Supporting Materials for Confirmation of progressive plate motion during the Midcontinent Rift's early magmatic stage from the Osler Volcanic Group, Ontario, Canada

April 18, 2014

Corresponding Author: Nicholas L. Swanson-Hysell (swanson-hysell@berkeley.edu)

1 Introduction

This IPython notebook contains data analysis on paleomagnetic data developed from ca. 1.1 billion year old lava flows of the Osler Volcanic Group that are part of the North American Midcontinent Rift as well as supporting analyses on other data sets. These analyses accompany a **Geochemistry, Geophysics, Geosystems** manuscript entitled "Confirmation of progressive plate motion during the Midcontinent Rift's early magmatic stage from the Osler Volcanic Group, Ontario, Canada" by N. L. Swanson-Hysell, A. A. Vaughan, M. R. Mustain and K. Asp. This notebook is part of the supporting online materials and is available at `https://github.com/Swanson-Hysell/2014_Swanson-Hysell-et-al_Osler` along with the necessary files to execute all of the code. Within this github repository are the following folders:

1. 2014_Osler_Code This folder contains this notebook file as well as necessary libraries.
2. 2014_Osler_Data This folder contains the data generated in the study at the specimen level as well as the flow means that are imported into this notebook for data analysis.
3. 2014_Osler_Manuscript This folder contains the manuscript text and figures.

If you are viewing this supporting material as a PDF document and want to run the code in the notebook you will need to download the code from the github repository and you will also need a Python distribution that includes IPython. There are good instructions for installing IPython here: http://ipython.org/install.html. Alternatively you can view the notebook online with the IPython nbviewer at this link: http://bit.ly/19XRdjJ

2 Import libraries

This code blocks imports necessary libraries that define functions that will be used in the data analysis below. The code below uses the pmag.py and pmagplotlib.py files of the the PmagPy software package (version pmagpy-2.206) authored by Lisa Tauxe (https://github.com/ltauxe/PmagPy). The check_updates and get_version functions of the PmagPy libraries cause errors in the interactive environment of IPython so the code below calls a slightly modified version of those libraries that is included in the Github repository for this work. There are other functions that are necessary for the interactive plotting of paleomagnetic data (some of them edited from PmagPy) that are within the IPmag.py library that is imported in the code block below.

```
In [1]: #import paleomagnetic specific libraries
        import pmag, pmagplotlib, IPmag
        import pandas
        pandas.set_option('display.max_columns', 500)
        from IPython.core.display import HTML
        from mpl_toolkits.basemap import Basemap
```

This notebook runs with pylab inline which imports the numpy, scipy and matplotlib functions and allows for the plots to be viewed inline in the IPython notebook (instead of opening up in another window).

```
In [2]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

3 Import Simpson Island paleomagnetic data

The data file FlowDataAll.csv has data in this format:

SITE, STRAT_HEIGHT, DEC_GEO, INC_GEO, Dec_TC, INC_TC, A95, n, Plat, ABS_Plat, VGP_lat, VGP_long

where:

SITE is the name of the site and corresponds to an individual lava flow where in the first part of the site name (e.g. SI1) corresponds to the name of a stratigraphic section while the second part (e.g. (11.8 to 26.4)) corresponds to the stratigraphic ranges within the measured section over which the flows is exposed.

STRAT_HEIGHT is the cumulative stratigraphic height of the Simpson Island stratigraphy starting with the base of the SI1 section (see map in the main manuscript).

DEC_GEO is the site mean declination in geographic (i.e. *in situ*) coordinates prior to tilt-correction.

INC_GEO is the site mean inclination in geographic (i.e. *in situ*) coordinates prior to tilt-correction.

Dec_TC is the site mean declination in tilt-corrected coordinates following correction for bedding tilt.

INC_TC is the site mean inclination in tilt-corrected coordinates following correction for bedding tilt.

A95 is the 95% Fisher confidence ellipse around the calculated site mean.

n is the number of individually oriented samples that are being included to calculate the site mean.

Plat is the paleolatitude calculated from the site mean using the dipole equation.

ABS_Plat is the absolute value of this paleolatitude.

VGP_lat is the latitude of the virtual geomagnetic pole calculated using the site location and the site mean direction.

VGP_long is the longitude of the virtual geomagnetic pole calculated using the site location and the site mean direction.

These data are displayed in a table below and imported by variable in the cell below the table.

```
In [3]: data = pandas.read_csv('../2014_Osler_Data/SimpsonIsland_OslerData.csv')
        #write these data to a latex file for the supplemental PDF
        with open('OslerData.txt','w') as f:
            f.write(data.to_latex(index=False,columns=['Site','Strat_Height','Site_Lat','Site_Long','De
        display(HTML(data.to_html()))
```

Simpson Island paleomagnetic site mean data

Site	Strat_Height	Site_Lat	Site_Long	Dec_Geo	Inc_Geo	Dec_TC	Inc_TC	n	VGP_lat	VGP_long
SI1(11.8 to 26.4)	11.8	48.8122	-87.6620	120.3	-77.1	79.7	-70.5	7	33.1	229.6
SI1(28.3 to 29.2)	28.3	48.8107	-87.6623	141.0	-67.4	109.8	-66.8	7	45.8	210.9
SI1(29.2 to 29.7)	29.2	48.8104	-87.6622	131.2	-69.1	99.7	-66.2	3	39.6	214.5
SI1(33.3 to 37.1)	33.3	48.8100	-87.6623	127.7	-72.3	92.6	-68.1	5	37.2	220.5
SI1(40.6 to 42.5)	40.6	48.8095	-87.6626	134.9	-77.7	85.4	-73.3	6	38.2	231.6
SI1(42.5 to 44.4)	42.5	48.8095	-87.6627	103.0	-71.6	77.8	-63.5	6	25.8	222.3
SI1(58.1 to 64.1)	58.1	48.8086	-87.6622	114.4	-70.8	86.1	-64.4	5	30.8	218.8
SI1(87.6 to 88.5)	87.6	48.8073	-87.6620	134.3	-75.8	89.9	-71.9	6	39.0	227.7
SI1(88.5 to 89.2)	88.5	48.8071	-87.6619	128.2	-74.6	88.9	-70.0	5	36.9	224.9
SI1(89.2 to 91.5)	89.2	48.8070	-87.6618	120.1	-72.8	87.0	-67.2	5	33.6	221.8
SI1(91.5 to 92.6)	91.5	48.8069	-87.6618	118.4	-74.8	82.8	-68.5	6	32.7	225.5
SI1(94.3 to 94.9)	94.3	48.8068	-87.6620	145.5	-76.5	94.1	-74.4	5	42.8	230.8
SI1(94.9 to 96.6)	94.9	48.8066	-87.6621	136.6	-80.0	79.1	-75.0	5	37.2	236.6
SI1(116.3 to 118.8)	116.3	48.8062	-87.6634	112.9	-78.2	74.1	-70.3	6	30.4	231.8
SI1(119.7 to 122.1)	119.7	48.8060	-87.6634	124.2	-74.0	87.5	-68.9	5	35.3	223.9
SI1(122.1 to 123.7)	122.1	48.8058	-87.6633	111.3	-76.8	75.7	-69.1	6	30.0	229.6
SI2a (1.1 to 1.6)	442.1	48.7943	-87.6556	139.0	-54.1	117.1	-58.1	6	44.8	194.2
SI2a (10.2 to 12.8)	451.2	48.7940	-87.6555	171.8	-57.3	152.3	-68.8	6	72.2	203.7
SI2b (2.2 to 2.6)	519.2	48.7921	-87.6546	144.7	-57.3	120.0	-62.4	8	49.3	198.7
SI2b (10.5 to 10.6)	527.5	48.7916	-87.6539	140.7	-64.3	107.4	-67.3	5	44.7	212.8
SI2c (0.0 to 2.1)	592.0	48.7903	-87.6545	120.9	-38.7	108.7	-39.2	6	28.7	184.0
SI2c (2.1 to 18.6)	594.1	48.7902	-87.6545	127.2	-45.6	111.3	-47.3	6	34.7	187.5
SI3(2.3 to 5.5)	745.3	48.7828	-87.6326	156.0	-56.8	135.4	-69.3	6	62.0	208.0
SI3(5.5 to 12.5)	748.5	48.7826	-87.6323	154.3	-60.9	127.5	-72.5	10	58.2	218.9
SI3(12.5 to 20.1)	755.5	48.7823	-87.6318	145.9	-64.8	109.1	-73.7	6	49.3	225.3
SI3(20.1 to 28.9)	763.1	48.7821	-87.6316	140.1	-61.4	108.1	-69.3	9	46.7	216.0
SI3(36.9 to 48.4)	779.9	48.7817	-87.6310	136.0	-64.7	99.6	-70.5	6	42.6	221.6
SI3(55.8 to 60.9)	798.8	48.7812	-87.6304	139.9	-63.2	105.6	-70.7	6	45.9	219.8
SI3(103.3 to 110.3)	846.3	48.7799	-87.6295	122.4	-65.9	84.0	-68.1	6	32.9	224.5
SI3(124.8 to 130.4)	867.8	48.7791	-87.6292	114.1	-63.1	81.5	-63.7	5	27.8	220.5
SI9(0.0 to 9.2)	1183.0	48.7721	-87.6222	104.4	-73.3	54.5	-67.3	7	19.4	238.8
SI9(12.0 to 21.2)	1195.0	48.7718	-87.6221	107.7	-69.0	64.4	-65.3	8	21.1	231.5
SI9(66.9 to 84.0)	1249.9	48.7700	-87.6203	116.0	-70.0	67.4	-68.1	7	25.3	232.7
SI9(84.0 to 88.9)	1267.0	48.7696	-87.6204	104.7	-76.7	47.1	-69.2	8	19.1	244.4
SI9(139.8 to 160.0)	1322.8	48.7680	-87.6192	117.6	-48.6	96.0	-51.8	8	27.5	201.2
SI9(160.0 to 166.5)	1343.0	48.7676	-87.6189	120.6	-49.2	98.5	-53.2	8	29.9	200.8
SI9(166.5 to 171.4)	1349.5	48.7674	-87.6185	129.3	-45.9	109.6	-53.0	8	36.9	193.5
SI9(171.4 to 188.8)	1354.4	48.7672	-87.6190	127.2	-39.1	111.7	-46.1	6	34.3	186.3
SI9(312.1 to 321.0)	1495.1	48.7662	-87.6181	143.6	-50.4	121.8	-61.2	8	49.7	195.8
SI9(336.4 to 353.8)	1519.4	48.7630	-87.6151	135.7	-47.6	115.2	-56.4	8	42.6	193.4
SI9(375.9 to 381.0)	1558.9	48.7623	-87.6142	136.1	-46.3	116.6	-55.3	4	42.8	191.3
SI9(387.6 to 395.3)	1570.6	48.7620	-87.6140	137.6	-46.8	117.8	-56.2	7	44.1	191.4
SI9(395.3 to 399.3)	1578.3	48.7617	-87.6138	143.2	-53.4	118.3	-63.8	7	49.0	201.9
SI4(0.0 to 13.8)	1983.0	48.7525	-87.5971	152.7	-60.8	119.4	-72.7	6	54.1	220.8
SI4(13.8 to 20.2)	1996.8	48.7524	-87.5973	148.6	-62.2	111.4	-72.6	6	49.9	222.2
SI4(21.4 to 30.0)	2004.4	48.7520	-87.5972	140.7	-63.1	100.7	-70.8	7	43.4	221.9
SI4(39.7 to 44.7)	2022.7	48.7516	-87.5967	165.5	-58.5	143.2	-73.9	6	66.2	224.4
SI4(72.3 to 74.6)	2055.3	48.7508	-87.5957	159.1	-58.6	132.4	-72.7	6	60.9	219.0
SI4(74.6 to 80.0)	2057.6	48.7507	-87.5953	152.5	-52.0	131.4	-65.0	6	57.9	198.0
SI4(80.2 to 100.7)	2063.2	48.7505	-87.5951	145.1	-54.1	119.8	-64.8	8	50.5	203.1
SI4(106.0 to 121.4)	2089.0	48.7499	-87.5953	140.0	-46.6	120.7	-56.8	7	46.4	190.3
SI4(121.4 to 127.3)	2104.4	48.7494	-87.5954	129.6	-43.6	111.5	-50.9	7	36.9	190.4
SI4(133.8 to 143.1)	2116.8	48.7490	-87.5955	134.8	-52.7	109.6	-60.6	7	41.7	201.8
SI4(160.4 to 171.1)	2143.4	48.7485	-87.5958	163.7	-46.6	151.1	-62.4	8	69.5	179.1
SI8(0.0 to 3.9)	2336.0	48.7466	-87.6194	152.4	-56.4	126.1	-68.9	8	56.4	209.4
SI8(3.9 to 19.3)	2339.9	48.7465	-87.6195	153.2	-59.3	123.0	-71.6	7	55.6	217.1
SI8(19.3 to 45.0)	2355.3	48.7462	-87.6200	132.9	-51.6	108.7	-59.1	8	40.1	200.5
SI8(47.5 to 56.7)	2383.5	48.7458	-87.6211	145.6	-53.4	121.2	-64.4	6	51.2	201.6
SI8(62.9 to 84.4)	2398.9	48.7456	-87.6223	140.2	-61.5	103.3	-69.6	8	43.9	218.6
SI8(84.4 to 102.6)	2420.4	48.7453	-87.6239	121.2	-64.7	80.9	-66.0	8	29.5	223.4
SI8(106.6 to 115.4)	2442.6	48.7449	-87.6276	135.2	-58.0	103.8	-65.2	8	41.2	211.2
SI6(0.0 to 3.0)	2451.0	48.7462	-87.6393	137.0	-55.3	103.2	-63.3	5	39.5	208.7
SI6(12.0 to 28.4)	2463.0	48.7460	-87.6394	120.8	-62.8	78.3	-63.0	8	25.5	221.6
SI6(40.6 to 57.8)	2491.6	48.7451	-87.6393	134.8	-59.1	95.2	-65.3	6	36.5	215.5
SI6(76.5 to 94.1)	2527.5	48.7448	-87.6486	149.8	-61.3	105.2	-72.2	7	46.5	223.0
SI6(94.1 to 104.2)	2545.1	48.7445	-87.6486	160.6	-65.4	103.4	-78.6	5	49.0	238.8
SI6(122.3 to 127.6)	2573.3	48.7439	-87.6493	159.0	-70.0	79.9	-79.5	7	41.7	245.2
SI6(180.6 to 188.8)	2631.6	48.7423	-87.6500	105.6	-56.6	75.3	-53.3	8	16.2	215.6
SI6(208.4 to 219.9)	2659.4	48.7415	-87.6483	125.1	-46.8	100.9	-52.0	5	30.7	198.3
SI6(245.4 to 252.7)	2696.4	48.7405	-87.6476	133.1	-47.3	108.3	-55.3	7	37.5	196.6
SI6(268.4 to 289.5)	2719.4	48.7399	-87.6467	135.0	-60.4	93.1	-66.3	8	36.1	217.8
SI6(289.8 to 301.1)	2740.8	48.7393	-87.6464	133.7	-54.0	101.7	-61.0	8	37.1	206.7
SI7 (0.0 to 3.9)	2798.0	48.7406	-87.6682	143.6	-37.9	128.6	-48.4	8	46.7	175.2
SI7 (3.9 to 5.4)	2801.9	48.7405	-87.6684	149.4	-41.7	132.8	-53.6	7	52.4	177.4
SI7 (9.6 to 18.0)	2807.6	48.7403	-87.6684	141.2	-46.6	120.1	-55.6	7	45.3	189.2
SI5b(3.2 to 30.7)	2843.2	48.7408	-87.6765	136.3	-50.0	111.9	-57.1	6	40.9	196.2
SI5b(57.2 to 63.3)	2897.2	48.7402	-87.6838	115.1	-41.3	97.9	-42.5	6	23.3	193.7
SI5b(64.6 to 70.4)	2904.6	48.7401	-87.6841	121.4	-47.9	99.2	-50.4	5	28.6	198.0
SI5b(72.6 to 79.0)	2912.6	48.7399	-87.6845	118.8	-47.7	97.0	-49.4	6	26.6	198.8
SI5b(115.0 to 120.2)	2955.0	48.7388	-87.6855	137.6	-48.3	114.7	-56.0	9	42.0	193.2
SI5b(132.4 to 171.9)	2972.4	48.7385	-87.6864	133.6	-55.5	103.5	-60.9	7	38.1	205.5
SI5b(183.4 to 188.7)	3023.4	48.7376	-87.6934	142.4	-52.3	116.0	-61.0	5	45.9	198.8
SI5b(197.5 to 216.7)	3037.5	48.7373	-87.6935	142.5	-63.7	98.7	-70.0	6	41.8	221.0
SI5a(0.0 to 9.0)	3115.0	48.7363	-87.6962	133.7	-45.7	112.8	-52.5	12	38.7	190.9

The .csv of these data can be downloaded here: SimpsonIsland_OslerData.csv

In [4]:
```
data_file='../2014_Osler_Data/SimpsonIsland_OslerData.csv'
SimpsonIsland_Strat = np.genfromtxt(data_file,delimiter=",", skip_header=1)

strat_height=SimpsonIsland_Strat[:,1]
Dec_TC=SimpsonIsland_Strat[:,6]
Inc_TC=SimpsonIsland_Strat[:,7]
A95=SimpsonIsland_Strat[:,8]
plat=SimpsonIsland_Strat[:,10]
abs_plat=SimpsonIsland_Strat[:,11]
VGP_lat=SimpsonIsland_Strat[:,12]
VGP_long=SimpsonIsland_Strat[:,13]
```

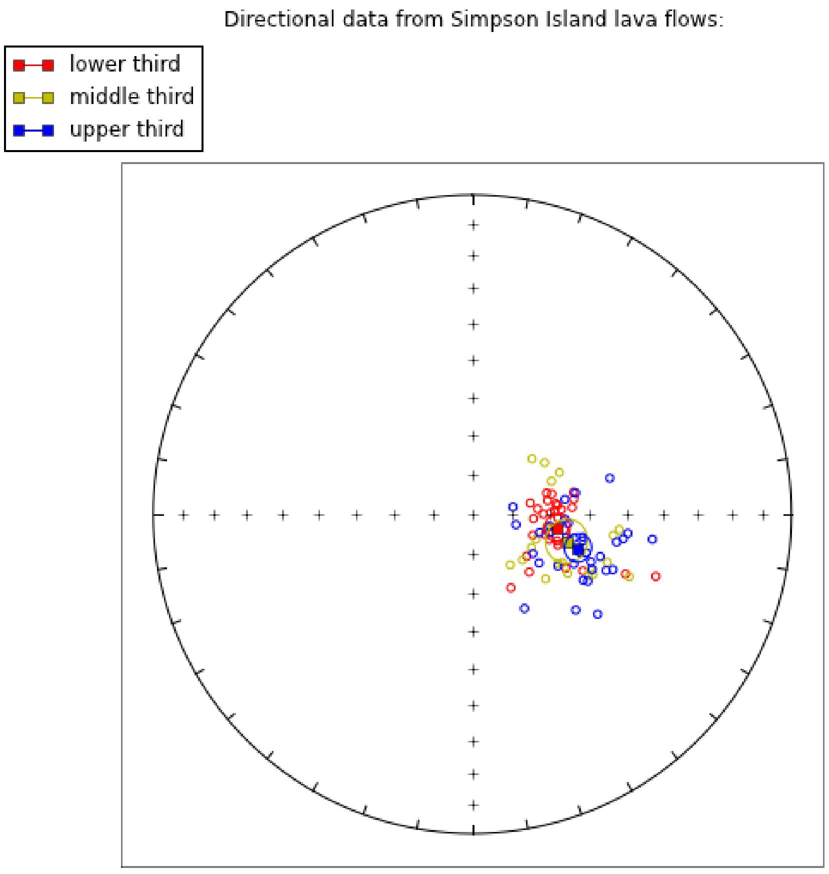
4 Comparison between data from the lower 1/3, middle 1/3 and upper 1/3 of Simpson Island stratigraphy

Let's consider the data by stratigraphically grouping flow data from the lower third (0 to 1041 meters), middle third (1041 to 2083 meters) and the upper third (2083 to 3124 meters) of the stratigraphy. The Fisher means for these groups are calculated with the resulting dictionaries from the fisher_mean pmag.py fuction being printed below and with the data being displayed on equal area plots with the calculated Fisher means also being plotted.

In [5]:
```
SI_Directions=[]
SI_Poles=[]

for n in range(0,84):
    Dec,Inc=Dec_TC[n],Inc_TC[n]
    SI_Directions.append([Dec,Inc,1.])
    Plong,Plat=VGP_long[n],VGP_lat[n]
    SI_Poles.append([Plong,Plat,1.])

SI_LowerThird_Directions=SI_Directions[0:30]
SI_LowerThird_Poles=SI_Poles[0:30]
SI_MiddleThird_Directions=SI_Directions[30:50]
SI_MiddleThird_Poles=SI_Poles[30:50]
SI_UpperThird_Directions=SI_Directions[50:84]
SI_UpperThird_Poles=SI_Poles[50:84]

#calculate and display the Fisher means for each subset of the directions
lower_third_mean=pmag.fisher_mean(SI_LowerThird_Directions)
middle_third_mean=pmag.fisher_mean(SI_MiddleThird_Directions)
upper_third_mean=pmag.fisher_mean(SI_UpperThird_Directions)

print 'The Fisher mean parameters for SI_LowerThird_Directions are: '
print 'Dec = ' + str(lower_third_mean['dec']) + ' Inc = ' + str(lower_third_mean['inc'])
print 'alpha95 = ' + str(lower_third_mean['alpha95']) + ' k= ' + str(lower_third_mean['k'])
print ''
print 'The Fisher mean parameters for SI_MiddleThird_Directions are: '
print 'Dec = ' + str(middle_third_mean['dec']) + ' Inc = ' + str(middle_third_mean['inc'])
print 'alpha95 = ' + str(middle_third_mean['alpha95']) + ' k= ' + str(middle_third_mean['k'])
print ''
print 'The Fisher mean parameters for SI_UpperThird_Directions are: '
print 'Dec = ' + str(upper_third_mean['dec']) + ' Inc = ' + str(upper_third_mean['inc'])
print 'alpha95 = ' + str(upper_third_mean['alpha95']) + ' k= ' + str(upper_third_mean['k'])
```

```
#plot the direction of each flow mean on an equal area plot
fignum = 1
pylab.figure(num=fignum,figsize=(10,10),dpi=160)
pmagplotlib.plotNET(fignum)
IPmag.iplotDI(SI_LowerThird_Directions,color='r')
IPmag.iplotDI(SI_MiddleThird_Directions,color='y')
IPmag.iplotDI(SI_UpperThird_Directions,color='b')
title('Directional data from Simpson Island lava flows:')

#plot the group means and a95 ellipses on same equal area plot
IPmag.iplotDImean(lower_third_mean['dec'],lower_third_mean['inc'],lower_third_mean["alpha95"],
            color='r',marker='s',label='lower third')
IPmag.iplotDImean(middle_third_mean['dec'],middle_third_mean['inc'],middle_third_mean["alpha95"
            color='y',marker='s',label='middle third')
IPmag.iplotDImean(upper_third_mean['dec'],upper_third_mean['inc'],upper_third_mean["alpha95"],
            color='b',marker='s',label='upper third')
```

```
The Fisher mean parameters for SI_LowerThird_Directions are:
Dec = 99.3387210946 Inc = -68.0428839022
alpha95 = 3.358896873 k= 62.2034940674

The Fisher mean parameters for SI_MiddleThird_Directions are:
Dec = 105.906122722 Inc = -64.89830144
alpha95 = 5.47951200533 k= 36.4544611363

The Fisher mean parameters for SI_UpperThird_Directions are:
Dec = 107.624499973 Inc = -61.552568869
alpha95 = 3.51420656821 k= 50.020994936
```



4.1 Test for a common mean between paleomagnetic data from the lower, middle and upper thirds of the stratigraphy

The code below will compare the subsets of data using these two statistical tests:

1. The V_w test of Watson (1983) slightly modified from the implementation of PmagPy (Tauxe, 2013). The test is comprised of calculating the V_w statistic for the two data sets and then determining the critical value of V_w through a Monte Carlo simulation. For the simulation, two Fisher-distributed data sets with a common mean are simulated using their precisions (k_1 and k_2) and the number of points (N_1 and N_2) of the datasets being evaluated. Then the V_w statistic that comes from comparison between the two simulated data sets is calculated. A large number of simulations are done (default is 1000; it can be set as the NumSims parameter of the function) in order to determine the V_w values that would result by chance through sampling distributions with the same direction. The critical value of V_w is at the 95% level of confidence (i.e. if 1000 V_w values are simulated, it is the 950th one). This

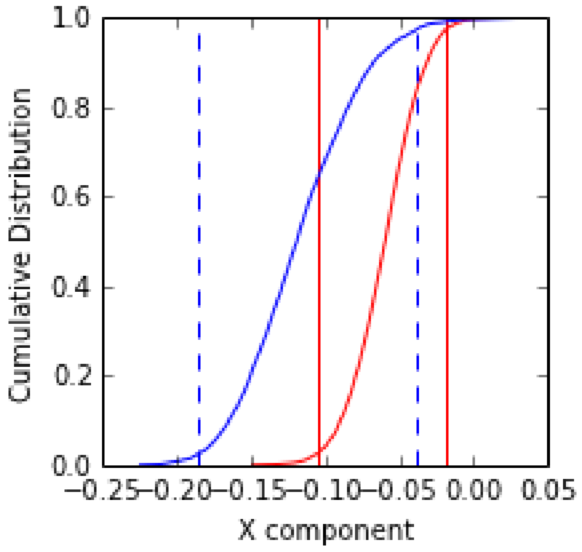
implementation of the test also provides the critical angle between the two sample mean directions and the corresponding McFadden and McElhinny (1990) classification.

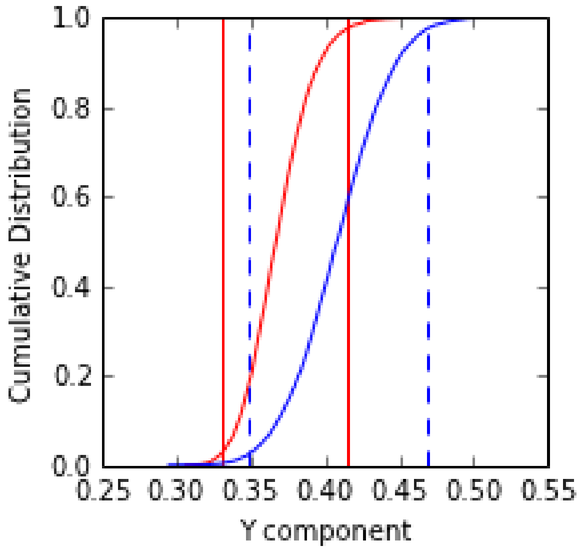
2. The bootstrap fold test of Tauxe (developed by Tauxe et al., 1991 and implemented per Tauxe (2010)). This approach determines the cumulative distributions of the Cartesian coordinates of bootstrapped means and compares the cumulative distributions to see if the confidence intervals overlap. If they do all overlap, then the two means cannot be distinguished at the 95% level of confidence and they pass the bootstrap test for a common mean. Otherwise, if the two sets of directions are distinct in the X, Y or Z component the two means can be distinguished at the 95% confidence level.

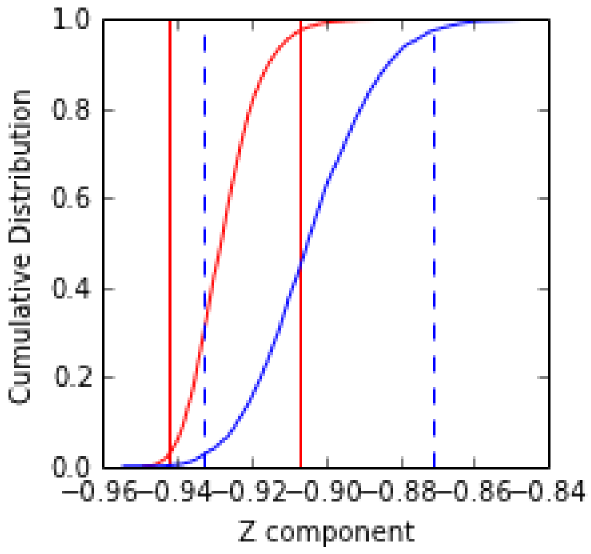
4.1.1 Common mean tests between SI_LowerThird_Directions and SI_MiddleThird_Directions:

```
In [6]: IPmag.iWatsonV(SI_LowerThird_Directions,SI_MiddleThird_Directions)
        IPmag.iBootstrap(SI_LowerThird_Directions,SI_MiddleThird_Directions)
```

```
Results of Watson V test:

Watson's V:           2.6
Critical value of V:  6.2
"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: 4.1
Critical angle for M&M1990:   6.3
The McFadden and McElhinny (1990) classification for
this test is: 'B'

===============

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





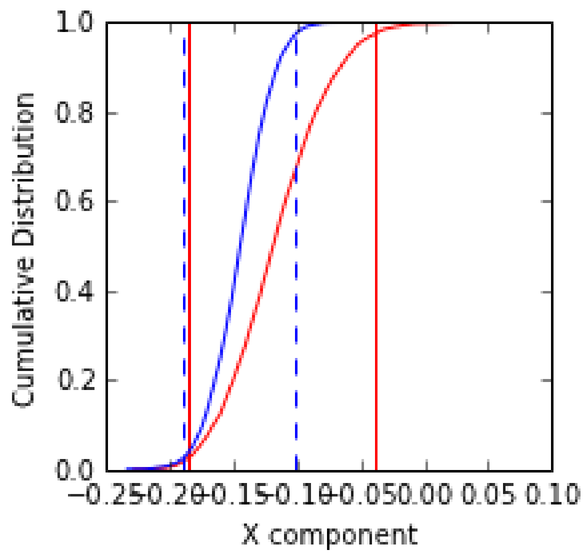


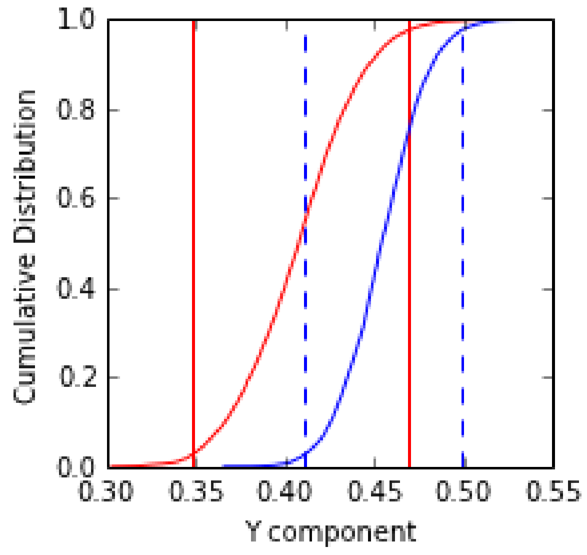
4.1.2 Common mean tests between SI_MiddleThird_Directions and SI_UpperThird_Directions:

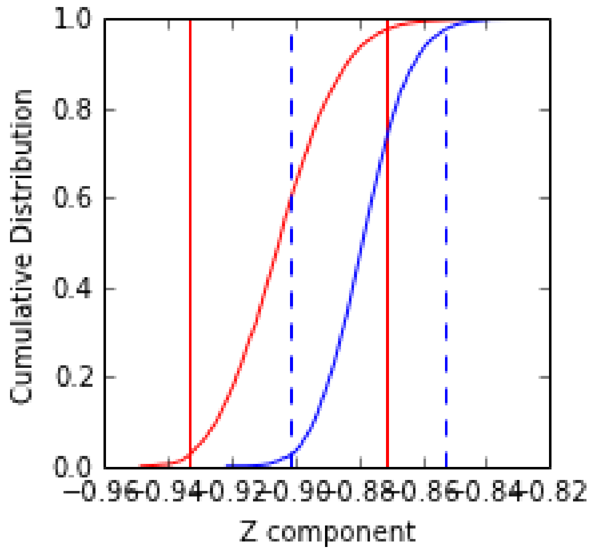
```
In [7]: IPmag.iWatsonV(SI_MiddleThird_Directions,SI_UpperThird_Directions)
        IPmag.iBootstrap(SI_MiddleThird_Directions,SI_UpperThird_Directions)
```

```
Results of Watson V test:

Watson's V:           1.8
Critical value of V:  6.2
"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: 3.4
Critical angle for M&M1990:   6.4
The McFadden and McElhinny (1990) classification for
this test is: 'B'

===============

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



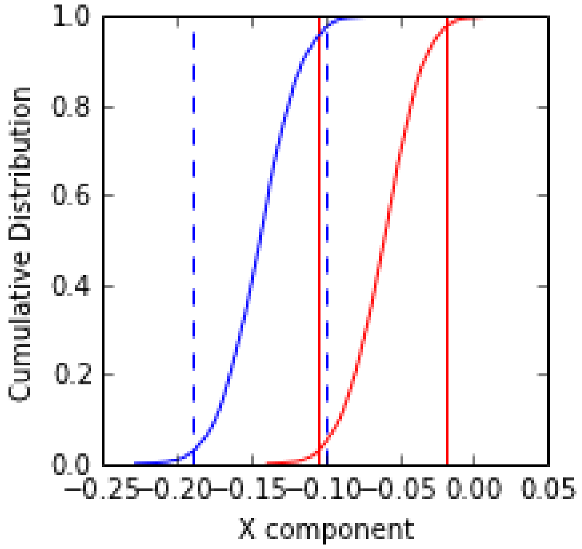




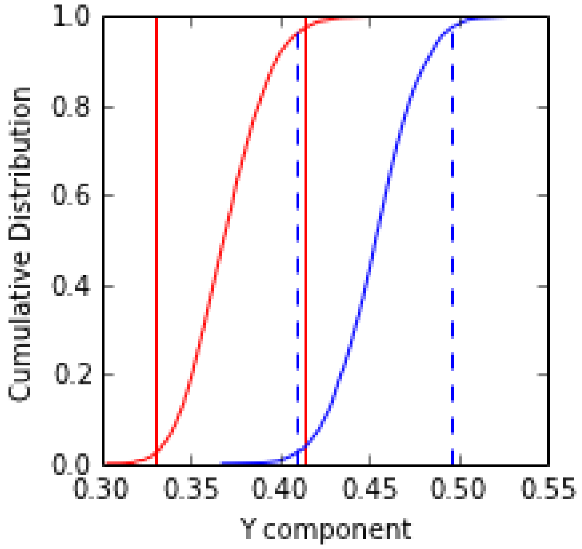
4.1.3 Common mean tests between SI_LowerThird_Directions and SI_UpperThird_Directions:

```
In [8]: IPmag.iWatsonV(SI_LowerThird_Directions,SI_UpperThird_Directions)
        IPmag.iBootstrap(SI_LowerThird_Directions,SI_UpperThird_Directions)
```

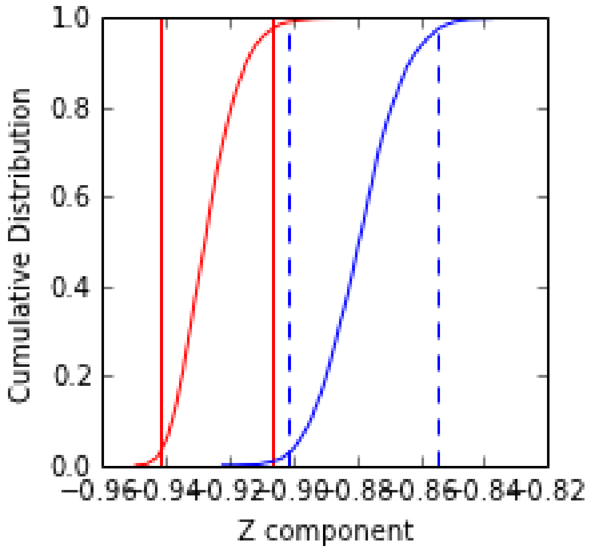
```
Results of Watson V test:

Watson's V:           14.5
Critical value of V:  6.0
"Fail": Since V is greater than Vcrit, the two means can
be distinguished at the 95% confidence level.

M&M1990 classification:

Angle between data set means: 7.4
Critical angle for M&M1990:   4.8


===============

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



1.0
0.8
0.6
0.4
0.2
0.0
Cumulative Distribution
−0.25
−0.20
−0.15
−0.10
−0.05
0.00
0.05
X component



1.0
0.8
0.6
0.4
0.2
0.0
Cumulative Distribution
0.30
0.35
0.40
0.45
0.50
0.55
Y component



4.1.4 Common mean test on directions results summary

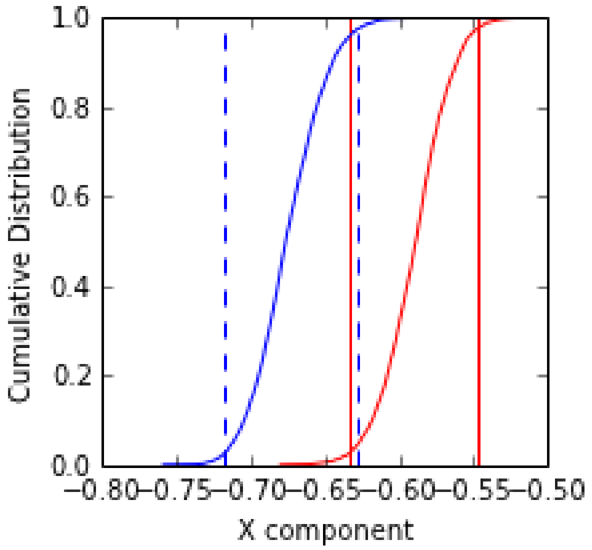
The above results from common mean statistical tests show that the directions from the lower third cannot be distinguished as being obtained from a distribution distinct from that of the middle third nor can the directions from the upper third be distinguished from those of the middle third. In contrast, directions from the lower third of the stratigraphy can be distinguished from those of upper third at the 95% confidence level.

4.1.5 Common mean tests between SI_LowerThird_Poles and SI_UpperThird_Poles:

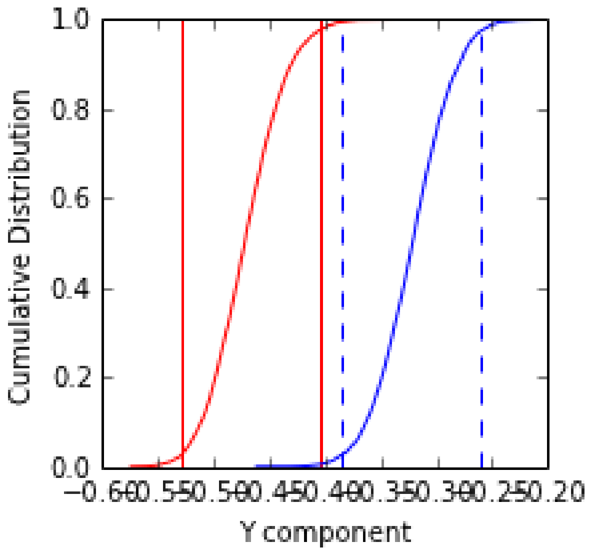
For completeness, let's also conduct tests for a common mean between the virtual geomagnetic poles (VGPs) from the lower third and upper third of the stratigraphy in addition to the tests on the site mean directions in directional space (declination, inclination).

```
In [9]: IPmag.iWatsonV(SI_LowerThird_Poles,SI_UpperThird_Poles)
        IPmag.iBootstrap(SI_LowerThird_Poles,SI_UpperThird_Poles)
```

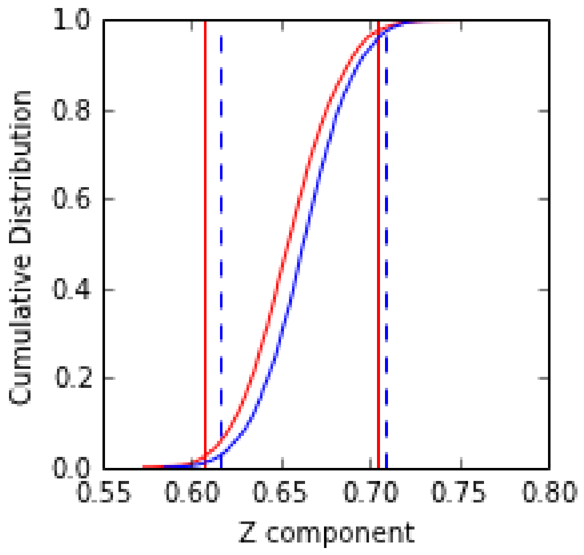
```
Results of Watson V test:

Watson's V:           13.6
Critical value of V:  6.1
"Fail": Since V is greater than Vcrit, the two means can
be distinguished at the 95% confidence level.

M&M1990 classification:

Angle between data set means: 10.0
Critical angle for M&M1990:   6.7


===============


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



1.0
0.8
0.6
0.4
0.2
0.0
Cumulative Distribution
−0.80
−0.75
−0.70
−0.65
−0.60
−0.55
−0.50
X component



1.0
0.8
0.6
0.4
0.2
0.0
Cumulative Distribution
−0.60
−0.55
−0.50
−0.45
−0.40
−0.35
−0.30
−0.25
−0.20
Y component



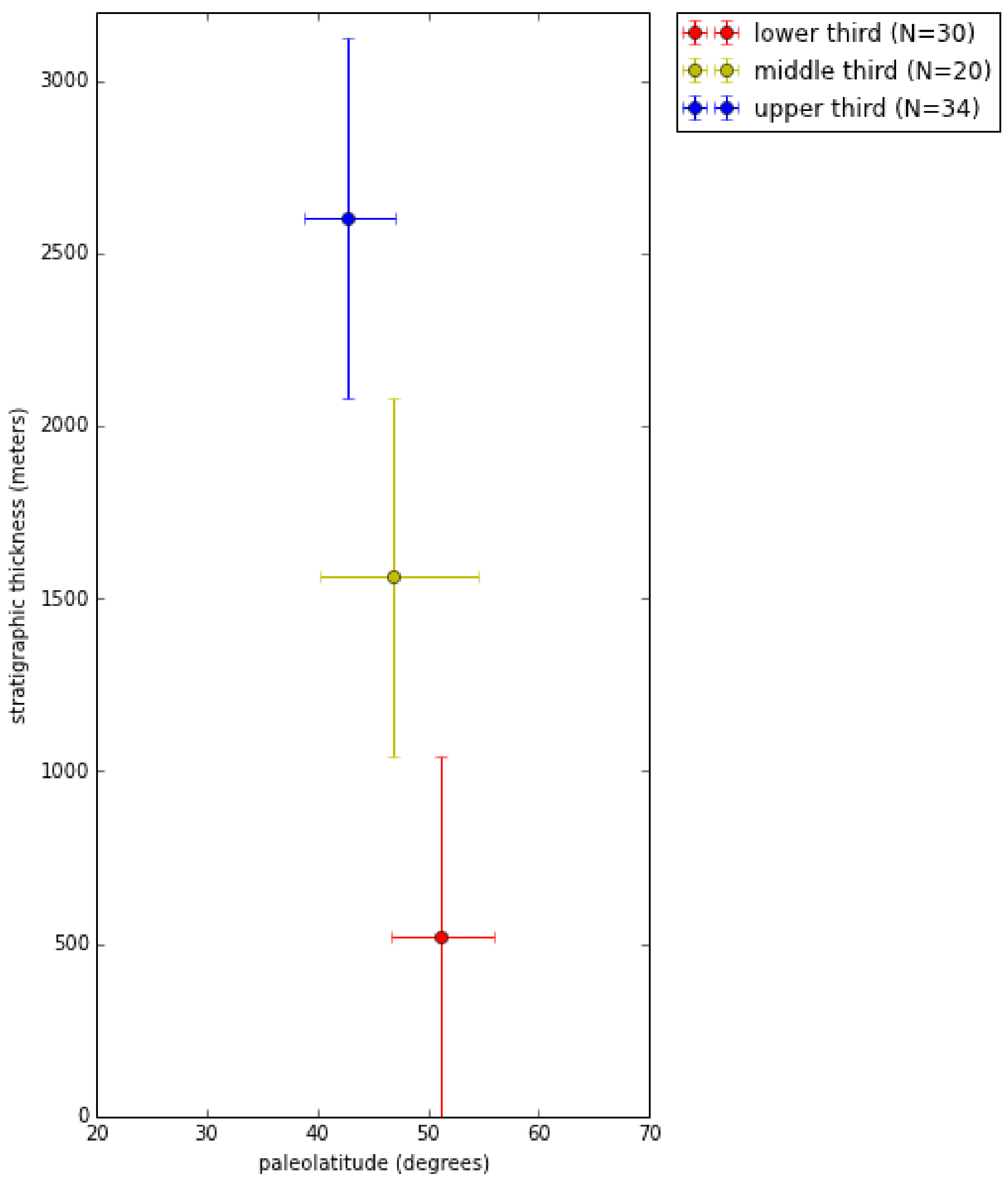
As when the site declinations and inclinations are considered, the virtual geomagnetic poles from the lower third and upper third of the stratigraphy can be distibuguished at the 95% confidence level according to these tests.

4.2 Plotting the data stratigraphically

Using the flow means calculated for each portion of the stratigraphy, the paleolatitude and 95% confidence bounds on paleolatitude can be calculated. The stratigraphic bounds (y-axis error bars) and paleolatitude bounds (x-axis error bars) are displayed for points plotted in the middle of the stratigraphic bin and at the mean paleolatitude.

```
In [10]: SI_LowerThird_plat=numpy.abs(IPmag.lat_from_i(lower_third_mean['inc']))
         SI_LowerThird_plat_max=numpy.abs(IPmag.lat_from_i(lower_third_mean['inc']-
                                                   lower_third_mean['alpha95']))
         SI_LowerThird_plat_min=numpy.abs(IPmag.lat_from_i(lower_third_mean['inc']+
                                                   lower_third_mean['alpha95']))

         SI_MiddleThird_plat=numpy.abs(IPmag.lat_from_i(middle_third_mean['inc']))
         SI_MiddleThird_plat_max=numpy.abs(IPmag.lat_from_i(middle_third_mean['inc']-
                                                    middle_third_mean['alpha95']))
         SI_MiddleThird_plat_min=numpy.abs(IPmag.lat_from_i(middle_third_mean['inc']+
                                                    middle_third_mean['alpha95']))

         SI_UpperThird_plat=numpy.abs(IPmag.lat_from_i(upper_third_mean['inc']))
         SI_UpperThird_plat_max=numpy.abs(IPmag.lat_from_i(upper_third_mean['inc']-
                                                   upper_third_mean['alpha95']))
         SI_UpperThird_plat_min=numpy.abs(IPmag.lat_from_i(upper_third_mean['inc']+
                                                   upper_third_mean['alpha95']))

         SI_LowerThird_meanstrat=521
         SI_MiddleThird_meanstrat=521+1041
         SI_UpperThird_meanstrat=521+1041+1041

         figure()
```

```
errorbar(SI_LowerThird_plat, SI_LowerThird_meanstrat, yerr=[[521],[521]],
         xerr=[[SI_LowerThird_plat-SI_LowerThird_plat_min],
               [SI_LowerThird_plat_max-SI_LowerThird_plat]],
         fmt='ro',label="lower third (N=30)")
errorbar(SI_MiddleThird_plat, SI_MiddleThird_meanstrat, yerr=[[521],[521]],
         xerr=[[SI_MiddleThird_plat-SI_MiddleThird_plat_min],
               [SI_MiddleThird_plat_max-SI_MiddleThird_plat]],
         fmt='yo',label="middle third (N=20)")
errorbar(SI_UpperThird_plat, SI_UpperThird_meanstrat, yerr=[[521],[521]],
         xerr=[[SI_UpperThird_plat-SI_UpperThird_plat_min],
               [SI_UpperThird_plat_max-SI_UpperThird_plat]],
         fmt='bo',label="upper third (N=34)")
xlabel('paleolatitude (degrees)')
ylabel('stratigraphic thickness (meters)')
legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(5,10)
pylab.xlim([20,70])
pylab.ylim([0,3200])
plt.show()
```



5 Comparing larger and smaller stratigraphic subsets of the Simpson Island data

5.1 Comparing data from the lowermost 500 meters to data from the uppermost 500 meters

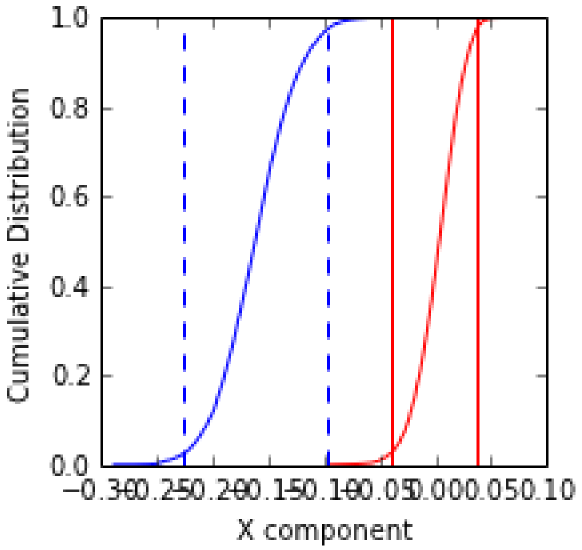
The analysis above demonstrates that the directions from the lower third of the stratigraphy (0 to 1041 meters; 30 flows) and the upper third of the stratigraphy (2083 to 3124 meters; 34 flows) are statistically distinct and thus indicate statistically significant motion to lower latitudes. While it is preferable to have a high total number of flows for such comparisons in order to average out secular variation, let's conduct this same analysis, but for a more restricted range from the bottom 500 meters (17 flows) and top 500 meters (17 flows) of the stratigraphy.

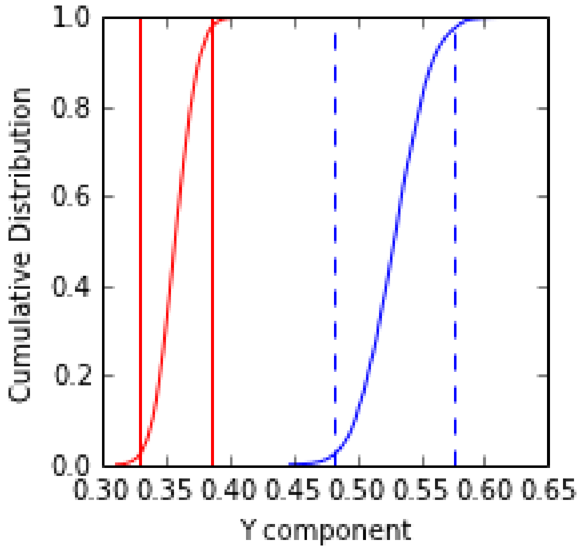
```
In [11]: SI_Lower500m_Directions=SI_Directions[0:17]
         SI_Lower500m_Poles=SI_Poles[0:17]

         SI_Upper500m_Directions=SI_Directions[67:84]
         SI_Upper500m_Poles=SI_Poles[67:84]

         Lower500m_mean=pmag.fisher_mean(SI_Lower500m_Directions)
         Upper500m_mean=pmag.fisher_mean(SI_Upper500m_Directions)

         print 'The Fisher mean parameters for SI_Lower500m_Directions are: '
         print 'Dec = ' + str(Lower500m_mean['dec']) + ' Inc = ' + str(Lower500m_mean['inc'])
         print 'alpha95 = ' + str(Lower500m_mean['alpha95']) + ' k= ' + str(Lower500m_mean['k'])
         print ''
         print 'The Fisher mean parameters for SI_Upper500m_Directions are: '
         print 'Dec = ' + str(Upper500m_mean['dec']) + ' Inc = ' + str(Upper500m_mean['inc'])
         print 'alpha95 = ' + str(Upper500m_mean['alpha95']) + ' k= ' + str(Upper500m_mean['k'])
```

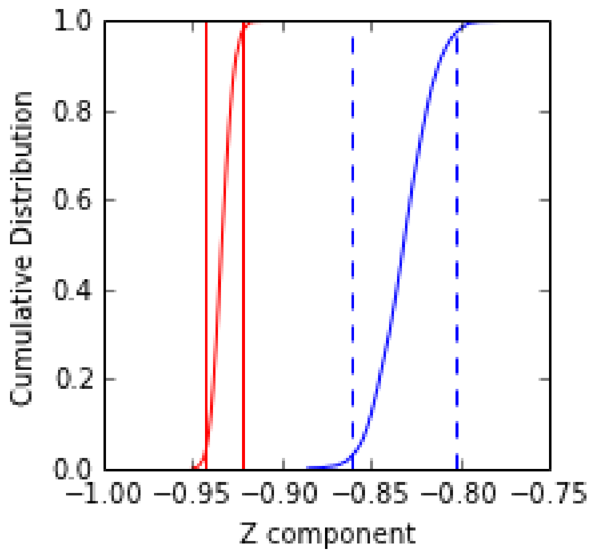
```
The Fisher mean parameters for SI_Lower500m_Directions are:
Dec = 89.5592116942 Inc = -69.0370984662
alpha95 = 2.72130375778 k= 172.790689067

The Fisher mean parameters for SI_Upper500m_Directions are:
Dec = 106.91559555 Inc = -56.3717140613
alpha95 = 4.61609695128 k= 60.6867572134
```

```
In [12]: IPmag.iWatsonV(SI_Lower500m_Directions,SI_Upper500m_Directions)
         IPmag.iBootstrap(SI_Lower500m_Directions,SI_Upper500m_Directions)
```

```
Results of Watson V test:

Watson's V:           50.4
Critical value of V:  6.3
"Fail": Since V is greater than Vcrit, the two means can
be distinguished at the 95% confidence level.

M&M1990 classification:

Angle between data set means: 14.8
Critical angle for M&M1990:   5.2


===============

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


Cumulative Distribution
X component
1.0
0.8
0.6
0.4
0.2
0.0
−0.30
−0.25
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10


Cumulative Distribution
Y component
1.0
0.8
0.6
0.4
0.2
0.0
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65



The results of this comparision between the lower 500 meters and upper 500 meters show that they are distinct populations at the 95% confidence level. The difference between the populations is more dramatic in both the Watson V and bootstrap test from the tests on the lower third and upper third subsets that were conducted above. This increase in difference with tighter stratigraphic groups from lower and higher in the sequence is the expected result if the sequence is a record of progressive paleogeographic change.

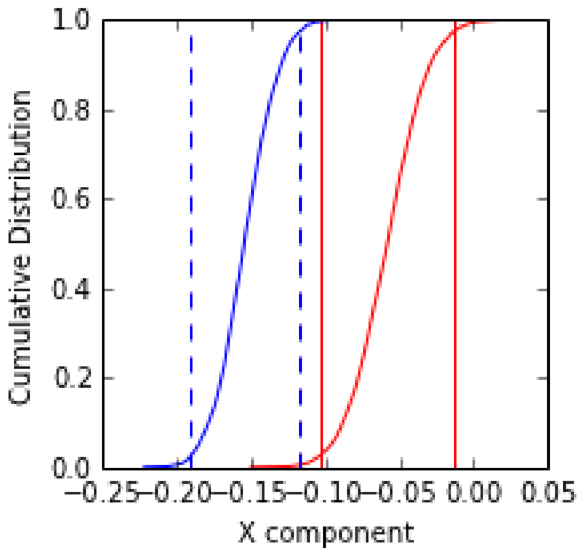
5.2 Comparison between data from the lower 1/2 and upper 1/2 of Simpson Island stratigraphy

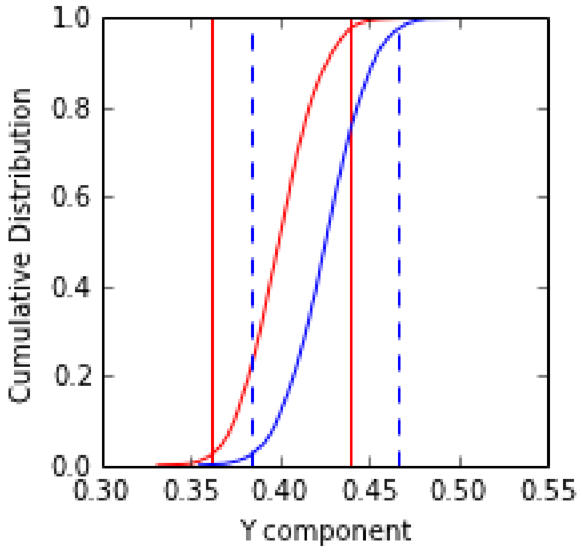
```
In [13]: SI_Directions=[]
         SI_Poles=[]

         for n in range(0,84):
             Dec,Inc=Dec_TC[n],Inc_TC[n]
             SI_Directions.append([Dec,Inc,1.])
             Plong,Plat=VGP_long[n],VGP_lat[n]
             SI_Poles.append([Plong,Plat,1.])

         SI_LowerHalf_Directions=SI_Directions[0:41]
         SI_LowerHalf_Poles=SI_Poles[0:41]
         SI_UpperHalf_Directions=SI_Directions[41:84]
         SI_UpperHalf_Poles=SI_Poles[41:84]

         LowerHalf_mean=pmag.fisher_mean(SI_LowerHalf_Directions)
         UpperHalf_mean=pmag.fisher_mean(SI_UpperHalf_Directions)

         print 'The fisher mean parameters for SI_LowerHalf_Directions are: '
         print 'Dec = ' + str(LowerHalf_mean['dec']) + ' Inc = ' + str(LowerHalf_mean['inc'])
         print 'alpha95 = ' + str(LowerHalf_mean['alpha95']) + ' k= ' + str(LowerHalf_mean['k'])
         print ''
         print 'The fisher mean parameters for SI_UpperHalf_Directions are: '
         print 'Dec = ' + str(UpperHalf_mean['dec']) + ' Inc = ' + str(UpperHalf_mean['inc'])
         print 'alpha95 = ' + str(UpperHalf_mean['alpha95']) + ' k= ' + str(UpperHalf_mean['k'])
```

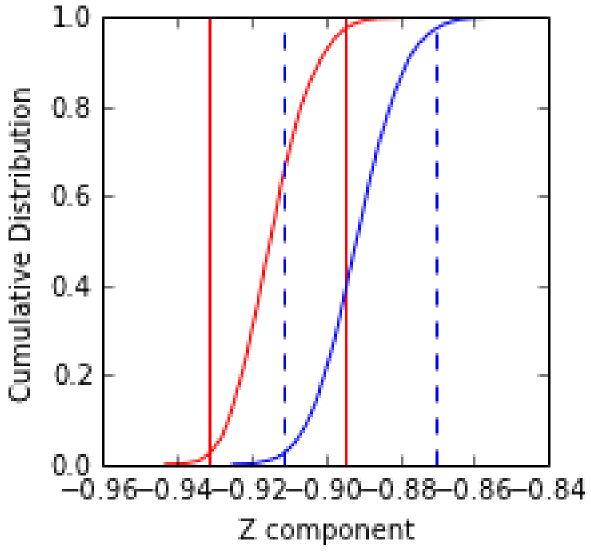
```
The fisher mean parameters for SI_LowerHalf_Directions are:
Dec = 98.3683994117 Inc = -66.1920879279
alpha95 = 3.31197629718 k= 46.4016899603

The fisher mean parameters for SI_UpperHalf_Directions are:
Dec = 109.926576191 Inc = -63.0929234842
alpha95 = 3.05246279053 k= 51.8737553101
```

```
In [14]: IPmag.iWatsonV(SI_LowerHalf_Directions,SI_UpperHalf_Directions)
         IPmag.iBootstrap(SI_LowerHalf_Directions,SI_UpperHalf_Directions)
```

```
Results of Watson V test:

Watson's V:           10.4
Critical value of V:  6.1
"Fail": Since V is greater than Vcrit, the two means can
be distinguished at the 95% confidence level.

M&M1990 classification:

Angle between data set means: 5.8
Critical angle for M&M1990:   4.5


===============

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







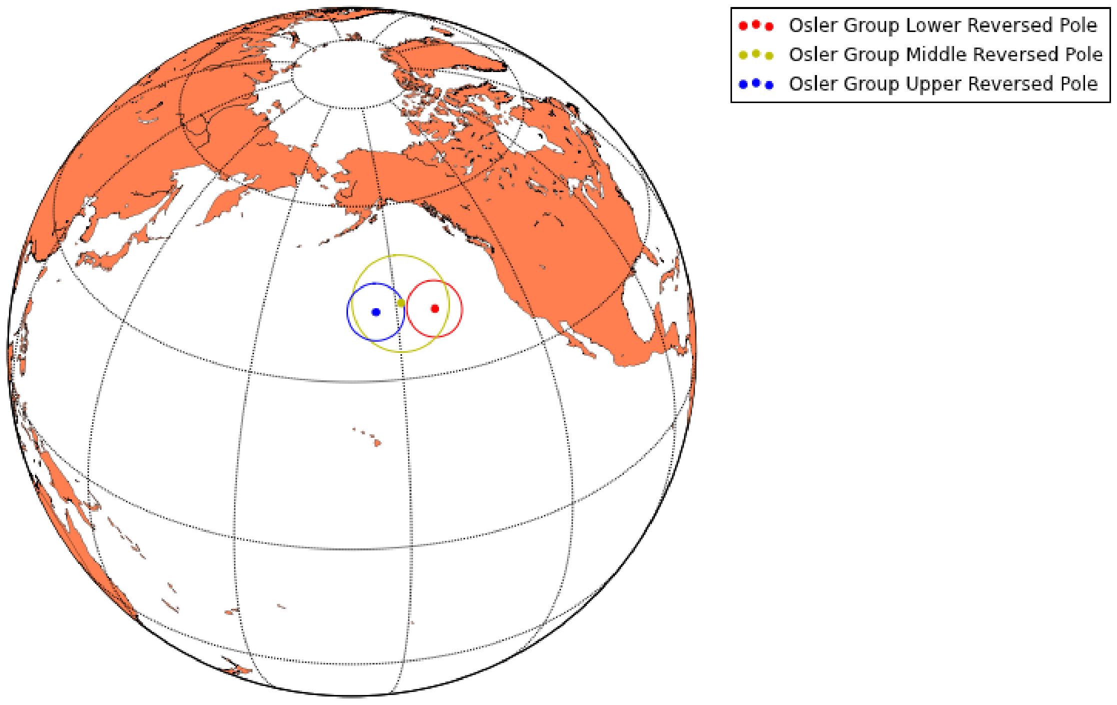
6 Calculating and plotting mean pole positions

The code below calculates the Fisher mean parameters for poles of the stratigraphically grouped virtual geomagnetic poles (VGPs) and then plots them on a “view from space” globe. Note that while where the textual output says “dec” it actually referes to pole longitude and where it says “inc” it actually refers to pole latitude since the pmag.fisher_mean function is being conducted on VGPs.

```
In [15]: LowerThird_MeanPole=pmag.fisher_mean(SI_LowerThird_Poles)
         MiddleThird_MeanPole=pmag.fisher_mean(SI_MiddleThird_Poles)
         UpperThird_MeanPole=pmag.fisher_mean(SI_UpperThird_Poles)
```

```
print 'The Fisher mean parameters for the LowerThird mean pole are: '
print 'Plong = ' + str(LowerThird_MeanPole['dec']) + ' Plat = ' + str(LowerThird_MeanPole['inc
print 'A95 = ' + str(LowerThird_MeanPole['alpha95']) + ' k= ' + str(LowerThird_MeanPole['k'])
print ''
print 'The Fisher mean parameters for the MiddleThird mean pole are: '
print 'Plong = ' + str(MiddleThird_MeanPole['dec']) + ' Plat = ' + str(MiddleThird_MeanPole['i
print 'A95 = ' + str(MiddleThird_MeanPole['alpha95']) + ' k= ' + str(MiddleThird_MeanPole['k']
print ''
print 'The Fisher mean parameters for the UpperThird mean pole are: '
print 'Plong = ' + str(UpperThird_MeanPole['dec']) + ' Plat = ' + str(UpperThird_MeanPole['inc
print 'A95 = ' + str(UpperThird_MeanPole['alpha95']) + ' k= ' + str(UpperThird_MeanPole['k'])

#setup the figure and map
figure(figsize=(8, 8))
m = Basemap(projection='ortho',lat_0=35,lon_0=200,resolution='l')
# draw coastlines, country boundaries, fill continents.
m.drawcoastlines(linewidth=0.25)
#map.drawcountries(linewidth=0.25)
m.fillcontinents(color='coral',lake_color='white')
m.drawmapboundary(fill_color='white')
m.drawmeridians(np.arange(0,360,30))
m.drawparallels(np.arange(-90,90,30))

#plot the mean poles
IPmag.poleplot(m,LowerThird_MeanPole['dec'],LowerThird_MeanPole['inc'],
               LowerThird_MeanPole['alpha95'],color='r',label='Osler Group Lower Reversed Pole

IPmag.poleplot(m,MiddleThird_MeanPole['dec'],MiddleThird_MeanPole['inc'],
               MiddleThird_MeanPole['alpha95'],color='y',label='Osler Group Middle Reversed Po

IPmag.poleplot(m,UpperThird_MeanPole['dec'],UpperThird_MeanPole['inc'],
               UpperThird_MeanPole['alpha95'],color='b',label='Osler Group Upper Reversed Pole

#show the plot
legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```

```
The Fisher mean parameters for the LowerThird mean pole are:
Plong = 218.637832609 Plat = 40.937318541
A95 = 4.76006232039 k= 31.4671112908

The Fisher mean parameters for the MiddleThird mean pole are:
Plong = 211.261736125 Plat = 42.7378078852
A95 = 8.1783359746 k= 16.9030328287

The Fisher mean parameters for the UpperThird mean pole are:
Plong = 205.410227513 Plat = 41.5645761829
A95 = 4.80394668574 k= 27.2260167637
```


Osler Group Lower Reversed Pole
Osler Group Middle Reversed Pole
Osler Group Upper Reversed Pole

Note that in the print out for the mean pole calculations above 'dec' is really the longitude of the pole (Plong) and 'inc' is actually the latitude of the pole (Plat).

Pole ID	Plong	Plat	A95	N
SI_Lower	218.6	40.9	4.8	30
SI_Middle	211.3	42.7	8.2	20
SI_Upper	205.4	41.6	4.8	34

7 Comparision between data from Halls (1974) study and Simpson Island VGPs

Halls (1974) developed paleomagnetic data from the Osler Volcanic Group in the Nipogon Straits region (bibtex citation below; also see http://earthref.org/MAGIC/9518). For the purposes of the previously contribued MagIC database entries, site locations were determined using georeferenced versions of the maps provided in Halls (1974). These data allow for the site means of Halls (1974) to be used to calculate VGPs for purposes of directional statistical tests and for combining into a mean paleopole. The Halls (1974) data are predominantly from flows with reversed magnetization below an angular unconformity on Puff Island that seperates the reversedly magnetized flows from younger flows of normal polarity (only a few of which are preserved before the sequence is submerged beneath Lake Superior). The analysis below compares paleomagnetic data from this study with the reversed flows from the Halls (1974) data. The result from this analysis is that the Halls (1974) data is statistically distinct from the lower third of the Simpson Island stratigraphy, but is statitically indistinguishable from the upper third of the Simpson Island stratigraphy. This result fits with stratigraphic considerations that place the sites studied by Halls towards the top of the stratigraphy studied herein at Simpson Island.

Bibtex citation info for Halls (1974) @articleHalls1974a, Author = Halls, H.C., Journal = Canadian

Journal of Earth Science, Pages = 1200-1207, Title = A paleomagnetic reversal in the Osler Volcanic Group, northern Lake Superior, Volume = 11, Year = 1974

The code below imports the directional data from Halls (1974) and than uses this data to calculate virtual geomagnetic poles for each site location.

```
In [16]: Halls1974_Osler_Data=pandas.read_csv('../2014_Osler_Data/Halls1974_data.csv',sep=',')
         Halls1974_Osler_Data
```

```
Out[16]:          Study  Site_ID  AFstep_Oe  dec_tc  inc_tc  kappa  n  site_lat  \
         0   Halls1974a        1        200   289.8    43.6    517  5     48.63
         1   Halls1974a        2        150   285.7    42.0    243  5     48.64
         2   Halls1974a        3        100   307.4    37.9   2485  5     48.64
         3   Halls1974a        4        150   302.8    29.9    331  5     48.66
         4   Halls1974a        5        300   294.7    42.5    130  5     48.66
         5   Halls1974a        6        100   106.1   -44.2     70  5     48.66
         6   Halls1974a        7        250   109.8   -62.4    351  6     48.68
         7   Halls1974a        8        400   140.7   -64.8    679  5     48.68
         8   Halls1974a        9        150   110.0   -56.6    321  3     48.68
         9   Halls1974a       10        150   128.2   -42.0     59  5     48.66
         10  Halls1974a       11        400    88.3   -59.1    456  3     48.66
         11  Halls1974a       12        250   118.6   -48.9    240  5     48.66
         12  Halls1974a       13        250   128.1   -38.0    886  3     48.65
         13  Halls1974a       14        200   108.2   -55.8    121  6     48.65
         14  Halls1974a       15        300   118.5   -46.0    402  6     48.65
         15  Halls1974a       16        150   106.0   -54.2    125  6     48.64
         16  Halls1974a       17        300    93.7   -54.1    362  4     48.63
         17  Halls1974a       18        200   116.0   -53.0    224  4     48.61
         18  Halls1974a       19        200   131.7   -64.9    871  4     48.61
         19  Halls1974a       20        250   119.9   -55.8    108  4     48.59
         20  Halls1974a       21        300    94.3   -65.5     84  5     48.64
         21  Halls1974a       22        200    91.8   -58.2     94  5     48.64
         22  Halls1974a       23        300   114.3   -64.0     69  4     48.62
         23  Halls1974a       24        400   116.8   -40.4     77  3     48.68
         24  Halls1974a       25        150   126.3   -58.7    166  3     48.69
         25  Halls1974a       26        300   112.0   -75.1   3634  3     48.71
         26  Halls1974a       27        400   118.6   -70.3    729  3     48.67
         27  Halls1974a       28        150   132.6   -54.1   1039  3     48.68
         28  Halls1974a       29        250   129.1   -54.5   1149  3     48.68
         29  Halls1974a       30        300    80.5   -80.4    267  5     48.59

             site_long
         0      -88.11
         1      -88.09
         2      -88.07
         3      -88.04
         4      -88.04
         5      -88.05
         6      -88.02
         7      -88.03
         8      -88.06
         9      -88.07
         10     -88.07
         11     -88.08
         12     -88.09
         13     -88.08
```

```
14     -88.10
15     -88.10
16     -88.11
17     -88.16
18     -88.17
19     -88.19
20     -88.12
21     -88.12
22     -88.16
23     -88.06
24     -88.05
25     -88.07
26     -88.10
27     -88.10
28     -88.10
29     -88.22

[30 rows x 9 columns]
```

Calculate VGPs using the Halls (1974) data:

```
In [17]: IPmag.VGP_calc(Halls1974_Osler_Data)
         Halls1974_Osler_Data.head()
```

```
Out[17]:         Study  Site_ID  AFstep_Oe  dec_tc  inc_tc  kappa  n  site_lat  \
         0  Halls1974a        1        200   289.8    43.6    517  5     48.63
         1  Halls1974a        2        150   285.7    42.0    243  5     48.64
         2  Halls1974a        3        100   307.4    37.9   2485  5     48.64
         3  Halls1974a        4        150   302.8    29.9    331  5     48.66
         4  Halls1974a        5        300   294.7    42.5    130  5     48.66

            site_long  paleolatitude   pole_lat   pole_long  pole_lat_rev  \
         0     -88.11      25.461154  31.651781  185.568167    -31.651781
         1     -88.09      24.237370  28.110360  187.509415    -28.110360
         2     -88.07      21.267947  40.260801  167.888919    -40.260801
         3     -88.04      16.040618  33.459149  167.496979    -33.459149
         4     -88.04      24.615622  34.309281  181.259648    -34.309281

            pole_long_rev
         0       5.568167
         1       7.509415
         2     347.888919
         3     347.496979
         4       1.259648

         [5 rows x 14 columns]
```

Split the dataframe into one for the normally magnetized sites and one for the reversed polarity sites

```
In [18]: Halls1974_Osler_Data_N = Halls1974_Osler_Data[:5]
         Halls1974_Osler_Data_R = Halls1974_Osler_Data[5:]
         Halls1974_Osler_Data_R=Halls1974_Osler_Data_R.reset_index()
```

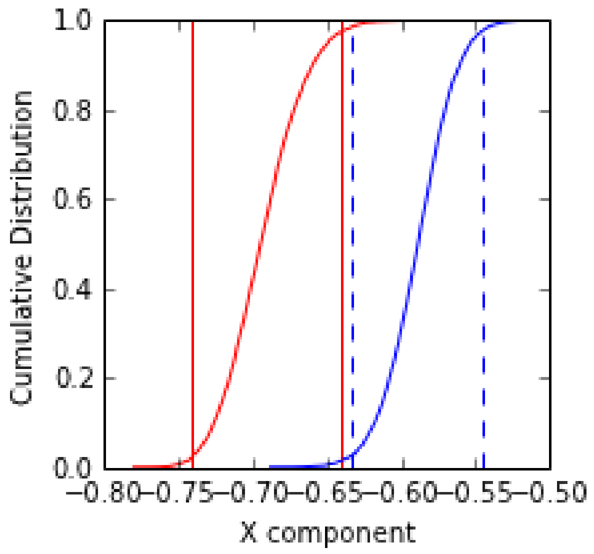
Develop a list of unit vectors of the reversed Halls (1974) VGPs in order to conduct common mean tests with the Simpson Island data using pmag.py functions.

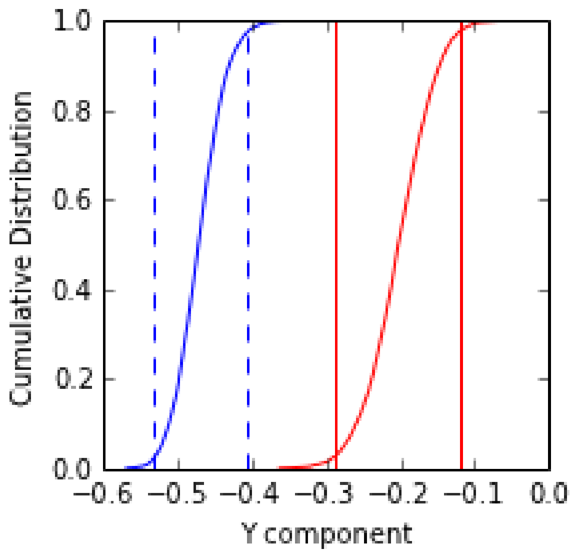
```
In [19]: Halls1974_Osler_R_VGPs=[]
         Halls1974_Osler_R_Plong=[]
         Halls1974_Osler_R_Plat=[]
         for n in range(0,len(Halls1974_Osler_Data_R)):
             Plong,Plat=Halls1974_Osler_Data_R['pole_long_rev'][n],Halls1974_Osler_Data_R['pole_lat_rev
             Halls1974_Osler_R_Plong.append(Plong)
             Halls1974_Osler_R_Plat.append(Plat)
             Halls1974_Osler_R_VGPs.append([Plong,Plat,1.])
```

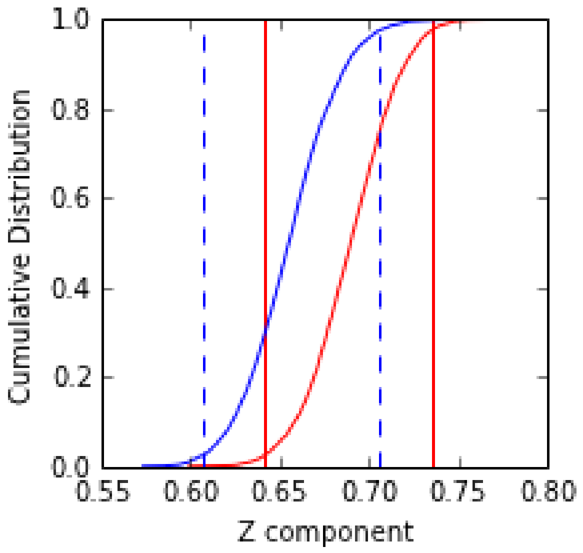
7.0.1 Common mean tests between Halls (1974) data and the data from the lower third of the Simpson Island stratigraphy

```
In [20]: IPmag.iWatsonV(Halls1974_Osler_R_VGPs,SI_LowerThird_Poles)
         IPmag.iBootstrap(Halls1974_Osler_R_VGPs,SI_LowerThird_Poles)
```

```
Results of Watson V test:

Watson's V:           31.0
Critical value of V:  6.3
"Fail": Since V is greater than Vcrit, the two means can
be distinguished at the 95% confidence level.

M&M1990 classification:

Angle between data set means: 16.7
Critical angle for M&M1990:   7.5


===============

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







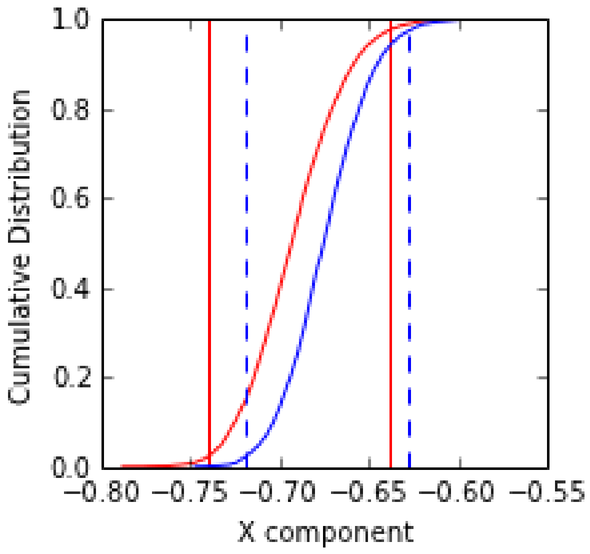
A comparision between VGPs calculated from Halls, 1974 data and data from the lower third of the Simpson Island stratigraphy fails Watson's V and bootstrap tests for a common mean indicating that the populations of directions do not share a common mean at the 95% confidence level.

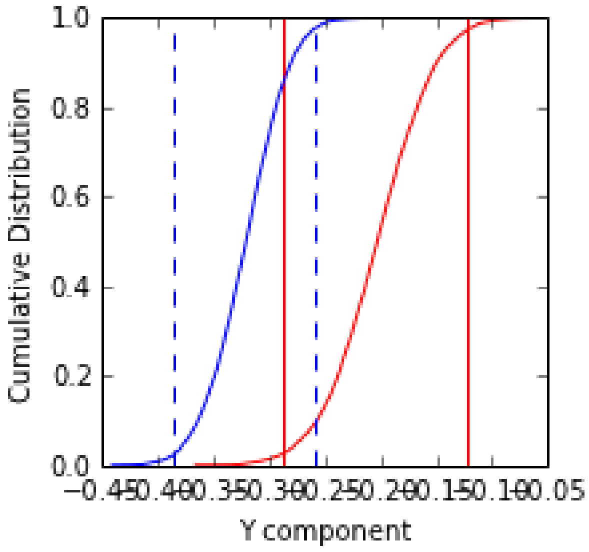
7.0.2 Common mean tests between Halls (1974) data and the data from the upper third of the Simpson Island stratigraphy

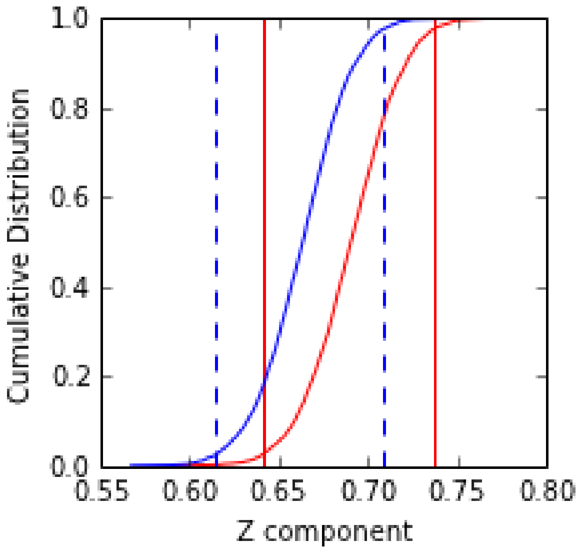
```
In [21]: IPmag.iWatsonV(Halls1974_Osler_R_VGPs,SI_UpperThird_Poles)
         IPmag.iBootstrap(Halls1974_Osler_R_VGPs,SI_UpperThird_Poles)
```

```
Results of Watson V test:
```

```
Watson's V:           5.4
Critical value of V:  6.5
"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: 7.0
Critical angle for M&M1990:   7.7
The McFadden and McElhinny (1990) classification for
this test is: 'B'

===============

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







A comparision between VGPs calculated from Halls, 1974 data (see http://earthref.org/MAGIC/9518) and data from the upper third of the Simpson Island stratigraphy passes Watson's V and bootstrap tests for a common mean.

7.0.3 Combining the data from the upper third of the Simpson Island stratigraphy and the Halls (1974) reversed polarity data into a single pole for the upper portion of the reversed Osler Group stratigraphy

```
In [22]: upperR_Osler_VGPs=Halls1974_Osler_R_VGPs+SI_UpperThird_Poles

         upperR_Osler_meanpole=pmag.fisher_mean(upperR_Osler_VGPs)
```

```
print 'The fisher mean parameters for the pole calculated from upper reversed'
print 'Osler Group VGPs (SI_UpperThird_Poles+Halls, 1974 reversed data) are: '
print 'Plong = ' + str(upperR_Osler_meanpole['dec']) + ' Plat = ' + str(upperR_Osler_meanpole[
print 'A95 = ' + str(upperR_Osler_meanpole['alpha95']) + ' k= ' + str(upperR_Osler_meanpole['k
```

```
The fisher mean parameters for the pole calculated from upper reversed
Osler Group VGPs (SI_UpperThird_Poles+Halls, 1974 reversed data) are:
Plong = 201.646829641 Plat = 42.5371761039
A95 = 3.72282691107 k= 25.6773898601 N= 59
```

Pole ID	Plong	Plat	A95	N	approximate age
upper Osler Group reversed pole	201.6	42.5	3.7	59	1105±2 Ma
(this work and Halls, 1974)					(Davis and Green, 1997)

8 Paleogeographic work outside of this IPython notebook using the software package GPlates.

For details on working with paleomagnetic data in GPlates check out this tutorial.

The steps necessary to develop the reconstruction using these Osler Volcanic Group poles are detailed below.

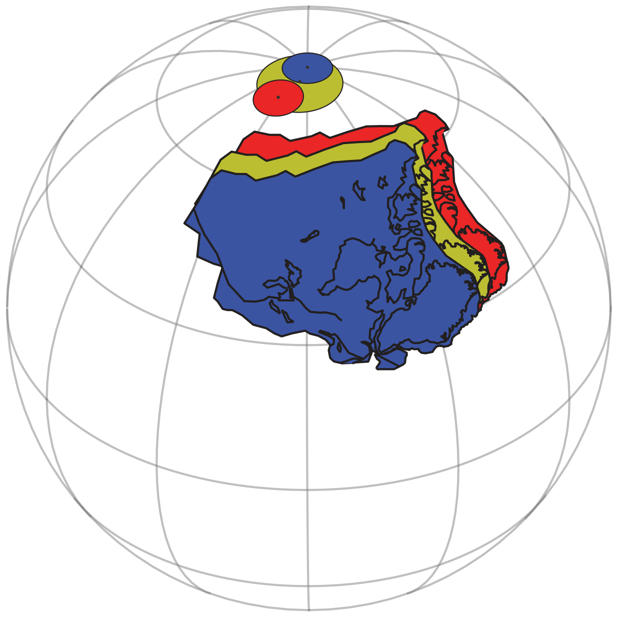
(1) Develop .rot file that rotates Laurentia such that the Osler poles are at geographic north. In the text for the .rot file below the Osler_lowerthird pole is assigned to 1110 Ma, the Osler_middlethird pole is assigned to 1107.5 and the Osler_upperthird pole is assigned to 1105 Ma.

The .rot file has the format:
|moving plate ID number|age in millions of years|latitude of Euler pole|longitude of Euler pole|angle|fixed plate|comment| |——————-|————————|——————-|————————|—–|————|——-|
In this .rot file, the fixed reference frame is 000 and 001 (which could be split into a relative and absolute reference frame that seperates out TPW), while the plate ID for Laurentia is 199.
001 0.0 0.0 0.0 0.0 000 !
001 3900.0 0.0 0.0 0.0 000 !
199 0.0 0.0 0.0 0.0 001 !
199 1105.0 0.0 115.4 48.4 001 !Put Osler-upperthird at N for Laurentia
199 1107.5 0.0 121.3 47.3 001 !Put Osler-middlethird at N for Laurentia
199 1110.0 0.0 128.7 49.0 001 !Put Osler-lowerthird at N for Laurentia

(2) Open the following feature collections in GPlates: 199-Laurentia nogrid.dat, OslerPoles.gpml and Osler.rot where 199-Laurentia nogrid.dat is the outline of Laurentia in the late Proterozoic and OslerPoles.gpml contains the three calculated Osler paleomagnetic poles. An animation from 1110 to 1105 Ma yields the following reconstruction shown here with a time slice for Osler_lowerthird (red), Osler_middlethird (yellow) and Osler_upperthird (blue).

```
In [23]: from IPython.display import SVG
         SVG("Paleogeo.svg")
```

```
Out[23]:
```



9 Plate rate estimate

This portion of the analysis calculates plate motion rates and estimates the uncertainity associated with these rates using Monte Carlo simulations that utilize both the uncertainity on the position and age of two paleomagnetic poles. For this analysis we are using these two poles: 1. Pole 1: Osler Volcanic Group (early magmatic stage). The pole is the combined mean of the upper portion of the reversed stratigraphy that uses data from this study and Halls (1974). The date of 1105 ± 2 Ma is used to constrain the age of the pole since it comes from an extrusive unit in close stratigraphic proximity to the units from which the paleomagnetic data were obtained. 2. Pole 2: North Shore Volcanic Group (main magmatic stage). The North Shore Volcanic Group spans more than 10 million years and subsets of the data need to be considered for a pole to provide precise constraints. The pole used here is calculated from data developed by Tauxe and Kodama (2009) and is the mean of 47 sites from between the 40th Ave icelandite and the Palisade rhyolite within the southwest limb of the North Shore Volcanic Group. The 40th Ave icelandite has a U-Pb date of 1098.4 ± 1.9 Ma and the Palisade rhyolite has a U-Pb date of 1096.6 ± 1.7 Ma.

```
In [24]: #import a couple special functions from scipy
         from scipy import special
         from scipy import stats
```

9.1 Calculating the NSVG pole

```
In [25]: Tauxe_NSVG_Data=pandas.read_csv('../2014_Osler_Data/Tauxe2009a_data.csv',sep=',')
         #show first 5 rows
         Tauxe_NSVG_Data.head()

Out[25]:   site_ID  dec_tc  inc_tc  a95  n  site_lat  site_long  pole_lat  pole_long  \
         0   ns002   283.5    38.7  6.6  4   47.7341   -90.4292      24.9      185.2
         1   ns003   286.9    47.9  3.8  5   47.7371   -90.4114      32.0      188.9
         2   ns004   290.0    47.8  5.6  5   47.7328   -90.4363      34.0      186.7
         3   ns005   299.8    37.6  5.3  4   47.7254   -90.4430      35.3      172.4
         4   ns006   294.8    38.8  3.0  5   47.7161   -90.4904      32.5      177.0
```

```
  rem_type sequence  unit
0      hem     nneu  ngha
1    mixed     nneu  ngha
2      mag     nneu  ngha
3      mag     nneu   ngt
4      mag      nsl   NaN

[5 rows x 12 columns]
```

The North Shore Volcanic Group (NSVG) is comprised of two main limbs with distinct stratigraphy and radiometric age control. The southwest limb of the NSVG was particularly well-sampled by Tauxe and Kodama (2009) and those sites can be bracketed with age control from the 40th Ave icelandite (Davis and Green, 1997; 1098.4 ± 1.9 Ma) and the Palisade rhyolite (Davis and Green, 1997; 1096.6 ± 1.7 Ma).

```
In [26]: NSVG_nswu=Tauxe_NSVG_Data.ix[Tauxe_NSVG_Data['sequence'] == 'nswu']
         NSVG_nswu.reset_index(inplace=True)

         NSVG_nswu_VGPs=[]
         NSVG_nswu_Plong=[]
         NSVG_nswu_Plat=[]
         for n in range(0,len(NSVG_nswu)):
             Plong,Plat=NSVG_nswu['pole_long'][n],NSVG_nswu['pole_lat'][n]
             NSVG_nswu_Plong.append(Plong)
             NSVG_nswu_Plat.append(Plat)
             NSVG_nswu_VGPs.append([Plong,Plat,1.])
         NSVG_nswu_mean=pmag.fisher_mean(NSVG_nswu_VGPs)
         print 'Here are the details for the calculated pole from the SW limb of the NSVG'
         print 'Pole longitude is: ' + str(round(NSVG_nswu_mean['dec'],1))
         print 'Pole latitude is: ' + str(round(NSVG_nswu_mean['inc'],1))
         print 'Pole kappa is: ' + str(round(NSVG_nswu_mean['k'],1))
         print 'Pole A95 is: ' + str(round(NSVG_nswu_mean['alpha95'],1))
         print 'Pole N is: ' + str(int(NSVG_nswu_mean['n']))
```

```
Here are the details for the calculated pole from the SW limb of the NSVG
Pole longitude is: 182.1
Pole latitude is: 35.8
Pole kappa is: 45.7
Pole A95 is: 3.1
Pole N is: 47
```

9.2 Input pole data and simulation sample size

The code box below is where the input parameters for the Monte Carlo rate simulation are entered. These parameters are: - `samplesize` determines the number of random pole pairs that will be made through Monte Carlo simulations of each pole and its age to determine rate. - Paleomagnetic pole longitude (`plong`) and latitude (`plat`) for each pole - The A_{95} confidence ellipse (`A95`), the Fisher precision parameter (`kappa`), and number of virtual geomagnetic poles used to calculate the pole mean (`N`). - The age assigned to the pole and the 1σ uncertainity on that age (`age_error`).

```
In [27]: samplesize=100000
         #set at 100,000 this code will take a long time to run

         #parameters for pole 1 (OVG pole calculated above)
         pole1_plong=201.6
```

```
        pole1_plat=42.5
        pole1_A95=3.7
        pole1_kappa=25.7
        pole1_N=59
        pole1_age=1105
        #1 sigma age uncertainity
        pole1_age_error=1

        #parameters for pole 2 (NSVG pole calculated above)
        pole2_plong=182.1
        pole2_plat=35.8
        pole2_A95=3.1
        pole2_kappa=45.7
        pole2_N=47
        #taking the average of the 40th Ave and Palisade
        pole2_age=1097.5
        #2sigma uncertainity min age on Palisade is 1094.9
        #2sigma uncertainity min age on 40th Ave is 1100.3
        #a 2sigma of 2.7 on the pole2_age approx spans this range
        #giving a 1 sigma age uncertainty of
        pole2_age_error=1.35

        #the longitude, latitude of Duluth, MN as a reference location
        Duluth=[267.9, 46.8]
```

```
In [28]: pole1=(pole1_plong, pole1_plat)
         pole1_paleolat=90-pmag.angle(pole1,Duluth)
         pole2=(pole2_plong, pole2_plat)
         pole2_paleolat=90-pmag.angle(pole2,Duluth)
         print "The paleolatitude for Duluth resulting from pole 1 is:" + str(pole1_paleolat)
         print "The paleolatitude for Duluth resulting from pole 2 is:" + str(pole2_paleolat)
         rate=((pole1_paleolat-pole2_paleolat)*111*100000)/((pole1_age-pole2_age)*1000000)
         print "The rate of paleolatitudinal change implied by the poles pairs in cm/yr is:" + str(rate
```

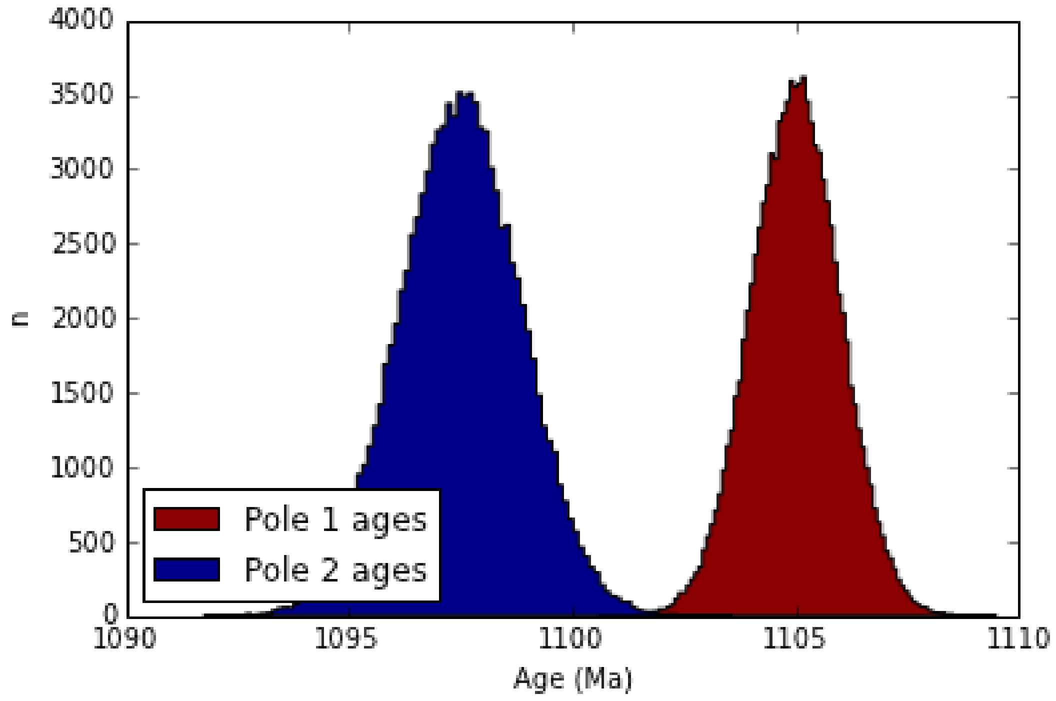
```
The paleolatitude for Duluth resulting from pole 1 is:[ 44.05491798]
The paleolatitude for Duluth resulting from pole 2 is:[ 27.84482051]
The rate of paleolatitudinal change implied by the poles pairs in cm/yr is:[ 23.99094426]
```

9.3 Sampling the age distribution

Weighted means calculated from geochronological data are typically assumed to have uncertainity that follows a Gaussian distribution. Using this assumption, we can use the numpy function `random` to randomly sample a normal distribution. This approach is straightforward for a paleomagnetic pole where we are assigning a single age (e.g. `pole1` in this case). It is less straightforward when using two dates that stratigraphically bracket flows from which the paleomagnetic data was obtained without an a priori assumption of the distribution of time between those two dated units. In this analysis, we approximate the age error distribution for `pole2` by taking the mean of the dated units that bracket the data as the mean age and then specifying an age uncertainity wherein 2σ reaches out to approximately reach the maximum and minimum dates implied by the 2σ uncertainities on the older and younger dates respectively. This approach specifies a distribution that is centered on the mean of the two dates while also including the possibility that the actual age of the pole could come from a distribution that spans out to include the older and younger possibilities of each dated unit.

```
In [29]: pole1_MCages=np.random.normal(pole1_age,pole1_age_error,samplesize)
         pole2_MCages=np.random.normal(pole2_age,pole2_age_error,samplesize)
```

```
plt.hist(pole1_MCages,100,histtype='stepfilled',color='darkred',label='Pole 1 ages')
plt.hist(pole2_MCages,100,histtype='stepfilled',color='darkblue',label='Pole 2 ages')
plt.xlabel('Age (Ma)')
plt.ylabel('n')
plt.legend(loc=3)
plt.show()
```

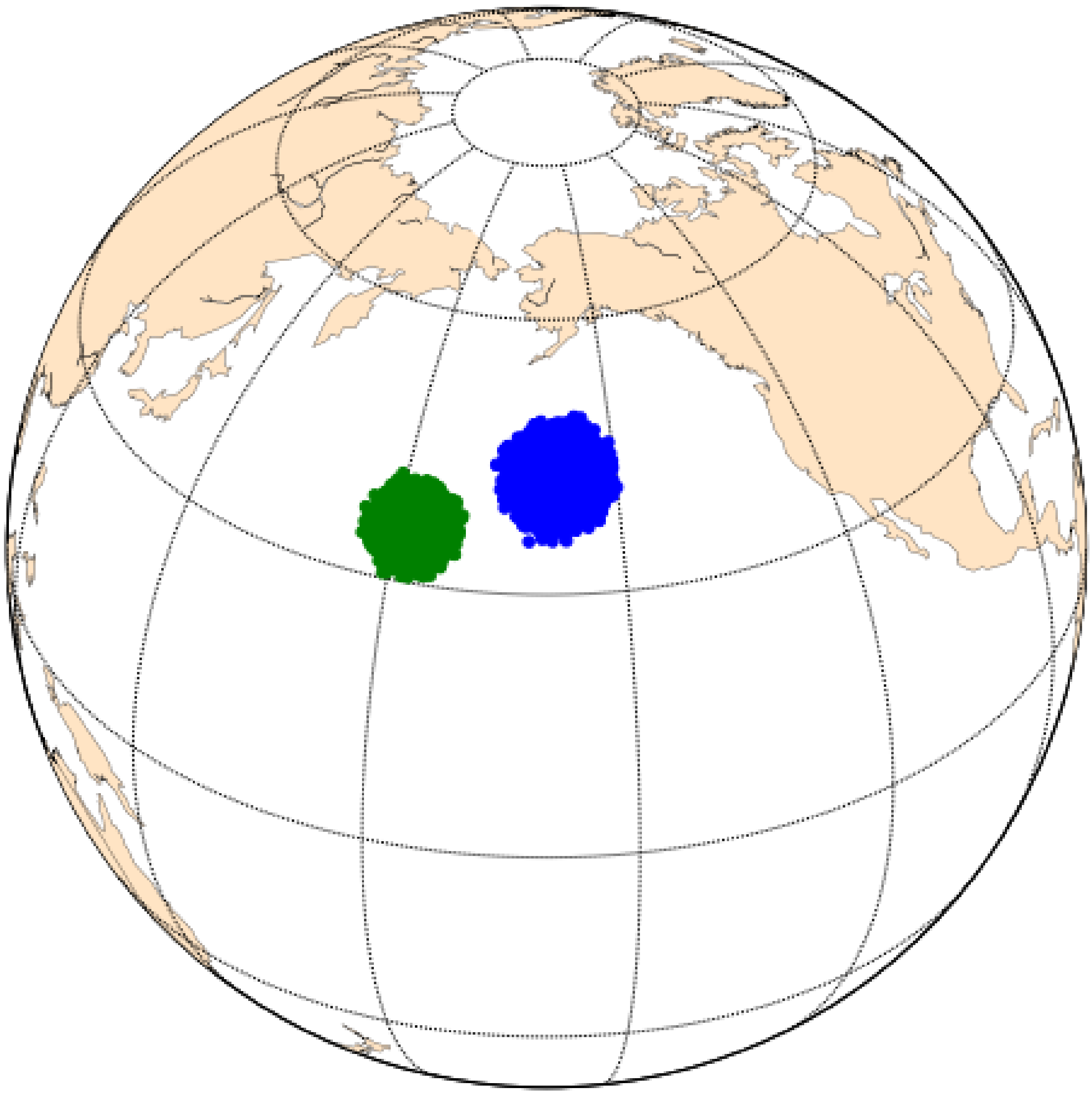


9.4 Sampling poles

To determine the uncertainity on the pole position and associated paleolatitude, we can take the approach of random sampling from the Fisher distribution with the precision of the VGPs used to develop the pole. To do this in a Monte Carlo fashion, we can grab random samples utilizing the fshdev function from pmag.py which takes random samples from a distribution with a given Fisher precision parameter (kappa) centered on (Dec=0, Inc=90). We can then define a tilt_direction to be the longitude of the pole mean and a tilt_amount to be the conjugate of the latitude of the pole mean and then use the dotilt function of pmag.py to rotate the polar-centered distribution to be centered about the pole mean. Doing this for the same number of VGPs used to calculate the actual pole allows us to then calculate a mean pole. This can then be done repeated (as dictated by the sample size number) to get a population of mean poles that can result from sampling the distribution.

```
In [30]: pole1_MCpoles=[]
         pole1_MCpole_lat=[]
         pole1_MCpole_long=[]
         pole1_MCpaleolat=[]
         for n in range(samplesize):
             vgp_samples=[]
             for vgp in range(pole1_N):
                 #pmag.dev returns a direction from a fisher distribution with specified kappa
                 direction_atN=pmag.fshdev(pole1_kappa)
                 #this direction is centered at latitude of 90° and needs to be rotated
                 #to be centered on the mean pole position
                 tilt_direction=pole1_plong
                 tilt_amount=90-pole1_plat
                 direction=pmag.dotilt(direction_atN[0],direction_atN[1],tilt_direction,tilt_amount)
                 vgp_samples.append([direction[0],direction[1],1.])
             mean=pmag.fisher_mean(vgp_samples)
             mean_pole_position=(mean['dec'],mean['inc'])
             pole1_MCpoles.append([mean['dec'],mean['inc'],1.])
             pole1_MCpole_lat.append(mean['inc'])
             pole1_MCpole_long.append(mean['dec'])
             paleolat=90-pmag.angle(mean_pole_position,Duluth)
             pole1_MCpaleolat.append(paleolat[0])

         pole2_MCpoles=[]
         pole2_MCpole_lat=[]
         pole2_MCpole_long=[]
         pole2_MCpaleolat=[]
         for n in range(samplesize):
             vgp_samples=[]
             for vgp in range(pole2_N):
                 #pmag.dev returns a direction from a fisher distribution with specified kappa
                 direction_atN=pmag.fshdev(pole2_kappa)
                         #this direction is centered at latitude of 90° and needs to be rotated
                 #to be centered on the mean pole position
                 tilt_direction=pole2_plong
                 tilt_amount=90-pole2_plat
                 direction=pmag.dotilt(direction_atN[0],direction_atN[1],tilt_direction,tilt_amount)
                 vgp_samples.append([direction[0],direction[1],1.])
             mean=pmag.fisher_mean(vgp_samples)
             mean_pole_position=(mean['dec'],mean['inc'])
             pole2_MCpoles.append([mean['dec'],mean['inc'],1.])
             pole2_MCpole_lat.append(mean['inc'])
             pole2_MCpole_long.append(mean['dec'])
             paleolat=90-pmag.angle(mean_pole_position,Duluth)
             pole2_MCpaleolat.append(paleolat[0])
```

```
In [31]: plt.figure(figsize=(8, 8))
         m = Basemap(projection='ortho',lat_0=35,lon_0=200,resolution='c',area_thresh=50000)
         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.vgpplot(m,pole1_MCpole_long,pole1_MCpole_lat,color='b')
         IPmag.vgpplot(m,pole2_MCpole_long,pole2_MCpole_lat,color='g')
```



9.5 Calculate and graphically display the confidence interval on the minimum rate estimate implied by the Monte Carlo sampled pairs of sampled ages and paleolatitudes from the two poles

The pairs of sampled ages and poles can be used to calcuate rates (with the total number of rates being set by `samplesize` in the input box above). The rate calculated above of 24.0 cm/year can now be given confidence bounds by taking 2.5 percentile and 97.5 percentile of the Monte Carlo simulated rates.

```
In [32]: #calculating the change in paleolatitude between the Monte Carlo pairs
         pole1_pole2_Delta_degrees=[]
         pole1_pole2_Delta_kilometers=[]
         pole1_pole2_Delta_myr=[]
         pole1_pole2_degrees_per_myr=[]
         pole1_pole2_cm_per_yr=[]
```

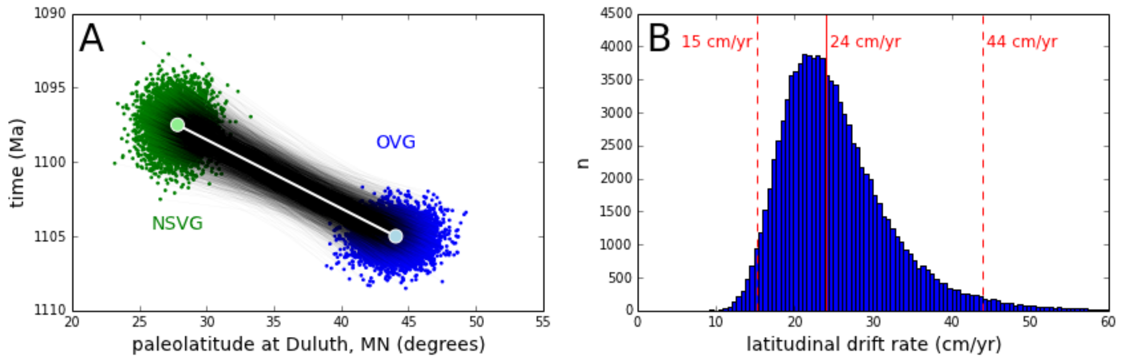
```
    for n in range(samplesize):
        Delta_degrees=pole1_MCpaleolat[n]-pole2_MCpaleolat[n]
        Delta_Myr=pole1_MCages[n]-pole2_MCages[n]
        pole1_pole2_Delta_degrees.append(Delta_degrees)
        degrees_per_myr=Delta_degrees/Delta_Myr
        cm_per_yr=((Delta_degrees*111)*100000)/(Delta_Myr*1000000)
        pole1_pole2_degrees_per_myr.append(degrees_per_myr)
        pole1_pole2_cm_per_yr.append(cm_per_yr)
```

```
In [33]: twopointfive_percentile=stats.scoreatpercentile(pole1_pole2_cm_per_yr,2.5)
         fifty_percentile=stats.scoreatpercentile(pole1_pole2_cm_per_yr,50)
         ninetysevenpointfive_percentile=stats.scoreatpercentile(pole1_pole2_cm_per_yr,97.5)
         print "2.5th percentile is: " + str(twopointfive_percentile)
         print "50th percentile is: " + str(fifty_percentile)
         print "97.5th percentile is: " + str(ninetysevenpointfive_percentile)
```

```
2.5th percentile is: 15.1860289306
50th percentile is: 23.9833375899
97.5th percentile is: 44.0185724854
```

We can see here that the 50th percentile from these analysis is the same as the mean rate previously calculated (24.0 cm/yr). The 2.5th and 97.5th percentile give a 95% confidence range of 15.2 to 44.4 cm/yr. The code below generates a plot where 5,000 of the 100,000 (that's what was set as `samplesize` when code was executed) pole pairs and rates are shown (in A) and a histogram of all the rates is plotted along with the above percentiles being marked. This plot is Figure 4 of the manuscript.

```
In [46]: plotnumber=5000
         figure(num=None, figsize=(14, 4))

         plt.subplot(1, 2, 1)
         for n in range(plotnumber):
             plt.plot([pole1_MCpaleolat[n],pole2_MCpaleolat[n]],
                      [pole1_MCages[n],pole2_MCages[n]],'k-',linewidth=0.05,alpha=0.3)
         plt.scatter(pole1_MCpaleolat[:plotnumber],pole1_MCages[:plotnumber],color='b',s=3)
         plt.scatter(pole1_paleolat,pole1_age,color='lightblue',s=100, edgecolor='w', zorder=10000)
         plt.scatter(pole2_MCpaleolat[:plotnumber],pole2_MCages[:plotnumber],color='g',s=3)
         plt.scatter(pole2_paleolat,pole2_age,color='lightgreen',s=100, edgecolor='w', zorder=10000)
         plt.plot([pole1_paleolat,pole2_paleolat],[pole1_age,pole2_age],'w-',linewidth=2)
         plt.gca().invert_yaxis()
         plt.xlabel('paleolatitude at Duluth, MN (degrees)',size=14)
         plt.ylabel('time (Ma)',size=14)
         plt.text(pole1_paleolat,1099,'OVG', color='b',ha ='center',size=14)
         plt.text(pole2_paleolat,1104.5,'NSVG', color='g',ha ='center',size=14)
         plt.text(20.5,1092.5,'A', color='k',ha ='left',size=28)

         plt.subplot(1, 2, 2)
         plt.hist(pole1_pole2_cm_per_yr,bins=600)
         plt.vlines(twopointfive_percentile,0,4500,'r', linestyles='dashed')
         plt.text(twopointfive_percentile-0.5,4000,'15 cm/yr', color='r',ha ='right',size=12)
         plt.vlines(fifty_percentile,0,4500,'r', linestyles='solid')
         plt.text(fifty_percentile+0.5,4000,'24 cm/yr', color='r',ha ='left',size=12)
         plt.vlines(ninetysevenpointfive_percentile,0,4500,'r', linestyles='dashed')
         plt.text(ninetysevenpointfive_percentile+0.5,4000,'44 cm/yr', color='r',ha ='left',size=12)
         plt.text(1.5,3930,'B', color='k',ha ='left',size=28)
         plt.ylabel('n',size=14)
         plt.xlabel('latitudinal drift rate (cm/yr)',size=14)
         plt.xlim([0,60])
         plt.ylim([0,4500])
         plt.savefig('../2014_Osler_Manuscript_Files/2014_Osler_Figures/MonteCarlo.pdf',
                     bbox_inches='tight')
         plt.show()
```



9.6 Latitudinal change from beginning of early magmatic stage to main magmatic stage

We can use the pole from the lower third of the Simpson Island stratigraphy and the pole calculated above from the NSVG to estimate the magnitude of paleolatitudinal change that occured during the early magmatic stage, the latent magmatic stage and into the main magmatic stage. This paleolatitudinal difference is discussed in the manuscript text.

```
In [35]: lower_Osler_pole=(218.6,40.9)
         NSVG_nswu_pole=(182.1,35.8)
         lower_Osler_pole_paleolat=90-pmag.angle(lower_Osler_pole,Duluth)
         NSVG_nswu_paleolat=90-pmag.angle(NSVG_nswu_pole,Duluth)
         difference=lower_Osler_pole_paleolat-NSVG_nswu_paleolat
         print 'The difference in paleolatitude resulting from these poles is: ' + str(difference)
         print 'from ' + str(lower_Osler_pole_paleolat) + ' to ' + str(NSVG_nswu_paleolat) + ' using Du
```

```
The difference in paleolatitude resulting from these poles is: [ 26.7121489]
from [ 54.5569694] to [ 27.84482051] using Duluth, MN as a reference location
```

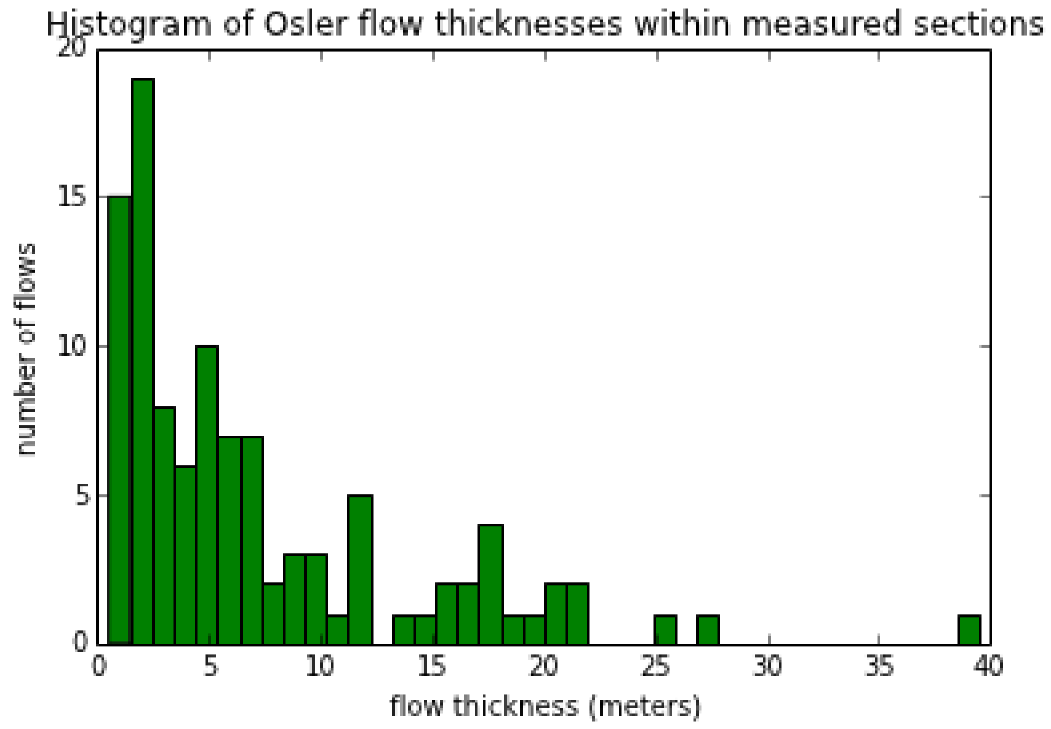
10 Lava flow thickness data

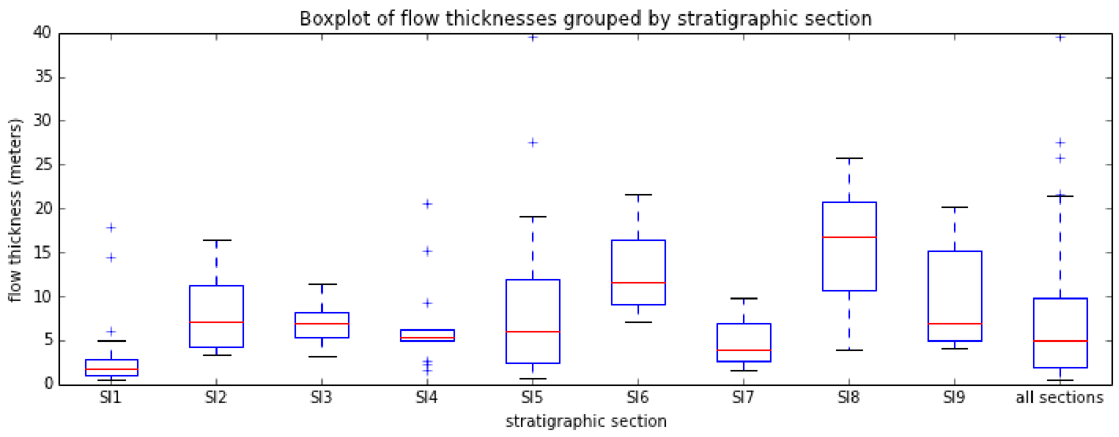
In the section of the text entitled "Osler Volcanic Group lithologies" the median flow thickness is reported from the subset of flows within measured sections where there is sufficient exposure of the flow base, interior and top to determine thickness. In the text it is reported that the median flow thickness is 4.9 meters with a first quartile thickness of 2.0 meters and third quartile thickness of 9.8 meters. The flow thickness data used in this analysis is presented and analyzed below. Note that not every flow within the measured stratigraphic sections was sampled for paleomagnetic analysis and not every flow on Simpson Island was measured in a stratigraphic section.

```
In [36]: #flow thicknesses where bottom and top are exposed without cover in between
         #or with minimal cover and assumption of continuity
         SI1_flowthickness = array([1.2, 1.1+0.7,0.2+2.9+0.3,5.1+5.4+0.6+3.4,0.6+1+.3,0.9,0.5,2.2,0.8,0
         SI2_flowthickness = array([0.6+2.8,5.7+2+3.6,1.3+0.7+1.1+1.1,6.2+1,1.8+3.3+11.4])
         SI3_flowthickness = array([0.6+0.8+1+.8,1+4.9+1.1,0.3+1.6+4.5+1.2,0.2+1.5+7.2+0.9,0.5+1.7+9.3,
         SI4_flowthickness = array([0.2+1.2+0.9,0.3+1.5+0.4+3.2,14+6.5,0.7+1+3.6,8.2+2.4+4.6,3.1+1.4+1.
         SI5_flowthickness = array([6.6+9.8+7.4+1.4+2.3,10.2+0.8,5.6+2.8+1.4+2.2,0.2+2.8+2.4+0.7,0.6,0.
         SI6_flowthickness = array([13.1+3.3,13.5+3.7,6.5+0.6,6.1+2.1+0.9,0.9+6.6+4.0,7.3+5.6+1.3+7.4,0
         SI7_flowthickness = array([0.4+0.9+1.2+1.4,0.3+1.2,0.5+2.5+5+1.9])
         SI8_flowthickness = array([1.4+1.9+0.6,0.5+1+8.8+4+1.1,5+15.8+2.8+2.1,5+4.2,8.4+8.1+5,1.8+11.8
         SI9_flowthickness = array([5.0+4.2,14.0+1+2.1,2.8+2.1,3.6+12.3+3.6+0.7,4.2+2.3,1.3+1.4+2.2,0.8

         SIall_flowthickness = concatenate((SI1_flowthickness, SI2_flowthickness,
                                            SI3_flowthickness, SI4_flowthickness,
                                            SI5_flowthickness, SI6_flowthickness,
                                            SI7_flowthickness, SI8_flowthickness,
                                            SI9_flowthickness))
         print "The number of flows for which thickness estimates could be obtained is:"
         print len(SIall_flowthickness)
         print "The mean flow thickness is:"
         print np.mean(SIall_flowthickness)
         print "The median flow thickness is:"
         print np.median(SIall_flowthickness)

         print "The first quartile thickness is:"
         print np.percentile(SIall_flowthickness,25)
         print "The third quartile thickness is:"
         print np.percentile(SIall_flowthickness,75)

         hist(SIall_flowthickness, 40, facecolor='green')
         title('Histogram of Osler flow thicknesses within measured sections')
         ylabel('number of flows')
         xlabel('flow thickness (meters)')
         plt.show()
         data=(SI1_flowthickness,SI2_flowthickness,SI3_flowthickness,
               SI4_flowthickness,SI5_flowthickness, SI6_flowthickness,
               SI7_flowthickness,SI8_flowthickness,SI9_flowthickness,SIall_flowthickness)

         figure(figsize=(12,4))
         title('Boxplot of flow thicknesses grouped by stratigraphic section')
         boxplot(data)
         labels = ('SI1', 'SI2', 'SI3', 'SI4', 'SI5', 'SI6',
                   'SI7', 'SI8', 'SI9', 'all sections')
         xticks(range(1,11),labels)
         xlabel('stratigraphic section')
         ylabel('flow thickness (meters)')
         plt.show()
```

```
The number of flows for which thickness estimates could be obtained is:
105
The mean flow thickness is:
7.16666666667
The median flow thickness is:
4.9
The first quartile thickness is:
2.0
The third quartile thickness is:
```

```
9.8
```





11 The IPmag.py library of functions

Some of the functions used in the analysis above come from the imported IPmag.py library that was developed for this data analysis project relying heavily on the pmag.py functions developed by Lisa Tauxe. The code that is within IPmag.py is shown below so that the PDF output of this IPython notebook can document more of the tools used in the analysis without the reader needing to look into the .py files themselves. This library utilizes functions from the pmag.py and pmagplotlib.py libraries of PmagPy version 2.206.

```
In [37]: %load IPmag.py

In [ ]: import pmag, pmagplotlib
        import pylab
        import numpy as np
        import matplotlib.pyplot as plt

        def iflip(D): #function simplified from PmagPy pmag.flip function
            """
            This function returns the antipode (flips) of the unit vectors in D (dec,inc,length).
            """
            Dflip=[]
            for rec in D:
                d,i=(rec[0]-180.)%360.,-rec[1]
                Dflip.append([d,i,1.])
            return Dflip

        def iBootstrap(Data1,Data2,NumSims=1000):
            """
            Conduct a bootstrap test (Tauxe, 2010) for a common mean on two declination,
            inclination data sets

            This function modifies code from PmagPy for use calculating and plotting
            bootstrap statistics. Three plots are generated (one for x, one for y and
            one for z). If the 95 percent confidence bounds for each component overlap
            each other, the two directions are not significantly different.

            Parameters
            ----------

            Data1 : a list of directional data [dec,inc]
            Data2 : a list of directional data [dec,inc]
            NumSims : number of bootstrap samples (default is 1000)
            """
            counter=0
            BDI1=pmag.di_boot(Data1)
            BDI2=pmag.di_boot(Data2)
            print ""
            print "==============="
            print ""
            print "Here are the results of the bootstrap test for a common mean"
            CDF={'X':1,'Y':2,'Z':3}
            pylab.figure(CDF['X'],figsize=(3,3),dpi=160)
            pylab.figure(CDF['Y'],figsize=(3,3),dpi=160)
            pylab.figure(CDF['Z'],figsize=(3,3),dpi=160)
            pmagplotlib.plotCOM(CDF,BDI1,BDI2,["",""])

        def iWatsonV(Data1,Data2,NumSims=5000):
            """
            Conduct a Watson V test for a common mean on two declination, inclination data sets

            This function calculates Watson's V statistic from input files through Monte Carlo
            simulation in order to test whether two populations of directional data could have
            been drawn from a common mean. The critical angle between the two sample mean
```

```
directions and the corresponding McFadden and McElhinny (1990) classification is printed.


Parameters
----------

Data1 : a list of directional data [dec,inc]
Data2 : a list of directional data [dec,inc]
NumSims : number of Monte Carlo simulations (default is 5000)
"""
pars_1=pmag.fisher_mean(Data1)
pars_2=pmag.fisher_mean(Data2)

cart_1=pmag.dir2cart([pars_1["dec"],pars_1["inc"],pars_1["r"]])
cart_2=pmag.dir2cart([pars_2['dec'],pars_2['inc'],pars_2["r"]])
Sw=pars_1['k']*pars_1['r']+pars_2['k']*pars_2['r'] # k1*r1+k2*r2
xhat_1=pars_1['k']*cart_1[0]+pars_2['k']*cart_2[0] # k1*x1+k2*x2
xhat_2=pars_1['k']*cart_1[1]+pars_2['k']*cart_2[1] # k1*y1+k2*y2
xhat_3=pars_1['k']*cart_1[2]+pars_2['k']*cart_2[2] # k1*z1+k2*z2
Rw=np.sqrt(xhat_1**2+xhat_2**2+xhat_3**2)
V=2*(Sw-Rw)
# keep weighted sum for later when determining the "critical angle"
# let's save it as Sr (notation of McFadden and McElhinny, 1990)
Sr=Sw

# do monte carlo simulation of datasets with same kappas as data,
# but a common mean
counter=0
Vp=[] # set of Vs from simulations
for k in range(NumSims):

# get a set of N1 fisher distributed vectors with k1,
# calculate fisher stats
    Dirp=[]
    for i in range(pars_1["n"]):
        Dirp.append(pmag.fshdev(pars_1["k"]))
    pars_p1=pmag.fisher_mean(Dirp)
# get a set of N2 fisher distributed vectors with k2,
# calculate fisher stats
    Dirp=[]
    for i in range(pars_2["n"]):
        Dirp.append(pmag.fshdev(pars_2["k"]))
    pars_p2=pmag.fisher_mean(Dirp)
# get the V for these
    Vk=pmag.vfunc(pars_p1,pars_p2)
    Vp.append(Vk)

# sort the Vs, get Vcrit (95th percentile one)

Vp.sort()
k=int(.95*NumSims)
Vcrit=Vp[k]

# equation 18 of McFadden and McElhinny, 1990 calculates the critical
# value of R (Rwc)

Rwc=Sr-(Vcrit/2)

# following equation 19 of McFadden and McElhinny (1990) the critical
# angle is calculated. If the observed angle (also calculated below)
# between the data set means exceeds the critical angle the hypothesis
# of a common mean direction may be rejected at the 95% confidence
# level. The critical angle is simply a different way to present
# Watson's V parameter so it makes sense to use the Watson V parameter
# in comparison with the critical value of V for considering the test
# results. What calculating the critical angle allows for is the
# classification of McFadden and McElhinny (1990) to be made
# for data sets that are consistent with sharing a common mean.

k1=pars_1['k']
k2=pars_2['k']
R1=pars_1['r']
R2=pars_2['r']
critical_angle=np.degrees(np.arccos(((Rwc**2)-((k1*R1)**2)
                                          -((k2*R2)**2))/
                                         (2*k1*R1*k2*R2)))
D1=(pars_1['dec'],pars_1['inc'])
D2=(pars_2['dec'],pars_2['inc'])
angle=pmag.angle(D1,D2)

print "Results of Watson V test: "
print ""
print "Watson's V:           " '%.1f' %(V)
print "Critical value of V:  " '%.1f' %(Vcrit)

if V<Vcrit:
    print '"Pass": Since V is less than Vcrit, the null hypothesis'
    print 'that the two populations are drawn from distributions'
    print 'that share a common mean direction can not be rejected.'
elif V>Vcrit:
    print '"Fail": Since V is greater than Vcrit, the two means can'
    print 'be distinguished at the 95% confidence level.'
print ""
print "M&M1990 classification:"
print ""
print "Angle between data set means: " '%.1f'%(angle)
print "Critical angle for M&M1990:   " '%.1f'%(critical_angle)

if V>Vcrit:
    print ""
elif V<Vcrit:
    if critical_angle<5:
        print "The McFadden and McElhinny (1990) classification for"
        print "this test is: 'A'"
    elif critical_angle<10:
        print "The McFadden and McElhinny (1990) classification for"
        print "this test is: 'B'"
    elif critical_angle<20:
            print "The McFadden and McElhinny (1990) classification for"
            print "this test is: 'C'"
        else:
            print "The McFadden and McElhinny (1990) classification for"
            print "this test is: 'INDETERMINATE;"

def lat_from_i(inc):
    """
    Calculate paleolatitude from inclination using the dipole equation
    """
    rad=np.pi/180.
    paleo_lat=np.arctan( 0.5*np.tan(inc*rad))/rad
    return paleo_lat


def iplotDI(DIblock,color='k'):
    """
    Plot declination, inclination data on an equal area plot

    This function modifies the plotDI function of PmagPy for use in the IPython notebook environ

    Parameters
    ----------

    DIblock : a DIblock is comprise of a list of unit vectors [dec,inc,1.]
    color : the default color is black. Other colors can be chosen (e.g. 'r')
    """
    # initialize the variables
    X_down,X_up,Y_down,Y_up=[],[],[],[]
    for rec in DIblock:
        Up,Down=0,0
        XY=pmag.dimap(rec[0],rec[1])
        if rec[1] >= 0:
            X_down.append(XY[0])
            Y_down.append(XY[1])
        else:
            X_up.append(XY[0])
            Y_up.append(XY[1])

    if len(X_up)>0:
        pylab.scatter(X_up,Y_up,facecolors='none', edgecolors=color)

    if len(X_down)>0:
        pylab.scatter(X_down,Y_down,facecolors=color, edgecolors=color)

def iplotDImean(Dec,Inc,a95,color='k',marker='o',label=''):
    """
    Plot a mean declination, inclination with alpha_95 ellipse on an equal area plot.

    Before this function is called a plot needs to be initialized with code that looks
    something like:
    >fignum = 1
    >pylab.figure(num=fignum,figsize=(10,10),dpi=160)
    >pmagplotlib.plotNET(fignum)
    Parameters
    ----------

    Dec : declination of mean being plotted
    Inc : inclination of mean being plotted
    a95 : a95 confidence ellipse of mean being plotted
    color : the default color is black. Other colors can be chosen (e.g. 'r')
    marker : the default is a circle. Other symbols can be chose (e.g. 's')
    label : the default is no label. Labels can be assigned
    """
    DI_dimap=pmag.dimap(Dec,Inc)
    pylab.plot(DI_dimap[0],DI_dimap[1],color=color,marker=marker,label=label)
    Xcirc,Ycirc=[],[]
    Da95,Ia95=pmag.circ(Dec,Inc,a95)
    pylab.legend(loc=2)
    for k in  range(len(Da95)):
        XY=pmag.dimap(Da95[k],Ia95[k])
        Xcirc.append(XY[0])
        Ycirc.append(XY[1])
    pylab.plot(Xcirc,Ycirc,color)

def shoot(lon, lat, azimuth, maxdist=None):
    """
    This function enables A95 error ellipses to be drawn in basemap around paleomagnetic poles i
    (from: http://www.geophysique.be/2011/02/20/matplotlib-basemap-tutorial-09-drawing-circles/)
    """
    glat1 = lat * np.pi / 180.
    glon1 = lon * np.pi / 180.
    s = maxdist / 1.852
    faz = azimuth * np.pi / 180.

    EPS= 0.00000000005
    if ((np.abs(np.cos(glat1))<EPS) and not (np.abs(np.sin(faz))<EPS)):
        alert("Only N-S courses are meaningful, starting at a pole!")

    a=6378.13/1.852
    f=1/298.257223563
    r = 1 - f
    tu = r * np.tan(glat1)
    sf = np.sin(faz)
    cf = np.cos(faz)
    if (cf==0):
        b=0.
    else:
        b=2. * np.arctan2 (tu, cf)

    cu = 1. / np.sqrt(1 + tu * tu)
    su = tu * cu
    sa = cu * sf
    c2a = 1 - sa * sa
    x = 1. + np.sqrt(1. + c2a * (1. / (r * r) - 1.))
    x = (x - 2.) / x
    c = 1. - x
    c = (x * x / 4. + 1.) / c
    d = (0.375 * x * x - 1.) * x
    tu = s / (r * a * c)
    y = tu
    c = y + 1
    while (np.abs (y - c) > EPS):

        sy = np.sin(y)
        cy = np.cos(y)
        cz = np.cos(b + y)
        e = 2. * cz * cz - 1.
        c = y
        x = e * cy
        y = e + e - 1.
        y = (((sy * sy * 4. - 3.) * y * cz * d / 6. + x) *
              d / 4. - cz) * sy * d + tu

    b = cu * cy * cf - su * sy
    c = r * np.sqrt(sa * sa + b * b)
    d = su * cy + cu * sy * cf
    glat2 = (np.arctan2(d, c) + np.pi) % (2*np.pi) - np.pi
    c = cu * cy - su * sy * cf
    x = np.arctan2(sy * sf, c)
    c = ((-3. * c2a + 4.) * f + 4.) * c2a * f / 16.
    d = ((e * cy * c + cz) * sy * c + y) * sa
    glon2 = ((glon1 + x - (1. - c) * d * f + np.pi) % (2*np.pi)) - np.pi

    baz = (np.arctan2(sa, b) + np.pi) % (2 * np.pi)

    glon2 *= 180./np.pi
    glat2 *= 180./np.pi
    baz *= 180./np.pi

    return (glon2, glat2, baz)

def equi(m, centerlon, centerlat, radius, color):
    """
    This function enables A95 error ellipses to be drawn in basemap around paleomagnetic poles i
    (from: http://www.geophysique.be/2011/02/20/matplotlib-basemap-tutorial-09-drawing-circles/)
    """
    glon1 = centerlon
    glat1 = centerlat
    X = []
    Y = []
    for azimuth in range(0, 360):
        glon2, glat2, baz = shoot(glon1, glat1, azimuth, radius)
        X.append(glon2)
        Y.append(glat2)
    X.append(X[0])
    Y.append(Y[0])

    X,Y = m(X,Y)
    plt.plot(X,Y,color)
```

```python
def poleplot(mapname,plong,plat,A95,label='',color='k',marker='o'):
    """
    This function plots a paleomagnetic pole and A95 error ellipse on whatever
    current map projection has been set using the basemap plotting library.

    Parameters
    -----------
    mapname : the name of the current map that has been developed using basemap
    plong : the longitude of the paleomagnetic pole being plotted (in degrees E)
    plat : the latitude of the paleomagnetic pole being plotted (in degrees)
    A95 : the A_95 confidence ellipse of the paleomagnetic pole (in degrees)
    label : a string that is the label for the paleomagnetic pole being plotted
    color : the color desired for the symbol and its A95 ellipse (default is 'k' aka black)
    marker : the marker shape desired for the pole mean symbol (default is 'o' aka a circle)
    """
    centerlon, centerlat = mapname(plong,plat)
    A95_km=A95*111.32
    mapname.scatter(centerlon,centerlat,20,marker=marker,color=color,label=label,zorder=101)
    equi(mapname, plong, plat, A95_km,color)

def vgpplot(mapname,plong,plat,label='',color='k',marker='o'):
    """
    This function plots a paleomagnetic pole on whatever current map projection
    has been set using the basemap plotting library.

    Parameters
    -----------
    mapname : the name of the current map that has been developed using basemap
    plong : the longitude of the paleomagnetic pole being plotted (in degrees E)
    plat : the latitude of the paleomagnetic pole being plotted (in degrees)
    color : the color desired for the symbol and its A95 ellipse (default is 'k' aka black)
    marker : the marker shape desired for the pole mean symbol (default is 'o' aka a circle)
    """
    centerlon, centerlat = mapname(plong,plat)
    mapname.scatter(centerlon,centerlat,20,marker=marker,color=color,label=label,zorder=100)

def VGP_calc(dataframe):
    """
    This function calculates paleomagnetic poles from directional data within a pandas.DataFrame

    Parameters
    -----------
    dataframe : the name of the pandas.DataFrame containing the data
    dataframe['site_lat'] : the latitude of the site
    dataframe['site_long'] : the longitude of the site
    dataframe['inc_tc'] : the tilt-corrected inclination
    dataframe['dec_tc'] : the tilt-corrected declination
    """
    #calculate the paleolatitude/colatitude
    dataframe['paleolatitude']=np.degrees(np.arctan(0.5*np.tan(np.radians(dataframe['inc_tc']))))
    dataframe['colatitude']=90-dataframe['paleolatitude']
    #calculate the latitude of the pole
    dataframe['pole_lat']=np.degrees(np.arcsin(np.sin(np.radians(dataframe['site_lat']))*
                                                     np.cos(np.radians(dataframe['colatitude
                                                            np.cos(np.radians(dataframe['site_lat']
                                                            np.sin(np.radians(dataframe['colatitude
                                                            np.cos(np.radians(dataframe['dec_tc']))
            #calculate the longitudinal difference between the pole and the site (beta)
            dataframe['beta']=np.degrees(np.arcsin((np.sin(np.radians(dataframe['colatitude']))*
                                                np.sin(np.radians(dataframe['dec_tc'])))/
                                               (np.cos(np.radians(dataframe['pole_lat'])))))
            #generate a boolean array (mask) to use to distinguish between the two possibilities for pol
            #and then calculate pole longitude using the site location and calculated beta
            mask = np.cos(np.radians(dataframe['colatitude']))>np.sin(np.radians(dataframe['site_lat']))
            dataframe['pole_long']=np.where(mask,(dataframe['site_long']+dataframe['beta'])%360.,(datafra
            #calculate the antipode of the poles
            dataframe['pole_lat_rev']=-dataframe['pole_lat']
            dataframe['pole_long_rev']=(dataframe['pole_long']-180.)%360.
            #the 'colatitude' and 'beta' columns were created for the purposes of the pole calculations
            #but aren't of further use and are deleted
            del dataframe['colatitude']
            del dataframe['beta']
```

```
In [38]: import pmag, pmagplotlib
         import pylab
         import numpy
         import matplotlib.pyplot as plt

         def iplotDI(DIblock,color='k'):
             """
             Plot declination, inclination data on a equal area plot

             This function modifies the plotDI function of PmagPy for
             use in the IPython notebook environment

             Parameters
             ----------

             DIblock : a DIblock is comprise of a list of unit vectors [dec,inc,1.]
             color : the default color is black. Other colors can be chosen (e.g. 'r')
             """
             # initialize the variables
             X_down,X_up,Y_down,Y_up=[],[],[],[]
             for rec in DIblock:
                 Up,Down=0,0
                 XY=pmag.dimap(rec[0],rec[1])
                 if rec[1] >= 0:
                     X_down.append(XY[0])
                     Y_down.append(XY[1])
                 else:
                     X_up.append(XY[0])
                     Y_up.append(XY[1])

             if len(X_up)>0:
                 pylab.scatter(X_up,Y_up,facecolors='none', edgecolors=color)

             if len(X_down)>0:
                 pylab.scatter(X_down,Y_down,facecolors=color, edgecolors=color)
```

```
def iplotDImean(Dec,Inc,a95,color='k',marker='o',label=''):
    """
    Plot a mean declination, inclination with alpha_95 ellipse on an equal area plot.

    Before this function is called a plot needs to be initialized with code that looks
    something like:
    >fignum = 1
    >pylab.figure(num=fignum,figsize=(10,10),dpi=160)
    >pmagplotlib.plotNET(fignum)

    Parameters
    ----------

    Dec : declination of mean being plotted
    Inc : inclination of mean being plotted
    a95 : a95 confidence ellipse of mean being plotted
    color : the default color is black. Other colors can be chosen (e.g. 'r')
    marker : the default is a circle. Other symbols can be chose (e.g. 's')
    label : the default is no label. Labels can be assigned
    """
    DI_dimap=pmag.dimap(Dec,Inc)
    pylab.plot(DI_dimap[0],DI_dimap[1],color=color,marker=marker,label=label)
    Xcirc,Ycirc=[],[]
    Da95,Ia95=pmag.circ(Dec,Inc,a95)
    legend(loc=2)
    for k in  range(len(Da95)):
        XY=pmag.dimap(Da95[k],Ia95[k])
        Xcirc.append(XY[0])
        Ycirc.append(XY[1])
    pylab.plot(Xcirc,Ycirc,color)

def poleplot(mapname,plong,plat,A95,label='',color='k',marker='o'):
    """
    This function plots a paleomagnetic pole and A95 error ellipse on whatever
    current map projection has been set using the basemap plotting library.

    Parameters
    -----------
    mapname : the name of the current map that has been developed using basemap
    plong : the longitude of the paleomagnetic pole being plotted (in degrees E)
    plat : the latitude of the paleomagnetic pole being plotted (in degrees)
    A95 : the A_95 confidence ellipse of the paleomagnetic pole (in degrees)
    label : a string that is the label for the paleomagnetic pole being plotted
    color : the color desired for the symbol and its A95 ellipse (default is 'k' aka black)
    marker : the marker shape desired for the pole mean symbol (default is 'o' aka a circle)
    """
    centerlon, centerlat = mapname(plong,plat)
    A95_km=A95*111.32
    mapname.scatter(centerlon,centerlat,20,marker=marker,color=color,label=label,zorder=101)
    equi(mapname, plong, plat, A95_km,color)

def iBootstrap(Data1,Data2,NumSims=1000):
    """
    Conduct a bootstrap test (Tauxe, 2010) for a common mean on two declination, inclination d

    This function modifies code from PmagPy for use calculating and plotting bootstrap statist
    Three plots are generated (one for x, one for y and one for z).
    If the 95 percent confidence bounds for each component overlap each other, the two directi

    Parameters
    ----------

    Data1 : a list of directional data [dec,inc]
    Data2 : a list of directional data [dec,inc]
    NumSims : number of bootstrap samples (default is 1000)
    """
    counter=0
    BDI1=pmag.di_boot(Data1)
    BDI2=pmag.di_boot(Data2)
    print ""
    print "==============="
    print ""
    print "Here are the results of the bootstrap test for a common mean"
    CDF={'X':1,'Y':2,'Z':3}
    pylab.figure(CDF['X'],figsize=(4,4),dpi=160)
    pylab.figure(CDF['Y'],figsize=(4,4),dpi=160)
    pylab.figure(CDF['Z'],figsize=(4,4),dpi=160)
    pmagplotlib.plotCOM(CDF,BDI1,BDI2,["",""])

def iWatsonV(Data1,Data2,NumSims=1000):
    """
    Conduct a Watson V test for a common mean on two declination, inclination data sets

    This function calculates Watson's V statistic from inumpyut files through Monte Carlo simu
    in order to test whether two populations of directional data could have been drawn from a
    The critical angle between the two sample mean directions and the corresponding McFadden a


    Parameters
    ----------

    Data1 : a list of directional data [dec,inc]
    Data2 : a list of directional data [dec,inc]
    NumSims : number of Monte Carlo simulations (default is 1000)
    """
    pars_1=pmag.fisher_mean(Data1)
    pars_2=pmag.fisher_mean(Data2)

    cart_1=pmag.dir2cart([pars_1["dec"],pars_1["inc"],pars_1["r"]])
    cart_2=pmag.dir2cart([pars_2['dec'],pars_2['inc'],pars_2["r"]])
    Sw=pars_1['k']*pars_1['r']+pars_2['k']*pars_2['r'] # k1*r1+k2*r2
    xhat_1=pars_1['k']*cart_1[0]+pars_2['k']*cart_2[0] # k1*x1+k2*x2
    xhat_2=pars_1['k']*cart_1[1]+pars_2['k']*cart_2[1] # k1*y1+k2*y2
    xhat_3=pars_1['k']*cart_1[2]+pars_2['k']*cart_2[2] # k1*z1+k2*z2
    Rw=numpy.sqrt(xhat_1**2+xhat_2**2+xhat_3**2)
    V=2*(Sw-Rw)
    # keep weighted sum for later when determining the "critical angle"
    # let's save it as Sr (notation of McFadden and McElhinny, 1990)
    Sr=Sw

    # do monte carlo simulation of datasets with same kappas as data,
    # but a common mean
    counter=0
    Vp=[] # set of Vs from simulations
    for k in range(NumSims):

    # get a set of N1 fisher distributed vectors with k1,
    # calculate fisher stats
        Dirp=[]
        for i in range(pars_1["n"]):
            Dirp.append(pmag.fshdev(pars_1["k"]))
        pars_p1=pmag.fisher_mean(Dirp)
    # get a set of N2 fisher distributed vectors with k2,
    # calculate fisher stats
        Dirp=[]
        for i in range(pars_2["n"]):
            Dirp.append(pmag.fshdev(pars_2["k"]))
        pars_p2=pmag.fisher_mean(Dirp)
    # get the V for these
        Vk=pmag.vfunc(pars_p1,pars_p2)
        Vp.append(Vk)

    # sort the Vs, get Vcrit (95th percentile one)

    Vp.sort()
    k=int(.95*NumSims)
    Vcrit=Vp[k]

    # equation 18 of McFadden and McElhinny, 1990 calculates the critical
    # value of R (Rwc)

    Rwc=Sr-(Vcrit/2)

    # following equation 19 of McFadden and McElhinny (1990) the critical
    # angle is calculated. If the observed angle (also calculated below)
    # between the data set means exceeds the critical angle the hypothesis
    # of a common mean direction may be rejected at the 95% confidence
    # level. The critical angle is simply a different way to present
    # Watson's V parameter so it makes sense to use the Watson V parameter
    # in comparison with the critical value of V for considering the test
    # results. What calculating the critical angle allows for is the
    # classification of McFadden and McElhinny (1990) to be made
    # for data sets that are consistent with sharing a common mean.

    k1=pars_1['k']
    k2=pars_2['k']
    R1=pars_1['r']
    R2=pars_2['r']
    critical_angle=numpy.degrees(numpy.arccos(((Rwc**2)-((k1*R1)**2)
                                               -((k2*R2)**2))/
                                              (2*k1*R1*k2*R2)))
        D1=(pars_1['dec'],pars_1['inc'])
        D2=(pars_2['dec'],pars_2['inc'])
        angle=pmag.angle(D1,D2)

        print "Results of Watson V test: "
        print ""
        print "Watson's V:           " '%.1f' %(V)
        print "Critical value of V:  " '%.1f' %(Vcrit)

        if V<Vcrit:
            print '"Pass": Since V is less than Vcrit, the null hypothesis'
            print 'that the two populations are drawn from distributions'
            print 'that share a common mean direction can not be rejected.'
        elif V>Vcrit:
            print '"Fail": Since V is greater than Vcrit, the two means can'
            print 'be distinguished at the 95% confidence level.'
        print ""
        print "M&M1990 classification:"
        print ""
        print "Angle between data set means: " '%.1f'%(angle)
        print "Critical angle for M&M1990:   " '%.1f'%(critical_angle)

        if V>Vcrit:
            print ""
        elif V<Vcrit:
            if critical_angle<5:
                print "The McFadden and McElhinny (1990) classification for"
                print "this test is: 'A'"
            elif critical_angle<10:
                print "The McFadden and McElhinny (1990) classification for"
                print "this test is: 'B'"
            elif critical_angle<20:
                print "The McFadden and McElhinny (1990) classification for"
                print "this test is: 'C'"
            else:
                print "The McFadden and McElhinny (1990) classification for"
                print "this test is: 'INDETERMINATE;"

    def lat_from_i(inc):
        """
        Calculate paleolatitude from inclination using the dipole equation
        """
        rad=numpy.pi/180.
        paleo_lat=numpy.arctan( 0.5*numpy.tan(inc*rad))/rad
        return paleo_lat

    def shoot(lon, lat, azimuth, maxdist=None):
        """
        This function enables A95 error ellipses to be drawn in basemap around paleomagnetic poles
        (from: http://www.geophysique.be/2011/02/20/matplotlib-basemap-tutorial-09-drawing-circles
        """
        glat1 = lat * numpy.pi / 180.
        glon1 = lon * numpy.pi / 180.
        s = maxdist / 1.852
```

```
faz = azimuth * numpy.pi / 180.

EPS= 0.00000000005
if ((numpy.abs(numpy.cos(glat1))<EPS) and not (numpy.abs(numpy.sin(faz))<EPS)):
    alert("Only N-S courses are meaningful, starting at a pole!")

a=6378.13/1.852
f=1/298.257223563
r = 1 - f
tu = r * numpy.tan(glat1)
sf = numpy.sin(faz)
cf = numpy.cos(faz)
if (cf==0):
    b=0.
else:
    b=2. * numpy.arctan2 (tu, cf)

cu = 1. / numpy.sqrt(1 + tu * tu)
su = tu * cu
sa = cu * sf
c2a = 1 - sa * sa
x = 1. + numpy.sqrt(1. + c2a * (1. / (r * r) - 1.))
x = (x - 2.) / x
c = 1. - x
c = (x * x / 4. + 1.) / c
d = (0.375 * x * x - 1.) * x
tu = s / (r * a * c)
y = tu
c = y + 1
while (numpy.abs (y - c) > EPS):

    sy = numpy.sin(y)
    cy = numpy.cos(y)
    cz = numpy.cos(b + y)
    e = 2. * cz * cz - 1.
    c = y
    x = e * cy
    y = e + e - 1.
    y = (((sy * sy * 4. - 3.) * y * cz * d / 6. + x) *
          d / 4. - cz) * sy * d + tu

b = cu * cy * cf - su * sy
c = r * numpy.sqrt(sa * sa + b * b)
d = su * cy + cu * sy * cf
glat2 = (numpy.arctan2(d, c) + numpy.pi) % (2*numpy.pi) - numpy.pi
c = cu * cy - su * sy * cf
x = numpy.arctan2(sy * sf, c)
c = ((-3. * c2a + 4.) * f + 4.) * c2a * f / 16.
d = ((e * cy * c + cz) * sy * c + y) * sa
glon2 = ((glon1 + x - (1. - c) * d * f + numpy.pi) % (2*numpy.pi)) - numpy.pi

baz = (numpy.arctan2(sa, b) + numpy.pi) % (2 * numpy.pi)

glon2 *= 180./numpy.pi
```

```
    glat2 *= 180./numpy.pi
    baz *= 180./numpy.pi

    return (glon2, glat2, baz)

def equi(m, centerlon, centerlat, radius, color):
    """
    This function enables A95 error ellipses to be drawn in basemap around paleomagnetic poles
    (from: http://www.geophysique.be/2011/02/20/matplotlib-basemap-tutorial-09-drawing-circles,
    """
    glon1 = centerlon
    glat1 = centerlat
    X = []
    Y = []
    for azimuth in range(0, 360):
        glon2, glat2, baz = shoot(glon1, glat1, azimuth, radius)
        X.append(glon2)
        Y.append(glat2)
    X.append(X[0])
    Y.append(Y[0])

    X,Y = m(X,Y)
    plt.plot(X,Y,color)

def VGP_calc(dataframe):
    """
    This function calculates paleomagnetic poles from directional data within a pandas.DataFra

    Parameters
    -----------
    dataframe : the name of the pandas.DataFrame containing the data
    dataframe['site_lat'] : the latitude of the site
    dataframe['site_long'] : the longitude of the site
    dataframe['inc_tc'] : the tilt-corrected inclination
    dataframe['dec_tc'] : the tilt-corrected declination
    """
    #calculate the paleolatitude/colatitude
    dataframe['paleolatitude']=np.degrees(np.arctan(0.5*np.tan(np.radians(dataframe['inc_tc'])
    dataframe['colatitude']=90-dataframe['paleolatitude']
    #calculate the latitude of the pole
    dataframe['pole_lat']=np.degrees(np.arcsin(np.sin(np.radians(dataframe['site_lat']))*
                                                           np.cos(np.radians(dataframe['colatitu
                                                           np.cos(np.radians(dataframe['site_lat
                                                           np.sin(np.radians(dataframe['colatitu
                                                           np.cos(np.radians(dataframe['dec_tc']
    #calculate the longitudinal difference between the pole and the site (beta)
    dataframe['beta']=np.degrees(np.arcsin((np.sin(np.radians(dataframe['colatitude']))*
                                     np.sin(np.radians(dataframe['dec_tc'])))/
                                    (np.cos(np.radians(dataframe['pole_lat'])))))
    #generate a boolean array (mask) to use to distinguish between the two possibilities for p
    #and then calculate pole longitude using the site location and calculated beta
    mask = np.cos(np.radians(dataframe['colatitude']))>np.sin(np.radians(dataframe['site_lat']
    dataframe['pole_long']=np.where(mask,(dataframe['site_long']+dataframe['beta'])%360.,(data
    #calculate the antipode of the poles
```

```
dataframe['pole_lat_rev']=-dataframe['pole_lat']
dataframe['pole_long_rev']=(dataframe['pole_long']-180.)%360.
#the 'colatitude' and 'beta' columns were created for the purposes of the pole calculation
#but aren't of further use and are deleted
del dataframe['colatitude']
del dataframe['beta']
```