PmagPy_notebook

December 10, 2014

1 An example IPython (Jupyter) notebook for paleomagnetic data analysis

This notebook demonstrates some of the functionality that is possible when using PmagPy functions in an interactive notebook environment

1.1 Import necessary function libraries for the data analysis

The code block below imports necessary libraries from PmagPy that define functions that will be used in the data analysis. Using 'sys.path.insert' allows you to point to the directory where you keep PmagPy in order to import it. **You will need to change the path to match where the PmagPy folder is on your computer.**

```
In [1]: import sys
        #change to match where the PmagPy folder is on your computer
        sys.path.insert(0, '/Users/ltauxe/PmagPy')
        import pmag,pmagplotlib,ipmag # import PmagPy functions
        import numpy, pandas, matplotlib.pylot # import scientic python functions
        %matplotlib inline # allow plots to be generated in the notebook
```

1.1.1 Scientific Python functions

The numpy, scipy, matplotlib and pandas libraries are standard libraries for scientific python (see http://www.scipy.org). '%matplotlib inline' is necessary to allow the plots to be generated within the notebook instead of in an external window.

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
```

1.2 Analyzing data from McMurdo Sound

Let's look at data from this study (http://earthref.org/doi/10.1029/2008GC002072):

Lawrence, K.P., Tauxe, L., Staudigel, H., Constable, C.G., Koppers, A., McIntosh, W., Johnson, C.L., Paleomagnetic field properties near the southern hemisphere tangent cylinder, Geochem. Geophys. Geosys., 10, Q01005, doi:10.1029/2008GC002072, 2009 http://onlinelibrary.wiley.com/doi/10.1029/2008GC002072/abstract

1.2.1 Reading data from MagIC format results files

First, the data needs to be imported into the notebook environment.

These data were downloaded from the MagIC database (http://earthref.org/doi/10.1029/2008GC002072) as a .txt file. The data were then unpacked on the command line using the `download_magic.py` program (which could also be done using the `unpack download file` button in QuickMagIC.py). We will concentrate on importing and using the resulting `pmag_results.txt` file below. The code below relies on the pandas (pd) dataframe structure which is a useful and user friendly way to wrangle data.

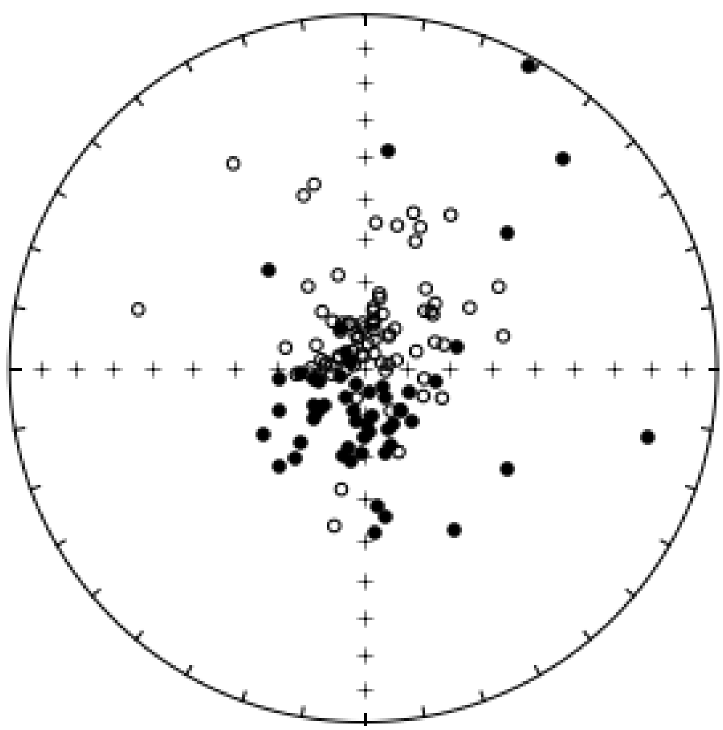
In [3]:

```
Out[3]:    antipodal  average_age  average_age_sigma average_age_unit  \
        0        NaN        1.180              0.005               Ma
        1        NaN        0.330              0.010               Ma
        2        NaN        0.348              0.004               Ma
        3        NaN        0.340              0.003               Ma
        4        NaN        4.000              4.000               Ma

           average_alpha95  average_dec  average_inc  average_int  average_int_n  \
        0              4.2        258.6         78.6          NaN            NaN
        1              2.1        328.6        -80.0          NaN            NaN
        2              2.3        352.0        -82.7          NaN            NaN
        3              4.6        352.1        -86.8          NaN            NaN
        4              4.8         13.6        -78.8          NaN            NaN

           average_int_sigma  ...   vadm_sigma  vdm  vdm_n  vdm_sigma  vgp_alpha95  \
        0                NaN  ...          NaN  NaN    NaN        NaN          NaN
        1                NaN  ...          NaN  NaN    NaN        NaN          NaN
        2                NaN  ...          NaN  NaN    NaN        NaN          NaN
        3                NaN  ...          NaN  NaN    NaN        NaN          NaN
        4                NaN  ...          NaN  NaN    NaN        NaN          NaN

           vgp_dm  vgp_dp  vgp_lat  vgp_lon vgp_n
        0     4.5     8.1    -67.3     95.2     7
        1     2.5     4.1     79.0    101.2     6
        2     3.8     4.4     87.1    123.1     6
        3    17.4     8.9     84.1    355.2     5
        4     5.2     9.3     79.8    196.0     5

        [5 rows x 47 columns]
```

1.2.2 Plotting site mean directions

First, the data needs to be imported into the notebook environment. These data were downloaded from the MagIC database (http://earthref.org/doi/10.1029/2008GC002072) as a .txt file. The data were then unpacked on the command line using the download_magic.py program (which could also be done using the unpack download file button in QuickMagIC.py). We will concentrate on importing and using the resulting pmag_results.txt file below. The code below relies on the pandas (pd) dataframe structure which is a useful and user friendly way to wrangle data. Now we can] plot it using `ipmag.plot_di`.

```
In [4]: data = pd.read_csv('Lawrence09_MagIC/pmag_results.txt',sep='        ',header=1)
        # screen out records with no directional data
        DI_results = data.dropna(subset = ['average_dec'])
        fignum = 1
        plt.figure(num=fignum,figsize=(6,6),dpi=160)
        ipmag.plot_net(fignum)
        plt.title('McMurdo site mean equal area plot')
        ipmag.plot_di(DI_results['average_dec'],DI_results['average_inc'])
```



1.2.3 Calculating and plotting Fisher means from the data

It can be seen in the plot above that the data are of dual polarity. To split the data by polarity, the function `pmag.doprinc` can be used to calculate the principal direction of the data set. This function takes a DIblock which is an array of [dec, inc] values. Results within 90° of the principal direction are of one polarity (reverse in this case), while results greater than 90° from that direction are of the other. This angle can be calculated using the `pmag.angle` function. This `pmag.angle` function can accept single values or arrays of values as is done here.

In [5]:
```
#make an 2xn array with all the declinations and inclinations
DIblock=np.array([DI_results.average_dec,DI_results.average_inc]).transpose()
# calculate the principle direction for the data set
principal=pmag.doprinc(DIblock)
print 'Principal direction declination: ' + str(principal['dec'])
print 'Principal direction inclination: ' + str(principal['inc'])
```

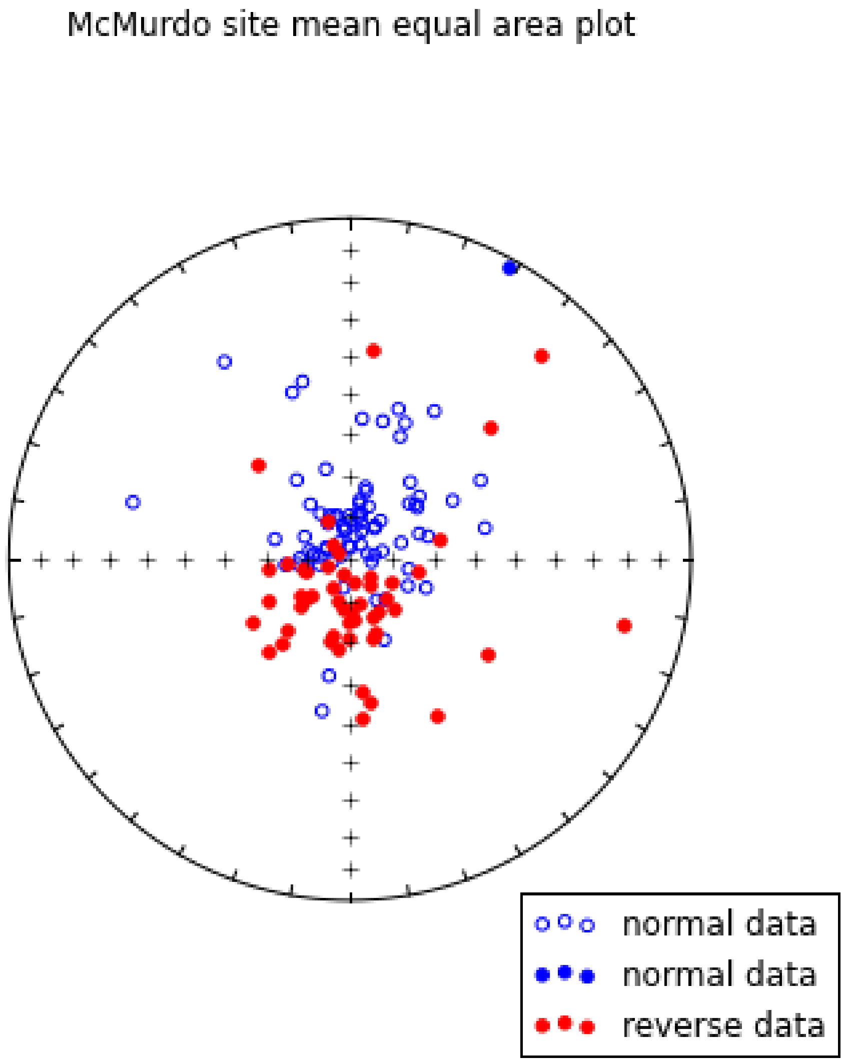
```
Principal direction declination: 189.094639423
Principal direction inclination: 80.8584727976
```

```
In [6]: DI_results['principal_dec'] = principal['dec']
        DI_results['principal_inc'] = principal['inc']
        principal_block=np.array([DI_results.principal_dec,DI_results.principal_inc]).transpose()
        DI_results['angle'] = pmag.angle(DIblock,principal_block)
        DI_results.ix[DI_results.angle<=90,'polarity'] = 'Reverse'
        DI_results.ix[DI_results.angle>90,'polarity'] = 'Normal'
        DI_results.head()
```

```
Out[6]:    antipodal  average_age  average_age_sigma average_age_unit  \
        0        NaN        1.180              0.005               Ma
        1        NaN        0.330              0.010               Ma
        2        NaN        0.348              0.004               Ma
        3        NaN        0.340              0.003               Ma
        4        NaN        4.000              4.000               Ma

           average_alpha95  average_dec  average_inc  average_int  average_int_n  \
        0              4.2        258.6         78.6          NaN            NaN
        1              2.1        328.6        -80.0          NaN            NaN
        2              2.3        352.0        -82.7          NaN            NaN
        3              4.6        352.1        -86.8          NaN            NaN
        4              4.8         13.6        -78.8          NaN            NaN

           average_int_sigma   ...     vgp_alpha95  vgp_dm  vgp_dp  vgp_lat  vgp_lon  \
        0                NaN   ...             NaN     4.5     8.1    -67.3     95.2
        1                NaN   ...             NaN     2.5     4.1     79.0    101.2
        2                NaN   ...             NaN     3.8     4.4     87.1    123.1
        3                NaN   ...             NaN    17.4     8.9     84.1    355.2
        4                NaN   ...             NaN     5.2     9.3     79.8    196.0

           vgp_n  principal_dec  principal_inc       angle polarity
        0      7     189.094639      80.858473   11.814579  Reverse
        1      6     189.094639      80.858473  173.353659   Normal
        2      6     189.094639      80.858473  176.958830   Normal
        3      5     189.094639      80.858473  173.847834   Normal
        4      5     189.094639      80.858473  177.794663   Normal

        [5 rows x 51 columns]
```

Now that polarity is assigned using the angle from the principal component, let's filter the data by polarity and then plot using different colors in order to visually inspect the polarity assignments.

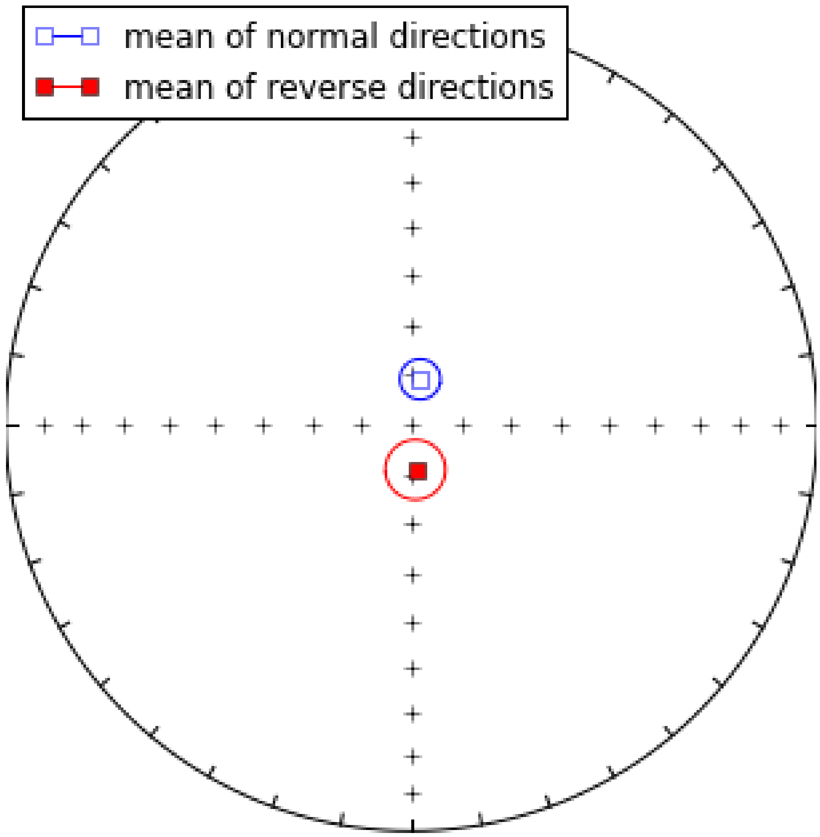
```
In [7]: normal_data = DI_results.ix[DI_results.polarity=='Normal'].reset_index(drop=True)
        reverse_data = DI_results.ix[DI_results.polarity=='Reverse'].reset_index(drop=True)

        fignum = 1
        plt.figure(num=fignum,figsize=(6,6),dpi=160)
        ipmag.plot_net(fignum)
        plt.title('McMurdo site mean equal area plot')
        ipmag.plot_di(normal_data['average_dec'],normal_data['average_inc'],color='b',label='normal data
        ipmag.plot_di(reverse_data['average_dec'],reverse_data['average_inc'],color='r',label='reverse d
        plt.legend(loc=4)
        plt.show()
```



Because these data are from a study in the sourthern hemisphere the normal direction is up. Fisher means for each polarity can be calculated using the Fisher mean pmag.py function (`pmag.fisher_mean`). This function returns a dictionary that gives the parameters associated with calculating a Fisher mean. These individual values can be called upon (e.g. normal_mean['dec']). A plot can be made of these calculate means along with their $\alpha95$ confidence ellipses using ipmag.plot_di_mean.

In [8]:
```
normal_directions = normal_data[['average_dec','average_inc']].values
reverse_directions = reverse_data[['average_dec','average_inc']].values

normal_mean = pmag.fisher_mean(normal_directions)
reverse_mean = pmag.fisher_mean(reverse_directions)

fignum = 1
plt.figure(num=fignum,figsize=(5,5))
ipmag.plot_net(fignum)
ipmag.plot_di_mean(normal_mean['dec'],normal_mean['inc'],normal_mean['alpha95'],
```

```
                        color='b',marker='s',label='mean of normal directions',legend='yes')
ipmag.plot_di_mean(reverse_mean['dec'],reverse_mean['inc'],reverse_mean['alpha95'],
                        color='r',marker='s',label='mean of reverse directions',legend='yes')
```



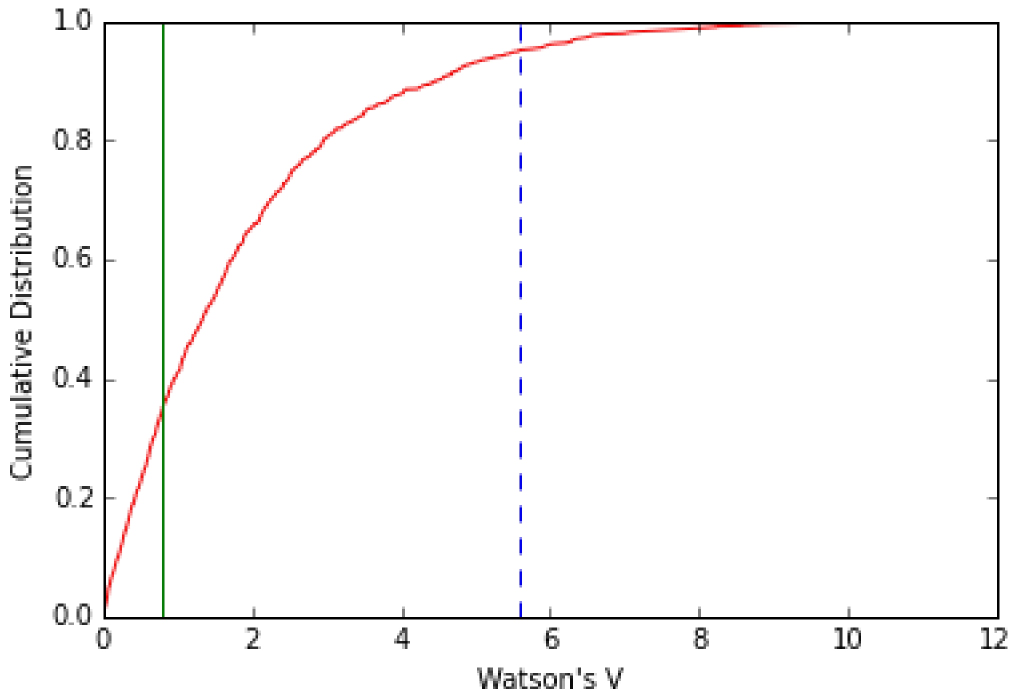
1.2.4 Conducting reversal tests on the data

Reversal tests are tests for a common mean between two data sets wherein the antipode vectors from one population are compared to the vectors of another population. The code below conducts the Watson V test (also returning the McFadden and McEllhinny (1990) reversal test classification) and conducts a bootstrap reversal test. In order to conduct the test, the antipode of one of the directional populations needs to be taken. The `ipmag.flip` function is used below to return the antipode of the reverse directions in order to conduct the tests.

```
In [9]: ipmag.watson_common_mean(normal_directions,ipmag.flip(reverse_directions),NumSims=1000,plot='ye
```

```
Results of Watson V test:

Watson's V:           0.8
Critical value of V:  5.6
"Pass": Since V is less than Vcrit, the null hypothesis
that the two populations are drawn from distributions
that share a common mean direction can not be rejected.
```

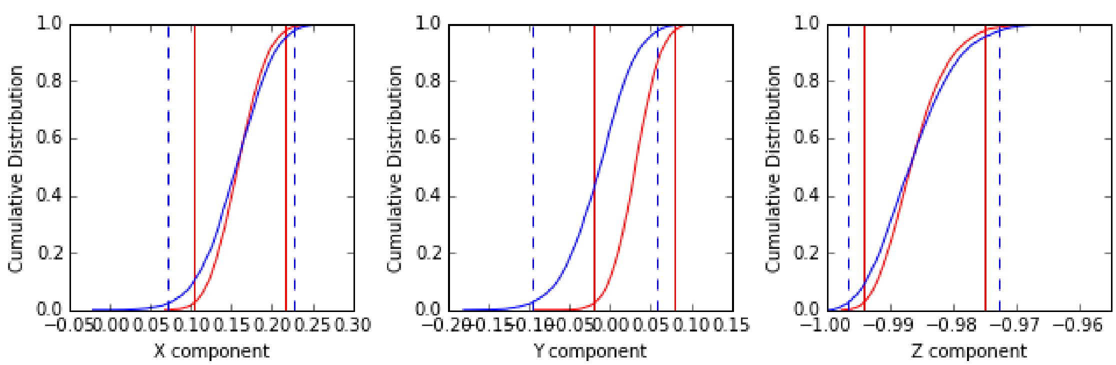
```
M&M1990 classification:

Angle between data set means: 2.6
Critical angle for M&M1990:   6.9
The McFadden and McElhinny (1990) classification for
this test is: 'B'
```



In [10]: `ipmag.bootstrap_common_mean(normal_directions,ipmag.flip(reverse_directions),NumSims=1000)`

```
Here are the results of the bootstrap test for a common mean:
```

Out[10]: `<matplotlib.axes._subplots.AxesSubplot at 0x112001190>`



1.2.5 Plotting virtual geomagnetic and mean poles

Now we can use the reverse_data and normal_data dataframes again to look at and analyze the virtual geomagnetic poles (VGPs). After taking the antipode of the reverse VGPs, a `combined_VGP_mean` is calculated using `pmag.fisher_mean`.

```
In [11]: # let's take the antipode of the reverse VGPs
         reverse_data['vgp_lon_flip'] = reverse_data['vgp_lon']-180.
         reverse_data['vgp_lat_flip'] = -reverse_data['vgp_lat']
         # here we combine the two sets of VGPs into one 2XN array
         combined_vgps = np.concatenate((normal_data[['vgp_lon','vgp_lat']].values,
                                         reverse_data[['vgp_lon_flip','vgp_lat_flip']].values))
         # and calculate the mean
         combined_VGP_mean = pmag.fisher_mean(combined_vgps)
         print 'Mean pole longitude: ' + str(combined_VGP_mean['dec'])
         print 'Mean pole latitude: ' + str(combined_VGP_mean['inc'])
         print 'A_95: ' + str(combined_VGP_mean['alpha95'])
```

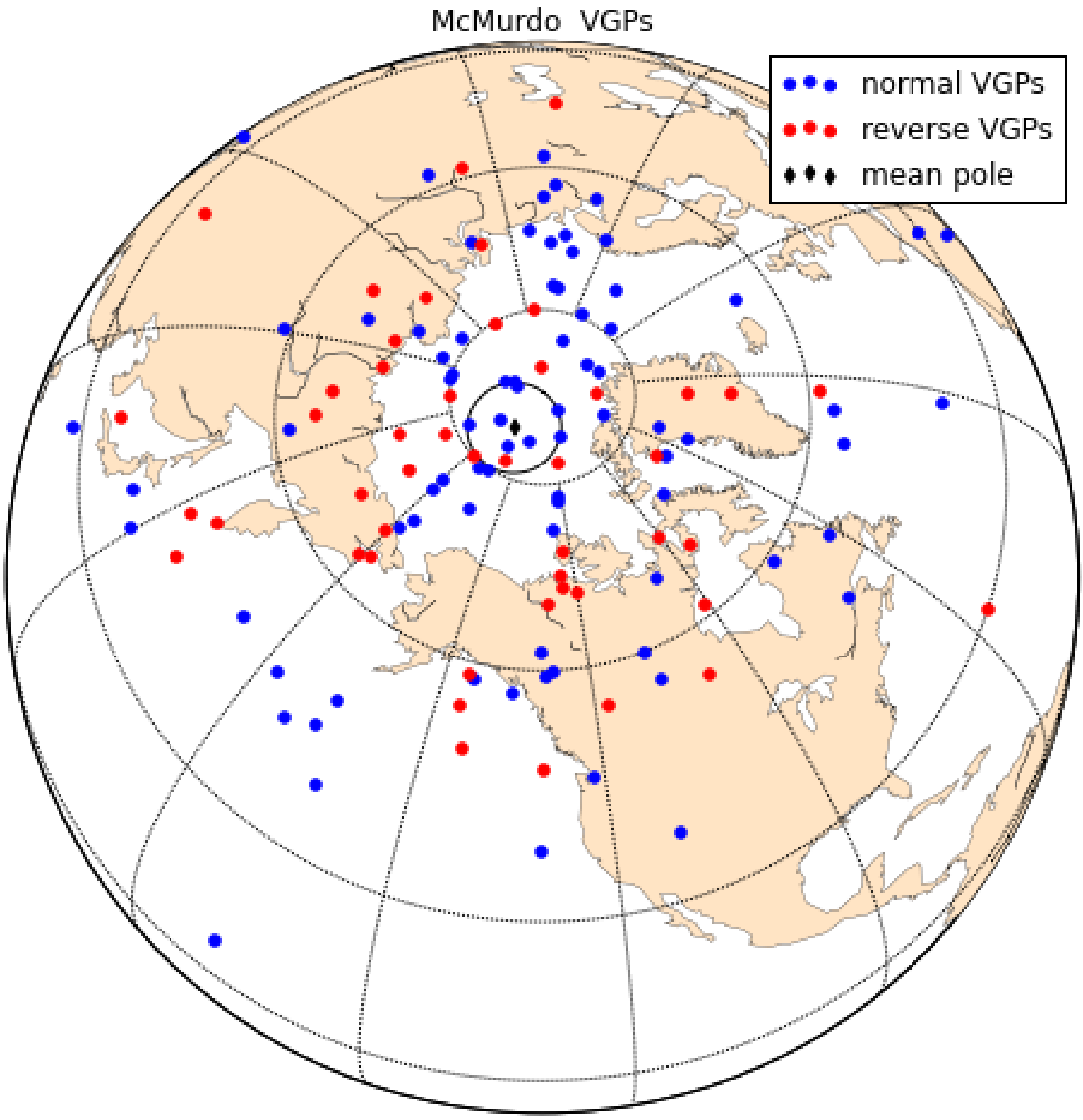
```
Mean pole longitude: 190.633317978
Mean pole latitude: 85.2294312198
A_95: 5.05059654685
```

To plot poles we first need a map projection. The matplotlib basemap package does a nice job of that, so let's import it below. Note that not all installations of python will have the basemap package so you may need to download it. It is an availible package through Enthought Canopy.

```
In [12]: from mpl_toolkits.basemap import Basemap
```

The basemap is highly customizable. A view from space type projection ('ortho') is a nice way to view data on a sphere so let's use that one for this example. We will define an object ('m') that is a map projection and then draw additional things on the map such as continents and lat/long lines. Then the VGPs from the study can be plotted using the `ipmag.plot_vgp` function and the mean pole can be plotted using `ipmag.plot_pole`.

```
In [13]: m = Basemap(projection='ortho',lat_0=70,lon_0=230,resolution='c',area_thresh=50000)
         plt.figure(figsize=(8, 8))
         m.drawcoastlines(linewidth=0.25)
         m.fillcontinents(color='bisque',lake_color='white',zorder=1)
         m.drawmapboundary(fill_color='white')
         m.drawmeridians(np.arange(0,360,30))
         m.drawparallels(np.arange(-90,90,30))

         ipmag.plot_vgp(m,normal_data['vgp_lon'].tolist(),
                        normal_data['vgp_lat'].tolist(),
                        color='b',label='normal VGPs')
         ipmag.plot_vgp(m,reverse_data['vgp_lon_flip'].tolist(),
                        reverse_data['vgp_lat_flip'].tolist(),
                        color='r',label='reverse VGPs')
         ipmag.plot_pole(m,combined_VGP_mean['dec'],combined_VGP_mean['inc'],
                        combined_VGP_mean['alpha95'],marker='d',label='mean pole')
         plt.legend()
         plt.title('McMurdo  VGPs')
         plt.show()
```



1.2.6 Previous code for reading in the data

```
In [14]: #read in the data
         data,file_type=pmag.magic_read('Lawrence09_MagIC/pmag_results.txt')
         # screen out records with no directional data
         directions=pmag.get_dictitem(data,'average_dec',"",'F')
         directions=pmag.get_dictitem(directions,'average_inc',"",'F')
         #create a pandas dataframe
         results = pd.DataFrame(directions)
         #the data come in as strings so need to be defined as floating point numbers
         results.average_dec = results.average_dec.astype(float)
         results.average_inc = results.average_inc.astype(float)
         # while we are at it, lets do the same on the VGPs because we will need them later
         results.vgp_lat = results.vgp_lat.astype(float)
```

```
results.vgp_lon= results.vgp_lon.astype(float)
#display the first 5 rows of the results dataframe
results.head()
```

Out[14]:
```
  antipodal average_age average_age_sigma average_age_unit average_alpha95  \
0                  1.18             0.005               Ma             4.2
1                  0.33              0.01               Ma             2.1
2                 0.348             0.004               Ma             2.3
3                  0.34             0.003               Ma             4.6
4                     4                 4               Ma             4.8

   average_dec  average_inc average_int average_int_n average_int_sigma  ...   \
0        258.6         78.6                                             ...
1        328.6        -80.0                                             ...
2        352.0        -82.7                                             ...
3        352.1        -86.8                                             ...
4         13.6        -78.8                                             ...

  vadm_sigma vdm vdm_n vdm_sigma vgp_alpha95 vgp_dm vgp_dp vgp_lat vgp_lon  \
0                                               4.5    8.1   -67.3    95.2
1                                               2.5    4.1    79.0   101.2
2                                               3.8    4.4    87.1   123.1
3                                              17.4    8.9    84.1   355.2
4                                               5.2    9.3    79.8   196.0

  vgp_n
0     7
1     6
2     6
3     5
4     5

[5 rows x 47 columns]
```

1.2.7 Previous code for assigning polarity

In [15]:
```
#make an 2xn array with all the declinations and inclinations
DIblock=np.array([DI_results.average_dec,DI_results.average_inc]).transpose()
# calculate the principle direction for the data set
principle=pmag.doprinc(DIblock)
# initialize arrays of length N with zeros
V1_dec,V1_inc=np.zeros(len(DIblock)),np.zeros(len(DIblock))
# fill arrays with declination and inclination of principal direction (V1)
V1_dec.fill(principle['dec'])
V1_inc.fill(principle['inc'])
# create array of length N,
V1=np.array([V1_dec,V1_inc]).transpose()
results.angle=pmag.angle(DIblock,V1)
print 'dec =', principle['dec'],'inc = ', principle['inc']

#separate into normal and reverse, based on  principle direction printed out above
NormalRecords=results[results.angle>90]
ReverseRecords=results[results.angle<=90]
normal_directions = NormalRecords[['average_dec','average_inc']].values
```

```
    reverse_directions = ReverseRecords[['average_dec','average_inc']].values
    # calculate the normal and reverse means with pmag.fisher_mean
```

```
dec = 189.094639423 inc =  80.8584727976
```