eqtools Documentation

Release 1.0

Mark Chilenski, Ian Faust and John Walk

September 26, 2014

CONTENTS

Homepage: https://github.com/PSFCPlasmaTools/eqtools

CHAPTER ONE

OVERVIEW

`eqtools` is a Python package for working with magnetic equilibrium reconstructions from magnetic plasma confinement devices. At present, interfaces exist for data from the Alcator C-Mod and NSTX MDSplus trees as well as eqdsk a- and g-files. `eqtools` is designed to be flexible and extensible such that it can become a uniform interface to perform mapping operations and accessing equilibrium data for any magnetic confinement device, regardless of how the data are accessed.

The main class of `eqtools` is the `Equilibrium`, which contains all of the coordinate mapping functions as well as templates for methods to fetch data (primarily dictated to the quantities computed by EFIT). Subclasses such as `EFITTree`, `CModEFITTree`, `NSTXEFITTree` and `EqdskReader` implement specific methods to access the data and convert it to the form needed for the routines in `Equilibrium`. These classes are smart about caching intermediate results, so you will get a performance boost by using the same instance throughout your analysis of a given shot.

CHAPTER

TWO

TUTORIAL: PERFORMING COORDINATE TRANSFORMS ON ALCATOR C-MOD DATA

The basic class for manipulating EFIT results stored in the Alcator C-Mod MDSplus tree is `CModEFITTree`. To load the data from a specific shot, simply create the `CModEFITTree` object with the shot number as the argument:

```
e = eqtools.CModEFITTree(1140729030)
```

The default EFIT to use is "ANALYSIS." If you want to use a different tree, such as "EFIT20," then you simply set this with the *tree* keyword:

```
e = eqtools.CModEFITTree(1140729030, tree='EFIT20')
```

`eqtools` understands units. The default is to convert all lengths to meters (whereas quantities in the tree are inconsistent – some are meters, some centimeters). If you want to specify a different default unit, use the *length_unit* keyword:

```
e = eqtools.CModEFITTree(1140729030, length_unit='cm')
```

Once this is loaded, you can access the data you would normally have to pull from specific nodes in the tree using convenient getter methods. For instance, to get the elongation as a function of time, you can run:

```
kappa = e.getElongation()
```

The timebase used for quantities like this is accessed with:

```
t = e.getTimeBase()
```

For length/area/volume quantities, `eqtools` understands units. The default is to return in whatever units you specified when creating the `CModEFITTree`, but you can override this with the *length_unit* keyword. For instance, to get the vertical position of the magnetic axis in mm, you can run:

```
Z_mag = e.getMagZ(length_unit='mm')
```

`eqtools` can map from almost any coordinate to any common flux surface label. For instance, say you want to know what the square root of normalized toroidal flux corresponding to a normalized flux surface volume of 0.5 is at t=1.0s. You can simply call:

```
rho = e.volnorm2phinorm(0.5, 1.0, sqrt=True)
```

If a list of times is provided, the default behavior is to evaluate all of the points to be converted at each of the times. So, to follow the mapping of normalized poloidal flux values [0.1, 0.5, 1.0] to outboard midplane major radius at time points [1.0, 1.25, 1.5, 1.75], you could call:

```
psinorm = e.psinorm2rmid([0.1, 0.5, 1.0], [1.0, 1.25, 1.5, 1.75])
```

This will return a 4-by-3 array: one row for each time, one column for each location. If you want to override this behavior and instead consider a sequence of (psi, t) points, set the *each_t* keyword to False:

```
psinorm = e.psinorm2rmid([0.3, 0.35], [1.0, 1.1], each_t=False)
```

This will return a two-element array with the Rmid values for (psinorm=0.3, t=1.0) and (psinorm=0.35, t=1.1).

For programmatically mapping between coordinates, the `rho2rho()` method is quite useful. To map from outboard midplane major radius to normalized flux surface volume, you can simply call:

```
e.rho2rho('Rmid', 'volnorm', 0.75, 1.0)
```

Finally, to get a look at the flux surfaces, simply run:

```
e.plotFlux()
```

CHAPTER

THREE

PACKAGE REFERENCE

3.1 eqtools package

3.1.1 Submodules

3.1.2 eqtools.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=True*)

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** (*integer*) – C-Mod shot index (long)

Keyword Arguments

- **tree** (*string*) – 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** (*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).

- **gfile** (*string*) – Optional input for EFIT geqdsk location name, defaults to 'g_eqdsk' (i.e., EFIT data are under tree::top.results.G_EQDSK)
- **afile** (*string*) – 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).

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

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

`getF`()
returns F=RB_{Phi}(Psi), often calculated for grad-shafranov solutions [t,psi]

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

`getFFPrime`()
returns FF' function used for grad-shafranov solutions [t,psi]

`getPPrime`()
returns plasma pressure gradient as a function of psi [t,psi]

`getQProfile`()
returns safety factor q [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]

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

Parameters **shot** (*integer*) – 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=True*)

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.

3.1.3 eqtools.EFIT module

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

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** (*integer*) – Shot number
- **tree** (*string*) – MDSplus tree to open to fetch EFIT data.
- **root** (*string*) – 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 [t,psi]

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

getF()
returns F=RB_{Phi}(Psi), often calculated for grad-shafranov solutions [t,psi]

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

getFFPrime()
returns FF' function used for grad-shafranov solutions [t,psi]

getPPrime()
returns plasma pressure gradient as a function of psi [t,psi]

getElongation()
returns LCFS elongation [t]

getUpperTriangularity()
returns LCFS upper triangularity [t]

getLowerTriangularity()
returns LCFS lower triangularity [t]

getSlhaping()
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 [t,psi]

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** (*string*) – The path to the MDSplus node you wish to pull in.

Returns The requested data.

3.1.4 eqtools.FromArrays module

class eqtools.FromArrays.**ArrayEquilibrium**(*psiRZ*, *rGrid*, *zGrid*, *time*, *q*, *fluxVol*, *psiLCFS*, *psiAxis*, *rmag*, *zmag*, *Rout*, ***kwargs*)
Bases: eqtools.core.Equilibrium

Class to represent an equilibrium specified as arrays of data.

Create ArrayEquilibrium instance from arrays of data.

Has very little checking on the shape/type of the arrays at this point.

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, (S, M). q profile evaluated at S 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.
- **psiLCFS** – Array-like, (M,). Flux at the last closed flux surface, given at M times.
- **psiAxis** – Array-like, (M,). Flux at the magnetic axis, given at M times.
- **rmag** – Array-like, (M,). Radial coordinate of the magnetic axis, given at M times.
- **zmag** – Array-like, (M,). Vertical coordinate of the magnetic axis, given at M times.
- **Rout** – Outboard midplane radius of the last closed flux surface.

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.

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)

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

getFluxLCFS()
returns psi at separatrix [t]

getFluxAxis()
returns psi on magnetic axis [t]

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

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

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

getRLCFS (*length_unit=1*)

getZLCFS (*length_unit=1*)

getCurrentSign ()

3.1.5 eqtools.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=True*)

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** (*integer*) – NSTX shot index (long)

Keyword Arguments

- **tree** (*string*) – Optional input for EFIT tree, defaults to 'EFIT01' (i.e., EFIT data are under EFIT01::top.results).
- **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
'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** (*string*) – Optional input for EFIT geqdsk location name, defaults to 'geqdsk' (i.e., EFIT data are under tree::top.results.GEQDSK)
- **afile** (*string*) – Optional input for EFIT aeqdsk location name, defaults to 'aeqdsk' (i.e., EFIT data are under tree::top.results.AEQDSK)
- **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]

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

Returns The requested data.

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]

getIpCalc()
Returns: EFIT-calculated plasma current [t]

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

getJp()
Not implemented in NSTXEFIT tree.

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

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=True*)
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.

3.1.6 eqtools.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** (*String*) – file 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.

3.1.7 eqtools.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 do the same thing.

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=True*, *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 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.

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 but module trispline did not load successfully.

`__str__``()`

String representation of this instance.

Returns **string** (*String*) – String describing this object.

`rho2rho`(*origin*, *destination*, **args*, ***kwargs*)

Convert from one coordinate to another.

Parameters

- **origin** (*String*) – Indicates which coordinates the data are given in. Valid options are:

RZ	R,Z coordinates
psinorm	Normalized poloidal flux
phinorm	Normalized toroidal flux
volnorm	Normalized volume
Rmid	Midplane major radius
r/a	Normalized minor radius

Additionally, each valid option may be prepended with 'sqrt' to specify the square root of the desired unit.

- **destination** (*String*) – Indicates which coordinates to convert to. Valid options are:

psinorm	Normalized poloidal flux
phinorm	Normalized toroidal flux
volnorm	Normalized volume
Rmid	Midplane major radius
r/a	Normalized minor radius

Additionally, each valid option may be prepended with 'sqrt' to specify the square root of the desired unit.

- **rho** (*Array-like or scalar float*) – Values of the starting coordinate to map to the new coordinate. Will be two arguments *R*, *Z* if *origin* is 'RZ'.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *rho*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *rho* (or the meshgrid of *R* and *Z* if *make_grid* is True).

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of *rho*. 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.
- **each_t** (*Boolean*) – When True, the elements in *rho* 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 *rho* or be a scalar. Default is True (evaluate ALL *rho* at EACH element in *t*).
- **make_grid** (*Boolean*) – Only applicable if *origin* is 'RZ'. Set to True to pass *R* and *Z* through `scipy.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 Rmid when *destination* is Rmid. Default is False (return major radius, Rmid).
- **length_unit** (*String or 1*) – Length unit that quantities are given/returned in, as applicable. 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 (use meters).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from psinorm to Rmid/phinorm/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.

- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

rho or (*rho*, *time_idxs*)

- **rho** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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 *origin* 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/a=0.6, t=0.26s:

```
psi_val = Eq_instance.rho2rho('r/a', 'psinorm', 0.6, 0.26)
```

Find psinorm values at r/a points 0.6 and 0.8 at the single time t=0.26s:

```
psi_arr = Eq_instance.rho2rho('r/a', 'psinorm', [0.6, 0.8], 0.26)
```

Find psinorm values at r/a of 0.6 at times t=[0.2s, 0.3s]:

```
psi_arr = Eq_instance.rho2rho('r/a', 'psinorm', 0.6, [0.2, 0.3])
```

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

```
psi_arr = Eq_instance.rho2rho('r/a', 'psinorm', [0.6, 0.5], [0.2, 0.3], each_t=False)
```

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 (unnormalized poloidal flux) values.

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 exactly one dimension.
- **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 exactly one dimension.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R*, *Z*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R* and *Z* (or their meshgrid if *make_grid* is True).

Keyword Arguments

- **each_t** (*Boolean*) – When True, the elements in *R*, *Z* 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 be a scalar. Default is True (evaluate ALL *R*, *Z* at EACH element in *t*).

- **make_grid** (*Boolean*) – Set to True to pass *R* and *Z* through `scipy.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*, *Z* are 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 (use meters).
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

psi or (*psi*, *time_idxs*)

- **psi** (*Array or scalar float*) - The unnormalized poloidal flux. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned. If *R* and *Z* both have the same shape then *psi* has this shape as well, unless the *make_grid* keyword was True, in which case *psi* has shape (len(*Z*), len(*R*)).
- **time_idxs** (Array with same shape as *psi*) - 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 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:

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

Parameters

- **R** (*Array-like or scalar float*) – Values of the radial coordinate to map to psinorm. 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 exactly one dimension.
- **Z** (*Array-like or scalar float*) – Values of the vertical coordinate to map to psinorm. 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 exactly one dimension.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R*, *Z*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R* and *Z* (or their meshgrid if *make_grid* is True).

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of psinorm. 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.
- **each_t** (*Boolean*) – When True, the elements in *R*, *Z* 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 be a scalar. Default is True (evaluate ALL *R*, *Z* at EACH element in *t*).
- **make_grid** (*Boolean*) – Set to True to pass *R* and *Z* through `scipy.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*, *Z* are 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 (use meters).
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

psinorm or (*psinorm*, *time_idxs*)

- **psinorm** (*Array or scalar float*) - The normalized poloidal flux. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned. If *R* and *Z* both have the same shape then *psinorm* has this shape as well, unless the *make_grid* keyword was True, in which case *psinorm* has shape (len(*Z*), len(*R*)).
- **time_idxs** (Array with same shape as *psinorm*) - 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 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:

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

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

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

Parameters

- **R** (*Array-like or scalar float*) – Values of the radial coordinate to map to phinorm. 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 exactly one dimension.

- **Z** (*Array-like or scalar float*) – Values of the vertical coordinate to map to phinorm. 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 exactly one dimension.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R*, *Z*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R* and *Z* (or their meshgrid if *make_grid* is True).

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of phinorm. 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.
- **each_t** (*Boolean*) – When True, the elements in *R*, *Z* 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 be a scalar. Default is True (evaluate ALL *R*, *Z* at EACH element in *t*).
- **make_grid** (*Boolean*) – Set to True to pass *R* and *Z* through `scipy.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*, *Z* are 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 (use meters).
- **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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

phinorm or (*phinorm*, *time_idxs*)

- **phinorm** (*Array or scalar float*) - The normalized toroidal flux. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned. If *R* and *Z* both have the same shape then *phinorm* has this shape as well, unless the *make_grid* keyword was True, in which case *phinorm* has shape (len(*Z*), len(*R*)).

- **time_idxs** (Array with same shape as *phinorm*) - 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 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.

Parameters

- **R** (*Array-like or scalar float*) – Values of the radial coordinate to map to volnorm. 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 exactly one dimension.
- **Z** (*Array-like or scalar float*) – Values of the vertical coordinate to map to volnorm. 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 exactly one dimension.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R*, *Z*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R* and *Z* (or their meshgrid if *make_grid* is True).

Keyword Arguments

- **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.

- **each_t** (*Boolean*) – When True, the elements in *R*, *Z* 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 be a scalar. Default is True (evaluate ALL *R*, *Z* at EACH element in *t*).
- **make_grid** (*Boolean*) – Set to True to pass *R* and *Z* through `scipy.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*, *Z* are 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 (use meters).
- **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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

volnorm or (*volnorm*, *time_idxs*)

- **volnorm** (*Array or scalar float*) - The normalized volume. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned. If *R* and *Z* both have the same shape then *volnorm* has this shape as well, unless the *make_grid* keyword was True, in which case *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)
```

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

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

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 Rmid. 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 exactly one dimension.
- **Z** (*Array-like or scalar float*) – Values of the vertical coordinate to map to Rmid. 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 exactly one dimension.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R*, *Z*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R* and *Z* (or their meshgrid if *make_grid* is True).

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of Rmid. 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.
- **each_t** (*Boolean*) – When True, the elements in *R*, *Z* 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 be a scalar. Default is True (evaluate ALL *R*, *Z* at EACH element in *t*).
- **make_grid** (*Boolean*) – Set to True to pass *R* and *Z* through `scipy.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 Rmid. Default is False (return major radius, Rmid).
- **length_unit** (*String or 1*) – Length unit that *R*, *Z* are given in, AND that *Rmid* 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 (use meters).

- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from psinorm to Rmid. 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

Rmid or (*Rmid*, *time_idxs*)

- **Rmid** (*Array or scalar float*) - The outboard midplan major radius. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned. If *R* and *Z* both have the same shape then *Rmid* has this shape as well, unless the *make_grid* keyword was True, in which case *Rmid* has shape (len(*Z*), len(*R*)).
- **time_idxs** (Array with same shape as *Rmid*) - 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 Rmid 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 Rmid 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 Rmid 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 Rmid 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)
```

rz2roa(**args*, ***kwargs*)

Maps the given points to the normalized minor radius, r/a.

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 r/a. 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 exactly one dimension.
- **Z** (*Array-like or scalar float*) – Values of the vertical coordinate to map to r/a. 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 exactly one dimension.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R*, *Z*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R* and *Z* (or their meshgrid if *make_grid* is True).

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of 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.
- **each_t** (*Boolean*) – When True, the elements in *R*, *Z* 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 be a scalar. Default is True (evaluate ALL *R*, *Z* at EACH element in *t*).
- **make_grid** (*Boolean*) – Set to True to pass *R* and *Z* through `scipy.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*, *Z* are 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 (use meters).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from psinorm to Rmid. 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.

- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

roa or (*roa*, *time_idxs*)

- **roa** (*Array or scalar float*) - The normalized minor radius. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned. If *R* and *Z* both have the same shape then *roa* has this shape as well, unless the *make_grid* keyword was True, in which case *roa* has shape (len(*Z*), len(*R*)).
- **time_idxs** (Array with same shape as *roa*) - 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 r/a value at R=0.6m, Z=0.0m, t=0.26s:

```
roa_val = Eq_instance.rz2roa(0.6, 0, 0.26)
```

Find r/a 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:

```
roa_arr = Eq_instance.rz2roa([0.6, 0.8], [0, 0], 0.26)
```

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

```
roa_arr = Eq_instance.rz2roa(0.6, 0, [0.2, 0.3])
```

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

```
roa_arr = Eq_instance.rz2roa([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)
```

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

```
roa_mat = Eq_instance.rz2roa(R, Z, 0.2, make_grid=True)
```

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

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

Parameters

- **method** (*String*) – Indicates which coordinates to convert to. Valid options are:

psinorm	Normalized poloidal flux
phinorm	Normalized toroidal flux
volnorm	Normalized volume
Rmid	Midplane major radius
r/a	Normalized minor radius

Additionally, each valid option may be prepended with 'sqrt' to specify the square root of the desired unit.

- **R** (*Array-like or scalar float*) – Values of the radial coordinate to map to *rho*. 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 exactly one dimension.
- **Z** (*Array-like or scalar float*) – Values of the vertical coordinate to map to *rho*. 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 exactly one dimension.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R*, *Z*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R* and *Z* (or their meshgrid if *make_grid* is True).

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of *rho*. 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.
- **each_t** (*Boolean*) – When True, the elements in *R*, *Z* 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 be a scalar. Default is True (evaluate ALL *R*, *Z* at EACH element in *t*).
- **make_grid** (*Boolean*) – Set to True to pass *R* and *Z* through `scipy.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 Rmid when *destination* is Rmid. Default is False (return major radius, Rmid).
- **length_unit** (*String or 1*) – Length unit that *R*, *Z* are given in, AND that *Rmid* 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 (use meters).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from psinorm to Rmid/phinorm/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.

- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

rho or (*rho*, *time_idxs*)

- **rho** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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)
```

rmid2roa (*R_mid*, *t*, *each_t=True*, *return_t=False*, *sqrt=False*, *time_idxs=None*, *length_unit=1*)
Convert the passed (R_mid, t) coordinates into r/a.

Parameters

- **R_mid** (*Array-like or scalar float*) – Values of the outboard midplane major radius to map to r/a.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R_mid*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R_mid*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of 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.

- **each_t** (*Boolean*) – When True, the elements in *R_mid* 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_mid* or be a scalar. Default is True (evaluate ALL *R_mid* at EACH element in *t*).
- **length_unit** (*String or 1*) – Length unit that *R_mid* is 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 (use meters).
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

roa or (*roa*, *time_idxs*)

- **roa** (*Array or scalar float*) - Normalized midplane minor radius. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *roa*) - 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 r/a value at R_mid=0.6m, t=0.26s:

```
roa_val = Eq_instance.rmid2roa(0.6, 0.26)
```

Find roa values at R_mid points 0.6m and 0.8m at the single time t=0.26s.:

```
roa_arr = Eq_instance.rmid2roa([0.6, 0.8], 0.26)
```

Find roa values at R_mid of 0.6m at times t=[0.2s, 0.3s]:

```
roa_arr = Eq_instance.rmid2roa(0.6, [0.2, 0.3])
```

Find r/a values at (R_mid, t) points (0.6m, 0.2s) and (0.5m, 0.3s):

```
roa_arr = Eq_instance.rmid2roa([0.6, 0.5], [0.2, 0.3], each_t=False)
```

rmid2psinorm(*R_mid*, *t*, ***kwargs*)
Calculates the normalized poloidal flux corresponding to the passed R_mid (mapped outboard midplane major radius) values.

Parameters

- **R_mid** (*Array-like or scalar float*) – Values of the outboard midplane major radius to map to psinorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R_mid*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R_mid*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of psinorm. 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.
- **each_t** (*Boolean*) – When True, the elements in *R_mid* 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_mid* or be a scalar. Default is True (evaluate ALL *R_mid* at EACH element in *t*).
- **length_unit** (*String or 1*) – Length unit that *R_mid* is 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 (use meters).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from Rmid to psinorm. 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

psinorm or (*psinorm*, *time_idxs*)

- **psinorm** (*Array or scalar float*) - Normalized poloidal flux. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *psinorm*) - 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 psinorm value for Rmid=0.7m, t=0.26s:

```
psinorm_val = Eq_instance.rmid2psinorm(0.7, 0.26)
```

Find psinorm values at R_mid values of 0.5m and 0.7m at the single time t=0.26s:

```
psinorm_arr = Eq_instance.rmid2psinorm([0.5, 0.7], 0.26)
```

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

```
psinorm_arr = Eq_instance.rmid2psinorm(0.5, [0.2, 0.3])
```

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

```
psinorm_arr = Eq_instance.rmid2psinorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

rmid2phinorm(**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.

Parameters

- **R_mid** (*Array-like or scalar float*) – Values of the outboard midplane major radius to map to phinorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R_mid*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R_mid*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of phinorm. 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.
- **each_t** (*Boolean*) – When True, the elements in *R_mid* 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_mid* or be a scalar. Default is True (evaluate ALL *R_mid* at EACH element in *t*).
- **length_unit** (*String or 1*) – Length unit that *R_mid* is 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 (use meters).

- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from Rmid to psinorm and 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

phinorm or (*phinorm*, *time_idxs*)

- **phinorm** (*Array or scalar float*) - Normalized toroidal flux. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *phinorm*) - 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 phinorm value at R_mid=0.6m, t=0.26s:

```
phi_val = Eq_instance.rmid2phinorm(0.6, 0.26)
```

Find phinorm values at R_mid points 0.6m and 0.8m at the single time t=0.26s:

```
phi_arr = Eq_instance.rmid2phinorm([0.6, 0.8], 0.26)
```

Find phinorm values at R_mid point 0.6m at times t=[0.2s, 0.3s]:

```
phi_arr = Eq_instance.rmid2phinorm(0.6, [0.2, 0.3])
```

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

```
phi_arr = Eq_instance.rmid2phinorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

rmid2volnorm(**args*, ***kwargs*)

Calculates the normalized flux surface volume.

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

Parameters

- **R_mid** (*Array-like or scalar float*) – Values of the outboard midplane major radius to map to volnorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R_mid*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R_mid*.

Keyword Arguments

- **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.
- **each_t** (*Boolean*) – When True, the elements in *R_mid* 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_mid* or be a scalar. Default is True (evaluate ALL *R_mid* at EACH element in *t*).
- **length_unit** (*String or 1*) – Length unit that *R_mid* is 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 (use meters).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from Rmid to psinorm and 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

volnorm or (*volnorm*, *time_idxs*)

- **volnorm** (*Array or scalar float*) - Normalized volume. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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_mid=0.6m, t=0.26s:

```
vol_val = Eq_instance.rmid2volnorm(0.6, 0.26)
```

Find volnorm values at R_mid points 0.6m and 0.8m at the single time t=0.26s:

```
vol_arr = Eq_instance.rmid2volnorm([0.6, 0.8], 0.26)
```

Find volnorm values at R_mid points 0.6m at times t=[0.2s, 0.3s]:

```
vol_arr = Eq_instance.rmid2volnorm(0.6, [0.2, 0.3])
```

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

```
vol_arr = Eq_instance.rmid2volnorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

rmid2rho(*method*, *R_mid*, *t*, ***kwargs*)

Convert the passed (R_mid, t) coordinates into one of several coordinates.

Parameters

- **method** (*String*) – Indicates which coordinates to convert to. Valid options are:

psinorm	Normalized poloidal flux
phinorm	Normalized toroidal flux
volnorm	Normalized volume
r/a	Normalized minor radius

  Additionally, each valid option may be prepended with ‘sqrt’ to specify the square root of the desired unit.

- **R_mid** (*Array-like or scalar float*) – Values of the outboard midplane major radius to map to rho.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *R_mid*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *R_mid*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of rho. 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.
- **each_t** (*Boolean*) – When True, the elements in *R_mid* 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_mid* or be a scalar. Default is True (evaluate ALL *R_mid* at EACH element in *t*).
- **length_unit** (*String or 1*) – Length unit that *R_mid* is 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 (use meters).

- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from Rmid to psinorm and psinorm to volnorm or 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

rho or (*rho*, *time_idxs*)

- **rho** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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.

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_mid=0.6m, t=0.26s:

```
psi_val = Eq_instance.rmid2rho('psinorm', 0.6, 0.26)
```

Find psinorm values at R_mid points 0.6m and 0.8m at the single time t=0.26s.:

```
psi_arr = Eq_instance.rmid2rho('psinorm', [0.6, 0.8], 0.26)
```

Find psinorm values at R_mid of 0.6m at times t=[0.2s, 0.3s]:

```
psi_arr = Eq_instance.rmid2rho('psinorm', 0.6, [0.2, 0.3])
```

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

```
psi_arr = Eq_instance.rmid2rho('psinorm', [0.6, 0.5], [0.2, 0.3], each_t=False)
```

roa2rmid(*roa*, *t*, *each_t=True*, *return_t=False*, *time_idxs=None*, *length_unit=1*)

Convert the passed (r/a, t) coordinates into Rmid.

Parameters

- **roa** (*Array-like or scalar float*) – Values of the normalized minor radius to map to Rmid.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *roa*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *roa*.

Keyword Arguments

- **each_t** (*Boolean*) – When True, the elements in *roa* 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 *roa* or be a scalar. Default is True (evaluate ALL *roa* at EACH element in *t*).
- **length_unit** (*String or 1*) – Length unit that *Rmid* 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 (use meters).
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

Rmid or (*Rmid*, *time_idxs*)

- **Rmid** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *Rmid*) - 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 R_mid value at r/a=0.6, t=0.26s:

```
R_mid_val = Eq_instance.roa2rmid(0.6, 0.26)
```

Find R_mid values at r/a points 0.6 and 0.8 at the single time t=0.26s.:

```
R_mid_arr = Eq_instance.roa2rmid([0.6, 0.8], 0.26)
```

Find R_mid values at r/a of 0.6 at times t=[0.2s, 0.3s]:

```
R_mid_arr = Eq_instance.roa2rmid(0.6, [0.2, 0.3])
```

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

```
R_mid_arr = Eq_instance.roa2rmid([0.6, 0.5], [0.2, 0.3], each_t=False)
```

roa2psinorm(**args*, ***kwargs*)

Convert the passed (r/a, t) coordinates into psinorm.

Parameters

- **roa** (*Array-like or scalar float*) – Values of the normalized minor radius to map to psinorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *roa*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *roa*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of psinorm. 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.
- **each_t** (*Boolean*) – When True, the elements in *roa* 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 *roa* or be a scalar. Default is True (evaluate ALL *roa* at EACH element in *t*).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from Rmid to psinorm and psinorm to volnorm or 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

psinorm or (*psinorm*, *time_idxs*)

- **psinorm** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *psinorm*) - 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 psinorm value at r/a=0.6, t=0.26s:

```
psinorm_val = Eq_instance.roa2psinorm(0.6, 0.26)
```

Find psinorm values at r/a points 0.6 and 0.8 at the single time t=0.26s.:

```
psinorm_arr = Eq_instance.roa2psinorm([0.6, 0.8], 0.26)
```

Find psinorm values at r/a of 0.6 at times t=[0.2s, 0.3s]:

```
psinorm_arr = Eq_instance.roa2psinorm(0.6, [0.2, 0.3])
```

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

```
psinorm_arr = Eq_instance.roa2psinorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

roa2phinorm(**args*, ***kwargs*)

Convert the passed (r/a, t) coordinates into phinorm.

Parameters

- **roa** (*Array-like or scalar float*) – Values of the normalized minor radius to map to phinorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *roa*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *roa*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of phinorm. 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.
- **each_t** (*Boolean*) – When True, the elements in *roa* 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 *roa* or be a scalar. Default is True (evaluate ALL *roa* at EACH element in *t*).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from Rmid to psinorm and psinorm to volnorm or 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

phinorm or (*phinorm*, *time_idxs*)

- **phinorm** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *phinorm*) - 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 phinorm value at r/a=0.6, t=0.26s:

```
phinorm_val = Eq_instance.roa2phinorm(0.6, 0.26)
```

Find phinorm values at r/a points 0.6 and 0.8 at the single time t=0.26s.:

```
phinorm_arr = Eq_instance.roa2phinorm([0.6, 0.8], 0.26)
```

Find phinorm values at r/a of 0.6 at times t=[0.2s, 0.3s]:

```
phinorm_arr = Eq_instance.roa2phinorm(0.6, [0.2, 0.3])
```

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

```
phinorm_arr = Eq_instance.roa2phinorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

roa2volnorm(**args*, ***kwargs*)

Convert the passed (r/a, t) coordinates into volnorm.

Parameters

- **roa** (*Array-like or scalar float*) – Values of the normalized minor radius to map to volnorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *roa*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *roa*.

Keyword Arguments

- **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.
- **each_t** (*Boolean*) – When True, the elements in *roa* 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 *roa* or be a scalar. Default is True (evaluate ALL *roa* at EACH element in *t*).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from Rmid to psinorm and psinorm to volnorm or 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

volnorm or (*volnorm*, *time_idxs*)

- **volnorm** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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/a=0.6, t=0.26s:

```
volnorm_val = Eq_instance.roa2volnorm(0.6, 0.26)
```

Find volnorm values at r/a points 0.6 and 0.8 at the single time t=0.26s.:

```
volnorm_arr = Eq_instance.roa2volnorm([0.6, 0.8], 0.26)
```

Find volnorm values at r/a of 0.6 at times t=[0.2s, 0.3s]:

```
volnorm_arr = Eq_instance.roa2volnorm(0.6, [0.2, 0.3])
```

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

```
volnorm_arr = Eq_instance.roa2volnorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

roa2rho(*method*, **args*, ***kwargs*)
Convert the passed (r/a, t) coordinates into one of several coordinates.

Parameters

- **method** (*String*) – Indicates which coordinates to convert to. Valid options are:

psinorm	Normalized poloidal flux
phinorm	Normalized toroidal flux
volnorm	Normalized volume
Rmid	Midplane major radius

  Additionally, each valid option may be prepended with 'sqrt' to specify the square root of the desired unit.

- **roa** (*Array-like or scalar float*) – Values of the normalized minor radius to map to rho.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *roa*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *roa*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of rho. 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.
- **each_t** (*Boolean*) – When True, the elements in *roa* 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 *roa* or be a scalar. Default is True (evaluate ALL *roa* at EACH element in *t*).
- **length_unit** (*String or 1*) – Length unit that *Rmid* 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 (use meters).

- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from Rmid to psinorm and psinorm to volnorm or 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

rho or (*rho*, *time_idxs*)

- **rho** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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.

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/a=0.6, t=0.26s:

```
psi_val = Eq_instance.roa2rho('psinorm', 0.6, 0.26)
```

Find psinorm values at r/a points 0.6 and 0.8 at the single time t=0.26s:

```
psi_arr = Eq_instance.roa2rho('psinorm', [0.6, 0.8], 0.26)
```

Find psinorm values at r/a of 0.6 at times t=[0.2s, 0.3s]:

```
psi_arr = Eq_instance.roa2rho('psinorm', 0.6, [0.2, 0.3])
```

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

```
psi_arr = Eq_instance.roa2rho('psinorm', [0.6, 0.5], [0.2, 0.3], each_t=False)
```

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

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

Parameters

- **psi_norm** (*Array-like or scalar float*) – Values of the normalized poloidal flux to map to Rmid.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *psi_norm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *psi_norm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of Rmid. 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.
- **each_t** (*Boolean*) – When True, the elements in *psi_norm* 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 *psi_norm* or be a scalar. Default is True (evaluate ALL *psi_norm* at EACH element in *t*).
- **rho** (*Boolean*) – Set to True to return r/a (normalized minor radius) instead of Rmid. Default is False (return major radius, Rmid).
- **length_unit** (*String or 1*) – Length unit that *Rmid* 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 (use meters).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from psinorm to Rmid. 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

Rmid or (*Rmid*, *time_idxs*)

- **Rmid** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *Rmid*) - 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 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:

```
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)
```

`psinorm2roa`(*psi_norm*, *t*, ***kwargs*)

Calculates the normalized minor radius 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 r/a.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *psi_norm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *psi_norm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of 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.
- **each_t** (*Boolean*) – When True, the elements in *psi_norm* 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 *psi_norm* or be a scalar. Default is True (evaluate ALL *psi_norm* at EACH element in *t*).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from psinorm to Rmid. 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

roa or (*roa*, *time_idxs*)

- **roa** (*Array or scalar float*) - Normalized midplane minor radius. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *roa*) - 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 r/a value for psinorm=0.7, t=0.26s:

```
roa_val = Eq_instance.psinorm2roa(0.7, 0.26)
```

Find r/a values at psi_norm values of 0.5 and 0.7 at the single time t=0.26s:

```
roa_arr = Eq_instance.psinorm2roa([0.5, 0.7], 0.26)
```

Find r/a values at psi_norm=0.5 at times t=[0.2s, 0.3s]:

```
roa_arr = Eq_instance.psinorm2roa(0.5, [0.2, 0.3])
```

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

```
roa_arr = Eq_instance.psinorm2roa([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.

Parameters

- **psi_norm** (*Array-like or scalar float*) – Values of the normalized poloidal flux to map to volnorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *psi_norm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *psi_norm*.

Keyword Arguments

- **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.
- **each_t** (*Boolean*) – When True, the elements in *psi_norm* 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 *psi_norm* or be a scalar. Default is True (evaluate ALL *psi_norm* at EACH element in *t*).
- **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.

- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

volnorm or (*volnorm*, *time_idxs*)

- **volnorm** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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 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:

```
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.

Parameters

- **psi_norm** (*Array-like or scalar float*) – Values of the normalized poloidal flux to map to phinorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *psi_norm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *psi_norm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of phinorm. 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.
- **each_t** (*Boolean*) – When True, the elements in *psi_norm* 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 *psi_norm* or be a scalar. Default is True (evaluate ALL *psi_norm* at EACH element in *t*).

- **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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

phinorm or (*phinorm*, *time_idxs*)

- **phinorm** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *phinorm*) - 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 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:

```
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)
```

psinorm2rho(*method*, **args*, ***kwargs*)

Convert the passed (psinorm, t) coordinates into one of several coordinates.

Parameters

- **method** (*String*) – Indicates which coordinates to convert to. Valid options are:

phinorm	Normalized toroidal flux
volnorm	Normalized volume
Rmid	Midplane major radius
r/a	Normalized minor radius

  Additionally, each valid option may be prepended with 'sqrt' to specify the square root of the desired unit.

- **psi_norm** (*Array-like or scalar float*) – Values of the normalized poloidal flux to map to rho.

- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *psi_norm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *psi_norm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of rho. 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.
- **each_t** (*Boolean*) – When True, the elements in *psi_norm* 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 *psi_norm* or be a scalar. Default is True (evaluate ALL *psi_norm* at EACH element in *t*).
- **rho** (*Boolean*) – Set to True to return r/a (normalized minor radius) instead of Rmid. Default is False (return major radius, Rmid).
- **length_unit** (*String or 1*) – Length unit that *Rmid* 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 (use meters).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from psinorm to Rmid/phinorm/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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

rho or (*rho*, *time_idxs*)

- **rho** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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 phinorm value at psinorm=0.6, t=0.26s:

```
phi_val = Eq_instance.psinorm2rho('phinorm', 0.6, 0.26)
```

Find phinorm values at phinorm of 0.6 and 0.8 at the single time t=0.26s:

```
phi_arr = Eq_instance.psinorm2rho('phinorm', [0.6, 0.8], 0.26)
```

Find phinorm values at psinorm of 0.6 at times t=[0.2s, 0.3s]:

```
phi_arr = Eq_instance.psinorm2rho('phinorm', 0.6, [0.2, 0.3])
```

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

```
phi_arr = Eq_instance.psinorm2rho('phinorm', [0.6, 0.5], [0.2, 0.3], each_t=False)
```

`phinorm2psinorm`(*phinorm*, *t*, ***kwargs*)

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

Parameters

- **phinorm** (*Array-like or scalar float*) – Values of the normalized toroidal flux to map to psinorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *phinorm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *phinorm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of psinorm. 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.
- **each_t** (*Boolean*) – When True, the elements in *phinorm* 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 *phinorm* or be a scalar. Default is True (evaluate ALL *phinorm* at EACH element in *t*).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from phinorm to psinorm. 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

psinorm or (*psinorm*, *time_idxs*)

- **psinorm** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *psinorm*) - 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 psinorm value for phinorm=0.7, t=0.26s:

```
psinorm_val = Eq_instance.phinorm2psinorm(0.7, 0.26)
```

Find psinorm values at phinorm values of 0.5 and 0.7 at the single time t=0.26s:

```
psinorm_arr = Eq_instance.phinorm2psinorm([0.5, 0.7], 0.26)
```

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

```
psinorm_arr = Eq_instance.phinorm2psinorm(0.5, [0.2, 0.3])
```

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

```
psinorm_arr = Eq_instance.phinorm2psinorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

phinorm2volnorm(**args*, ***kwargs*)

Calculates the normalized flux surface volume corresponding to the passed phinorm (normalized toroidal flux) values.

Parameters

- **phinorm** (*Array-like or scalar float*) – Values of the normalized toroidal flux to map to volnorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *phinorm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *phinorm*.

Keyword Arguments

- **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.
- **each_t** (*Boolean*) – When True, the elements in *phinorm* 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 *phinorm* or be a scalar. Default is True (evaluate ALL *phinorm* at EACH element in *t*).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from phinorm to psinorm and 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.

- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

volnorm or (*volnorm*, *time_idxs*)

- **volnorm** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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 for phinorm=0.7, t=0.26s:

```
volnorm_val = Eq_instance.phinorm2volnorm(0.7, 0.26)
```

Find volnorm values at phinorm values of 0.5 and 0.7 at the single time t=0.26s:

```
volnorm_arr = Eq_instance.phinorm2volnorm([0.5, 0.7], 0.26)
```

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

```
volnorm_arr = Eq_instance.phinorm2volnorm(0.5, [0.2, 0.3])
```

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

```
volnorm_arr = Eq_instance.phinorm2volnorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

phinorm2rmid(**args*, ***kwargs*)

Calculates the mapped outboard midplane major radius corresponding to the passed phinorm (normalized toroidal flux) values.

Parameters

- **phinorm** (*Array-like or scalar float*) – Values of the normalized toroidal flux to map to Rmid.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *phinorm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *phinorm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of Rmid. 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.
- **each_t** (*Boolean*) – When True, the elements in *phinorm* 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 *phinorm* or be a scalar. Default is True (evaluate ALL *phinorm* at EACH element in *t*).
- **rho** (*Boolean*) – Set to True to return r/a (normalized minor radius) instead of Rmid. Default is False (return major radius, Rmid).

- **length_unit** (*String or 1*) – Length unit that *Rmid* 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 (use meters).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from phinorm to psinorm and psinorm to Rmid. 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

Rmid or (*Rmid*, *time_idxs*)

- **Rmid** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *Rmid*) - 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 Rmid value for phinorm=0.7, t=0.26s:

```
Rmid_val = Eq_instance.phinorm2rmid(0.7, 0.26)
```

Find Rmid values at phinorm values of 0.5 and 0.7 at the single time t=0.26s:

```
Rmid_arr = Eq_instance.phinorm2rmid([0.5, 0.7], 0.26)
```

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

```
Rmid_arr = Eq_instance.phinorm2rmid(0.5, [0.2, 0.3])
```

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

```
Rmid_arr = Eq_instance.phinorm2rmid([0.6, 0.5], [0.2, 0.3], each_t=False)
```

phinorm2roa(*phi_norm*, *t*, ***kwargs*)

Calculates the normalized minor radius corresponding to the passed phinorm (normalized toroidal flux) values.

Parameters

- **phinorm** (*Array-like or scalar float*) – Values of the normalized toroidal flux to map to r/a.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *phinorm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *phinorm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of 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.
- **each_t** (*Boolean*) – When True, the elements in *phinorm* 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 *phinorm* or be a scalar. Default is True (evaluate ALL *phinorm* at EACH element in *t*).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from phinorm to psinorm and psinorm to Rmid. 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

roa or (*roa*, *time_idxs*)

- **roa** (*Array or scalar float*) - Normalized midplane minor radius. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *roa*) - 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 r/a value for phinorm=0.7, t=0.26s:

```
roa_val = Eq_instance.phinorm2roa(0.7, 0.26)
```

Find r/a values at phinorm values of 0.5 and 0.7 at the single time t=0.26s:

```
roa_arr = Eq_instance.phinorm2roa([0.5, 0.7], 0.26)
```

Find r/a values at phinorm=0.5 at times t=[0.2s, 0.3s]:

```
roa_arr = Eq_instance.phinorm2roa(0.5, [0.2, 0.3])
```

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

```
roa_arr = Eq_instance.phinorm2roa([0.6, 0.5], [0.2, 0.3], each_t=False)
```

phinorm2rho(*method*, **args*, ***kwargs*)

Convert the passed (phinorm, t) coordinates into one of several coordinates.

Parameters

- **method** (*String*) – Indicates which coordinates to convert to. Valid options are:

psinorm	Normalized poloidal flux
volnorm	Normalized volume
Rmid	Midplane major radius
r/a	Normalized minor radius

  Additionally, each valid option may be prepended with 'sqrt' to specify the square root of the desired unit.

- **phinorm** (*Array-like or scalar float*) – Values of the normalized toroidal flux to map to rho.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *phinorm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *phinorm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of rho. 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.
- **each_t** (*Boolean*) – When True, the elements in *phinorm* 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 *phinorm* or be a scalar. Default is True (evaluate ALL *phinorm* at EACH element in *t*).
- **rho** (*Boolean*) – Set to True to return r/a (normalized minor radius) instead of Rmid. Default is False (return major radius, Rmid).
- **length_unit** (*String or 1*) – Length unit that *Rmid* 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 (use meters).

- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from psinorm to Rmid/phinorm/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.

- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

rho or (*rho*, *time_idxs*)

- **rho** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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 phinorm=0.6, t=0.26s:

```
psi_val = Eq_instance.phinorm2rho('psinorm', 0.6, 0.26)
```

Find psinorm values at phinorm of 0.6 and 0.8 at the single time t=0.26s:

```
psi_arr = Eq_instance.phinorm2rho('psinorm', [0.6, 0.8], 0.26)
```

Find psinorm values at phinorm of 0.6 at times t=[0.2s, 0.3s]:

```
psi_arr = Eq_instance.phinorm2rho('psinorm', 0.6, [0.2, 0.3])
```

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

```
psi_arr = Eq_instance.phinorm2rho('psinorm', [0.6, 0.5], [0.2, 0.3], each_t=False)
```

volnorm2psinorm(**args*, ***kwargs*)

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

Parameters

- **volnorm** (*Array-like or scalar float*) – Values of the normalized flux surface volume to map to psinorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *volnorm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *volnorm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of psinorm. 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.

- **each_t** (*Boolean*) – When True, the elements in *volnorm* 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 *volnorm* or be a scalar. Default is True (evaluate ALL *volnorm* at EACH element in *t*).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from volnorm to psinorm. 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

psinorm or (*psinorm*, *time_idxs*)

- **psinorm** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *psinorm*) - 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 psinorm value for volnorm=0.7, t=0.26s:

```
psinorm_val = Eq_instance.volnorm2psinorm(0.7, 0.26)
```

Find psinorm values at volnorm values of 0.5 and 0.7 at the single time t=0.26s:

```
psinorm_arr = Eq_instance.volnorm2psinorm([0.5, 0.7], 0.26)
```

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

```
psinorm_arr = Eq_instance.volnorm2psinorm(0.5, [0.2, 0.3])
```

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

```
psinorm_arr = Eq_instance.volnorm2psinorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

volnorm2phinorm(**args*, ***kwargs*)

Calculates the normalized toroidal flux corresponding to the passed volnorm (normalized flux surface volume) values.

Parameters

- **volnorm** (*Array-like or scalar float*) – Values of the normalized flux surface volume to map to phinorm.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *volnorm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *volnorm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of phinorm. 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.
- **each_t** (*Boolean*) – When True, the elements in *volnorm* 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 *volnorm* or be a scalar. Default is True (evaluate ALL *volnorm* at EACH element in *t*).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from volnorm to psinorm and 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

phinorm or (*phinorm*, *time_idxs*)

- **phinorm** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *phinorm*) - 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 phinorm value for volnorm=0.7, t=0.26s:

```
phinorm_val = Eq_instance.volnorm2phinorm(0.7, 0.26)
```

Find phinorm values at volnorm values of 0.5 and 0.7 at the single time t=0.26s:

```
phinorm_arr = Eq_instance.volnorm2phinorm([0.5, 0.7], 0.26)
```

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

```
phinorm_arr = Eq_instance.volnorm2phinorm(0.5, [0.2, 0.3])
```

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

```
phinorm_arr = Eq_instance.volnorm2phinorm([0.6, 0.5], [0.2, 0.3], each_t=False)
```

volnorm2rmid(**args*, ***kwargs*)

Calculates the mapped outboard midplane major radius corresponding to the passed volnorm (normalized flux surface volume) values.

Parameters

- **volnorm** (*Array-like or scalar float*) – Values of the normalized flux surface volume to map to Rmid.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *volnorm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *volnorm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of Rmid. 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.
- **each_t** (*Boolean*) – When True, the elements in *volnorm* 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 *volnorm* or be a scalar. Default is True (evaluate ALL *volnorm* at EACH element in *t*).
- **rho** (*Boolean*) – Set to True to return r/a (normalized minor radius) instead of Rmid. Default is False (return major radius, Rmid).
- **length_unit** (*String or 1*) – Length unit that *Rmid* 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 (use meters).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from volnorm to psinorm and psinorm to Rmid. 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

Rmid or (*Rmid*, *time_idxs*)

- **Rmid** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *Rmid*) - 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 Rmid value for volnorm=0.7, t=0.26s:

```
Rmid_val = Eq_instance.volnorm2rmid(0.7, 0.26)
```

Find Rmid values at volnorm values of 0.5 and 0.7 at the single time t=0.26s:

```
Rmid_arr = Eq_instance.volnorm2rmid([0.5, 0.7], 0.26)
```

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

```
Rmid_arr = Eq_instance.volnorm2rmid(0.5, [0.2, 0.3])
```

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

```
Rmid_arr = Eq_instance.volnorm2rmid([0.6, 0.5], [0.2, 0.3], each_t=False)
```

volnorm2roa(**args*, ***kwargs*)

Calculates the normalized minor radius corresponding to the passed volnorm (normalized flux surface volume) values.

Parameters

- **volnorm** (*Array-like or scalar float*) – Values of the normalized flux surface volume to map to r/a.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *volnorm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *volnorm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of 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.
- **each_t** (*Boolean*) – When True, the elements in *volnorm* 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 *volnorm* or be a scalar. Default is True (evaluate ALL *volnorm* at EACH element in *t*).
- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from volnorm to psinorm and psinorm to Rmid. 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.
- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

roa or (*roa*, *time_idxs*)

- **roa** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **time_idxs** (Array with same shape as *roa*) - 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 r/a value for volnorm=0.7, t=0.26s:

```
roa_val = Eq_instance.volnorm2roa(0.7, 0.26)
```

Find r/a values at volnorm values of 0.5 and 0.7 at the single time t=0.26s:

```
roa_arr = Eq_instance.volnorm2roa([0.5, 0.7], 0.26)
```

Find r/a values at volnorm=0.5 at times t=[0.2s, 0.3s]:

```
roa_arr = Eq_instance.volnorm2roa(0.5, [0.2, 0.3])
```

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

```
roa_arr = Eq_instance.volnorm2roa([0.6, 0.5], [0.2, 0.3], each_t=False)
```

volnorm2rho(*method*, **args*, ***kwargs*)

Convert the passed (volnorm, t) coordinates into one of several coordinates.

Parameters

- **method** (*String*) – Indicates which coordinates to convert to. Valid options are:

psinorm	Normalized poloidal flux
phinorm	Normalized toroidal flux
Rmid	Midplane major radius
r/a	Normalized minor radius

  Additionally, each valid option may be prepended with 'sqrt' to specify the square root of the desired unit.

- **volnorm** (*Array-like or scalar float*) – Values of the normalized flux surface volume to map to rho.
- **t** (*Array-like or scalar float*) – Times to perform the conversion at. If *t* is a single value, it is used for all of the elements of *volnorm*. If the *each_t* keyword is True, then *t* must be scalar or have exactly one dimension. If the *each_t* keyword is False, *t* must have the same shape as *volnorm*.

Keyword Arguments

- **sqrt** (*Boolean*) – Set to True to return the square root of rho. 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.
- **each_t** (*Boolean*) – When True, the elements in *volnorm* 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 *volnorm* or be a scalar. Default is True (evaluate ALL *volnorm* at EACH element in *t*).
- **rho** (*Boolean*) – Set to True to return r/a (normalized minor radius) instead of Rmid. Default is False (return major radius, Rmid).

- **length_unit** (*String or 1*) – Length unit that *Rmid* 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 (use meters).

- **kind** (*String or non-negative int*) – Specifies the type of interpolation to be performed in getting from volnorm to Rmid/phinorm/psinorm. 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.

- **return_t** (*Boolean*) – Set to True to return a tuple of (*rho*, *time_idxs*), where *time_idxs* is the array of time indices actually used in evaluating *rho* with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return *rho*).

Returns

rho or (*rho*, *time_idxs*)

- **rho** (*Array or scalar float*) - The converted coordinates. If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array is returned.
- **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 volnorm=0.6, t=0.26s:

```
psi_val = Eq_instance.volnorm2rho('psinorm', 0.6, 0.26)
```

Find psinorm values at volnorm of 0.6 and 0.8 at the single time t=0.26s:

```
psi_arr = Eq_instance.volnorm2rho('psinorm', [0.6, 0.8], 0.26)
```

Find psinorm values at volnorm of 0.6 at times t=[0.2s, 0.3s]:

```
psi_arr = Eq_instance.volnorm2rho('psinorm', 0.6, [0.2, 0.3])
```

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

```
psi_arr = Eq_instance.volnorm2rho('psinorm', [0.6, 0.5], [0.2, 0.3], each_t=False)
```

getMagRSpline(*length_unit=1*, *kind='nearest'*)

Gets the univariate spline to interpolate R_mag as a function of time.

Only used if the instance was created with keyword tspline=True.

Keyword Arguments

- **length_unit** (*String or 1*) –

  Length unit that R_mag 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_out returned in meters).

- **kind** (*String or non-negative int*) –

  Specifies the type of interpolation to be performed in getting from t to R_mag. 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 scipy.interpolate.interp1d to convert from t to R_mid.

getMagZSpline(*length_unit=1*, *kind='nearest'*)

Gets the univariate spline to interpolate Z_mag as a function of time.

Generated for completeness of the core position calculation when using tspline = True

Keyword Arguments

- **length_unit** (*String or 1*) –

  Length unit that R_mag 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_out returned in meters).

- **kind** (*String or non-negative int*) –

  Specifies the type of interpolation to be performed in getting from t to R_mag. 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 scipy.interpolate.interp1d to convert from t to R_mid.

getRmidOutSpline(*length_unit=1*, *kind='nearest'*)

Gets the univariate spline to interpolate R_mid_out as a function of time.

Generated for completeness of the core position calculation when using tspline = True

Keyword Arguments

- **length_unit** (*String or 1*) –

  Length unit that R_mag 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_out returned in meters).

- **kind** (*String or non-negative int*) –

  Specifies the type of interpolation to be performed in getting from t to R_mag. 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 scipy.interpolate.interp1d to convert from t to R_mid.

`getAOutSpline`(*length_unit=1*, *kind='nearest'*)

Gets the univariate spline to interpolate a_out as a function of time.

Keyword Arguments

- **length_unit** (*String or 1*) –

  Length unit that a_out 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 (a_out returned in meters).

- **kind** (*String or non-negative int*) –

  Specifies the type of interpolation to be performed in getting from t to a_out. 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 scipy.interpolate.interp1d to convert from t to a_out.

`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]

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

Returns F=RB_{Phi}(Psi), often calculated for grad-shafranov solutions [psi,t]

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

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

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

Returns FF' function used for grad-shafranov solutions [psi,t]

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

Returns plasma pressure gradient as a function of psi [psi,t]

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.

gfile(*time=None*, *nw=None*, *nh=None*, *shot=None*, *name=None*, *tunit='ms'*, *title='EQTOOLS'*, *nbbbs=100*)

Generates an EFIT gfile with gfile naming convention

Keyword Arguments

- **time** (*scalar float*) – Time of equilibrium to generate the gfile from. This will use the specified spline functionality to do so. Allows for it to be unspecified for single-time-frame equilibria.
- **nw** (*scalar integer*) – Number of points in R. R is the major radius, and describes the 'width' of the gfile.
- **nh** (*scalar integer*) – Number of points in Z. In cylindrical coordinates Z is the height, and nh describes the 'height' of the gfile.
- **shot** (*scalar integer*) – The shot numer of the equilibrium. Used to help generate the gfile name if unspecified.
- **name** (*String*) – Name of the gfile. If unspecified, will follow standard gfile naming convention (g+shot.time) under current python operating directory. This allows for it to be saved in other directories, etc.
- **tunit** (*String*) – Specified unit for tin. It can only be 'ms' for milliseconds or 's' for seconds.
- **title** (*String*) – Title of the gfile on the first line. Name cannot exceed 10 digits. This is so that the style of the first line is preserved.
- **nbbbs** (*scalar integer*) – Number of points to define the plasma seperatrix within the gfile. The points are defined equally spaced in angle about the plasma center. This will cause the x-point to be poorly defined.

Raises `ValueError` – If title is longer than 10 characters.

Examples

All assume that *Eq_instance* is a valid instance of the appropriate extension of the `Equilibrium` abstract class (example shot number of 1001).

Generate a gfile at t=0.26s, output of g1001.26:

```
Eq_instance.gfile(.26)
```

`plotFlux` (*fill=True*, *mask=True*)

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.

3.1.8 eqtools.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** (*Integer*) – Shot index.
- **time** (*Integer*) – 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.

`aaaaa`()

Foo the bar baz

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

Converts passed, R,Z arrays to psi values.

Wrapper for `Equilibrium.rz2psi()` removing 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)*).

Examples

All assume that Eq_instance is a valid instance EqdskReader:

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

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

Find psi 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.rz2psi([0.6, 0.8], [0, 0])
```

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

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

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. 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. 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, with the same shape as *R* and *Z*. 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), using

$$
exttphi = \int q(\psi)\,d\psi
$$
$$
exttphi\_norm =
$$

rac{phi}{phi(a)}

Wrapper for Equilibrium.rz2phinorm masking out timebase dependence.

Args:

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 Args:

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

Keyword Arguments **mask** – Boolean. Default False. 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.

getF()
returns F=RB_{Phi}(Psi), often calculated for grad-shafranov solutions [psi,t]

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

getFFPrime()
returns FF' function used for grad-shafranov solutions [psi,t]

getPPrime()
returns plasma pressure gradient as a function of psi [psi,t]

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().

gfile(*time=None*, *nw=None*, *nh=None*, *shot=None*, *name=None*, *tunit='ms'*, *title='EQTOOLS'*, *nbbbs=100*)

Generates an EFIT gfile with gfile naming convention

Keyword Arguments

- **time** (*scalar float*) – Time of equilibrium to generate the gfile from. This will use the specified spline functionality to do so. Allows for it to be unspecified for single-time-frame equilibria.
- **nw** (*scalar integer*) – Number of points in R. R is the major radius, and describes the 'width' of the gfile.
- **nh** (*scalar integer*) – Number of points in Z. In cylindrical coordinates Z is the height, and nh describes the 'height' of the gfile.
- **shot** (*scalar integer*) – The shot numer of the equilibrium. Used to help generate the gfile name if unspecified.
- **name** (*String*) – Name of the gfile. If unspecified, will follow standard gfile naming convention (g+shot.time) under current python operating directory. This allows for it to be saved in other directories, etc.
- **tunit** (*String*) – Specified unit for tin. It can only be 'ms' for milliseconds or 's' for seconds.
- **title** (*String*) – Title of the gfile on the first line. Name cannot exceed 10 digits. This is so that the style of the first line is preserved.
- **nbbbs** (*scalar integer*) – Number of points to define the plasma seperatrix within the gfile. The points are defined equally spaced in angle about the plasma center. This will cause the x-point to be poorly defined.

Raises `ValueError` – If title is longer than 10 characters.

Examples

All assume that *Eq_instance* is a valid instance of the appropriate extension of the `Equilibrium` abstract class (example shot number of 1001).

Generate a gfile (time at t=.26s) output of g1001.26:

```
Eq_instance.gfile()
```

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.

3.1.9 eqtools.filewriter module

`eqtools.filewriter.`**`gfile`**(*obj*, *tin*, *nw=None*, *nh=None*, *shot=None*, *name=None*, *tunit='ms'*, *title='EQTOOLS'*, *nbbbs=100*)

Generates an EFIT gfile with gfile naming convention

Parameters

- **obj** (*eqtools Equilibrium Object*) – Object which describes the tokamak This functionality is dependent on matplotlib, and is not not retained in core.py for this reason. It is a hidden function which takes an arbitrary equilibrium object and generates a gfile.
- **tin** (*scalar float*) – Time of equilibrium to generate the gfile from. This will use the specified spline functionality to do so.

Keyword Arguments

- **nw** (*scalar integer*) – Number of points in R. R is the major radius, and describes the 'width' of the gfile.
- **nh** (*scalar integer*) – Number of points in Z. In cylindrical coordinates Z is the height, and nh describes the 'height' of the gfile.
- **shot** (*scalar integer*) – The shot numer of the equilibrium. Used to help generate the gfile name if unspecified.
- **name** (*String*) – Name of the gfile. If unspecified, will follow standard gfile naming convention (g+shot.time) under current python operating directory. This allows for it to be saved in other directories, etc.
- **tunit** (*String*) – Specified unit for tin. It can only be 'ms' for milliseconds or 's' for seconds.
- **title** (*String*) – Title of the gfile on the first line. Name cannot exceed 10 digits. This is so that the style of the first line is preserved.
- **nbbbs** (*scalar integer*) – Number of points to define the plasma seperatrix within the gfile. The points are defined equally spaced in angle about the plasma center. This will cause the x-point to be poorly defined.

Raises `ValueError` – If title is longer than 10 characters.

Examples

All assume that *Eq_instance* is a valid instance of the appropriate extension of the `Equilibrium` abstract class (example shot number of 1001).

Generate a gfile at t=0.26s, output of g1001.26:

```
gfile(Eq_instance,.26)
```

3.1.10 eqtools.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.

3.1.11 eqtools.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. The ordering of f[z,y,x] is extremely important for the proper evaluation of the spline. It assumes that f is in C order.

Create a new Spline instance.

Parameters

- **z** (*1-dimensional float array*) – Values of the positions of the 1st Dimension of f. Must be monotonic without duplicates.
- **y** (*1-dimensional float array*) – Values of the positions of the 2nd dimension of f. Must be monotonic without duplicates.

- **x** (*1-dimensional float array*) – 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 hamper performance and 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

All assume that *x*, *y*, *z*, and *f* are valid instances of the appropriate numpy arrays which take independent variables x,y,z and create numpy array f. *x1*, *y1*, and *z1* are numpy arrays which data f is to be interpolated.

Generate a Trispline instance map with data x, y, z and f:

```
map = Spline(z, y, x, f)
```

Evaluate Trispline instance map at x1, y1, z1:

```
output = map.ev(z1, y1, x1)
```

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

evaluates tricubic spline at point (x1,y1,z1) which is f[z1,y1,x1].

Parameters

- **z1** (*scalar float or 1-dimensional float*) – Position in z dimension. This is the first dimension of 3d-valued grid.
- **y1** (*scalar float or 1-dimensional float*) – Position in y dimension. This is the second dimension of 3d-valued grid.
- **x1** (*scalar float or 1-dimensional float*) – Position in x dimension. This is the third dimension of 3d-valued grid.

Returns

val

- **val** (*array or scalar float*) - 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 scipy.interpolate.RectBivariateSpline with a proper bound checker and value filler such that it will not fail in use for EqTools

Can be used for both smoothing and interpolating data.

Parameters

- **x** (*1-dimensional float array*) –

  1-D array of coordinates in monotonically increasing order.

- **y** (*1-dimensional float array*) –

  1-D array of coordinates in monotonically increasing order.

- **z** (*2-dimensional float array*) –

  2-D array of data with shape (x.size,y.size).

Keyword Arguments

- **bbox** (*1-dimensional float*) – Sequence of length 4 specifying the boundary of the rectangular approximation domain. By default, `bbox=[min(x,tx),max(x,tx), min(y,ty),max(y,ty)]`.
- **kx** (*integer*) – Degrees of the bivariate spline. Default is 3.
- **ky** (*integer*) – Degrees of the bivariate spline. Default is 3.
- **s** (*float*) – Positive smoothing factor defined for estimation condition, `sum((w[i]*(z[i]-s(x[i], y[i])))**2, axis=0) <= s` Default is `s=0`, which is for interpolation.

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

3.1.12 Module contents

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

CHAPTER

FOUR

INDICES AND TABLES

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