# eqtools Documentation 

Release 1.3.2

Mark Chilenski, Ian Faust and John Walk

## Contents

1 Overview ..... 3
2 Installation ..... 5
3 Tutorial: Performing Coordinate Transforms on Alcator C-Mod Data ..... 7
4 Package Reference ..... 9
4.1 eqtools package ..... 9
4.1.1 Submodules ..... 9
4.1.2 eqtools.AUGData module ..... 9
4.1.3 eqtools.CModEFIT module ..... 22
4.1.4 eqtools.D3DEFIT module ..... 24
4.1.5 eqtools.EFIT module ..... 26
4.1.6 eqtools.FromArrays module ..... 36
4.1.7 eqtools.NSTXEFIT module ..... 37
4.1.8 eqtools.TCVLIUQE module ..... 40
4.1.9 eqtools.afilereader module ..... 47
4.1.10 eqtools.core module ..... 47
4.1.11 eqtools.eqdskreader module ..... 166
4.1.12 eqtools.filewriter module ..... 184
4.1.13 eqtools.pfilereader module ..... 184
4.1.14 eqtools.trispline module ..... 185
4.1.15 Module contents ..... 188
5 Indices and tables ..... 189
Python Module Index ..... 191
Index ..... 193

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

## CHAPTER 1

## 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 2

## Installation

The easiest way to install the latest release version is with pip:
pip install eqtools

To install from source, uncompress the source files and, from the directory containing setup.py, run the following command:

```
python setup.py install
```

Or, to build in place, run:

```
python setup.py build_ext --inplace
```


## 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 $\mathrm{t}=1.0 \mathrm{~s}$. 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 ( $\mathrm{psinorm}=0.3, \mathrm{t}=1.0$ ) and ( $\mathrm{psinorm}=0.35, \mathrm{t}=1.1$ ).
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 4

## Package Reference

## 4.1 eqtools package

### 4.1.1 Submodules

### 4.1.2 eqtools.AUGData module

This module provides classes inheriting eqtools.Equilibrium for working with ASDEX Upgrade experimental data.
class eqtools.AUGData.AUGDDData (shot, shotfile='EQH', edition=0, shotfile $2=$ None, length_unit='m', tspline=False, monotonic=True, experiment= 'AUGD')
Bases: eqtools.core.Equilibrium
Inherits eqtools.Equilibrium class. Machine-specific data handling class for ASDEX Upgrade. Pulls AFS data from selected location and shotfile, stores as object attributes. Each data variable or set of variables is recovered with a corresponding getter method. Essential data for mapping are pulled on initialization (e.g. psirz grid). Additional data are pulled at the first request and stored for subsequent usage.

Intializes ASDEX Upgrade version of the Equilibrium object. Pulls data to storage in instance attributes. Core attributes are populated from the AFS data on initialization. Additional attributes are initialized as None, filled on the first request to the object.

Parameters shot (integer) - ASDEX Upgrade shot index.

## Keyword Arguments

- shotfile (string) - Optional input for alternate shotfile, defaults to 'EQH' (i.e., CLISTE results are in EQH,EQI with other reconstructions Available (FPP, EQE, ect.).
- edition (integer) - Describes the edition of the shotfile to be used
- shotfile2 (string) - Describes companion 0D equilibrium data, will automatically reference based off of shotfile, but can be manually specified for unique reconstructions, etc.
- 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- <br> fault' | whatever the default in the tree is (no conversion is performed, units may <br> 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).
- experiment - Used to describe the work space that the shotfile is located It defaults to 'AUGD' but can be set to other values
getInfo()
returns namedtuple of shot information


## Returns

namedtuple containing

| shot | ASDEX Upgrage shot index (long) |
| :--- | :--- |
| tree | shotfile (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 time base vector.
Returns [nt] array of time points.
Return type time (array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getFluxGrid()
returns flux grid.
Note that this method preserves whatever sign convention is used in AFS.
Returns [nt,nz,nr] array of (non-normalized) flux on grid.

Return type psiRZ (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getRGrid(length_unit=1)
returns R-axis.
Returns [nr] array of R-axis of flux grid.
Return type rGrid (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getZGrid(length_unit=1)
returns Z-axis.
Returns [nz] array of Z-axis of flux grid.
Return type zGrid (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getFluxAxis()
returns psi on magnetic axis.
Returns [nt] array of psi on magnetic axis.
Return type psiAxis (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getFluxLCFS()
returns psi at separatrix.
Returns [nt] array of psi at LCFS.
Return type psiLCFS (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getFluxVol (length_unit=3)
returns volume within flux surface.
Keyword Arguments length_unit (String or 3) - unit for plasma volume. Defaults to 3 , indicating default volumetric unit (typically $\mathrm{m}^{\wedge} 3$ ).

Returns [nt,npsi] array of volume within flux surface.
Return type fluxVol (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getVolLCFS (length_unit=3)
returns volume within LCFS.
Keyword Arguments length_unit (String or 3) - unit for LCFS volume. Defaults to 3 , denoting default volumetric unit (typically $\mathrm{m}^{\wedge} 3$ ).

Returns [nt] array of volume within LCFS.
Return type volLCFS (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getRmidPsi (length_unit=1)
returns maximum major radius of each flux surface.

Keyword Arguments length_unit (String or 1) - unit of Rmid. Defaults to 1, indicating the default parameter unit (typically m).
Returns [nt,npsi] array of maximum (outboard) major radius of flux surface psi.
Return type Rmid (Array)
Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getRLCFS (length_unit=1)
returns R-values of LCFS position.
Returns [nt,n] array of R of LCFS points.
Return type RLCFS (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getZLCFS (length_unit=1)
returns Z-values of LCFS position.
Returns [nt,n] array of Z of LCFS points.
Return type ZLCFS (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
remapLCFS (mask=False)
Overwrites RLCFS, ZLCFS values pulled 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 $\mathrm{psi}=\mathrm{psiLCFS}$.

## Raises

- NotImplementedError - if matplotlib. pyplot is not loaded.
- ValueError - if limiter outline is not available.
getF ()
returns $\mathrm{F}=\mathrm{RB} \_\{\mathrm{Phi}\}(\mathrm{Psi})$, often calculated for grad-shafranov solutions.
Returns [nt,npsi] array of $\mathrm{F}=\mathrm{RB}_{-}\{$Phi $\}$(Psi)
Return type F (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.

```
getFluxPres()
```

returns pressure at flux surface.
Returns [nt,npsi] array of pressure on flux surface psi.
Return type p (Array)
Raises ValueError - if module cannot retrieve data from AUG AFS system.
getFPrime()
returns $\mathrm{F}^{\prime}$, often calculated for grad-shafranov solutions.
Returns [nt,npsi] array of $\mathrm{F}=$ RB_\{Phi\}(Psi)
Return type F (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getFFPrime()
returns FF' function used for grad-shafranov solutions.
Returns [nt,npsi] array of FF' fromgrad-shafranov solution.
Return type FFprime (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getPPrime()
returns plasma pressure gradient as a function of psi.
Returns [nt,npsi] array of pressure gradient on flux surface psi from grad-shafranov solution.
Return type pprime (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getElongation()
returns LCFS elongation.
Returns [nt] array of LCFS elongation.
Return type kappa (Array)
Raises ValueError - if module cannot retrieve data from AFS.
getUpperTriangularity()
returns LCFS upper triangularity.
Returns [nt] array of LCFS upper triangularity.
Return type deltau (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getLowerTriangularity()
returns LCFS lower triangularity.
Returns [nt] array of LCFS lower triangularity.
Return type deltal (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getShaping()
pulls LCFS elongation and upper/lower triangularity.
Returns namedtuple containing (kappa, delta_u, delta_l)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getMagR(length_unit=1)
returns magnetic-axis major radius.
Returns [nt] array of major radius of magnetic axis.
Return type magR (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getMagZ (length_unit=1)
returns magnetic-axis Z .
Returns [nt] array of Z of magnetic axis.
Return type magZ (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getAreaLCFS (length_unit=2)
returns LCFS cross-sectional area.
Keyword Arguments length_unit (String or 2) - unit for LCFS area. Defaults to 2, denoting default areal unit (typically $\mathrm{m}^{\wedge} 2$ ).

Returns [nt] array of LCFS area.
Return type areaLCFS (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getAOut (length_unit=1)
returns outboard-midplane minor radius at LCFS.
Keyword Arguments length_unit (String or 1) - unit for minor radius. Defaults to 1 , denoting default length unit (typically m).
Returns [nt] array of LCFS outboard-midplane minor radius.
Return type aOut (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getRmidOut (length_unit=1)
returns outboard-midplane major radius.
Keyword Arguments length_unit (String or 1) - unit for major radius. Defaults to 1 , denoting default length unit (typically m).
Raises Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getGeometry (length_unit=None)
pulls dimensional geometry parameters.
Returns namedtuple containing (magR,magZ,areaLCFS,aOut,RmidOut)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.

```
getQProfile()
```

returns profile of safety factor q .
Returns [nt,npsi] array of q on flux surface psi.
Return type qpsi (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.

```
getQ0()
```

returns $q$ on magnetic axis,q0.
Returns [nt] array of $\mathrm{q}(\mathrm{psi}=0)$.
Return type q0 (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.

```
getQ95()
```

returns q at $95 \%$ flux surface.
Returns [nt] array of $\mathrm{q}(\mathrm{psi}=0.95)$.
Return type q95 (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.

```
getQLCFS()
```

returns q on LCFS (interpolated).

Raises Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getQ1Surf (length_unit=1)
returns outboard-midplane minor radius of $\mathrm{q}=1$ surface.
Raises Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getQ2Surf (length_unit=1)
returns outboard-midplane minor radius of $\mathrm{q}=2$ surface.
Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions. getQ3Surf (length_unit=1)
returns outboard-midplane minor radius of $\mathrm{q}=3$ surface.
Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getQs (length_unit=1)
pulls $q$ values.
Raises Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getBtVac ()
Returns vacuum toroidal field on-axis. THIS MAY BE INCORRECT
Returns [nt] array of vacuum toroidal field.
Return type BtVac (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
getBtPla()
returns on-axis plasma toroidal field.
Raises Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions.

## getBpAvg ()

returns average poloidal field.
Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getFields()
pulls vacuum and plasma toroidal field, avg poloidal field.
Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.

```
getIpCalc()
```

returns Plasma Current, is the same as getIpMeas.
Returns [nt] array of the reconstructed plasma current.
Return type IpCalc (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.

```
getIpMeas()
```

returns magnetics-measured plasma current.
Returns [nt] array of measured plasma current.
Return type IpMeas (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.

```
getJp()
``` returns the calculated plasma current density Jp on flux grid.

Returns [nt,nz,nr] array of current density.

Return type Jp (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.

\section*{getBetaT()}
returns the calculated toroidal beta.
Raises Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getBetaP ()
returns the calculated poloidal beta.
Returns [nt] array of the calculated average poloidal beta.
Return type BetaP (Array)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.

\section*{getLi()}
returns the calculated internal inductance.
Returns [nt] array of the calculated internal inductance.
Return type Li (Array)
Raises ValueError - if module cannot retrieve data from the AUG afs system.
getBetas()
pulls calculated betap, betat, internal inductance.
Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getDiamagFlux()
returns the measured diamagnetic-loop flux.
Raises Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions. getDiamagBetaT()
returns diamagnetic-loop toroidal beta.
Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
```

getDiamagBetaP()

```
returns diamagnetic-loop avg poloidal beta.
Raises Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getDiamagTauE ()
returns diamagnetic-loop energy confinement time.
Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getDiamagWp()
returns diamagnetic-loop plasma stored energy.
Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.

\section*{getDiamag()}
pulls diamagnetic flux measurements, toroidal and poloidal beta, energy confinement time and stored energy.

Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getWMHD ()
returns calculated MHD stored energy.
Returns [nt] array of the calculated stored energy.

\section*{Return type WMHD (Array)}

Raises ValueError - if module cannot retrieve data from the AUG afs system.

\section*{get TauMHD ()}
returns the calculated MHD energy confinement time.
Raises Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getPinj()
returns the injected power.

\section*{Raises}
- Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
- -
getWbdot()
returns the calculated \(\mathrm{d} / \mathrm{dt}\) of magnetic stored energy.
Raises Not ImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getWpdot ()
returns the calculated \(\mathrm{d} / \mathrm{dt}\) of plasma stored energy.
Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
```

getBCentr()

```
returns Vacuum toroidal magnetic field at center of plasma
Returns [nt] array of B_t at center [T]
Return type B_cent (Array)
Raises ValueError - if module cannot retrieve data from the AUG afs system.
getRCentr (length_unit=1)
Returns Radius of BCenter measurement
Returns Radial position where Bcent calculated [m]

\section*{Return type R}
```

getEnergy()

```
pulls the calculated energy parameters - stored energy, tau_E, injected power, \(\mathrm{d} / \mathrm{dt}\) of magnetic and plasma stored energy.

Raises NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getMachineCrossSection()
Returns R,Z coordinates of vacuum-vessel wall for masking, plotting routines.

\section*{Returns}
( __limiter, Z_limiter)
- R_limiter (Array) - [n] array of x -values for machine cross-section.
- Z_limiter (Array) - [n] array of y -values for machine cross-section.
getMachineCrossSectionFull()
Returns R,Z coordinates of vacuum-vessel wall for plotting routines.
Absent additional vector-graphic data on machine cross-section, returns getMachineCrossSection().

Returns result from getMachineCrossSection().
getCurrentSign()
Returns the sign of the current, based on the check in Steve Wolfe's IDL implementation efit_rz2psi.pro.
Returns 1 for positive-direction current, -1 for negative.
Return type currentSign (Integer)
getParam (path)
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 NotImplementedError - Not implemented on ASDEX-Upgrade reconstructions.
getSSQ (inp, **kwargs)
returns single value quantities in the case SV file doesn't exist and coniditions the data in a way that is expected from a dd SV shotfile. This seamlessly hides the lack of an SV file.

Returns corresponding data
Return type signal (dd.signal Object)
Raises ValueError - if module cannot retrieve data from the AUG AFS system.
rz2BR \((R, Z, t\), return_t=False, make_grid=False, each_t=True, length_unit=1)
Calculates the major radial component of the magnetic field at the given \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) coordinates.
Uses
\[
B_{R}=-\frac{1}{2 \pi R} \frac{\partial \psi}{\partial Z}
\]

\section*{Parameters}
- R (Array-like or scalar float) - Values of the radial coordinate to map to radial field. 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 radial field. 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (BR, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(B R\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(B R\) ).

\section*{Returns}
\(B R\) or (BR, time_idxs)
- BR (Array or scalar float) - The major radial component of the magnetic field. 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 \(B R\) has this shape as well, unless the make \(\_\)grid keyword was True, in which case \(B R\) has shape (len \((Z)\), len \((R)\) ).
- time_idxs (Array with same shape as \(B R\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single \(B R\) value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
BR_val = Eq_instance.rz2BR(0.6, 0, 0.26)
Find BR values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\) at the single time \(\mathrm{t}=0.26 \mathrm{~s}\). Note that the \(Z\) vector must be fully specified, even if the values are all the same:

BR_arr \(=\) Eq_instance.rz2BR([0.6, 0.8], [0, 0], 0.26)
Find \(B R\) values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

BR_arr = Eq_instance.rz2BR(0.6, 0, [0.2, 0.3])

```

Find BR values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

BR_arr = Eq_instance.rz2BR([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find BR values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

BR_mat = Eq_instance.rz2BR(R, Z, 0.2, make_grid=True)

```
rz2BZ \((R, Z, t\), return_t=False, make_grid=False, each_t=True, length_unit=1)
Calculates the vertical component of the magnetic field at the given \((R, Z, t)\) coordinates.
Uses
\[
B_{Z}=\frac{1}{2 \pi R} \frac{\partial \psi}{\partial R}
\]

\section*{Parameters}
- \(\mathbf{R}\) (Array-like or scalar float) - Values of the radial coordinate to map to vertical field. 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 vertical field. 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (BZ, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(B Z\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(B Z\) ).

\section*{Returns}
\(B Z\) or (BZ, time_idxs)
- BZ (Array or scalar float) - The vertical component of the magnetic field. 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 \(B Z\) has this shape as well, unless the make \(\_\)grid keyword was True, in which case \(B Z\) has shape (len \((Z)\), len \((R)\) ).
- time_idxs (Array with same shape as \(B Z\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single BZ value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

BZ_val = Eq_instance.rz2BZ(0.6, 0, 0.26)

```

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

BZ_arr = Eq_instance.rz2BZ([0.6, 0.8], [0, 0], 0.26)

```

Find BZ values at \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

BZ_arr = Eq_instance.rz2BZ(0.6, 0, [0.2, 0.3])

```

Find BZ values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

BZ_arr = Eq_instance.rz2BZ([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find BZ values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

BZ_mat = Eq_instance.rz2BZ(R, Z, 0.2, make_grid=True)

```
```

class eqtools.AUGData.YGCAUGInterface

```
    Bases: object
    getMachineCrossSection (shot)
        Returns R,Z coordinates of vacuum-vessel wall for masking, plotting routines.

\section*{Returns}
( __limiter, Z_limiter)
- R_limiter (Array) - [n] array of x -values for machine cross-section.
- Z_limiter (Array) - [ n\(]\) array of y -values for machine cross-section.
getMachineCrossSectionFull (shot)
Returns R,Z coordinates of vacuum-vessel wall for plotting routines.
Absent additional vector-graphic data on machine cross-section, returns getMachineCrossSection().

Returns result from getMachineCrossSection().
class eqtools.AUGData.AUGDDDataProp (shot, shotfile \({ }^{\prime} E Q H\) ', edition=0, shotfile \(2=\) None, length_unit='m', tspline=False, monotonic=True, experiment = 'AUGD')
Bases: eqtools.AUGData.AUGDDData, eqtools.core.PropertyAccessMixin

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

\subsection*{4.1.3 eqtools.CModEFIT module}

This module provides classes inheriting eqtools.EFIT.EFITTree 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, mono-
tonic=True)
Bases: eqtools.EFIT.EFITTree

```

Inherits eqtools.EFIT.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.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline \begin{tabular}{l} 
'de- \\
fault'
\end{tabular} & \begin{tabular}{l} 
whatever the default in the tree is (no conversion is performed, units may \\
be inconsistent)
\end{tabular} \\
\hline
\end{tabular}

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.
Keyword Arguments length_unit (String or 3) - unit for plasma volume. Defaults to 3 , indicating default volumetric unit (typically \(\mathrm{m}^{\wedge} 3\) ).

Returns [nt,npsi] array of volume within flux surface.
Return type fluxVol (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getRmidPsi (length_unit=1)
returns maximum major radius of each flux surface.
Keyword Arguments length_unit (String or 1) - unit of Rmid. Defaults to 1, indicating the default parameter unit (typically m).

Returns [nt,npsi] array of maximum (outboard) major radius of flux surface psi.
Return type Rmid (Array)
Raises Value Error - if module cannot retrieve data from MDS tree.
getF ()
returns \(\mathrm{F}=\mathrm{RB}_{-}\{\)Phi \(\}(\mathrm{Psi})\), often calculated for grad-shafranov solutions.
Returns [nt,npsi] array of F=RB_\{Phi\}(Psi)
Return type F (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getFluxPres()
returns pressure at flux surface.
Returns [nt,npsi] array of pressure on flux surface psi.
Return type p (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getFFPrime()
returns FF' function used for grad-shafranov solutions.
Returns [nt,npsi] array of FF' fromgrad-shafranov solution.
Return type FFprime (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getPPrime()
returns plasma pressure gradient as a function of psi.
Returns [nt,npsi] array of pressure gradient on flux surface psi from grad-shafranov solution.
Return type pprime (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQProfile()
returns profile of safety factor q .
Returns [nt,npsi] array of q on flux surface psi.

Return type qpsi (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getRLCFS (length_unit=1) returns R-values of LCFS position.

Returns [nt,n] array of R of LCFS points.
Return type RLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getZLCFS (length_unit=1)
returns Z-values of LCFS position.
Returns [nt,n] array of Z of LCFS points.
Return type ZLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getMachineCrossSectionFull()
Pulls C-Mod cross-section data from tree, converts to plottable vector format for use in other plotting routines

\section*{Returns}
\((x, y)\)
- \(\mathbf{x}\) (Array) - [n] array of x -values for machine cross-section.
- \(\mathbf{y}\) (Array) - [n] array of y -values for machine cross-section.

Raises ValueError - if module cannot retrieve data from MDS tree.
getRCentr (length_unit=1)
returns EFIT radius where Bcentr evaluated
Returns Radial position where Bcent calculated [m]
Return type R
Raises ValueError - if module cannot retrieve data from MDS tree.
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.

\subsection*{4.1.4 eqtools.D3DEFIT module}

This module provides classes inheriting eqtools.EFIT.EFITTree for working with DIII-D EFIT data.
```

class eqtools.D3DEFIT.D3DEFITTree(shot, tree='EFIT01', length_unit='m', gfile='geqdsk',
afile='aeqdsk', tspline=False, monotonic=True)

```

Bases: eqtools.EFIT.EFITTree
Inherits eqtools.EFIT.EFITTree class. Machine-specific data handling class for DIII-D. 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 DIII-D 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) - DIII-D shot index.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline \begin{tabular}{l} 
'de- \\
fault'
\end{tabular} & \begin{tabular}{l} 
whatever the default in the tree is (no conversion is performed, units may \\
be inconsistent)
\end{tabular} \\
\hline
\end{tabular}

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).
getFluxVol()
Not implemented in D3DEFIT tree.
Returns volume within flux surface [psi,t]
getRmidPsi (length_unit=1)
returns maximum major radius of each flux surface.
Keyword Arguments length_unit (String or 1) - unit of Rmid. Defaults to 1, indicating the default parameter unit (typically m).

Returns [nt,npsi] array of maximum (outboard) major radius of flux surface psi.
Return type Rmid (Array)
Raises Value Error - if module cannot retrieve data from MDS tree.
class eqtools.D3DEFIT.D3DEFITTreeProp (shot, tree='EFIT01', length_unit='m', gfile='geqdsk', afile='aeqdsk', tspline=False, monotonic=True)
Bases: eqtools.D3DEFIT.D3DEFITTree, eqtools.core.PropertyAccessMixin
D3DEFITTree with the PropertyAccessMixin added to enable property-style access. This is good for interactive use, but may drag the performance down.

\subsection*{4.1.5 eqtools.EFIT module}

Provides class inheriting eqtools.core.Equilibrium for working with EFIT data.
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.

\section*{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.

\section*{Keyword Arguments}
- length_unit (string) - Sets the base unit used for any quantity whose dimensions are length to any power. Valid options are:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline \begin{tabular}{l} 
'de- \\
fault'
\end{tabular} & \begin{tabular}{l} 
whatever the default in the tree is (no conversion is performed, units may \\
be inconsistent)
\end{tabular} \\
\hline
\end{tabular}

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).
getInfo()
returns namedtuple of shot information

\section*{Returns}
namedtuple containing
\begin{tabular}{|l|l|}
\hline shot & C-Mod shot index (long) \\
\hline tree & EFIT tree (string) \\
\hline nr & size of R-axis for spatial grid \\
\hline nz & size of Z-axis for spatial grid \\
\hline nt & size of timebase for flux grid \\
\hline
\end{tabular}

\section*{getTimeBase()}
returns EFIT time base vector.
Returns [nt] array of time points.
Return type time (array)
Raises ValueError - if module cannot retrieve data from MDS tree.

\section*{getFluxGrid()}
returns EFIT flux grid.
Note that this method preserves whatever sign convention is used in the tree. For C-Mod, this means that the result should be multiplied by \(-1 *\) getCurrentSign () in most cases.

Returns [nt,nz,nr] array of (non-normalized) flux on grid.
Return type psiRZ (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getRGrid(length_unit=1)
returns EFIT R-axis.
Returns [nr] array of R-axis of flux grid.
Return type rGrid (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getZGrid (length_unit=1)
returns EFIT Z-axis.
Returns [nz] array of Z-axis of flux grid.
Return type zGrid (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.

\section*{getFluxAxis()}
returns psi on magnetic axis.
Returns [nt] array of psi on magnetic axis.
Return type psiAxis (Array)

Raises ValueError - if module cannot retrieve data from MDS tree.
getFluxLCFS()
returns psi at separatrix.
Returns [nt] array of psi at LCFS.
Return type psiLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getFluxVol (length_unit=3)
returns volume within flux surface.
Keyword Arguments length_unit (String or 3) - unit for plasma volume. Defaults to 3 , indicating default volumetric unit (typically \(\mathrm{m}^{\wedge} 3\) ).
Returns [nt,npsi] array of volume within flux surface.
Return type fluxVol (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getVolLCFS (length_unit=3)
returns volume within LCFS.
Keyword Arguments length_unit (String or 3) - unit for LCFS volume. Defaults to 3 , denoting default volumetric unit (typically \(\mathrm{m}^{\wedge} 3\) ).
Returns [nt] array of volume within LCFS.
Return type volLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getRmidPsi (length_unit=1)
returns maximum major radius of each flux surface.
Keyword Arguments length_unit (String or 1) - unit of Rmid. Defaults to 1, indicating the default parameter unit (typically m).

Returns [nt,npsi] array of maximum (outboard) major radius of flux surface psi.
Return type Rmid (Array)
Raises Value Error - if module cannot retrieve data from MDS tree.
getRLCFS (length_unit=1)
returns R-values of LCFS position.
Returns [nt,n] array of R of LCFS points.
Return type RLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getZLCFS (length_unit=1) returns Z-values of LCFS position.

Returns [nt,n] array of Z of LCFS points.
Return type ZLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
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 \(\mathrm{psi}=\mathrm{psiLCFS}\).

\section*{Raises}
- NotImplementedError - if matplotlib. pyplot is not loaded.
- ValueError - if limiter outline is not available.
getF ()
returns \(\mathrm{F}=\mathrm{RB} \_\{\mathrm{Phi}\}(\mathrm{Psi})\), often calculated for grad-shafranov solutions.
Note that this method preserves whatever sign convention is used in the tree. For C-Mod, this means that the result should be multiplied by \(-1 *\) getCurrentSign () in most cases.

Returns [nt,npsi] array of \(\mathrm{F}=\) RB_\{Phi\}(Psi)
Return type F (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getFluxPres()
returns pressure at flux surface.
Returns [nt,npsi] array of pressure on flux surface psi.
Return type p (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.

\section*{getFFPrime()}
returns FF' function used for grad-shafranov solutions.
Returns [nt,npsi] array of FF' fromgrad-shafranov solution.
Return type FFprime (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getPPrime()
returns plasma pressure gradient as a function of psi.
Returns [nt,npsi] array of pressure gradient on flux surface psi from grad-shafranov solution.
Return type pprime (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getElongation()
returns LCFS elongation.
Returns [nt] array of LCFS elongation.
Return type kappa (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getUpperTriangularity()
returns LCFS upper triangularity.
Returns [nt] array of LCFS upper triangularity.
Return type deltau (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getLowerTriangularity()
returns LCFS lower triangularity.
Returns [nt] array of LCFS lower triangularity.
Return type deltal (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getShaping()
pulls LCFS elongation and upper/lower triangularity.
Returns namedtuple containing (kappa, delta_u, delta_l)
Raises ValueError - if module cannot retrieve data from MDS tree.
getMagR (length_unit=1)
returns magnetic-axis major radius.
Returns [nt] array of major radius of magnetic axis.
Return type magR (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getMagZ (length_unit=1)
returns magnetic-axis Z .
Returns [nt] array of Z of magnetic axis.
Return type magZ (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getAreaLCFS (length_unit=2)
returns LCFS cross-sectional area.
Keyword Arguments length_unit (String or 2) - unit for LCFS area. Defaults to 2, denoting default areal unit (typically \(\mathrm{m}^{\wedge} 2\) ).

Returns [nt] array of LCFS area.
Return type areaLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getAOut (length_unit=1)
returns outboard-midplane minor radius at LCFS.
Keyword Arguments length_unit (String or 1) - unit for minor radius. Defaults to 1 , denoting default length unit (typically m).
Returns [nt] array of LCFS outboard-midplane minor radius.
Return type aOut (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getRmidOut (length_unit=1)
returns outboard-midplane major radius.
Keyword Arguments length_unit (String or 1) - unit for major radius. Defaults to 1 , denoting default length unit (typically m).
Returns [nt] array of major radius of LCFS.
Return type RmidOut (Array)

Raises ValueError - if module cannot retrieve data from MDS tree.
getGeometry (length_unit=None)
pulls dimensional geometry parameters.
Returns namedtuple containing (magR,magZ,areaLCFS,aOut,RmidOut)
Raises ValueError - if module cannot retrieve data from MDS tree.
```

getQProfile()

```
returns profile of safety factor q .
Returns [nt,npsi] array of q on flux surface psi.
Return type qpsi (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQ0 ()
returns \(q\) on magnetic axis,q0.
Returns [nt] array of \(\mathrm{q}(\mathrm{psi}=0)\).
Return type q0 (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQ95 ()
returns q at \(95 \%\) flux surface.
Returns [nt] array of \(\mathrm{q}(\mathrm{psi}=0.95)\).
Return type q95 (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQLCFS ()
returns q on LCFS (interpolated).
Returns [nt] array of \(q^{*}\) (interpolated).
Return type qLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQ1Surf (length_unit=1)
returns outboard-midplane minor radius of \(\mathrm{q}=1\) surface.
Keyword Arguments length_unit (String or 1) - unit for minor radius. Defaults to 1 , denoting default length unit (typically m).
Returns [nt] array of minor radius of \(\mathrm{q}=1\) surface.
Return type qr1 (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQ2Surf (length_unit=1)
returns outboard-midplane minor radius of \(\mathrm{q}=2\) surface.
Keyword Arguments length_unit (String or 1) - unit for minor radius. Defaults to 1 , denoting default length unit (typically m).

Returns [nt] array of minor radius of \(\mathrm{q}=2\) surface.
Return type qr2 (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQ3Surf (length_unit=1)
returns outboard-midplane minor radius of \(\mathrm{q}=3\) surface.
Keyword Arguments length_unit (String or 1) - unit for minor radius. Defaults to 1 , denoting default length unit (typically m ).

Returns [nt] array of minor radius of \(\mathrm{q}=3\) surface.
Return type qr3 (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQs (length_unit=1)
pulls q values.
Returns namedtuple containing (q0,q95,qLCFS,rq1,rq2,rq3).
Raises ValueError - if module cannot retrieve data from MDS tree.
getBtVac ()
Returns vacuum toroidal field on-axis.
Returns [nt] array of vacuum toroidal field.
Return type BtVac (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
```

getBtPla()

```
returns on-axis plasma toroidal field.
Returns [nt] array of toroidal field including plasma effects.
Return type BtPla (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getBpAvg ()
returns average poloidal field.
Returns [nt] array of average poloidal field.
Return type BpAvg (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getFields()
pulls vacuum and plasma toroidal field, avg poloidal field.
Returns namedtuple containing (btaxv,btaxp,bpolav).
Raises ValueError - if module cannot retrieve data from MDS tree.
getIpCalc()
returns EFIT-calculated plasma current.
Returns [nt] array of EFIT-reconstructed plasma current.
Return type IpCalc (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getIpMeas()
returns magnetics-measured plasma current.
Returns [nt] array of measured plasma current.
Return type IpMeas (Array)

Raises ValueError - if module cannot retrieve data from MDS tree.
getJp()
returns EFIT-calculated plasma current density Jp on flux grid.
Returns [nt,nz,nr] array of current density.
Return type Jp (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getBetaT()
returns EFIT-calculated toroidal beta.
Returns [nt] array of EFIT-calculated average toroidal beta.
Return type BetaT (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.

\section*{getBetaP()}
returns EFIT-calculated poloidal beta.
Returns [nt] array of EFIT-calculated average poloidal beta.
Return type BetaP (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getLi()
returns EFIT-calculated internal inductance.
Returns [nt] array of EFIT-calculated internal inductance.
Return type Li (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getBetas()
pulls calculated betap, betat, internal inductance
Returns namedtuple containing (betat,betap,Li)
Raises ValueError - if module cannot retrieve data from MDS tree.
getDiamagFlux ()
returns measured diamagnetic-loop flux.
Returns [nt] array of diamagnetic-loop flux.
Return type Flux (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getDiamagBetaT()
returns diamagnetic-loop toroidal beta.
Returns [nt] array of measured toroidal beta.
Return type BetaT (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getDiamagBetaP()
returns diamagnetic-loop avg poloidal beta.
Returns [nt] array of measured poloidal beta.
Return type BetaP (Array)

Raises ValueError - if module cannot retrieve data from MDS tree.
getDiamagTauE ()
returns diamagnetic-loop energy confinement time.
Returns [nt] array of measured energy confinement time.
Return type tauE (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
```

getDiamagWp()

```
returns diamagnetic-loop plasma stored energy.
Returns [nt] array of measured plasma stored energy.
Return type Wp (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getDiamag ()
pulls diamagnetic flux measurements, toroidal and poloidal beta, energy confinement time and stored energy.

Returns namedtuple containing (diamag. flux, betatd, betapd, tauDiamag, WDiamag)
Raises ValueError - if module cannot retrieve data from MDS tree.

\section*{getWMHD ()}
returns EFIT-calculated MHD stored energy.
Returns [nt] array of EFIT-calculated stored energy.
Return type WMHD (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getTauMHD ()
returns EFIT-calculated MHD energy confinement time.
Returns [nt] array of EFIT-calculated energy confinement time.
Return type tauMHD (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getPinj()
returns EFIT-calculated injected power.
Returns [nt] array of EFIT-reconstructed injected power.
Return type Pinj (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getWbdot ()
returns EFIT-calculated d/dt of magnetic stored energy.
Returns [nt] array of \(\mathrm{d}(\mathrm{Wb}) / \mathrm{dt}\)
Return type dWdt (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getWpdot()
returns EFIT-calculated d/dt of plasma stored energy.
Returns [nt] array of \(\mathrm{d}(\mathrm{Wp}) / \mathrm{dt}\)

Return type dWdt (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getBCentr()
returns EFIT-Vacuum toroidal magnetic field in Tesla at Rcentr
Returns [nt] array of B_t at center [T]
Return type B_cent (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
```

getRCentr(length_unit=l)

```
returns EFIT radius where Bcentr evaluated
Returns Radial position where Bcent calculated [m]

\section*{Return type R}

Raises ValueError - if module cannot retrieve data from MDS tree.
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)
Raises ValueError - if module cannot retrieve data from MDS tree.
getMachineCrossSection()
Returns R,Z coordinates of vacuum-vessel wall for masking, plotting routines.

\section*{Returns}
( __limiter, Z_limiter)
- R_limiter (Array) - [ n\(]\) array of x -values for machine cross-section.
- Z_limiter (Array) - [n] array of y-values for machine cross-section.
getMachineCrossSectionFull()
Returns R,Z coordinates of vacuum-vessel wall for plotting routines.
Absent additional vector-graphic data on machine cross-section, returns getMachineCrossSection().

Returns result from getMachineCrossSection().
getCurrentSign()
Returns the sign of the current, based on the check in Steve Wolfe's IDL implementation efit_rz2psi.pro.
Returns 1 for positive-direction current, -1 for negative.
Return type currentSign (Integer)
getParam (path)
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.

\subsection*{4.1.6 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.

\section*{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, \((\mathrm{S}, \mathrm{M})\). 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.

\section*{Keyword Arguments}
- length_unit - String. Sets the base unit used for any quantity whose dimensions are length to any power. Valid options are:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline \begin{tabular}{l} 
'de- \\
fault'
\end{tabular} & \begin{tabular}{l} 
whatever the default in the tree is (no conversion is performed, units may \\
be inconsistent)
\end{tabular} \\
\hline
\end{tabular}

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 ( \(\mathrm{M}, \mathrm{N}, \mathrm{P}\) ), 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 \((\mathrm{Q}, \mathrm{M})\)
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)
Abstract method. See child classes for implementation.
Returns R-positions (n points) mapping LCFS [t, n ]
getZLCFS (length_unit=1)
Abstract method. See child classes for implementation.
Returns Z-positions (n points) mapping LCFS [t, n ]
getCurrentSign()
Abstract method. See child classes for implementation.
Returns calculated current direction, where \(\mathrm{CCW}=+\)

\subsection*{4.1.7 eqtools.NSTXEFIT module}

This module provides classes inheriting eqtools.EFIT.EFITTree 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.EFITIree
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)

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline \begin{tabular}{l} 
'de- \\
fault'
\end{tabular} & \begin{tabular}{l} 
whatever the default in the tree is (no conversion is performed, units may \\
be inconsistent)
\end{tabular} \\
\hline
\end{tabular}

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).

\section*{getFluxGrid()}
returns EFIT flux grid.
Returns [nt,nz,nr] array of (non-normalized) flux on grid.
Return type psiRZ (Array)

Raises ValueError - if module cannot retrieve data from MDS tree.
getMachineCrossSection()
Returns R,Z coordinates of vacuum-vessel wall for masking, plotting routines.
Returns The requested data.

\section*{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.
Keyword Arguments length_unit (String or 1) - unit of Rmid. Defaults to 1, indicating the default parameter unit (typically m).

Returns [nt,npsi] array of maximum (outboard) major radius of flux surface psi.
Return type Rmid (Array)
Raises Value Error - if module cannot retrieve data from MDS tree.
```

getIpCalc()

```
returns EFIT-calculated plasma current.
Returns [nt] array of EFIT-reconstructed plasma current.
Return type IpCalc (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getVolLCFS (length_unit=3)
returns volume within LCFS.
Keyword Arguments length_unit (String or 3) - unit for LCFS volume. Defaults to 3 , denoting default volumetric unit (typically \(\mathrm{m}^{\wedge} 3\) ).

Returns [nt] array of volume within LCFS.
Return type volLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
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.
Returns All mapping with Volnorm not implemented
psinorm2volnorm (*args, **kwargs)
Calculated normalized volume of flux surfaces not stored in NSTX EFIT.
Returns 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.

\subsection*{4.1.8 eqtools.TCVLIUQE module}

This module provides classes inheriting eqtools.EFIT.EFITTree for working with TCV LIUQE Equilibrium.
```

eqtools.TCVLIUQE.greenArea(vs)

```
class eqtools.TCVLIUQE.TCVLIUQETree (shot, tree='tcv_shot', length_unit='m', gfile='g_eqdsk', afile ='a_eqdsk', tspline \(=\) False, monotonic \(=\) True )
Bases: eqtools.EFIT.EFITTree
Inherits eqtools.EFIT.EFITTree class. Machine-specific data handling class for TCV Machine. Pulls LIUQUE data from selected MDS tree and shot, stores as object attributes eventually transforming it in the equivalent quantity for EFIT. Each variable or set of variables is recovered with a corresponding getter method. Essential data for LIUQUE mapping are pulled on initialization (e.g. psirz grid). Additional data are pulled at the first request and stored for subsequent usage.
Intializes TCV 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) - TCV shot index.

\section*{Keyword Arguments}
- tree (string) - Optional input for LIUQE tree, defaults to 'RESULTS' (i.e., LIUQE data are under results::).
- length_unit (string) - Sets the base unit used for any quantity whose dimensions are length to any power. Valid options are:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline \begin{tabular}{l} 
'de- \\
fault'
\end{tabular} & \begin{tabular}{l} 
whatever the default in the tree is (no conversion is performed, units may \\
be inconsistent)
\end{tabular} \\
\hline
\end{tabular}

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).
getInfo()
returns namedtuple of shot information

\section*{Returns}
namedtuple containing
\begin{tabular}{|l|l|}
\hline shot & TCV shot index (long) \\
\hline tree & LIUQE tree (string) \\
\hline nr & size of R-axis for spatial grid \\
\hline nz & size of Z-axis for spatial grid \\
\hline nt & size of timebase for flux grid \\
\hline
\end{tabular}

\section*{getTimeBase()}
returns LIUQE time base vector.
Returns [nt] array of time points.
Return type time (array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getFluxGrid()
returns LIUQE flux grid.
Returns [nt,nz,nr] array of (non-normalized) flux on grid.
Return type psiRZ (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getRGrid (length_unit=1)
returns LIUQE R-axis.
Returns [nr] array of R-axis of flux grid.
Return type rGrid (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getZGrid(length_unit=1)
returns LIUQE Z-axis.
Returns [nz] array of Z-axis of flux grid.
Return type zGrid (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getFluxAxis()
returns psi on magnetic axis.
Returns [nt] array of psi on magnetic axis.
Return type psiAxis (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getFluxLCFS()
returns psi at separatrix.

Returns [nt] array of psi at LCFS.
Return type psiLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.

\section*{getFluxVol (length_unit=3)}
returns volume within flux surface. This is not implemented in LIUQE as default output. So we use contour and GREEN theorem to get the area within a default grid of the PSI. Then we compute the volume by multipling for \(2 \mathrm{pi} *\) VolLCFS / AreaLCFS.

Keyword Arguments length_unit (String or 3) - unit for plasma volume. Defaults to 3 , indicating default volumetric unit (typically \(\mathrm{m}^{\wedge} 3\) ).

Returns [nt,npsi] array of volume within flux surface.
Return type fluxVol (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getVolLCFS (length_unit=3)
returns volume within LCFS.
Keyword Arguments length_unit (String or 3) - unit for LCFS volume. Defaults to 3 , denoting default volumetric unit (typically m^3).

Returns [nt] array of volume within LCFS.
Return type volLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getRmidPsi (length_unit=1)
returns maximum major radius of each flux surface.
Keyword Arguments length_unit (String or 1) - unit of Rmid. Defaults to 1, indicating the default parameter unit (typically m).

Returns [nt,npsi] array of maximum (outboard) major radius of flux surface psi.
Return type Rmid (Array)
Raises Value Error - if module cannot retrieve data from MDS tree.
getRLCFS (length_unit=1)
returns R-values of LCFS position.
Returns [nt,n] array of R of LCFS points.
Return type RLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getZLCFS (length_unit=1)
returns Z-values of LCFS position.
Returns [nt,n] array of Z of LCFS points.
Return type ZLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getF ()
returns \(\mathrm{F}=\) RB_\{Phi\}(Psi), often calculated for grad-shafranov solutions. Not implemented on LIUQE
Returns [nt,npsi] array of \(\mathrm{F}=\) RB_\{Phi \(\}\) (Psi) Not stored on LIUQE nodes
Return type F (Array)

Raises ValueError - if module cannot retrieve data from MDS tree.
getFluxPres()
returns pressure at flux surface. Not implemented. We have pressure saved on the same grid of psi
Returns [nt,npsi] array of pressure on flux surface psi. Not implemented on LIUQE nodes. We have pressure on the grid use for psi

Return type p (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.

\section*{getFFPrime()}
returns FF' function used for grad-shafranov solutions.
Returns [nt,npsi] array of FF' fromgrad-shafranov solution.
Return type FFprime (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getPPrime()
returns plasma pressure gradient as a function of psi.
Returns [nt,npsi] array of pressure gradient on flux surface psi from grad-shafranov solution.
Return type pprime (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getElongation()
returns LCFS elongation.
Returns [nt] array of LCFS elongation.
Return type kappa (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getUpperTriangularity()
returns LCFS upper triangularity.
Returns [nt] array of LCFS upper triangularity.
Return type deltau (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getLowerTriangularity()
returns LCFS lower triangularity.
Returns [nt] array of LCFS lower triangularity.
Return type deltal (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getMagR (length_unit=1)
returns magnetic-axis major radius.
Returns [nt] array of major radius of magnetic axis.
Return type magR (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getMagZ (length_unit=1)
returns magnetic-axis Z .
Returns [nt] array of Z of magnetic axis.
Return type magZ (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getAreaLCFS (length_unit=2)
returns LCFS cross-sectional area.
Keyword Arguments length_unit (String or 2) - unit for LCFS area. Defaults to 2, denoting default areal unit (typically \(\mathrm{m}^{\wedge} 2\) ).

Returns [nt] array of LCFS area.
Return type areaLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getAOut (length_unit=1)
returns outboard-midplane minor radius at LCFS. In LIUQE it is the last value of
esults::r_max_psi

\section*{Keyword Args:}
length_unit (String or 1): unit for minor radius. Defaults to 1, denoting default length unit (typically m).

Returns: aOut (Array): [nt] array of LCFS outboard-midplane minor radius.
Raises: ValueError: if module cannot retrieve data from MDS tree.
getRmidOut (length_unit=1)
returns outboard-midplane major radius. It uses getA
Keyword Arguments length_unit (String or 1) - unit for major radius. Defaults to 1 , denoting default length unit (typically m).

Returns [nt] array of major radius of LCFS.
Return type RmidOut (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQProfile()
returns profile of safety factor \(q\).
Returns [nt,npsi] array of q on flux surface psi.
Return type qpsi (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQ0()
returns q on magnetic axis,q0.
Returns [nt] array of \(\mathrm{q}(\mathrm{psi}=0)\).
Return type q0 (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQ95 ()
returns q at \(95 \%\) flux surface.

Returns [nt] array of \(\mathrm{q}(\mathrm{psi}=0.95)\).
Return type q95 (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getQLCFS ()
returns q on LCFS (interpolated).
Returns [nt] array of \(q^{*}\) (interpolated).
Return type qLCFS (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getBtVac ()
Returns vacuum toroidal field on-axis. We use MDSplus.Connection for a proper use of the TDI function tcv_eq()

Returns [nt] array of vacuum toroidal field.
Return type BtVac (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getBtPla()
returns on-axis plasma toroidal field.
Returns [nt] array of toroidal field including plasma effects.
Return type BtPla (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getIpCalc()
returns EFIT-calculated plasma current.
Returns [nt] array of EFIT-reconstructed plasma current.
Return type IpCalc (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getIpMeas()
returns magnetics-measured plasma current.
Returns [nt] array of measured plasma current.
Return type IpMeas (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getBetaT()
returns LIUQE-calculated toroidal beta.
Returns [nt] array of LIUQE-calculated average toroidal beta.
Return type BetaT (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getBetaP()
returns LIUQE-calculated poloidal beta.
Returns [nt] array of LIUQE-calculated average poloidal beta.
Return type BetaP (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getLi()
returns LIUQE-calculated internal inductance.
Returns [nt] array of LIUQE-calculated internal inductance.
Return type Li (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getDiamagWp()
returns diamagnetic-loop plasma stored energy.
Returns [nt] array of measured plasma stored energy.
Return type Wp (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
get TauMHD ()
returns LIUQE-calculated MHD energy confinement time.
Returns [nt] array of LIUQE-calculated energy confinement time.
Return type tauMHD (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getMachineCrossSection()
Pulls TCV cross-section data from tree, converts to plottable vector format for use in other plotting routines

\section*{Returns}
\((x, y)\)
- \(\mathbf{x}\) (Array) - [ n\(]\) array of x -values for machine cross-section.
- \(\mathbf{y}\) (Array) - [n] array of y -values for machine cross-section.

Raises ValueError - if module cannot retrieve data from MDS tree.
getMachineCrossSectionPatch()
Pulls TCV cross-section data from tree, converts it directly to a matplotlib patch which can be simply added to the approriate axes call in plotFlux()

Returns tiles matplotlib Patch, vessel matplotlib Patch
Raises ValueError - if module cannot retrieve data from MDS tree.
plotFlux (fill=True, mask=False)
Plots LIQUE TCV flux contours directly from psi grid.
Returns the Figure instance created and the time slider widget (in case you need to modify the callback). f.axes contains the contour plot as the first element and the time slice slider as the second element.

Keyword Arguments fill (Boolean) - Set True to plot filled contours. Set False (default) to plot white-background color contours.
class eqtools.TCVLIUQE.TCVLIUQETreeProp (shot, tree='tcv_shot', length_unit='m', gfile='g_eqdsk', afile='a_eqdsk', tspline=False, monotonic=True)
Bases: eqtools.TCVLIUQE.TCVLIUQETree, eqtools.core.PropertyAccessMixin
TCVLIUQETree with the PropertyAccessMixin added to enable property-style access. This is good for interactive use, but may drag the performance down.

\subsection*{4.1.9 eqtools.afilereader module}

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

\section*{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: ob ject
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

\section*{Examples}

Load a-file data located at file_path:
```

afr = eqtools.AFileReader(file_path)

```

Recover a datapoint (for example, shot, stored as afr._shot), using copy-protected \(\qquad\) getattribute m method:
```

shot = afr.shot

```

Assign a new attribute to afr - note that this will raise an AttributeError if attempting to overwrite a previouslystored attribute:
```

afr.attribute = val

```

\subsection*{4.1.10 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 \(\qquad\) setattr_ \(\qquad\) () such that it will raise an AttributeError when attempting to write an attribute \(A T T R\) for which there is already a method getATTR.
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 \(\mathrm{x}, \mathrm{y}\) 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.

\section*{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 True/False result for whether the point is contained within the polygon.
Return type result (Boolean)
class eqtools.core.Equilibrium (length_unit='m', tspline=False, monotonic=True, ver-
Bases: object
bose=True)

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.

\section*{Keyword Arguments}
- length_unit (String) - Sets the base unit used for any quantity whose dimensions are length to any power. Valid options are:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline \begin{tabular}{l} 
'de- \\
fault'
\end{tabular} & \begin{tabular}{l} 
whatever the default in the tree is (no conversion is performed, units may \\
be inconsistent)
\end{tabular} \\
\hline
\end{tabular}

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.

\section*{Raises}
- ValueError - If length_unit is not a valid unit specifier.
- ValueError - If tspline is True but module trispline did not load successfully.
rho2rho (origin, destination, *args, **kwargs) Convert from one coordinate to another.

\section*{Parameters}
- origin (String) - Indicates which coordinates the data are given in. Valid options are:
\begin{tabular}{|l|l|}
\hline RZ & R,Z coordinates \\
\hline psinorm & Normalized poloidal flux \\
\hline phinorm & Normalized toroidal flux \\
\hline volnorm & Normalized volume \\
\hline Rmid & Midplane major radius \\
\hline r/a & Normalized minor radius \\
\hline
\end{tabular}

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:
\begin{tabular}{|l|l|}
\hline psinorm & Normalized poloidal flux \\
\hline phinorm & Normalized toroidal flux \\
\hline volnorm & Normalized volume \\
\hline Rmid & Midplane major radius \\
\hline r/a & Normalized minor radius \\
\hline q & Safety factor \\
\hline F & Flux function \(F=R B_{\phi}\) \\
\hline FFPrime & Flux function \(F F^{\prime}\) \\
\hline p & Pressure \\
\hline pprime & Pressure gradient \\
\hline v & Flux surface volume \\
\hline
\end{tabular}

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 \(r h o\). 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single psinorm value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find psinorm values at \(\mathrm{r} / \mathrm{a}\) points 0.6 and 0.8 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
psi_arr = Eq_instance.rho2rho('r/a', 'psinorm', [0.6, 0.8], 0.26)

Find \(p\) sinorm values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at \((\mathrm{r} / \mathrm{a}, \mathrm{t})\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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

```
rz2psi ( \(R, Z\), , , return_t=False, make_grid=False, each_t=True, length_unit=1)
Converts the passed \(\mathrm{R}, \mathrm{Z}\), t arrays to psi (unnormalized poloidal flux) values.
What is usually returned by EFIT is the stream function, \(\psi=\psi_{p} /(2 \pi)\) which has units of \(\mathrm{Wb} / \mathrm{rad}\).

\section*{Parameters}
- \(\mathbf{R}\) (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 \(e a c h \_t\) keyword is True, then \(t\) must be scalar or have exactly one dimension. If the each \(\_\)keyword is False, \(t\) must have the same shape as \(R\) and \(Z\) (or their meshgrid if make_grid is True).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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).

\section*{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 \(p s i\) has shape \((\operatorname{len}(Z)\), 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.

\section*{Examples}

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

Find single psi value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find psi values at \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\) at the single time \(\mathrm{t}=0.26 \mathrm{~s}\). 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 \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psi values at \((R, Z, t)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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 \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

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 ( \(\mathrm{R}, \mathrm{Z}, \mathrm{t}\) ).

Uses the definition:
\[
\text { psi_norm }=\frac{\psi-\psi(0)}{\psi(a)-\psi(0)}
\]

\section*{Parameters}
- \(\mathbf{R}\) (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 \(\_\)keyword is False, \(t\) must have the same shape as \(R\) and \(Z\) (or their meshgrid if make_grid is True).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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).

\section*{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.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single psinorm value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find psinorm values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\) at the single time \(t=0.26 \mathrm{~s}\). 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 \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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

```

Find psinorm values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

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

```
rz2phinorm (*args, **kwargs)

Calculates the normalized toroidal flux.
Uses the definitions:
\[
\begin{aligned}
\text { phi } & =\int q(\psi) d \psi \\
\text { phi_norm } & =\frac{\phi}{\phi(a)}
\end{aligned}
\]

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

\section*{Parameters}
- \(\mathbf{R}\) (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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting psinorm to phinorm.
- 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).

\section*{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.

\section*{Examples}

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

Find single phinorm value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find phinorm values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\) at the single time \(\mathrm{t}=0.26 \mathrm{~s}\). 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 \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find phinorm values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

phi_arr = Eq_instance.rz2phinorm([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_
\hookrightarrow=False)

```

Find phinorm values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

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.

\section*{Parameters}
- \(\mathbf{R}\) (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 \(\_\)keyword is False, \(t\) must have the same shape as \(R\) and \(Z\) (or their meshgrid if make_grid is True).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting psinorm to volnorm.
- 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).

\section*{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 \((\operatorname{len}(Z)\), 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.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single volnorm value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find volnorm values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\) at the single time \(\mathrm{t}=0.26 \mathrm{~s}\). 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 \((0.6 m, 0 m)\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
vol_arr \(=\) Eq_instance.rz2volnorm (0.6, 0, [0.2, 0.3])
Find volnorm values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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

```

Find volnorm values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

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.

\section*{Parameters}
- \(\mathbf{R}\) (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 \(e a c h \_t\) 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting psinorm to Rmid.
- 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).

\section*{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 \((\operatorname{len}(Z), \operatorname{len}(R))\).
- 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.

\section*{Examples}

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

Find single Rmid value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

R_mid_val = Eq_instance.rz2rmid(0.6, 0, 0. 26)

```

Find \(R \_m i d\) values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\) at the single time \(\mathrm{t}=0.26 \mathrm{~s}\). 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 \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find Rmid values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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

```

Find Rmid values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

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

```

\section*{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.

\section*{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 \(\mathrm{r} / \mathrm{a}\). 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting psinorm to Rmid.
- 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).

\section*{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 \((\operatorname{len}(Z)\), 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.

\section*{Examples}

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

Find single \(\mathrm{r} / \mathrm{a}\) value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find \(r / a\) values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\) at the single time \(\mathrm{t}=0.26 \mathrm{~s}\). 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 \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find \(\mathrm{r} / \mathrm{a}\) values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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 \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

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

```
rz2rho (method, *args, **kwargs)
Convert the passed ( \(\mathrm{R}, \mathrm{Z}, \mathrm{t}\) ) coordinates into one of several coordinates.

\section*{Parameters}
- method (String) - Indicates which coordinates to convert to. Valid options are:
\begin{tabular}{|l|l|}
\hline psinorm & Normalized poloidal flux \\
\hline phinorm & Normalized toroidal flux \\
\hline volnorm & Normalized volume \\
\hline Rmid & Midplane major radius \\
\hline r/a & Normalized minor radius \\
\hline q & Safety factor \\
\hline F & Flux function \(F=R B_{\phi}\) \\
\hline FFPrime & Flux function \(F F^{\prime}\) \\
\hline p & Pressure \\
\hline pprime & Pressure gradient \\
\hline v & Flux surface volume \\
\hline
\end{tabular}

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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single psinorm value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find psinorm values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\) at the single time \(\mathrm{t}=0.26 \mathrm{~s}\). 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 \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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 \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

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

```
rmid2roa ( \(R \_\)mid, \(t\), each_t=True, return_t=False, \(s q r t=\) False, blob=None, length_unit=1)
Convert the passed (R_mid, t) coordinates into r/a.

\section*{Parameters}
- R_mid (Array-like or scalar float) - Values of the outboard midplane major radius to map to \(\mathrm{r} / \mathrm{a}\).
- 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.

\section*{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 \(R \_m i d\) 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 \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1) - Length unit that \(R \_m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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).

\section*{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.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single \(\mathrm{r} / \mathrm{a}\) value at \(\mathrm{R} \_\mathrm{mid}=0.6 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

roa_val = Eq_instance.rmid2roa(0.6, 0.26)

```

Find roa values at \(\mathrm{R} \_\)mid points 0.6 m and 0.8 m at the single time \(\mathrm{t}=0.26 \mathrm{~s}\).:
roa_arr = Eq_instance.rmid2roa([0.6, 0.8], 0.26)

Find roa values at R_mid of 0.6 m at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find r/a values at ( \(\mathrm{R} \_\)mid, t ) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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

```
rmid2psinorm ( \(R \_m i d, t\), **kwargs)
Calculates the normalized poloidal flux corresponding to the passed R_mid (mapped outboard midplane major radius) values.

\section*{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 \(R \_m i d\).

\section*{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 \(R \_m i d\) or be a scalar. Default is True (evaluate ALL \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1 ) - Length unit that \(R \_m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single psinorm value for \(\mathrm{Rmid}=0.7 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

psinorm_val = Eq_instance.rmid2psinorm(0.7, 0.26)

```

Find psinorm values at \(R \_m i d\) values of 0.5 m and 0.7 m at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find psinorm values at \(R \_m i d=0.5 \mathrm{~m}\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at (R_mid, t) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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:
\[
\begin{aligned}
\text { phi } & =\int q(\psi) d \psi \\
\text { phi_norm } & =\frac{\phi}{\phi(a)}
\end{aligned}
\]

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

\section*{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 \(R \_m i d\).

\section*{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 \(R \_m i d\) 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 \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1) - Length unit that \(R \_m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single phinorm value at \(\mathrm{R} \_\mathrm{mid}=0.6 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

phi_val = Eq_instance.rmid2phinorm(0.6, 0.26)

```

Find phinorm values at \(R \_\)mid points 0.6 m and 0.8 m at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find phinorm values at \(\mathrm{R} \_\)mid point 0.6 m at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find phinorm values at \((\mathrm{R}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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.

\section*{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.

\section*{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 \(R \_m i d\) 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 \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1) - Length unit that \(R_{\_} m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single volnorm value at \(\mathrm{R} \_\)mid \(=0.6 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

vol_val = Eq_instance.rmid2volnorm(0.6, 0.26)

```

Find volnorm values at \(R \_m i d\) points 0.6 m and 0.8 m at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find volnorm values at \(\mathrm{R} \_\)mid points 0.6 m at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find volnorm values at ( \(\mathrm{R} \_\mathrm{mid}, \mathrm{t}\) ) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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

```
rmid2rho (method, \(R \_m i d, t\), **kwargs)
Convert the passed (R_mid, t) coordinates into one of several coordinates.

\section*{Parameters}
- method (String) - Indicates which coordinates to convert to. Valid options are:
\begin{tabular}{|l|l|}
\hline psinorm & Normalized poloidal flux \\
\hline phinorm & Normalized toroidal flux \\
\hline volnorm & Normalized volume \\
\hline r/a & Normalized minor radius \\
\hline F & Flux function \(F=R B_{\phi}\) \\
\hline FFPrime & Flux function \(F F^{\prime}\) \\
\hline p & Pressure \\
\hline pprime & Pressure gradient \\
\hline v & Flux surface volume \\
\hline
\end{tabular}

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.

\section*{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 \(R \_m i d\) or be a scalar. Default is True (evaluate ALL \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1) - Length unit that \(R \_m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single psinorm value at \(\mathrm{R} \_m i d=0.6 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find psinorm values at \(\mathrm{R} \_\)mid points 0.6 m and 0.8 m at the single time \(\mathrm{t}=0.26 \mathrm{~s}\).:
```

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

```

Find psinorm values at \(R \_\)mid of 0.6 m at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at ( \(\mathrm{R} \_\)mid, t ) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s}\) ) and ( \(0.5 \mathrm{~m}, 0.3 \mathrm{~s}\) ):
```

psi_arr = Eq_instance.rmid2rho('psinorm', [0.6, 0.5], [0.2, 0.3], each_
\hookrightarrowt=False)

```
roa 2 rmid (roa, \(t\), each_t=True, return_t=False, blob=None, length_unit=1)
Convert the passed ( \(\mathrm{r} / \mathrm{a}, \mathrm{t}\) ) coordinates into Rmid.

\section*{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.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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).

\section*{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.

\section*{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 \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

R_mid_val = Eq_instance.roa2rmid(0.6, 0.26)

```

Find \(R \_m i d\) values at \(r\) ra points 0.6 and 0.8 at the single time \(t=0.26 \mathrm{~s}\).:
```

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

```

Find \(R \_m i d\) values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find \(R \_m i d\) values at (roa, \(\left.t\right)\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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

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

Convert the passed ( \(\mathrm{r} / \mathrm{a}, \mathrm{t}\) ) coordinates into psinorm.

\section*{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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single psinorm value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

psinorm_val = Eq_instance.roa2psinorm(0.6, 0.26)

```

Find psinorm values at \(\mathrm{r} / \mathrm{a}\) points 0.6 and 0.8 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\).:
```

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

```

Find psinorm values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at (roa, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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

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

Convert the passed ( \(\mathrm{r} / \mathrm{a}, \mathrm{t}\) ) coordinates into phinorm.

\section*{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 \(\_\)keyword is False, \(t\) must have the same shape as roa.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single phinorm value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

phinorm_val = Eq_instance.roa2phinorm(0.6, 0.26)

```

Find phinorm values at \(\mathrm{r} /\) a points 0.6 and 0.8 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\).:
```

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

```

Find phinorm values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find phinorm values at (roa, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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

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

Convert the passed ( \(\mathrm{r} / \mathrm{a}, \mathrm{t}\) ) coordinates into volnorm.

\section*{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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single volnorm value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

volnorm_val = Eq_instance.roa2volnorm(0.6, 0.26)

```

Find volnorm values at \(\mathrm{r} / \mathrm{a}\) points 0.6 and 0.8 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\).:
```

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

```

Find volnorm values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find volnorm values at (roa, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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.

\section*{Parameters}
- method (String) - Indicates which coordinates to convert to. Valid options are:
\begin{tabular}{|l|l|}
\hline psinorm & Normalized poloidal flux \\
\hline phinorm & Normalized toroidal flux \\
\hline volnorm & Normalized volume \\
\hline Rmid & Midplane major radius \\
\hline q & Safety factor \\
\hline F & Flux function \(F=R B_{\phi}\) \\
\hline FFPrime & Flux function \(F F^{\prime}\) \\
\hline p & Pressure \\
\hline pprime & Pressure gradient \\
\hline v & Flux surface volume \\
\hline
\end{tabular}

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.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single psinorm value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(t=0.26 \mathrm{~s}\) :
```

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

```

Find psinorm values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at \((\mathrm{r} / \mathrm{a}, \mathrm{t})\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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.

\section*{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.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int)-The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibriumabstract class.
Find single R_mid value for psinorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

R_mid_val = Eq_instance.psinorm2rmid(0.7, 0.26)

```

Find \(\mathrm{R} \_\)mid values at \(\mathrm{psi} \_\)norm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

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

```

Find \(\mathrm{R} \_\)mid values at psi _norm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find \(R \_m i d\) values at (psinorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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.

\section*{Parameters}
- psi_norm(Array-like or scalar float)-Values of the normalized poloidal flux to map to \(\mathrm{r} / \mathrm{a}\).
- 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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single \(\mathrm{r} / \mathrm{a}\) value for \(\mathrm{psinorm}=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

roa_val = Eq_instance.psinorm2roa(0.7, 0.26)

```

Find \(\mathrm{r} / \mathrm{a}\) values at psi _norm values of 0.5 and 0.7 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find \(\mathrm{r} / \mathrm{a}\) values at \(\mathrm{psi} \_\)norm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find \(\mathrm{r} / \mathrm{a}\) values at \((\mathrm{psinorm}, \mathrm{t})\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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.

\section*{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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single volnorm value for \(\mathrm{psinorm}=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(t=0.26 \mathrm{~s}\) :
```

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

```

Find volnorm values at psi_norm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find volnorm values at (psinorm, t\()\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

volnorm_arr = Eq_instance.psinorm2volnorm([0.6, 0.5], [0.2, 0.3], each_
\hookrightarrow=False)

```
psinorm2phinorm (psi_norm, \(t\), **kwargs)
Calculates the normalized toroidal flux corresponding to the passed psi_norm (normalized poloidal flux) values.

\section*{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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{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 \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find phinorm values at psi_norm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find phinorm values at (psinorm, t) points ( \(0.6,0.2 \mathrm{~s}\) ) and ( \(0.5,0.3 \mathrm{~s}\) ):
```

phinorm_arr = Eq_instance.psinorm2phinorm([0.6, 0.5], [0.2, 0.3], each_
\hookrightarrow=False)

```
psinorm2rho (method, *args, **kwargs)
Convert the passed (psinorm, t ) coordinates into one of several coordinates.

\section*{Parameters}
- method (String) - Indicates which coordinates to convert to. Valid options are:
\begin{tabular}{|l|l|}
\hline phinorm & Normalized toroidal flux \\
\hline volnorm & Normalized volume \\
\hline Rmid & Midplane major radius \\
\hline r/a & Normalized minor radius \\
\hline q & Safety factor \\
\hline F & Flux function \(F=R B_{\phi}\) \\
\hline FFPrime & Flux function \(F F^{\prime}\) \\
\hline p & Pressure \\
\hline pprime & Pressure gradient \\
\hline v & Flux surface volume \\
\hline
\end{tabular}

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.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibriumabstract class.
Find single phinorm value at \(\mathrm{psinorm}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(t=0.26 \mathrm{~s}\) :
```

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

```

Find phinorm values at psinorm of 0.6 at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find phinorm values at (psinorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

phi_arr = Eq_instance.psinorm2rho('phinorm', [0.6, 0.5], [0.2, 0.3], each_
\hookrightarrow=False)

```
phinorm2psinorm(phinorm, \(t\), **kwargs)
Calculates the normalized poloidal flux corresponding to the passed phinorm (normalized toroidal flux) values.

\section*{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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single psinorm value for \(\mathrm{phinorm}=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find psinorm values at phinorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at (phinorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

psinorm_arr = Eq_instance.phinorm2psinorm([0.6, 0.5], [0.2, 0.3], each_
\hookrightarrow=False)

```
phinorm2volnorm (*args, **kwargs)
Calculates the normalized flux surface volume corresponding to the passed phinorm (normalized toroidal flux) values.

\section*{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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{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 \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find volnorm values at phinorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find volnorm values at (phinorm, t) points ( \(0.6,0.2 \mathrm{~s}\) ) and ( \(0.5,0.3 \mathrm{~s}\) ):
```

volnorm_arr = Eq_instance.phinorm2volnorm([0.6, 0.5], [0.2, 0.3], each_
G=False)

```
phinorm2rmid (*args, **kwargs)
Calculates the mapped outboard midplane major radius corresponding to the passed phinorm (normalized toroidal flux) values.

\section*{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.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{Returns}

\section*{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.

\section*{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 \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(t=0.26 \mathrm{~s}\) :
```

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

```

Find Rmid values at phinorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find Rmid values at (phinorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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.

\section*{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 \(\_\)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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single \(\mathrm{r} / \mathrm{a}\) value for phinorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

roa_val = Eq_instance.phinorm2roa(0.7, 0.26)

```

Find \(\mathrm{r} / \mathrm{a}\) values at phinorm values of 0.5 and 0.7 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find \(\mathrm{r} / \mathrm{a}\) values at phinorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find \(\mathrm{r} / \mathrm{a}\) values at (phinorm, t ) points ( \(0.6,0.2 \mathrm{~s}\) ) and ( \(0.5,0.3 \mathrm{~s}\) ):
```

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.

\section*{Parameters}
- method (String) - Indicates which coordinates to convert to. Valid options are:
\begin{tabular}{|l|l|}
\hline psinorm & Normalized poloidal flux \\
\hline volnorm & Normalized volume \\
\hline Rmid & Midplane major radius \\
\hline r/a & Normalized minor radius \\
\hline q & Safety factor \\
\hline F & Flux function \(F=R B_{\phi}\) \\
\hline FFPrime & Flux function \(F F^{\prime}\) \\
\hline p & Pressure \\
\hline pprime & Pressure gradient \\
\hline v & Flux surface volume \\
\hline
\end{tabular}

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.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{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 \(=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(t=0.26 \mathrm{~s}\) :
```

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

```

Find \(p\) sinorm values at phinorm of 0.6 at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at (phinorm, t ) points \((0.6,0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

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.

\section*{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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{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 \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(t=0.26 \mathrm{~s}\) :
```

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

```

Find psinorm values at volnorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at (volnorm, t\()\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

psinorm_arr = Eq_instance.volnorm2psinorm([0.6, 0.5], [0.2, 0.3], each_
\hookrightarrowt=False)

```
volnorm2phinorm (*args, **kwargs)
Calculates the normalized toroidal flux corresponding to the passed volnorm (normalized flux surface volume) values.

\section*{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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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. get TimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{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 \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find phinorm values at volnorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find phinorm values at (volnorm, t) points ( \(0.6,0.2 \mathrm{~s}\) ) and ( \(0.5,0.3 \mathrm{~s}\) ):
```

phinorm_arr = Eq_instance.volnorm2phinorm([0.6, 0.5], [0.2, 0.3], each_
๑t=False)

```
volnorm2rmid (*args, **kwargs)
Calculates the mapped outboard midplane major radius corresponding to the passed volnorm (normalized flux surface volume) values.

\section*{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.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single Rmid value for volnorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
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 \(t=0.26 \mathrm{~s}\) :
```

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

```

Find Rmid values at volnorm=0.5 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find Rmid values at (volnorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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.

\section*{Parameters}
- volnorm (Array-like or scalar float) - Values of the normalized flux surface volume to map to \(\mathrm{r} / \mathrm{a}\).
- 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.

\section*{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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{Examples}

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

Find single \(\mathrm{r} / \mathrm{a}\) value for volnorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

roa_val = Eq_instance.volnorm2roa(0.7, 0.26)

```

Find \(\mathrm{r} / \mathrm{a}\) values at volnorm values of 0.5 and 0.7 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

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

```

Find \(\mathrm{r} / \mathrm{a}\) values at volnorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find r/a values at (volnorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

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.

\section*{Parameters}
- method (String) - Indicates which coordinates to convert to. Valid options are:
\begin{tabular}{|l|l|}
\hline psinorm & Normalized poloidal flux \\
\hline phinorm & Normalized toroidal flux \\
\hline Rmid & Midplane major radius \\
\hline \(\mathrm{r} / \mathrm{a}\) & Normalized minor radius \\
\hline q & Safety factor \\
\hline F & Flux function \(F=R B_{\phi}\) \\
\hline FFPrime & Flux function \(F F^{\prime}\) \\
\hline p & Pressure \\
\hline pprime & Pressure gradient \\
\hline v & Flux surface volume \\
\hline
\end{tabular}

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.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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).

\section*{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.

\section*{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 \(=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

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 \(t=0.26 \mathrm{~s}\) :
```

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

```

Find psinorm values at volnorm of 0.6 at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

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

```

Find psinorm values at (volnorm, t\()\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

psi_arr = Eq_instance.volnorm2rho('psinorm', [0.6, 0.5], [0.2, 0.3], each_

```
    \(\hookrightarrow t=\) False)
rz2q ( \(R, Z, t\), **kwargs \()\)
Calculates the safety factor (" \(q\) ") at the given ( \(R, Z, t\) ).
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- \(\mathbf{R}\) (Array-like or scalar float) - Values of the radial coordinate to map to \(q\). 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 q. 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).

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of q. 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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 ( \(q\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(q\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(q\) ).

\section*{Returns}
\(q\) or ( \(q\), time_idxs)
- \(\mathbf{q}\) (Array or scalar float) - The safety factor (" \(q\) "). 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 \(q\) has this shape as well, unless the make_grid keyword was True, in which case \(q\) has shape \((\operatorname{len}(Z)\), len \((R))\).
- time_idxs (Array with same shape as \(q\) ) - The indices (in self.get TimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single q value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

q_val = Eq_instance.rz2q(0.6, 0, 0.26)

```

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

q_arr = Eq_instance.rz2q([0.6, 0.8], [0, 0], 0.26)

```

Find \(q\) values at \((R, Z)\) points \((0.6 m, 0 m)\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

q_arr = Eq_instance.rz2q(0.6, 0, [0.2, 0.3])

```

Find q values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

q_arr = Eq_instance.rz2q([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find q values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

q_mat = Eq_instance.rz2q(R, Z, 0.2, make_grid=True)

```
rmid2q ( \(R\) _mid, \(t\), **kwargs)
Calculates the safety factor (" \(q\) ") corresponding to the passed R_mid (mapped outboard midplane major radius) values.

By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R_mid(Array-like or scalar float) - Values of the outboard midplane major radius to map to \(q\).
- 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 \(R \_m i d\).

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of q. 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 \(R \_m i d\) 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 \(R \_m i d\) or be a scalar. Default is True (evaluate ALL \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1) - Length unit that \(R \_m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (q, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(q\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(q\) ).

\section*{Returns}
\(q\) or ( \(q\), time_idxs)
- \(\mathbf{q}\) (Array or scalar float) - The safety factor (" \(q\) "). 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 \(q\) ) - The indices (in self.getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single q value for \(\mathrm{Rmid}=0.7 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

q_val = Eq_instance.rmid2q(0.7, 0.26)

```

Find \(q\) values at \(R \_m i d\) values of 0.5 m and 0.7 m at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

q_arr = Eq_instance.rmid2q([0.5, 0.7], 0.26)

```

Find \(q\) values at \(R \_m i d=0.5 \mathrm{~m}\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

q_arr = Eq_instance.rmid2q(0.5, [0.2, 0.3])

```

Find \(q\) values at \(\left(\mathrm{R} \_m i d, t\right)\) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

q_arr = Eq_instance.rmid2q([0.6, 0.5], [0.2, 0.3], each_t=False)

```
roa2q (roa, \(t\), **kwargs)
Convert the passed (r/a, t) coordinates into safety factor ("q").
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- roa (Array-like or scalar float) - Values of the normalized minor radius to map to q.
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of q. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(q\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(q\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(q\) ).

\section*{Returns}
\(q\) or ( \(q\), time_idxs)
- \(\mathbf{q}\) (Array or scalar float) - The safety factor ("q"). 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 \(q\) ) - The indices (in self.getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single q value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

q_val = Eq_instance.roa2q(0.6, 0.26)

```

Find q values at \(\mathrm{r} / \mathrm{a}\) points 0.6 and 0.8 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\).:
```

q_arr = Eq_instance.roa2q([0.6, 0.8], 0.26)

```

Find \(q\) values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

q_arr = Eq_instance.roa2q(0.6, [0.2, 0.3])

```

Find \(q\) values at (roa, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

q_arr = Eq_instance.roa2q([0.6, 0.5], [0.2, 0.3], each_t=False)

```
psinorm2q (psinorm, \(t\), **kwargs)
Calculates the safety factor (" \(q\) ") corresponding to the passed psi _norm (normalized poloidal flux) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- psi_norm (Array-like or scalar float)- Values of the normalized poloidal flux to map to \(q\).
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of q. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (q, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(q\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(q\) ).

\section*{Returns}
\(q\) or ( \(q\), time_idxs)
- q(Array or scalar float) - The safety factor ("q"). 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 \(q\) ) - The indices (in self.getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single q value for \(\mathrm{psinorm}=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

q_val = Eq_instance.psinorm2q(0.7, 0.26)

```

Find q values at psi _norm values of 0.5 and 0.7 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

q_arr = Eq_instance.psinorm2q([0.5, 0.7], 0.26)

```

Find \(q\) values at psi_norm \(=0.5\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

q_arr = Eq_instance.psinorm2q(0.5, [0.2, 0.3])

```

Find \(q\) values at (psinorm, t ) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

q_arr = Eq_instance.psinorm2q([0.6, 0.5], [0.2, 0.3], each_t=False)

```
phinorm2q (phinorm, \(t\), **kwargs)
Calculates the safety factor ("q") corresponding to the passed phinorm (normalized toroidal flux) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- phinorm (Array-like or scalar float) - Values of the normalized toroidal flux to map to \(q\).
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of q. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (q, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(q\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(q\) ).

\section*{Returns}
\(q\) or ( \(q\), time_idxs)
- q (Array or scalar float) - The safety factor ("q"). 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 \(q\) ) - The indices (in self.get TimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single \(q\) value for phinorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

q_val = Eq_instance.phinorm2q(0.7, 0.26)

```

Find \(q\) values at phinorm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

q_arr = Eq_instance.phinorm2q([0.5, 0.7], 0.26)

```

Find \(q\) values at phinorm \(=0.5\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

q_arr = Eq_instance.phinorm2q(0.5, [0.2, 0.3])

```

Find q values at (phinorm, t ) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
q_arr = Eq_instance.phinorm2q([0.6, 0.5], [0.2, 0.3], each_t=False)
volnorm2q (volnorm, \(t\), **kwargs)
Calculates the safety factor (" q ") corresponding to the passed volnorm (normalized flux surface volume) values.

By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- volnorm (Array-like or scalar float) - Values of the normalized flux surface volume to map to q .
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of q. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(q\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(q\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(q\) ).

\section*{Returns}
\(q\) or ( \(q\), time_idxs)
- \(\mathbf{q}\) (Array or scalar float) - The safety factor ("q"). 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 \(q\) ) - The indices (in self.getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single q value for volnorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

q_val = Eq_instance.volnorm2q(0.7, 0.26)

```

Find \(q\) values at volnorm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

q_arr = Eq_instance.volnorm2q([0.5, 0.7], 0.26)

```

Find \(q\) values at volnorm \(=0.5\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

q_arr = Eq_instance.volnorm2q(0.5, [0.2, 0.3])

```

Find q values at (volnorm, t ) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

q_arr = Eq_instance.volnorm2q([0.6, 0.5], [0.2, 0.3], each_t=False)

```
rz2F ( \(R, Z, t\), **kwargs)
Calculates the flux function \(F=R B_{\phi}\) at the given (R, \(\mathrm{Z}, \mathrm{t}\) ).
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R (Array-like or scalar float) - Values of the radial coordinate to map to F. 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 F. 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).

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of F. 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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (F, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(F\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(F\) ).

\section*{Returns}
\(F\) or (F, time_idxs)
- \(\mathbf{F}\) (Array or scalar float) - The flux function \(F=R B_{\phi}\). 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 \(F\) has this shape as well, unless the make_grid keyword was True, in which case \(F\) has shape \((\operatorname{len}(Z)\), \(\operatorname{len}(R))\).
- time_idxs (Array with same shape as \(F\) ) - The indices (in self.getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single F value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

F_val = Eq_instance.rz2F(0.6, 0, 0.26)

```

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

F_arr = Eq_instance.rz2F([0.6, 0.8], [0, 0], 0.26)

```

Find \(F\) values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

F_arr = Eq_instance.rz2F(0.6, 0, [0.2, 0.3])

```

Find F values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

F_arr = Eq_instance.rz2F([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find F values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

F_mat = Eq_instance.rz2F(R, Z, 0.2, make_grid=True)

```

\section*{rmid2F ( \(R\) _mid, \(t\), **kwargs)}

Calculates the flux function \(F=R B_{\phi}\) corresponding to the passed R_mid (mapped outboard midplane major radius) values.

By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R_mid(Array-like or scalar float) - Values of the outboard midplane major radius to map to \(F\).
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of F. 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 \(R \_m i d\) 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 \(R \_m i d\) or be a scalar. Default is True (evaluate ALL \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1) - Length unit that \(R \_m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(F\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(F\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(F\) ).

\section*{Returns}
\(F\) or ( \(F\), time_idxs)
- F (Array or scalar float) - The flux function \(F=R B_{\phi}\). 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 \(F\) ) - The indices (in self.getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single F value for \(\mathrm{Rmid}=0.7 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

F_val = Eq_instance.rmid2F(0.7, 0.26)

```

Find \(F\) values at \(R \_m i d\) values of 0.5 m and 0.7 m at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

F_arr = Eq_instance.rmid2F([0.5, 0.7], 0.26)

```

Find \(F\) values at \(R \_m i d=0.5 \mathrm{~m}\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

F_arr = Eq_instance.rmid2F(0.5, [0.2, 0.3])

```

Find F values at \(\left(\mathrm{R} \_m i d, t\right)\) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

F_arr = Eq_instance.rmid2F([0.6, 0.5], [0.2, 0.3], each_t=False)

```
roa2F (roa, \(t\), **kwargs)
Convert the passed (r/a, t) coordinates into the flux function \(F=R B_{\phi}\).
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- roa (Array-like or scalar float) - Values of the normalized minor radius to map to F .
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of F. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(F\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(F\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(F\) ).

\section*{Returns}
\(F\) or (F, time_idxs)
- F (Array or scalar float) - The flux function \(F=R B_{\phi}\). 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 \(F\) ) - The indices (in self.getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single F value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

F_val = Eq_instance.roa2F(0.6, 0.26)

```

Find F values at \(\mathrm{r} /\) a points 0.6 and 0.8 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\).:
```

F_arr = Eq_instance.roa2F([0.6, 0.8], 0.26)

```

Find F values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

F_arr = Eq_instance.roa2F(0.6, [0.2, 0.3])

```

Find F values at (roa, t\()\) points ( \(0.6,0.2 \mathrm{~s}\) ) and ( \(0.5,0.3 \mathrm{~s}\) ):
```

F_arr = Eq_instance.roa2F([0.6, 0.5], [0.2, 0.3], each_t=False)

```
psinorm2F (psinorm, \(t\), **kwargs)
Calculates the flux function \(F=R B_{\phi}\) corresponding to the passed psi_norm (normalized poloidal flux) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- psi_norm(Array-like or scalar float) - Values of the normalized poloidal flux to map to F .
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of F. 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 \(t\) ).
- \(\mathbf{k}\) (positive int)- The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(F\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(F\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(F\) ).

\section*{Returns}
\(F\) or ( \(F\), time_idxs)
- F (Array or scalar float) - The flux function \(F=R B_{\phi}\). 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 \(F\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single F value for \(\mathrm{psinorm}=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

F_val = Eq_instance.psinorm2F(0.7, 0.26)

```

Find F values at psi_norm values of 0.5 and 0.7 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

F_arr = Eq_instance.psinorm2F([0.5, 0.7], 0.26)

```

Find F values at psi_norm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

F_arr = Eq_instance.psinorm2F(0.5, [0.2, 0.3])

```

Find F values at (psinorm, t\()\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

F_arr = Eq_instance.psinorm2F([0.6, 0.5], [0.2, 0.3], each_t=False)

```
phinorm2F (phinorm, \(t\), **kwargs)
Calculates the flux function \(F=R B_{\phi}\) corresponding to the passed phinorm (normalized toroidal flux) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- phinorm (Array-like or scalar float) - Values of the normalized toroidal flux to map to \(F\).
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of F. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(F\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(F\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(F\) ).

\section*{Returns}
\(F\) or ( \(F\), time_idxs)
- \(\mathbf{F}\) (Array or scalar float) - The flux function \(F=R B_{\phi}\). 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 \(F\) ) - The indices (in self. getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single F value for phinorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

F_val = Eq_instance.phinorm2F(0.7, 0.26)

```

Find F values at phinorm values of 0.5 and 0.7 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

F_arr = Eq_instance.phinorm2F([0.5, 0.7], 0.26)

```

Find F values at phinorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

F_arr = Eq_instance.phinorm2F(0.5, [0.2, 0.3])

```

Find F values at (phinorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

F_arr = Eq_instance.phinorm2F([0.6, 0.5], [0.2, 0.3], each_t=False)

```
volnorm2F (volnorm, \(t\), **kwargs)
Calculates the flux function \(F=R B_{\phi}\) corresponding to the passed volnorm (normalized flux surface volume) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- volnorm (Array-like or scalar float) - Values of the normalized flux surface volume to map to F .
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of F. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(F\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(F\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(F\) ).

\section*{Returns}
\(F\) or ( \(F\), time_idxs)
- \(\mathbf{F}\) (Array or scalar float) - The flux function \(F=R B_{\phi}\). 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 \(F\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single F value for volnorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

F_val = Eq_instance.volnorm2F(0.7, 0.26)

```

Find \(F\) values at volnorm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

F_arr = Eq_instance.volnorm2F([0.5, 0.7], 0.26)

```

Find F values at volnorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

F_arr = Eq_instance.volnorm2F(0.5, [0.2, 0.3])

```

Find F values at (volnorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

F_arr = Eq_instance.volnorm2F([0.6, 0.5], [0.2, 0.3], each_t=False)

```

Fnorm2psinorm ( \(F, t, * * k w a r g s\) )
Calculates the psinorm (normalized poloidal flux) corresponding to the passed normalized flux function \(F=R B_{\phi}\) values.

This is provided as a convenience method to plot current lines with the correct spacing: current lines launched from a grid uniformly-spaced in Fnorm will have spacing directly proportional to the magnitude.
By default, EFIT only computes this inside the LCFS. Furthermore, it is truncated at the radius at which is becomes non-monotonic.

\section*{Parameters}
- \(\boldsymbol{F}\) (Array-like or scalar float) - Values of F 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.

\section*{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 \(F\) 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 \(F\) or be a scalar. Default is True (evaluate ALL volnorm at EACH element in \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (psinorm, time_idxs), where time_idxs is the array of time indices actually used in evaluating psinorm with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return psinorm).

\section*{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.
- 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.

\section*{Examples}

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

Find single psinorm value for \(\mathrm{F}=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

psinorm_val = Eq_instance.F2psinorm(0.7, 0.26)

```

Find psinorm values at F values of 0.5 and 0.7 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

psinorm_arr = Eq_instance.F2psinorm([0.5, 0.7], 0.26)

```

Find psinorm values at \(\mathrm{F}=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

psinorm_arr = Eq_instance.F2psinorm(0.5, [0.2, 0.3])

```

Find psinorm values at \((\mathrm{F}, \mathrm{t})\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

psinorm_arr = Eq_instance.F2psinorm([0.6, 0.5], [0.2, 0.3], each_t=False)

```
rz2FFPrime ( \(R, Z, t\), **kwargs)
Calculates the flux function \(F F^{\prime}\) at the given \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\).
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R (Array-like or scalar float) - Values of the radial coordinate to map to FFPrime. 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 FFPrime. 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 \(\_\)keyword is False, \(t\) must have the same shape as \(R\) and \(Z\) (or their meshgrid if make_grid is True).

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of FFPrime. 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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (FFPrime, time_idxs), where time_idxs is the array of time indices actually used in evaluating FFPrime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return FFPrime).

\section*{Returns}

FFPrime or (FFPrime, time_idxs)
- FFPrime (Array or scalar float) - The flux function \(F F^{\prime}\). 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 FFPrime has this shape as well, unless the make_grid keyword was True, in which case FFPrime has shape \((\operatorname{len}(Z)\), len \((R)\) ).
- time_idxs (Array with same shape as FFPrime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single FFPrime value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

FFPrime_val = Eq_instance.rz2FFPrime(0.6, 0, 0.26)

```

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

FFPrime_arr = Eq_instance.rz2FFPrime([0.6, 0.8], [0, 0], 0.26)

```

Find FFPrime values at \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

FFPrime_arr = Eq_instance.rz2FFPrime(0.6, 0, [0.2, 0.3])

```

Find FFPrime values at \((R, Z, t)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

FFPrime_arr = Eq_instance.rz2FFPrime([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_
๑=False)

```

Find FFPrime values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

FFPrime_mat = Eq_instance.rz2FFPrime(R, Z, 0.2, make_grid=True)

```
rmid2FFPrime ( \(R \_\)mid, \(t\), **kwargs)

Calculates the flux function \(F F^{\prime}\) corresponding to the passed R_mid (mapped outboard midplane major radius) values.

By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R_mid(Array-like or scalar float) - Values of the outboard midplane major radius to map to FFPrime.
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of FFPrime. 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 \(R \_m i d\) or be a scalar. Default is True (evaluate ALL \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1) - Length unit that \(R \_m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (FFPrime, time_idxs), where time_idxs is the array of time indices actually used in evaluating FFPrime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return FFPrime).

\section*{Returns}

FFPrime or (FFPrime, time_idxs)
- FFPrime (Array or scalar float) - The flux function \(F F^{\prime}\). 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 FFPrime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single FFPrime value for \(\mathrm{Rmid}=0.7 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
FFPrime_val = Eq_instance.rmid2FFPrime (0.7, 0.26)
Find FFPrime values at R_mid values of 0.5 m and 0.7 m at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

FFPrime_arr = Eq_instance.rmid2FFPrime([0.5, 0.7], 0.26)

```

Find FFPrime values at \(R \_\)mid \(=0.5 \mathrm{~m}\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

FFPrime_arr = Eq_instance.rmid2FFPrime(0.5, [0.2, 0.3])

```

Find FFPrime values at ( \(\mathrm{R} \_\mathrm{mid}, \mathrm{t}\) ) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

FFPrime_arr = Eq_instance.rmid2FFPrime([0.6, 0.5], [0.2, 0.3], each_t=False)

```
roa2FFPrime (roa, \(t\), **kwargs)
Convert the passed ( \(\mathrm{r} / \mathrm{a}, \mathrm{t}\) ) coordinates into the flux function \(F F^{\prime}\).
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- roa (Array-like or scalar float)-Values of the normalized minor radius to map to FFPrime.
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of FFPrime. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (FFPrime, time_idxs), where time_idxs is the array of time indices actually used in evaluating FFPrime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return FFPrime).

\section*{Returns}

FFPrime or (FFPrime, time_idxs)
- FFPrime (Array or scalar float) - The flux function \(F F^{\prime}\). 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 FFPrime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single FFPrime value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

FFPrime_val = Eq_instance.roa2FFPrime(0.6, 0.26)

```

Find FFPrime values at \(\mathrm{r} / \mathrm{a}\) points 0.6 and 0.8 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\).:
```

FFPrime_arr = Eq_instance.roa2FFPrime([0.6, 0.8], 0.26)

```

Find FFPrime values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

FFPrime_arr = Eq_instance.roa2FFPrime(0.6, [0.2, 0.3])

```

Find FFPrime values at (roa, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

FFPrime_arr = Eq_instance.roa2FFPrime([0.6, 0.5], [0.2, 0.3], each_t=False)

```
psinorm2FFPrime (psinorm, \(t\), **kwargs)

Calculates the flux function \(F F^{\prime}\) corresponding to the passed psi_norm (normalized poloidal flux) values. By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- psi_norm(Array-like or scalar float) - Values of the normalized poloidal flux to map to FFPrime.
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of FFPrime. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (FFPrime, time_idxs), where time_idxs is the array of time indices actually used in evaluating FFPrime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return FFPrime).

\section*{Returns}

FFPrime or (FFPrime, time_idxs)
- FFPrime (Array or scalar float) - The flux function \(F F^{\prime}\). 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 FFPrime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single FFPrime value for \(\mathrm{psinorm}=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

FFPrime_val = Eq_instance.psinorm2FFPrime(0.7, 0.26)

```

Find FFPrime values at psi_norm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

FFPrime_arr = Eq_instance.psinorm2FFPrime([0.5, 0.7], 0.26)

```

Find FFPrime values at psi_norm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

FFPrime_arr = Eq_instance.psinorm2FFPrime(0.5, [0.2, 0.3])

```

Find FFPrime values at (psinorm, t\()\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

FFPrime_arr = Eq_instance.psinorm2FFPrime([0.6, 0.5], [0.2, 0.3], each_
\hookrightarrow=False)

```
phinorm2FFPrime (phinorm, \(t\), **kwargs)
Calculates the flux function \(F F^{\prime}\) corresponding to the passed phinorm (normalized toroidal flux) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- phinorm (Array-like or scalar float) - Values of the normalized toroidal flux to map to FFPrime.
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of FFPrime. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (FFPrime, time_idxs), where time_idxs is the array of time indices actually used in evaluating FFPrime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return FFPrime).

\section*{Returns}

FFPrime or (FFPrime, time_idxs)
- FFPrime (Array or scalar float) - The flux function \(F F^{\prime}\). 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 FFPrime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibriumabstract class.
Find single FFPrime value for phinorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

FFPrime_val = Eq_instance.phinorm2FFPrime(0.7, 0.26)

```

Find FFPrime values at phinorm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

FFPrime_arr = Eq_instance.phinorm2FFPrime([0.5, 0.7], 0.26)

```

Find FFPrime values at phinorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

FFPrime_arr = Eq_instance.phinorm2FFPrime(0.5, [0.2, 0.3])

```

Find FFPrime values at (phinorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

FFPrime_arr = Eq_instance.phinorm2FFPrime([0.6, 0.5], [0.2, 0.3], each_
\hookrightarrow=False)

```
volnorm2FFPrime (volnorm, \(t\), **kwargs)
Calculates the flux function \(F F^{\prime}\) corresponding to the passed volnorm (normalized flux surface volume) values.

By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- volnorm (Array-like or scalar float) - Values of the normalized flux surface volume to map to FFPrime.
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of FFPrime. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (FFPrime, time_idxs), where time_idxs is the array of time indices actually used in evaluating FFPrime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return FFPrime).

\section*{Returns}

FFPrime or (FFPrime, time_idxs)
- FFPrime (Array or scalar float) - The flux function \(F F^{\prime}\). 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 FFPrime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single FFPrime value for volnorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

FFPrime_val = Eq_instance.volnorm2FFPrime(0.7, 0.26)

```

Find FFPrime values at volnorm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

FFPrime_arr = Eq_instance.volnorm2FFPrime([0.5, 0.7], 0.26)

```

Find FFPrime values at volnorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

FFPrime_arr = Eq_instance.volnorm2FFPrime(0.5, [0.2, 0.3])

```

Find FFPrime values at (volnorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

FFPrime_arr = Eq_instance.volnorm2FFPrime([0.6, 0.5], [0.2, 0.3], each_
\hookrightarrow=False)

```
rz2p ( \(R, Z, t\), **kwargs)
Calculates the pressure at the given \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\).

By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R (Array-like or scalar float) - Values of the radial coordinate to map to p. 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 p. 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).

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of p. 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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 ( \(p\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(p\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(p\) ).

\section*{Returns}
\(p\) or ( \(p\), time_idxs)
- p (Array or scalar float) - The pressure. 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 \(p\) has this shape as well, unless the make_grid keyword was True, in which case \(p\) has shape \((\operatorname{len}(Z)\), len \((R))\).
- time_idxs (Array with same shape as \(p\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single p value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

p_val= Eq_instance.rz2p(0.6, 0, 0.26)

```

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

p_arr = Eq_instance.rz2p([0.6, 0.8], [0, 0], 0.26)

```

Find \(p\) values at \((R, Z)\) points \((0.6 m, 0 m)\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

p_arr = Eq_instance.rz2p(0.6, 0, [0.2, 0.3])

```

Find p values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

p_arr = Eq_instance.rz2p([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find p values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

p_mat = Eq_instance.rz2p(R, Z, 0.2, make_grid=True)

```
rmid2p ( \(R\) _mid, \(t\), **kwargs)
Calculates the pressure corresponding to the passed R_mid (mapped outboard midplane major radius) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R_mid(Array-like or scalar float) - Values of the outboard midplane major radius to map to p .
- 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 \(R \_m i d\).

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of p. 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 \(R \_m i d\) 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 \(R \_m i d\) or be a scalar. Default is True (evaluate ALL \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1) - Length unit that \(R \_m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(p\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(p\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(p\) ).

\section*{Returns}
\(p\) or ( \(p\), time_idxs)
- p (Array or scalar float) - The pressure. 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 \(p\) ) - The indices (in self.getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single \(p\) value for \(\mathrm{Rmid}=0.7 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

p_val = Eq_instance.rmid2p(0.7, 0.26)

```

Find \(p\) values at \(R \_m i d\) values of 0.5 m and 0.7 m at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

p_arr = Eq_instance.rmid2p([0.5, 0.7], 0.26)

```

Find p values at \(\mathrm{R} \_\)mid \(=0.5 \mathrm{~m}\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

p_arr = Eq_instance.rmid2p(0.5, [0.2, 0.3])

```

Find p values at ( \(\mathrm{R} \_\)mid, t ) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

p_arr = Eq_instance.rmid2p([0.6, 0.5], [0.2, 0.3], each_t=False)

```
roa2p (roa, \(t\), **kwargs)

Convert the passed (r/a, t) coordinates into pressure.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
-roa (Array-like or scalar float)-Values of the normalized minor radius to map to p .
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of p . 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(p\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(p\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(p\) ).

\section*{Returns}
\(p\) or ( \(p\), time_idxs)
- p (Array or scalar float) - The pressure. 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 \(p\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single p value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

p_val = Eq_instance.roa2p(0.6, 0.26)

```

Find p values at \(\mathrm{r} / \mathrm{a}\) points 0.6 and 0.8 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\).:
```

p_arr = Eq_instance.roa2p([0.6, 0.8], 0.26)

```

Find p values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

p_arr = Eq_instance.roa2p(0.6, [0.2, 0.3])

```

Find \(p\) values at (roa, \(t)\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

p_arr = Eq_instance.roa2p([0.6, 0.5], [0.2, 0.3], each_t=False)

```
psinorm2p (psinorm, \(t\), **kwargs)
Calculates the pressure corresponding to the passed psi_norm (normalized poloidal flux) values.

By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- psi_norm(Array-like or scalar float) - Values of the normalized poloidal flux to map to p .
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of p. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(p\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(p\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(p\) ).

\section*{Returns}
\(p\) or ( \(p\), time_idxs)
- p (Array or scalar float) - The pressure. 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 \(p\) ) - The indices (in self.getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single \(p\) value for \(p \operatorname{sinorm}=0.7, t=0.26 s\) :
p_val = Eq_instance.psinorm2p(0.7, 0.26)
Find \(p\) values at psi_norm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

p_arr = Eq_instance.psinorm2p([0.5, 0.7], 0.26)

```

Find p values at psi norm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

p_arr = Eq_instance.psinorm2p(0.5, [0.2, 0.3])

```

Find \(p\) values at (psinorm, \(t\) ) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

p_arr = Eq_instance.psinorm2p([0.6, 0.5], [0.2, 0.3], each_t=False)

```
```

phinorm2p (phinorm, $t$, **kwargs)

```

Calculates the pressure corresponding to the passed phinorm (normalized toroidal flux) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- phinorm (Array-like or scalar float) - Values of the normalized toroidal flux to map to \(p\).
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of p. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(p\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(p\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(p\) ).

\section*{Returns}
\(p\) or ( \(p\), time_idxs)
- p (Array or scalar float) - The pressure. 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 \(p\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single p value for \(\mathrm{phinorm}=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

p_val = Eq_instance.phinorm2p(0.7, 0.26)

```

Find \(p\) values at phinorm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

p_arr = Eq_instance.phinorm2p([0.5, 0.7], 0.26)

```

Find p values at \(\mathrm{phinorm}=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

p_arr = Eq_instance.phinorm2p(0.5, [0.2, 0.3])

```

Find p values at (phinorm, t ) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
p_arr = Eq_instance.phinorm2p([0.6, 0.5], [0.2, 0.3], each_t=False)
volnorm2p (volnorm, \(t\), **kwargs)
Calculates the pressure corresponding to the passed volnorm (normalized flux surface volume) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- volnorm (Array-like or scalar float) - Values of the normalized flux surface volume to map to p .
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of p. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(p\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(p\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(p\) ).

\section*{Returns}
\(p\) or ( \(p\), time_idxs)
- p (Array or scalar float) - The pressure. 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 \(p\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibriumabstract class.
Find single \(p\) value for volnorm \(=0.7, t=0.26 s\) :
```

p_val = Eq_instance.volnorm2p(0.7, 0.26)

```

Find p values at volnorm values of 0.5 and 0.7 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

p_arr = Eq_instance.volnorm2p([0.5, 0.7], 0.26)

```

Find \(p\) values at volnorm \(=0.5\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

p_arr = Eq_instance.volnorm2p(0.5, [0.2, 0.3])

```

Find p values at (volnorm, t\()\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

p_arr = Eq_instance.volnorm2p([0.6, 0.5], [0.2, 0.3], each_t=False)

```
rz2pprime ( \(R, Z, t\), **kwargs)
Calculates the pressure gradient at the given \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\).
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R (Array-like or scalar float) - Values of the radial coordinate to map to pprime. 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 pprime. 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).

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of pprime. 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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (pprime, time_idxs), where time_idxs is the array of time indices actually used in evaluating pprime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return pprime).

\section*{Returns}
pprime or (pprime, time_idxs)
- pprime (Array or scalar float) - The pressure gradient. 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 \(p\) has this shape as well, unless the make_grid keyword was True, in which case \(p\) has shape \((\operatorname{len}(Z)\), len \((R)\) ).
- time_idxs (Array with same shape as pprime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single pprime value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

pprime_val = Eq_instance.rz2pprime(0.6, 0, 0.26)

```

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

pprime_arr = Eq_instance.rz2pprime([0.6, 0.8], [0, 0], 0.26)

```

Find pprime values at \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

pprime_arr = Eq_instance.rz2pprime(0.6, 0, [0.2, 0.3])

```

Find pprime values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

pprime_arr = Eq_instance.rz2pprime([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_
->=False)

```

Find pprime values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

pprime_mat = Eq_instance.rz2pprime(R, Z, 0.2, make_grid=True)

```
```

rmid2pprime (R_mid, t, **kwargs)

```

Calculates the pressure gradient corresponding to the passed \(R \_\)mid (mapped outboard midplane major radius) values.

By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R_mid(Array-like or scalar float) - Values of the outboard midplane major radius to map to pprime.
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of pprime. 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 \(R \_m i d\) 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 \(R \_m i d\) or be a scalar. Default is True (evaluate ALL \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1) - Length unit that \(R \_m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (pprime, time_idxs), where time_idxs is the array of time indices actually used in evaluating pprime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return pprime).

\section*{Returns}
pprime or (pprime, time_idxs)
- pprime (Array or scalar float) - The pressure gradient. 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 pprime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single pprime value for \(\mathrm{Rmid}=0.7 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

pprime_val = Eq_instance.rmid2pprime(0.7, 0.26)

```

Find pprime values at \(R \_\)mid values of 0.5 m and 0.7 m at the single time \(t=0.26 \mathrm{~s}\) :
```

pprime_arr = Eq_instance.rmid2pprime([0.5, 0.7], 0.26)

```

Find pprime values at \(\mathrm{R} \_\)mid \(=0.5 \mathrm{~m}\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

pprime_arr = Eq_instance.rmid2pprime(0.5, [0.2, 0.3])

```

Find pprime values at ( \(\mathrm{R} \_\)mid, t ) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

pprime_arr = Eq_instance.rmid2pprime([0.6, 0.5], [0.2, 0.3], each_t=False)

```
roa2pprime (roa, \(t\), **kwargs)

Convert the passed ( \(\mathrm{r} / \mathrm{a}, \mathrm{t}\) ) coordinates into pressure gradient.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- roa (Array-like or scalar float) - Values of the normalized minor radius to map to pprime.
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of pprime. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (pprime, time_idxs), where time_idxs is the array of time indices actually used in evaluating pprime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return pprime).

\section*{Returns}
pprime or (pprime, time_idxs)
- pprime (Array or scalar float) - The pressure gradient. 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 pprime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return \(\_t\) is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single pprime value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

pprime_val = Eq_instance.roa2pprime(0.6, 0.26)

```

Find pprime values at \(\mathrm{r} / \mathrm{a}\) points 0.6 and 0.8 at the single time \(\mathrm{t}=0.26 \mathrm{~s}\).:
```

pprime_arr = Eq_instance.roa2pprime([0.6, 0.8], 0.26)

```

Find pprime values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

pprime_arr = Eq_instance.roa2pprime(0.6, [0.2, 0.3])

```

Find pprime values at (roa, t) points ( \(0.6,0.2 \mathrm{~s}\) ) and ( \(0.5,0.3 \mathrm{~s}\) ):
```

pprime_arr = Eq_instance.roa2pprime([0.6, 0.5], [0.2, 0.3], each_t=False)

```
psinorm2pprime (psinorm, \(t\), **kwargs)
Calculates the pressure gradient corresponding to the passed psi_norm (normalized poloidal flux) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- psi_norm (Array-like or scalar float)- Values of the normalized poloidal flux to map to pprime.
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of pprime. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (pprime, time_idxs), where time_idxs is the array of time indices actually used in evaluating pprime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return pprime).

\section*{Returns}
pprime or (pprime, time_idxs)
- pprime (Array or scalar float) - The pressure gradient. 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 pprime) - The indices (in self. getTimeBase()) that were used for nearest-neighbor interpolation. Only returned if return \(\_t\) is True.

\section*{Examples}

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

Find single pprime value for \(\mathrm{psinorm}=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

pprime_val = Eq_instance.psinorm2pprime(0.7, 0.26)

```

Find pprime values at psi_norm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

pprime_arr = Eq_instance.psinorm2pprime([0.5, 0.7], 0.26)

```

Find pprime values at psi_norm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

pprime_arr = Eq_instance.psinorm2pprime(0.5, [0.2, 0.3])

```

Find pprime values at (psinorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

pprime_arr = Eq_instance.psinorm2pprime([0.6, 0.5], [0.2, 0.3], each_t=False)

```
phinorm2pprime (phinorm, \(t\), **kwargs)
Calculates the pressure gradient corresponding to the passed phinorm (normalized toroidal flux) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- phinorm (Array-like or scalar float) - Values of the normalized toroidal flux to map to pprime.
- 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 \(\_\)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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of pprime. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (pprime, time_idxs), where time_idxs is the array of time indices actually used in evaluating pprime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return pprime).

\section*{Returns}
pprime or (pprime, time_idxs)
- pprime (Array or scalar float) - The pressure gradient. 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 pprime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single pprime value for phinorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

pprime_val = Eq_instance.phinorm2pprime(0.7, 0.26)

```

Find pprime values at phinorm values of 0.5 and 0.7 at the single time \(t=0.26\) s:
```

pprime_arr = Eq_instance.phinorm2pprime([0.5, 0.7], 0.26)

```

Find pprime values at phinorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

pprime_arr = Eq_instance.phinorm2pprime(0.5, [0.2, 0.3])

```

Find pprime values at (phinorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

pprime_arr = Eq_instance.phinorm2pprime([0.6, 0.5], [0.2, 0.3], each_t=False)

```
volnorm2pprime (volnorm, \(t\), **kwargs)
Calculates the pressure gradient corresponding to the passed volnorm (normalized flux surface volume) values.

By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- volnorm (Array-like or scalar float) - Values of the normalized flux surface volume to map to pprime.
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of pprime. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of (pprime, time_idxs), where time_idxs is the array of time indices actually used in evaluating pprime with nearestneighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return pprime).

\section*{Returns}
pprime or (pprime, time_idxs)
- pprime (Array or scalar float) - The pressure gradient. 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 pprime) - The indices (in self. getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single pprime value for volnorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

pprime_val = Eq_instance.volnorm2pprime(0.7, 0.26)

```

Find pprime values at volnorm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

pprime_arr = Eq_instance.volnorm2pprime([0.5, 0.7], 0.26)

```

Find pprime values at volnorm \(=0.5\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

pprime_arr = Eq_instance.volnorm2pprime(0.5, [0.2, 0.3])

```

Find pprime values at (volnorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

pprime_arr = Eq_instance.volnorm2pprime([0.6, 0.5], [0.2, 0.3], each_t=False)

```
\(\mathbf{r z 2 v}\left(R, Z, t,{ }^{* *}\right.\) kwargs)
Calculates the flux surface volume at the given ( \(\mathrm{R}, \mathrm{Z}, \mathrm{t}\) ).
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R (Array-like or scalar float) - Values of the radial coordinate to map to v . 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 \(v\). 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 \(\_\)keyword is False, \(t\) must have the same shape as \(R\) and \(Z\) (or their meshgrid if make_grid is True).

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of v. 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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 ( \(v\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(v\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(v\) ).

\section*{Returns}
\(v\) or ( v , time_idxs)
- \(\mathbf{v}\) (Array or scalar float) - The flux surface 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 \(v\) has this shape as well, unless the make_grid keyword was True, in which case \(v\) has shape \((\operatorname{len}(Z), \operatorname{len}(R))\).
- time_idxs (Array with same shape as \(v\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single v value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

v_val = Eq_instance.rz2v(0.6, 0, 0.26)

```

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

v_arr = Eq_instance.rz2v([0.6, 0.8], [0, 0], 0.26)

```

Find v values at \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

v_arr = Eq_instance.rz2v(0.6, 0, [0.2, 0.3])

```

Find \(v\) values at \((R, Z, t)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

v_arr = Eq_instance.rz2v([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find v values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

v_mat = Eq_instance.rz2v(R, Z, 0.2, make_grid=True)

```
```

rmid2v( R_mid, t, **kwargs)

```

Calculates the flux surface volume corresponding to the passed R_mid (mapped outboard midplane major radius) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- R_mid(Array-like or scalar float)-Values of the outboard midplane major radius to map to \(v\).
- 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 \(R \_m i d\).

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of v. 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 \(R \_m i d\) 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 \(R \_m i d\) or be a scalar. Default is True (evaluate ALL \(R \_m i d\) at EACH element in \(t\) ).
- length_unit (String or 1) - Length unit that \(R \_m i d\) is given in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 (use meters).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(p\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(p\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(p\) ).

\section*{Returns}
\(v\) or ( \(v\), time_idxs)
- \(\mathbf{v}\) (Array or scalar float) - The flux surface 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 \(v\) ) - The indices (in self.get TimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single \(v\) value for \(\mathrm{Rmid}=0.7 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

v_val = Eq_instance.rmid2v(0.7, 0.26)

```

Find \(v\) values at \(R \_m i d\) values of 0.5 m and 0.7 m at the single time \(\mathrm{t}=0.26 \mathrm{~s}\) :
```

v_arr = Eq_instance.rmid2v([0.5, 0.7], 0.26)

```

Find \(v\) values at \(R \_m i d=0.5 \mathrm{~m}\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

v_arr = Eq_instance.rmid2v(0.5, [0.2, 0.3])

```

Find \(v\) values at ( \(\left.\mathrm{R} \_m i d, t\right)\) points \((0.6 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

v_arr = Eq_instance.rmid2v([0.6, 0.5], [0.2, 0.3], each_t=False)

```
roa2v (roa, \(t\), **kwargs)
Convert the passed (r/a, t) coordinates into flux surface volume.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- roa (Array-like or scalar float) - Values of the normalized minor radius to map to v .
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of v. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(v\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(v\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(v\) ).

\section*{Returns}
\(v\) or ( \(v\), time_idxs)
- \(\mathbf{v}\) (Array or scalar float) - The flux surface 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 \(v\) ) - The indices (in self.get TimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single v value at \(\mathrm{r} / \mathrm{a}=0.6, \mathrm{t}=0.26 \mathrm{~s}\) :
```

v_val = Eq_instance.roa2v(0.6, 0.26)

```

Find \(v\) values at \(r / a\) points 0.6 and 0.8 at the single time \(t=0.26 s\) s:
```

v_arr = Eq_instance.roa2v([0.6, 0.8], 0.26)

```

Find \(v\) values at \(\mathrm{r} / \mathrm{a}\) of 0.6 at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

v_arr = Eq_instance.roa2v(0.6, [0.2, 0.3])

```

Find v values at (roa, t\()\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

v_arr = Eq_instance.roa2v([0.6, 0.5], [0.2, 0.3], each_t=False)

```
psinorm2v (psinorm, \(t\), **kwargs)
Calculates the flux surface volume corresponding to the passed psi_norm (normalized poloidal flux) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- psi_norm(Array-like or scalar float)-Values of the normalized poloidal flux to map to \(v\).
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of v. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(v\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(v\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(v\) ).

\section*{Returns}
\(v\) or ( \(v\), time_idxs)
- \(\mathbf{v}\) (Array or scalar float) - The pressure. 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 \(v\) ) - The indices (in self.getTimeBase () ) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single \(v\) value for \(p\) sinorm \(=0.7, t=0.26 \mathrm{~s}\) :
```

v_val = Eq_instance.psinorm2v(0.7, 0.26)

```

Find \(v\) values at psi_norm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

v_arr = Eq_instance.psinorm2v([0.5, 0.7], 0.26)

```

Find \(v\) values at \(p s i \_\)norm \(=0.5\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

v_arr = Eq_instance.psinorm2v(0.5, [0.2, 0.3])

```

Find \(v\) values at (psinorm, t\()\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

v_arr = Eq_instance.psinorm2v([0.6, 0.5], [0.2, 0.3], each_t=False)

```
phinorm2v (phinorm, \(t\), **kwargs)
Calculates the flux surface volume corresponding to the passed phinorm (normalized toroidal flux) values.
By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- phinorm (Array-like or scalar float) - Values of the normalized toroidal flux to map to \(v\).
- 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 \(\_\)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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of v. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(v\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(v\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(v\) ).

\section*{Returns}
\(v\) or ( \(v\), time_idxs)
- \(\mathbf{v}\) (Array or scalar float) - The flux surface 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 \(v\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single \(v\) value for phinorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

v_val = Eq_instance.phinorm2v(0.7, 0.26)

```

Find \(v\) values at phinorm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

v_arr = Eq_instance.phinorm2v([0.5, 0.7], 0.26)

```

Find \(v\) values at phinorm \(=0.5\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

v_arr = Eq_instance.phinorm2v(0.5, [0.2, 0.3])

```

Find v values at (phinorm, t\()\) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

v_arr = Eq_instance.phinorm2v([0.6, 0.5], [0.2, 0.3], each_t=False)

```
volnorm 2 v (volnorm, \(t\), **kwargs)

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

By default, EFIT only computes this inside the LCFS.

\section*{Parameters}
- volnorm (Array-like or scalar float) - Values of the normalized flux surface volume to map to \(v\).
- 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.

\section*{Keyword Arguments}
- sqrt (Boolean) - Set to True to return the square root of v. 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 \(t\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- return_t (Boolean) - Set to True to return a tuple of ( \(v\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(v\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(v\) ).

\section*{Returns}
\(v\) or ( \(v\), time_idxs)
- \(\mathbf{v}\) (Array or scalar float) - The flux surface 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 \(v\) ) - The indices (in self.get TimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single v value for volnorm \(=0.7, \mathrm{t}=0.26 \mathrm{~s}\) :
```

v_val = Eq_instance.volnorm2p(0.7, 0.26)

```

Find \(v\) values at volnorm values of 0.5 and 0.7 at the single time \(t=0.26 \mathrm{~s}\) :
```

v_arr = Eq_instance.volnorm2v([0.5, 0.7], 0.26)

```

Find \(v\) values at volnorm \(=0.5\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

v_arr = Eq_instance.volnorm2v(0.5, [0.2, 0.3])

```

Find \(v\) values at (volnorm, t) points \((0.6,0.2 \mathrm{~s})\) and \((0.5,0.3 \mathrm{~s})\) :
```

v_arr = Eq_instance.volnorm2v([0.6, 0.5], [0.2, 0.3], each_t=False)

```
\(\operatorname{rz2BR}(R, Z, t\), return_t=False, make_grid=False, each_t=True, length_unit=1)
Calculates the major radial component of the magnetic field at the given \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) coordinates.
Uses
\[
B_{R}=-\frac{1}{R} \frac{\partial \psi}{\partial Z}
\]

\section*{Parameters}
- \(\mathbf{R}\) (Array-like or scalar float) - Values of the radial coordinate to map to radial field. 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 radial field. 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (BR, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(B R\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(B R\) ).

\section*{Returns}

BR or (BR, time_idxs)
- BR (Array or scalar float) - The major radial component of the magnetic field. 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 \(B R\) has this shape as well, unless the make_grid keyword was True, in which case \(B R\) has shape (len \((Z)\), len \((R)\) ).
- time_idxs (Array with same shape as \(B R\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single \(B R\) value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

BR_val = Eq_instance.rz2BR(0.6, 0, 0.26)

```

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

BR_arr = Eq_instance.rz2BR([0.6, 0.8], [0, 0], 0.26)

```

Find \(B R\) values at \((R, Z)\) points \((0.6 m, 0 m)\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

BR_arr = Eq_instance.rz2BR(0.6, 0, [0.2, 0.3])

```

Find BR values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

BR_arr = Eq_instance.rz2BR([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find BR values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

BR_mat = Eq_instance.rz2BR(R, Z, 0.2, make_grid=True)

```
rz2BZ \((R, Z, t\), return_t=False, make_grid=False, each_t=True, length_unit=1)
Calculates the vertical component of the magnetic field at the given \((R, Z, t)\) coordinates.
Uses
\[
B_{Z}=\frac{1}{R} \frac{\partial \psi}{\partial R}
\]

\section*{Parameters}
- \(\mathbf{R}\) (Array-like or scalar float) - Values of the radial coordinate to map to vertical field. 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 vertical field. 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (BZ, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(B Z\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(B Z\) ).

\section*{Returns}
\(B Z\) or (BZ, time_idxs)
- BZ (Array or scalar float) - The vertical component of the magnetic field. 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 \(B Z\) has this shape as well, unless the make_grid keyword was True, in which case \(B Z\) has shape (len \((Z)\), len \((R)\) ).
- time_idxs (Array with same shape as BZ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single \(B Z\) value at \(R=0.6 m, Z=0.0 m, t=0.26 s\) :
```

BZ_val = Eq_instance.rz2BZ(0.6, 0, 0.26)

```

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

BZ_arr = Eq_instance.rz2BZ([0.6, 0.8], [0, 0], 0.26)

```

Find BZ values at \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

BZ_arr = Eq_instance.rz2BZ(0.6, 0, [0.2, 0.3])

```

Find \(B Z\) values at \((R, Z, t)\) points \((0.6 m, 0 m, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

BZ_arr = Eq_instance.rz2BZ([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find BZ values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

BZ_mat = Eq_instance.rz2BZ(R, Z, 0.2, make_grid=True)

```
rz2BT ( \(R, Z, t\), **kwargs)
Calculates the toroidal component of the magnetic field at the given \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\).
Uses \(B_{\phi}=F / R\).
By default, EFIT only computes this inside the LCFS. To approximate the field outside of the LCFS, \(B_{\phi} \approx B_{t, v a c} R_{0} / R\) is used, where \(B_{t, v a c}\) is obtained with getBtVac () and \(R_{0}\) is the major radius of the magnetic axis obtained from getMagR ().

The coordinate system used is right-handed, such that "forward" field on Alcator C-Mod (clockwise when seen from above) has negative BT.

\section*{Parameters}
- \(\mathbf{R}\) (Array-like or scalar float) - Values of the radial coordinate to map to BT. 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 BT. 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (BT, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(B T\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(B T\) ).

\section*{Returns}
\(B T\) or (BT, time_idxs)
- BT (Array or scalar float) - The toroidal magnetic field. 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 \(B T\) has this shape as well, unless the make_grid keyword was True, in which case \(B T\) has shape \((\operatorname{len}(Z)\), len \((R)\) ).
- time_idxs (Array with same shape as \(B T\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single BT value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

BT_val = Eq_instance.rz2BT(0.6, 0, 0.26)

```

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

BT_arr = Eq_instance.rz2BT([0.6, 0.8], [0, 0], 0.26)

```

Find \(B T\) values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

BT_arr = Eq_instance.rz2BT(0.6, 0, [0.2, 0.3])

```

Find BT values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

BT_arr = Eq_instance.rz2BT([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find BT values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

BT_mat = Eq_instance.rz2BT(R, Z, 0.2, make_grid=True)

```
rz2B \((R, Z, t, * *\) wargs \()\)
Calculates the magnitude of the magnetic field at the given ( \(\mathrm{R}, \mathrm{Z}, \mathrm{t}\) ).

\section*{Parameters}
- R (Array-like or scalar float) - Values of the radial coordinate to map to B. 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 \(\quad\) 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 B. 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (B, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(B\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(B\) ).

\section*{Returns}
\(B\) or ( \(B\), time_idxs)
- B (Array or scalar float) - The magnitude of the magnetic field. 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 \(B\) has this shape as well, unless the make_grid keyword was True, in which case \(B\) has shape \((\operatorname{len}(Z)\), len \((R))\).
- time_idxs (Array with same shape as \(B\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single \(B\) value at \(R=0.6 m, Z=0.0 m, t=0.26 \mathrm{~s}\) :
```

B_val = Eq_instance.rz2B(0.6, 0, 0.26)

```

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

B_arr = Eq_instance.rz2B([0.6, 0.8], [0, 0], 0.26)

```

Find \(B\) values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

B_arr = Eq_instance.rz2B(0.6, 0, [0.2, 0.3])

```

Find B values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

B_arr = Eq_instance.rz2B([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find B values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

B_mat = Eq_instance.rz2B(R, Z, 0.2, make_grid=True)

```
rz2jR ( \(R, Z, t\), **kwargs)
Calculates the major radial component of the current density at the given \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) coordinates.
\[
j_{R}=-\frac{1}{\mu_{0} R} F^{\prime} \frac{\partial \psi}{\partial Z}=\frac{F^{\prime} B_{R}}{\mu_{0}}
\]

\section*{Parameters}
- \(\mathbf{R}\) (Array-like or scalar float) - Values of the radial coordinate to map to radial current density. 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 radial current density. 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 ( \(j R\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(j R\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(j R\) ).

\section*{Returns}
\(j R\) or ( \(j R\), time_idxs)
- \(\mathbf{j R}\) (Array or scalar float) - The major radial component of the current density. 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 \(j R\) has this shape as well, unless the make \(\_\)grid keyword was True, in which case \(j R\) has shape (len \((Z)\), len \((R)\) ).
- time_idxs (Array with same shape as \(j R\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single \(j R\) value at \(R=0.6 m, Z=0.0 m, t=0.26 s\) :
```

jR_val = Eq_instance.rz2jR(0.6, 0, 0.26)

```

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

jR_arr = Eq_instance.rz2jR([0.6, 0.8], [0, 0], 0.26)

```

Find \(j R\) values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

jR_arr = Eq_instance.rz2jR(0.6, 0, [0.2, 0.3])

```

Find jR values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

jR_arr = Eq_instance.rz2jR([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find jR values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

jR_mat = Eq_instance.rz2jR(R, Z, 0.2, make_grid=True)

```
rz2jZ ( \(R, Z, t\), **kwargs)
Calculates the vertical component of the current density at the given \((R, Z, t)\) coordinates.
Uses
\[
j_{Z}=\frac{1}{\mu_{0} R} F^{\prime} \frac{\partial \psi}{\partial R}=\frac{F^{\prime} B_{Z}}{\mu_{0}}
\]

Note that this function includes a factor of -1 to correct the FF' from Alcator C-Mod's EFIT implementation. You should check the sign of your data.

\section*{Parameters}
- \(\mathbf{R}\) (Array-like or scalar float) - Values of the radial coordinate to map to vertical current density. 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 vertical current density. 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (jZ, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(j Z\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(j Z\) ).

\section*{Returns}
\(j Z\) or (jZ, time_idxs)
- \(\mathbf{j Z}\) (Array or scalar float) - The vertical component of the current density. 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 \(j Z\) has this shape as well, unless the make \(\_\)grid keyword was True, in which case \(j Z\) has shape (len \((Z)\), len \((R)\) ).
- time_idxs (Array with same shape as \(j Z\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single jZ value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

jZ_val = Eq_instance.rz2jZ(0.6, 0, 0.26)

```

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

jZ_arr = Eq_instance.rz2jZ([0.6, 0.8], [0, 0], 0.26)

```

Find jZ values at \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

jZ_arr = Eq_instance.rz2jZ(0.6, 0, [0.2, 0.3])

```

Find jZ values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

jZ_arr = Eq_instance.rz2jZ([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find jZ values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

jZ_mat = Eq_instance.rz2jZ(R, Z, 0.2, make_grid=True)

```
rz2jT ( \(R, Z, t\), **kwargs)
Calculates the toroidal component of the current density at the given \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) coordinates.
Uses
\[
j_{\phi}=R p^{\prime}+\frac{F F^{\prime}}{\mu_{0} R}
\]

The coordinate system used is right-handed, such that "forward" field on Alcator C-Mod (clockwise when seen from above) has negative j T.

\section*{Parameters}
- \(\mathbf{R}\) (Array-like or scalar float) - Values of the radial coordinate to map to toroidal current density. 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 toroidal current density. 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 ( \(j T\), time_idxs), where time_idxs is the array of time indices actually used in evaluating \(j T\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(j T\) ).

\section*{Returns}
\(j T\) or ( \(j T\), time_idxs)
- jT (Array or scalar float) - The major radial component of the current density. 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 \(j T\) has this shape as well, unless the make_grid keyword was True, in which case \(j T\) has shape \((\operatorname{len}(Z)\), len \((R)\) ).
- time_idxs (Array with same shape as \(j T\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single jT value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

jT_val = Eq_instance.rz2jT(0.6, 0, 0.26)

```

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

jT_arr = Eq_instance.rz2jT([0.6, 0.8], [0, 0], 0.26)

```

Find jT values at \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) at times \(\mathrm{t}=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

jT_arr = Eq_instance.rz2jT(0.6, 0, [0.2, 0.3])

```

Find jT values at (R, Z, t) points ( \(0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s}\) ) and ( \(0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s}\) ):
```

jT_arr = Eq_instance.rz2jT([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find jT values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

jT_mat = Eq_instance.rz2jT(R, Z, 0.2, make_grid=True)

```
rz2j( \(R, Z, t,{ }^{* *}\) kwargs)
Calculates the magnitude of the current density at the given ( \(\mathrm{R}, \mathrm{Z}, \mathrm{t}\) ) coordinates.

\section*{Parameters}
- \(\mathbf{R}\) (Array-like or scalar float) - Values of the radial coordinate to map to current density magnitude. 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 current density magnitude. 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).

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 (j, time_idxs), where time_idxs is the array of time indices actually used in evaluating \(j\) with nearest-neighbor interpolation. (This is mostly present as an internal helper.) Default is False (only return \(j\) ).

\section*{Returns}
\(j\) or (j, time_idxs)
- \(\mathbf{j}\) (Array or scalar float) - The magnitude of the current density. 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 \(j\) has this shape as well, unless the make_grid keyword was True, in which case \(j\) has shape \((\operatorname{len}(Z)\), len \((R))\).
- time_idxs (Array with same shape as \(j\) ) - The indices (in self.getTimeBase ()) that were used for nearest-neighbor interpolation. Only returned if return_t is True.

\section*{Examples}

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

Find single j value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}, \mathrm{t}=0.26 \mathrm{~s}\) :
```

j_val = Eq_instance.rz2j(0.6, 0, 0.26)

```

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

j_arr = Eq_instance.rz2j([0.6, 0.8], [0, 0], 0.26)

```

Find \(j\) values at \((R, Z)\) points \((0.6 m, 0 m)\) at times \(t=[0.2 \mathrm{~s}, 0.3 \mathrm{~s}]\) :
```

j_arr = Eq_instance.rz2j(0.6, 0, [0.2, 0.3])

```

Find j values at \((\mathrm{R}, \mathrm{Z}, \mathrm{t})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m}, 0.2 \mathrm{~s})\) and \((0.5 \mathrm{~m}, 0.2 \mathrm{~m}, 0.3 \mathrm{~s})\) :
```

j_arr = Eq_instance.rz2j([0.6, 0.5], [0, 0.2], [0.2, 0.3], each_t=False)

```

Find j values on grid defined by 1D vector of radial positions \(R\) and 1D vector of vertical positions \(Z\) at time \(\mathrm{t}=0.2 \mathrm{~s}\) :
```

j_mat = Eq_instance.rz2j(R, Z, 0.2, make_grid=True)

```
rz2FieldLineTrace (R0, Z0, \(t\), phi0=0.0, field='B', num_rev=1.0, rev_method='toroidal', dphi=0.06283185307179587, integrator='dopri5')
Trace a field line starting from a given ( \(\mathrm{R}, \mathrm{phi}, \mathrm{Z}\) ) point.

\section*{Parameters}
- R0 (float) - Major radial coordinate of starting point.
- Z0 (float) - Vertical coordinate of starting point.
- \(t\) (float) - Time to trace field line at.

\section*{Keyword Arguments}
- phi0 (float) - Toroidal angle of starting point in radians. Default is 0.0.
- field (\{'B', ' \(\left.j^{\prime}\right\}\) ) - The field to use. Can be magnetic field ('B') or current density (' j '). Default is ' B ' (magnetic field).
- num_rev (float) - The number of revolutions to trace the field line through. Whether this refers to toroidal or poloidal revolutions is determined by the rev_method keyword. Default is 1.0 .
- rev_method ('toroidal', 'poloidal') - Whether num_rev refers to the number of toroidal or poloidal revolutions the field line should make. Note that 'poloidal' only makes sense for close field lines. Default is 'toroidal'.
- dphi (float) - Toroidal step size, in radians. Default is \(0.02 *\) pi. The number of steps taken is then \(2 *\) pi times the number of toroidal rotations divided by dphi. This can be negative to trace a field line clockwise instead of counterclockwise.
- integrator (str) - The integrator to use with scipy.integrate.ode. Default is 'dopri5' (explicit Dormand-Prince of order (4)5). Can also be an instance of scipy. integrate. ode for which the integrator and its options has been set.
Returns Containing the ( \(\mathrm{R}, \mathrm{Z}\), phi) coordinates.
Return type array, (nsteps \(+1,3\) )
rho2FieldLineTrace (rho, \(t\), origin='psinorm', **kwargs)
Trace a field line starting from a given normalized coordinate point.
The field line is started at the outboard midplane.

\section*{Parameters}
- rho (float) - Flux surface label of starting point.
- t (float) - Time to trace field line at.

\section*{Keyword Arguments}
- origin (\{'psinorm', 'phinorm', 'volnorm', 'r/a', 'Rmid', 'Fnorm'\}) - The flux surface coordinates which rhovals is given in. Default is 'psinorm'.
- phi0 (float) - Toroidal angle of starting point in radians. Default is 0.0.
- field ( \(\left\{\right.\) ' \(\left.B^{\prime}, \quad ' j^{\prime}\right\}\) ) - The field to use. Can be magnetic field ('B’) or current density (' j '). Default is ' B ' (magnetic field).
- num_rev (float) - The number of revolutions to trace the field line through. Whether this refers to toroidal or poloidal revolutions is determined by the rev_method keyword. Default is 1.0 .
- rev_method ('toroidal', 'poloidal') - Whether num_rev refers to the number of toroidal or poloidal revolutions the field line should make. Note that 'poloidal' only makes sense for close field lines. Default is 'toroidal'.
- dphi (float) - Toroidal step size, in radians. Default is \(0.02 *\) pi. The number of steps taken is then \(2^{*}\) pi times the number of toroidal rotations divided by dphi. This can be negative to trace a field line clockwise instead of counterclockwise.
- integrator (str) - The integrator to use with scipy.integrate.ode. Default is 'dopri5' (explicit Dormand-Prince of order (4)5). Can also be an instance of scipy. integrate. ode for which the integrator and its options has been set.
Returns Containing the ( \(\mathrm{R}, \mathrm{Z}\), phi) coordinates.
Return type array, (nsteps \(+1,3\) )
plotField ( \(t\), rhovals \(=6\), rhomin \(=0.05\), rhomax \(=0.95\), color \(=\) 'b', cmap='plasma', alpha \(=0.5\), arrows \(=\) True, linewidth \(=1.0\), arrowlinewidth \(=3.0, a=\) None, \(* *\) kwargs \()\)
Plot the field lines starting from a number of points.
The field lines are started at the outboard midplane.
If uniformly-spaced psinorm points are used, the spacing of the magnetic field lines will be directly proportional to the field strength, assuming a sufficient number of revolutions is traced.

Parameters t (float) - Time to trace field line at.

\section*{Keyword Arguments}
- rhovals (int or array of int) - The number of uniformly-spaced rho points between rhomin and rhomax to use, or an explicit grid of rho points to use. Default is 6 .
- rhomin (float) - The minimum value of rho to use when using a uniformly-spaced grid. Default is 0.05 .
- rhomax (float) - The maximum value of rho to use when using a uniformly-spaced grid. Default is 0.95 .
- color (str) - The color to plot the field lines in. Default is 'b'. If set to 'sequential', each field line will be a different color, in the sequence matplotlib assigns them. If set to 'magnitude', the coloring will be proportional to the magnitude of the field. Note that this is very time-consuming, as the limitations of matplotlib mean that each line segment must be plotted individually.
- cmap (str) - The colormap to use when color is 'magnitude'. Default is 'plasma', a perceptually uniform sequential colormap.
- alpha (float) - The transparency to plot the field lines with. Default is 0.5.
- arrows (bool) - If True, an arrowhead indicating the field direction will be drawn at the start of each field line. Default is True.
- linewidth (float) - The line width to use when plotting the field lines. Default is 1.0 .
- arrowlinewidth (float) - The line width to use when plotting the arrows. Default is 3.0
- a (matplotlib.axes._subplots.Axes3DSubplot) - The axes to plot the field lines on. Default is to make a new figure. Note that a colorbar will be drawn when color is magnitude, but only if \(a\) is not provided.
- origin (\{'psinorm', 'phinorm', 'volnorm', 'r/a', 'Rmid', 'Fnorm'\}) - The flux surface coordinates which rhovals is given in. Default is 'psinorm'.
- phi0 (float) - Toroidal angle of starting point in radians. Default is 0.0.
- field (\{'B', 'j'\}) - The field to use. Can be magnetic field ('B') or current density (' j '). Default is ' B ' (magnetic field).
- num_rev (float) - The number of revolutions to trace the field line through. Whether this refers to toroidal or poloidal revolutions is determined by the rev_method keyword. Default is 1.0 .
- rev_method ('toroidal', 'poloidal') - Whether num_rev refers to the number of toroidal or poloidal revolutions the field line should make. Note that 'poloidal' only makes sense for close field lines. Default is 'toroidal'.
- dphi (float) - Toroidal step size, in radians. Default is 0.02 *pi. The number of steps taken is then \(2 *\) pi times the number of toroidal rotations divided by dphi. This can be negative to trace a field line clockwise instead of counterclockwise.
- integrator (str) - The integrator to use with scipy.integrate.ode. Default is 'dopri5' (explicit Dormand-Prince of order (4)5). Can also be an instance of scipy. integrate. ode for which the integrator and its options has been set.

Returns The figure and axis which the field lines were plotted in.
Return type (figure, axis)
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.

\section*{Keyword Arguments}
- length_unit (String or 1) - Length unit that \(\mathrm{R} \_\)mag is returned in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 ( \(\mathrm{R} \_\)out returned in meters).
- kind (String or non-negative int) - Specifies the type of interpolation to be performed in getting from \(t\) to \(R \_m a g\). 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) when trispline is True, 'nearest' otherwise.

\section*{Returns}
trispline.UnivariateInterpolator or scipy.interpolate.interpld to convert from t to MagR.
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

\section*{Keyword Arguments}
- length_unit (String or 1) - Length unit that \(\mathrm{R} \_m a g\) is returned in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 ( \(\mathrm{R} \_\)out returned in meters).
- kind (String or non-negative int) - Specifies the type of interpolation to be performed in getting from \(t\) to \(Z \_m a g\). 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) when trispline is True, 'nearest' otherwise.

\section*{Returns}
trispline.UnivariateInterpolator or scipy.interpolate.interpld to convert from t to MagZ.
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

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 ( \(\mathrm{R} \_\)out returned in meters).
- kind (String or non-negative int) - Specifies the type of interpolation to be
 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) when trispline is True, 'nearest' otherwise.

\section*{Returns}
trispline. UnivariateInterpolator or scipy.interpolate.interpld 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.

\section*{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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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) when trispline is True, 'nearest' otherwise.

\section*{Returns}
trispline.UnivariateInterpolator or scipy.interpolate.interp1d to convert from \(t\) to a_out.
getBtVacSpline (kind='nearest')
Gets the univariate spline to interpolate BtVac as a function of time.
Only used if the instance was created with keyword tspline=True.
Keyword Arguments kind (String or non-negative int) - Specifies the type of interpolation to be performed in getting from \(t\) to BtVac. 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) when trispline is True, 'nearest' otherwise.

\section*{Returns}
trispline. UnivariateInterpolator or scipy.interpolate.interpld to convert from t to BtVac .
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 [ $\mathrm{t}, \mathrm{n}$ ]
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 $\mathrm{psi}[\mathrm{psi}, \mathrm{t}]$. 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 \([0,1]\)

\section*{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 \(\mathrm{q}=1\) surface [ t ]
getQ2Surf()
Abstract method. See child classes for implementation.
Returns outboard-midplane minor radius of \(\mathrm{q}=2\) surface [ t ]
getQ3Surf ()
Abstract method. See child classes for implementation.
Returns outboard-midplane minor radius of \(\mathrm{q}=3\) surface [ t ]
getQs()
Abstract method. See child classes for implementation.
Returns specific q-profile values. Namedtuple containing \(\{q 0, q 95\), \(q L C F S\), 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 ]

\section*{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 \}

\section*{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 \(\mathrm{CCW}=+\)
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]
getBCentr()
Abstract method. See child classes for implementation.
Returns Vacuum Toroidal magnetic field at Rcent point [ t ]

\section*{getRCentr()}

Abstract method. See child classes for implementation.
Radial position for Vacuum Toroidal magnetic field calculation
getEnergy ()
Abstract method. See child classes for implementation.
Returns stored-energy parameters. Namedtuple of \{stored energy, confinement time, injected power, \(\mathrm{d} / \mathrm{dt}\) 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 ( \(\mathrm{R}, \mathrm{Z}\) ) coordinates of vacuum wall cross-section for plotting/masking routines.

\section*{getMachineCrossSectionFull()}

Abstract method. See child classes for implementation.
Returns ( \(\mathrm{R}, \mathrm{Z}\) ) 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', \(n b b b s=100\) )
Generates an EFIT gfile with gfile naming convention

\section*{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-timeframe 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 ( \(\mathrm{g}+\) 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.

\section*{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 \(\mathrm{t}=0.26 \mathrm{~s}\), output of g 1001.26 :
\[
\text { Eq_instance.gfile(. } 26 \text { ) }
\]
plotFlux (fill=True, mask=True, \(l w=3.0\), add_title=True)
Plots flux contours directly from psi grid.
Returns the Figure instance created and the time slider widget (in case you need to modify the callback). f.axes contains the contour plot as the first element and the time slice slider as the second element.

\section*{Keyword Arguments}
- fill (Boolean) - Set True to plot filled contours. Set False (default) to plot whitebackground color contours.
- mask (Boolean) - Set True (default) to mask the contours according to the vacuum vessel outline.
- lw (float) - Linewidth when plotting LCFS. Default is 3.0.
- add_title (Boolean) - Set True (default) to add a figure title with the time indicated.

\subsection*{4.1.11 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).

\section*{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.

\section*{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).

\section*{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.

\section*{Examples}

Instantiate EqdskReader for a given shot and time - will search current working directory for files of the form \(\mathrm{g}[\) shot \(]\).[time] and a[shot].[time], suppressing terminal outputs:
```

edr = eqtools.EqdskReader(shot,time,verbose=False)

```
or:
```

edr = eqtools.EqdskReader(shot=shot,time=time,verbose=False)

```

Instantiate EqdskReader with explicit file paths gfile_path and afile_path:
```

edr = eqtools.EqdskReader(gfile=gfile_path,afile=afile_path)

```
```

getInfo()

```
returns namedtuple of equilibrium information

\section*{Returns}
namedtuple containing
\begin{tabular}{|l|l|}
\hline shot & shot index \\
\hline time & time point of g-file \\
\hline nr & size of R-axis of spatial grid \\
\hline nz & size of Z-axis of spatial grid \\
\hline efittype & EFIT calculation type (magnetic, kinetic, MSE) \\
\hline
\end{tabular}
readAFile (afile)
Reads a-file (scalar time-history data) to pull additional equilibrium data not found in g-file, populates remaining data (initialized as None) in object.

Parameters afile (String) - Path to ASCII a-file.
Raises IOError - If afile is not found.
rz2psi ( \(R, Z\), *args, **kwargs)
Calculates the non-normalized poloidal flux at the given ( \(R, Z\) ). Wrapper for Equilibrium.rz2psi masking out timebase dependence.

\section*{Parameters}
- R (Array-like or scalar float) - Values of the radial coordinate to map to poloidal flux. If \(R\) and \(Z\) are both scalar, then a scalar \(p s i\) is returned. \(R\) and \(Z\) must have the same shape unless the make_grid keyword is set. If make_grid is True, \(R\) must have shape (len_R,).
- Z (Array-like or scalar float) - Values of the vertical coordinate to map to poloidal flux. If \(R\) and \(Z\) are both scalar, then a scalar psi is returned. \(R\) and \(Z\) must have the same shape unless the make_grid keyword is set. If make_grid is True, \(Z\) must have shape (len_Z,).

All keyword arguments are passed to the parent Equilibrium.rz2psi. Remaining arguments in *args are ignored.

Returns non-normalized poloidal flux. If all input arguments are scalar, then \(p s i\) is scalar. IF \(R\) and \(Z\) have the same shape, then \(p s i\) has this shape as well. If make \(\_\)grid is True, then \(p s i\) has the shape (len_R, len_Z).

Return type psi (Array-like or scalar float)

\section*{Examples}

All assume that Eq_instance is a valid instance EqdskReader:
Find single psi value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}\) :
```

psi_val = Eq_instance.rz2psi(0.6, 0)

```

Find psi values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\). 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 1 D 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:
\[
\text { psi_norm }=\frac{\psi-\psi(0)}{\psi(a)-\psi(0)}
\]

\section*{Parameters}
- \(\mathbf{R}\) (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,).

All keyword arguments are passed to the parent Equilibrium.rz2psinorm. Remaining arguments in *args are ignored.

Returns non-normalized poloidal flux. If all input arguments are scalar, then psinorm is scalar. IF \(R\) and \(Z\) have the same shape, then psinorm has this shape as well. If make_grid is True, then psinorm has the shape (len_R, len_Z).
Return type psinorm (Array-like or scalar float)

\section*{Examples}

All assume that Eq_instance is a valid instance of EqdskReader:
Find single psinorm value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}\) :
```

psi_val = Eq_instance.rz2psinorm(0.6, 0)

```

Find psinorm values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\). 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
\[
\begin{aligned}
\text { phi } & =\int q(\psi) d \psi \\
\text { phi_norm } & =\frac{\phi}{\phi(a)}
\end{aligned}
\]

Wrapper for Equilibrium.rz2phinorm masking out timebase dependence.

\section*{Parameters}
- R (Array-like or scalar float) - Values of the radial coordinate to map to normalized toroidal flux. Must have the same shape as \(Z\) unless the make_grid keyword is set. If the make_grid keyword is True, R must have shape (len_R,).
- Z (Array-like or scalar float) - Values of the vertical coordinate to map to normalized toroidal flux. Must have the same shape as \(R\) unless the make_grid keyword is set. If the make_grid keyword is True, Z must have shape (len_Z,).
All keyword arguments are passed to the parent Equilibrium.rz2phinorm. Remaining arguments in *args are ignored.

Returns non-normalized poloidal flux. If all input arguments are scalar, then phinorm is scalar. IF \(R\) and \(Z\) have the same shape, then phinorm has this shape as well. If make_grid is True, then phinorm has the shape (len_R, len_Z).

Return type phinorm (Array-like or scalar float)

\section*{Examples}

All assume that Eq_instance is a valid instance of EqdskReader.
Find single phinorm value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}\) :
```

phi_val = Eq_instance.rz2phinorm(0.6, 0)

```

Find phinorm values at \((R, Z)\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\). 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 \(\mathrm{a} / \mathrm{g}\)-files.
Raises Not ImplementedError - in all cases.
rz2rho ( method, \(R, Z, t=\) False, sqrt=False, make_grid=False, \(k=3\), length_unit=1)
Convert the passed ( \(\mathrm{R}, \mathrm{Z}\) ) coordinates into one of several normalized coordinates. Wrapper for Equilibrium.rz2rho masking timebase dependence.

\section*{Parameters}
- method (String) - Indicates which normalized coordinates to use. Valid options are:
\begin{tabular}{|l|l|}
\hline psinorm & Normalized poloidal flux \\
\hline phinorm & Normalized toroidal flux \\
\hline volnorm & Normalized volume \\
\hline
\end{tabular}
- \(\mathbf{R}\) (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,).

\section*{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).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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

Returns 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)).
Return type rho (Array-like or scalar float)
Raises ValueError - If method is not one of the supported values.

\section*{Examples}

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

Find single psinorm value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}\) :
```

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

```

Find psinorm values at \((\mathrm{R}, \mathrm{Z})\) points \((0.6 \mathrm{~m}, 0 \mathrm{~m})\) and \((0.8 \mathrm{~m}, 0 \mathrm{~m})\). 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, \(r\) ho=False, \(k=3\), 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.

\section*{Parameters}
- \(\mathbf{R}\) (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,).

\section*{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 \(R \_m i d\). Default is False (return major radius, R_mid).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- length_unit (String or 1) - Length unit that R and Z are being given in AND that \(R \_m i d\) is returned in. If a string is given, it must be a valid unit specifier:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

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 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 \(R \_m i d\) has this shape as well. If the make_grid keyword was True then \(R \_\)mid has shape \((\operatorname{len}(Z)\), len \((R))\).
Return type R_mid (Array or scalar float)

\section*{Examples}

All assume that Eq_instance is a valid instance of the appropriate extension of the Equilibrium abstract class.
Find single \(\mathrm{R} \_\)mid value at \(\mathrm{R}=0.6 \mathrm{~m}, \mathrm{Z}=0.0 \mathrm{~m}\) :
```

R_mid_val = Eq_instance.rz2rmid(0.6, 0)

```

Find \(R \_m i d\) values at \((R, Z)\) points \((0.6 m, 0 m)\) and \((0.8 m, 0 m)\). 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 ( \(p\) si_norm, \(t=\) False, \(r h o=\) False, \(k=3\), 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.

\section*{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 \(R \_m i d\). Default is False (return major radius, \(R \_m i d\) ).
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.
- 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:
\begin{tabular}{|l|l|}
\hline 'm' & meters \\
\hline 'cm' & centimeters \\
\hline 'mm' & millimeters \\
\hline 'in' & inches \\
\hline 'ft' & feet \\
\hline 'yd' & yards \\
\hline 'smoot' & smoots \\
\hline 'cubit' & cubits \\
\hline 'hand' & hands \\
\hline 'default' & meters \\
\hline
\end{tabular}

If length_unit is 1 or None, meters are assumed. The default value is 1 ( \(R \_m i d\) returned in meters).

Returns If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned.
Return type R_mid (Array-like or scalar float)

\section*{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 \(\mathrm{a} / \mathrm{g}\)-files.
Raises Not ImplementedError - in all cases.
psinorm2phinorm (psi_norm, \(t=\) False, \(k=3\) )
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.

\section*{Keyword Arguments}
- t (indeterminant) - Provides duck typing for inclusion of t values. Passed \(t\) values either as an Arg or Kwarg are neglected.
- \(\mathbf{k}\) (positive int) - The degree of polynomial spline interpolation to use in converting coordinates.

Returns If all of the input arguments are scalar, then a scalar is returned. Otherwise, a scipy Array instance is returned.
Return type phinorm (Array-like or scalar float)

\section*{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.
Returns 1-element, 1D array of time in s. Returns array for consistency with Equilibrium implementations with time variation.

Return type time (Array)
getCurrentSign()
Returns the sign of the current, based on the check in Steve Wolfe's IDL implementation efit_rz2psi.pro.
Returns 1 for positive current, -1 for reversed.
Return type currentSign (Int)
getFluxGrid()
Returns EFIT flux grid.
Returns [1,r,z] Array of flux values. Includes 1-element time axis for consistency with Equilibrium implementations with time variation.
Return type psiRZ (Array)
getRGrid(length_unit=1)
Returns EFIT R-axis.
Returns [r] array of R-axis values for RZ grid.
Return type R (Array)
getZGrid (length_unit=1)
Returns EFIT Z-axis.
Returns [z] array of Z-axis values for RZ grid.
Return type Z (Array)
getFluxAxis()
Returns psi on magnetic axis.
Returns [1] array of psi on magnetic axis. Returns array for consistency with Equilibrium implementations with time variation.
Return type psi0 (Array)
getFluxLCFS ()
Returns psi at separatrix.
Returns [1] array of psi at separatrix. Returns array for consistency with Equilibrium implementations with time variation.

Return type psia (Array)
getRLCFS (length_unit=1)
Returns array of R-values of LCFS.
Returns [1,n] array of R values describing LCFS. Returns array for consistency with Equilibrium implementations with time variation.

\section*{Return type RLCFS (Array)}
getZLCFS (length_unit=1)
Returns array of Z-values of LCFS.
Returns [1,n] array of Z values describing LCFS. Returns array for consistency with Equilibrium implementations with time variation.

Return type ZLCFS (Array)
remapLCFS (mask=False)
Overwrites RLCFS, ZLCFS values pulled from EFIT with explicitly-calculated contour of psinorm=1 surface.

Keyword Arguments mask (Boolean) - Set True to mask LCFS path to limiter outline (using inPolygon). Set False to draw full contour of \(\mathrm{psi}=\) psiLCFS. Defaults to False.
getFluxVol()
Returns volume contained within a flux surface as a function of psi .
Not implemented in EqdskReader, as required data is not stored in g/a-files.
Raises NotImplementedError - in all cases.
getVolLCFS (length_unit=3)
Returns volume with LCFS.
Returns [1] array of plasma volume. Returns array for consistency with Equilibrium implementations with time variation.
Return type Vol (Array)
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 Not ImplementedError - in all cases.
getF ()
returns F=RB_\{Phi\}(Psi), calculated for grad-shafranov solutions [psi,t]
Returns [1,n] array of \(\mathrm{F}(\mathrm{psi})\). Returns array for consistency with Equilibrium implementations with time variation.
Return type F (Array)
getFluxPres()
Returns pressure on flux surface \(\mathrm{p}(\mathrm{psi})\).

Returns [1,n] array of pressure. Returns array for consistency with Equilibrium implementations with time variation.

Return type p (Array)

\section*{getFFPrime()}
returns FF' function used for grad-shafranov solutions.
Returns [1,n] array of \(\mathrm{FF}^{\prime}(\mathrm{psi})\). Returns array for consistency with Equilibrium implementations with time variation.

Return type FF (Array)
getPPrime()
returns plasma pressure gradient as a function of psi.
Returns [1, n] array of pp'(psi). Returns array for consistency with Equilibrium implementations with time variation.

Return type pp (Array)
getElongation()
Returns elongation of LCFS.
Returns [1] array of plasma elongation. Returns array for consistency with Equilibrium implementations with time variation.

Return type kappa (Array)
Raises ValueError - if a-file data is not read.
getUpperTriangularity()
Returns upper triangularity of LCFS.
Returns [1] array of plasma upper triangularity. Returns array for consistency with Equilibrium implementations with time variation.
Return type delta (Array)
Raises ValueError - if a-file data is not read.
getLowerTriangularity()
Returns lower triangularity of LCFS.
Returns [1] array of plasma lower triangularity. Returns array for consistency with Equilibrium implementations with time variation.

Return type delta (Array)
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.
Keyword Arguments length_unit (String or 1) - length unit R is specified in. Defaults to 1 (default unit of rmagx, typically \(m\) ).
Returns [1] array of major radius of magnetic axis. Returns array for consistency with Equilibrium implementations with time variation.

Return type magR (Array)
Raises ValueError - if a-file data is not read.
getMagZ (length_unit=1)
Returns Z of magnetic axis.
Keyword Arguments length_unit (String or 1) - length unit Z is specified in. Defaults to 1 (default unit of zmagx, typically m).

Returns [1] array of Z of magnetic axis. Returns array for consistency with Equilibrium implementations with time variation.

Return type magZ (Array)
Raises ValueError - if a-file data is not read.
getAreaLCFS (length_unit=2)
Returns surface area of LCFS.
Keyword Arguments length_unit (String or 2) - unit area is specified in. Defaults to 2 (default unit, typically \(\mathrm{m}^{\wedge} 2\) ).

Returns [1] array of surface area of LCFS. Returns array for consistency with Equilibrium implementations with time variation.

Return type AreaLCFS (Array)
Raises ValueError - if a-file data is not read.
getAOut (length_unit=1)
Returns outboard-midplane minor radius of LCFS.
Keyword Arguments length_unit (String or 1) - unit radius is specified in. Defaults to 1 (default unit, typically m).
Returns [1] array of outboard-midplane minor radius at LCFS.
Return type AOut (Array)
Raises ValueError - if a-file data is not read.
getRmidOut (length_unit=1)
Returns outboard-midplane major radius of LCFS.
Keyword Arguments length_unit (String or 1) - unit radius is specified in. Defaults to 1 (default unit, typically m).
Returns [1] array of outboard-midplane major radius at LCFS. Returns array for consistency with Equilibrium implementations with time variation.
Return type Rmid (Array)
Raises ValueError - if a-file data is not read.
getGeometry (length_unit=None)
Pulls dimensional geometry parameters.
Keyword Arguments length_unit (String) - length unit parameters are specified in. Defaults to None, using default units for individual getter methods for constituent parameters.
Returns namedtuple containing [Rmag,Zmag,AreaLCFS,aOut,RmidOut]
Raises ValueError - if a-file data is not read.
```

getQProfile()

```

Returns safety factor \(\mathrm{q}(\mathrm{psi})\).

Returns [1,n] array of \(q(p s i)\).
Return type qpsi (Array)
getQ0 ()
Returns safety factor \(q\) on-axis, \(q 0\).
Returns [1] array of \(\mathrm{q}(\mathrm{psi}=0)\). Returns array for consistency with Equilibrium implementations with time variation.

Return type q0 (Array)
Raises ValueError - if a-file data is not read.
getQ95 ()
Returns safety factor q at \(95 \%\) flux surface.
Returns [1] array of \(\mathrm{q}(\mathrm{psi}=0.95)\). Returns array for consistency with Equilibrium implementations with time variation.
Return type q95 (Array)
Raises ValueError - if a-file data is not read.
getQLCFS ()
Returns safety factor \(q\) at LCFS (interpolated).
Returns [1] array of q* (interpolated). Returns array for consistency with Equilibrium implementations with time variation.

\section*{Return type qLCFS (Array)}

Raises ValueError - if a-file data is not loaded.
getQ1Surf (length_unit=1)
Returns outboard-midplane minor radius of \(\mathrm{q}=1\) surface.
Keyword Arguments length_unit (String or 1) - unit of minor radius. Defaults to 1 (default unit, typically m)

Returns [1] array of minor radius of \(q=1\) surface. Returns array for consistency with Equilibrium implementations with time variation.

\section*{Return type qr1 (Array)}

Raises ValueError - if a-file data is not read.
getQ2Surf (length_unit=1)
Returns outboard-midplane minor radius of \(\mathrm{q}=2\) surface.
Keyword Arguments length_unit (String or 1) - unit of minor radius. Defaults to 1 (default unit, typically m)

Returns [1] array of minor radius of \(\mathrm{q}=2\) surface. Returns array for consistency with Equilibrium implementations with time variation.

Return type qr2 (Array)
Raises ValueError - if a-file data is not read.
getQ3Surf (length_unit=1)
Returns outboard-midplane minor radius of \(\mathrm{q}=3\) surface.
Keyword Arguments length_unit (String or 1) - unit of minor radius. Defaults to 1 (default unit, typically m)

Returns [1] array of minor radius of \(q=3\) surface. Returns array for consistency with Equilibrium implementations with time variation.
Return type qr3 (Array)
Raises ValueError - if a-file data is not read.
getQs (length_unit=1)
Pulls q-profile data.
Keyword Arguments length_unit (String or 1) - unit of minor radius. Defaults to 1 (default unit, typically m)
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.
Returns [1] array of vacuum toroidal field. Returns array for consistency with Equilibrium implementations with time variation.

Return type BtVac (Array)
Raises ValueError - if a-file data is not read.
getBtPla()
Returns plasma toroidal field on-axis.
Returns [1] array of toroidal field including plasma effects. Returns array for consistency with Equilibrium implementations with time variation.
Return type BtPla (Array)
Raises ValueError - if a-file data is not read.
getBpAvg ()
Returns average poloidal field.
Returns [1] array of average poloidal field. Returns array for consistency with Equilibrium implementations with time variation.

Return type BpAvg (Array)
Raises ValueError - if a-file data is not read.
getFields()
Pulls vacuum and plasma toroidal field, poloidal field data.
Returns namedtuple containing [ \(\mathrm{BtVac}, \mathrm{BtPla}, \mathrm{BpAvg}\) ]
Raises ValueError - if a-file data is not read.
getIpCalc()
Returns EFIT-calculated plasma current.
Returns [1] array of EFIT-reconstructed plasma current. Returns array for consistency with Equilibrium implementations with time variation.
Return type IpCalc (Array)
getIpMeas()
Returns measured plasma current.
Returns [1] array of measured plasma current. Returns array for consistency with Equilibrium implementations with time variation.

Return type IpMeas (Array)
Raises ValueError - if a-file data is not read.
getJp ()
Returns ( \(\mathrm{r}, \mathrm{z}\) ) grid of toroidal plasma current density.
Data not read from g-file, not implemented for EqdskReader.
Raises Not ImplementedError - In all cases.
getBetaT()
Returns EFIT-calculated toroidal beta.
Returns [1] array of average toroidal beta. Returns array for consistency with Equilibrium implementations with time variation.

\section*{Return type BetaT (Array)}

Raises ValueError - if a-file data is not read.
getBetaP ()
Returns EFIT-calculated poloidal beta.
Returns [1] array of average poloidal beta. Returns array for consistency with Equilibrium implementations with time variation.

Return type BetaP (Array)
Raises ValueError - if a-file data is not read
getLi()
Returns internal inductance of plasma.
Returns [1] array of internal inductance. Returns array for consistency with Equilibrium implementations with time variation.

\section*{Return type Li (Array)}

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.
Returns [1] array of measured diamagnetic flux. Returns array for consistency with Equilibrium implementations with time variation.

\section*{Return type Flux (Array)}

Raises ValueError - if a-file data is not read.
getDiamagBetaT()
Returns diamagnetic-loop measured toroidal beta.
Returns [1] array of measured diamagnetic toroidal beta. Returns array for consistency with Equilibrium implementations with time variation.
Return type BetaT (Array)
Raises ValueError - if a-file data is not read.
getDiamagBetaP()
Returns diamagnetic-loop measured poloidal beta.
Returns [1] array of measured diamagnetic poloidal beta. Returns array for consistency with

\section*{Return type}

\section*{BetaP (Array)}

Equilibrium implementations with time variation.
Raises ValueError - if a-file data is not read.
getDiamagTauE()
Returns diamagnetic-loop energy confinement time.
Returns [1] array of measured energy confinement time. Returns array for consistency with Equilibrium implementations with time variation.
Return type TauE (Array)
Raises ValueError - if a-file data is not read.

\section*{getDiamagWp()}

Returns diamagnetic-loop measured stored energy.
Returns [1] array of diamagnetic stored energy. Returns array for consistency with Equilibrium implementations with time variation.

\section*{Return type Wp (Array)}

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.
Returns [1] array of EFIT-reconstructed stored energy. Returns array for consistency with Equilibrium implementations with time variation.

\section*{Return type WMHD (Array)}

Raises ValueError - if a-file data is not read.
getTauMHD ()
Returns EFIT-calculated energy confinement time.
Returns [1] array of EFIT-reconstructed energy confinement time. Returns array for consistency with Equilibrium implementations with time variation.

\section*{Return type tauMHD (Array)}

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

getPinj()

```

Returns EFIT injected power.
Returns [1] array of EFIT-reconstructed injected power. Returns array for consistency with Equilibrium implementations with time variation.

\section*{Return type Pinj (Array)}

Raises ValueError - if a-file data is not read.
getWbdot()
Returns EFIT d/dt of magnetic stored energy
Returns [1] array of \(\mathrm{d}(\mathrm{Wb}) / \mathrm{dt}\). Returns array for consistency with Equilibrium implementations with time variation.

Return type dWdt (Array)
Raises ValueError - if a-file data is not read.
getWpdot()
Returns EFIT d/dt of plasma stored energy.
Returns [1] array of \(\mathrm{d}(\mathrm{Wp}) / \mathrm{dt}\). Returns array for consistency with Equilibrium implementations with time variation.

Return type dWdt (Array)
Raises ValueError - if a-file data is not read.
getBCentr()
returns Vacuum toroidal magnetic field in Tesla at Rcentr
Returns [nt] array of B_t at center [T]
Return type B_cent (Array)
Raises ValueError - if module cannot retrieve data from MDS tree.
getRCentr (length_unit=1)
returns radius where Bcentr evaluated
Returns Radial position where Bcent calculated [m]

\section*{Return type R}

Raises ValueError - if module cannot retrieve data from MDS tree.
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.
Returns value stored as attribute in EqdskReader.
Return type param (Array-like or scalar float)
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.

\section*{Returns}
( __limiter, Z_limiter )
- R_limiter (Array) - [n] array of x -values for machine cross-section.
- Z_limiter (Array) - [n] array of y-values for machine cross-section.
getMachineCrossSectionFull()
Returns vectorization of machine cross-section.
Absent additional data (not found in eqdsks) simply returns self.getMachineCrossSection().
gfile (time=None, \(n w=\) None, \(n h=\) None, shot=None, name \(=\) None, tunit='ms', title='EQTOOLS', \(n b b b s=100\) )
Generates an EFIT gfile with gfile naming convention

\section*{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-timeframe 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 ( \(\mathrm{g}+\) 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.

\section*{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 \(\mathrm{t}=.26 \mathrm{~s}\) ) output of g 1001.26 :
```

Eq_instance.gfile()

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

\section*{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.

\subsection*{4.1.12 eqtools.filewriter module}
eqtools.filewriter.gfile(obj, tin, \(n w=\) None, \(n h=N o n e\), shot \(=N o n e\), name \(=N o n e\), \(t u n i t=' m s\) ', \(t i-\) \(t l e=\) 'EQTOOLS', \(n b b b s=100\) )
Generates an EFIT gfile with gfile naming convention

\section*{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.

\section*{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.

\section*{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 \(\mathrm{t}=0.26 \mathrm{~s}\), output of g 1001.26 :
```

gfile(Eq_instance,.26)

```

\subsection*{4.1.13 eqtools.pfilereader module}

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

\section*{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: ob ject

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
\begin{tabular}{|l|l|}
\hline 'name' & parameter name \\
\hline 'npts' & array size \\
\hline 'x' & abscissa array \\
\hline 'y' & data array \\
\hline 'dydx' & data gradient \\
\hline 'xunits' & abscissa units \\
\hline 'units' & data units \\
\hline
\end{tabular}
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.

\section*{Examples}

Load p-file data located at file_path, while suppressing terminal output of stored parameters:
```

pfr = eqtools.PFileReader(file_path,verbose=False)

```

Recover electron density data (for example):
```

ne_data = pfr.ne

```

Recover abscissa and electron density data (for example):
```

ne = pfr.ne.y
abscis = pfr.ne.x

```

Available parameters in pfr may be listed via the overridden __str__ command.

\subsection*{4.1.14 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 (x, \(y, \quad z, \quad f, \quad\) boundary='natural', \(\quad d x=0, \quad d y=0, \quad d z=0\), bounds_error=True, fill_value=nan)
Tricubic interpolating spline with forced edge derivative equal zero conditions. It assumes a cartesian grid. The ordering of \(\mathrm{f}[\mathrm{z}, \mathrm{y}, \mathrm{x}]\) is extremely important for the proper evaluation of the spline. It assumes that f is in C order.

Create a new Spline instance.

\section*{Parameters}
- \(\mathbf{x}\) (1-dimensional float array)-Values of the positions of the 1st Dimension of f. Must be monotonic without duplicates.
- \(\mathbf{y}\) (1-dimensional float array)-Values of the positions of the 2 nd dimension of f. Must be monotonic without duplicates.
- \(\mathbf{z}\) (1-dimensional float array) - Values of the positions of the 3rd dimension of f. Must be monotonic without duplicates.
- \(\mathbf{f}\) (3-dimensional float array)-f[x,y,z]. NaN and Inf will hamper performance and affect interpolation in \(4 \times 4 \times 4\) space about its value.

\section*{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 code vs internal python handling.

\section*{Raises}
- ValueError - If any of the dimensions do not match specified f dim
- ValueError - If \(x, y\), or \(z\) are not monotonic

\section*{Examples}

All assume that \(x, y, z\), and \(f\) are valid instances of the appropriate numpy arrays which take independent variables \(\mathrm{x}, \mathrm{y}, \mathrm{z}\) and create numpy array \(\mathrm{f} . x l, y l\), and \(z l\) are numpy arrays which data f is to be interpolated.
Generate a Trispline instance map with data \(\mathrm{x}, \mathrm{y}, \mathrm{z}\) and f :
```

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

```

Evaluate Trispline instance map at \(\mathrm{x} 1, \mathrm{y} 1, \mathrm{z} 1\) :
```

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

```
ev ( \(x i, y i, z i, d x=0, d y=0, d z=0\) )
    evaluates tricubic spline at point (xi,yi,zi) which is \(\mathrm{f}[\mathrm{xi}, \mathrm{yi}, \mathrm{zi}]\).

\section*{Parameters}
- xi (scalar float or 1-dimensional float)-Position in \(x\) dimension. This is the first dimension of 3 d -valued grid.
- yi (scalar float or 1-dimensional float)-Position in y dimension. This is the second dimension of 3 d -valued grid.
- zi (scalar float or 1-dimensional float)-Position in z dimension. This is the third dimension of 3 d -valued grid.

Returns The interpolated value at (xi,yi,zi).
Return type val (array or scalar float)
Raises ValueError - If any of the dimensions exceed the evaluation boundary of the grid
class eqtools.trispline.RectBivariateSpline ( \(x, y, z, b b o x=[N o n e, N o n e, ~ N o n e, ~ N o n e], ~\) \(k x=3, \quad k y=3, \quad s=0\), 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.

\section*{Parameters}
- \(\mathbf{x}\) (1-dimensional float array) - 1-D array of coordinates in monotonically increasing order.
- \(\mathbf{y}\) (1-dimensional float array) - 1-D array of coordinates in monotonically increasing order.
- \(\mathbf{z}\) (2-dimensional float array)-2-D array of data with shape (x.size,y.size).

\section*{Keyword Arguments}
- bbox (1-dimensional float) - Sequence of length 4 specifying the boundary of the rectangular approximation domain. By default, \(b b o x=[\min (x, t x), \max (x, t x)\), \(\min (y, t y), \max (y, t y)]\).
- \(\mathbf{k x}\) (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, \(\operatorname{sum}((w[i] *(z[i]-s(x[i], y[i]))) * * 2\), axis=0) \(<=s\) Default is \(s=0\), which is for interpolation.
\(\mathbf{e v}(x i, y i)\)
Evaluate the rectBiVariateSpline at \((\mathbf{x i}, \mathbf{y i})\). \((\mathbf{x}, \mathbf{y})\) values are checked for being in the bounds of the interpolated data.

\section*{Parameters}
- xi (float array)-input \(x\) dimensional values
- yi (float array) - input x dimensional values

Returns evaluated spline at points ( \(x[i], y[i]\) ), \(i=0, \ldots, \operatorname{len}(x)-1\)
Return type val (float array)
class eqtools.trispline.BivariateInterpolator \((x, y, z)\)
Bases: ob ject
This class provides a wrapper for scipy.interpolate.CloughTocher2DInterpolator.
This is necessary because scipy.interpolate.SmoothBivariateSpline cannot be made to interpolate, and gives inaccurate answers near the boundaries.
\(\mathbf{e v}(x i, y i)\)
class eqtools.trispline.UnivariateInterpolator(*args, **kwargs)
Bases: scipy.interpolate.fitpack2.InterpolatedUnivariateSpline
Interpolated spline class which overcomes the shortcomings of interp1d (inaccurate near edges) and InterpolatedUnivariateSpline (can't set NaN where it extrapolates).

\subsection*{4.1.15 Module contents}

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

\title{
chapter 5
}

Indices and tables
- genindex
- modindex
- search

\section*{Python Module Index}
```

e
eqtools,188
eqtools.afilereader,47
eqtools.AUGData,9
eqtools.CModEFIT, 22
eqtools.core,47
eqtools.D3DEFIT,24
eqtools.EFIT,26
eqtools.eqdskreader,166
eqtools.filewriter,184
eqtools.FromArrays,36
eqtools.NSTXEFIT, 37
eqtools.pfilereader,184
eqtools.TCVLIUQE,40
eqtools.trispline,185

```

\section*{A}

AFileReader (class in eqtools.afilereader), 47
ArrayEquilibrium (class in eqtools.FromArrays), 36
AUGDDData (class in eqtools.AUGData), 9
AUGDDDataProp (class in eqtools.AUGData), 21

\section*{B}

BivariateInterpolator (class in eqtools.trispline), 187

\section*{C}

CModEFITTree (class in eqtools.CModEFIT), 22
CModEFITTreeProp (class in eqtools.CModEFIT), 24

\section*{D}

D3DEFITTree (class in eqtools.D3DEFIT), 24
D3DEFITTreeProp (class in eqtools.D3DEFIT), 25

\section*{E}

EFITTree (class in eqtools.EFIT), 26
EqdskReader (class in eqtools.eqdskreader), 166
eqtools (module), 188
eqtools.afilereader (module), 47
eqtools.AUGData (module), 9
eqtools.CModEFIT (module), 22
eqtools.core (module), 47
eqtools.D3DEFIT (module), 24
eqtools.EFIT (module), 26
eqtools.eqdskreader (module), 166
eqtools.filewriter (module), 184
eqtools.FromArrays (module), 36
eqtools.NSTXEFIT (module), 37
eqtools.pfilereader (module), 184
eqtools.TCVLIUQE (module), 40
eqtools.trispline (module), 185
Equilibrium (class in eqtools.core), 48
ev() (eqtools.trispline.BivariateInterpolator method), 187
ev() (eqtools.trispline.RectBivariateSpline method), 187
ev () (eqtools.trispline.Spline method), 186
F
Fnorm2psinorm() (eqtools.core.Equilibrium
method), 111
G
getAOut () (eqtools.AUGData.AUGDDData method), 14
getAOut () (eqtools.core.Equilibrium method), 161
getAOut () (eqtools.EFIT.EFITTree method), 30
getAOut() (eqtools.eqdskreader.EqdskReader
method), 177
getAOut () (eqtools.TCVLIUQE.TCVLIUQETree method), 44
getAOutSpline() (eqtools.core.Equilibrium
method), 159
getAreaLCFS () (eqtools.AUGData.AUGDDData method), 13
getAreaLCFS() (eqtools.core.Equilibrium method), 161
getAreaLCFS () (eqtools.EFIT.EFITTree method), 30
getAreaLCFS() (eqtools.eqdskreader.EqdskReader method), 177
getAreaLCFS() (eqtools.TCVLIUQE.TCVLIUQETree method), 44
getBCentr() (eqtools.AUGData.AUGDDData method), 17
getBCentr() (eqtools.core.Equilibrium method), 164
getBCentr() (eqtools.EFIT.EFITTree method), 35
getBCentr() (eqtools.eqdskreader.EqdskReader method), 182
getBetaP () (eqtools.AUGData.AUGDDData method), 16
getBetaP () (eqtools.core.Equilibrium method), 163
getBetaP () (eqtools.EFIT.EFITTree method), 33
```

getBetaP() (eqtools.eqdskreader.EqdskReader
method),180
getBetaP() (eqtools.TCVLIUQE.TCVLIUQETree
method),45
getBetas() (eqtools.AUGData.AUGDDData
method),16
getBetas() (eqtools.core.Equilibrium method), 163
getBetas() (eqtools.EFIT.EFITTree method), }3
getBetas() (eqtools.eqdskreader.EqdskReader
method),180
getBetaT() (eqtools.AUGData.AUGDDData
method),16
getBetaT() (eqtools.core.Equilibrium method), 163
getBetaT() (eqtools.EFIT.EFITTree method), }3
getBetaT() (eqtools.eqdskreader.EqdskReader
method),180
getBetaT() (eqtools.TCVLIUQE.TCVLIUQETree
method),45
getBpAvg() (eqtools.AUGData.AUGDDData
method),15
getBpAvg() (eqtools.core.Equilibrium method),162
getBpAvg() (eqtools.EFIT.EFITTree method), 32
getBpAvg() (eqtools.eqdskreader.EqdskReader
method),179
getBtPla() (eqtools.AUGData.AUGDDData
method),15
getBtPla() (eqtools.core.Equilibrium method), 162
getBtPla() (eqtools.EFIT.EFITTree method), 32
getBtPla() (eqtools.eqdskreader.EqdskReader
method),179
getBtPla() (eqtools.TCVLIUQE.TCVLIUQETree
method),45
getBtVac()
(eqtools.AUGData.AUGDDData
method),15
getBtVac() (eqtools.core.Equilibrium method),162
getBtVac() (eqtools.EFIT.EFITTree method), 32
getBtVac() (eqtools.eqdskreader.EqdskReader
method),179
getBtVac() (eqtools.TCVLIUQE.TCVLIUQETree
method),45
getBtVacSpline() (eqtools.core.Equilibrium
method),159
getCurrentSign() (eqtools.AUGData.AUGDDData
method),17
getCurrentSign() (eqtools.core.Equilibrium
method),164
getCurrentSign()
method),35
getCurrentSign() (eq-
tools.eqdskreader.EqdskReader method),
174
getCurrentSign()
tools.FromArrays.ArrayEquilibrium method),
37

```
getDiamag() (eqtools.AUGData.AUGDDData method), 16
getDiamag() (eqtools.core.Equilibrium method), 163
getDiamag () (eqtools.EFIT.EFITTree method), 34
getDiamag() (eqtools.eqdskreader.EqdskReader method), 181
getDiamagBetaP() (eqtools.AUGData.AUGDDData method), 16
getDiamagBetaP() (eqtools.core.Equilibrium method), 163
getDiamagBetaP() (eqtools.EFIT.EFITTree method), 33
getDiamagBetaP() (eqtools.eqdskreader.EqdskReader method), 180
getDiamagBetaT() (eqtools.AUGData.AUGDDData method), 16
getDiamagBetaT() (eqtools.core.Equilibrium method), 163
getDiamagBetaT() (eqtools.EFIT.EFITTree method), 33
getDiamagBetaT() (eqtools.eqdskreader.EqdskReader method), 180
getDiamagFlux() (eqtools.AUGData.AUGDDData method), 16
getDiamagFlux() (eqtools.core.Equilibrium method), 163
getDiamagFlux() (eqtools.EFIT.EFITTree method), 33
getDiamagFlux()
(eqtools.eqdskreader.EqdskReader method), 180
getDiamagTauE() (eqtools.AUGData.AUGDDData method), 16
getDiamagTauE() (eqtools.core.Equilibrium method), 163
getDiamagTauE () (eqtools.EFIT.EFITTree method), 34
getDiamagTauE() (eqtools.eqdskreader.EqdskReader method), 181
getDiamagWp() (eqtools.AUGData.AUGDDData method), 16
getDiamagWp() (eqtools.core.Equilibrium method), 163
getDiamagWp() (eqtools.EFIT.EFITTree method), 34
getDiamagWp() (eqtools.eqdskreader.EqdskReader method), 181
getDiamagWp()
(eqtools.TCVLIUQE.TCVLIUQETree method), 46
getElongation() (eqtools.AUGData.AUGDDData method), 13
```

getElongation() (eqtools.core.Equilibrium
method),161
getElongation() (eqtools.EFIT.EFITTree method),
29
getElongation() (eq-
tools.eqdskreader.EqdskReader method),
176
getElongation()
tools.TCVLIUQETCVLIUQETree
4 3
getEnergy() (eqtools.AUGData.AUGDDData
method),17
getEnergy() (eqtools.core.Equilibrium method), 164
getEnergy() (eqtools.EFIT.EFITTree method), 35
getEnergy() (eqtools.eqdskreader.EqdskReader
method),182
getF () (eqtools.AUGData.AUGDDData method), 12
getF() (eqtools.CModEFIT.CModEFITTree method),
23
getF () (eqtools.core.Equilibrium method), 160
getF() (eqtools.EFIT.EFITTree method), }2
getF() (eqtools.eqdskreader.EqdskReader method),
175
getF() (eqtools.TCVLIUQE.TCVLIUQETree method),
42
getFFPrime() (eqtools.AUGData.AUGDDData
method), }1
getFFPrime() (eqtools.CModEFIT.CModEFITTree
method), }2
getFFPrime() (eqtools.core.Equilibrium method),
1 6 1
getFFPrime()(eqtools.EFIT.EFITTree method), 29
getFFPrime() (eqtools.eqdskreader.EqdskReader
method),176
getFFPrime() (eqtools.TCVLIUQE.TCVLIUQETree
method),43
getFields() (eqtools.AUGData.AUGDDData
method), }1
getFields()(eqtools.core.Equilibrium method),162
getFields() (eqtools.EFIT.EFITTree method), 32
getFields() (eqtools.eqdskreader.EqdskReader
method),179
getFluxAxis() (eqtools.AUGData.AUGDDData
method),11
getFluxAxis() (eqtools.core.Equilibrium method),
160
getFluxAxis()(eqtools.EFIT.EFITTree method), 27
getFluxAxis() (eqtools.eqdskreader.EqdskReader
method), 174
getFluxAxis()
tools.FromArrays.ArrayEquilibrium method),
37
getFluxAxis()
tools.TCVLIUQE.TCVLIUQETree
(eq-
method),

```
getFluxVol() (eqtools.NSTXEFIT.NSTXEFITTree method), 39
getFluxVol() (eqtools.TCVLIUQE.TCVLIUQETree method), 42
getFPrime() method), 12
getGeometry()
(eqtools.AUGData.AUGDDData method), 14
getGeometry() (eqtools.core.Equilibrium method), 161
get Geometry() (eqtools.EFIT.EFITTree method), 31
get Geometry() (eqtools.eqdskreader.EqdskReader method), 177
get Info() (eqtools.AUGData.AUGDDData method), 10
get \(\operatorname{Info()~(eqtools.core.Equilibrium~method),~} 159\)
get \(\operatorname{Info()~(eqtools.EFIT.EFITTree~method),~} 27\)
getInfo() (eqtools.eqdskreader.EqdskReader method), 167
getInfo() (eqtools.TCVLIUQE.TCVLIUQETree method), 41
getIpCalc() (eqtools.AUGData.AUGDDData method), 15
getIpCalc() (eqtools.core.Equilibrium method), 163
getIpCalc() (eqtools.EFIT.EFITTree method), 32
getIpCalc() (eqtools.eqdskreader.EqdskReader method), 179
getIpCalc() (eqtools.NSTXEFIT.NSTXEFITTree method), 39
getIpCalc() (eqtools.TCVLIUQE.TCVLIUQETree method), 45
getIpMeas() method), 15
get IpMeas () (eqtools.core.Equilibrium method), 163
get IpMeas () (eqtools.EFIT.EFITTree method), 32
getIpMeas() (eqtools.eqdskreader.EqdskReader method), 179
get IpMeas() (eqtools.TCVLIUQE.TCVLIUQETree method), 45
getJp () (eqtools.AUGData.AUGDDData method), 15
get Jp () (eqtools.core.Equilibrium method), 163
getJp () (eqtools.EFIT.EFITTree method), 33
getJp() (eqtools.eqdskreader.EqdskReader method), 180
get Jp () (eqtools.NSTXEFIT.NSTXEFITTree method), 39
getLi() (eqtools.AUGData.AUGDDData method), 16
getLi() (eqtools.core.Equilibrium method), 163
getLi () (eqtools.EFIT.EFITTree method), 33
getLi() (eqtools.eqdskreader.EqdskReader method), 180
getLi() (eqtools.TCVLIUQE.TCVLIUQETree method), 45
getLowerTriangularity()
tools.AUGData.AUGDDData method), 13
getLowerTriangularity() (eq
tools.core.Equilibrium method), 161
getLowerTriangularity()
tools.EFIT.EFITTree method), 29
getLowerTriangularity()
tools.eqdskreader.EqdskReader
176
getLowerTriangularity() (eq-
tools.TCVLIUQE.TCVLIUQETree method), 43
getMachineCrossSection() (eq-
tools.AUGData.AUGDDData method), 17
getMachineCrossSection() (eq-
tools.AUGData.YGCAUGInterface method), 21
getMachineCrossSection()
(eq-
tools.core.Equilibrium method), 164
getMachineCrossSection()
tools.EFIT.EFITTree method), 35
getMachineCrossSection() (eq-
tools.eqdskreader.EqdskReader method),
182
getMachineCrossSection() (eq-
tools.NSTXEFIT.NSTXEFITTree method), 39
getMachineCrossSection() (eqtools.TCVLIUQE.TCVLIUQETree method), 46
getMachineCrossSectionFull() (eqtools.AUGData.AUGDDData method), 17
getMachineCrossSectionFull() (eqtools.AUGData.YGCAUGInterface method), 21
getMachineCrossSectionFull() (eq-
tools.CModEFIT.CModEFITTree method), 24
getMachineCrossSectionFull() (eq-
tools.core.Equilibrium method), 164
getMachineCrossSectionFull() tools.EFIT.EFITTree method), 35
getMachineCrossSectionFull()
tools.eqdskreader.EqdskReader 183
getMachineCrossSectionPatch() (eq-
tools.TCVLIUQE.TCVLIUQETree method), 46
getMagR() (eqtools.AUGData.AUGDDData method), 13
getMagR() (eqtools.core.Equilibrium method), 161
getMagR() (eqtools.EFIT.EFITTree method), 30
getMagR() (eqtools.eqdskreader.EqdskReader
method), 176
(eq- getMagR() (eqtools.FromArrays.ArrayEquilibrium
```

    method),37
    getMagR() (eqtools.TCVLIUQE.TCVLIUQETree
method),43
getMagRSpline() (eqtools.core.Equilibrium
method),157
getMagZ() (eqtools.AUGData.AUGDDData method),
13
getMagZ () (eqtools.core.Equilibrium method),161
getMagZ() (eqtools.EFIT.EFITTree method), 30
getMagZ() (eqtools.eqdskreader.EqdskReader
method),177
getMagZ() (eqtools.FromArrays.ArrayEquilibrium
method),37
getMagZ() (eqtools.TCVLIUQE.TCVLIUQETree
method),43
getMagZSpline() (eqtools.core.Equilibrium
method),157
getParam() (eqtools.AUGData.AUGDDData
method),18
getParam() (eqtools.core.Equilibrium method),164
getParam() (eqtools.EFIT.EFITTree method), }3
getParam() (eqtools.eqdskreader.EqdskReader
method), }18
getPinj() (eqtools.AUGData.AUGDDData method),
1 7
getPinj()(eqtools.core.Equilibrium method),164
getPinj() (eqtools.EFIT.EFITTree method), 34
getPinj() (eqtools.eqdskreader.EqdskReader
method),181
getPPrime() (eqtools.AUGData.AUGDDData
method),13
getPPrime() (eqtools.CModEFIT.CModEFITTree
method),23
getPPrime() (eqtools.core.Equilibrium method), 161
getPPrime() (eqtools.EFIT.EFITTree method), 29
getPPrime() (eqtools.eqdskreader.EqdskReader
method),176
getPPrime() (eqtools.TCVLIUQE.TCVLIUQETree
method),43
getQ0()(eqtools.AUGData.AUGDDData method), 14
getQ0 () (eqtools.core.Equilibrium method), 162
getQ0() (eqtools.EFIT.EFITTree method), 31
getQ0() (eqtools.eqdskreader.EqdskReader method),
178
getQ0() (eqtools.TCVLIUQE.TCVLIUQETree
method),44
getQ1Surf() (eqtools.AUGData.AUGDDData
method),15
getQ1Surf()(eqtools.core.Equilibrium method),162
getQ1Surf() (eqtools.EFIT.EFITTree method), 31
getQ1Surf() (eqtools.eqdskreader.EqdskReader
method),178
getQ2Surf() (eqtools.AUGData.AUGDDData
method),15

```
getQ2Surf() (eqtools.core.Equilibrium method), 162
getQ2Surf() (eqtools.EFIT.EFITTree method), 31
getQ2Surf() (eqtools.eqdskreader.EqdskReader method), 178
getQ3Surf() (eqtools.AUGData.AUGDDData method), 15
getQ3Surf() (eqtools.core.Equilibrium method), 162
getQ3Surf() (eqtools.EFIT.EFITTree method), 31
getQ3Surf() (eqtools.eqdskreader.EqdskReader method), 178
getQ95() (eqtools.AUGData.AUGDDData method), 14
getQ95() (eqtools.core.Equilibrium method), 162
getQ95() (eqtools.EFIT.EFITTree method), 31
getQ95() (eqtools.eqdskreader.EqdskReader method), 178
getQ95() (eqtools.TCVLIUQE.TCVLIUQETree method), 44
getQLCFS () (eqtools.AUGData.AUGDDData method), 14
getQLCFS () (eqtools.core.Equilibrium method), 162
getQLCFS () (eqtools.EFIT.EFITTree method), 31
getQLCFS() (eqtools.eqdskreader.EqdskReader method), 178
getQLCFS () (eqtools.TCVLIUQE.TCVLIUQETree method), 45
getQProfile() (eqtools.AUGData.AUGDDData method), 14
getQProfile() (eqtools.CModEFIT.CModEFITTree method), 23
getQProfile() (eqtools.core.Equilibrium method), 162
getQProfile() (eqtools.EFIT.EFITTree method), 31
getQProfile() (eqtools.eqdskreader.EqdskReader method), 177
getQProfile() (eqtools.FromArrays.ArrayEquilibrium method), 37
getQProfile() (eqtools.TCVLIUQE.TCVLIUQETree method), 44
getQs() (eqtools.AUGData.AUGDDData method), 15
getQs() (eqtools.core.Equilibrium method), 162
getQs() (eqtools.EFIT.EFITTree method), 32
getQs() (eqtools.eqdskreader.EqdskReader method), 179
getRCentr() (eqtools.AUGData.AUGDDData method), 17
getRCentr() (eqtools.CModEFIT.CModEFITTree method), 24
getRCentr() (eqtools.core.Equilibrium method), 164 getRCentr() (eqtools.EFIT.EFITTree method), 35
getRCentr() (eqtools.eqdskreader.EqdskReader method), 182
getRGrid() (eqtools.AUGData.AUGDDData method), 11
getRGrid() (eqtools.core.Equilibrium method), 160
getRGrid() (eqtools.EFIT.EFITTree method), 27
getRGrid() (eqtools.eqdskreader.EqdskReader method), 174
getRGrid() (eqtools.FromArrays.ArrayEquilibrium method), 37
getRGrid() (eqtools.TCVLIUQE.TCVLIUQETree method), 41
getRLCFS () (eqtools.AUGData.AUGDDData method), 12
getRLCFS () (eqtools.CModEFIT.CModEFITTree method), 24
getRLCFS () (eqtools.core.Equilibrium method), 160
getRLCFS () (eqtools.EFIT.EFITTree method), 28
getRLCFS() (eqtools.eqdskreader.EqdskReader method), 175
getRLCFS () (eqtools.FromArrays.ArrayEquilibrium method), 37
getRLCFS () (eqtools.TCVLIUQE.TCVLIUQETree method), 42
getRmidOut () (eqtools.AUGData.AUGDDData method), 14
getRmidOut() (eqtools.core.Equilibrium method), 161
getRmidOut () (eqtools.EFIT.EFITTree method), 30
getRmidOut() (eqtools.eqdskreader.EqdskReader method), 177
getRmidOut () (eqtools.FromArrays.ArrayEquilibrium method), 37
getRmidOut () (eqtools.TCVLIUQE.TCVLIUQETree method), 44
getRmidOutSpline() (eqtools.core.Equilibrium method), 158
getRmidPsi() (eqtools.AUGData.AUGDDData method), 11
getRmidPsi() (eqtools.CModEFIT.CModEFITTree method), 23
getRmidPsi() (eqtools.core.Equilibrium method), 160
getRmidPsi() (eqtools.D3DEFIT.D3DEFITTree method), 25
getRmidPsi() (eqtools.EFIT.EFITTree method), 28
getRmidPsi() (eqtools.eqdskreader.EqdskReader method), 175
getRmidPsi() (eqtools.NSTXEFIT.NSTXEFITTree method), 39
getRmidPsi() (eqtools.TCVLIUQE.TCVLIUQETree method), 42
getShaping() (eqtools.AUGData.AUGDDData method), 13
getShaping() (eqtools.core.Equilibrium method), 161
getShaping() (eqtools.EFIT.EFITTree method), 30
getShaping() (eqtools.eqdskreader.EqdskReader method), 176
getSSQ() (eqtools.AUGData.AUGDDData method), 18
get TauMHD () (eqtools.AUGData.AUGDDData method), 17
get TauMHD () (eqtools.core.Equilibrium method), 164 get TauMHD () (eqtools.EFIT.EFITTree method), 34
get TauMHD () (eqtools.eqdskreader.EqdskReader method), 181
get TauMHD () (eqtools.TCVLIUQE.TCVLIUQETree method), 46
getTimeBase() (eqtools.AUGData.AUGDDData method), 10
getTimeBase() (eqtools.core.Equilibrium method), 159
get TimeBase() (eqtools.EFIT.EFITTree method), 27
getTimeBase() (eqtools.eqdskreader.EqdskReader method), 174
getTimeBase() (eqtools.FromArrays.ArrayEquilibrium method), 37
getTimeBase() (eqtools.TCVLIUQE.TCVLIUQETree method), 41
getUpperTriangularity() (eqtools.AUGData.AUGDDData method), 13
getUpperTriangularity() (eqtools.core.Equilibrium method), 161
getUpperTriangularity() (eqtools.EFIT.EFITTree method), 29
getUpperTriangularity() (eqtools.eqdskreader.EqdskReader method), 176
getUpperTriangularity() (eqtools.TCVLIUQE.TCVLIUQETree method), 43
getVolLCFS () (eqtools.AUGData.AUGDDData method), 11
getVolLCFS() (eqtools.core.Equilibrium method), 160
getVolLCFS () (eqtools.EFIT.EFITTree method), 28
getVolLCFS() (eqtools.eqdskreader.EqdskReader method), 175
getVolLCFS () (eqtools.NSTXEFIT.NSTXEFITTree method), 39
getVolLCFS () (eqtools.TCVLIUQE.TCVLIUQETree method), 42
getWbdot () (eqtools.AUGData.AUGDDData method), 17
getWbdot () (eqtools.core.Equilibrium method), 164 getWbdot () (eqtools.EFIT.EFITTTree method), 34
getWbdot() (eqtools.eqdskreader.EqdskReader
```

        method),182
    getWMHD () (eqtools.AUGData.AUGDDData method), 16
getWMHD () (eqtools.core.Equilibrium method), 164
getWMHD () (eqtools.EFIT.EFITTree method), 34
getWMHD() (eqtools.eqdskreader.EqdskReader
method),181
getWpdot() (eqtools.AUGData.AUGDDData
method),17
getWpdot() (eqtools.core.Equilibrium method),164
getWpdot () (eqtools.EFIT.EFITTree method), }3
getWpdot() (eqtools.eqdskreader.EqdskReader
method),182
getZGrid() (eqtools.AUGData.AUGDDData
method), 11
getZGrid() (eqtools.core.Equilibrium method), 160
getZGrid() (eqtools.EFIT.EFITTree method), }2
getZGrid() (eqtools.eqdskreader.EqdskReader
method), 174
getZGrid() (eqtools.FromArrays.ArrayEquilibrium
method), 37
getZGrid() (eqtools.TCVLIUQE.TCVLIUQETree
method),41
getZLCFS() (eqtools.AUGData.AUGDDData
method),12
getZLCFS() (eqtools.CModEFIT.CModEFITTree
method),24
getZLCFS() (eqtools.core.Equilibrium method),160
getZLCFS() (eqtools.EFIT.EFITTree method), }2
getZLCFS() (eqtools.eqdskreader.EqdskReader
method),175
getZLCFS() (eqtools.FromArrays.ArrayEquilibrium
method), 37
getZLCFS() (eqtools.TCVLIUQE.TCVLIUQETree
method),42
gfile()(eqtools.core.Equilibrium method),165
gfile() (eqtools.eqdskreader.EqdskReader method),
183
gfile()(in module eqtools.filewriter), 184
greenArea() (in module eqtools.TCVLIUQE),40

```

\section*{I}
```

inPolygon() (in module eqtools.core), 47

```

\section*{M}
```

ModuleWarning, 47

```

\section*{N}
```

NSTXEFITTree (class in eqtools.NSTXEFIT), 37
NSTXEFITTreeProp (class in eqtools.NSTXEFIT), 39

```

\section*{P}

PFileReader (class in eqtools.pfilereader), 184 phinorm2F() (eqtools.core.Equilibrium method), 109
phinorm2FFPrime() (eqtools.core.Equilibrium method), 117
phinorm2p() (eqtools.core.Equilibrium method), 124
phinorm2pprime() (eqtools.core.Equilibrium method), 132
phinorm2psinorm() (eqtools.core.Equilibrium method), 83
phinorm2q() (eqtools.core.Equilibrium method), 101
phinorm2rho() (eqtools.core.Equilibrium method), 88
phinorm2rmid() (eqtools.core.Equilibrium method), 85
phinorm2roa() (eqtools.core.Equilibrium method), 87
phinorm2v() (eqtools.core.Equilibrium method), 139
phinorm2volnorm() (eqtools.core.Equilibrium method), 84
plotField() (eqtools.core.Equilibrium method), 155
plotFlux() (eqtools.core.Equilibrium method), 165
plotFlux() (eqtools.eqdskreader.EqdskReader method), 183
plotFlux() (eqtools.TCVLIUQE.TCVLIUQETree method), 46
PropertyAccessMixin (class in eqtools.core), 47
psinorm2F() (eqtools.core.Equilibrium method), 108
psinorm2FFPrime() (eqtools.core.Equilibrium method), 116
psinorm2p() (eqtools.core.Equilibrium method), 123
psinorm2phinorm() (eqtools.core.Equilibrium method), 80
psinorm2phinorm() (eqtools.eqdskreader.EqdskReader method), 173
psinorm2pprime() (eqtools.core.Equilibrium method), 131
psinorm2q() (eqtools.core.Equilibrium method), 100
psinorm2rho() (eqtools.core.Equilibrium method), 81
psinorm2rmid() (eqtools.core.Equilibrium method), 77
psinorm2rmid() (eqtools.eqdskreader.EqdskReader method), 172
psinorm2roa() (eqtools.core.Equilibrium method), 78
psinorm2v() (eqtools.core.Equilibrium method), 138
psinorm2volnorm() (eqtools.core.Equilibrium method), 79
psinorm2volnorm() (eqtools.eqdskreader.EqdskReader method), 173
psinorm2volnorm() tools.NSTXEFIT.NSTXEFITTree method), 39

\section*{R}
readAFile() (eqtools.eqdskreader.EqdskReader method), 167
RectBivariateSpline (class in eqtools.trispline), 186
remapLCES () (eqtools.AUGData.AUGDDData method), 12
remapLCFS () (eqtools.core.Equilibrium method), 160
remaplCFS () (eqtools.EFIT.EFITTree method), 28
remapLCFS () (eqtools.eqdskreader.EqdskReader method), 175
rho2FieldLineTrace() (eqtools.core.Equilibrium method), 155
rho2rho() (eqtools.core.Equilibrium method), 49
rmid2F() (eqtools.core.Equilibrium method), 105
rmid2FFPrime() (eqtools.core.Equilibrium method), 114
rmid2p() (eqtools.core.Equilibrium method), 121
rmid2phinorm() (eqtools.core.Equilibrium method), 66
rmid2pprime() (eqtools.core.Equilibrium method), 128
rmid2psinorm() (eqtools.core.Equilibrium method), 65
rmid2q() (eqtools.core.Equilibrium method), 98
rmid2rho() (eqtools.core.Equilibrium method), 69
rmid2roa() (eqtools.core.Equilibrium method), 64
rmid2v() (eqtools.core.Equilibrium method), 136
rmid2volnorm() (eqtools.core.Equilibrium method), 68
roa2F () (eqtools.core.Equilibrium method), 107
roa2FFPrime() (eqtools.core.Equilibrium method), 115
roa2p () (eqtools.core.Equilibrium method), 122
roa2phinorm() (eqtools.core.Equilibrium method), 73
roa2pprime() (eqtools.core.Equilibrium method), 130
roa2psinorm() (eqtools.core.Equilibrium method), 72
roa2q() (eqtools.core.Equilibrium method), 99
roa 2 rho() (eqtools.core.Equilibrium method), 75
roa2rmid() (eqtools.core.Equilibrium method), 71
roa2v() (eqtools.core.Equilibrium method), 137
roa2volnorm() (eqtools.core.Equilibrium method), 74
rz2B() (eqtools.core.Equilibrium method), 146
rz2BR() (eqtools.AUGData.AUGDDData method), 18
rz2BR() (eqtools.core.Equilibrium method), 141
rz2BT() (eqtools.core.Equilibrium method), 145
rz2BZ() (eqtools.AUGData.AUGDDData method), 19
rz2BZ() (eqtools.core.Equilibrium method), 143
rz2F () (eqtools.core.Equilibrium method), 104
rz2FFPrime() (eqtools.core.Equilibrium method), 112
rz2FieldLineTrace() (eqtools.core.Equilibrium method), 154
rz2j() (eqtools.core.Equilibrium method), 153
rz2jR() (eqtools.core.Equilibrium method), 148
rz2jT() (eqtools.core.Equilibrium method), 151
rz2jZ() (eqtools.core.Equilibrium method), 149
rz2p () (eqtools.core.Equilibrium method), 119
rz2phinorm() (eqtools.core.Equilibrium method), 54
rz2phinorm() (eqtools.eqdskreader.EqdskReader method), 168
rz2pprime() (eqtools.core.Equilibrium method), 127
rz2psi() (eqtools.core.Equilibrium method), 51
rz2psi() (eqtools.eqdskreader.EqdskReader method), 167
rz2psinorm() (eqtools.core.Equilibrium method), 53
rz2psinorm() (eqtools.eqdskreader.EqdskReader method), 168
rz2q() (eqtools.core.Equilibrium method), 96
rz2rho() (eqtools.core.Equilibrium method), 61
rz2rho() (eqtools.eqdskreader.EqdskReader method), 169
rz2rmid() (eqtools.core.Equilibrium method), 58
rz2rmid() (eqtools.eqdskreader.EqdskReader method), 171
rz2roa() (eqtools.core.Equilibrium method), 60
\(\mathrm{rz2v}()\) (eqtools.core.Equilibrium method), 134
rz2volnorm() (eqtools.core.Equilibrium method), 56
rz2volnorm() (eqtools.eqdskreader.EqdskReader method), 169
rz2volnorm() (eqtools.NSTXEFIT.NSTXEFITTree method), 39

\section*{S}

Spline (class in eqtools.trispline), 185

\section*{T}

TCVLIUQETree (class in eqtools.TCVLIUQE), 40
TCVLIUQETreeProp (class in eqtools.TCVLIUQE), 46

U
UnivariateInterpolator (class in eqtools.trispline), 187

\section*{V}
volnorm2F () (eqtools.core.Equilibrium method), 110 volnorm2FFPrime() (eqtools.core.Equilibrium method), 118
volnorm2p () (eqtools.core.Equilibrium method), 126 volnorm2phinorm() (eqtools.core.Equilibrium method), 91
```

volnorm2pprime() (eqtools.core.Equilibrium
method),133
volnorm2psinorm() (eqtools.core.Equilibrium
method),}9
volnorm2q() (eqtools.core.Equilibrium method), 103
volnorm2rho() (eqtools.core.Equilibrium method),
94
volnorm2rmid() (eqtools.core.Equilibrium method),
92
volnorm2roa() (eqtools.core.Equilibrium method),
93
volnorm2v() (eqtools.core.Equilibrium method), 140
Y
YGCAUGInterface (class in eqtools.AUGData), 21

```
```

