mplstereonet Documentation

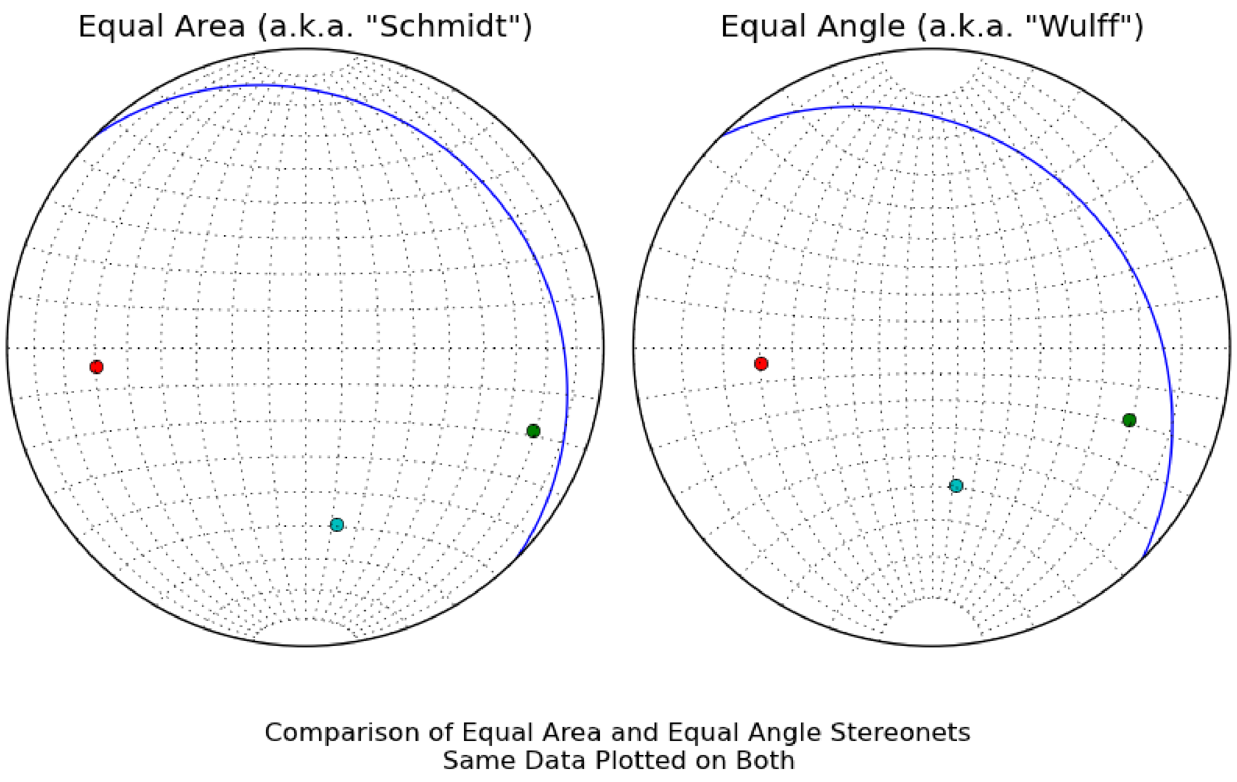
Release 0.6-dev

Joe Kington

Apr 19, 2020

Contents

`mplstereonet` provides lower-hemisphere equal-area and equal-angle stereonets for matplotlib.



CHAPTER 1

Install

`mplstereonet` can be installed from PyPi using `pip` by:

```
pip install mplstereonet
```

Alternatively, you can download the source and install locally using (from the main directory of the repository):

```
python setup.py install
```

If you're planning on developing `mplstereonet` or would like to experiment with making local changes, consider setting up a development installation so that your changes are reflected when you import the package:

```
python setup.py develop
```

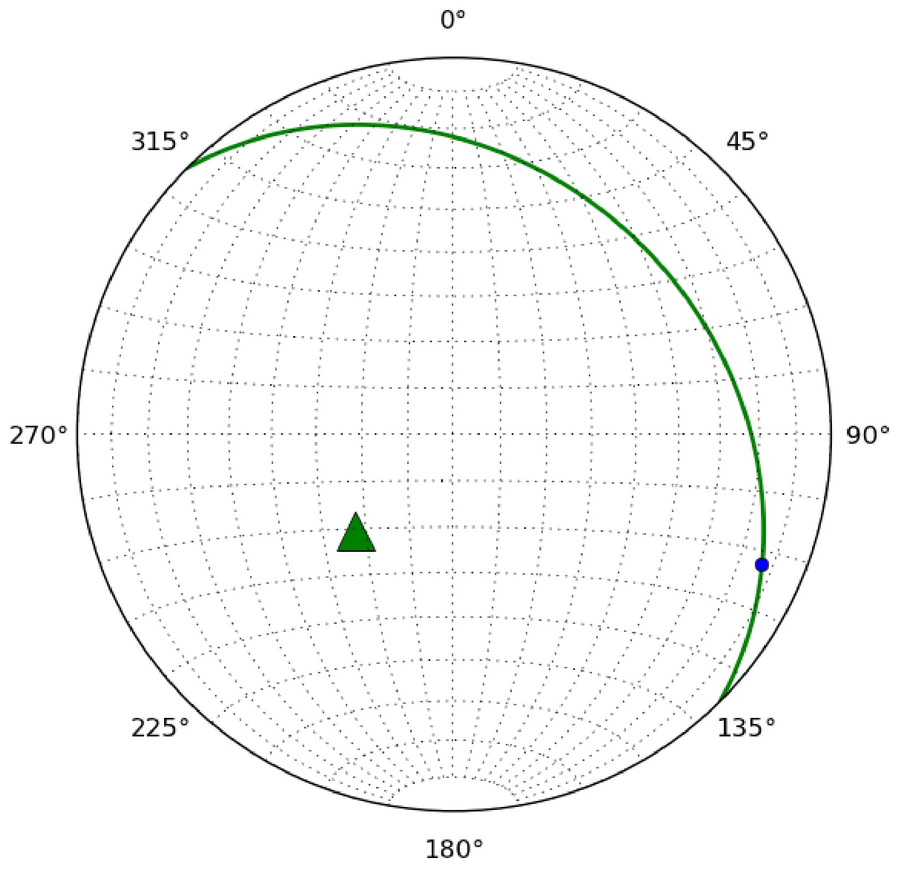
CHAPTER 2

Basic Usage

In most cases, you'll want to `import mplstereonet` and then make an axes with `projection="stereonet"` (By default, this is an equal-area stereonet). Alternately, you can use `mplstereonet.subplots`, which functions identically to `matplotlib.pyplot.subplots`, but creates stereonet axes.

As an example:

```
import matplotlib.pyplot as plt
import mplstereonet

fig = plt.figure()
ax = fig.add_subplot(111, projection='stereonet')

strike, dip = 315, 30
ax.plane(strike, dip, 'g-', linewidth=2)
ax.pole(strike, dip, 'g^', markersize=18)
ax.rake(strike, dip, -25)
ax.grid()

plt.show()
```



Planes, lines, poles, and rakes can be plotted using axes methods (e.g. `ax.line(plunge, bearing)` or `ax.rake(strike, dip, rake_angle)`).

All planar measurements are expected to follow the right-hand-rule to indicate dip direction. As an example, 315/30S would be 135/30 following the right-hand rule.

CHAPTER 3

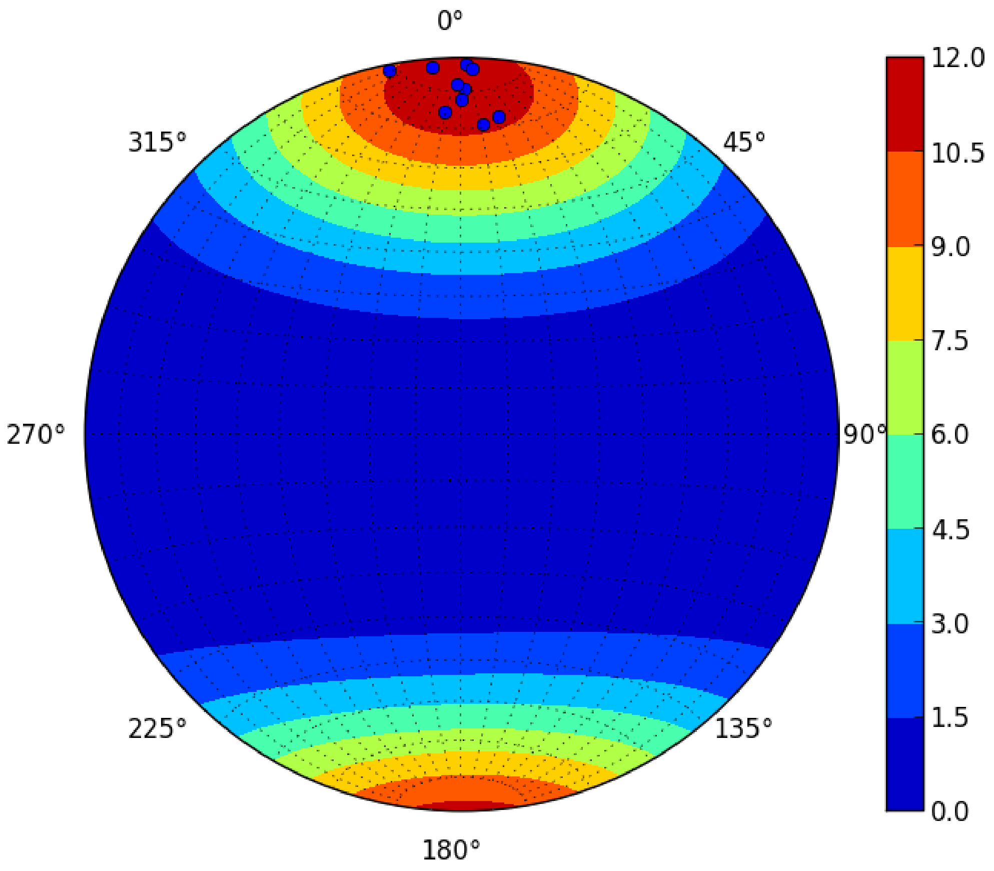
Density Contouring

mplstereonet also provides a few different methods of producing contoured orientation density diagrams.

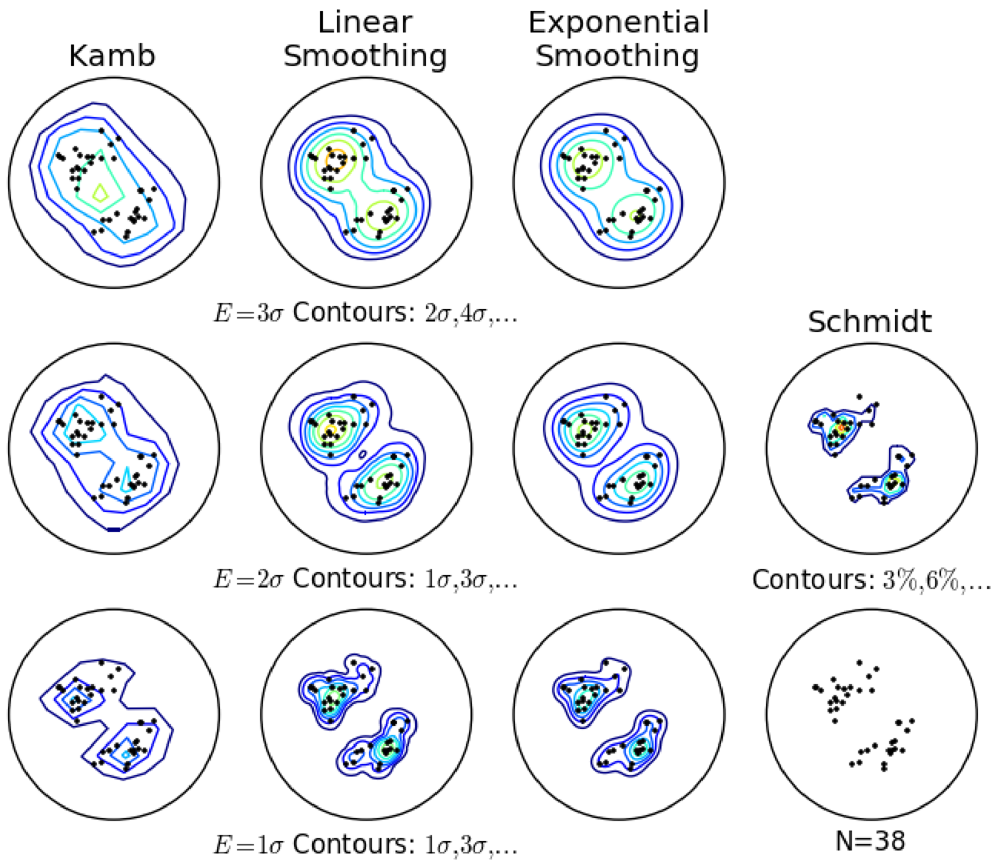
The ax.density_contour and ax.density_contourf axes methods provide density contour lines and filled density contours, respectively. “Raw” density grids can be produced with the mplstereonet.density_grid function.

As a basic example:

```python
import matplotlib.pyplot as plt
import numpy as np
import mplstereonet

fig, ax = mplstereonet.subplots()

strike, dip = 90, 80
num = 10
strikes = strike + 10 * np.random.randn(num)
dips = dip + 10 * np.random.randn(num)

cax = ax.density_contourf(strikes, dips, measurement='poles')

ax.pole(strikes, dips)
ax.grid(True)
fig.colorbar(cax)

plt.show()
```



By default, a modified Kamb method with exponential smoothing [Vollmer1995] is used to estimate the orientation density distribution. Other methods (such as the "traditional" Kamb [Kamb1956] and "Schmidt" (a.k.a. 1%) methods) are available as well. The method and expected count (in standard deviations) can be controlled by the `method` and `sigma` keyword arguments, respectively.


Kamb
Linear Smoothing
Exponential Smoothing
$E=3\sigma$ Contours: 2σ,4σ,...
Schmidt
$E=2\sigma$ Contours: 1σ,3σ,...
Contours: 3%,6%,...
$E=1\sigma$ Contours: 1σ,3σ,...
N=38

CHAPTER 4

Utilities

mplstereonet also includes a number of utilities to parse structural measurements in either quadrant or azimuth form such that they follow the right-hand-rule.

For an example, see parsing_example.py:

```
Parse quadrant azimuth measurements
"N30E" --> 30.0
"E30N" --> 60.0
"W10S" --> 260.0
"N 10 W" --> 350.0

Parse quadrant strike/dip measurements.
Note that the output follows the right-hand-rule.
"215/10" --> Strike: 215.0, Dip: 10.0
"215/10E" --> Strike: 35.0, Dip: 10.0
"215/10NW" --> Strike: 215.0, Dip: 10.0
"N30E/45NW" --> Strike: 210.0, Dip: 45.0
"E10N   20 N" --> Strike: 260.0, Dip: 20.0
"W30N/46.7 S" --> Strike: 120.0, Dip: 46.7

Similarly, you can parse rake measurements that don't follow the RHR.
"N30E/45NW 10NE" --> Strike: 210.0, Dip: 45.0, Rake: 170.0
"210 45 30N" --> Strike: 210.0, Dip: 45.0, Rake: 150.0
"N30E/45NW raking 10SW" --> Strike: 210.0, Dip: 45.0, Rake: 10.0
```

Additionally, you can find plane intersections and make other calculations by combining utility functions. See plane_intersection.py and parse_anglier_data.py for examples.

CHAPTER 5

References

CHAPTER 6

Examples

6.1 Examples

6.1.1 `axial_plane.py`

Illustrates fitting an axial plane to two clusters of dip measurements.

In this case, we're faking it by using Anglier's fault orientation data, but pretend these were bedding dips in two limbs of a fold instead of fault orientations.

The steps mimic what you'd do graphically:

1. Find the centers of the two modes of the bedding measurements
2. Fit a girdle to them to find the plunge axis of the fold
3. Find the midpoint along that girdle between the two centers
4. The axial plane will be the girdle that fits the midpoint and plunge axis of the fold.

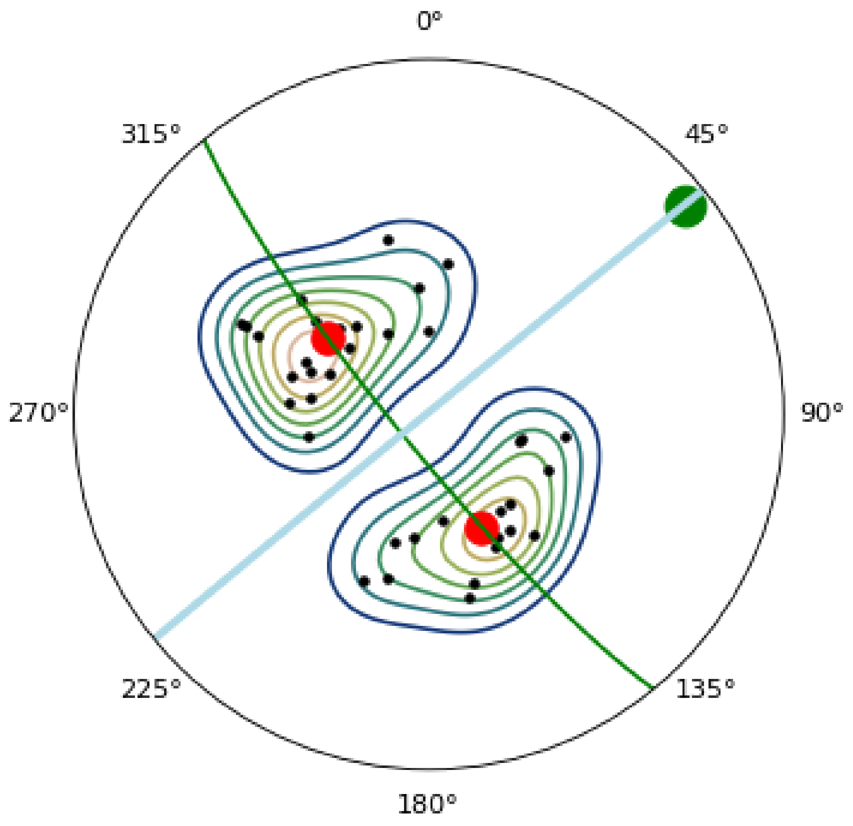
```
import matplotlib.pyplot as plt
import mplstereonet

import parse_angelier_data

# Load data from Angelier, 1979
strike, dip, rake = parse_angelier_data.load()

# Plot the raw data and contour it:
fig, ax = mplstereonet.subplots()
ax.density_contour(strike, dip, rake, measurement='rakes', cmap='gist_earth',
                    sigma=1.5)
ax.rake(strike, dip, rake, marker='.', color='black')

# Find the two modes
centers = mplstereonet.kmeans(strike, dip, rake, num=2, measurement='rakes')
```

(continues on next page)

(continued from previous page)

```
strike_cent, dip_cent = mplstereonet.geographic2pole(*zip(*centers))
ax.pole(strike_cent, dip_cent, 'ro', ms=12)

# Fit a girdle to the two modes
# The pole of this plane will be the plunge of the fold axis
axis_s, axis_d = mplstereonet.fit_girdle(*zip(*centers), measurement='radians')
ax.plane(axis_s, axis_d, color='green')
ax.pole(axis_s, axis_d, color='green', marker='o', ms=15)

# Now we'll find the midpoint. We could project the centers as rakes on the
# plane we just fit, but it's easier to get their mean vector instead.
mid, _ = mplstereonet.find_mean_vector(*zip(*centers), measurement='radians')
midx, midy = mplstereonet.line(*mid)

# Now let's find the axial plane by fitting another girdle to the midpoint
# and the pole of the plunge axis.
xp, yp = mplstereonet.pole(axis_s, axis_d)

x, y = [xp, midx], [yp, midy]
axial_s, axial_dip = mplstereonet.fit_girdle(x, y, measurement='radians')

ax.plane(axial_s, axial_dip, color='lightblue', lw=3)

plt.show()
```

Result

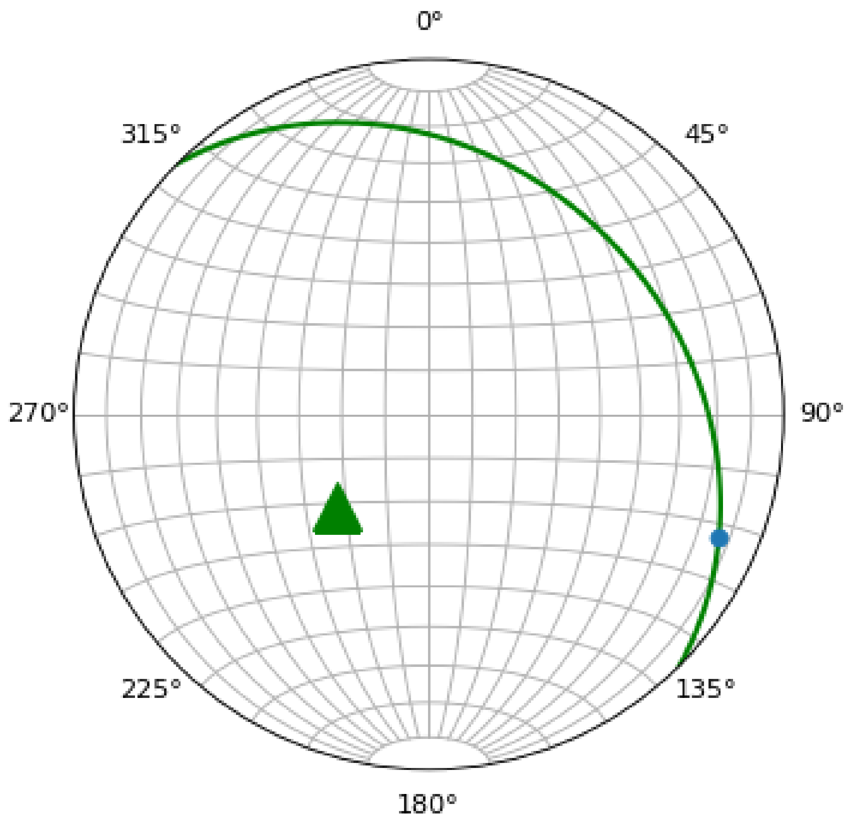


6.1.2 `basic.py`

As an example of basic functionality, let's plot a plane, the pole to the plane, and a rake along the plane.

```
import matplotlib.pyplot as plt
import mplstereonet

fig = plt.figure()
ax = fig.add_subplot(111, projection='stereonet')

# Measurements follow the right-hand-rule to indicate dip direction
strike, dip = 315, 30

ax.plane(strike, dip, 'g-', linewidth=2)
ax.pole(strike, dip, 'g^', markersize=18)
ax.rake(strike, dip, -25)

ax.grid()

plt.show()
```

Result



6.1.3 cone_aka_small_circle.py

Demonstrates plotting small circles (cones) on a stereonet.

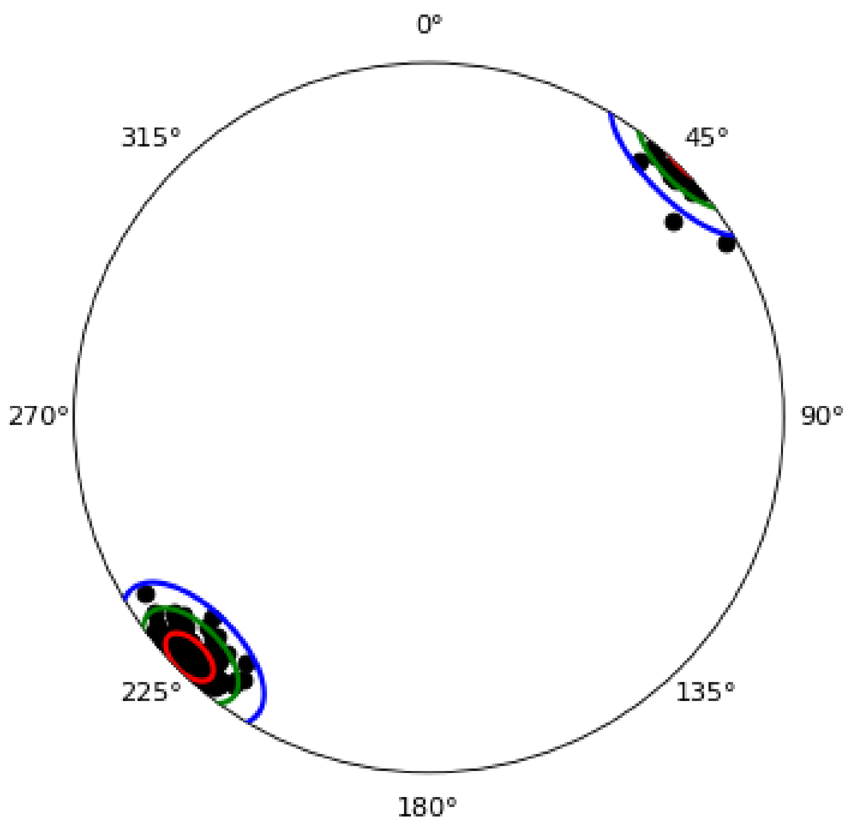
```python
import matplotlib.pyplot as plt
import numpy as np
import mplstereonet

# Generate some scattered strikes and dips
num = 100
strike0, dip0 = 315, 85
strike = np.random.normal(strike0, 5, num)
dip = np.random.normal(dip0, 5, num)

# Convert the strike/dip of the pole to plane to a plunge/bearing
plunge, bearing = mplstereonet.stereonet_math.pole2plunge_bearing(strike0, dip0)

fig, ax = mplstereonet.subplots()
ax.pole(strike, dip, color='k')

# We want the plunge and bearing repeated 3 times for three circles...
plunge, bearing = 3 * list(plunge), 3 * list(bearing)
```

(continues on next page)

(continued from previous page)

```
ax.cone(plunge, bearing, [5, 10, 15], facecolor='', zorder=4, linewidth=2,
        edgecolors=['red', 'green', 'blue'])

plt.show()
```

Result



6.1.4 `contour_angelier_data.py`

Reproduce Figure 5 from Vollmer, 1995 to illustrate different density contouring methods.

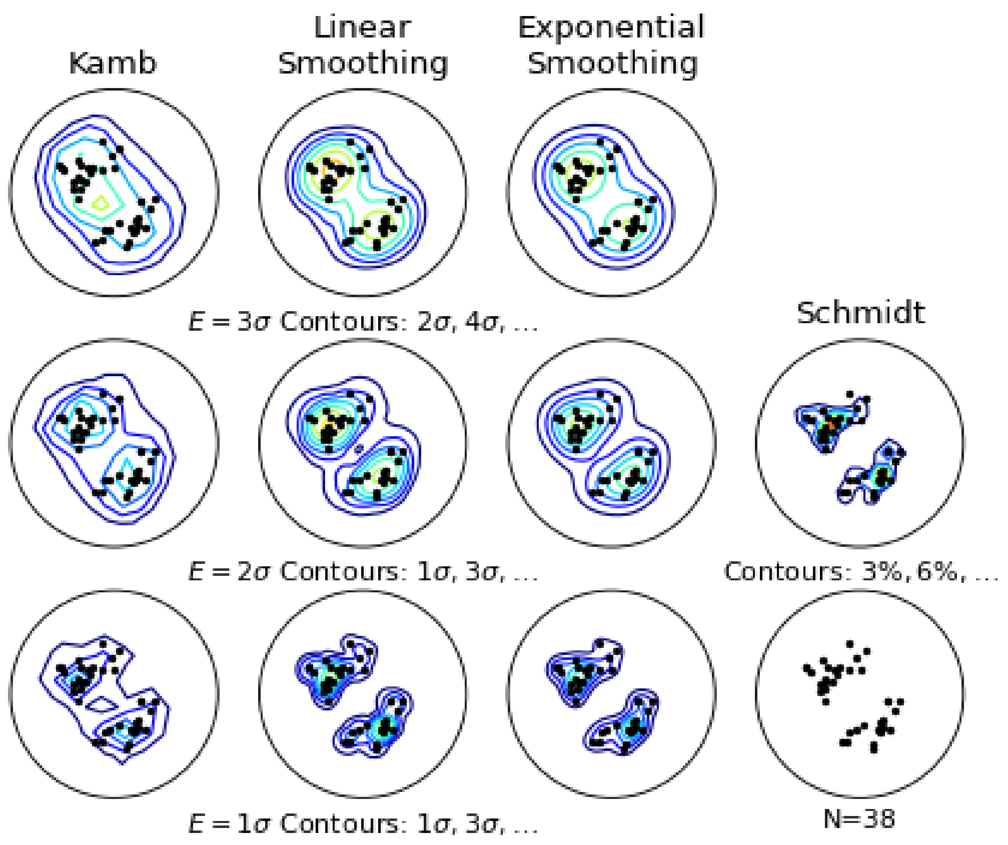
```
import matplotlib.pyplot as plt
import mplstereonet

import parse_angelier_data

def plot(ax, strike, dip, rake, **kwargs):
    ax.rake(strike, dip, rake, 'ko', markersize=2)
    ax.density_contour(strike, dip, rake, measurement='rakes', linewidths=1,
                       cmap='jet', **kwargs)

```

(continues on next page)

(continued from previous page)

```
# Load data from Angelier, 1979
strike, dip, rake = parse_angelier_data.load()

# Setup a subplot grid
fig, axes = mplstereonet.subplots(nrows=3, ncols=4)

# Hide azimuth tick labels
for ax in axes.flat:
    ax.set_azimuth_ticks([])

contours = [range(2, 18, 2), range(1, 21, 2), range(1, 22, 2)]

# "Standard" Kamb contouring with different confidence levels.
for sigma, ax, contour in zip([3, 2, 1], axes[:, 0], contours):
    # We're reducing the gridsize to more closely match a traditional
    # hand-contouring grid, similar to Kamb's original work and Vollmer's
    # Figure 5. `gridsize=10` produces a 10x10 grid of density estimates.
    plot(ax, strike, dip, rake, method='kamb', sigma=sigma,
        levels=contour, gridsize=10)

# Kamb contouring with inverse-linear smoothing (after Vollmer, 1995)
for sigma, ax, contour in zip([3, 2, 1], axes[:, 1], contours):
    plot(ax, strike, dip, rake, method='linear_kamb', sigma=sigma,
        levels=contour)
    template = r'$E={}\sigma$ Contours: ${}\sigma,{}\sigma,\ldots$'
    ax.set_xlabel(template.format(sigma, *contour[:2]))

# Kamb contouring with exponential smoothing (after Vollmer, 1995)
for sigma, ax, contour in zip([3, 2, 1], axes[:, 2], contours):
    plot(ax, strike, dip, rake, method='exponential_kamb', sigma=sigma,
        levels=contour)

# Title the different methods
methods = ['Kamb', 'Linear\nSmoothing', 'Exponential\nSmoothing']
for ax, title in zip(axes[0, :], methods):
    ax.set_title(title)

# Hide top-right axis... (Need to implement Diggle & Fisher's method)
axes[0, -1].set_visible(False)

# Schmidt contouring (a.k.a. 1%)
plot(axes[1, -1], strike, dip, rake, method='schmidt', gridsize=25,
     levels=range(3, 20, 3))
axes[1, -1].set_title('Schmidt')
axes[1, -1].set_xlabel(r'Contours: $3\%,6\%,\ldots$')

# Raw data.
axes[-1, -1].set_azimuth_ticks([])
axes[-1, -1].rake(strike, dip, rake, 'ko', markersize=2)
axes[-1, -1].set_xlabel('N={}'.format(len(strike)))

plt.show()
```

Result



6.1.5 `contour_normal_vectors.py`

Illustrates plotting normal vectors in "world" coordinates as orientations on a stereonet.

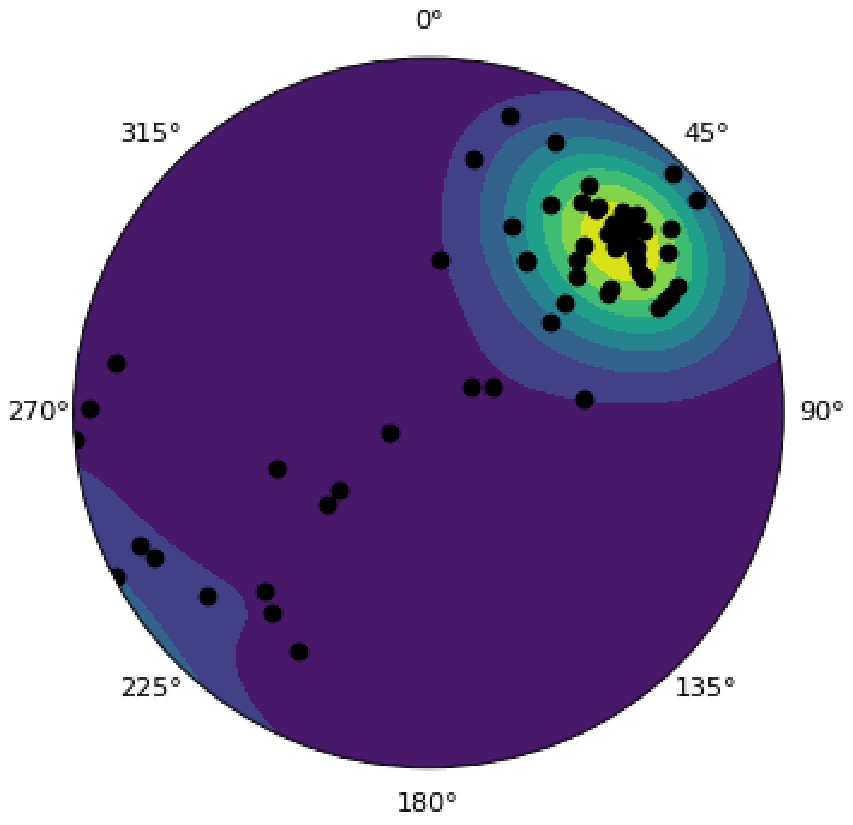
```
import numpy as np
import matplotlib.pyplot as plt
import mplstereonet

# Load in a series of normal vectors from a triangulated normal fault surface
normals = np.loadtxt('normal_vectors.txt')
x, y, z = normals.T

# Convert these to plunge/bearings for plotting.
# Alternately, we could use xyz2stereonet (it doesn't correct for bi-directional
# measurements, however) or vector2pole.
plunge, bearing = mplstereonet.vector2plunge_bearing(x, y, z)

# Set up the figure
fig = plt.figure()
ax = fig.add_subplot(111, projection='stereonet')

# Make a density contour plot of the orientations
```

(continues on next page)

(continued from previous page)

```
ax.density_contourf(plunge, bearing, measurement='lines')

# Plot the vectors as points on the stereonet.
ax.line(plunge, bearing, marker='o', color='black')

plt.show()
```

Result



6.1.6 contouring.py

A basic example of producing a density contour plot of poles to planes.

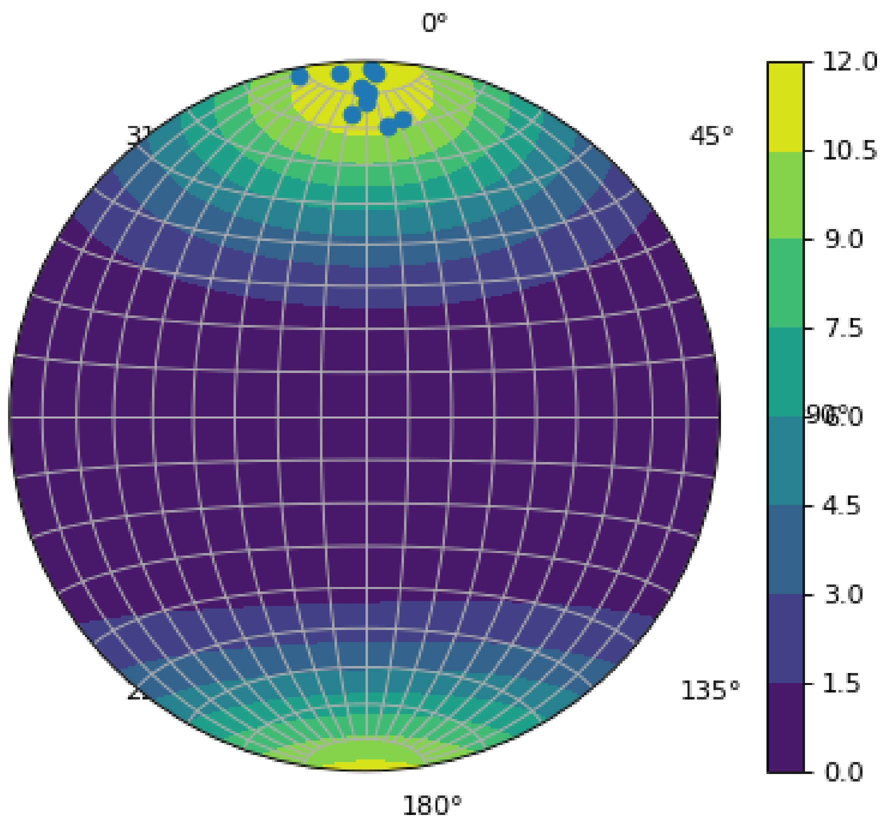
```
import matplotlib.pyplot as plt
import numpy as np
import mplstereonet

# Fix random seed so that output is consistent
np.random.seed(1977)

fig, ax = mplstereonet.subplots()
```

(continues on next page)

(continued from previous page)

```
# Generate a random scatter of planes around the given plane
# All measurements follow the right-hand-rule to indicate dip direction
strike, dip = 90, 80
num = 10
strikes = strike + 10 * np.random.randn(num)
dips = dip + 10 * np.random.randn(num)

# Create filled contours of the poles of the generated planes...
# By default this uses a modified Kamb contouring technique with exponential
# smoothing (See Vollmer, 1995)
cax = ax.density_contourf(strikes, dips, measurement='poles')

# Plot the poles as points on top of the contours
ax.pole(strikes, dips)

# Turn on a grid and add a colorbar
ax.grid(True)
fig.colorbar(cax)
plt.show()
```

Result

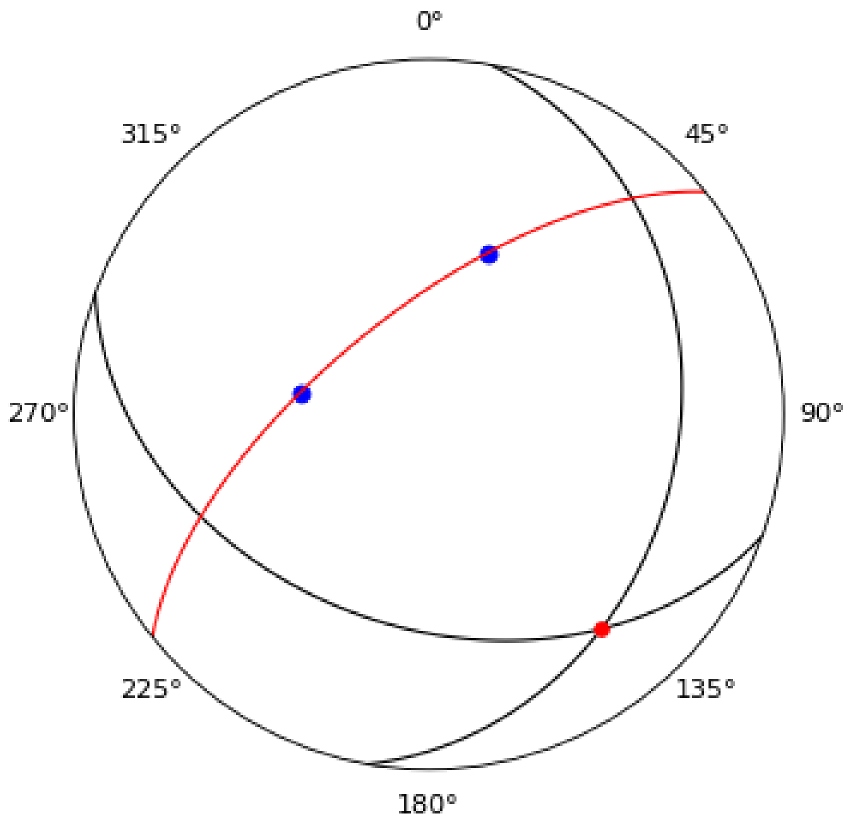


6.1.7 `cross_section_plane.py`

In this example two planes are plottet as great circles and poles. The planes are given as dip-direction/dip and converted to strike/dip. The strikes and dips are passed to the 'mplstereonet.fit_girdle()' function that calculates the best fitting plane for the poles of the planes. The resulting plane is the optimal cross-section plane for this structure. The pole of the resulting plane would correspond to the intersection-linear when looking at schistosities or the fold-axis when looking at fold-hinges.

```
import matplotlib.pyplot as plt
import numpy as np
import mplstereonet

fig, ax = mplstereonet.subplots()

dip_directions = [100, 200]
dips = [30, 40]
strikes = np.array(dip_directions) - 90

ax.pole(strikes, dips, "bo")
ax.plane(strikes, dips, color='black', lw=1)

fit_strike, fit_dip = mplstereonet.fit_girdle(strikes, dips)

ax.plane(fit_strike, fit_dip, color='red', lw=1)
ax.pole(fit_strike, fit_dip, marker='o', color='red', markersize=5)

plt.show()
```

Result



6.1.8 `equal_area_equal_angle_comparison.py`

A quick visual comparison of equal area vs. equal angle nets.

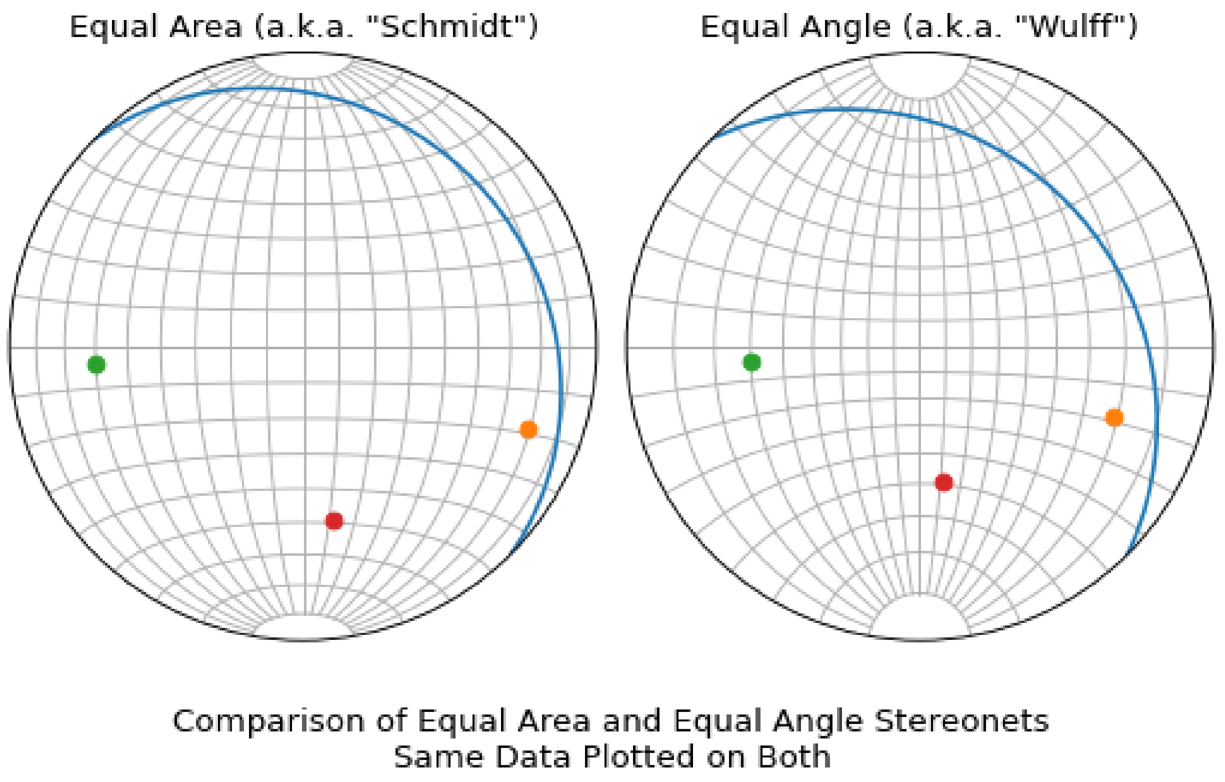
```python
import matplotlib.pyplot as plt
import mplstereonet

fig = plt.figure()

# Make an "equal area" (a.k.a. "Schmidt") stereonet
# (Lambert Azimuthal Equal Area Projection)
ax1 = fig.add_subplot(1,2,1, projection='equal_area_stereonet')

# Make an "equal angle" (a.k.a. "Wulff" or "True") stereonet
# (Stereographic projection)
ax2 = fig.add_subplot(1,2,2, projection='equal_angle_stereonet')

# Plot the same thing on both
for ax in [ax1, ax2]:
    ax.grid(True)
    ax.set_azimuth_ticklabels([])
    ax.plane(315, 20)
```

(continues on next page)

(continued from previous page)

```
    ax.line([20, 30, 40], [110, 265, 170])

ax1.set_title('Equal Area (a.k.a. "Schmidt")')
ax2.set_title('Equal Angle (a.k.a. "Wulff")')

# Make the subplots fit a bit more compactly (purely cosmetic)
fig.subplots_adjust(hspace=0, wspace=0.05, left=0.01, bottom=0.1, right=0.99)

fig.suptitle('Comparison of Equal Area and Equal Angle Stereonets\n'
             'Same Data Plotted on Both', y=0.1)
plt.show()
```

Result



6.1.9 `fault_slip_plot.py`

Illustrates two different methods of plotting fault slip data.

A fault-and-striae diagram is the traditional method. The tangent-lineation diagram follows Twiss & Unruh, 1988 (this style was originally introduced by Goldstein & Marshak, 1988 and also by Hoeppener, 1955, but both used the opposite convention for arrow direction).

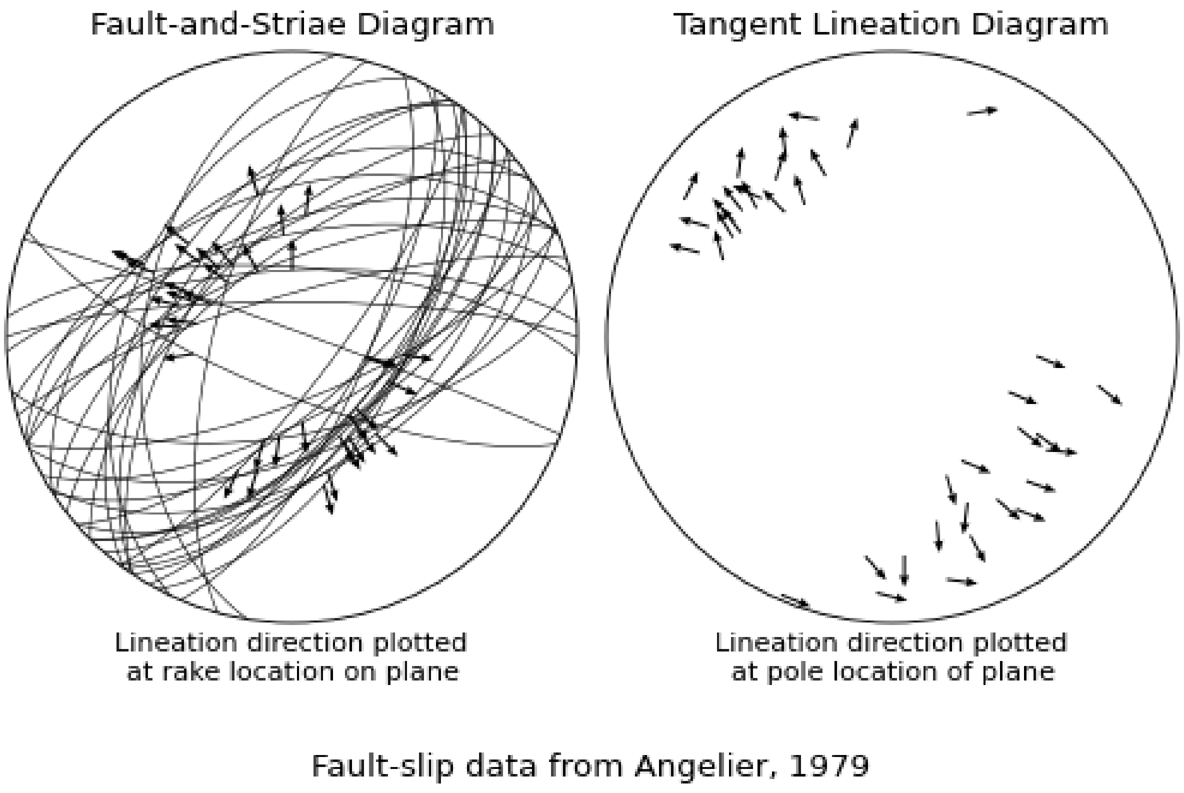
```python
import matplotlib.pyplot as plt
import numpy as np
import mplstereonet

import parse_angelier_data

def main():
    # Load data from Angelier, 1979
    strikes, dips, rakes = parse_angelier_data.load()

    params = dict(projection='stereonet', azimuth_ticks=[])
    fig, (ax1, ax2) = plt.subplots(ncols=2, subplot_kw=params)

    fault_and_striae_plot(ax1, strikes, dips, rakes)
    ax1.set_title('Fault-and-Striae Diagram')
    ax1.set_xlabel('Lineation direction plotted\nat rake location on plane')

    tangent_lineation_plot(ax2, strikes, dips, rakes)
    ax2.set_title('Tangent Lineation Diagram')
    ax2.set_xlabel('Lineation direction plotted\nat pole location of plane')

    fig.suptitle('Fault-slip data from Angelier, 1979', y=0.05)
    fig.tight_layout()

    plt.show()

def fault_and_striae_plot(ax, strikes, dips, rakes):
    """Makes a fault-and-striae plot (a.k.a. "Ball of String") for normal faults
    with the given strikes, dips, and rakes."""
    # Plot the planes
    lines = ax.plane(strikes, dips, 'k-', lw=0.5)

    # Calculate the position of the rake of the lineations, but don't plot yet
    x, y = mplstereonet.rake(strikes, dips, rakes)

    # Calculate the direction the arrows should point
    # These are all normal faults, so the arrows point away from the center
    # For thrusts, it would just be u, v = -x/mag, -y/mag
    mag = np.hypot(x, y)
    u, v = x / mag, y / mag

    # Plot the arrows at the rake locations...
    arrows = ax.quiver(x, y, u, v, width=1, headwidth=4, units='dots')
    return lines, arrows

def tangent_lineation_plot(ax, strikes, dips, rakes):
    """Makes a tangent lineation plot for normal faults with the given strikes,
    dips, and rakes."""
    # Calculate the position of the rake of the lineations, but don't plot yet
    rake_x, rake_y = mplstereonet.rake(strikes, dips, rakes)

    # Calculate the direction the arrows should point
    # These are all normal faults, so the arrows point away from the center
    # Because we're plotting at the pole location, however, we need to flip this
    # from what we plotted with the "ball of string" plot.
    mag = np.hypot(rake_x, rake_y)
    u, v = -rake_x / mag, -rake_y / mag
```

(continues on next page)

(continued from previous page)

```
    # Calculate the position of the poles
    pole_x, pole_y = mplstereonet.pole(strikes, dips)

    # Plot the arrows centered on the pole locations...
    arrows = ax.quiver(pole_x, pole_y, u, v, width=1, headwidth=4, units='dots',
                       pivot='middle')
    return arrows

if __name__ == '__main__':
    main()
```

Result



6.1.10 `fisher_stats.py`

This example shows how the Fisher statistics can be computed and displayed.

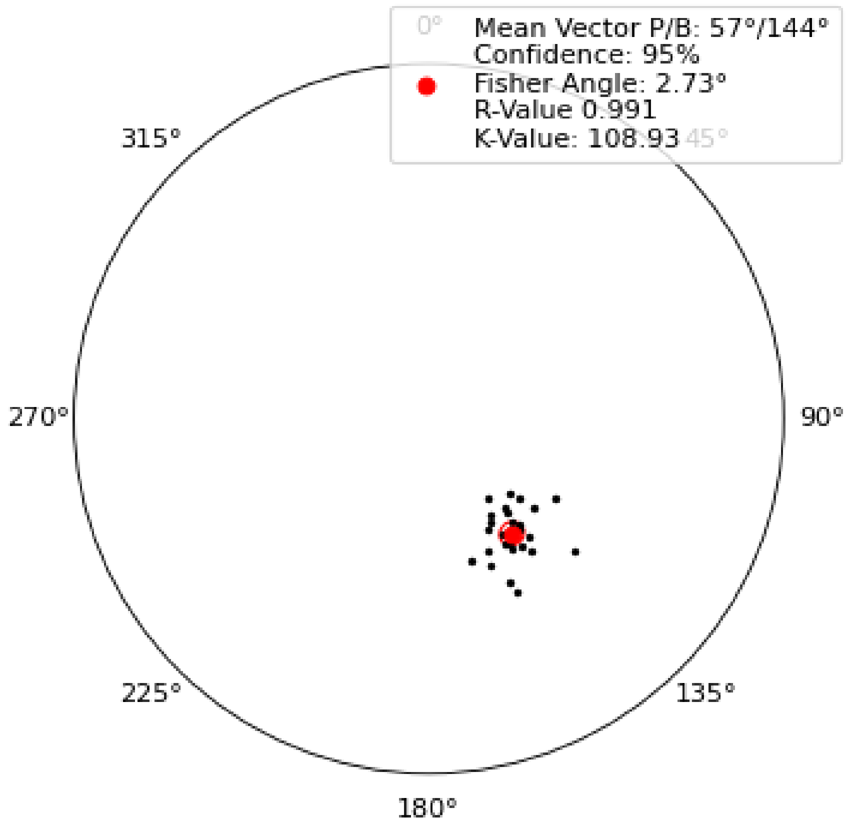
Based on example 5.21 and example 5.23 in [Fisher1993].

Data in:	Table B2	(page 279)
Mean Vector:	144.2/57.2	(page 130)
K-Value:	109	(page 130)
Fisher-Angle:	2.7 deg.	(page 132)

Reference

```python
import matplotlib.pyplot as plt
import mplstereonet as mpl


decl = [122.5, 130.5, 132.5, 148.5, 140.0, 133.0, 157.5, 153.0, 140.0, 147.5,
        142.0, 163.5, 141.0, 156.0, 139.5, 153.5, 151.5, 147.5, 141.0, 143.5,
        131.5, 147.5, 147.0, 149.0, 144.0, 139.5]
incl = [55.5, 58.0, 44.0, 56.0, 63.0, 64.5, 53.0, 44.5, 61.5, 54.5, 51.0, 56.0,
        59.5, 56.5, 54.0, 47.5, 61.0, 58.5, 57.0, 67.5, 62.5, 63.5, 55.5, 62.0,
        53.5, 58.0]
confidence = 95

fig = plt.figure()
ax = fig.add_subplot(111, projection='stereonet')
ax.line(incl, decl, color="black", markersize=2)

vector, stats = mpl.find_fisher_stats(incl, decl, conf=confidence)

template = (u"Mean Vector P/B: {plunge:0.0f}\u00B0/{bearing:0.0f}\u00B0\n"
            "Confidence: {conf}%\n"
            u"Fisher Angle: {fisher:0.2f}\u00B0\n"
            u"R-Value {r:0.3f}\n"
            "K-Value: {k:0.2f}")

label = template.format(plunge=vector[0], bearing=vector[1], conf=confidence,
                        r=stats[0], fisher=stats[1], k=stats[2])

ax.line(vector[0], vector[1], color="red", label=label)
ax.cone(vector[0], vector[1], stats[1], facecolor="None", edgecolor="red")

ax.legend(bbox_to_anchor=(1.1, 1.1), numpoints=1)
plt.show()
```

Result



6.1.11 `fit_girdle_example.py`

Illustrates fitting a plane to a “gridle” distribution using `fit_girdle`.

This example simulates finding the plunge and bearing of a cylindrical fold axis from strike/dip measurements of bedding in the fold limbs.

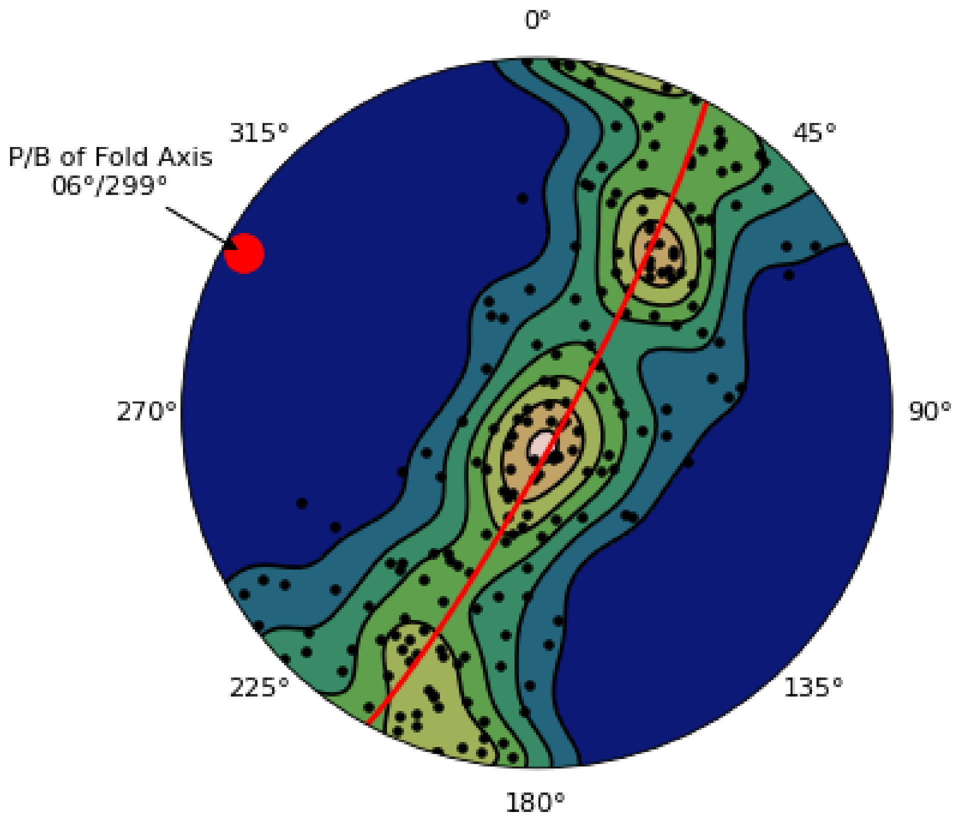
```python
import numpy as np
import matplotlib.pyplot as plt
import mplstereonet
np.random.seed(1)

# Generate a random girdle distribution from the plunge/bearing of a fold hinge
# In the end, we'll have strikes and dips as measured from bedding in the fold.
# *strike* and *dip* below would normally be your input.
num_points = 200
real_bearing, real_plunge = 300, 5
s, d = mplstereonet.plunge_bearing2pole(real_plunge, real_bearing)
lon, lat = mplstereonet.plane(s, d, segments=num_points)
lon += np.random.normal(0, np.radians(15), lon.shape)
lat += np.random.normal(0, np.radians(15), lat.shape)
strike, dip = mplstereonet.geographic2pole(lon, lat)
```

(continues on next page)

(continued from previous page)

```
# Plot the raw data and contour it:
fig, ax = mplstereonet.subplots()
ax.density_contourf(strike, dip, cmap='gist_earth')
ax.density_contour(strike, dip, colors='black')
ax.pole(strike, dip, marker='.', color='black')

# Fit a plane to the girdle of the distribution and display it.
fit_strike, fit_dip = mplstereonet.fit_girdle(strike, dip)
ax.plane(fit_strike, fit_dip, color='red', lw=2)
ax.pole(fit_strike, fit_dip, marker='o', color='red', markersize=14)

# Add some annotation of the result
lon, lat = mplstereonet.pole(fit_strike, fit_dip)
(plunge,), (bearing,) = mplstereonet.pole2plunge_bearing(fit_strike, fit_dip)
template = u'P/B of Fold Axis\n{:02.0f}\u00b0/{:03.0f}\u00b0'
ax.annotate(template.format(plunge, bearing), ha='center', va='bottom',
            xy=(lon, lat), xytext=(-50, 20), textcoords='offset points',
            arrowprops=dict(arrowstyle='-|>', facecolor='black'))

plt.show()
```

Result



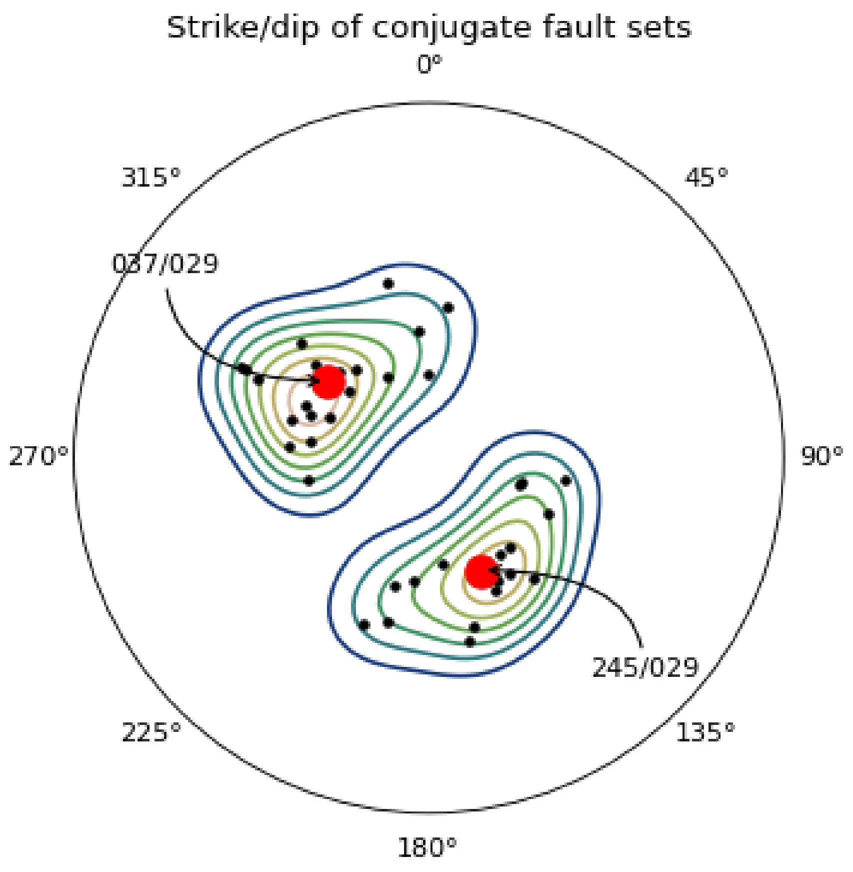
6.1.12 kmeans_example.py

Illustrates finding the average strike and dip of two conjugate sets of faults.

This uses a kmeans approach modified to work with bidirectional orientation measurements in 3D (`mplstereonet.kmeans`).

```
import matplotlib.pyplot as plt
import mplstereonet

import parse_angelier_data

# Load data from Angelier, 1979
strike, dip, rake = parse_angelier_data.load()

# Plot the raw data and contour it:
fig, ax = mplstereonet.subplots()
#ax.density_contourf(strike, dip, rake, measurement='rakes', cmap='gist_earth',
#                    sigma=1.5)
ax.density_contour(strike, dip, rake, measurement='rakes', cmap='gist_earth',
                    sigma=1.5)
ax.rake(strike, dip, rake, marker='.', color='black')

# Find the two modes
centers = mplstereonet.kmeans(strike, dip, rake, num=2, measurement='rakes')
strike_cent, dip_cent = mplstereonet.geographic2pole(*zip(*centers))
ax.pole(strike_cent, dip_cent, 'ro', ms=12)

# Label the modes
for (x0, y0) in centers:
    s, d = mplstereonet.geographic2pole(x0, y0)
    x, y = mplstereonet.pole(s, d) # Otherwise, we may get the antipode...

    if x > 0:
        kwargs = dict(xytext=(40, -40), ha='left')
    else:
        kwargs = dict(xytext=(-40, 40), ha='right')

    ax.annotate('{:03.0f}/{:03.0f}'.format(s[0], d[0]), xy=(x, y),
                xycoords='data', textcoords='offset points',
                arrowprops=dict(arrowstyle='->', connectionstyle='angle3'),
                **kwargs)

ax.set_title('Strike/dip of conjugate fault sets', y=1.07)

plt.show()
```

Result

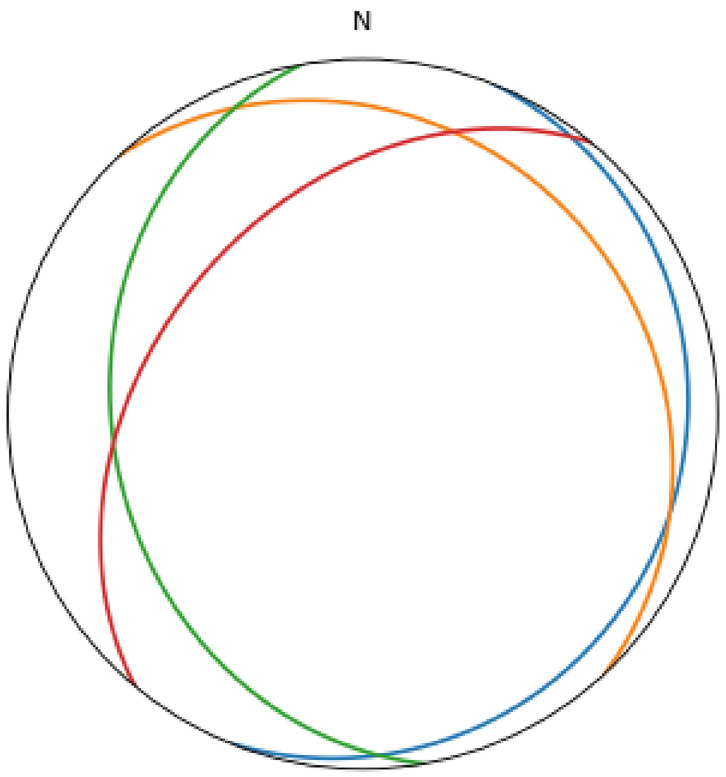


6.1.13 `multiple_planes.py`

plane, *rake*, *line*, etc all allow plotting of multiple measurements.

```
import matplotlib.pyplot as plt
import mplstereonet

# Make a figure with a single stereonet axes
fig, ax = mplstereonet.subplots()

# These follow the right hand rule to indicate dip direction
strikes = [22, 317, 170, 220]
dips = [10, 20, 30, 40]

# Plot the planes.
ax.plane(strikes, dips)

# Make only a single "N" azimuth tick label.
ax.set_azimuth_ticks([0], labels=['N'])

plt.show()
```

Result



6.1.14 `parse_angelier_data.py`

This is meant to serve as an example of slightly more complex parsing of orientation measurements.

Angelier, 1979's seminal paper on paleostress determination includes a table of slickenslide measurements from normal faults.

However, some of the measurements are rakes, while others are strike/dip and an azimuth of the slickenslides ("Rake" measurements without a direction letter are actually azimuthal measurements.).

Furthermore, the measurements do not follow the right-hand-rule for indicating dip direction of a plane and they indicate rake direction using a directional letter.

To unify the measurements for plotting, etc, we need to parse all of the measurements, and convert the azimuth measurements to rakes.

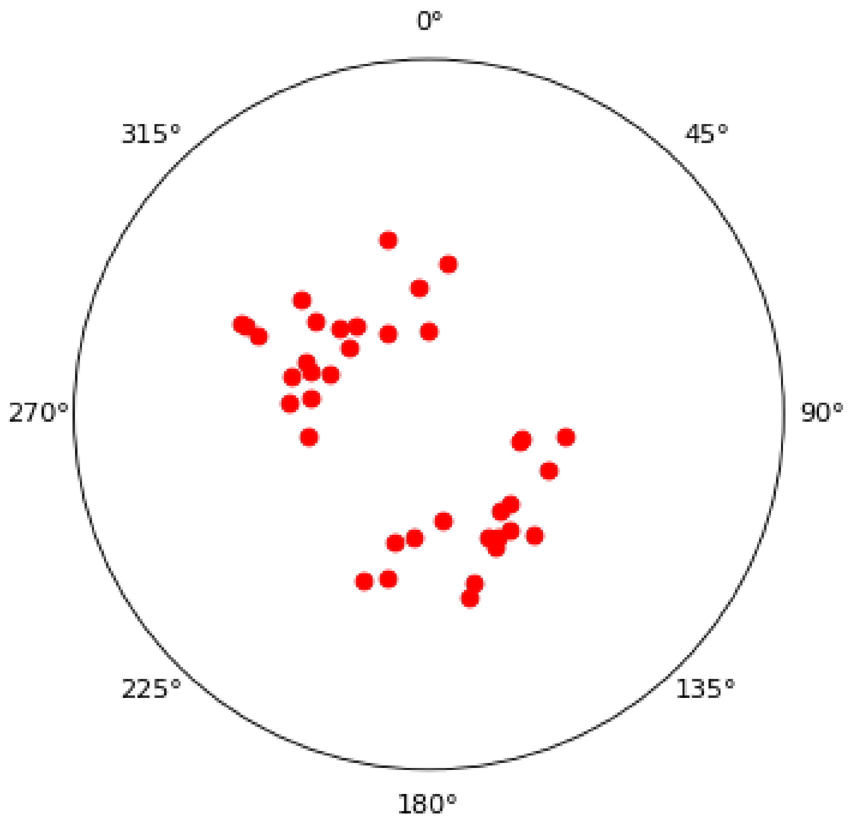
```python
import os
import matplotlib.pyplot as plt
import mplstereonet

def main():
    strike, dip, rake = load()
```

(continues on next page)

(continued from previous page)

```python
    # Plot the data.
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='stereonet')
    ax.rake(strike, dip, rake, 'ro')
    plt.show()

def load():
    """Read data from a text file on disk."""
    # Get the data file relative to this file's location...
    datadir = os.path.dirname(__file__)
    filename = os.path.join(datadir, 'angelier_data.txt')

    data = []
    with open(filename, 'r') as infile:
        for line in infile:
            # Skip comments
            if line.startswith('#'):
                continue

            # First column: strike, second: dip, third: rake.
            strike, dip, rake = line.strip().split()

            if rake[-1].isalpha():
                # If there's a directional letter on the rake column, parse it
                # normally.
                strike, dip, rake = mplstereonet.parse_rake(strike, dip, rake)
            else:
                # Otherwise, it's actually an azimuthal measurement of the
                # slickenslide directions, so we need to convert it to a rake.
                strike, dip = mplstereonet.parse_strike_dip(strike, dip)
                azimuth = float(rake)
                rake = mplstereonet.azimuth2rake(strike, dip, azimuth)

            data.append([strike, dip, rake])

    # Separate the columns back out
    strike, dip, rake = zip(*data)
    return strike, dip, rake

if __name__ == '__main__':
    main()
```

Result



6.1.15 `parsing_example.py`

Basic quadrant, strike/dip, and rake parsing.

mplstereonet expects measurements to follow the "right-hand-rule" (RHR) to indicate dip direction.

If you have a set of measurements that don't necessarily follow the RHR, there are a number of parsing and standardization functions in *mplstereonet* to correct for this.

```
import mplstereonet

print('Parse quadrant azimuth measurements')
for original in ['N30E', 'E30N', 'W10S', 'N 10 W']:
    azi = mplstereonet.parse_quadrant_measurement(original)
    print('"{}" --> {:.1f}'.format(original, azi))

print('\nParse quadrant strike/dip measurements.')
print('Note that the output follows the right-hand-rule.')

def parse_sd(original, seperator):
    strike, dip = mplstereonet.parse_strike_dip(*original.split(seperator))
    print('"{}" --> Strike: {:.1f}, Dip: {:.1f}'.format(original, strike, dip))
```

(continues on next page)

(continued from previous page)

```
parse_sd('215/10', '/')
parse_sd('215/10E', '/')
parse_sd('215/10NW', '/')
parse_sd('N30E/45NW', '/')
parse_sd('E10N\t20 N', '\t')
parse_sd('W30N/46.7 S', '/')

print("\nSimilarly, you can parse rake measurements that don't follow the RHR.")

def split_rake(original, sep1=None, sep2=None):
    components = original.split(sep1)
    if len(components) == 3:
        return components
    strike, rest = components
    dip, rake = rest.split(sep2)
    return strike, dip, rake

def display_rake(original, sep1, sep2=None):
    components = split_rake(original, sep1, sep2)
    strike, dip, rake = mplstereonet.parse_rake(*components)
    template = '"{}" --> Strike: {:.1f}, Dip: {:.1f}, Rake: {:.1f}'
    print(template.format(original, strike, dip, rake))

original = 'N30E/45NW 10NE'
display_rake(original, '/')

original = '210 45\t30N'
display_rake(original, None)

original = 'N30E/45NW raking 10SW'
display_rake(original, '/', 'raking')
```

Result

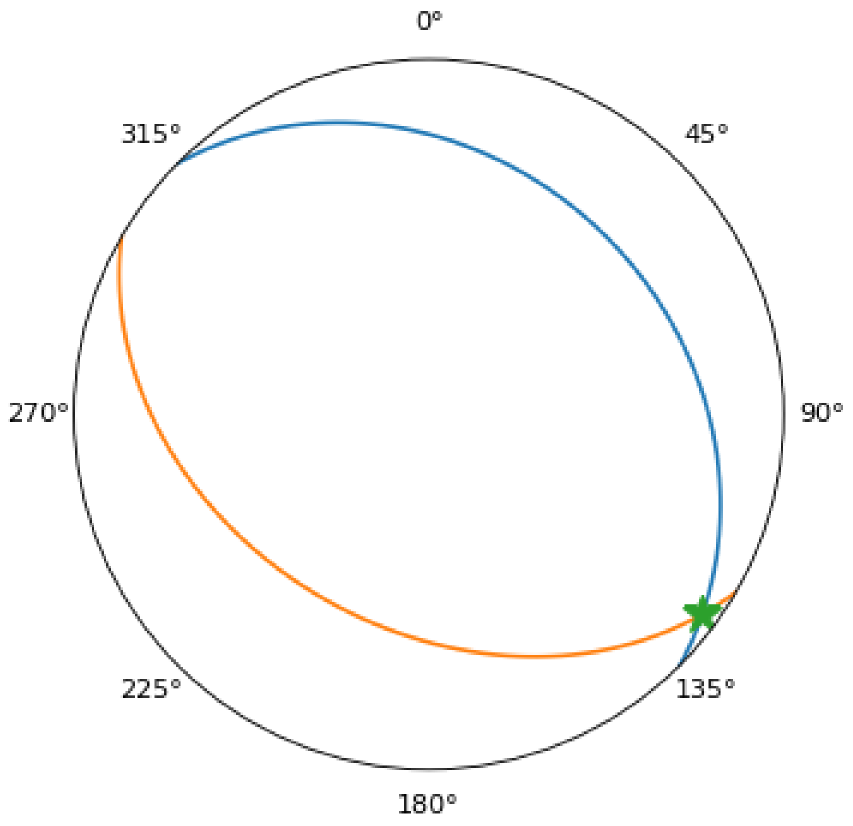
```
Parse quadrant azimuth measurements
"N30E" --> 30.0
"E30N" --> 60.0
"W10S" --> 260.0
"N 10 W" --> 350.0

Parse quadrant strike/dip measurements.
Note that the output follows the right-hand-rule.
"215/10" --> Strike: 215.0, Dip: 10.0
"215/10E" --> Strike: 35.0, Dip: 10.0
"215/10NW" --> Strike: 215.0, Dip: 10.0
"N30E/45NW" --> Strike: 210.0, Dip: 45.0
"E10N       20 N" --> Strike: 260.0, Dip: 20.0
"W30N/46.7 S" --> Strike: 120.0, Dip: 46.7

Similarly, you can parse rake measurements that don't follow the RHR.
"N30E/45NW 10NE" --> Strike: 210.0, Dip: 45.0, Rake: 170.0
"210 45     30N" --> Strike: 210.0, Dip: 45.0, Rake: 150.0
"N30E/45NW raking 10SW" --> Strike: 210.0, Dip: 45.0, Rake: 10.0
```

6.1.16 `plane_intersection.py`

Find the intersection of two planes and plot it.

```python
import matplotlib.pyplot as plt
import mplstereonet

strike1, dip1 = 315, 30
strike2, dip2 = 120, 40

fig, ax = mplstereonet.subplots()

# Plot the two planes...
ax.plane(strike1, dip1)
ax.plane(strike2, dip2)

# Find the intersection of the two as a plunge/bearing
plunge, bearing = mplstereonet.plane_intersection(strike1, dip1, strike2, dip2)

# Plot the plunge/bearing
ax.line(plunge, bearing, marker='*', markersize=15)

plt.show()
```

Result



6.1.17 `polar_overlay.py`

Demonstrates adding both polar and arbitrary grid overlays on a stereonet. Changing the grid overlay does not change the representation of the data. Notice that the plane, pole, and rake are all displayed identically in each case. Only the grid lines change.

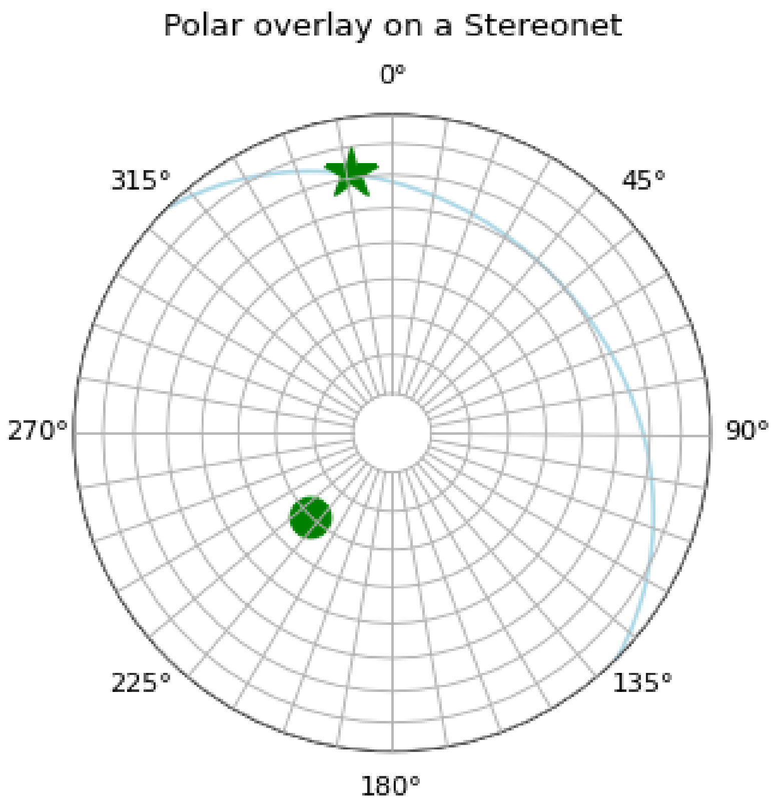
```python
import matplotlib.pyplot as plt
import mplstereonet

def main():
    # Display the data with a polar grid
    ax1 = basic()
    ax1.grid(kind='polar')
    ax1.set_title('Polar overlay on a Stereonet', y=1.1)

    # Display the data with a grid centered on the pole to the plotted plane.
    ax2 = basic()
    ax2.grid(center=mplstereonet.pole(315, 30))
    ax2.set_title('Arbitrary overlay on a Stereonet', y=1.1)

    plt.show()
```

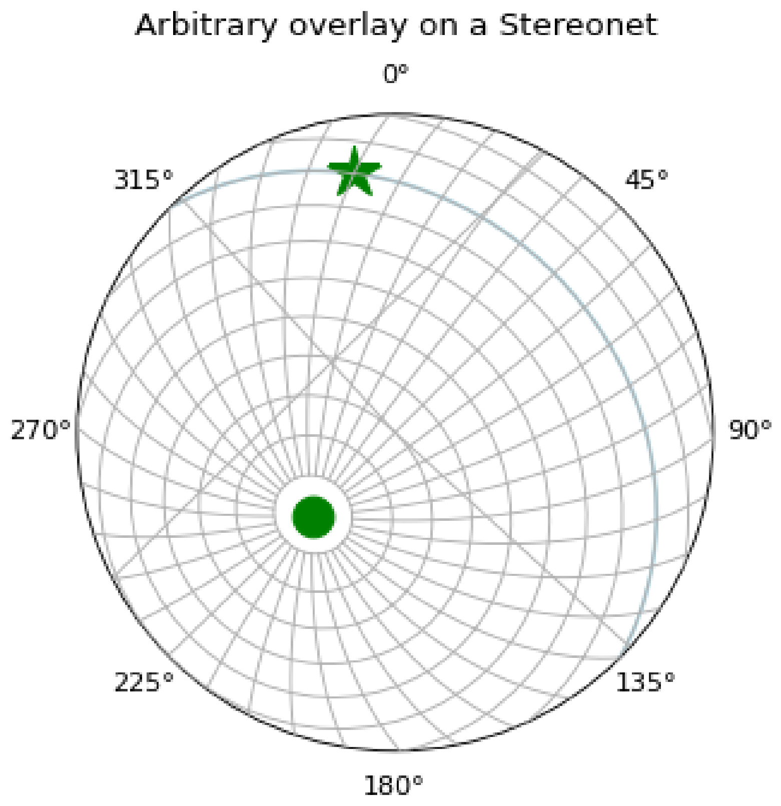
(continues on next page)

(continued from previous page)

```
def basic():
    """Set up a basic stereonet and plot the same data each time."""
    fig, ax = mplstereonet.subplots()

    strike, dip = 315, 30
    ax.plane(strike, dip, color='lightblue')
    ax.pole(strike, dip, color='green', markersize=15)
    ax.rake(strike, dip, 40, marker='*', markersize=20, color='green')

    # Make a bit of room for the title...
    fig.subplots_adjust(top=0.8)

    return ax

if __name__ == '__main__':
    main()
```

Result





6.1.18 `rotation_example.py`

As an exmaple of basic functionality, let’s plot a plane, the pole to the plane, and a rake along the plane.

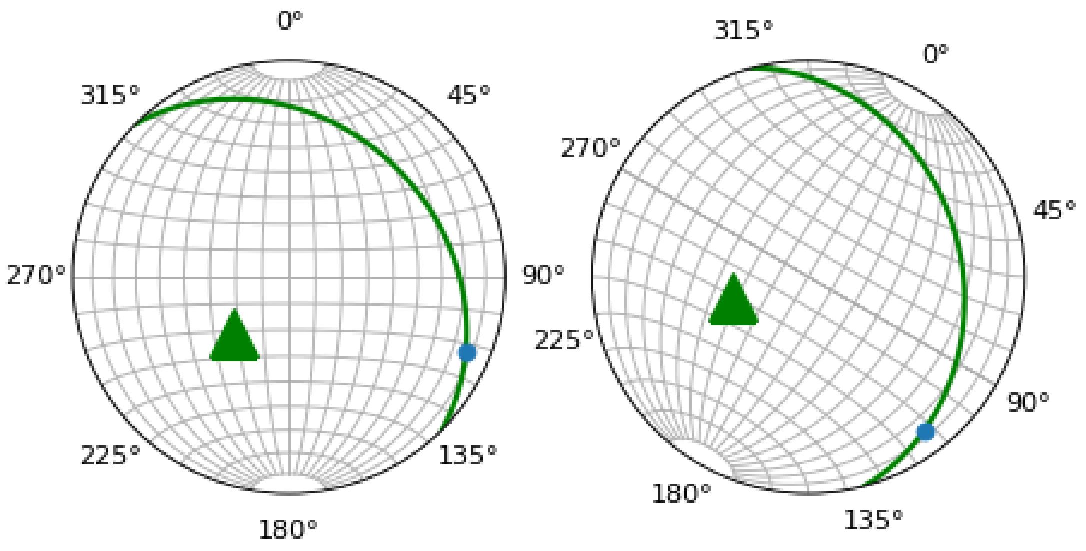
```
import matplotlib.pyplot as plt
import mplstereonet

fig = plt.figure()

# An un-rotated axes
ax1 = fig.add_subplot(121, projection='stereonet')

# Rotated 30 degrees clockwise from North
ax2 = fig.add_subplot(122, projection='stereonet', rotation=30)

# Measurements follow the right-hand-rule to indicate dip direction
strike, dip = 315, 30

# Plot the same data on both axes
for ax in [ax1, ax2]:
    ax.plane(strike, dip, 'g-', linewidth=2)
    ax.pole(strike, dip, 'g^', markersize=18)
    ax.rake(strike, dip, -25)

```

(continues on next page)

(continued from previous page)

```
    ax.grid()

plt.show()
```

Result



6.1.19 `scatter.py`

Example of how *ax.scatter* can be used to plot linear data on a stereonet varying color and/or size by other variables.

This also serves as a general example of how to convert orientation data into the coordinate system that the stereonet plot uses so that generic matplotlib plotting methods may be used.

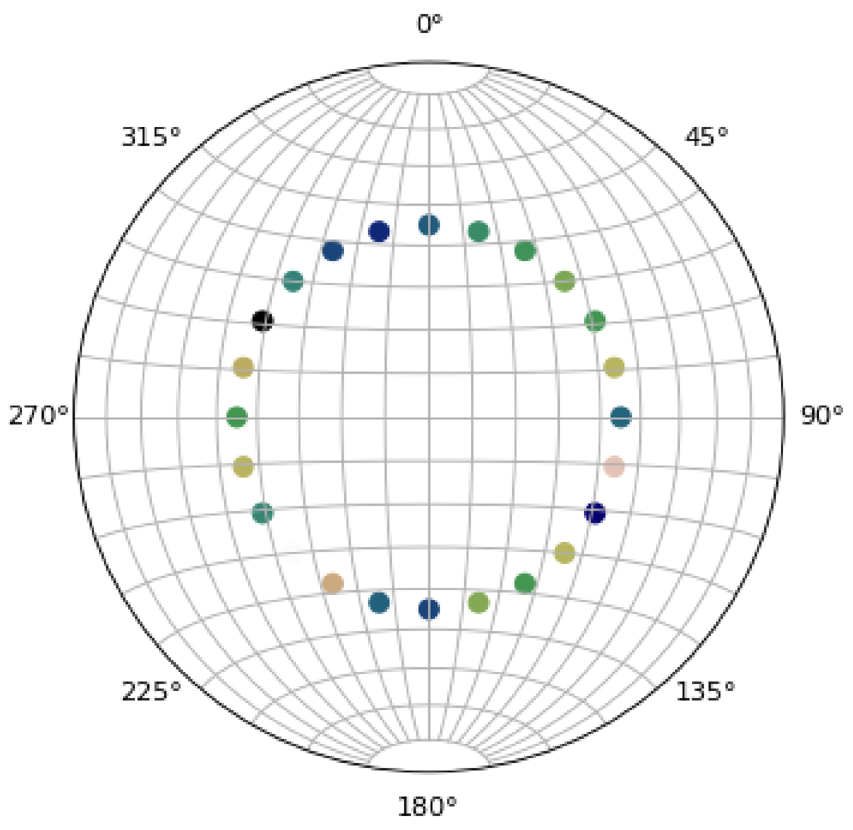
```
import numpy as np
import matplotlib.pyplot as plt
import mplstereonet
np.random.seed(1)

strikes = np.arange(0, 360, 15)
dips = 45 * np.ones(strikes.size)
magnitude = np.random.random(strikes.size)
```

(continues on next page)

(continued from previous page)

```

# Convert our strikes and dips to stereonet coordinates
lons, lats = mplstereonet.pole(strikes, dips)

# Now we'll plot our data and color by magnitude
fig, ax = mplstereonet.subplots()
sm = ax.scatter(lons, lats, c=magnitude, s=50, cmap='gist_earth')

ax.grid()
plt.show()
```

Result



6.1.20 `stereonet_explanation.py`

This figure illustrates the difference between the "internal" coordinate system of longitude and latitude that plotting actually takes place in (e.g. if you were to use *ax.plot* or any other "raw" matplotlib command) and the conceptual coordinate system that a lower-hemisphere stereonet represents.

```
import matplotlib.pyplot as plt
import numpy as np
```

(continues on next page)

(continued from previous page)

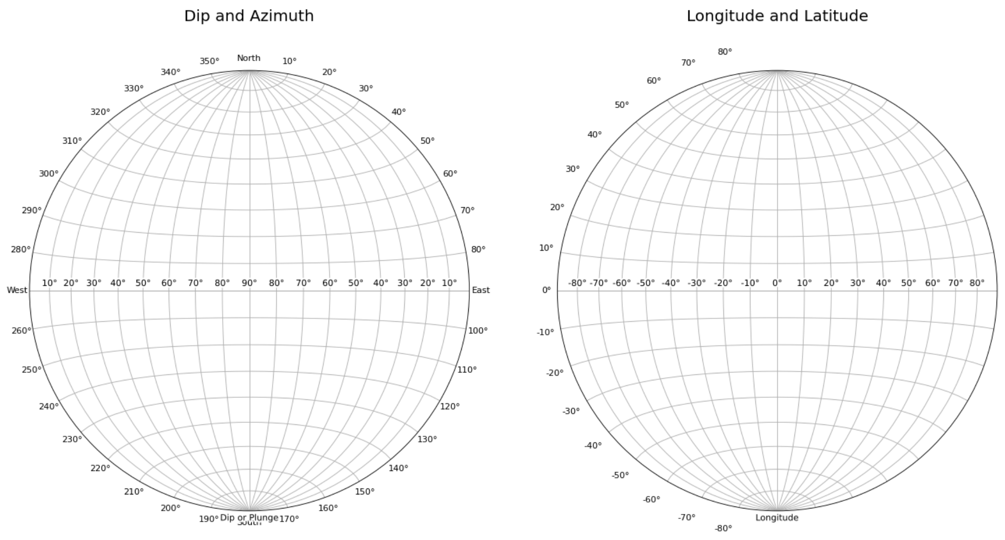
```python
import mplstereonet

def main():
    fig, (ax1, ax2) = setup_figure()
    stereonet_projection_explanation(ax1)
    native_projection_explanation(ax2)
    plt.show()

def setup_figure():
    """Setup the figure and axes"""
    fig, axes = mplstereonet.subplots(ncols=2, figsize=(20,10))
    for ax in axes:
        # Make the grid lines solid.
        ax.grid(ls='-')
        # Make the longitude grids continue all the way to the poles
        ax.set_longitude_grid_ends(90)
    return fig, axes

def stereonet_projection_explanation(ax):
    """Example to explain azimuth and dip on a lower-hemisphere stereonet."""
    ax.set_title('Dip and Azimuth', y=1.1, size=18)

    # Set the azimuth ticks to be just "N", "E", etc.
    ax.set_azimuth_ticks(range(0, 360, 10))

    # Hackishly set some of the azimuth labels to North, East, etc...
    fmt = ax.yaxis.get_major_formatter()
    labels = [fmt(item) for item in ax.get_azimuth_ticks()]
    labels[0] = 'North'
    labels[9] = 'East'
    labels[18] = 'South'
    labels[27] = 'West'
    ax.set_azimuth_ticklabels(labels)

    # Unhide the xticklabels and use them for dip labels
    ax.xaxis.set_tick_params(label1On=True)
    labels = list(range(10, 100, 10)) + list(range(80, 0, -10))
    ax.set_xticks(np.radians(np.arange(-80, 90, 10)))
    ax.set_xticklabels([fmt(np.radians(item)) for item in labels])

    ax.set_xlabel('Dip or Plunge')

    xlabel_halo(ax)
    return ax

def native_projection_explanation(ax):
    """Example showing how the "native" longitude and latitude relate to the
    stereonet projection."""
    ax.set_title('Longitude and Latitude', size=18, y=1.1)

    # Hide the azimuth labels
    ax.set_azimuth_ticklabels([])

    # Make the axis tick labels visible:
    ax.set_xticks(np.radians(np.arange(-80, 90, 10)))
    ax.tick_params(label1On=True)
```

(continues on next page)

(continued from previous page)

```
    ax.set_xlabel('Longitude')

    xlabel_halo(ax)
    return ax

def xlabel_halo(ax):
    """Add a white "halo" around the xlabels."""
    import matplotlib.patheffects as effects
    for tick in ax.get_xticklabels() + [ax.xaxis.label]:
        tick.set_path_effects([effects.withStroke(linewidth=4, foreground='w')])

if __name__ == '__main__':
    main()
```

Result



6.1.21 `two_point.py`

Demonstrates plotting multiple linear features with a single `ax.pole` call.

The real purpose of this example is to serve as an implicit regression test for some oddities in the way axes grid lines are handled in matplotlib and mplstereonet. A 2-vertex line can sometimes be confused for an axes grid line, and they need different handling on a stereonet.

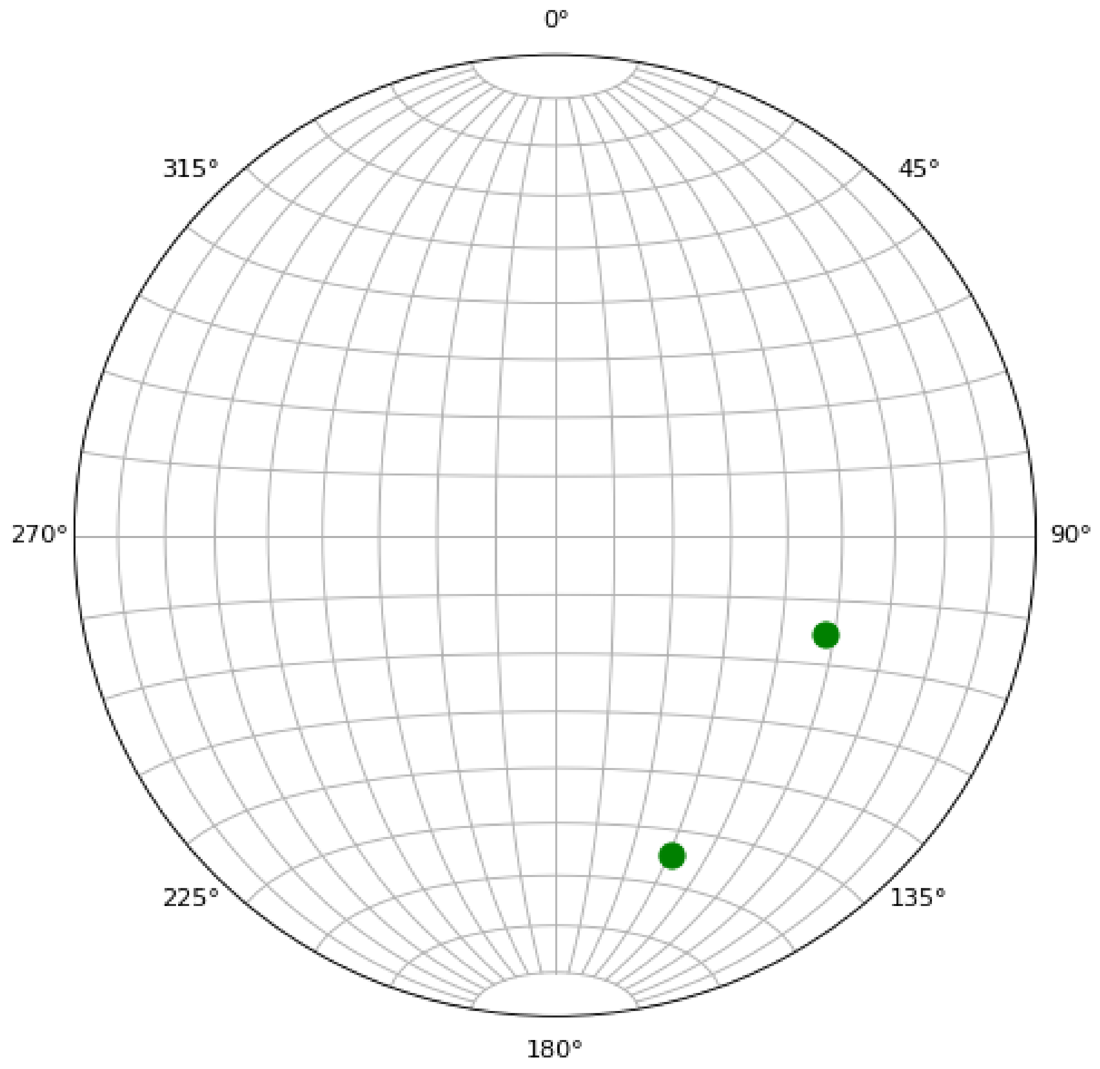
```
import matplotlib.pyplot as plt
import mplstereonet

fig, ax = mplstereonet.subplots(figsize=(7,7))
strike = [200, 250]
dip = [50, 60]
ax.pole(strike, dip, 'go', markersize=10)
```

(continues on next page)

(continued from previous page)

```
ax.grid()
plt.show()
```

Result



CHAPTER 7

Detailed Documentation

7.1 mplstereonet Package

7.1.1 `mplstereonet` Package

class `mplstereonet.`**`StereonetAxes`**(**args*, ***kwargs*)

Bases: `matplotlib.projections.geo.LambertAxes`

An axes representing a lower-hemisphere "schmitt" (a.k.a. equal area) projection.

`__init__`(*self*, **args*, ***kwargs*)

Initialization is identical to a normal Axes object except for the following kwarg:

Parameters

rotation [number] The rotation of the stereonet in degrees clockwise from North.

center_latitude [number] The center latitude of the stereonet in degrees.

center_longitude [number] The center longitude of the stereonet in degrees.

All additional args and kwargs are identical to Axes.__init__

`cla`(*self*)

Clear the current axes.

`cone`(*self*, *plunge*, *bearing*, *angle*, *segments=100*, *bidirectional=True*, ***kwargs*)

Plot a polygon of a small circle (a.k.a. a cone) with an angular radius of *angle* centered at a p/b of *plunge*, *bearing*. Additional keyword arguments are passed on to the `PathCollection`. (e.g. to have an unfilled small small circle, pass "facecolor='none'".)

Parameters

plunge [number or sequence of numbers] The plunge of the center of the cone in degrees.

bearing [number or sequence of numbers] The bearing of the center of the cone in degrees.

angle [number or sequence of numbers] The angular radius of the cone in degrees.

segments [int, optional] The number of vertices to use for the cone. Defaults to 100.

bidirectional [boolean, optional] Whether or not to draw two patches (the one given and its antipode) for each measurement. Defaults to True.

****kwargs** Additional parameters are `matplotlib.collections.PatchCollection` properties.

Returns

collection [`matplotlib.collections.PathCollection`]

Notes

If *bidirectional* is `True`, two circles will be plotted, even if only one of each pair is visible. This is the default behavior.

density_contour(*self*, **args*, ***kwargs*)

Estimates point density of the given linear orientation measurements (Interpreted as poles, lines, rakes, or "raw" longitudes and latitudes based on the *measurement* keyword argument.) and plots contour lines of the resulting density distribution.

Parameters

***args** [A variable number of sequences of measurements.] By default, this will be expected to be `strike` & `dip`, both array-like sequences representing poles to planes. (Rake measurements require three parameters, thus the variable number of arguments.) The `measurement` kwarg controls how these arguments are interpreted.

measurement [string, optional] Controls how the input arguments are interpreted. Defaults to `"poles"`. May be one of the following:

"poles" [strikes, dips] Arguments are assumed to be sequences of strikes and dips of planes. Poles to these planes are used for contouring.

"lines" [plunges, bearings] Arguments are assumed to be sequences of plunges and bearings of linear features.

"rakes" [strikes, dips, rakes] Arguments are assumed to be sequences of strikes, dips, and rakes along the plane.

"radians" [lon, lat] Arguments are assumed to be "raw" longitudes and latitudes in the stereonet's underlying coordinate system.

method [string, optional] The method of density estimation to use. Defaults to `"exponential_kamb"`. May be one of the following:

"exponential_kamb" [Kamb with exponential smoothing] A modified Kamb method using exponential smoothing [1]. Units are in numbers of standard deviations by which the density estimate differs from uniform.

"linear_kamb" [Kamb with linear smoothing] A modified Kamb method using linear smoothing [1]. Units are in numbers of standard deviations by which the density estimate differs from uniform.

"kamb" [Kamb with no smoothing] Kamb's method [2] with no smoothing. Units are in numbers of standard deviations by which the density estimate differs from uniform.

"schmidt" [1% counts] The traditional "Schmidt" (a.k.a. 1%) method. Counts points within a counting circle comprising 1% of the total area of the hemisphere. Does not take into account sample size. Units are in points per 1% area.

sigma [int or float, optional] The number of standard deviations defining the expected number of standard deviations by which a random sample from a uniform distribution of points would be expected to vary from being evenly distributed across the hemisphere. This controls the size of the counting circle, and therefore the degree of smoothing. Higher sigmas will lead to more smoothing of the resulting density distribution. This parameter only applies to Kamb-based methods. Defaults to 3.

gridsize [int or 2-item tuple of ints, optional] The size of the grid that the density is estimated on. If a single int is given, it is interpreted as an NxN grid. If a tuple of ints is given it is interpreted as (nrows, ncols). Defaults to 100.

weights [array-like, optional] The relative weight to be applied to each input measurement. The array will be normalized to sum to 1, so absolute value of the weights do not affect the result. Defaults to None.

****kwargs** Additional keyword arguments are passed on to matplotlib's *contour* function.

Returns

A matplotlib ContourSet.

See also:

mplstereonet.density_grid

mplstereonet.StereonetAxes.density_contourf

matplotlib.pyplot.contour

matplotlib.pyplot.clabel

References

[1], [2]

Examples

Plot density contours of poles to the specified planes using a modified Kamb method with exponential smoothing [1].

```
>>> strikes, dips = [120, 315, 86], [22, 85, 31]
>>> ax.density_contour(strikes, dips)
```

Plot density contours of a set of linear orientation measurements.

```
>>> plunges, bearings = [-10, 20, -30], [120, 315, 86]
>>> ax.density_contour(plunges, bearings, measurement='lines')
```

Plot density contours of a set of rake measurements.

```
>>> strikes, dips, rakes = [120, 315, 86], [22, 85, 31], [-5, 20, 9]
>>> ax.density_contour(strikes, dips, rakes, measurement='rakes')
```

Plot density contours of a set of "raw" longitudes and latitudes.

```
>>> lon, lat = np.radians([-40, 30, -85]), np.radians([21, -59, 45])
>>> ax.density_contour(lon, lat, measurement='radians')
```

Plot density contours of poles to planes using a Kamb method [2] with the density estimated on a 10x10 grid (in long-lat space)

```
>>> strikes, dips = [120, 315, 86], [22, 85, 31]
>>> ax.density_contour(strikes, dips, method='kamb', gridsize=10)
```

Plot density contours of poles to planes with contours at [1,2,3] standard deviations.

```
>>> strikes, dips = [120, 315, 86], [22, 85, 31]
>>> ax.density_contour(strikes, dips, levels=[1,2,3])
```

density_contourf (*self*, **args*, ***kwargs*)

Estimates point density of the given linear orientation measurements (Interpreted as poles, lines, rakes, or "raw" longitudes and latitudes based on the *measurement* keyword argument.) and plots filled contours of the resulting density distribution.

Parameters

***args** [A variable number of sequences of measurements.] By default, this will be expected to be `strike` & `dip`, both array-like sequences representing poles to planes. (Rake measurements require three parameters, thus the variable number of arguments.) The `measurement` kwarg controls how these arguments are interpreted.

measurement [string, optional] Controls how the input arguments are interpreted. Defaults to `"poles"`. May be one of the following:

"poles" [strikes, dips] Arguments are assumed to be sequences of strikes and dips of planes. Poles to these planes are used for contouring.

"lines" [plunges, bearings] Arguments are assumed to be sequences of plunges and bearings of linear features.

"rakes" [strikes, dips, rakes] Arguments are assumed to be sequences of strikes, dips, and rakes along the plane.

"radians" [lon, lat] Arguments are assumed to be "raw" longitudes and latitudes in the stereonet's underlying coordinate system.

method [string, optional] The method of density estimation to use. Defaults to `"exponential_kamb"`. May be one of the following:

"exponential_kamb" [Kamb with exponential smoothing] A modified Kamb method using exponential smoothing [1]. Units are in numbers of standard deviations by which the density estimate differs from uniform.

"linear_kamb" [Kamb with linear smoothing] A modified Kamb method using linear smoothing [1]. Units are in numbers of standard deviations by which the density estimate differs from uniform.

"kamb" [Kamb with no smoothing] Kamb's method [2] with no smoothing. Units are in numbers of standard deviations by which the density estimate differs from uniform.

"schmidt" [1% counts] The traditional "Schmidt" (a.k.a. 1%) method. Counts points within a counting circle comprising 1% of the total area of the hemisphere. Does not take into account sample size. Units are in points per 1% area.

sigma [int or float, optional] The number of standard deviations defining the expected number of standard deviations by which a random sample from a uniform distribution of points would be expected to vary from being evenly distributed across the hemisphere. This controls the size of the counting circle, and therefore the degree of smoothing. Higher sigmas will lead to more smoothing of the resulting density distribution. This parameter only applies to Kamb-based methods. Defaults to 3.

gridsize [int or 2-item tuple of ints, optional] The size of the grid that the density is estimated on. If a single int is given, it is interpreted as an NxN grid. If a tuple of ints is given it is interpreted as (nrows, ncols). Defaults to 100.

weights [array-like, optional] The relative weight to be applied to each input measurement. The array will be normalized to sum to 1, so absolute value of the weights do not affect the result. Defaults to None.

****kwargs** Additional keyword arguments are passed on to matplotlib's *contourf* function.

Returns

A matplotlib *QuadContourSet*.

See also:

mplstereonet.density_grid

mplstereonet.StereonetAxes.density_contour

matplotlib.pyplot.contourf

matplotlib.pyplot.clabel

References

[1], [2]

Examples

Plot filled density contours of poles to the specified planes using a modified Kamb method with exponential smoothing [1].

```
>>> strikes, dips = [120, 315, 86], [22, 85, 31]
>>> ax.density_contourf(strikes, dips)
```

Plot filled density contours of a set of linear orientation measurements.

```
>>> plunges, bearings = [-10, 20, -30], [120, 315, 86]
>>> ax.density_contourf(plunges, bearings, measurement='lines')
```

Plot filled density contours of a set of rake measurements.

```
>>> strikes, dips, rakes = [120, 315, 86], [22, 85, 31], [-5, 20, 9]
>>> ax.density_contourf(strikes, dips, rakes, measurement='rakes')
```

Plot filled density contours of a set of "raw" longitudes and latitudes.

```
>>> lon, lat = np.radians([-40, 30, -85]), np.radians([21, -59, 45])
>>> ax.density_contourf(lon, lat, measurement='radians')
```

Plot filled density contours of poles to planes using a Kamb method [2] with the density estimated on a 10x10 grid (in long-lat space)

```
>>> strikes, dips = [120, 315, 86], [22, 85, 31]
>>> ax.density_contourf(strikes, dips, method='kamb', gridsize=10)
```

Plot filled density contours of poles to planes with contours at [1,2,3] standard deviations.

```
>>> strikes, dips = [120, 315, 86], [22, 85, 31]
>>> ax.density_contourf(strikes, dips, levels=[1,2,3])
```

format_coord(*self*, *x*, *y*)
Format displayed coordinates during mouseover of axes.

get_azimuth_ticklabels(*self*, *minor=False*)
Get the azimuth tick labels as a list of Text artists.

get_rotation(*self*)
The rotation of the stereonet in degrees clockwise from North.

grid(*self*, *b=None*, *which='major'*, *axis='both'*, *kind='arbitrary'*, *center=None*, ***kwargs*)

Usage is identical to a normal axes grid except for the `kind` and `center` kwargs. `kind="polar"` will add a polar overlay.

The `center` and `kind` arguments allow you to add a grid from a differently-centered stereonet. This is useful for making "polar stereonets" that still use the same coordinate system as a standard stereonet. (i.e. a plane/line/whatever will have the same representation on both, but the grid is displayed differently.)

To display a polar grid on a stereonet, use `kind="polar"`.

It is also often useful to display a grid relative to an arbitrary measurement (e.g. a lineation axis). In that case, use the `lon_center` and `lat_center` arguments. Note that these are in radians in "stereonet coordinates". Therefore, you'll often want to use one of the functions in `stereonet_math` to convert a line/plane/rake into the longitude and latitude you'd input here. For example: `add_overlay(center=stereonet_math.line(plunge, bearing))`.

If no parameters are specified, this is equivalent to turning on the standard grid. Configure the grid lines.

Parameters

b [bool or None, optional] Whether to show the grid lines. If any *kwargs* are supplied, it is assumed you want the grid on and *b* will be set to True.

If *b* is *None* and there are no *kwargs*, this toggles the visibility of the lines.

which [{'major', 'minor', 'both'}, optional] The grid lines to apply the changes on.

axis [{'both', 'x', 'y'}, optional] The axis to apply the changes on.

****kwargs** [*.Line2D* properties] Define the line properties of the grid, e.g.:

```
grid(color='r', linestyle='-', linewidth=2)
```

Valid keyword arguments are:

Properties: agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array alpha: float or None animated: bool antialiased or aa: bool clip_box: *.Bbox* clip_on: bool clip_path: Patch or (Path, Transform) or None color or c: color contains: callable dash_capstyle: {'butt', 'round', 'projecting'} dash_joinstyle: {'miter', 'round', 'bevel'} dashes: sequence of floats (on/off ink in points) or (None, None) data: (2, N) array or two 1D arrays drawstyle or ds: {'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}, default: 'default' figure: *.Figure* fillstyle: {'full', 'left', 'right', 'bottom', 'top', 'none'} gid: str in_layout: bool label: object linestyle or ls: {'-', '–', '-.', ':', '', (offset, on-off-seq), …} linewidth or lw: float marker: marker style markeredgecolor or mec: color markeredgewidth or mew: float markerfacecolor or mfc: color markerfacecoloralt or mfcalt: color markersize or ms: float markevery: None or int or (int, int) or slice or List[int] or float or (float, float) path_effects: *.AbstractPathEffect* picker: float or callable[[Artist, Event], Tuple[bool, dict]] pickradius: float rasterized: bool or None sketch_params: (scale: float, length: float, randomness: float) snap: bool or None solid_capstyle: {'butt', 'round', 'projecting'} solid_joinstyle: {'miter', 'round', 'bevel'} transform: *matplotlib.transforms.Transform* url: str visible: bool xdata: 1D array ydata: 1D array zorder: float

Notes

The axis is drawn as a unit, so the effective zorder for drawing the grid is determined by the zorder of each axis, not by the zorder of the *.Line2D* objects comprising the grid. Therefore, to set grid zorder, use *.set_axisbelow* or, for more control, call the *~matplotlib.axis.Axis.set_zorder* method of each axis.

line (*self*, *plunge*, *bearing*, **args*, ***kwargs*)

Plot points representing linear features on the axes. Additional arguments and keyword arguments are passed on to *plot*.

Parameters

plunge, bearing [number or sequence of numbers] The plunge and bearing of the line(s) in degrees. The plunge is measured in degrees downward from the end of the feature specified by the bearing.

****kwargs** Additional parameters are passed on to *plot*.

Returns

A sequence of Line2D artists representing the point(s) specified by

***strike* and *dip*.**

plane (*self*, *strike*, *dip*, **args*, ***kwargs*)

Plot lines representing planes on the axes. Additional arguments and keyword arguments are passed on to *ax.plot*.

Parameters

strike, dip [number or sequences of numbers] The strike and dip of the plane(s) in degrees. The dip direction is defined by the strike following the "right-hand rule".

segments [int, optional] The number of vertices to use for the line. Defaults to 100.

****kwargs** Additional parameters are passed on to *plot*.

Returns

A sequence of Line2D artists representing the lines specified by

***strike* and *dip*.**

pole (*self*, *strike*, *dip*, **args*, ***kwargs*)

Plot points representing poles to planes on the axes. Additional arguments and keyword arguments are passed on to *ax.plot*.

Parameters

strike, dip [numbers or sequences of numbers] The strike and dip of the plane(s) in degrees. The dip direction is defined by the strike following the "right-hand rule".

****kwargs** Additional parameters are passed on to *plot*.

Returns

A sequence of Line2D artists representing the point(s) specified by

***strike* and *dip*.**

`rake` (*self*, *strike*, *dip*, *rake_angle*, **args*, ***kwargs*)
Plot points representing lineations along planes on the axes. Additional arguments and keyword arguments are passed on to *plot*.

Parameters

strike, dip [number or sequences of numbers] The strike and dip of the plane(s) in degrees. The dip direction is defined by the strike following the "right-hand rule".

rake_angle [number or sequences of numbers] The angle of the lineation(s) on the plane(s) measured in degrees downward from horizontal. Zero degrees corresponds to the "right hand" direction indicated by the strike, while negative angles are measured downward from the opposite strike direction.

****kwargs** Additional arguments are passed on to *plot*.

Returns

A sequence of Line2D artists representing the point(s) specified by

***strike* and *dip*.**

`rotation`
The rotation of the stereonet in degrees clockwise from North.

`set_azimuth_ticklabels` (*self*, *labels*, *fontdict=None*, ***kwargs*)
Sets the labels for the azimuthal ticks.

Parameters

labels [A sequence of strings] Azimuth tick labels

****kwargs** Additional parameters are text properties for the labels.

`set_azimuth_ticks` (*self*, *angles*, *labels=None*, *frac=None*, ***kwargs*)
Sets the azimuthal tick locations (Note: tick lines are not currently drawn or supported.).

Parameters

angles [sequence of numbers] The tick locations in degrees.

labels [sequence of strings] The tick label at each location. Defaults to a formatted version of the specified angles.

frac [number] The radial location of the tick labels. 1.0 is the along the edge, 1.1 would be outside, and 0.9 would be inside.

****kwargs** Additional parameters are text properties for the labels.

`set_longitude_grid` (*self*, *degrees*)
Set the number of degrees between each longitude grid.

`set_longitude_grid_ends` (*self*, *value*)
Set the latitude(s) at which to stop drawing the longitude grids.

set_position (*self*, *pos*, *which='both'*)

Set the axes position.

Axes have two position attributes. The 'original' position is the position allocated for the Axes. The 'active' position is the position the Axes is actually drawn at. These positions are usually the same unless a fixed aspect is set to the Axes. See *.set_aspect* for details.

Parameters

pos [[left, bottom, width, height] or *~matplotlib.transforms.Bbox*] The new position of the in *.Figure* coordinates.

which [{'both', 'active', 'original'}, optional] Determines which position variables to change.

set_rotation (*self*, *rotation*)

Set the rotation of the stereonet in degrees clockwise from North.

mplstereonet.**pole** (*strike*, *dip*)

Calculates the longitude and latitude of the pole(s) to the plane(s) specified by *strike* and *dip*, given in degrees.

Parameters

strike [number or sequence of numbers] The strike of the plane(s) in degrees, with dip direction indicated by the azimuth (e.g. 315 vs. 135) specified following the "right hand rule".

dip [number or sequence of numbers] The dip of the plane(s) in degrees.

Returns

lon, lat [Arrays of longitude and latitude in radians.]

mplstereonet.**plane** (*strike*, *dip*, *segments=100*, *center=(0, 0)*)

Calculates the longitude and latitude of *segments* points along the stereonet projection of each plane with a given *strike* and *dip* in degrees. Returns points for one hemisphere only.

Parameters

strike [number or sequence of numbers] The strike of the plane(s) in degrees, with dip direction indicated by the azimuth (e.g. 315 vs. 135) specified following the "right hand rule".

dip [number or sequence of numbers] The dip of the plane(s) in degrees.

segments [number or sequence of numbers] The number of points in the returned *lon* and *lat* arrays. Defaults to 100 segments.

center [sequence of two numbers (lon, lat)] The longitude and latitude of the center of the hemisphere that the returned points will be in. Defaults to 0,0 (approriate for a typical stereonet).

Returns

lon, lat [arrays] *num_segments* x *num_strikes* arrays of longitude and latitude in radians.

mplstereonet.**line** (*plunge*, *bearing*)

Calculates the longitude and latitude of the linear feature(s) specified by *plunge* and *bearing*.

Parameters

plunge [number or sequence of numbers] The plunge of the line(s) in degrees. The plunge is measured in degrees downward from the end of the feature specified by the bearing.

bearing [number or sequence of numbers] The bearing (azimuth) of the line(s) in degrees.

Returns

lon, lat [Arrays of longitude and latitude in radians.]

`mplstereonet.`**`rake`**(*strike*, *dip*, *rake_angle*)

Calculates the longitude and latitude of the linear feature(s) specified by *strike*, *dip*, and *rake_angle*.

Parameters

strike [number or sequence of numbers] The strike of the plane(s) in degrees, with dip direction indicated by the azimuth (e.g. 315 vs. 135) specified following the "right hand rule".

dip [number or sequence of numbers] The dip of the plane(s) in degrees.

rake_angle [number or sequence of numbers] The angle of the lineation on the plane measured in degrees downward from horizontal. Zero degrees corresponds to the "right-hand" direction indicated by the strike, while 180 degrees or a negative angle corresponds to the opposite direction.

Returns

lon, lat [Arrays of longitude and latitude in radians.]

`mplstereonet.`**`plunge_bearing2pole`**(*plunge*, *bearing*)

Converts the given *plunge* and *bearing* in degrees to a strike and dip of the plane whose pole would be parallel to the line specified. (i.e. The pole to the plane returned would plot at the same point as the specified plunge and bearing.)

Parameters

plunge [number or sequence of numbers] The plunge of the line(s) in degrees. The plunge is measured in degrees downward from the end of the feature specified by the bearing.

bearing [number or sequence of numbers] The bearing (azimuth) of the line(s) in degrees.

Returns

strike, dip [arrays] Arrays of strikes and dips in degrees following the right-hand-rule.

`mplstereonet.`**`geographic2pole`**(*lon*, *lat*)

Converts a longitude and latitude (from a stereonet) into the strike and dip of the plane whose pole lies at the given longitude(s) and latitude(s).

Parameters

lon [array-like] A sequence of longitudes (or a single longitude) in radians

lat [array-like] A sequence of latitudes (or a single latitude) in radians

Returns

strike [array] A sequence of strikes in degrees

dip [array] A sequence of dips in degrees

`mplstereonet.`**`vector2plunge_bearing`**(*x*, *y*, *z*)

Converts a vector or series of vectors given as x, y, z in world coordinates into plunge/bearings.

Parameters

x [number or sequence of numbers] The x-component(s) of the normal vector

y [number or sequence of numbers] The y-component(s) of the normal vector

z [number or sequence of numbers] The z-component(s) of the normal vector

Returns

plunge [array] The plunge of the vector in degrees downward from horizontal.

bearing [array] The bearing of the vector in degrees clockwise from north.

`mplstereonet.`**`geographic2plunge_bearing`**(*lon*, *lat*)

Converts longitude and latitude in stereonet coordinates into a plunge/bearing.

Parameters

lon, lat [numbers or sequences of numbers] Longitudes and latitudes in radians as measured from a lower-hemisphere stereonet

Returns

plunge [array] The plunge of the vector in degrees downward from horizontal.

bearing [array] The bearing of the vector in degrees clockwise from north.

`mplstereonet.`**`density_grid`**(**args*, ***kwargs*)

Estimates point density of the given linear orientation measurements (Interpreted as poles, lines, rakes, or “raw” longitudes and latitudes based on the *measurement* keyword argument.). Returns a regular (in lat-long space) grid of density estimates over a hemispherical surface.

Parameters

***args** [2 or 3 sequences of measurements] By default, this will be expected to be `strike` & `dip`, both array-like sequences representing poles to planes. (Rake measurements require three parameters, thus the variable number of arguments.) The `measurement` kwarg controls how these arguments are interpreted.

measurement [string, optional] Controls how the input arguments are interpreted. Defaults to `"poles"`. May be one of the following:

`"poles"` [strikes, dips] Arguments are assumed to be sequences of strikes and dips of planes. Poles to these planes are used for contouring.

`"lines"` [plunges, bearings] Arguments are assumed to be sequences of plunges and bearings of linear features.

`"rakes"` [strikes, dips, rakes] Arguments are assumed to be sequences of strikes, dips, and rakes along the plane.

`"radians"` [lon, lat] Arguments are assumed to be “raw” longitudes and latitudes in the stereonet’s underlying coordinate system.

method [string, optional] The method of density estimation to use. Defaults to `"exponential_kamb"`. May be one of the following:

`"exponential_kamb"` [Kamb with exponential smoothing] A modified Kamb method using exponential smoothing [1]. Units are in numbers of standard deviations by which the density estimate differs from uniform.

`"linear_kamb"` [Kamb with linear smoothing] A modified Kamb method using linear smoothing [1]. Units are in numbers of standard deviations by which the density estimate differs from uniform.

`"kamb"` [Kamb with no smoothing] Kamb’s method [2] with no smoothing. Units are in numbers of standard deviations by which the density estimate differs from uniform.

`"schmidt"` [1% counts] The traditional “Schmidt” (a.k.a. 1%) method. Counts points within a counting circle comprising 1% of the total area of the hemisphere. Does not take into account sample size. Units are in points per 1% area.

sigma [int or float, optional] The number of standard deviations defining the expected number of standard deviations by which a random sample from a uniform distribution of points would be expected to vary from being evenly distributed across the hemisphere. This controls the size of the counting circle, and therefore the degree of smoothing. Higher sigmas will lead to more smoothing of the resulting density distribution. This parameter only applies to Kamb-based methods. Defaults to 3.

gridsize [int or 2-item tuple of ints, optional] The size of the grid that the density is estimated on. If a single int is given, it is interpreted as an NxN grid. If a tuple of ints is given it is interpreted as (nrows, ncols). Defaults to 100.

weights [array-like, optional] The relative weight to be applied to each input measurement. The array will be normalized to sum to 1, so absolute value of the weights do not affect the result. Defaults to None.

Returns

xi, yi, zi [2D arrays] The longitude, latitude and density values of the regularly gridded density estimates. Longitude and latitude are in radians.

See also:

`mplstereonet.StereonetAxes.density_contourf`

`mplstereonet.StereonetAxes.density_contour`

References

[1], [2]

`mplstereonet.`**`plane_intersection`**(*strike1*, *dip1*, *strike2*, *dip2*)

Finds the intersection of two planes. Returns a plunge/bearing of the linear intersection of the two planes.

Also accepts sequences of strike1s, dip1s, strike2s, dip2s.

Parameters

strike1, dip1 [numbers or sequences of numbers] The strike and dip (in degrees, following the right-hand-rule) of the first plane(s).

strike2, dip2 [numbers or sequences of numbers] The strike and dip (in degrees, following the right-hand-rule) of the second plane(s).

Returns

plunge, bearing [arrays] The plunge and bearing(s) (in degrees) of the line representing the intersection of the two planes.

`mplstereonet.`**`xyz2stereonet`**(*x*, *y*, *z*)

Converts x, y, z in _world_ cartesian coordinates into lower-hemisphere stereonet coordinates.

Parameters

x, y, z [array-likes] Sequences of world coordinates

Returns

lon, lat [arrays] Sequences of longitudes and latitudes (in radians)

`mplstereonet.`**`stereonet2xyz`**(*lon*, *lat*)

Converts a sequence of longitudes and latitudes from a lower-hemisphere stereonet into _world_ x,y,z coordinates.

Parameters

lon, lat [array-likes] Sequences of longitudes and latitudes (in radians) from a lower-hemisphere stereonet

Returns

x, y, z [arrays] The world x,y,z components of the vectors represented by the lon, lat coordinates on the stereonet.

`mplstereonet.`**`vector2pole`**(*x*, *y*, *z*)

Converts a vector or series of vectors given as x, y, z in world coordinates into the strike/dip of the planes whose normal vectors are parallel to the specified vectors. (In other words, each xi,yi,zi is treated as a normal vector and this returns the strike/dip of the corresponding plane.)

Parameters

x [number or sequence of numbers] The x-component(s) of the normal vector

y [number or sequence of numbers] The y-component(s) of the normal vector

z [number or sequence of numbers] The z-component(s) of the normal vector

Returns

strike [array] The strike of the plane, in degrees clockwise from north. Dip direction is indicated by the "right hand rule".

dip [array] The dip of the plane, in degrees downward from horizontal.

`mplstereonet.`**`antipode`**(*lon*, *lat*)

Calculates the antipode (opposite point on the globe) of the given point or points. Input and output is expected to be in radians.

Parameters

lon [number or sequence of numbers] Longitude in radians

lat [number or sequence of numbers] Latitude in radians

Returns

lon, lat [arrays] Sequences (regardless of whether or not the input was a single value or a sequence) of longitude and latitude in radians.

`mplstereonet.`**`project_onto_plane`**(*strike*, *dip*, *plunge*, *bearing*)

Projects a linear feature(s) onto the surface of a plane. Returns a rake angle(s) along the plane.

This is also useful for finding the rake angle of a feature that already intersects the plane in question.

Parameters

strike, dip [numbers or sequences of numbers] The strike and dip (in degrees, following the right-hand-rule) of the plane(s).

plunge, bearing [numbers or sequences of numbers] The plunge and bearing (in degrees) or of the linear feature(s) to be projected onto the plane.

Returns

rake [array] A sequence of rake angles measured downwards from horizontal in degrees. Zero degrees corresponds to the "right- hand" direction indicated by the strike, while a negative angle corresponds to the opposite direction. Rakes returned by this function will always be between -90 and 90 (inclusive).

`mplstereonet.`**`azimuth2rake`**(*strike*, *dip*, *azimuth*)

Projects an azimuth of a linear feature onto a plane as a rake angle.

Parameters

strike, dip [numbers] The strike and dip of the plane in degrees following the right-hand-rule.

azimuth [numbers] The azimuth of the linear feature in degrees clockwise from north (i.e. a 0-360 azimuth).

Returns

rake [number] A rake angle in degrees measured downwards from horizontal. Negative values correspond to the opposite end of the strike.

`mplstereonet.`**`parse_azimuth`**(*azimuth*)

Parses an azimuth measurement in azimuth or quadrant format.

Parameters

azimuth [string or number] An azimuth measurement in degrees or a quadrant measurement of azimuth.

Returns

azi [float] The azimuth in degrees clockwise from north (range: 0-360)

See also:

`parse_quadrant_measurement`

`parse_strike_dip`

`parse_plunge_bearing`

`mplstereonet.`**`parse_quadrant_measurement`**(*quad_azimuth*)

Parses a quadrant measurement of the form "AxxB", where A and B are cardinal directions and xx is an angle measured relative to those directions.

In other words, it converts a measurement such as E30N into an azimuth of 60 degrees, or W10S into an azimuth of 260 degrees.

For ambiguous quadrant measurements such as "N30S", a ValueError is raised.

Parameters

quad_azimuth [string] An azimuth measurement in quadrant form.

Returns

azi [float] An azimuth in degrees clockwise from north.

See also:

`parse_azimuth`

`mplstereonet.`**`parse_strike_dip`**(*strike*, *dip*)

Parses strings of strike and dip and returns strike and dip measurements following the right-hand-rule.

Dip directions are parsed, and if the measurement does not follow the right-hand-rule, the opposite end of the strike measurement is returned.

Accepts either quadrant-formatted or azimuth-formatted strikes.

For example, this would convert a strike of "N30E" and a dip of "45NW" to a strike of 210 and a dip of 45.

Parameters

strike [string] A strike measurement. May be in azimuth or quadrant format.

dip [string] The dip angle and direction of a plane.

Returns

azi [float] Azimuth in degrees of the strike of the plane with dip direction indicated following the right-hand-rule.

dip [float] Dip of the plane in degrees.

`mplstereonet.`**`parse_rake`**(*strike*, *dip*, *rake*)

Parses strings of strike, dip, and rake and returns a strike, dip, and rake measurement following the right-hand-rule, with the "end" of the strike that the rake is measured from indicated by the sign of the rake (positive rakes correspond to the strike direction, negative rakes correspond to the opposite end).

Accepts either quadrant-formatted or azimuth-formatted strikes.

For example, this would convert a strike of "N30E", dip of "45NW", with a rake of "10NE" to a strike of 210, dip of 45, and rake of 170.

Rake angles returned by this function will always be between 0 and 180

If no directions are specified, the measuriement is assumed to follow the usual right-hand-rule convention.

Parameters

strike [string] A strike measurement. May be in azimuth or quadrant format.

dip [string] The dip angle and direction of a plane.

rake [string] The rake angle and direction that the rake is measured from.

Returns

strike, dip, rake [floats] Measurements of strike, dip, and rake following the conventions outlined above.

`mplstereonet.`**`parse_plunge_bearing`**(*plunge*, *bearing*)

Parses strings of plunge and bearing and returns a consistent plunge and bearing measurement as floats. Plunge angles returned by this function will always be between 0 and 90.

If no direction letter(s) is present, the plunge is assumed to be measured from the end specified by the bearing. If a direction letter(s) is present, the bearing will be switched to the opposite (180 degrees) end if the specified direction corresponds to the opposite end specified by the bearing.

Parameters

plunge [string] A plunge measurement.

bearing [string] A bearing measurement. May be in azimuth or quadrant format.

Returns

plunge, bearing: floats The plunge and bearing following the conventions outlined above.

Examples

```
>>> parse_plunge_bearing("30NW", 160)
... (30, 340)
```

mplstereonet.**subplots**(*nrows=1*, *ncols=1*, *sharex=False*, *sharey=False*, *squeeze=True*, *subplot_kw=None*, *hemisphere='lower'*, *projection='equal_area'*, ***fig_kw*)

Identical to matplotlib.pyplot.subplots, except that this will default to producing equal-area stereonet axes.

This prevents constantly doing:

```
>>> fig, ax = plt.subplot(subplot_kw=dict(projection='stereonet'))
```

or

```
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111, projection='stereonet')
```

Using this function also avoids having `mplstereonet` continually appear to be an unused import when one of the above methods are used.

Parameters

nrows [int] Number of rows of the subplot grid. Defaults to 1.

ncols [int] Number of columns of the subplot grid. Defaults to 1.

hemisphere [string] Currently this has no effect. When upper hemisphere and dual hemisphere plots are implemented, this will control which hemisphere is displayed.

projection [string] The projection for the axes. Defaults to 'equal_area'–an equal-area (a.k.a. "Schmidtt") stereonet. May also be 'equal_angle' for an equal-angle (a.k.a. "Wulff") stereonet or any other valid matplotlib projection (e.g. 'polar' or 'rectilinear' for a "normal" axes).

The following parameters are identical to matplotlib.pyplot.subplots:

sharex [string or bool] If *True*, the X axis will be shared amongst all subplots. If *True* and you have multiple rows, the x tick labels on all but the last row of plots will have visible set to *False* If a string must be one of "row", "col", "all", or "none". "all" has the same effect as *True*, "none" has the same effect as *False*. If "row", each subplot row will share a X axis. If "col", each subplot column will share a X axis and the x tick labels on all but the last row will have visible set to *False*.

sharey [string or bool] If *True*, the Y axis will be shared amongst all subplots. If *True* and you have multiple columns, the y tick labels on all but the first column of plots will have visible set to *False* If a string must be one of "row", "col", "all", or "none". "all" has the same effect as *True*, "none" has the same effect as *False*. If "row", each subplot row will share a Y axis. If "col", each subplot column will share a Y axis and the y tick labels on all but the last row will have visible set to *False*.

squeeze [bool]

If *True*, extra dimensions are squeezed out from the returned axis object:

- if only one subplot is constructed (nrows=ncols=1), the resulting single Axis object is returned as a scalar.
- for Nx1 or 1xN subplots, the returned object is a 1-d numpy object array of Axis objects are returned as numpy 1-d arrays.
- for NxM subplots with N>1 and M>1 are returned as a 2d array.

If *False*, no squeezing at all is done: the returned axis object is always a 2-d array contaning Axis instances, even if it ends up being 1x1.

subplot_kw [dict] Dict with keywords passed to the `add_subplot()` call used to create each subplots.

fig_kw [dict] Dict with keywords passed to the `figure()` call. Note that all keywords not recognized above will be automatically included here.

Returns

fig, ax [tuple]

- *fig* is the `matplotlib.figure.Figure` object
- ***ax* can be either a single axis object or an array of axis** objects if more than one supblot was created. The dimensions of the resulting array can be controlled with the squeeze keyword, see above.

`mplstereonet.`**`pole2plunge_bearing`**(*strike*, *dip*)

Converts the given *strike* and *dip* in dgrees of a plane(s) to a plunge and bearing of its pole.

Parameters

strike [number or sequence of numbers] The strike of the plane(s) in degrees, with dip direction indicated by the azimuth (e.g. 315 vs. 135) specified following the "right hand rule".

dip [number or sequence of numbers] The dip of the plane(s) in degrees.

Returns

plunge, bearing [arrays] Arrays of plunges and bearings of the pole to the plane(s) in degrees.

`mplstereonet.`**`fit_girdle`**(**args*, ***kwargs*)

Fits a plane to a scatter of points on a stereonet (a.k.a. a "girdle").

Input arguments will be interpreted as poles, lines, rakes, or "raw" longitudes and latitudes based on the `measurement` keyword argument. (Defaults to `"poles"`.)

Parameters

***args** [2 or 3 sequences of measurements] By default, this will be expected to be `strikes` & `dips`, both array-like sequences representing poles to planes. (Rake measurements require three parameters, thus the variable number of arguments.) The *measurement* kwarg controls how these arguments are interpreted.

measurement [{'poles', 'lines', 'rakes', 'radians'}, optional] Controls how the input arguments are interpreted. Defaults to `"poles"`. May be one of the following:

`"poles"` [Arguments are assumed to be sequences of strikes and] dips of planes. Poles to these planes are used for density contouring.

`"lines"` [Arguments are assumed to be sequences of plunges and] bearings of linear features.

`"rakes"` [Arguments are assumed to be sequences of strikes,] dips, and rakes along the plane.

`"radians"` [Arguments are assumed to be "raw" longitudes and] latitudes in the underlying projection's coordinate system.

bidirectional [boolean, optional] Whether or not the antipode of each measurement will be used in the calculation. For almost all use cases, it should. Defaults to True.

Returns

strike, dip: floats The strike and dip of the plane.

Notes

The pole to the best-fit plane is extracted by calculating the smallest eigenvector of the covariance matrix of the input measurements in cartesian 3D space.

Examples

Calculate the plunge of a cylindrical fold axis from a series of strike/dip measurements of bedding from the limbs:

```
>>> strike = [270, 334, 270, 270]
>>> dip = [20, 15, 80, 78]
>>> s, d = mplstereonet.fit_girdle(strike, dip)
>>> plunge, bearing = mplstereonet.pole2plunge_bearing(s, d)
```

`mplstereonet.`**`fit_pole`**`(`**args*, ***kwargs*`)`

Fits the pole to a plane to a “bullseye” of points on a stereonet.

Input arguments will be interpreted as poles, lines, rakes, or “raw” longitudes and latitudes based on the `measurement` keyword argument. (Defaults to `"poles"`.)

Parameters

***args** [2 or 3 sequences of measurements] By default, this will be expected to be `strike` & `dip`, both array-like sequences representing poles to planes. (Rake measurements require three parameters, thus the variable number of arguments.) The *measurement* kwarg controls how these arguments are interpreted.

measurement [{‘poles’, ‘lines’, ‘rakes’, ‘radians’}, optional] Controls how the input arguments are interpreted. Defaults to `"poles"`. May be one of the following:

`"poles"` [Arguments are assumed to be sequences of strikes and] dips of planes. Poles to these planes are used for density contouring.

`"lines"` [Arguments are assumed to be sequences of plunges and] bearings of linear features.

`"rakes"` [Arguments are assumed to be sequences of strikes,] dips, and rakes along the plane.

`"radians"` [Arguments are assumed to be “raw” longitudes and] latitudes in the underlying projection’s coordinate system.

bidirectional [boolean, optional] Whether or not the antipode of each measurement will be used in the calculation. For almost all use cases, it should. Defaults to True.

Returns

strike, dip: floats The strike and dip of the plane.

Notes

The pole to the best-fit plane is extracted by calculating the largest eigenvector of the covariance matrix of the input measurements in cartesian 3D space.

Examples

Find the average strike/dip of a series of bedding measurements

```
>>> strike = [270, 65, 280, 300]
>>> dip = [20, 15, 10, 5]
>>> strike0, dip0 = mplstereonet.fit_pole(strike, dip)
```

`mplstereonet.`**`eigenvectors`**`(`**args*, ***kwargs*`)`

Finds the 3 eigenvectors and eigenvalues of the 3D covariance matrix of a series of geometries. This can be used to fit a plane/pole to a dataset or for shape fabric analysis (e.g. Flinn/Hsu plots).

Input arguments will be interpreted as poles, lines, rakes, or "raw" longitudes and latitudes based on the *measurement* keyword argument. (Defaults to `"poles"`.)

Parameters

***args** [2 or 3 sequences of measurements] By default, this will be expected to be `strike` & `dip`, both array-like sequences representing poles to planes. (Rake measurements require three parameters, thus the variable number of arguments.) The *measurement* kwarg controls how these arguments are interpreted.

measurement [{'poles', 'lines', 'rakes', 'radians'}, optional] Controls how the input arguments are interpreted. Defaults to `"poles"`. May be one of the following:

`"poles"` [Arguments are assumed to be sequences of strikes and] dips of planes. Poles to these planes are used for density contouring.

`"lines"` [Arguments are assumed to be sequences of plunges and] bearings of linear features.

`"rakes"` [Arguments are assumed to be sequences of strikes,] dips, and rakes along the plane.

`"radians"` [Arguments are assumed to be "raw" longitudes and] latitudes in the underlying projection's coordinate system.

bidirectional [boolean, optional] Whether or not the antipode of each measurement will be used in the calculation. For almost all use cases, it should. Defaults to True.

Returns

plunges, bearings, values [sequences of 3 floats each] The plunges, bearings, and eigenvalues of the three eigenvectors of the covariance matrix of the input data. The measurements are returned sorted in descending order relative to the eigenvalues. (i.e. The largest eigenvector/eigenvalue is first.)

Examples

Find the eigenvectors as plunge/bearing and eigenvalues of the 3D covariance matrix of a series of planar measurements:

```
>>> strikes = [270, 65, 280, 300]
>>> dips = [20, 15, 10, 5]
>>> plu, azi, vals = mplstereonet.eigenvectors(strikes, dips)
```

`mplstereonet.`**`kmeans`**`(`**args*, ***kwargs*`)`

Find centers of multi-modal clusters of data using a kmeans approach modified for spherical measurements.

Parameters

***args** [2 or 3 sequences of measurements] By default, this will be expected to be `strike` & `dip`, both array-like sequences representing poles to planes. (Rake measurements require three parameters, thus the variable number of arguments.) The `measurement` kwarg controls how these arguments are interpreted.

num [int] The number of clusters to find. Defaults to 2.

bidirectional [bool] Whether or not the measurements are bi-directional linear/planar features or directed vectors. Defaults to True.

tolerance [float] Iteration will continue until the centers have not changed by more than this amount. Defaults to 1e-5.

measurement [string, optional] Controls how the input arguments are interpreted. Defaults to `"poles"`. May be one of the following:

`"poles"` [strikes, dips] Arguments are assumed to be sequences of strikes and dips of planes. Poles to these planes are used for analysis.

`"lines"` [plunges, bearings] Arguments are assumed to be sequences of plunges and bearings of linear features.

`"rakes"` [strikes, dips, rakes] Arguments are assumed to be sequences of strikes, dips, and rakes along the plane.

`"radians"` [lon, lat] Arguments are assumed to be “raw” longitudes and latitudes in the stereonet’s underlying coordinate system.

Returns

centers [An Nx2 array-like] Longitude and latitude in radians of the centers of each cluster.

`mplstereonet.`**`find_mean_vector`**`(`**args*, ***kwargs*`)`

Returns the mean vector for a set of measurments. By default, this expects the input to be plunges and bearings, but the type of input can be controlled through the `measurement` kwarg.

Parameters

***args** [2 or 3 sequences of measurements] By default, this will be expected to be `plunge` & `bearing`, both array-like sequences representing linear features. (Rake measurements require three parameters, thus the variable number of arguments.) The *measurement* kwarg controls how these arguments are interpreted.

measurement [string, optional] Controls how the input arguments are interpreted. Defaults to `"lines"`. May be one of the following:

`"poles"` [strikes, dips] Arguments are assumed to be sequences of strikes and dips of planes. Poles to these planes are used for analysis.

`"lines"` [plunges, bearings] Arguments are assumed to be sequences of plunges and bearings of linear features.

`"rakes"` [strikes, dips, rakes] Arguments are assumed to be sequences of strikes, dips, and rakes along the plane.

`"radians"` [lon, lat] Arguments are assumed to be “raw” longitudes and latitudes in the stereonet’s underlying coordinate system.

Returns

mean_vector [tuple of two floats] The plunge and bearing of the mean vector (in degrees).

r_value [float] The length of the mean vector (a value between 0 and 1).

`mplstereonet.`**`find_fisher_stats`**(**args*, ***kwargs*)

Returns the mean vector and summary statistics for a set of measurements. By default, this expects the input to be plunges and bearings, but the type of input can be controlled through the `measurement` kwarg.

Parameters

***args** [2 or 3 sequences of measurements] By default, this will be expected to be `plunge` & `bearing`, both array-like sequences representing linear features. (Rake measurements require three parameters, thus the variable number of arguments.) The *measurement* kwarg controls how these arguments are interpreted.

conf [number] The confidence level (0-100). Defaults to 95%, similar to 2 sigma.

measurement [string, optional] Controls how the input arguments are interpreted. Defaults to `"lines"`. May be one of the following:

`"poles"` [strikes, dips] Arguments are assumed to be sequences of strikes and dips of planes. Poles to these planes are used for analysis.

`"lines"` [plunges, bearings] Arguments are assumed to be sequences of plunges and bearings of linear features.

`"rakes"` [strikes, dips, rakes] Arguments are assumed to be sequences of strikes, dips, and rakes along the plane.

`"radians"` [lon, lat] Arguments are assumed to be "raw" longitudes and latitudes in the stereonet's underlying coordinate system.

Returns

mean_vector: tuple of two floats A set consisting of the plunge and bearing of the mean vector (in degrees).

stats [tuple of three floats] `(r_value, confidence, kappa)` The `r_value` is the magnitude of the mean vector as a number between 0 and 1. The `confidence` radius is the opening angle of a small circle that corresponds to the confidence in the calculated direction, and is dependent on the input `conf`. The `kappa` value is the dispersion factor that quantifies the amount of dispersion of the given vectors, analgous to a variance/stddev.

`mplstereonet.`**`angular_distance`**(*first*, *second*, *bidirectional=True*)

Calculate the angular distance between two linear features or elementwise angular distance between two sets of linear features. (Note: a linear feature in this context is a point on a stereonet represented by a single latitude and longitude.)

Parameters

first [(lon, lat) 2xN array-like or sequence of two numbers] The longitudes and latitudes of the first measurements in radians.

second [(lon, lat) 2xN array-like or sequence of two numbers] The longitudes and latitudes of the second measurements in radians.

bidirectional [boolean] If True, only "inner" angles will be returned. In other words, all angles returned by this function will be in the range [0, pi/2] (0 to 90 in degrees). Otherwise, `first` and `second` will be treated as vectors going from the origin outwards instead of bidirectional infinite lines. Therefore, with `bidirectional=False`, angles returned by this function will be in the range [0, pi] (zero to 180 degrees).

Returns

dist [array] The elementwise angular distance between each pair of measurements in (lon1, lat1) and (lon2, lat2).

Examples

Calculate the angle between two lines specified as a plunge/bearing

```
>>> angle = angular_distance(line(30, 270), line(40, 90))
>>> np.degrees(angle)
array([ 70.])
```

Let’s do the same, but change the “bidirectional” argument:

```
>>> first, second = line(30, 270), line(40, 90)
>>> angle = angular_distance(first, second, bidirectional=False)
>>> np.degrees(angle)
array([ 110.])
```

Calculate the angle between two planes.

```
>>> angle = angular_distance(pole(0, 10), pole(180, 10))
>>> np.degrees(angle)
array([ 20.])
```

# :undoc-members:

Bibliography

[Kamb1956] Kamb, 1959. Ice Petrofabric Observations from Blue Glacier, Washington, in Relation to Theory and Experiment. Journal of Geophysical Research, Vol. 64, No. 11, pp. 1891–1909.

[Vollmer1995] Vollmer, 1995. C Program for Automatic Contouring of Spherical Orientation Data Using a Modified Kamb Method. Computers & Geosciences, Vol. 21, No. 1, pp. 31–49.

[Fisher1993] Fisher, N.I., Lewis, T., Embleton, B.J.J. (1993) “Statistical Analysis of Spherical Data”

[1] Vollmer, 1995. C Program for Automatic Contouring of Spherical Orientation Data Using a Modified Kamb Method. Computers & Geosciences, Vol. 21, No. 1, pp. 31–49.

[2] Kamb, 1959. Ice Petrofabric Observations from Blue Glacier, Washington, in Relation to Theory and Experiment. Journal of Geophysical Research, Vol. 64, No. 11, pp. 1891–1909.

[1] Vollmer, 1995. C Program for Automatic Contouring of Spherical Orientation Data Using a Modified Kamb Method. Computers & Geosciences, Vol. 21, No. 1, pp. 31–49.

[2] Kamb, 1959. Ice Petrofabric Observations from Blue Glacier, Washington, in Relation to Theory and Experiment. Journal of Geophysical Research, Vol. 64, No. 11, pp. 1891–1909.

[1] Vollmer, 1995. C Program for Automatic Contouring of Spherical Orientation Data Using a Modified Kamb Method. Computers & Geosciences, Vol. 21, No. 1, pp. 31–49.

[2] Kamb, 1959. Ice Petrofabric Observations from Blue Glacier, Washington, in Relation to Theory and Experiment. Journal of Geophysical Research, Vol. 64, No. 11, pp. 1891–1909.

Python Module Index

Index