
pfsspy Documentation

pfsspy contributors

Apr 04, 2019

Contents

1	Improving performance	3
2	Citing	5
3	Code reference	7
	Python Module Index	29

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

Improving performance

pfsspy automatically detects an installation of `numba`, which compiles some of the numerical code to speed up pfss calculations. To enable this simply `install numba` and use pfsspy as normal.

CHAPTER 2

Citing

If you use pfsspy in work that results in publication, please cite the archived code at *both*

- <https://zenodo.org/record/2566462>
- <https://zenodo.org/record/1472183>

Citation details can be found at the lower right hand of each web page.

For the main user-facing code and a changelog see

3.1 pfsspy Package

3.1.1 Functions

<code>load_output(file)</code>	Load a saved output file.
<code>pfss(input)</code>	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*]

3.1.2 Classes

<i>FieldLine</i> (*args[, copy])	A single magnetic field line.
<i>Grid</i> (ns, nphi, nr, rss)	Grid on which the solution is calculated.
<i>Input</i> (br, nr, rss[, dtime])	Input to PFSS modelling.
<i>Output</i> (alr, als, alp, grid[, dtime])	Output of PFSS modelling.

FieldLine

class pfsspy.**FieldLine** (*args, copy=True, **kwargs)
Bases: `astropy.coordinates.sky_coordinate.SkyCoord`

A single magnetic field line.

This is a sub-class of `astropy.coordinates.SkyCoord`. For more details on

Attributes

representation_type [str] Coordinate system representation. By default is 'cartesian', but can also be manually set to 'spherical'.

x, y, z : Field line cartesian coordinates. Can only be accessed if *representation_type* is 'cartesian'.

r, theta, phi : field line spherical coordinates. Can only be accessed if *representation_type* is 'spherical'.

Attributes Summary

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

Attributes Documentation

expansion_factor

Magnetic field expansion factor.

The expansion factor is defined 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 $\text{sign}(B_r)$ 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 $(\cos(\theta), \phi, \log(r))$. See `pfsspy.coords` for more information.

Attributes Summary

<i>dp</i>	Cell size in ϕ .
<i>dr</i>	Cell size in $\log(r)$.
<i>ds</i>	Cell size in $\cos(\theta)$.
<i>pc</i>	Location of the centre of cells in ϕ .
<i>pg</i>	Location of the edges of grid cells in ϕ .
<i>rc</i>	Location of the centre of cells in $\log(r)$.
<i>rg</i>	Location of the edges of grid cells in $\log(r)$.
<i>sc</i>	Location of the centre of cells in $\cos(\theta)$.
<i>sg</i>	Location of the edges of grid cells in $\cos(\theta)$.

Attributes Documentation**dp**

Cell size in ϕ .

dr

Cell size in $\log(r)$.

ds

Cell size in $\cos(\theta)$.

pc

Location of the centre of cells in ϕ .

pg

Location of the edges of grid cells in ϕ .

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, dtime=None*)
Bases: `object`
Input to PFSS modelling.

Warning: The input must be on a regularly spaced grid in ϕ and $s = \cos(\theta)$. See [pfsspy.coords](#) for more information on the coordinate system.

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 to calculate the PFSS solution on.

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

dtime [datetime, optional] Datetime at which the input map was measured. If given it is attached to the output and any field lines traced from the output.

Methods Summary

<code>plot_input([ax])</code>	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, dtime=None*)
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 calculated on.

dttime [datetime, optional] Datetime at which the input was measured.

Attributes Summary

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

Methods Summary

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

Attributes Documentation

al

Vector potential times cell edge lengths.

Returns $ar \cdot L_r$, $as \cdot L_s$, $ap \cdot L_p$ on cell edges.

bc

B on the centres of the cell faces.

bg

B as a (weighted) averaged on grid points.

Returns

br [array]

bs [array]

bp [array]

source_surface_br

Br on the source surface.

Methods Documentation

plot_pil (*ax=None*, ***kwargs*)

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.

****kwargs** : Keyword arguments are handed to *ax.contour*.

plot_source_surface (*ax=None, **kwargs*)

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.

kwargs : Additional keyword arguments are handed to *pcolormesh* that renders the source surface. A useful option here is handing *rasterized=True* to rasterize the image.

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*. *x0* **must** be a cartesian coordinate. See [pfsspy.coords](#) for more information on coordinate transforms, and helper functions for transforming between coordinate systems.

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

Parameters

x0 [array] Starting coordinate, in cartesian coordinates. [pfsspy.coords](#) can be used to convert from spherical coordinates to cartesian coordinates and vice versa.

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

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

Returns

fl [[FieldLine](#)]

3.2 Changelog

3.2.1 0.2.0

- [pfsspy.Input](#) and [pfsspy.Output](#) now take the optional keyword argument *dtime*, which stores the datetime on which the magnetic field measurements were made. This is then propagated to the *obstime* attribute of computed field lines, allowing them to be transformed into coordinate systems other than Carrington frames.
- [pfsspy.FieldLine](#) no longer overrides the SkyCoord `__init__`; this should not matter to users, as FieldLine objects are constructed internally by calling [pfsspy.Output.trace\(\)](#)

3.2.2 0.1.5

- [Output.plot_source_surface](#) now accepts keyword arguments that are given to Matplotlib to control the plotting of the source surface.

3.2.3 0.1.4

- Added more explanatory comments to the examples
- Corrected the dipole solution calculation
- Added `pfsspy.coords.sph2cart()` to transform from spherical to cartesian coordinates.

3.2.4 0.1.3

- `pfsspy.Output.plot_pil()` now accepts keyword arguments that are given to Matplotlib to control the style of the contour.
- `pfsspy.FieldLine.expansion_factor` is now cached, and is only calculated once if accessed multiple times.

for usage examples see

3.3 pfsspy examples

Note: Click [here](#) to download the full example code

3.3.1 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
import pfsspy.coords as coords
```

To start with we need to construct an input for the PFSS model. To do this, first set up a regular 2D grid in (ϕ, s) , where $s = \cos(\theta)$ and (ϕ, θ) are the standard spherical coordinate system angular coordinates. In this case the resolution is (360×180) .

```
nphi = 360
ns = 180

phi = np.linspace(0, 2 * np.pi, nphi)
s = np.linspace(-1, 1, ns)
s, phi = np.meshgrid(s, phi)
```

Now we can take the grid and calculate the boundary condition magnetic field.

```
def dipole_Br(r, s):
    return 2 * s / r**3

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

The PFSS solution is calculated on a regular 3D grid in (ϕ , s , ρ), where $\rho = \ln(r)$, and r is the standard spherical radial coordinate. We need to define the number of ρ grid points, and the source surface radius.

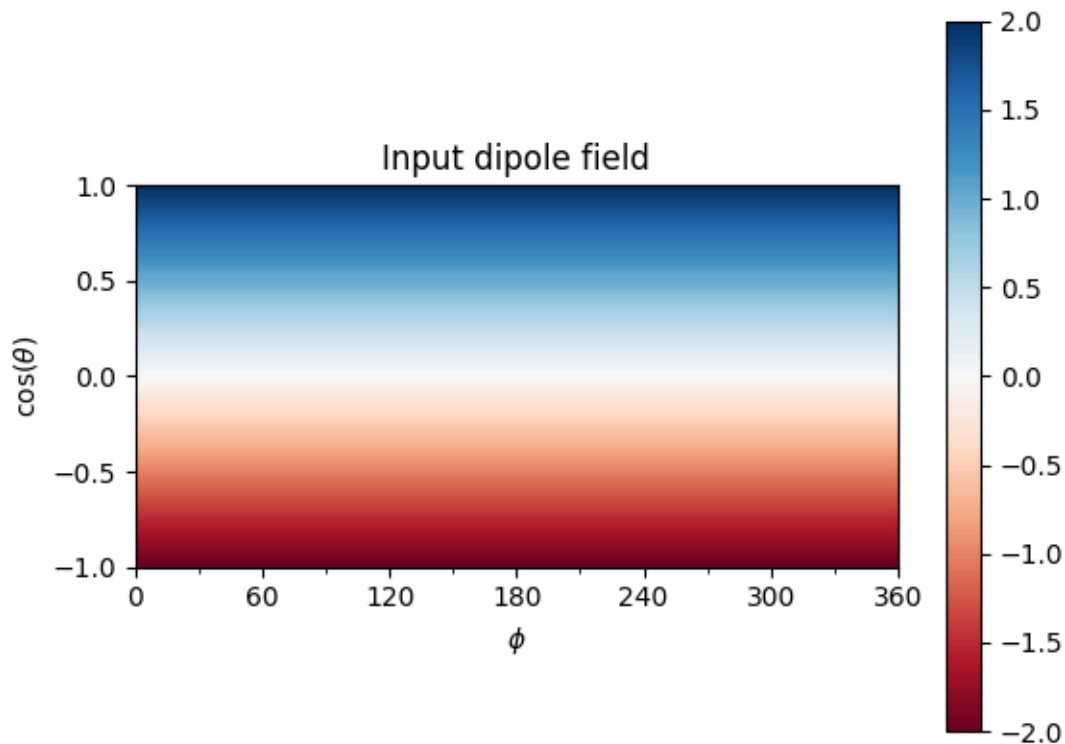
```
nrho = 50
rss = 2.5
```

From the boundary condition, number of radial grid points, and source surface, we now construct an Input object that stores this information

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

Using the Input object, plot the input field

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



Now calculate the PFSS solution.

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

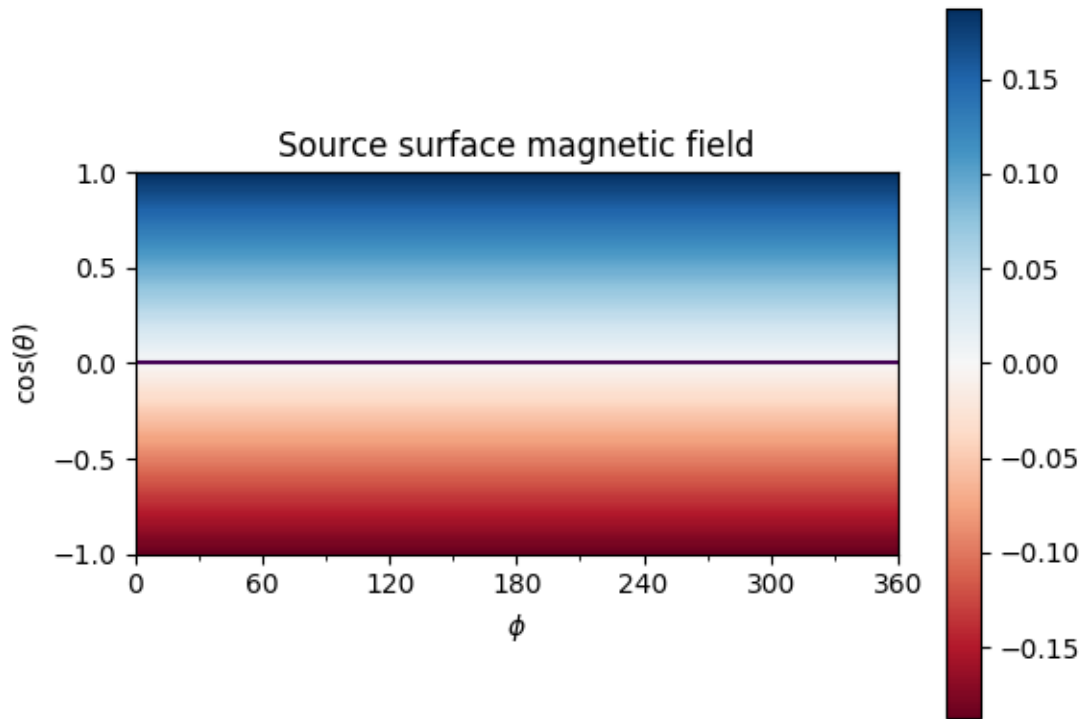
Using the Output object we can plot the source surface field, and the polarity inversion line.

```
fig, ax = plt.subplots()
mesh = output.plot_source_surface(ax)
```

(continues on next page)

(continued from previous page)

```
fig.colorbar(mesh)
output.plot_pil(ax)
ax.set_title('Source surface magnetic field')
```



Finally, using the 3D magnetic field solution we can trace some field lines. In this case 32 points equally spaced in theta are chosen and traced from the source surface outwards.

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

# Take 32 start points spaced equally in theta
r = 1.01
phi = np.pi / 2
for theta in np.linspace(0, np.pi, 33):
    x0 = coords.sph2cart(r, theta, phi)
    field_line = output.trace(np.array(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)

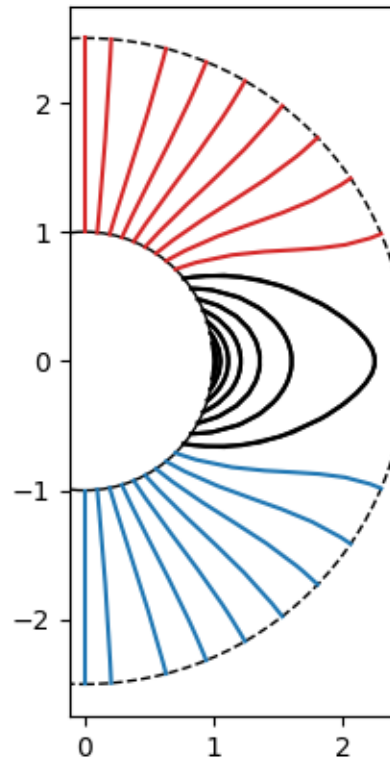
# Add inner and outer boundary circles
ax.add_patch(mpatch.Circle((0, 0), 1, color='k', fill=False))
ax.add_patch(mpatch.Circle((0, 0), input.grid.rss, color='k', linestyle='--',
                           fill=False))
```

(continues on next page)

(continued from previous page)

```
ax.set_title('PFSS solution for a dipole source field')
plt.show()
```

PFSS solution for a dipole source field



Total running time of the script: (0 minutes 5.447 seconds)

Note: Click [here](#) to download the full example code

3.3.2 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 sunpy.map

import pfsspy
import pfsspy.coords as coords
```

Load a GONG magnetic field map. If 'gong.fits' is present in the current directory, just use that, otherwise download a sample GONG map.

```
if not os.path.exists('190310t0014gong.fits') and not os.path.exists('190310t0014gong.
↳fits.gz'):
    import urllib.request
    urllib.request.urlretrieve(
        'https://gong2.nso.edu/oQR/zqs/201903/mrzqs190310/mrzqs190310t0014c2215_333.
↳fits.gz',
        '190310t0014gong.fits.gz')

if not os.path.exists('190310t0014gong.fits'):
    import gzip
    with gzip.open('190310t0014gong.fits.gz', 'rb') as f:
        with open('190310t0014gong.fits', 'wb') as g:
            g.write(f.read())
```

We can now use SunPy to load the GONG fits file, and extract the magnetic field data.

The mean is subtracted to enforce $\text{div}(\mathbf{B}) = 0$ on the solar surface: n.b. it is not obvious this is the correct way to do this, so use the following lines at your own risk!

```
[[br, header]] = sunpy.io.fits.read('190310t0014gong.fits')
br = br - np.mean(br)
```

GONG maps have their LH edge at -180deg in Carrington Longitude, so roll to get it at 0deg. This way the input magnetic field is in a Carrington frame of reference, which matters later when lining the field lines up with the AIA image.

```
br = np.roll(br, header['CRVAL1'] + 180, axis=1)
```

The PFSS solution is calculated on a regular 3D grid in (phi, s, rho), where $\rho = \ln(r)$, and r is the standard spherical radial coordinate. We need to define the number of rho grid points, and the source surface radius.

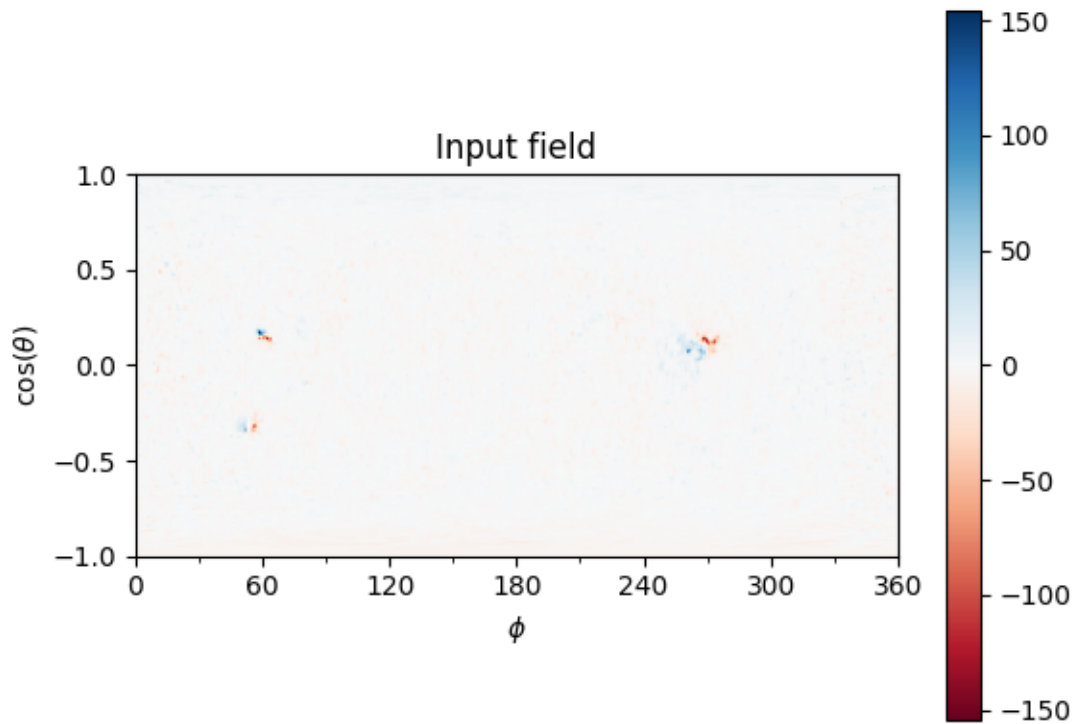
```
nrho = 60
rss = 2.5
```

From the boundary condition, number of radial grid points, and source surface, we now construct an Input object that stores this information

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

Using the Input object, plot the input field

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

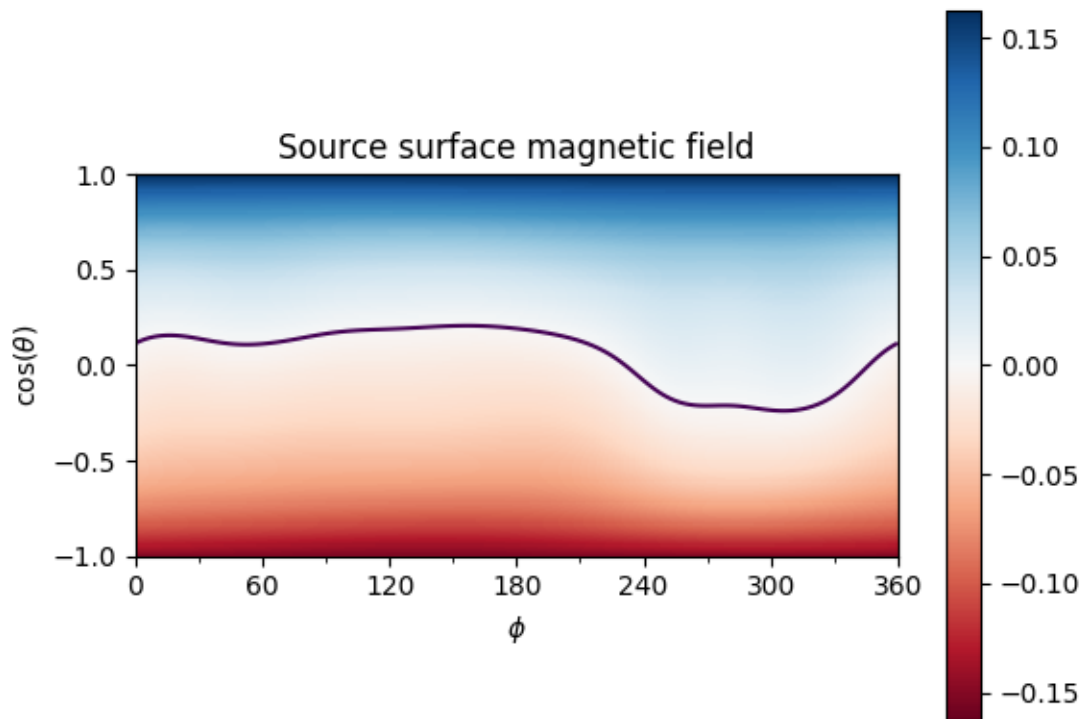


Now calculate the PFSS solution, and plot the polarity inversion line.

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

Using the Output object we can plot the source surface field, and the polarity inversion line.

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



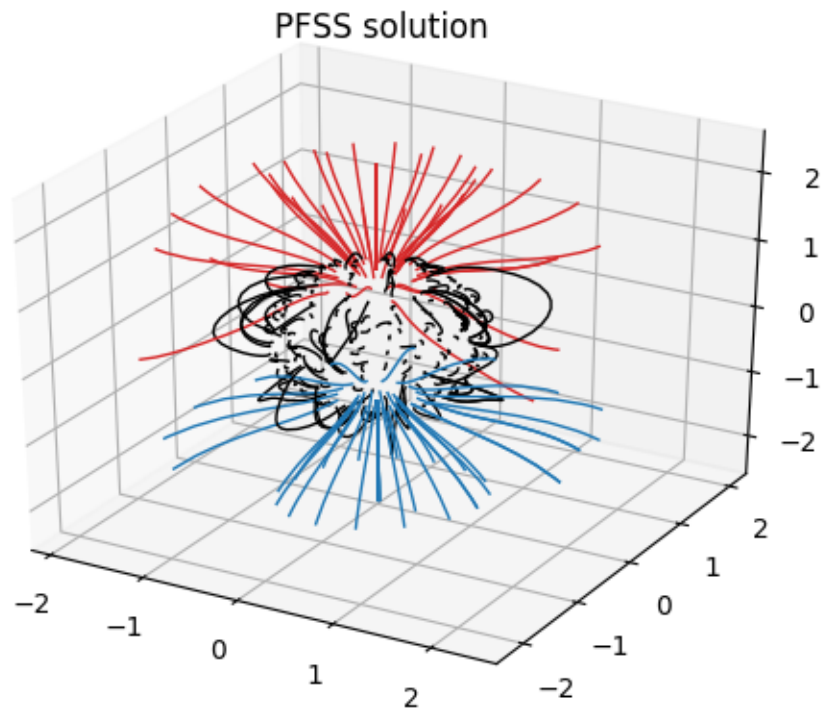
Finally, using the 3D magnetic field solution we can trace some field lines. In this case 256 points equally gridded in theta and phi are chosen and traced from the source surface outwards.

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

# 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(coords.sph2cart(r, theta, 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)

ax.set_title('PFSS solution')
plt.show()

# sphinx_gallery_thumbnail_number = 3
```



Total running time of the script: (0 minutes 14.247 seconds)

Note: Click [here](#) to download the full example code

3.3.3 Overplotting field lines on AIA maps

This example shows how to take a PFSS solution, trace some field lines, and overplot the traced field lines on an AIA 193 map.

First, we import the required modules

```
from datetime import datetime
import os

import astropy.units as u
import matplotlib.pyplot as plt
import numpy as np
import sunpy.map
import sunpy.io.fits

import pfsspy
import pfsspy.coords as coords
```

Load a GONG magnetic field map. The map date is 10/03/2019

```

if not os.path.exists('190310t0014gong.fits') and not os.path.exists('190310t0014gong.
↳fits.gz'):
    import urllib.request
    urllib.request.urlretrieve(
        'https://gong2.nso.edu/oQR/zqs/201903/mrzqs190310/mrzqs190310t0014c2215_333.
↳fits.gz',
        '190310t0014gong.fits.gz')

if not os.path.exists('190310t0014gong.fits'):
    import gzip
    with gzip.open('190310t0014gong.fits.gz', 'rb') as f:
        with open('190310t0014gong.fits', 'wb') as g:
            g.write(f.read())

```

Load the corresponding AIA 193 map

```

if not os.path.exists('AIA20190310.fits'):
    import urllib.request
    urllib.request.urlretrieve(
        'http://jsoc2.stanford.edu/data/aia/synoptic/2019/03/10/H0000/AIA20190310_
↳0000_0193.fits',
        'AIA20190310.fits')

aia = sunpy.map.Map('AIA20190310.fits')
datetime = aia.date

```

We can now use SunPy to load the GONG fits file, and extract the magnetic field data.

The mean is subtracted to enforce $\text{div}(\mathbf{B}) = 0$ on the solar surface: n.b. it is not obvious this is the correct way to do this, so use the following lines at your own risk!

```

[[br, header]] = sunpy.io.fits.read('190310t0014gong.fits')
br = br - np.mean(br)

```

GONG maps have their LH edge at -180deg in Carrington Longitude, so roll to get it at 0deg. This way the input magnetic field is in a Carrington frame of reference, which matters later when lining the field lines up with the AIA image.

```
br = np.roll(br, header['CRVAL1'] + 180, axis=1)
```

The PFSS solution is calculated on a regular 3D grid in (ϕ, s, ρ) , where $\rho = \ln(r)$, and r is the standard spherical radial coordinate. We need to define the number of grid points in ρ , and the source surface radius.

```
nrho = 60
rss = 2.5
```

From the boundary condition, number of radial grid points, and source surface, we now construct an *Input* object that stores this information

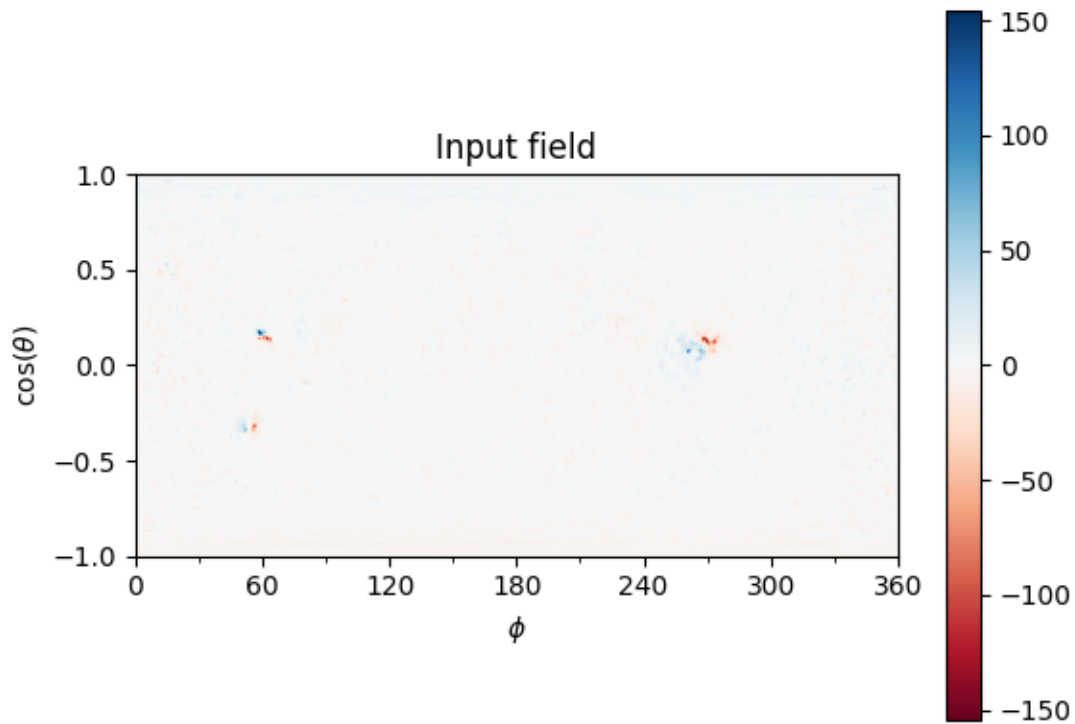
```
input = pfsspy.Input(br, nrho, rss, datetime=datetime)
```

Using the *Input* object, plot the input photospheric magnetic field

```

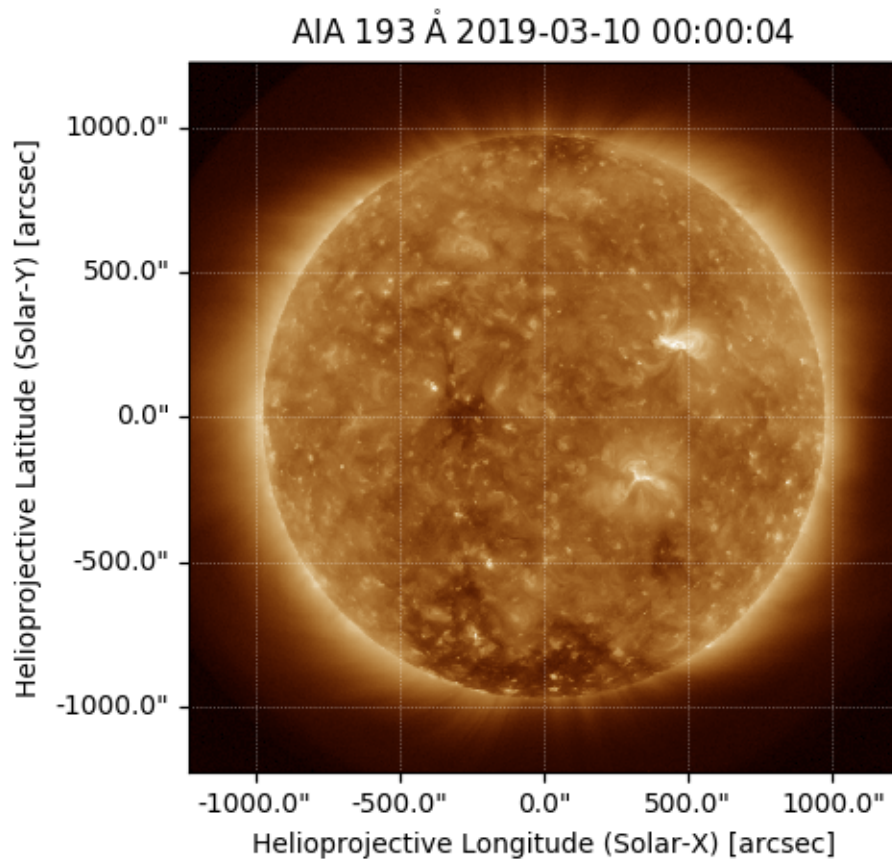
fig, ax = plt.subplots()
mesh = input.plot_input(ax)
fig.colorbar(mesh)
ax.set_title('Input field')

```



We can also plot the AIA map to give an idea of the global picture. There is a nice active region in the top right of the AIA plot, that can also be seen in the top left of the photospheric field plot above.

```
ax = plt.subplot(1, 1, 1, projection=aia)
aia.plot(ax)
```



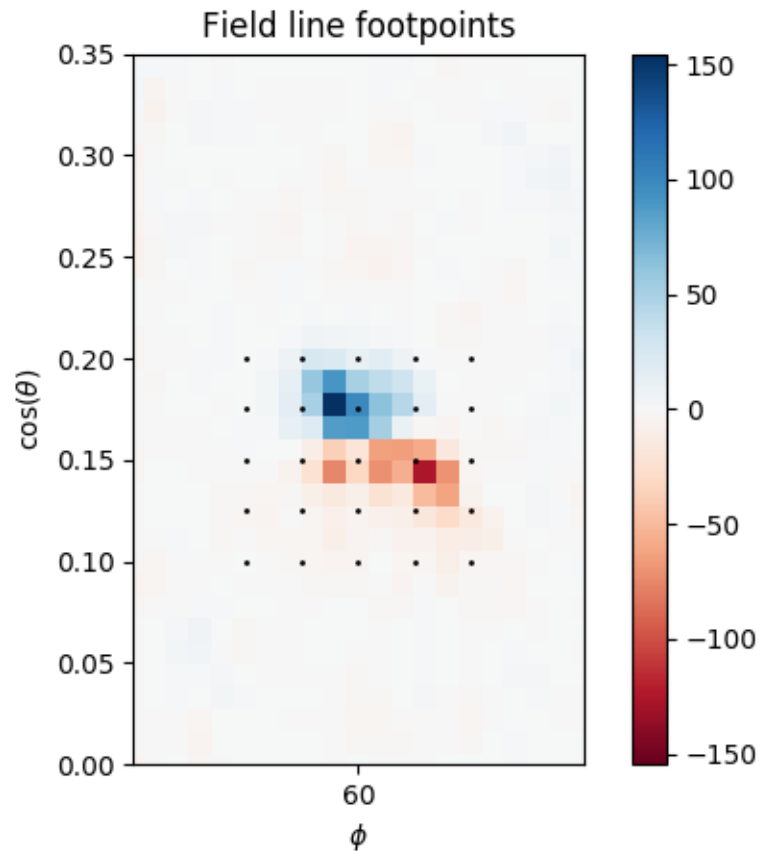
Now we construct a 10 x 10 grid of footpoints to trace some magnetic field lines from.

The figure shows a zoom in of the magnetic field map, with the footpoints overplotted. The footpoints are centered around the active region mentioned above.

```
s, phi = np.meshgrid(np.linspace(0.1, 0.2, 5),
                    np.deg2rad(np.linspace(55, 65, 5)))

fig, ax = plt.subplots()
mesh = input.plot_input(ax)
fig.colorbar(mesh)
ax.scatter(np.rad2deg(phi), s, color='k', s=1)

ax.set_xlim(50, 70)
ax.set_ylim(0, 0.35)
ax.set_title('Field line footpoints')
```



Compute the PFSS solution from the GONG magnetic field input

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

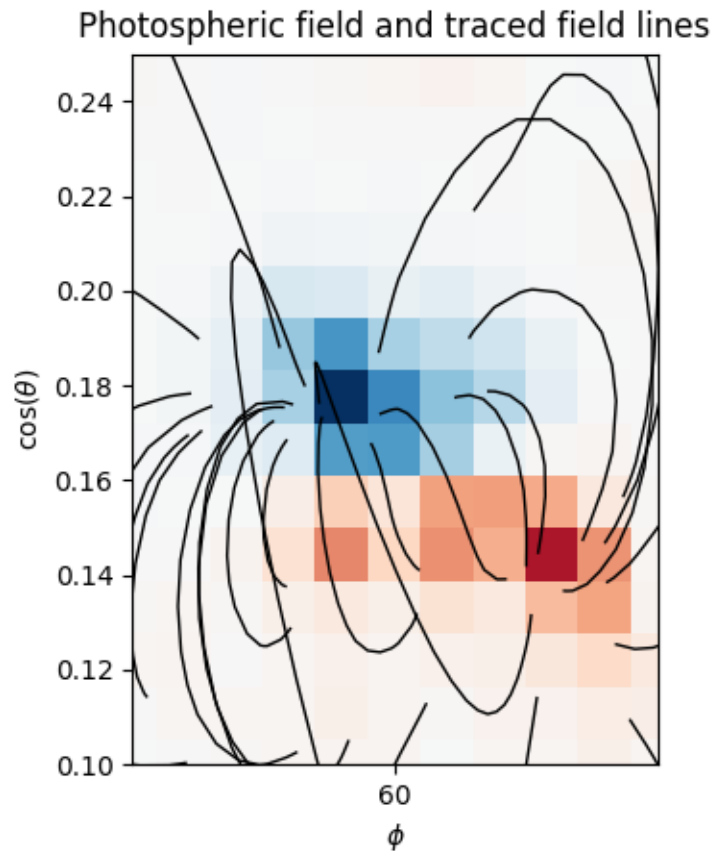
Trace field lines from the footpoints defined above. `pfsspy.coords` is used to convert the `s`, `phi` coordinates into the cartesian coordinates that are needed by the tracer.

```
fines = []
for s, phi in zip(s.ravel(), phi.ravel()):
    x0 = np.array(pfsspy.coords.strum2cart(0.01, s, phi))
    fines.append(output.trace(x0, atol=1e-8))
```

Plot the input GONG magnetic field map, along with the traced magnetic field lines.

```
fig, ax = plt.subplots()
mesh = input.plot_input(ax)
for fline in fines:
    fline.representation_type = 'spherical'
    ax.plot(fline.lon / u.deg, np.sin(fline.lat), color='black', linewidth=1)

ax.set_xlim(55, 65)
ax.set_ylim(0.1, 0.25)
ax.set_title('Photospheric field and traced field lines')
```

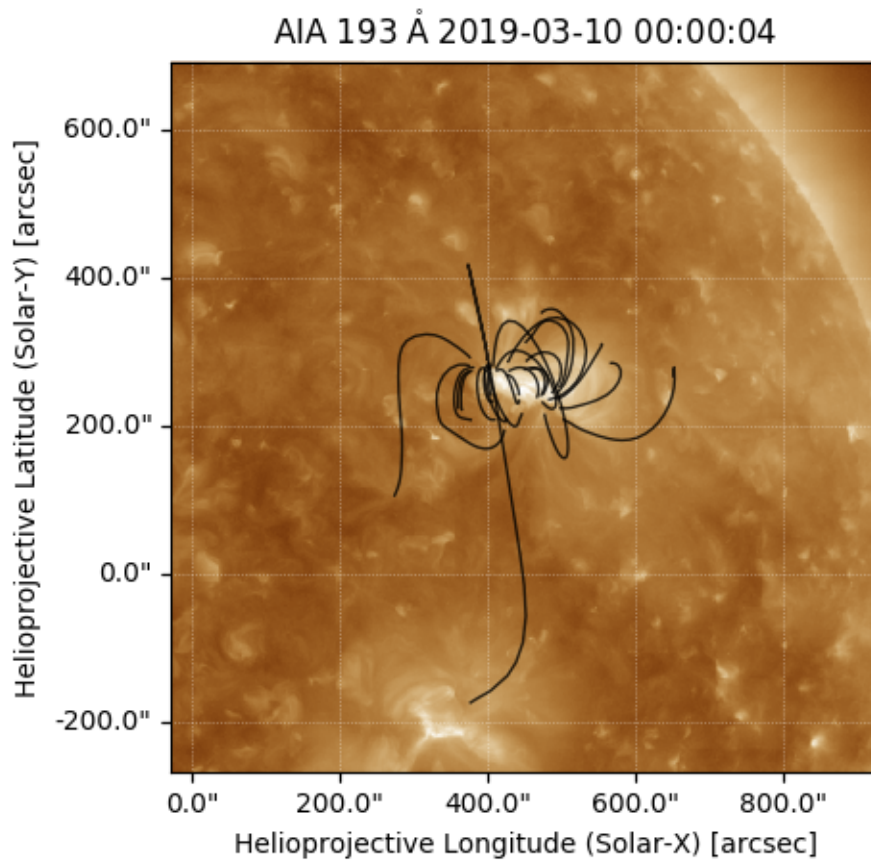


Plot the AIA map, along with the traced magnetic field lines. Inside the loop the field lines are converted to the AIA observer coordinate frame, and then plotted on top of the map.

```
fig = plt.figure()
ax = plt.subplot(1, 1, 1, projection=aia)
transform = ax.get_transform('world')
aia.plot(ax)
for fline in flines:
    fline = fline.transform_to(aia.coordinate_frame)
    Tx = fline.Tx.to(u.deg)
    Ty = fline.Ty.to(u.deg)
    ax.plot(Tx, Ty, transform=transform,
            alpha=0.8, linewidth=1, color='black')

ax.set_xlim(500, 900)
ax.set_ylim(400, 800)
plt.show()

# sphinx_gallery_thumbnail_number = 5
```



Total running time of the script: (0 minutes 10.290 seconds)

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

3.4 Helper modules

3.4.1 pfsspy.plot Module

Functions

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

Parameters

`radial_cut(phi, costheta, field[, ax])`

contour

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

Parameters

phi :

costheta :
field :
levels :
ax [Axes, optional] Axes to plot to. If None a new figure is created.
****kwargs :** Keyword arguments are handed to *ax.contour*.

radial_cut

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

3.4.2 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 coordinates 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

<code>cart2strum(x, y, z)</code>	Convert cartesian coordinates to strumfric coordinates.
<code>sph2cart(r, theta, phi)</code>	Convert spherical coordinates to cartesian coordinates.
<code>strum2cart(rho, s, phi)</code>	Convert strumfric coordinates to cartesian coordinates.

cart2strum

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

sph2cart

`pfsspy.coords.sph2cart(r, theta, phi)`
 Convert spherical coordinates to cartesian coordinates.

strum2cart

`pfsspy.coords.strum2cart` (*rho, s, phi*)

Convert strumfric coordinates to cartesian coordinates.

3.5 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

p

`pfsspy`, [7](#)
`pfsspy.coords`, [27](#)
`pfsspy.plot`, [26](#)

A

al (*pfsspy.Output attribute*), 11

B

bc (*pfsspy.Output attribute*), 11

bg (*pfsspy.Output attribute*), 11

C

cart2strum() (*in module pfsspy.coords*), 27

contour() (*in module pfsspy.plot*), 26

D

dp (*pfsspy.Grid attribute*), 9

dr (*pfsspy.Grid attribute*), 9

ds (*pfsspy.Grid attribute*), 9

E

expansion_factor (*pfsspy.FieldLine attribute*), 8

F

FieldLine (*class in pfsspy*), 8

G

Grid (*class in pfsspy*), 9

I

Input (*class in pfsspy*), 10

is_open (*pfsspy.FieldLine attribute*), 8

L

load_output() (*in module pfsspy*), 7

O

Output (*class in pfsspy*), 10

P

pc (*pfsspy.Grid attribute*), 9

pfss() (*in module pfsspy*), 7

pfsspy (*module*), 7

pfsspy.coords (*module*), 27

pfsspy.plot (*module*), 26

pg (*pfsspy.Grid attribute*), 9

plot_input() (*pfsspy.Input method*), 10

plot_pil() (*pfsspy.Output method*), 11

plot_source_surface() (*pfsspy.Output method*),
11

polarity (*pfsspy.FieldLine attribute*), 9

R

radial_cut() (*in module pfsspy.plot*), 27

rc (*pfsspy.Grid attribute*), 9

rg (*pfsspy.Grid attribute*), 9

S

save() (*pfsspy.Output method*), 12

sc (*pfsspy.Grid attribute*), 9

sg (*pfsspy.Grid attribute*), 10

source_surface_br (*pfsspy.Output attribute*), 11

sph2cart() (*in module pfsspy.coords*), 27

strum2cart() (*in module pfsspy.coords*), 28

T

trace() (*pfsspy.Output method*), 12