. Sets whether or not interpolation in time is performed using a tricubic spline or nearest-neighbor interpolation. Tricubic spline interpolation requires at least four complete equilibria at different times. It is also assumed that they are functionally correlated, and that parameters do not vary out of their boundaries (derivative = 0 boundary condition). Default is False (use nearest neighbor interpolation).
- **monotonic** – Boolean. Sets whether or not the "monotonic" form of time window finding is used. If True, the timebase must be monotonically increasing. Default is False (use slower, safer method).

`__str__``()`

string formatting for EFITTree class.

`getInfo``()`

returns namedtuple of shot information

Returns

namedtuple containing

shot	C-Mod shot index (long)
tree	EFIT tree (string)
nr	size of R-axis for spatial grid
nz	size of Z-axis for spatial grid
nt	size of timebase for flux grid

`getTimeBase``()`

returns EFIT time base vector

`getFluxGrid``()`

returns EFIT flux grid, [t,z,r]

getRGrid(*length_unit=1*)
returns EFIT R-axis [r]

getZGrid(*length_unit=1*)
returns EFIT Z-axis [z]

getFluxAxis()
returns psi on magnetic axis [t]

getFluxLCFS()
returns psi at separatrix [t]

getFluxVol(*length_unit=3*)
returns volume within flux surface [psi,t]

getVolLCFS(*length_unit=3*)
returns volume within LCFS [t]

getRmidPsi(*length_unit=1*)
returns maximum major radius of each flux surface [t,psi]

getRLCFS(*length_unit=1*)
returns R-values of LCFS position [t,n]

getZLCFS(*length_unit=1*)
returns Z-values of LCFS position [t,n]

remapLCFS(*mask=True*)
Overwrites RLCFS, ZLCFS values pulled from EFIT with explicitly-calculated contour of psinorm=1 surface. This is then masked down by the limiter array using core.inPolygon, restricting the contour to the closed plasma surface and the divertor legs.

Keyword Arguments **mask** – Boolean. Default True. Set True to mask LCFS path to limiter outline (using inPolygon). Set False to draw full contour of psi = psiLCFS.

getFluxPres()
returns pressure at flux surface [psi,t]

getElongation()
returns LCFS elongation [t]

getUpperTriangularity()
returns LCFS upper triangularity [t]

getLowerTriangularity()
returns LCFS lower triangularity [t]

getShaping()
pulls LCFS elongation and upper/lower triangularity

Returns namedtuple containing {kappa, delta_u, delta_l}

getMagR(*length_unit=1*)
returns magnetic-axis major radius [t]

getMagZ(*length_unit=1*)
returns magnetic-axis Z [t]

getAreaLCFS(*length_unit=2*)
returns LCFS cross-sectional area [t]

getAOut(*length_unit=1*)
returns outboard-midplane minor radius at LCFS [t]

getRmidOut(*length_unit=1*)
returns outboard-midplane major radius [t]

getGeometry(*length_unit=None*)
pulls dimensional geometry parameters

Returns namedtuple containing {magnetic-axis R,Z, LCFS area, outboard-midplane LCFS a,R}

getQProfile()
returns safety factor q [psi,t]

getQ0()
returns q on magnetic axis [t]

getQ95()
returns q at 95% flux surface [t]

getQLCFS()
returns q on LCFS [t]

getQ1Surf(*length_unit=1*)
returns outboard-midplane minor radius of q=1 surface [t]

getQ2Surf(*length_unit=1*)
returns outboard-midplane minor radius of q=2 surface [t]

getQ3Surf(*length_unit=1*)
returns outboard-midplane minor radius of q=3 surface [t]

getQs(*length_unit=1*)
pulls q values

Returns namedtuple containing {q0,q95,qLCFS,rq1,rq2,rq3}

getBtVac()
returns on-axis vacuum toroidal field [t]

getBtPla()
returns on-axis plasma toroidal field [t]

getBpAvg()
returns average poloidal field [t]

getFields()
pulls vacuum and plasma toroidal field, avg poloidal field

Returns namedtuple containing {btaxv,btaxp,bpolav}

getIpCalc()
returns EFIT-calculated plasma current [t]

getIpMeas()
returns magnetics-measured plasma current [t]

getJp()
returns EFIT-calculated plasma current density Jp on flux grid [t,r,z]

getBetaT()
returns EFIT-calculated toroidal beta [t]

getBetaP()
returns EFIT-calculated poloidal beta [t]

getLi()
returns EFIT-calculated internal inductance [t]

getBetas()
pulls calculated betap, betat, internal inductance

Returns namedtuple containing {betat,betap,Li}

getDiamagFlux()
returns measured diamagnetic-loop flux [t]

getDiamagBetaT()
returns diamagnetic-loop toroidal beta [t]

getDiamagBetaP()
returns diamagnetic-loop avg poloidal beta [t]

getDiamagTauE()
returns diamagnetic-loop energy confinement time [t]

getDiamagWp()
returns diamagnetic-loop plasma stored energy [t]

getDiamag()
pulls diamagnetic flux measurements, toroidal and poloidal beta, energy confinement time and stored energy

Returns namedtuple containing {diamag. flux, betatd, betapd, tauDiamag, WDiamag}

getWMHD()
returns EFIT-calculated MHD stored energy [t]

getTauMHD()
returns EFIT-calculated MHD energy confinement time [t]

getPinj()
returns EFIT-calculated injected power [t]

getWbdot()
returns EFIT-calculated d/dt of magnetic stored energy [t]

getWpdot()
returns EFIT-calculated d/dt of plasma stored energy [t]

getEnergy()
pulls EFIT-calculated energy parameters - stored energy, tau_E, injected power, d/dt of magnetic and plasma stored energy

Returns namedtuple containing {WMHD,tauMHD,Pinj,Wbdot,Wpdot}

getMachineCrossSection()
Returns R,Z coordinates of vacuum-vessel wall for masking, plotting routines.

Returns The requested data.

getMachineCrossSectionFull()
Returns R,Z coordinates of vacuum-vessel wall for plotting routines.

Absent additional vector-graphic data on machine cross-section, returns self.getMachineCrossSection().

Returns The requested data.

getCurrentSign()
Returns the sign of the current, based on the check in Steve Wolfe's IDL implementation efit_rz2psi.pro.

getParam(*path*)
backup function - path to parameter as input, returns desired variable acts as wrapper for MDS call

Parameters **path** – str The path to the MDSplus node you wish to pull in.

Returns The requested data.

2.1.4 `FromArrays` Module

class `eqtools.FromArrays.`**`ArrayEquilibrium`**(*psiRZ*, *rGrid*, *zGrid*, *time*, *q*, *fluxVol*, *length_unit='m'*, *tspline=False*, *fast=False*)

Bases: `eqtools.core.Equilibrium`

Class to represent an equilibrium specified as arrays of data.

Create ArrayEquilibrium instance from arrays of data.

Parameters

- **psiRZ** – Array-like, (M, N, P). Flux values at M times, N Z locations and P R locations.
- **rGrid** – Array-like, (P,). R coordinates that psiRZ is given at.
- **zGrid** – Array-like, (N,). Z coordinates that psiRZ is given at.
- **time** – Array-like, (M,). Times that psiRZ is given at.
- **q** – Array-like, (Q, M). q profile evaluated at Q values of psinorm from 0 to 1, given at M times.
- **fluxVol** – Array-like, (S, M). Flux surface volumes evaluated at S values of psinorm from 0 to 1, given at M times.

Keyword Arguments **length_unit** – String. Base unit for any quantity whose dimensions are length to any power. Default is ‘m’. Valid options are:

‘m’	meters
‘cm’	centimeters
‘mm’	millimeters
‘in’	inches
‘ft’	feet
‘yd’	yards
‘smoot’	smoots
‘cubit’	cubits
‘hand’	hands
‘default’	meters

`getTimeBase`()
Returns a copy of the time base vector, array dimensions are (M,).

`getFluxGrid`()
Returns a copy of the flux array, dimensions are (M, N, P), corresponding to (time, Z, R).

`getRGrid`(*length_unit=1*)
Returns a copy of the radial grid, dimensions are (P,).

`getZGrid`(*length_unit=1*)
Returns a copy of the vertical grid, dimensions are (N,).

`getQProfile`()
Returns safety factor q profile (over Q values of psinorm from 0 to 1), dimensions are (Q, M)

2.1.5 `NSTXEFIT` Module

This module provides classes for working with NSTX EFIT data.

class `eqtools.NSTXEFIT.`**`NSTXEFITTree`**(*shot*, *tree='EFIT01'*, *length_unit='m'*, *gfile='geqdsk'*, *afile='aeqdsk'*, *tspline=False*, *monotonic=False*)

Bases: `eqtools.EFIT.EFITTree`

Inherits `EFITTree` class. Machine-specific data handling class for the National Spherical Torus Experiment (NSTX). Pulls EFIT data from selected MDS tree and shot, stores as object attributes. Each EFIT variable or set of variables is recovered with a corresponding getter method. Essential data for EFIT mapping are pulled on initialization (e.g. psirz grid). Additional data are pulled at the first request and stored for subsequent usage.

Intializes NSTX version of EFITTree object. Pulls data from MDS tree for storage in instance attributes. Core attributes are populated from the MDS tree on initialization. Additional attributes are initialized as None, filled on the first request to the object.

Parameters **shot** – (long) int NSTX shot index (long)

Keyword Arguments

- **tree** – str Optional input for EFIT tree, defaults to 'EFIT01' (i.e., EFIT data are under EFIT01::top.results).
- **length_unit** – str Sets the base unit used for any quantity whose dimensions are length to any power. Valid options are:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	whatever the default in the tree is (no conversion is performed, units may be inconsistent)

  Default is 'm' (all units taken and returned in meters).
- **gfile** – str Optional input for EFIT geqdsk location name, defaults to 'g_eqdsk' (i.e., EFIT data are under tree::top.results.G_EQDSK)
- **afile** – str Optional input for EFIT aeqdsk location name, defaults to 'a_eqdsk' (i.e., EFIT data are under tree::top.results.A_EQDSK)
- **tspline** – Boolean Sets whether or not interpolation in time is performed using a tricubic spline or nearest-neighbor interpolation. Tricubic spline interpolation requires at least four complete equilibria at different times. It is also assumed that they are functionally correlated, and that parameters do not vary out of their boundaries (derivative = 0 boundary condition). Default is False (use nearest neighbor interpolation).
- **monotonic** – Boolean Sets whether or not the "monotonic" form of time window finding is used. If True, the timebase must be monotonically increasing. Default is False (use slower, safer method).

`getFluxGrid`()

returns EFIT flux grid, [t,z,r]

getFluxVol()
: Not implemented in NSTXEFIT tree.

  Returns volume within flux surface [psi,t]

getRmidPsi(*length_unit=1*)
: returns maximum major radius of each flux surface [t,psi]

getVolLCFS(*length_unit=3*)
: returns volume within LCFS [t]

rz2volnorm(**args*, ***kwargs*)
: Calculated normalized volume of flux surfaces not stored in NSTX EFIT. All maping with Volnorm not implemented

psinorm2volnorm(**args*, ***kwargs*)
: Calculated normalized volume of flux surfaces not stored in NSTX EFIT. All maping with Volnorm not implemented

class eqtools.NSTXEFIT.**NSTXEFITTreeProp**(*shot*, *tree='EFIT01'*, *length_unit='m'*, *gfile='geqdsk'*, *afile='aeqdsk'*, *tspline=False*, *monotonic=False*)
: Bases: `eqtools.NSTXEFIT.NSTXEFITTree`, `eqtools.core.PropertyAccessMixin`

  NSTXEFITTree with the PropertyAccessMixin added to enable property-style access. This is good for interactive use, but may drag the performance down.

2.1.6 `afilereader` Module

This module contains the AFileReader class, a lightweight data handler for a-file (time-history) datasets.

Classes:

AFileReader: Data-storage class for a-file data. Reads data from ASCII a-file, storing as copy-safe object attributes.

class eqtools.afilereader.**AFileReader**(*afile*)
: Bases: `object`

  Class to read ASCII a-file (time-history data storage) into lightweight, user-friendly data structure.

  A-files store data blocks of scalar time-history data for EFIT plasma equilibrium. Each parameter is read into a pseudo-private object attribute (marked by a leading underscore), followed by the standard EFIT variable names.

  initialize object, reading from file.

  Parameters **afile** – str path to a-file

__str__()
: overrides default __str__method with more useful output.

__getattribute__(*name*)
: Copy-safe attribute retrieval method overriding default object.__getattribute__.

  Tries to retrieve attribute as-written (first check for default object attributes). If that fails, looks for pseudo-private attributes, marked by preceding underscore, to retrieve data values. If this fails, raise AttributeError.

  Parameters **name** – String. Name (without leading underscore for data variables) of attribute.

  Raises `AttributeError` – if no attribute can be found.

__setattr__(*name*, *value*)
: Copy-safe attribute setting method overriding default object.__setattr__.

Raises error if object already has attribute _{name} for input name, as such an attribute would interfere with automatic property generation in __getattribute__.

Parameters **name** – String. Attribute name.

Raises `AttributeError` – if attempting to create attribute with protected pseudo-private name.

2.1.7 `core` Module

This module provides the core classes for eqtools, including the base Equilibrium class.

exception `eqtools.core.`**`ModuleWarning`**

Bases: `exceptions.Warning`

Warning class to notify the user of unavailable modules.

class `eqtools.core.`**`PropertyAccessMixin`**

Bases: `object`

Mixin to implement access of getter methods through a property-type interface without the need to apply a decorator to every property.

For any getter obj.getSomething(), the call obj.Something will work.

This is accomplished by overriding __getattribute__ such that if an attribute ATTR does not exist it then attempts to call self.getATTR(). If self.getATTR() does not exist, an AttributeError will be raised as usual.

Also overrides __setattr__ such that it will raise an AttributeError when attempting to write an attribute ATTR for which there is already a method getATTR.

`__getattribute__`(*name*)

Get an attribute.

Tries to get attribute as-written. If this fails, tries to call the method get<name> with no arguments. If this fails, raises AttributeError. This effectively generates a Python 'property' for each getter method.

Parameters **name** – String. Name of the attribute to retrieve. If the instance has an attribute with this name, the attribute is returned. If the instance does not have an attribute with this name but does have a method called 'get'+name, this method is called and the result is returned.

Returns The value of the attribute requested.

Raises `AttributeError` – If neither attribute name or method 'get'+name exist.

`__setattr__`(*name*, *value*)

Set an attribute.

Raises AttributeError if the object already has a method get[name], as creation of such an attribute would interfere with the automatic property generation in __getattribute__.

Parameters

- **name** – String. Name of the attribute to set.
- **value** – Object. Value to set the attribute to.

Raises `AttributeError` – If a method called 'get'+name already exists.

`eqtools.core.`**`inPolygon`**(*polyx*, *polyy*, *pointx*, *pointy*)

Function calculating whether a given point is within a 2D polygon.

Given an array of X,Y coordinates describing a 2D polygon, checks whether a point given by x,y coordinates lies within the polygon. Operates via a ray-casting approach - the function projects a semi-infinite ray parallel to the positive horizontal axis, and counts how many edges of the polygon this ray intersects. For a simply-connected polygon, this determines whether the point is inside (even number of crossings) or outside (odd number of crossings) the polygon, by the Jordan Curve Theorem.

Parameters

- **polyx** – Array-like. Array of x-coordinates of the vertices of the polygon.
- **polyy** – Array-like. Array of y-coordinates of the vertices of the polygon.
- **pointx** – Int or float. x-coordinate of test point.
- **pointy** – Int or float. y-coordinate of test point.

Returns

result – Boolean.

True/False result for whether the point is contained within the polygon.

class `eqtools.core.`**`Equilibrium`**(*length_unit='m'*, *tspline=False*, *monotonic=False*, *verbose=True*)

Bases: `object`

Abstract class of data handling object for magnetic reconstruction outputs.

Defines the mapping routines and method fingerprints necessary. Each variable or set of variables is recovered with a corresponding getter method. Essential data for mapping are pulled on initialization (psirz grid, for example) to frontload timing overhead. Additional data are pulled at the first request and stored for subsequent usage.

Note: This abstract class should not be used directly. Device- and code- specific subclasses are set up to account for inter-device/-code differences in data storage.

Create a new Equilibrium instance.

Keyword Arguments

- **length_unit** – String. Sets the base unit used for any quantity whose dimensions are length to any power. Valid options are:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	whatever the default in the tree is (no conversion is performed, units may be inconsistent)

  Default is 'm' (all units taken and returned in meters).

- **tspline** – Boolean. Sets whether or not interpolation in time is performed using a tricubic spline or nearest-neighbor interpolation. Tricubic spline interpolation requires at least four complete equilibria at different times. It is also assumed that they are functionally correlated, and that parameters do not vary out of their boundaries (derivative = 0 boundary condition). Default is False (use nearest neighbor interpolation).

- **monotonic** – Boolean. Sets whether or not the "monotonic" form of time window finding is used. If True, the timebase must be monotonically increasing. Default is False (use slower, safer method).
- **verbose** – Boolean. Allows or blocks console readout during operation. Defaults to True, displaying useful information for the user. Set to False for quiet usage or to avoid console clutter for multiple instances.

Raises

- `ValueError` – If length_unit is not a valid unit specifier.
- `ValueError` – If tspline is True by module trispline did not load successfully.

`__str__`()

String representation of this instance.

Returns String describing this object.

`rz2psi`(*R*, *Z*, *t*, *return_t=False*, *make_grid=False*, *each_t=True*, *length_unit=1*)

Converts the passed R, Z, t arrays to psi values.

If tspline is False for this Equilibrium instance, uses scipy.interpolate.RectBivariateSpline to interpolate in terms of R and Z. Finds the nearest time slices to those given: nearest-neighbor interpolation in time. Otherwise, uses the tricubic package to perform a trivariate interpolation in space and time.

Parameters

- **R** – Array-like or scalar float. Values of the radial coordinate to map to poloidal flux. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as Z unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to poloidal flux. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).
- **t** – Array-like or single value. If t is a single value, it is used for all of the elements of R, Z. If t is array-like and the make_grid keyword is False, t must have the same dimensions as R and Z. If t is array-like and the make_grid keyword is True, t must have shape (len(Z), len(R)).

Keyword Arguments

- **return_t** – Boolean. Set to True to return a tuple of (psi, time_idxs), where time_idxs is the array of time indices actually used in evaluating psi with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return psi).
- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. When using this option, it is highly recommended to only pass a scalar value for t (such that each point in the flux grid is evaluated at this same value t). Otherwise, t must have the same shape as the resulting meshgrid, and each element in the returned psi array will be at the corresponding time in the t array. Default is False (do not form meshgrid).
- **each_t** – Boolean. When True, the elements in *R* and *Z* (or the meshgrid thereof if *make_grid* is True) are evaluated at each value in *t*. If True, *t* must have only one dimension (or be a scalar). If False, *t* must match the shape of *R* and *Z* (or their meshgrid if *make_grid* is True) or be a scalar. Default is True (evaluate ALL *R*, *Z* at each element in *t*).

- **length_unit** – String or 1. Length unit that R and Z are being given in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

  If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters).

Returns

psi or (psi, time_idxs)

- **psi** - Array or scalar float. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then psi has this shape as well. If the make_grid keyword was True then psi has shape (len(Z), len(R)).
- **time_idxs** - Array with same shape as psi. The indices (in the timebase as returned by `getTimeBase()`) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single psi value at R=0.6m, Z=0.0m, t=0.26s:

```
psi_val = Eq_instance.rz2psi(0.6, 0, 0.26)
```

Find psi values at (R, Z) points (0.6m, 0m) and (0.8m, 0m) at the single time t=0.26s. Note that the Z vector must be fully specified, even if the values are all the same:

```
psi_arr = Eq_instance.rz2psi([0.6, 0.8], [0, 0], 0.26)
```

Find psi values at (R, Z) points (0.6m, 0m) at times t=[0.2s, 0.3s]:

```
psi_arr = Eq_instance.rz2psi(0.6, 0, [0.2, 0.3])
```

Find psi values at (R, Z, t) points (0.6m, 0m, 0.2s) and (0.5m, 0.2m, 0.3s):

```
psi_arr = Eq_instance.rz2psi([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)
```

Find psi values on grid defined by 1D vector of radial positions R and 1D vector of vertical positions Z at time t=0.2s:

```
psi_mat = Eq_instance.rz2psi(R, Z, 0.2, make_grid=True)
```

rz2psinorm(*R*, *Z*, *t*, *return_t=False*, *sqrt=False*, *make_grid=False*, *each_t=True*, *length_unit=1*)
Calculates the normalized poloidal flux at the given (R, Z, t).

Uses the definition:

$$\texttt{psi\_norm} = \frac{\psi - \psi(0)}{\psi(a) - \psi(0)}$$

If tspline is False for this Equilibrium instance, uses scipy.interpolate.RectBivariateSpline to interpolate in terms of R and Z. Finds the nearest time slices to those given: nearest-neighbor interpolation in time. Otherwise, uses the tricubic package to perform a trivariate interpolation in space and time.

Parameters

- **R** – Array-like or scalar float. Values of the radial coordinate to map to normalized poloidal flux. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as Z unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to normalized poloidal flux. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).
- **t** – Array-like or single value. If t is a single value, it is used for all of the elements of R, Z. If t is array-like and the make_grid keyword is False, t must have the same dimensions as R and Z. If t is array-like and the make_grid keyword is True, t must have shape (len(Z), len(R)).

Keyword Arguments

- **return_t** – Boolean. Set to True to return a tuple of (psinorm, time_idxs), where time_idxs is the array of time indices actually used in evaluating psi with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return psinorm).
- **sqrt** – Boolean. Set to True to return the square root of normalized flux. Only the square root of positive psi_norm values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return psinorm).
- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. When using this option, it is highly recommended to only pass a scalar value for t (such that each point in the flux grid is evaluated at this same value t). Otherwise, t must have the same shape as the resulting meshgrid, and each element in the returned psi array will be at the corresponding time in the t array. Default is False (do not form meshgrid).
- **each_t** – Boolean. When True, the elements in *R* and *Z* (or the meshgrid thereof if *make_grid* is True) are evaluated at each value in *t*. If True, *t* must have only one dimension (or be a scalar). If False, *t* must match the shape of *R* and *Z* (or their meshgrid if *make_grid* is True) or be a scalar. Default is True (evaluate ALL *R*, *Z* at each element in *t*).
- **length_unit** – String or 1. Length unit that R and Z are being given in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters).

Returns

psinorm or (psinorm, time_idxs)

- **psinorm** - Array or scalar float. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then psinorm has this shape as well. If the make_grid keyword was True then psinorm has shape (len(Z), len(R)).
- **time_idxs** - Array with same shape as psinorm. The indices (in the timebase returned by `getTimeBase()`) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single psinorm value at R=0.6m, Z=0.0m, t=0.26s:

```
psi_val = Eq_instance.rz2psinorm(0.6, 0, 0.26)
```

Find psinorm values at (R, Z) points (0.6m, 0m) and (0.8m, 0m) at the single time t=0.26s. Note that the Z vector must be fully specified, even if the values are all the same:

```
psi_arr = Eq_instance.rz2psinorm([0.6, 0.8], [0, 0], 0.26)
```

Find psinorm values at (R, Z) points (0.6m, 0m) at times t=[0.2s, 0.3s]:

```
psi_arr = Eq_instance.rz2psinorm(0.6, 0, [0.2, 0.3])
```

Find psinorm values at (R, Z, t) points (0.6m, 0m, 0.2s) and (0.5m, 0.2m, 0.3s):

```
psi_arr = Eq_instance.rz2psinorm([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)
```

Find psinorm values on grid defined by 1D vector of radial positions R and 1D vector of vertical positions Z at time t=0.2s:

```
psi_mat = Eq_instance.rz2psinorm(R, Z, 0.2, make_grid=True)
```

rz2phinorm(**args*, ***kwargs*)
Calculates the normalized toroidal flux.

Uses the definitions:

$$\texttt{phi} = \int q(\psi)\,d\psi$$

$$\texttt{phi\_norm} = \frac{\phi}{\phi(a)}$$

This is based on the IDL version efit_rz2rho.pro by Steve Wolfe.

If tspline is False for this Equilibrium instance, uses scipy.interpolate.RectBivariateSpline to interpolate in terms of R and Z. Finds the nearest time slices to those given: nearest-neighbor interpolation in time. Otherwise, uses the tricubic package to perform a trivariate interpolation in space and time.

Parameters

- **R** – Array-like or scalar float. Values of the radial coordinate to map to normalized toroidal flux. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as Z unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to normalized toroidal flux. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).
- **t** – Array-like or single value. If t is a single value, it is used for all of the elements of R, Z. If t is array-like and the make_grid keyword is False, t must have the same dimensions as R and Z. If t is array-like and the make_grid keyword is True, t must have shape (len(Z), len(R)).

Keyword Arguments

- **return_t** – Boolean. Set to True to return a tuple of (phinorm, time_idxs), where time_idxs is the array of time indices actually used in evaluating phinorm with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return phinorm).
- **sqrt** – Boolean. Set to True to return the square root of normalized flux. Only the square root of positive phi_norm values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return phinorm).
- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. When using this option, it is highly recommended to only pass a scalar value for t (such that each point in the flux grid is evaluated at this same value t). Otherwise, t must have the same shape as the resulting meshgrid, and each element in the returned psi array will be at the corresponding time in the t array. Default is False (do not form meshgrid).
- **each_t** – Boolean. When True, the elements in *R* and *Z* (or the meshgrid thereof if *make_grid* is True) are evaluated at each value in *t*. If True, *t* must have only one dimension (or be a scalar). If False, *t* must match the shape of *R* and *Z* (or their meshgrid if *make_grid* is True) or be a scalar. Default is True (evaluate ALL *R*, *Z* at each element in *t*).
- **rho** – Boolean. For phinorm, this should always be set to False, the default value.
- **kind** – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to phinorm. This is passed to scipy.interpolate.interp1d. Valid options are: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more

details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.

- **length_unit** – String or 1. Length unit that R and Z are being given in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters).

Returns

phinorm or (phinorm, time_idxs)

- **phinorm** - Array or scalar float. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then phinorm has this shape as well. If the make_grid keyword was True then phinorm has shape (len(Z), len(R)).
- **time_idxs** - Array with same shape as phinorm. The indices (in the timebase returned by `getTimeBase()`) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single phinorm value at R=0.6m, Z=0.0m, t=0.26s:

```
phi_val = Eq_instance.rz2phinorm(0.6, 0, 0.26)
```

Find phinorm values at (R, Z) points (0.6m, 0m) and (0.8m, 0m) at the single time t=0.26s. Note that the Z vector must be fully specified, even if the values are all the same:

```
phi_arr = Eq_instance.rz2phinorm([0.6, 0.8], [0, 0], 0.26)
```

Find phinorm values at (R, Z) points (0.6m, 0m) at times t=[0.2s, 0.3s]:

```
phi_arr = Eq_instance.rz2phinorm(0.6, 0, [0.2, 0.3])
```

Find phinorm values at (R, Z, t) points (0.6m, 0m, 0.2s) and (0.5m, 0.2m, 0.3s):

```
phi_arr = Eq_instance.rz2phinorm([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)
```

Find phinorm values on grid defined by 1D vector of radial positions R and 1D vector of vertical positions Z at time t=0.2s:

```
phi_mat = Eq_instance.rz2phinorm(R, Z, 0.2, make_grid=True)
```

rz2volnorm(**args*, ***kwargs*)

Calculates the normalized flux surface volume.

Based on the IDL version efit_rz2rho.pro by Steve Wolfe.

If tspline is False for this Equilibrium instance, uses scipy.interpolate.RectBivariateSpline to interpolate in terms of R and Z. Finds the nearest time slices to those given: nearest-neighbor interpolation in time. Otherwise, uses the tricubic package to perform a trivariate interpolation in space and time.

Parameters

- **R** – Array-like or scalar float. Values of the radial coordinate to map to normalized volume. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as Z unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to normalized volume. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).
- **t** – Array-like or single value. If t is a single value, it is used for all of the elements of R, Z. If t is array-like and the make_grid keyword is False, t must have the same dimensions as R and Z. If t is array-like and the make_grid keyword is True, t must have shape (len(Z), len(R)).

Keyword Arguments

- **return_t** – Boolean. Set to True to return a tuple of (volnorm, time_idxs), where time_idxs is the array of time indices actually used in evaluating volnorm with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return volnorm).
- **sqrt** – Boolean. Set to True to return the square root of normalized volume. Only the square root of positive volnorm values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return volnorm).
- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. When using this option, it is highly recommended to only pass a scalar value for t (such that each point in the flux grid is evaluated at this same value t). Otherwise, t must have the same shape as the resulting meshgrid, and each element in the returned psi array will be at the corresponding time in the t array. Default is False (do not form meshgrid).
- **each_t** – Boolean. When True, the elements in *R* and *Z* (or the meshgrid thereof if *make_grid* is True) are evaluated at each value in *t*. If True, *t* must have only one dimension (or be a scalar). If False, *t* must match the shape of *R* and *Z* (or their meshgrid if *make_grid* is True) or be a scalar. Default is True (evaluate ALL *R*, *Z* at each element in *t*).
- **rho** – Boolean. For volnorm, this should always be set to False, the default value.
- **kind** – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to volnorm. This is passed to scipy.interpolate.interp1d. Valid options are: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy,

this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.

- **length_unit** – String or 1. Length unit that R and Z are being given in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

  If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters).

Returns

volnorm or (volnorm, time_idxs)

- **volnorm** - Array or scalar float. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then volnorm has this shape as well. If the make_grid keyword was True then volnorm has shape (len(Z), len(R)).
- **time_idxs** - Array with same shape as volnorm. The indices (in self.getTimeBase()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single volnorm value at R=0.6m, Z=0.0m, t=0.26s:

```
psi_val = Eq_instance.rz2volnorm(0.6, 0, 0.26)
```

Find volnorm values at (R, Z) points (0.6m, 0m) and (0.8m, 0m) at the single time t=0.26s. Note that the Z vector must be fully specified, even if the values are all the same:

```
vol_arr = Eq_instance.rz2volnorm([0.6, 0.8], [0, 0], 0.26)
```

Find volnorm values at (R, Z) points (0.6m, 0m) at times t=[0.2s, 0.3s]:

```
vol_arr = Eq_instance.rz2volnorm(0.6, 0, [0.2, 0.3])
```

Find volnorm values at (R, Z, t) points (0.6m, 0m, 0.2s) and (0.5m, 0.2m, 0.3s):

```
vol_arr = Eq_instance.rz2volnorm([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)
```

Find volnorm values on grid defined by 1D vector of radial positions R and 1D vector of vertical positions Z at time t=0.2s:

```
vol_mat = Eq_instance.rz2volnorm(R, Z, 0.2, make_grid=True)
```

rz2rho(*method*, **args*, ***kwargs*)

Convert the passed (R, Z, t) coordinates into one of several normalized coordinates.

If tspline is False for this Equilibrium instance, uses scipy.interpolate.RectBivariateSpline to interpolate in terms of R and Z. Finds the nearest time slices to those given: nearest-neighbor interpolation in time. Otherwise, uses the tricubic package to perform a trivariate interpolation in space and time.

Parameters

- **method** – String. Indicates which normalized coordinates to use. Valid options are:

psinorm	Normalized poloidal flux
phinorm	Normalized toroidal flux
volnorm	Normalized volume

- **R** – Array-like or scalar float. Values of the radial coordinate to map to normalized coordinate. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as Z unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to normalized coordinate. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).
- **t** – Array-like or single value. If t is a single value, it is used for all of the elements of R, Z. If t is array-like and the make_grid keyword is False, t must have the same dimensions as R and Z. If t is array-like and the make_grid keyword is True, t must have shape (len(Z), len(R)).

Keyword Arguments

- **return_t** – Boolean. Set to True to return a tuple of (volnorm, time_idxs), where time_idxs is the array of time indices actually used in evaluating volnorm with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return volnorm).
- **sqrt** – Boolean. Set to True to return the square root of normalized coordinate. Only the square root of positive values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return normalized coordinate itself).
- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. When using this option, it is highly recommended to only pass a scalar value for t (such that each point in the flux grid is evaluated at this same value t). Otherwise, t must have the same shape as the resulting meshgrid, and each element in the returned psi array will be at the corresponding time in the t array. Default is False (do not form meshgrid).
- **each_t** – Boolean. When True, the elements in *R* and *Z* (or the meshgrid thereof if *make_grid* is True) are evaluated at each value in *t*. If True, *t* must have only one dimension (or be a scalar). If False, *t* must match the shape of *R* and *Z* (or their meshgrid if *make_grid* is True) or be a scalar. Default is True (evaluate ALL *R*, *Z* at each element in *t*).
- **rho** (*phinorm and volnorm only*) – Boolean. For phinorm and volnorm, this should always be set to False, the default value.
- **kind** (*phinorm and volnorm only*) – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to phinorm or volnorm. This is passed to scipy.interpolate.interp1d. Valid options are: 'linear', 'nearest', 'zero', 'slinear',

'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.

- **length_unit** – String or 1. Length unit that R and Z are being given in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters).

Returns

rho or (rho, time_idxs)

- **rho** - Array or scalar float. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then rho has this shape as well. If the make_grid keyword was True then rho has shape (len(Z), len(R)).
- **time_idxs** - Array with same shape as rho. The indices (in self.getTimeBase()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

Raises `ValueError` – If method is not one of the supported values.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single psinorm value at R=0.6m, Z=0.0m, t=0.26s:

```
psi_val = Eq_instance.rz2rho('psinorm', 0.6, 0, 0.26)
```

Find psinorm values at (R, Z) points (0.6m, 0m) and (0.8m, 0m) at the single time t=0.26s. Note that the Z vector must be fully specified, even if the values are all the same:

```
psi_arr = Eq_instance.rz2rho('psinorm', [0.6, 0.8], [0, 0], 0.26)
```

Find psinorm values at (R, Z) points (0.6m, 0m) at times t=[0.2s, 0.3s]:

```
psi_arr = Eq_instance.rz2rho('psinorm', 0.6, 0, [0.2, 0.3])
```

Find psinorm values at (R, Z, t) points (0.6m, 0m, 0.2s) and (0.5m, 0.2m, 0.3s):

```
psi_arr = Eq_instance.rz2rho('psinorm', [0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)
```

Find psinorm values on grid defined by 1D vector of radial positions R and 1D vector of vertical positions Z at time t=0.2s:

```
psi_mat = Eq_instance.rz2rho('psinorm', R, Z, 0.2, make_grid=True)
```

rz2rmid(**args*, ***kwargs*)

Maps the given points to the outboard midplane major radius, R_mid.

Based on the IDL version efit_rz2rmid.pro by Steve Wolfe.

If tspline is False for this Equilibrium instance, uses scipy.interpolate.RectBivariateSpline to interpolate in terms of R and Z. Finds the nearest time slices to those given: nearest-neighbor interpolation in time. Otherwise, uses the tricubic package to perform a trivariate interpolation in space and time.

Parameters

- **R** – Array-like or scalar float. Values of the radial coordinate to map to midplane radius. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as Z unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to midplane radius. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).
- **t** – Array-like or single value. If t is a single value, it is used for all of the elements of R, Z. If t is array-like and the make_grid keyword is False, t must have the same dimensions as R and Z. If t is array-like and the make_grid keyword is True, t must have shape (len(Z), len(R)).

Keyword Arguments

- **return_t** – Boolean. Set to True to return a tuple of (R_mid, time_idxs), where time_idxs is the array of time indices actually used in evaluating R_mid with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return R_mid).
- **sqrt** – Boolean. Set to True to return the square root of midplane radius. Only the square root of positive values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return R_mid itself).
- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. When using this option, it is highly recommended to only pass a scalar value for t (such that each point in the flux grid is evaluated at this same value t). Otherwise, t must have the same shape as the resulting meshgrid, and each element in the returned psi array will be at the corresponding time in the t array. Default is False (do not form meshgrid).
- **each_t** – Boolean. When True, the elements in *R* and *Z* (or the meshgrid thereof if *make_grid* is True) are evaluated at each value in *t*. If True, *t* must have only one dimension (or be a scalar). If False, *t* must match the shape of *R* and *Z* (or their meshgrid if *make_grid* is True) or be a scalar. Default is True (evaluate ALL *R*, *Z* at each element in *t*).
- **rho** – Boolean. Set to True to return r/a (normalized minor radius) instead of R_mid. Default is False (return major radius, R_mid).
- **kind** – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to R_mid. This is passed to scipy.interpolate.interp1d. Valid options are: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can

cause problems, in which case you should try 'linear' until you can rebuild your scipy install.

- **length_unit** – String or 1. Length unit that R and Z are being given in AND that R_mid is returned in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters, R_mid returned in meters).

Returns

R_mid or (R_mid, time_idxs)

- **R_mid** - Array or scalar float. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then R_mid has this shape as well. If the make_grid keyword was True then R_mid has shape (len(Z), len(R)).
- **time_idxs** - Array with same shape as R_mid. The indices (in the timebase returned by `getTimeBase()`) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single R_mid value at R=0.6m, Z=0.0m, t=0.26s:

```
R_mid_val = Eq_instance.rz2rmid(0.6, 0, 0.26)
```

Find R_mid values at (R, Z) points (0.6m, 0m) and (0.8m, 0m) at the single time t=0.26s. Note that the Z vector must be fully specified, even if the values are all the same:

```
R_mid_arr = Eq_instance.rz2rmid([0.6, 0.8], [0, 0], 0.26)
```

Find R_mid values at (R, Z) points (0.6m, 0m) at times t=[0.2s, 0.3s]:

```
R_mid_arr = Eq_instance.rz2rmid(0.6, 0, [0.2, 0.3])
```

Find R_mid values at (R, Z, t) points (0.6m, 0m, 0.2s) and (0.5m, 0.2m, 0.3s):

```
R_mid_arr = Eq_instance.rz2rmid([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)
```

Find R_mid values on grid defined by 1D vector of radial positions R and 1D vector of vertical positions Z at time t=0.2s:

```
R_mid_mat = Eq_instance.rz2rmid(R, Z, 0.2, make_grid=True)
```

psinorm2rmid(*psi_norm*, *t*, ***kwargs*)

Calculates the outboard R_mid location corresponding to the passed psi_norm (normalized poloidal flux) values.

If tspline is False for this Equilibrium instance, uses scipy.interpolate.RectBivariateSpline to interpolate in terms of R and Z. Finds the nearest time slices to those given: nearest-neighbor interpolation in time. Otherwise, uses the tricubic package to perform a trivariate interpolation in space and time.

Parameters

- **psi_norm** – Array-like or scalar float. Values of the normalized poloidal flux to map to midplane radius. If psi_norm is a scalar, it is used as the value for all of the values in t.
- **t** – Array-like or single value. If t is a single value, it is used for all of the elements of psi_norm. If neither t nor psi_norm are scalars, t must have the same shape as psi_norm.

Keyword Arguments

- **each_t** – Boolean. When True, the elements in *R* and *Z* (or the meshgrid thereof if *make_grid* is True) are evaluated at each value in *t*. If True, *t* must have only one dimension (or be a scalar). If False, *t* must match the shape of *R* and *Z* (or their meshgrid if *make_grid* is True) or be a scalar. Default is True (evaluate ALL *R*, *Z* at each element in *t*).
- **return_t** – Boolean. Set to True to return a tuple of (R_mid, time_idxs), where time_idxs is the array of time indices actually used in evaluating R_mid with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return R_mid).
- **sqrt** – Boolean. Set to True to return the square root of the quantity obtained (R_mid or r/a). Only the square root of positive values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return Quan itself).
- **rho** – Boolean. Set to True to return r/a (normalized minor radius) instead of R_mid. Default is False (return major radius, R_mid).
- **kind** – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to R_mid. This is passed to scipy.interpolate.interp1d. Valid options are: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.
- **length_unit** – String or 1. Length unit that R_mid is returned in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

If length_unit is 1 or None, meters are assumed. The default value is 1 (R_mid returned in meters).

Returns

R_mid or (R_mid, time_idxs)

- **R_mid** - Array or scalar float. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. R_mid will have the same shape as t and psi_norm (or whichever one is Array-like).
- **time_idxs** - Array with same shape as R_mid. The indices (in the timebase returned by `getTimeBase()`) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single R_mid value for psinorm=0.7, t=0.26s:

```
R_mid_val = Eq_instance.psinorm2rmid(0.7, 0.26)
```

Find R_mid values at psi_norm values of 0.5 and 0.7 at the single time t=0.26s. Note that the Z vector must be fully specified, even if the values are all the same:

```
R_mid_arr = Eq_instance.psinorm2rmid([0.5, 0.7], 0.26)
```

Find R_mid values at psi_norm=0.5 at times t=[0.2s, 0.3s]:

```
R_mid_arr = Eq_instance.psinorm2rmid(0.5, [0.2, 0.3])
```

Find R_mid values at (psinorm, t) points (0.6, 0.2s) and (0.5, 0.3s):

```
R_mid_arr = Eq_instance.psinorm2rmid([0.6, 0.5], [0.2, 0.3], each_t=False)
```

psinorm2volnorm(*psi_norm*, *t*, ***kwargs*)

Calculates the normalized volume corresponding to the passed psi_norm (normalized poloidal flux) values.

If tspline is False for this Equilibrium instance, uses scipy.interpolate.RectBivariateSpline to interpolate in terms of R and Z. Finds the nearest time slices to those given: nearest-neighbor interpolation in time. Otherwise, uses the tricubic package to perform a trivariate interpolation in space and time.

Parameters

- **psi_norm** – Array-like or scalar float. Values of the normalized poloidal flux to map to normalized volume. If psi_norm is a scalar, it is used as the value for all of the values in t.

- **t** – Array-like or single value. If t is a single value, it is used for all of the elements of psi_norm. If neither t nor psi_norm are scalars, t must have the same shape as psi_norm.

Keyword Arguments

- **each_t** – Boolean. When True, the elements in *R* and *Z* (or the meshgrid thereof if *make_grid* is True) are evaluated at each value in *t*. If True, *t* must have only one dimension (or be a scalar). If False, *t* must match the shape of *R* and *Z* (or their meshgrid if *make_grid* is True) or be a scalar. Default is True (evaluate ALL *R*, *Z* at each element in *t*).
- **return_t** – Boolean. Set to True to return a tuple of (volnorm, time_idxs), where time_idxs is the array of time indices actually used in evaluating volnorm with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return volnorm).
- **sqrt** – Boolean. Set to True to return the square root of volnorm. Only the square root of positive values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return volnorm itself).
- **kind** – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to volnorm. This is passed to scipy.interpolate.interp1d. Valid options are: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.

Returns

volnorm or (volnorm, time_idxs)

- **volnorm** - Array or scalar float. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. volnorm will have the same shape as t and psi_norm (or whichever one is Array-like).
- **time_idxs** - Array with same shape as volnorm. The indices (in the timebase returned by `getTimeBase()`) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single volnorm value for psinorm=0.7, t=0.26s:

```
volnorm_val = Eq_instance.psinorm2volnorm(0.7, 0.26)
```

Find volnorm values at psi_norm values of 0.5 and 0.7 at the single time t=0.26s. Note that the Z vector must be fully specified, even if the values are all the same:

```
volnorm_arr = Eq_instance.psinorm2volnorm([0.5, 0.7], 0.26)
```

Find volnorm values at psi_norm=0.5 at times t=[0.2s, 0.3s]:

```
volnorm_arr = Eq_instance.psinorm2volnorm(0.5, [0.2, 0.3])
```

Find volnorm values at (psinorm, t) points (0.6, 0.2s) and (0.5, 0.3s):

```
volnorm_arr = Eq_instance.psinorm2volnorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

psinorm2phinorm (*psi_norm*, *t*, ***kwargs*)

Calculates the normalized toroidal flux corresponding to the passed psi_norm (normalized poloidal flux) values.

If tspline is False for this Equilibrium instance, uses scipy.interpolate.RectBivariateSpline to interpolate in terms of R and Z. Finds the nearest time slices to those given: nearest-neighbor interpolation in time. Otherwise, uses the tricubic package to perform a trivariate interpolation in space and time.

Parameters

- **psi_norm** – Array-like or scalar float. Values of the normalized poloidal flux to map to normalized toroidal flux. If psi_norm is a scalar, it is used as the value for all of the values in t.
- **t** – Array-like or single value. If t is a single value, it is used for all of the elements of psi_norm. If neither t nor psi_norm are scalars, t must have the same shape as psi_norm.

Keyword Arguments

- **each_t** – Boolean. When True, the elements in *R* and *Z* (or the meshgrid thereof if *make_grid* is True) are evaluated at each value in *t*. If True, *t* must have only one dimension (or be a scalar). If False, *t* must match the shape of *R* and *Z* (or their meshgrid if *make_grid* is True) or be a scalar. Default is True (evaluate ALL *R*, *Z* at each element in *t*).
- **return_t** – Boolean. Set to True to return a tuple of (phinorm, time_idxs), where time_idxs is the array of time indices actually used in evaluating phinorm with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return phinorm).
- **kind** – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to phinorm. This is passed to scipy.interpolate.interp1d. Valid options are: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.

Returns

phinorm or (phinorm, time_idxs)

- **phinorm** - Array or scalar float. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. phinorm will have the same shape as t and psi_norm (or whichever one is Array-like).
- **time_idxs** - Array with same shape as phinorm. The indices (in the timebase returned by `getTimeBase()`) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single phinorm value for psinorm=0.7, t=0.26s:

```
phinorm_val = Eq_instance.psinorm2phinorm(0.7, 0.26)
```

Find phinorm values at psi_norm values of 0.5 and 0.7 at the single time t=0.26s. Note that the Z vector must be fully specified, even if the values are all the same:

```
phinorm_arr = Eq_instance.psinorm2phinorm([0.5, 0.7], 0.26)
```

Find phinorm values at psi_norm=0.5 at times t=[0.2s, 0.3s]:

```
phinorm_arr = Eq_instance.psinorm2phinorm(0.5, [0.2, 0.3])
```

Find phinorm values at (psinorm, t) points (0.6, 0.2s) and (0.5, 0.3s):

```
phinorm_arr = Eq_instance.psinorm2phinorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

getInfo()

Abstract method. See child classes for implementation.

Returns namedtuple of instance parameters (shot, equilibrium type, size, timebase, etc.)

getTimeBase()

Abstract method. See child classes for implementation.

Returns timebase array [t]

getFluxGrid()

Abstract method. See child classes for implementation.

returns 3D grid of psi(r,z,t)

The array returned should have the following dimensions: First dimension: time Second dimension: Z Third dimension: R

getRGrid()

Abstract method. See child classes for implementation.

Returns vector of R-values for psiRZ grid [r]

getZGrid()

Abstract method. See child classes for implementation.

Returns vector of Z-values for psiRZ grid [z]

getFluxAxis()

Abstract method. See child classes for implementation.

Returns psi at magnetic axis [t]

getFluxLCFS()

Abstract method. See child classes for implementation.

Returns psi a separatrix [t]

getRLCFS()

Abstract method. See child classes for implementation.

Returns R-positions (n points) mapping LCFS [t,n]

getZLCFS()

Abstract method. See child classes for implementation.

Returns Z-positions (n points) mapping LCFS [t,n]

remapLCFS()
Abstract method. See child classes for implementation.

Overwrites stored R,Z positions of LCFS with explicitly calculated psinorm=1 surface. This surface is then masked using core.inPolygon() to only draw within vacuum vessel, the end result replacing RLCFS, ZLCFS with an R,Z array showing the divertor legs of the flux surface in addition to the core-enclosing closed flux surface.

getFluxVol()
Abstract method. See child classes for implementation.

Returns volume contained within flux surface as function of psi [psi,t]. Psi assumed to be evenly-spaced grid on [0,1]

getVolLCFS()
Abstract method. See child classes for implementation.

Returns plasma volume within LCFS [t]

getRmidPsi()
Abstract method. See child classes for implementation.

Returns outboard-midplane major radius of flux surface [t,psi]

getFluxPres()
Abstract method. See child classes for implementation.

Returns calculated pressure profile [psi,t]. Psi assumed to be evenly-spaced grid on [0,1]

getElongation()
Abstract method. See child classes for implementation.

Returns LCFS elongation [t]

getUpperTriangularity()
Abstract method. See child classes for implementation.

Returns LCFS upper triangularity [t]

getLowerTriangularity()
Abstract method. See child classes for implementation.

Returns LCFS lower triangularity [t]

getShaping()
Abstract method. See child classes for implementation.

Returns dimensionless shaping parameters for plasma. Namedtuple containing {LCFS elongation, LCFS upper/lower triangularity}

getMagR()
Abstract method. See child classes for implementation.

Returns magnetic-axis major radius [t]

getMagZ()
Abstract method. See child classes for implementation.

Returns magnetic-axis Z [t]

getAreaLCFS()
Abstract method. See child classes for implementation.

Returns LCFS surface area [t]

getAOut()
Abstract method. See child classes for implementation.

Returns outboard-midplane minor radius [t]

getRmidOut()
Abstract method. See child classes for implementation.

Returns outboard-midplane major radius [t]

getGeometry()
Abstract method. See child classes for implementation.

Returns dimensional geometry parameters Namedtuple containing {mag axis R,Z, LCFS area, volume, outboard-midplane major radius}

getQProfile()
Abstract method. See child classes for implementation.

Returns safety factor q profile [psi,t] Psi assumed to be evenly-spaced grid on [0,1]

getQ0()
Abstract method. See child classes for implementation.

Returns q on magnetic axis [t]

getQ95()
Abstract method. See child classes for implementation.

Returns q on 95% flux surface [t]

getQLCFS()
Abstract method. See child classes for implementation.

Returns q on LCFS [t]

getQ1Surf()
Abstract method. See child classes for implementation.

Returns outboard-midplane minor radius of q=1 surface [t]

getQ2Surf()
Abstract method. See child classes for implementation.

Returns outboard-midplane minor radius of q=2 surface [t]

getQ3Surf()
Abstract method. See child classes for implementation.

Returns outboard-midplane minor radius of q=3 surface [t]

getQs()
Abstract method. See child classes for implementation.

Returns specific q-profile values. Namedtuple containing {q0, q95, qLCFS, minor radius of q=1,2,3 surfaces}

getBtVac()
Abstract method. See child classes for implementation.

Returns vacuum on-axis toroidal field [t]

getBtPla()
Abstract method. See child classes for implementation.

Returns plasma on-axis toroidal field [t]

getBpAvg()
Abstract method. See child classes for implementation.

Returns average poloidal field [t]

getFields()
Abstract method. See child classes for implementation.

Returns magnetic-field values. Namedtuple containing {Btor on magnetic axis (plasma and vacuum), avg Bpol}

getIpCalc()
Abstract method. See child classes for implementation.

Returns calculated plasma current [t]

getIpMeas()
Abstract method. See child classes for implementation.

Returns measured plasma current [t]

getJp()
Abstract method. See child classes for implementation.

Returns grid of calculated toroidal current density [t,z,r]

getBetaT()
Abstract method. See child classes for implementation.

Returns calculated global toroidal beta [t]

getBetaP()
Abstract method. See child classes for implementation.

Returns calculated global poloidal beta [t]

getLi()
Abstract method. See child classes for implementation.

Returns calculated internal inductance of plasma [t]

getBetas()
Abstract method. See child classes for implementation.

Returns calculated betas and inductance. Namedtuple of {betat,betap,Li}

getDiamagFlux()
Abstract method. See child classes for implementation.

Returns diamagnetic flux [t]

getDiamagBetaT()
Abstract method. See child classes for implementation.

Returns diamagnetic-loop toroidal beta [t]

getDiamagBetaP()
Abstract method. See child classes for implementation.

Returns diamagnetic-loop poloidal beta [t]

getDiamagTauE()
Abstract method. See child classes for implementation.

Returns diamagnetic-loop energy confinement time [t]

getDiamagWp()
Abstract method. See child classes for implementation.

Returns diamagnetic-loop plasma stored energy [t]

getDiamag()
Abstract method. See child classes for implementation.

Returns diamagnetic measurements of plasma parameters. Namedtuple of {diamag. flux, betat, betap from coils, tau_E from diamag., diamag. stored energy}

getWMHD()
Abstract method. See child classes for implementation.

Returns calculated MHD stored energy [t]

getTauMHD()
Abstract method. See child classes for implementation.

Returns calculated MHD energy confinement time [t]

getPinj()
Abstract method. See child classes for implementation.

Returns calculated injected power [t]

getCurrentSign()
Abstract method. See child classes for implementation.

Returns calculated current direction, where CCW = +

getWbdot()
Abstract method. See child classes for implementation.

Returns calculated d/dt of magnetic stored energy [t]

getWpdot()
Abstract method. See child classes for implementation.

Returns calculated d/dt of plasma stored energy [t]

getEnergy()
Abstract method. See child classes for implementation.

Returns stored-energy parameters. Namedtuple of {stored energy, confinement time, injected power, d/dt of magnetic, plasma stored energy}

getParam(*path*)
Abstract method. See child classes for implementation.

Backup function: takes parameter name for variable, returns variable directly. Acts as wrapper to direct data-access routines from within object.

getMachineCrossSection()
Abstract method. See child classes for implementation.

Returns (R,Z) coordinates of vacuum wall cross-section for plotting/masking routines.

getMachineCrossSectionFull()
Abstract method. See child classes for implementation.

Returns (R,Z) coordinates of machine wall cross-section for plotting routines. Returns a more detailed cross-section than getLimiter(), generally a vector map displaying non-critical cross-section information. If this is unavailable, this should point to self.getMachineCrossSection(), which pulls the limiter outline stored by default in data files e.g. g-eqdsk files.

plotFlux(*fill=False*)

Plots flux contours directly from psi grid.

Keyword Arguments **fill** – Boolean. Set True to plot filled contours. Set False (default) to plot white-background color contours.

2.1.8 `eqdskreader` Module

This module contains the EqdskReader class, which creates Equilibrium class functionality for equilibria stored in eqdsk files from EFIT(a- and g-files).

Classes:

EqdskReader: class inheriting Equilibrium reading g- and a-files for equilibrium data.

class `eqtools.eqdskreader.`**`EqdskReader`**(*shot=None*, *time=None*, *gfile=None*, *afile=None*, *length_unit='m'*, *verbose=True*)

Bases: `eqtools.core.Equilibrium`

Equilibrium subclass working from eqdsk ASCII-file equilibria.

Inherits mapping and structural data from Equilibrium, populates equilibrium and profile data from g- and a-files for a selected shot and time window.

Create instance of EqdskReader.

Generates object and reads data from selected g-file (either manually set or autodetected based on user shot and time selection), storing as object attributes for usage in Equilibrium mapping methods.

Calling structure - user may call class with shot and time (ms) values, set by keywords (or positional placement allows calling without explicit keyword syntax). EqdskReader then attempts to construct filenames from the shot/time, of the form 'g[shot].[time]' and 'a[shot].[time]'. Alternately, the user may skip this input and explicitly set paths to the g- and/or a-files, using the gfile and afile keyword arguments. If both types of calls are set, the explicit g-file and a-file paths override the auto-generated filenames from the shot and time.

Keyword Arguments

- **shot** – Int. Shot index.
- **time** – Int. Time index (typically ms). Shot and Time used to autogenerate filenames.
- **gfile** – String. Manually selects ASCII file for equilibrium read.
- **afile** – String. Manually selects ASCII file for time-history read.
- **length_unit** – String. Flag setting length unit for equilibrium scales. Defaults to 'm' for lengths in meters.
- **verbose** – Boolean. When set to False, suppresses terminal outputs during CSV read. Defaults to True (prints terminal output).

Raises

- `IOError` – if both name/shot and explicit filenames are not set.
- `ValueError` – if the g-file cannot be found, or if multiple valid g/a-files are found.

getInfo()

returns namedtuple of equilibrium information

Returns

namedtuple containing

shot	shot index
time	time point of g-file
nr	size of R-axis of spatial grid
nz	size of Z-axis of spatial grid
efittype	EFIT calculation type (magnetic, kinetic, MSE)

readAFile (*afile*)

Reads a-file (scalar time-history data) to pull additional equilibrium data not found in g-file, populates remaining data (initialized as None) in object.

Parameters **afile** – String. Path to ASCII a-file.

Raises `IOError` – If afile is not found.

rz2psi (*R*, *Z*, **args*, ***kwargs*)

Converts passed, R,Z arrays to psi values.

Wrapper for Equilibrium.rz2psi masking out timebase dependence.

Parameters

- **R** – Array-like or scalar float. Values of the radial coordinate to map to poloidal flux. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to poloidal flux. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).
- ***args** –

  Slot for time input for consistent syntax with Equilibrium.rz2psi. will return dummy value for time if input in EqdskReader.

Keyword Arguments

- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. Default is False (do not form meshgrid).
- **length_unit** – String or 1. Length unit that R and Z are being given in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

  If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters).
- ****kwargs** –

  Other keywords (i.e., return_t) to rz2psi are valid (necessary for proper inheritance and usage in other mapping routines) but will return dummy values.

Returns

psi – Array or scalar float. If all of the input arguments are scalar,

then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then psi has this shape as well. If the make_grid keyword was True then psi has shape (len(Z), len(R)).

rz2psinorm(*R*, *Z*, **args*, ***kwargs*)

Calculates the normalized poloidal flux at the given (R,Z). Wrapper for Equilibrium.rz2psinorm masking out timebase dependence.

Uses the definition: psi_norm = (psi - psi(0)) / (psi(a) - psi(0))

Parameters

- **R** – Array-like or scalar float. Values of the radial coordinate to map to normalized poloidal flux. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as Z unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to normalized poloidal flux. If R and Z are both scalar values, they are used as the coordinate pair for all of the values in t. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).
- ***args** –

  Slot for time input for consistent syntax with Equilibrium.rz2psinorm. will return dummy value for time if input in EqdskReader.

Keyword Arguments

- **sqrt** – Boolean. Set to True to return the square root of normalized flux. Only the square root of positive psi_norm values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return psinorm).
- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. Default is False (do not form meshgrid).
- **length_unit** – String or 1. Length unit that R and Z are being given in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

  If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters).
- ****kwargs** –

  Other keywords passed to Equilibrium.rz2psinorm are valid, but will return dummy values (i.e. for timebase keywords)

Returns

psinorm – Array or scalar float. If all of the input arguments are

scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then psinorm has this shape as well. If the make_grid keyword was True then psinorm has shape (len(Z), len(R)).

Examples

All assume that Eq_instance is a valid instance EqdskReader:

Find single psinorm value at R=0.6m, Z=0.0m:

```
psi_val = Eq_instance.rz2psinorm(0.6, 0)
```

Find psinorm values at (R, Z) points (0.6m, 0m) and (0.8m, 0m). Note that the Z vector must be fully specified, even if the values are all the same:

```
psi_arr = Eq_instance.rz2psinorm([0.6, 0.8], [0, 0])
```

Find psinorm values on grid defined by 1D vector of radial positions R and 1D vector of vertical positions Z:

```
psi_mat = Eq_instance.rz2psinorm(R, Z, make_grid=True)
```

rz2phinorm(*R*, *Z*, **args*, ***kwargs*)

Calculates normalized toroidal flux at a given (R,Z).

Wrapper for Equilibrium.rz2phinorm masking out timebase dependence.

Parameters

- **R** – Array-like or scalar float. Values of the radial coordinate to map to normalized toroidal flux. Must have the same shape as Z unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to normalized toroidal flux. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).
- ***args** –

  Slot for time input for consistent syntax with Equilibrium.rz2phinorm. will return dummy value for time if input in EqdskReader.

Keyword Arguments

- **sqrt** – Boolean. Set to True to return the square root of normalized flux. Only the square root of positive phi_norm values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return phinorm).
- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. Default is False (do not form meshgrid).
- **kind** – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to phinorm. This is passed to scipy.interpolate.interp1d. Valid options are: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more

details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.

- **length_unit** – String or 1. Length unit that R and Z are being given in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

  If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters).

- ****kwargs** –

  Other keywords passed to Equilibrium.rz2phinorm are valid, but will return dummy values (i.e. for timebase keywords)

Returns

phinorm – Array or scalar float. If all of the input arguments are

scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then phinorm has this shape as well. If the make_grid keyword was True then phinorm has shape (len(Z), len(R)).

Examples

All assume that Eq_instance is a valid instance of EqdskReader.

Find single phinorm value at R=0.6m, Z=0.0m:

```
phi_val = Eq_instance.rz2phinorm(0.6, 0)
```

Find phinorm values at (R, Z) points (0.6m, 0m) and (0.8m, 0m). Note that the Z vector must be fully specified, even if the values are all the same:

```
phi_arr = Eq_instance.rz2phinorm([0.6, 0.8], [0, 0])
```

Find phinorm values on grid defined by 1D vector of radial positions R and 1D vector of vertical positions Z:

```
phi_mat = Eq_instance.rz2phinorm(R, Z, make_grid=True)
```

rz2volnorm(**args*, ***kwargs*)

Calculates the normalized flux surface volume.

Not implemented for EqdskReader, as necessary parameter is not read from a/g-files.

Raises `NotImplementedError` – in all cases.

rz2rho (*method*, *R*, *Z*, *t=False*, *sqrt=False*, *make_grid=False*, *kind='cubic'*, *length_unit=1*)

Convert the passed (R, Z) coordinates into one of several normalized coordinates. Wrapper for Equilibrium.rz2rho masking timebase dependence.

Parameters

- **method** – String. Indicates which normalized coordinates to use. Valid options are:

psinorm	Normalized poloidal flux
phinorm	Normalized toroidal flux
volnorm	Normalized volume

- **R** – Array-like or scalar float. Values of the radial coordinate to map to normalized coordinate. Must have the same shape as Z unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to normalized coordinate. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).

Keyword Arguments

- **t** – indeterminant. Provides duck typing for inclusion of t values. Passed t values either as an Arg or Kwarg are neglected.
- **sqrt** – Boolean. Set to True to return the square root of normalized coordinate. Only the square root of positive values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return normalized coordinate itself).
- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. Default is False (do not form meshgrid).
- **kind** (*phinorm and volnorm only*) – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to phinorm or volnorm. This is passed to scipy.interpolate.interp1d. Valid options are: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.
- **length_unit** – String or 1. Length unit that R and Z are being given in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters).

Returns

rho – Array or scalar float. If all of the input arguments are

scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then rho has this shape as well. If the make_grid keyword was True then rho has shape (len(Z), len(R)).

Raises `ValueError` – If method is not one of the supported values.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single psinorm value at R=0.6m, Z=0.0m:

```
psi_val = Eq_instance.rz2rho('psinorm', 0.6, 0)
```

Find psinorm values at (R, Z) points (0.6m, 0m) and (0.8m, 0m). Note that the Z vector must be fully specified, even if the values are all the same:

```
psi_arr = Eq_instance.rz2rho('psinorm', [0.6, 0.8], [0, 0])
```

Find psinorm values on grid defined by 1D vector of radial positions R and 1D vector of vertical positions Z:

```
psi_mat = Eq_instance.rz2rho('psinorm', R, Z, make_grid=True)
```

rz2rmid(*R*, *Z*, *t=False*, *sqrt=False*, *make_grid=False*, *rho=False*, *kind='cubic'*, *length_unit=1*)
Maps the given points to the outboard midplane major radius, R_mid. Wrapper for Equilibrium.rz2rmid masking timebase dependence.

Based on the IDL version efit_rz2rmid.pro by Steve Wolfe.

Parameters

- **R** – Array-like or scalar float. Values of the radial coordinate to map to midplane radius. Must have the same shape as Z unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- **Z** – Array-like or scalar float. Values of the vertical coordinate to map to midplane radius. Must have the same shape as R unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).

Keyword Arguments

- **t** – indeterminant. Provides duck typing for inclusion of t values. Passed t values either as an Arg or Kwarg are neglected.
- **sqrt** – Boolean. Set to True to return the square root of midplane radius. Only the square root of positive values is taken. Negative values are replaced with zeros, consistent with Steve Wolfe's IDL implementation efit_rz2rho.pro. Default is False (return R_mid itself).
- **make_grid** – Boolean. Set to True to pass R and Z through meshgrid before evaluating. If this is set to True, R and Z must each only have a single dimension, but can have different lengths. Default is False (do not form meshgrid).
- **rho** – Boolean. Set to True to return r/a (normalized minor radius) instead of R_mid. Default is False (return major radius, R_mid).
- **kind** – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to R_mid. This is passed to scipy.interpolate.interp1d. Valid options

are: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.

- **length_unit** – String or 1.

  Length unit that R and Z are being given in AND that R_mid is returned in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

If length_unit is 1 or None, meters are assumed. The default value is 1 (R and Z given in meters, R_mid returned in meters).

Returns

R_mid – Array or scalar float. If all of the input arguments are

scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned. If R and Z both have the same shape then R_mid has this shape as well. If the make_grid keyword was True then R_mid has shape (len(Z), len(R)).

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single R_mid value at R=0.6m, Z=0.0m:

```
R_mid_val = Eq_instance.rz2rmid(0.6, 0)
```

Find R_mid values at (R, Z) points (0.6m, 0m) and (0.8m, 0m). Note that the Z vector must be fully specified, even if the values are all the same:

```
R_mid_arr = Eq_instance.rz2rmid([0.6, 0.8], [0, 0])
```

Find R_mid values on grid defined by 1D vector of radial positions R and 1D vector of vertical positions Z:

```
R_mid_mat = Eq_instance.rz2rmid(R, Z, make_grid=True)
```

psinorm2rmid(*psi_norm*, *t=False*, *rho=False*, *kind='cubic'*, *length_unit=1*)

Calculates the outboard R_mid location corresponding to the passed psi_norm (normalized poloidal flux) values.

Parameters **psi_norm** – Array-like or scalar float. Values of the normalized poloidal flux to map to midplane radius.

Keyword Arguments

- **t** – indeterminant. Provides duck typing for inclusion of t values. Passed t values either as an Arg or Kwarg are neglected.
- **rho** – Boolean. Set to True to return r/a (normalized minor radius) instead of R_mid. Default is False (return major radius, R_mid).
- **kind** – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to R_mid. This is passed to scipy.interpolate.interp1d. Valid options are: 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.
- **length_unit** – String or 1. Length unit that R_mid is returned in. If a string is given, it must be a valid unit specifier:

'm'	meters
'cm'	centimeters
'mm'	millimeters
'in'	inches
'ft'	feet
'yd'	yards
'smoot'	smoots
'cubit'	cubits
'hand'	hands
'default'	meters

  If length_unit is 1 or None, meters are assumed. The default value is 1 (R_mid returned in meters).

Returns

R_mid – Array or scalar float. If all of the input arguments are

scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single R_mid value for psinorm=0.7:

```
R_mid_val = Eq_instance.psinorm2rmid(0.7)
```

Find R_mid values at psi_norm values of 0.5 and 0.7. Note that the Z vector must be fully specified, even if the values are all the same:

```
R_mid_arr = Eq_instance.psinorm2rmid([0.5, 0.7])
```

psinorm2volnorm(**args*, ***kwargs*)
Calculates the outboard R_mid location corresponding to psi_norm (normalized poloidal flux) values. Not implemented for EqdskReader, as necessary parameter is not read from a/g-files.

psinorm2phinorm(*psi_norm*, *t=False*, *kind='cubic'*)
Calculates the normalized toroidal flux corresponding to the passed psi_norm (normalized poloidal flux) values.

Parameters **psi_norm** – Array-like or scalar float. Values of the normalized poloidal flux to map to normalized toroidal flux.

Keyword Arguments

- **t** – indeterminant. Provides duck typing for inclusion of t values. Passed t values either as an Arg or Kwarg are neglected.
- **kind** – String or non-negative int. Specifies the type of interpolation to be performed in getting from psinorm to phinorm. This is 'linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic' passed to scipy.interpolate.interp1d. Valid options are: If this keyword is an integer, it specifies the order of spline to use. See the documentation for interp1d for more details. Default value is 'cubic' (3rd order spline interpolation). On some builds of scipy, this can cause problems, in which case you should try 'linear' until you can rebuild your scipy install.

Returns

phinorm – Array or scalar float. If all of the input arguments are

scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned.

Examples

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.

Find single phinorm value for psinorm=0.7:

```
phinorm_val = Eq_instance.psinorm2phinorm(0.7)
```

Find phinorm values at psi_norm values of 0.5 and 0.7. Note that the Z vector must be fully specified, even if the values are all the same:

```
phinorm_arr = Eq_instance.psinorm2phinorm([0.5, 0.7])
```

getTimeBase()
Returns EFIT time point

getCurrentSign()
Returns the sign of the current, based on the check in Steve Wolfe's IDL implementation efit_rz2psi.pro.

getFluxGrid()
Returns EFIT flux grid, [r,z]

getRGrid(*length_unit=1*)
Returns EFIT R-axis [r]

getZGrid(*length_unit=1*)
Returns EFIT Z-axis [z]

getFluxAxis()
Returns psi on magnetic axis

getFluxLCFS()
Returns psi at separatrix

getRLCFS(*length_unit=1*)
Returns array of R-values of LCFS

getZLCFS(*length_unit=1*)
Returns array of Z-values of LCFS

remapLCFS(*mask=True*)
Overwrites RLCFS, ZLCFS values pulled from EFIT with explicitly-calculated contour of psinorm=1 surface.

Keyword Arguments mask – Boolean. Default True. Set True to mask LCFS path to limiter outline (using inPolygon). Set False to draw full contour of psi = psiLCFS.

getFluxVol()

getVolLCFS(*length_unit=3*)
Returns volume with LCFS.

Raises `ValueError` – if a-file data is not read.

getRmidPsi()
Returns outboard-midplane major radius of flux surfaces.

Data not read from a/g-files, not implemented for EqdskReader.

Raises `NotImplementedError` – RmidPsi not read from a/g-files.

getFluxPres()
Returns pressure on flux surface p(psi)

getElongation()
Returns elongation of LCFS.

Raises `ValueError` – if a-file data is not read.

getUpperTriangularity()
Returns upper triangularity of LCFS.

Raises `ValueError` – if a-file data is not read.

getLowerTriangularity()
Returns lower triangularity of LCFS.

Raises `ValueError` – if a-file data is not read.

getShaping()
Pulls LCFS elongation, upper/lower triangularity.

Returns namedtuple containing [kappa,delta_u,delta_l].

Raises `ValueError` – if a-file data is not read.

getMagR(*length_unit=1*)
Returns major radius of magnetic axis.

Raises `ValueError` – if a-file data is not read.

getMagZ(*length_unit=1*)
Returns Z of magnetic axis.

Raises `ValueError` – if a-file data is not read.

getAreaLCFS(*length_unit=2*)
Returns surface area of LCFS.

Raises `ValueError` – if a-file data is not read.

getAOut(*length_unit=1*)
Returns outboard-midplane minor radius of LCFS.

Raises `ValueError` – if a-file data is not read.

getRmidOut(*length_unit=1*)
Returns outboard-midplane major radius of LCFS.

Raises `ValueError` – if a-file data is not read.

getGeometry(*length_unit=None*)
Pulls dimensional geometry parameters.

Returns namedtuple containing [Rmag,Zmag,AreaLCFS,aOut,RmidOut]

Keyword Arguments **length_unit** – TODO

Raises `ValueError` – if a-file data is not read.

getQProfile()
Returns safety factor q(psi).

getQ0()
Returns safety factor q on-axis, q0.

Raises `ValueError` – if a-file data is not read.

getQ95()
Returns safety factor q at 95% flux surface.

Raises `ValueError` – if a-file data is not read.

getQLCFS()
Returns safety factor q at LCFS (interpolated).

Raises `ValueError` – if a-file data is not loaded.

getQ1Surf(*length_unit=1*)
Returns outboard-midplane minor radius of q=1 surface.

Raises `ValueError` – if a-file data is not read.

getQ2Surf(*length_unit=1*)
Returns outboard-midplane minor radius of q=2 surface.

Raises `ValueError` – if a-file data is not read.

getQ3Surf(*length_unit=1*)
Returns outboard-midplane minor radius of q=3 surface.

Raises `ValueError` – if a-file data is not read.

getQs(*length_unit=1*)
Pulls q-profile data.

Returns namedtuple containing [q0,q95,qLCFS,rq1,rq2,rq3]

Raises `ValueError` – if a-file data is not read.

getBtVac()
Returns vacuum toroidal field on-axis.

Raises `ValueError` – if a-file data is not read.

getBtPla()
Returns plasma toroidal field on-axis.

Raises `ValueError` – if a-file data is not read.

getBpAvg()
Returns average poloidal field.

Raises `ValueError` – if a-file data is not read.

getFields()
Pulls vacuum and plasma toroidal field, poloidal field data.

Returns namedtuple containing [BtVac,BtPla,BpAvg]

Raises `ValueError` – if a-file data is not read.

getIpCalc()
Returns EFIT-calculated plasma current.

getIpMeas()
Returns measured plasma current.

Raises `ValueError` – if a-file data is not read.

getJp()
Returns (r,z) grid of toroidal plasma current density.

Data not read from g-file, not implemented for EqdskReader.

Raises `NotImplementedError` – Jp not read from g-file.

getBetaT()
Returns EFIT-calculated toroidal beta.

Raises `ValueError` – if a-file data is not read.

getBetaP()
Returns EFIT-calculated poloidal beta.

Raises `ValueError` – if a-file data is not read

getLi()
Returns internal inductance of plasma.

Raises `ValueError` – if a-file data is not read.

getBetas()
Pulls EFIT-calculated betas and internal inductance.

Returns namedtuple containing [betat,betap,Li]

Raises `ValueError` – if a-file data is not read.

getDiamagFlux()
Returns diamagnetic flux.

Raises `ValueError` – if a-file data is not read.

getDiamagBetaT()
Returns diamagnetic-loop measured toroidal beta.

Raises `ValueError` – if a-file data is not read.

getDiamagBetaP()
Returns diamagnetic-loop measured poloidal beta.

Raises `ValueError` – if a-file data is not read.

getDiamagTauE()
Returns diamagnetic-loop energy confinement time.

Raises `ValueError` – if a-file data is not read.

getDiamagWp()
Returns diamagnetic-loop measured stored energy.

Raises `ValueError` – if a-file data is not read.

getDiamag()
Pulls diamagnetic flux, diamag. measured toroidal and poloidal beta, stored energy, and energy confinement time.

Returns namedtuple containing [diaFlux,diaBetat,diaBetap,diaTauE,diaWp]

Raises `ValueError` – if a-file data is not read

getWMHD()
Returns EFIT-calculated stored energy.

Raises `ValueError` – if a-file data is not read.

getTauMHD()
Returns EFIT-calculated energy confinement time.

Raises `ValueError` – if a-file data is not read.

getPinj()
Returns EFIT injected power.

Raises `ValueError` – if a-file data is not read.

getWbdot()
Returns EFIT d/dt of magnetic stored energy

Raises `ValueError` – if a-file data is not read.

getWpdot()
Returns EFIT d/dt of plasma stored energy.

Raises `ValueError` – if a-file data is not read.

getEnergy()
Pulls EFIT stored energy, energy confinement time, injected power, and d/dt of magnetic and plasma stored energy.

Returns namedtuple containing [WMHD,tauMHD,Pinj,Wbdot,Wpdot]

Raises `ValueError` – if a-file data is not read.

getParam(*name*)
Backup function, applying a direct path input for tree-like data storage access for parameters not typically found in Equilbrium object. Directly calls attributes read from g/a-files in copy-safe manner.

Parameters **name** – String. Parameter name for value stored in EqdskReader instance.

Raises `AttributeError` – raised if no attribute is found.

getMachineCrossSection()
Method to pull machine cross-section from data storage, convert to standard format for plotting routine.

getMachineCrossSectionFull()
Returns vectorization of machine cross-section.

Absent additional data (not found in eqdsks) simply returns self.getMachineCrossSection().

plotFlux(*fill=True*, *mask=True*)
streamlined plotting of flux contours directly from psi grid

Keyword Arguments

- **fill** – Boolean. Default True. Set True to plot filled contours of flux delineated by black outlines. Set False to instead plot color-coded line contours on a blank background.
- **mask** – Boolean. Default True. Set True to draw a clipping mask based on the limiter outline for the flux contours. Set False to draw the full RZ grid.

2.1.9 `pfilereader` Module

This module contains the PFileReader class, a lightweight data handler for p-file (radial profile) datasets.

Classes:

PFileReader: Data-storage class for p-file data. Reads data from ASCII p-file, storing as copy-safe object attributes.

class `eqtools.pfilereader.`**`PFileReader`**(*pfile*, *verbose=True*)

Bases: `object`

Class to read ASCII p-file (profile data storage) into lightweight, user-friendly data structure.

P-files store data blocks containing the following: a header with parameter name, parameter units, x-axis units, and number of data points, followed by values of axis x, parameter y, and derivative dy/dx. Each parameter block is read into a namedtuple storing ['name','npts','units','xunits','x','y','dydx'], with each namedtuple stored as an attribute of the PFileReader instance. This gracefully handles variable formats of p-files (differing versions of p-files will have different parameters stored). Data blocks are accessed as attributes in a copy-safe manner.

Creates instance of PFileReader.

Parameters **pfile** – String. Path to ASCII p-file to be loaded.

Keyword Arguments **verbose** – Boolean. Option to print message on object creation listing available data parameters. Defaults to True.

`__str__`()

overrides default string method for useful output.

`__getattribute__`(*name*)

Copy-safe attribute retrieval method overriding default object.__getattribute__.

Tries to retrieve attribute as-written (first check for default object attributes). If that fails, looks for pseudo-private attributes, marked by preceding underscore, to retrieve data blocks. If this fails, raise AttributeError.

Parameters **name** – String. Name (without leading underscore for data variables) of attribute.

Raises `AttributeError` – if no attribute can be found.

`__setattr__`(*name*, *value*)

Copy-safe attribute setting method overriding default object.__setattr__.

Raises error if object already has attribute _{name} for input name, as such an attribute would interfere with automatic property generation in __getattribute__.

Parameters **name** – String. Attribute name.

Raises `AttributeError` – if attempting to create attribute with protected pseudo-private name.

2.1.10 `trispline` Module

This module provides interface to the tricubic spline interpolator. It also contains an enhanced bivariate spline which generates bounds errors.

class `eqtools.trispline.`**`Spline`**(*z*, *y*, *x*, *f*, *regular=True*, *fast=False*)

Tricubic interpolating spline with forced edge derivative equal zero conditions. It assumes a cartesian grid.

Create a new Spline instance.

Parameters

- **z** – 1-dimensional float. Values of the positions of the 1st Dimension of f. Must be monotonic without duplicates.
- **y** – 1-dimensional float. Values of the positions of the 2nd dimension of f. Must be monotonic without duplicates.
- **x** – 1-dimensional float. Values of the positions of the 3rd dimension of f. Must be monotonic without duplicates.
- **f** – 3-dimensional float array. f[z,y,x]. NaN and Inf will affect interpolation in 4x4x4 space about its value.

Keyword Arguments

- **regular** – Boolean. If the grid is known to be regular, forces matrix-based fast evaluation of interpolation.
- **fast** – Boolean Outdated input to test the indexing performance of the c code vs internal python handling.

Raises

- `ValueError` – If any of the dimensions do not match specified f dim
- `ValueError` – If x,y, or z are not monotonic

Examples

temp

`ev`(*z1*, *y1*, *x1*)

evaluates tricubic spline at point (x1,y1,z1) which is f[z1,y1,x1]. Data is grouped into the grid voxels so as to reuse calculated spline coefficents, thus speeding evaluation. It is recommended that it is evaluated outside of for loops to best utilize this feature.

Parameters

- **z1** – float or 1-dimensional float Position in z dimension. (First dimension of 3d valued grid)
- **y1** – float or 1-dimensional float Position in y dimension. (Second dimension of 3d valued grid)
- **x1** – float or 1-dimensional float Position in x dimension. (Third dimension of 3d valued grid)

Returns *val* – The interpolated value at (x1,y1,z1).

Raises `ValueError` – If any of the dimensions exceed the evaluation boundary of the grid

class `eqtools.trispline.`**`RectBivariateSpline`**(*x*, *y*, *z*, *bbox=[None, None, None, None]*, *kx=3*, *ky=3*, *s=0*, *bounds_error=True*, *fill_value=nan*)

Bases: `scipy.interpolate.fitpack2.RectBivariateSpline`

the lack of a graceful bounds error causes the fortran to fail hard. This masks the scipy.interpolate.RectBivariateSpline with a proper bound checker and value filler such that it will not fail in use for EqTools

ev (*xi*, *yi*)

Evaluate the rectBiVariateSpline at (xi,yi). (x,y)values are checked for being in the bounds of the interpolated data.

Parameters

- **xi** – float array input x dimensional values
- **yi** – float array input x dimensional values

Returns

val – float array

evaluated spline at points (x[i], y[i]), i=0,...,len(x)-1

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX