pfsspy Documentation

pfsspy contributors

Mar 25, 2019

Contents

pfsspy is a python package for carrying out Potential Field Source Surface modelling. For more information on the actually PFSS calculation see this document.

Note: pfsspy is a very new package, so elements of the API are liable to change with the first few releases. If you find any bugs or have any suggestions for improvement, please raise an issue here: https://github.com/dstansby/pfsspy/issues

pfsspy can be installed from PyPi using

```
pip install pfsspy
```

CHAPTER 1

Code reference

For the main user-facing code see

1.1 pfsspy Package

1.1.1 Functions

load_output(file)	Load a saved output file.
pfss(input)	Compute PFSS model.

load_output

pfsspy.**load_output**(*file*)

Load a saved output file.

Loads a file saved using *Output.save()*.

Parameters

file [str, file, Path] File to load.

Returns

:class:'Output'

pfss

pfsspy.**pfss**(*input*)

Compute PFSS model.

Extrapolates a 3D PFSS using an eigenfunction method in r, s, p coordinates, on the dumfric grid (equally spaced in $\rho = \ln(r/r_{sun})$, $s = \cos(\theta)$, and $p = \phi$).

The output should have zero current to machine precision, when computed with the DuMFriC staggered discretization.

Parameters

input [*Input*] Input parameters.

Returns

out [*Output*]

1.1.2 Classes

FieldLine(x, y, z, output)	A single magnetic field line.
Grid(ns, nphi, nr, rss)	Grid on which the solution is calculated.
Input(br, nr, rss)	Input to PFSS modelling.
Output(alr, als, alp, grid)	Output of PFSS modelling.

FieldLine

class `pfsspy.FieldLine`(*x*, *y*, *z*, *output*)

Bases: `astropy.coordinates.sky_coordinate.SkyCoord`

A single magnetic field line.

Parameters

x :

y :

z :

output [*Output*]

Attributes Summary

expansion_factor	Magnetic field expansion factor.
is_open	Returns `True` if one of the field line is connected to the solar surface and one to the outer boundary, `False` otherwise.
polarity	Magnetic field line polarity.

Attributes Documentation

expansion_factor

Magnetic field expansion factor.

The expansion factor is defnied as $(r_{\odot}^2 B_{\odot})/(r_{ss}^2 B_{ss}))$

Returns

exp_fact [float] Field line expansion factor. If field line is closed, returns `None`.

is_open

Returns `True` if one of the field line is connected to the solar surface and one to the outer boundary, `False` otherwise.

polarity

Magnetic field line polarity.

Returns

pol [int] 0 if the field line is closed, otherwise sign(Br) of the magnetic field on the solar surface.

Grid

class pfsspy.**Grid**(*ns*, *nphi*, *nr*, *rss*)

Bases: `object`

Grid on which the solution is calculated.

The grid is evenly spaced in (cos(theta), phi, log(r)).

Attributes Summary

dp	Cell size in phi.
dr	Cell size in log(r).
ds	Cell size in cos(theta).
pc	Location of the centre of cells in phi.
pg	Location of the edges of grid cells in phi.
rc	Location of the centre of cells in log(r).
rg	Location of the edges of grid cells in log(r).
sc	Location of the centre of cells in cos(theta).
sg	Location of the edges of grid cells in cos(theta).

Attributes Documentation

dp

Cell size in phi.

dr

Cell size in log(r).

ds

Cell size in cos(theta).

pc

Location of the centre of cells in phi.

pg

Location of the edges of grid cells in phi.

rc

Location of the centre of cells in log(r).

rg

Location of the edges of grid cells in log(r).

sc

Location of the centre of cells in cos(theta).

sg

Location of the edges of grid cells in cos(theta).

Input

class pfsspy.**Input** (*br*, *nr*, *rss*)

Bases: object

Input to PFSS modelling.

Parameters

br [2D array, sunpy.map.Map] Boundary condition of radial magnetic field at the inner surface. If a SunPy map is automatically extracted as map.data with *no* processing.

nr [int] Number of cells in the radial direction.

rss [float] Radius of the source surface, as a fraction of the solar radius.

Methods Summary

plot_input([ax])	Plot a 2D image of the magnetic field boundary condition.

Methods Documentation

plot_input (*ax=None*)

Plot a 2D image of the magnetic field boundary condition.

Parameters

ax [Axes] Axes to plot to. If None, creates a new figure.

Output

class pfsspy.**Output** (*alr*, *als*, *alp*, *grid*)

Bases: object

Output of PFSS modelling.

Parameters

alr : Vector potential * grid spacing in radial direction.

als : Vector potential * grid spacing in elevation direction.

alp : Vector potential * grid spacing in azimuth direction.

grid [Grid] Grid that the output was caclulated on.

Attributes Summary

al	Vector potential times cell edge lenghts.
bc	B on the centres of the cell faces.
bg	B as a (weighted) averaged on grid points.
source_surface_br	Br on the source surface.

Methods Summary

plot_pil([ax])	Plot the polarity inversion line on the source surface.
plot_source_surface([ax])	Plot a 2D image of the magnetic field at the source surface.
save(file)	Save the output to file.
trace(x0[, atol, rtol])	Traces a field-line from *x0*.

Attributes Documentation

al
Vector potential times cell edge lenghts.

Returns ar*Lr, as*Ls, ap*Lp on cell edges.

bc
B on the centres of the cell faces.

bg
B as a (weighted) averaged on grid points.

source_surface_br
Br on the source surface.

Methods Documentation

plot_pil(*ax=None*)
Plot the polarity inversion line on the source surface.

The PIL is where Br = 0.

Parameters

ax [Axes] Axes to plot to. If `None`, creates a new figure.

plot_source_surface(*ax=None*)
Plot a 2D image of the magnetic field at the source surface.

Parameters

ax [Axes] Axes to plot to. If `None`, creates a new figure.

save(*file*)
Save the output to file.

This saves the required information to reconstruct an Output object in a compressed binary numpy file (see `numpy.savez_compressed()` for more information). The file extension is `.npz`, and is automatically added if not present.

Parameters

file [str, file, `Path`] File to save to. If `.npz` extension isn't present it is added when saving the file.

trace(*x0*, *atol=0.0001*, *rtol=0.0001*)
Traces a field-line from *x0*.

Uses *scipy.integrate.solve_ivp*, with an LSODA method.

Parameters

x0 [array] Starting coordinate, in cartesian coordinates.

dtf [float, optional] Absolute tolerance of the tracing.

rtol [float, optional] Relative tolerance of the tracing.

Returns

fl [*FieldLine*]

for usage examples see

1.2 pfsspy examples

Note: Click *here* to download the full example code

1.2.1 Dipole source solution

A simple example showing how to use pfsspy to compute the solution to a dipole source field.

First, import required modules

```
import astropy.constants as const
import matplotlib.pyplot as plt
import matplotlib.patches as mpatch
import numpy as np
import pfsspy
```

Set up a 1degree by 1degree grid in theta and phi

```
nphi = 360
ntheta = 180

phi = np.linspace(0, 2 * np.pi, nphi)
theta = np.linspace(-np.pi / 2, np.pi / 2, ntheta)
theta, phi = np.meshgrid(theta, phi)
```

Define the number of radial grid points and the source surface radius

```
nr = 50
rss = 2.5
```

Compute radial component ofa dipole field

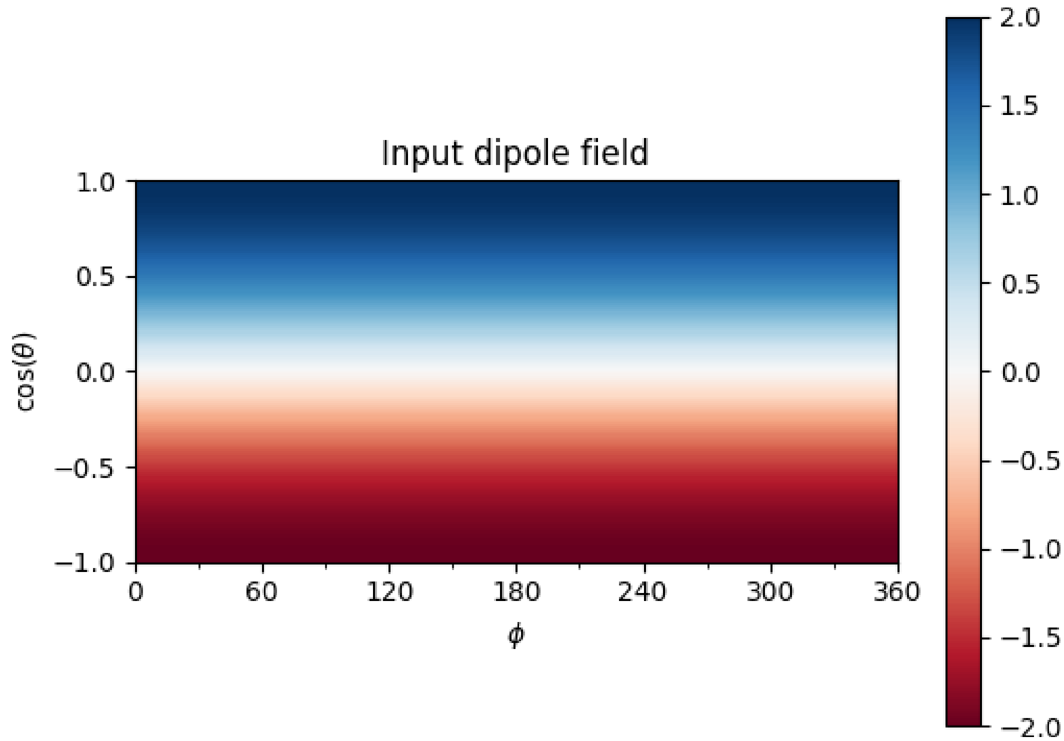
```
def dipole_Br(r, theta):
    return 2 * np.sin(theta) / r**3


br = dipole_Br(1, theta).T
```

Create PFSS input object

```
input = pfsspy.Input(br, nr, rss)
```

Plot input magnetic field

```
fig, ax = plt.subplots()
mesh = input.plot_input(ax)
fig.colorbar(mesh)
ax.set_title('Input dipole field')
```

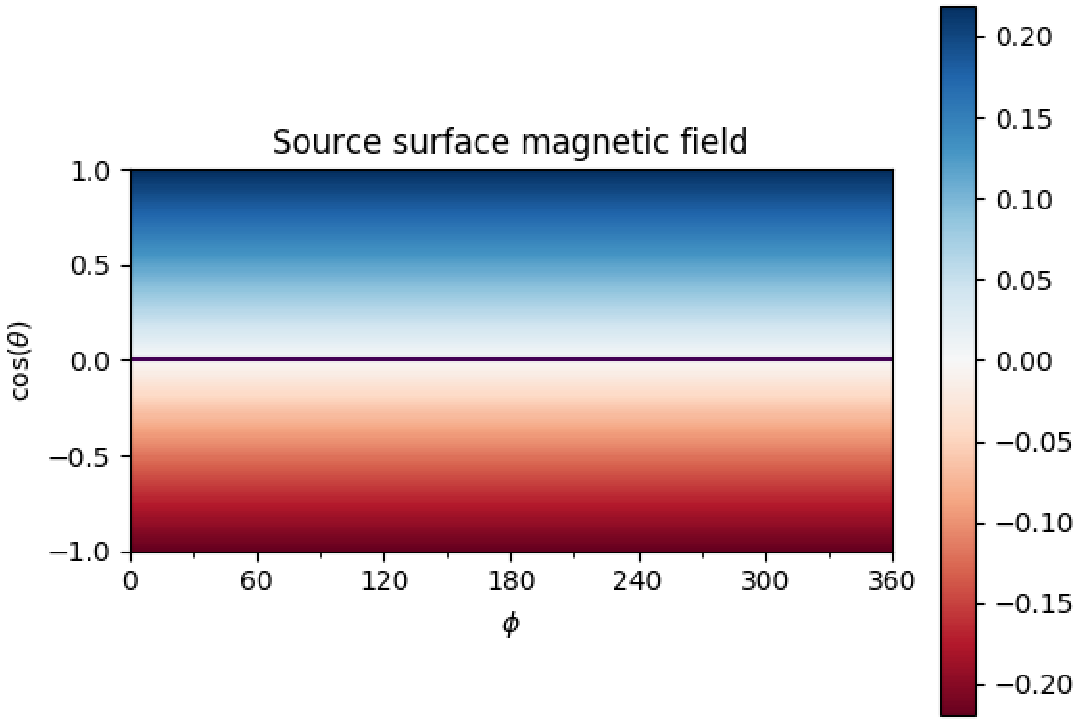


Calculate PFSS solution

```
output = pfsspy.pfss(input)
```

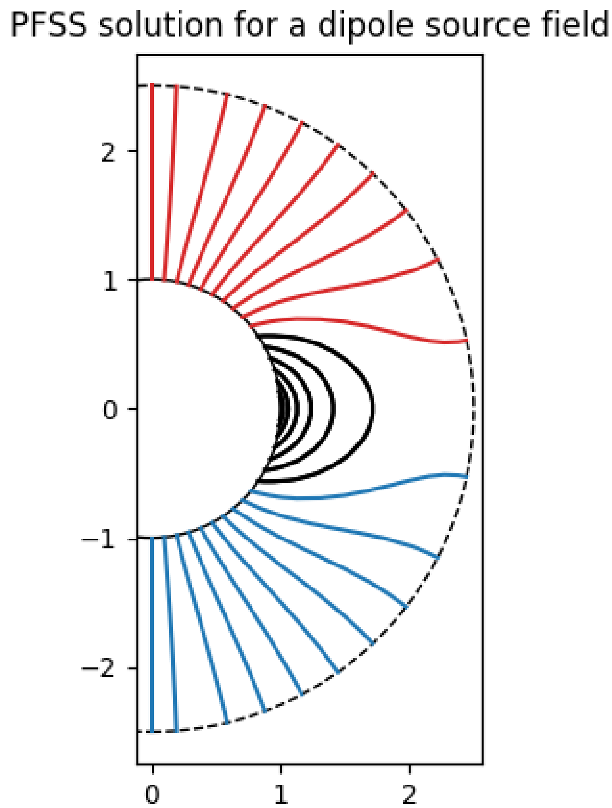
Plot output field

```
fig, ax = plt.subplots()
mesh = output.plot_source_surface(ax)
fig.colorbar(mesh)
output.plot_pil(ax)
ax.set_title('Source surface magnetic field')
```



Trace some field lines

```
br, btheta, bphi = output.bg

fig, ax = plt.subplots()
ax.set_aspect('equal')

# Take 32 start points spaced equally in theta
r = 1.01
for theta in np.linspace(0, np.pi, 33):
    x0 = np.array([0, r * np.sin(theta), r * np.cos(theta)])
    field_line = output.trace(x0)
    color = {0: 'black', -1: 'tab:blue', 1: 'tab:red'}.get(field_line.polarity)
    ax.plot(field_line.y / const.R_sun,
            field_line.z / const.R_sun, color=color)

# Add inner and outer boundary circles
ax.add_patch(mpatch.Circle((0, 0), 1, color='k', fill=False))
ax.add_patch(mpatch.Circle((0, 0), input.grid.rss, color='k', linestyle='--',
                           fill=False))
ax.set_title('PFSS solution for a dipole source field')
plt.show()
```



Total running time of the script: (0 minutes 9.827 seconds)

Note: Click *here* to download the full example code

1.2.2 GONG PFSS extrapolation

Calculating PFSS solution for a GONG synoptic magnetic field map.

First, import required modules

```
import os
import astropy.constants as const
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pfsspy
import sunpy.map
```

If a gong magnetic field map isn't present, download one

```
if not os.path.exists('gong.fits') and not os.path.exists('gong.fits.gz'):
    import urllib.request
    urllib.request.urlretrieve(
```

(continues on next page)

(continued from previous page)

```
        'https://gong2.nso.edu/oQR/zqs/201901/mrzqs190108/mrzqs190108t1114c2212_050.
↪fits.gz',
        'gong.fits.gz')

if not os.path.exists('gong.fits'):
    import gzip
    with gzip.open('gong.fits.gz', 'rb') as f:
        with open('gong.fits', 'wb') as g:
            g.write(f.read())
```

Use SunPy to read the .fits file with the data

```
map = sunpy.map.Map('gong.fits')
nr = 60
rss = 2.5
```

Extract the data, and remove the mean to enforce div(B) = 0 on the solar surface

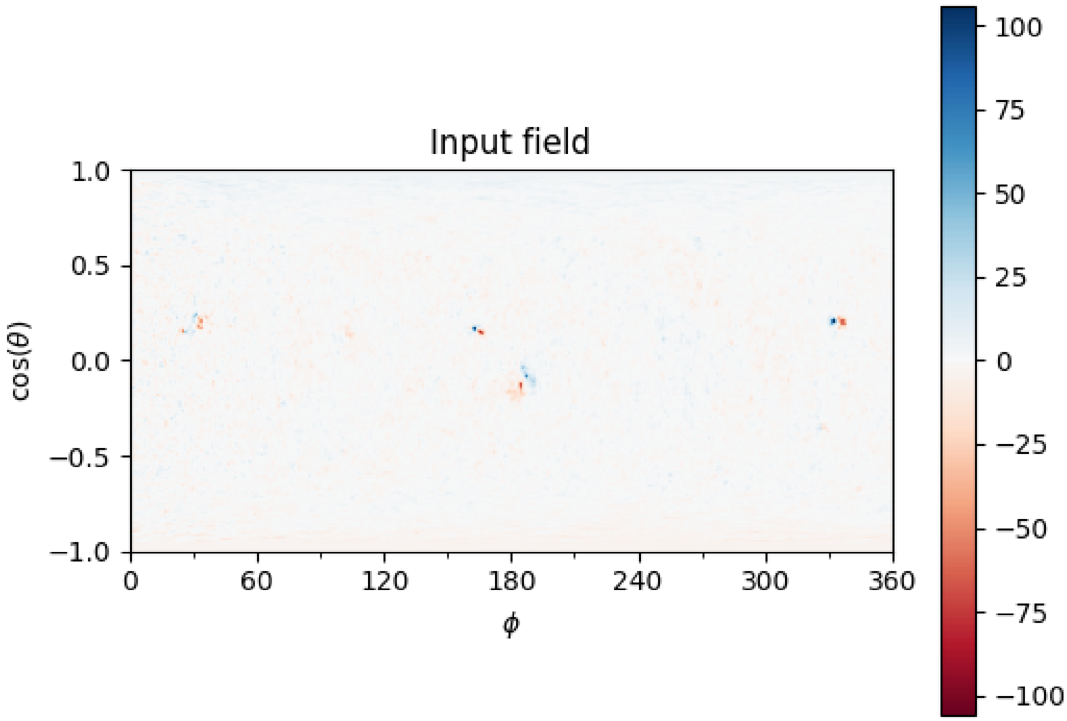
```
br = map.data
br = br - np.mean(br)
```

Create PFSS input object

```
input = pfsspy.Input(br, nr, rss)
```

Plot input magnetic field

```
fig, ax = plt.subplots()
mesh = input.plot_input(ax)
fig.colorbar(mesh)
ax.set_title('Input field')
```

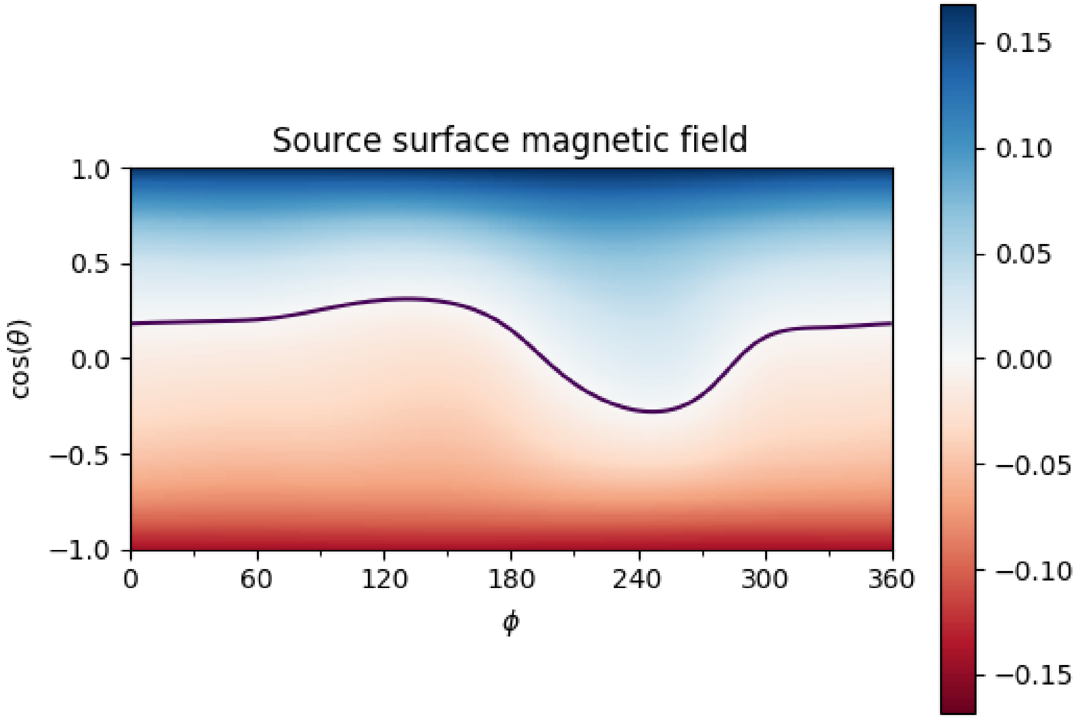


Calculate PFSS solution

```
output = pfsspy.pfss(input)
output.plot_pil(ax)
```

Plot output field

```
fig, ax = plt.subplots()
mesh = output.plot_source_surface(ax)
fig.colorbar(mesh)
output.plot_pil(ax)
ax.set_title('Source surface magnetic field')
```



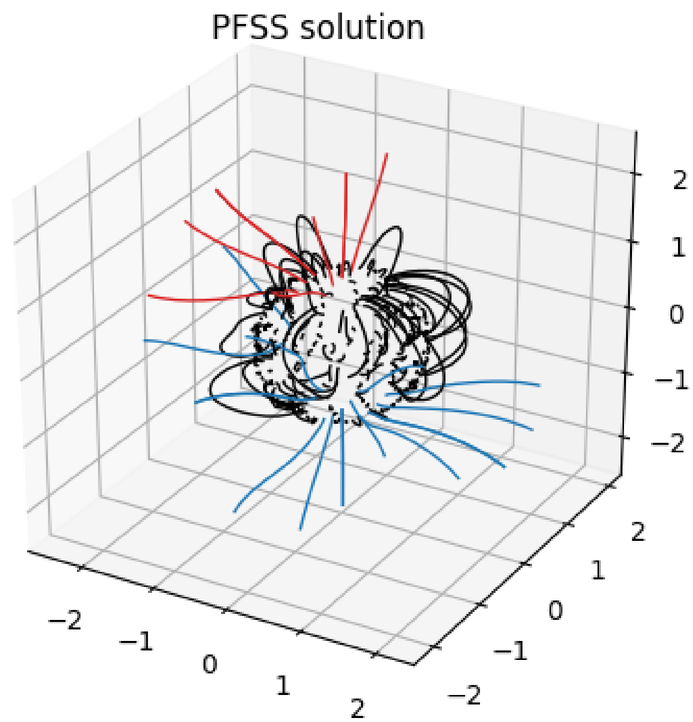
Trace some field lines

```
br, btheta, bphi = output.bg

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_aspect('equal')

# Loop through 16 values in theta and 16 values in phi
r = 1.01
for theta in np.linspace(0, np.pi, 17):
    for phi in np.linspace(0, 2 * np.pi, 17):
        x0 = np.array([r * np.cos(phi),
                       r * np.sin(theta) * np.sin(phi),
                       r * np.cos(theta) * np.sin(phi)])
        field_line = output.trace(x0)
        color = {0: 'black', -1: 'tab:blue', 1: 'tab:red'}.get(field_line.polarity)
        ax.plot(field_line.x / const.R_sun,
                field_line.y / const.R_sun,
                field_line.z / const.R_sun,
                color=color, linewidth=1)

# Add inner and outer boundary circles
ax.set_title('PFSS solution')
plt.show()
```

(continues on next page)

(continued from previous page)

```
# sphinx_gallery_thumbnail_number = 3
```



Total running time of the script: (0 minutes 22.873 seconds)

and for the helper modules (behind the scense!) see

1.3 Helper modules

1.3.1 pfsspy.plot Module

Functions

contour(phi, costheta, field, levels[, ax])	
radial_cut(phi, costheta, field[, ax])	

contour

`pfsspy.plot.`**`contour`**(*phi*, *costheta*, *field*, *levels*, *ax=None*)

radial_cut

`pfsspy.plot.`**`radial_cut`**(*phi*, *costheta*, *field*, *ax=None*)

1.3.2 pfsspy.coords Module

Helper functions for coordinate transformations used in the PFSS domain.

The PFSS solution is calculated on a "strumfric" grid defined by

- $\rho = \log(r)$
- $s = \cos(\theta)$
- ϕ

where r, θ, ϕ are spherical cooridnates that have ranges

- $1 < r < r_{ss}$
- $0 < \theta < \pi$
- $0 < \phi < 2\pi$

The transformation between cartesian coordinates used by the tracer and the above coordinates is given by

- $x = r\sin(\theta)\cos(\phi)$
- $y = r\sin(\theta)\sin(\phi)$
- $z = r\cos(\theta)$

Functions

`cart2strum`(x, y, z)	Convert cartesian coordinates to strumfric coordinates.
`strum2cart`(rho, s, phi)	Convert strumfric coordinates to cartesian coordinates.

cart2strum

`pfsspy.coords.`**`cart2strum`**(*x*, *y*, *z*)
Convert cartesian coordinates to strumfric coordinates.

strum2cart

`pfsspy.coords.`**`strum2cart`**(*rho*, *s*, *phi*)
Convert strumfric coordinates to cartesian coordinates.

1.4 Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index