Chapter 6. Finding a pole of relative motion

6.1 Introduction

Plates have been in motion throughout most of Earth history, and almost certainly for the past 2.5-3 billion years (Condie and Pease, 2008). The specific plates that existed in the early Precambrian bear no geometric resemblance to the plates of the current Earth system. In this treatment of plate kinematics, we will generally restrict ourselves to the portion of Earth history from the oldest oceanic crust still in an active ocean basin (~170 Ma in the western Pacific; Lancelot and Larsen, 1990) to the present.

All motion is discerned and described relative to a specific reference frame. Plate motions might be described relative to another plate, or relative to a reference system external to any individual plate as might be provided by GPS, or relative to a reference frame in which the lithosphere undergoes no net rotation, or relative to one or more sub-lithospheric hotspots (ignoring for the moment that we don't have a consensus understanding of hotspots). We will investigate these options further as our work progresses. All motion is relative. Hence, using the phrase "relative plate motion" or "relative velocity" is about as uninformative as the phrase "age date." If we are to use the word "relative" in our work in kinematics, it should be in the service of identifying the reference frame we are using to discern motion, as in "the motion of plate A relative to plate B."

"Instantaneous" motion in the context of plate kinematics can mean motion detected over the course of years using technologies like the Global Positioning System (GPS), very-long baseline interferometry (VLBI), or satellite laser rangefinding (SLR), or it can mean motions within the past ~3 Ma interpreted from the reconstruction of marine magnetic anomalies or other geological displacement markers along plate boundaries. "Finite" motion generally involves motion over longer time intervals -- millions of years or more.

We learned in an earlier chapter that the displacement of a rigid body on a sphere between some initial position and any other position can be described as a rotation around an axis that passes through the center of the sphere. The places where the axis intersects the sphere are called the *poles of rotation*. If the motion is anti-clockwise around that pole, as observed from outside the sphere looking toward its center along the rotational axis, the motion is considered a positive motion and the pole is called the *positive pole*. Otherwise, it is called the negative pole (or sometimes the antipole, if the word "pole" has been reserved for use in describing a positive rotation).

Given a suitable frame of reference from which to observe the motion, we can describe an instantaneous (or infinitesimal) motion around an *instantaneous pole of rotation*. The usual way we describe this motion is to provide the latitude and longitude of the positive pole, an instantaneous angular speed, and information about the frame of reference and the boundaries of the plate that is moving relative to that reference frame. I might assert that instantaneous motion information is an essential prerequisite for understanding the finite motion of plates before and after the present-day.

Finite poles of rotation are properly used in two ways. Bullard, Everett and Smith (1965) sought to define the pole around which Africa and South America can be rotated in order to bring the two continents together as they were before rifting began ~150 Ma. This kind of pole is often called a *total-opening pole*. We can also use marine magnetic anomalies to reconstruct the evolution of an ocean basin using *stage poles* that generally reflect motion that occurs during a relatively short time interval whose boundaries are well marked by distinctive magnetic reversals.

When working with finite poles of rotation, there is no implication that the two plates actually moved around a fixed pole from their initial to their final relative locations. Imagine that you are doing your best to photograph a baseball pitcher throwing the ball, but all you manage to capture is a photo as the ball leaves his fingertips and another as the ball hits the catcher's mitt. The simplest path between the release point and the catcher's mitt is a straight line, but no pitcher throws a baseball along a linear path. One reason is that it is physically impossible given that the pitcher is throwing a 5-ounce, 2.9-inch diameter ball with raised seams through air in a gravitational field at less than 105 miles an hour over a distance of about 60 feet. The other reason is that pitchers exert a great deal of effort to induce the ball to move in a deceptive way as it travels toward the batter. The actual path of motion or *trajectory* of the baseball, as observed in a reference frame fixed to the baseball field and the photographer, is not a straight line. Refocusing on plate kinematics, the shortest and simplest path for finite motion on the surface of a sphere between known initial and final points is a great circle, but that is generally not the actual path of finite motion for a point on a plate, as observed from the reference frames we often use.

In this chapter, we will learn how to define an instantaneous pole of rotation for a specific plate relative to a frame of reference external to the plate, given instantaneous tangential velocity vectors for two or more different points on the plate. We will also define a total-displacement pole of rotation for two adjacent plates given two or more pairs of points (one point from each pair on each plate) that were once adjacent to each other.

6.2 Some user-defined functions

We will use some of the user-defined functions developed in previous chapters.

```
In[1]:= convert2Cart[lat_, long_] := {Cos[lat Degree] Cos[long Degree],
          Cos[lat Degree] Sin[long Degree], Sin[lat Degree]};

In[2]:= unitVect3D[vect_] :=
        {(vect[[1]] / Norm[vect]), (vect[[2]] / Norm[vect]), (vect[[3]] / Norm[vect])};

In[3]:= zRotation[angle_] := {{Cos[(angle) Degree], -Sin[(angle) Degree], 0},
          {Sin[(angle) Degree], Cos[(angle) Degree], 0}, {0, 0, 1}};

In[4]:= circMotion[x_, m1_, angleIn_] := Module[{m2, m3, answer}, m2 = zRotation[angleIn];
          m3 = Inverse[m1];
          answer = m3.m2.m1.x;
          answer];
```

The user-defined function **findGeogCoord** takes a unit location vector and determines the latitude and longitude associated with the input Cartesian coordinates.

```
In[5]:= findGeogCoord[vect_] := Module[{lat, long, a, b, c, d, e, f}, a = ArcSin[vect[[3]]];
          b = {vect[[1]], vect[[2]], 0};
          c = If[((Abs[vect[[1]]] < (1 × 10^-14)) && (Abs[vect[[2]]] < (1 × 10^-14))),
            {1, 1, 0}, {vect[[1]] / Norm[b], vect[[2]] / Norm[b], 0}];
          d = {1, 0, 0};
          e = VectorAngle[c, d];
          f = If[(vect[[2]] < 0), (-e), (e)];
          lat = a (180 / π);
          long = If[
            ((Abs[vect[[1]]] < (1 × 10^-14)) && (Abs[vect[[2]]] < (1 × 10^-14))), 0, (f (180 / π))];
          {lat, long}];
```

The user-defined function **fourColorDotSphere** plots four sets of 3D unit location vectors with common origins at the center of a unit sphere. The color of the dots in each set is determined by the position in the function's argument, so the first set is green, second is blue, third is red and fourth is black.

```
In[6]:= fourColorDotSphere[green_, blue_, red_, black_] :=
        Module[{element1, element2, element3, element4, element5, answer},
         element1 = Graphics3D[Sphere[{0, 0, 0}, 1], AspectRatio → 1,
           BoxRatios → {1, 1, 1}, PlotRange → All, PlotRangePadding → 0.1,
           ColorOutput → GrayLevel, Lighting → "Neutral"];
         element2 = ListPointPlot3D[green, AspectRatio → 1, BoxRatios → {1, 1, 1},
           PlotStyle → Green, PlotRange → All, PlotRangePadding → 0.1];
         element3 = ListPointPlot3D[blue, AspectRatio → 1, BoxRatios → {1, 1, 1},
           PlotStyle → Blue, PlotRange → All, PlotRangePadding → 0.1];
         element4 = ListPointPlot3D[red, AspectRatio → 1, BoxRatios → {1, 1, 1},
           PlotStyle → Red, PlotRange → All, PlotRangePadding → 0.1];
         element5 = ListPointPlot3D[black, AspectRatio → 1, BoxRatios → {1, 1, 1},
           PlotStyle → Black, PlotRange → All, PlotRangePadding → 0.1];
         answer = Show[element1, element2, element3, element4, element5];
         answer];
```


6.3 Instantaneous pole from two instantaneous velocity vectors

We need the instantaneous tangential velocities of at least two different points on the same plate to determine the pole of instantaneous relative velocity for that plate. In this section, we will learn how to find the relative pole with the minimum amount of input data.

First, we establish appropriate variables using the input data. The location of point 1 is latitude 34.5°N, longitude 15°E, and point 2 is at latitude 40°N, longitude 25°E. The instantaneous tangential velocity vector of point 1 is directed toward an azimuth of 80°, and point 2 is moving toward 75°.

In[7]:=
```
lat1 = 34.5;
```

In[8]:=
```
long1 = 15;
```

In[9]:=
```
az1 = 80;
```

In[10]:=
```
lat2 = 40;
```

In[11]:=
```
long2 = 25;
```

In[12]:=
```
az2 = 75;
```

We define the unit vectors along the positive X, Y and Z axes of the Cartesian geographic coordinate system.

In[13]:=
```
xGeog = {1, 0, 0};
```

In[14]:=
```
yGeog = {0, 1, 0};
```

In[15]:=
```
zGeog = {0, 0, 1};
```

We convert the input location data for point 1 and point 2 into unit vectors in the Cartesian geographic coordinate system, yielding vectors **point1** and **point2**.

In[16]:=
```
point1 = convert2Cart[lat1, long1];
```

In[17]:=
```
point2 = convert2Cart[lat2, long2];
```

For each of the two reference points, we define a separate coordinate system. Unit vectors along the positive coordinate axes for point 1 are called **xPt1**, **yPt1** and **zPt1**. The **zPt1** vector is the same as vector **point1**. The **xPt1** axis is found by taking the cross product of **zPt1** with the geographic Z axis at the north pole: **zGeog**. So as you are looking down on point 1 with the **zPt1** axial vector pointing right at you, the **xPt1** axial vector is pointing to the left, **yPt1** is pointing down, and the negative **yPt1** axis is pointing up toward the north.

In[18]:=
```
zPt1 = point1;
```

In[19]:=
```
xPt1 = unitVect3D[Cross[zPt1, zGeog]];
```

In[20]:=
```
yPt1 = unitVect3D[Cross[zPt1, xPt1]];
```

In[21]:=
```
negYPt1 = -yPt1;
```

The **xPt2**, **yPt2** and **zPt2** axes are defined in a similar way.

In[22]:=
```
zPt2 = point2;
```

In[23]:=
```
xPt2 = unitVect3D[Cross[zPt2, zGeog]];
```

In[24]:=
```
yPt2 = unitVect3D[Cross[zPt2, xPt2]];
```

In[25]:=
```
negYPt2 = -yPt2;
```

The transformation matrices from the Cartesian geographic coordinate system and the coordinate systems defined above are as follows:

In[26]:=
```
jPt1 = {{Dot[xGeog, xPt1], Dot[yGeog, xPt1], Dot[zGeog, xPt1]},
        {Dot[xGeog, yPt1], Dot[yGeog, yPt1], Dot[zGeog, yPt1]},
        {Dot[xGeog, zPt1], Dot[yGeog, zPt1], Dot[zGeog, zPt1]}};
```

In[27]:=
```
jPt2 = {{Dot[xGeog, xPt2], Dot[yGeog, xPt2], Dot[zGeog, xPt2]},
        {Dot[xGeog, yPt2], Dot[yGeog, yPt2], Dot[zGeog, yPt2]},
        {Dot[xGeog, zPt2], Dot[yGeog, zPt2], Dot[zGeog, zPt2]}};
```

Two non-colinear vectors tangent to the same surface at the same point are elements of the tangent plane to that surface at that point. So if

one of those tangent vectors defined at point 1 is pointing north and the other is the tangential velocity vector, we can use plane geometry to determine the direction point 1 is moving relative to north. Angle **theta1** is the angle between a north-directed tangential vector and the instantaneous tangential velocity vector at point 1. Angle **theta2** is defined in a similar way.

```
In[28]:= theta1 = If[az1 > 180, 360 - az1, -az1];
```

```
In[29]:= theta2 = If[az2 > 180, 360 - az2, -az2];
```

The unit vector **normal1** is perpendicular to vector **point1** and makes an angle of **az1** (the azimuth of the instantaneous tangential velocity vector at point 1) with the **negYPt1** axis. Looking down from above point 1, the **normal1** vector (whose origin is at the center of Earth) is parallel to the instantaneous tangential velocity vector at point 1 (whose origin is at point 1).

```
In[30]:= normal1 = circMotion[negYPt1, jPt1, theta1];
```

Similarly, the unit vector **normal2** is perpendicular to vector **point12** and makes an angle of **az2** with the **negYPt2** axis. Looking down from above point 2, the **normal2** vector is parallel to the instantaneous tangential velocity vector at point 2.

```
In[31]:= normal2 = circMotion[negYPt2, jPt2, theta2];
```

The axis of instantaneous relative motion is coincident with the vector result of the cross product **normal1** × **normal2**. The intersection of that axis with the surface of the unit sphere is pole A and its antipode, -pole A

```
In[32]:= poleA = unitVect3D[Cross[normal1, normal2]]
Out[32]= {0.278077, -0.0432362, 0.959585}
```

```
In[33]:= findGeogCoord[poleA]
Out[33]= {73.6552, -8.83775}
```

```
In[34]:= findGeogCoord[-poleA]
Out[34]= {-73.6552, 171.162}
```

Verify that pole A and -pole A are at an azimuth from point 1 that is 90° from the azimuth of the instantaneous velocity vector at point 1. The azimuth of pole A from point 1 is

```
In[35]:= pt1Vect1 = unitVect3D[Cross[point1, zGeog]];
```

```
In[36]:= pt1Vect2 = unitVect3D[Cross[point1, poleA]];
```

```
In[37]:= anglePt1A = VectorAngle[pt1Vect1, pt1Vect2] (180 / π);
```

```
In[38]:= azPoleA1 =
          If[(VectorAngle[poleA, pt1Vect1] (180 / π)) < 90, (360 - anglePt1A), anglePt1A]
Out[38]= 350.
```

```
In[39]:= VectorAngle[point1, poleA] (180 / π)
Out[39]= 40.9174
```

The azimuth of -pole A from point 1 is

```
In[40]:= pt1Vect3 = unitVect3D[Cross[point1, -poleA]];
```

```
In[41]:= anglePt1B = VectorAngle[pt1Vect1, pt1Vect3] (180 / π);
```

```
In[42]:= azPoleB1 =
          If[(VectorAngle[-poleA, pt1Vect1] (180 / π)) < 90, (360 - anglePt1B), anglePt1B]
Out[42]= 170.
```

```
In[43]:= VectorAngle[point1, -poleA] (180 / π)
Out[43]= 139.083
```

The azimuth of the instantaneous velocity vector at point 1 is


```
In[44]:= N[az1]
Out[44]= 80.
```

Verify that pole A and -pole A are at an azimuth from point 2 that is 90° from the azimuth of the instantaneous velocity vector at point 2.

```
In[45]:= pt2Vect1 = unitVect3D[Cross[point2, zGeog]];
In[46]:= pt2Vect2 = unitVect3D[Cross[point2, poleA]];
In[47]:= anglePt2A = VectorAngle[pt2Vect1, pt2Vect2] (180 / π);
In[48]:= azPoleA2 =
          If[(VectorAngle[poleA, pt2Vect1] (180 / π)) < 90, (360 - anglePt2A), anglePt2A]
Out[48]= 345.
In[49]:= VectorAngle[point2, poleA] (180 / π)
Out[49]= 37.2622
```

The azimuth of -pole A from point 2 is

```
In[50]:= pt2Vect3 = unitVect3D[Cross[point2, -poleA]];
In[51]:= anglePt2B = VectorAngle[pt2Vect1, pt2Vect3] (180 / π);
In[52]:= azPoleB2 =
          If[(VectorAngle[-poleA, pt2Vect1] (180 / π)) < 90, (360 - anglePt2B), anglePt2B]
Out[52]= 165.
In[53]:= VectorAngle[point2, -poleA] (180 / π)
Out[53]= 142.738
```

The azimuth of the instantaneous velocity vector at point 2 is

```
In[54]:= N[az2]
Out[54]= 75.
```

6.4 Boiling out the fat, leaving the meat, part 1

The preceding sea of text and code can be boiled down to the following code stream that takes input data for two instantaneous vectors on the same plate and computes a relative-motion pole.

Input

```
In[55]:= lat1 = 34.5;
In[56]:= long1 = 15;
In[57]:= az1 = 80;
In[58]:= lat2 = 40;
In[59]:= long2 = 25;
In[60]:= az2 = 75;
```

Computation

```
In[61]:= xGeog = {1, 0, 0};
In[62]:= yGeog = {0, 1, 0};
```



```
In[63]:= zGeog = {0, 0, 1};

In[64]:= point1 = convert2Cart[lat1, long1];

In[65]:= point2 = convert2Cart[lat2, long2];

In[66]:= zPt1 = point1;

In[67]:= xPt1 = unitVect3D[Cross[zPt1, zGeog]];

In[68]:= yPt1 = unitVect3D[Cross[zPt1, xPt1]];

In[69]:= negYPt1 = -yPt1;

In[70]:= zPt2 = point2;

In[71]:= xPt2 = unitVect3D[Cross[zPt2, zGeog]];

In[72]:= yPt2 = unitVect3D[Cross[zPt2, xPt2]];

In[73]:= negYPt2 = -yPt2;

In[74]:= jPt1 = ( Dot[xGeog, xPt1]  Dot[yGeog, xPt1]  Dot[zGeog, xPt1]
                  Dot[xGeog, yPt1]  Dot[yGeog, yPt1]  Dot[zGeog, yPt1]
                  Dot[xGeog, zPt1]  Dot[yGeog, zPt1]  Dot[zGeog, zPt1] );

In[75]:= jPt2 = ( Dot[xGeog, xPt2]  Dot[yGeog, xPt2]  Dot[zGeog, xPt2]
                  Dot[xGeog, yPt2]  Dot[yGeog, yPt2]  Dot[zGeog, yPt2]
                  Dot[xGeog, zPt2]  Dot[yGeog, zPt2]  Dot[zGeog, zPt2] );

In[76]:= theta1 = If[az1 > 180, 360 - az1, -az1];

In[77]:= theta2 = If[az2 > 180, 360 - az2, -az2];

In[78]:= normal1 = circMotion[negYPt1, jPt1, theta1];

In[79]:= normal2 = circMotion[negYPt2, jPt2, theta2];
```

Output

```
In[80]:= poleA = unitVect3D[Cross[normal1, normal2]]

Out[80]= {0.278077, -0.0432362, 0.959585}

In[81]:= findGeogCoord[poleA]

Out[81]= {73.6552, -8.83775}

In[82]:= findGeogCoord[-poleA]

Out[82]= {-73.6552, 171.162}
```

The rotational poles are at latitude 73.7°N, longitude 8.8°W, and latitude 73.7°S, longitude 171.2°E.

6.5 Instantaneous pole from more than two instantaneous velocity vectors

When you have two tangential instantaneous velocity vectors that are not parallel to each other, you can define a single axis of relative motion. When you have three, you can define one possible axis for each pair of input velocities, or three axes in all. In other words, the system is overdetermined when you have more than two tangential velocity vectors. Four tangential velocity vectors correspond to six axes. Before this recitation gets any more boring, the number of axial solutions associated with n tangential velocity vectors (where $n \geq 2$) is given by

$$\sum_{i=1}^{n-1} (n/2)$$

If the crust between the points is perfectly rigid, and the tangential velocities are perfectly accurate, then the various axial computations will yield the same axis solution over and over. In real life, the axial solutions will be similar but not the same. The simplest way to address the diversity of answers is to compute the mean unit vector by adding together all of the unit vectors directed approximately toward pole A (or

all directed approximately at -pole A), and then estimate the uncertainty using Fisher statistics (Fisher, 1953; Cronin, 2008) or some other appropriate statistical technique (see Batschelet, 1981; Borradaile, 2003; Fisher and others, 1987; Mardia, 1972).

Let us imagine that we have five different points on a given plate, each with an instantaneous tangential velocity vector. Those five velocities correspond to 10 axial solutions. Below, we will compute the average and 95% confidence interval of those 10 unit vectors using Fisher statistics.

In[83]:=
```
inData = {{0.278077, -0.0432362, 0.95958}, {0.272955, -0.0383613, 0.961262}, {0.238247, -0.0420093, 0.970296}, {0.307324, -0.0323011, 0.951057}, {0.273583, -0.0335917, 0.961262}, {0.272244, -0.0431192, 0.961262}, {0.173225, -0.0121131, 0.984808}, {0.18846, -0.0298491, 0.981627}, {0.304322, -0.0536602, 0.951057}, {0.292194, -0.0102036, 0.956305}};
```

Define the sample size

In[84]:=
```
matrixSize1 = Dimensions[inData];
```

Variable **noInputVectors** is the number of vectors in the set. It is equivalent to the variable "N" in Cronin (2008).

In[85]:=
```
noInputVectors = matrixSize1[[1]];
```

Variable **meanDirCos** includes the set of three direction cosines for the mean axial vector (equation 2 from Cronin, 2008).

In[86]:=
```
meanDirCos = Total[inData, 1];
```

Variable **capR** is the length of the mean axial vector (equation 3 from Cronin, 2008). It is equivalent to the variable "R" in Cronin (2008).

In[87]:=
```
capR = Norm[meanDirCos]
```

Out[87]=
```
9.98893
```

Variable **meanUnitVector** is the set of coordinates for the mean axial vector (equation 4 from Cronin, 2008).

In[88]:=
```
meanUnitVector = unitVect3D[meanDirCos];
```

In[89]:=
```
geogCoord = findGeogCoord[meanUnitVector];
```

Variable **k** is the precision parameter, which ranges from 0 for a vector set that is strongly noncolinear to infinity for vectors that are perfectly colinear, as when N = R (equation 7 from Cronin, 2008).

In[90]:=
```
k = (noInputVectors - 1) / (noInputVectors - capR);
```

Variable **alpha95** is the angular radius of the 95% confidence cone around the mean vector, in radians (equation 8 from Cronin, 2008). It is equivalent to the variable "α_{95}" in Cronin (2008).

In[91]:=
```
alpha95 = ArcCos[1 - ((noInputVectors - capR) / capR) ((1/0.05)^(1/(noInputVectors-1)) - 1))];
```

To change the previous equation to compute the alpha90 radius, you would change the 0.05 to 0.10. Finally, we round numerical output to integers using the built-in *Mathematica* function **Round**, which returns the nearest integer (nearest even integer for numbers of the form *x*.5).

In[92]:=
```
alphaNinetyFive = Round[alpha95 (180 / π)];
```

In[93]:=
```
precisionParameterK = Round[k];
```

Output

Other than the coordinates of the mean unit vector, the numerical output below is rounded to the nearest integer value. All angles and azimuths are expressed in degrees.

In[94]:=
```
noInputVectors
```

Out[94]=
```
10
```


Mean unit location vector to the rotational pole

```
In[95]:= meanUnitVector
Out[95]= {0.260351, -0.033882, 0.964919}
```

Latitude of mean pole

```
In[96]:= Round[geogCoord[[1]]]
Out[96]= 75
```

Longitude of mean pole

```
In[97]:= Round[geogCoord[[2]]]
Out[97]= -7
In[98]:= alphaNinetyFive
Out[98]= 2
In[99]:= precisionParameterK
Out[99]= 813
```

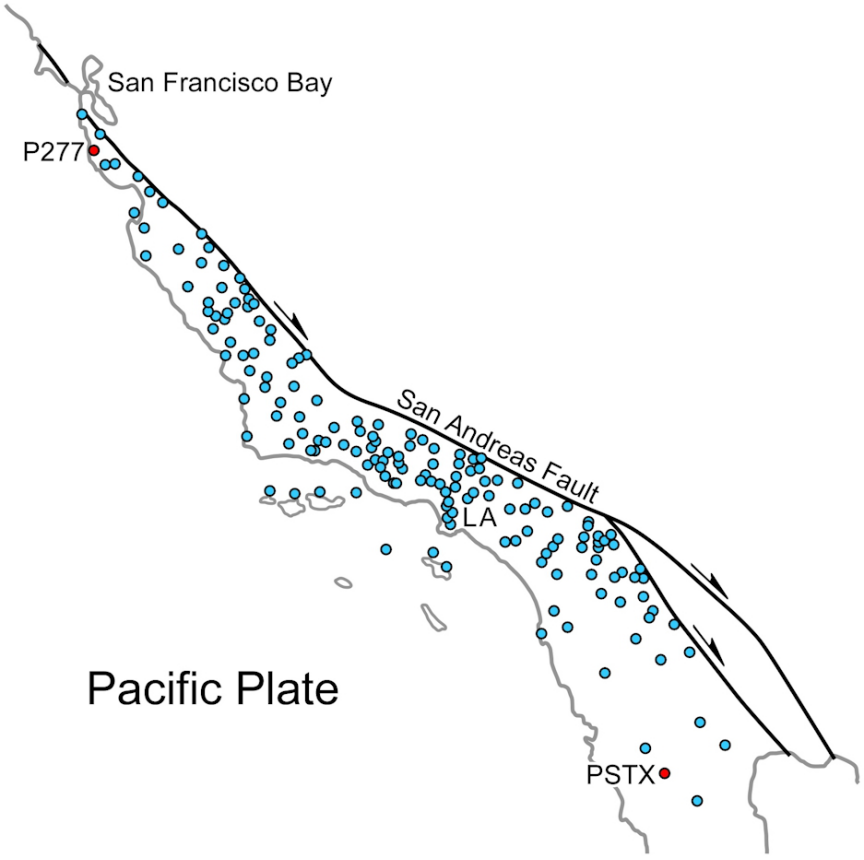


Figure 6-1. Plate Boundary Observatory points located west of the San Andreas fault (blue and red dots). Data are from www.unavco.org/instrumentation/networks/status/pbo.

Exercise 6.1 Approximately 1100 permanent GPS stations were established between 2003 and 2008 in the continental US and Alaska, along with a few in Mexico and Canada, to form the bulk of the Plate Boundary Observatory (PBO). The PBO is an important element of the EarthScope Project. Individual PBO sites collect location information at a rate of 1, 2, 5 or 10 Hz. Data generated by these stations are available in a variety of raw and processed formats at no cost via the web.

The PBO Network Monitoring map shows the location of all of the PBO GPS sites (www.unavco.org/instrumentation/networks/status/pbo). Clicking on the blue circle for a given site opens a small window that provides a little bit of information about the site, as well as a link to the site overview page. Site velocities are provided on the overview page as north/south, east/west and up/down components expressed in either a reference frame that is fixed to the relatively stable interior of North America (NAM08) or to the global IGS 2008 reference frame (IGS08). The GPS Velocity Viewer provides a summary of total-horizontal-velocity vectors for PBO sites in an easy-to-use format (www.unavco.org/software/ visualization/GPS-Velocity-Viewer/GPS-Velocity-Viewer.html).

Go to the overview pages for PBO station P277 near Pescadero, California (www.unavco.org/ instrumentation/networks/status/p-bo/overview/P277) and station PSTX near San Isidoro, Mexico (www.unavco.org/instrumentation/networks/status/ pbo/overview/P-STX). As you can verify using the GPS Velocity Viewer, both of these stations are on the Pacific plate, just west of the plate boundary. For each station, find the north-south and east-west velocities relative to the NAM08 reference frame. These velocities and their associated 1-σ uncertainties are included as part of the time-series graphs. Use those component velocities to determine the azimuth and tangential velocity of the sites' instantaneous motion relative to the stable interior of North America.

$$\text{total tangential velocity} = \sqrt{(\text{east velocity})^2 + (\text{north velocity})^2}$$

Do the same for three other PBO GPS sites of your choosing that are all located west of the plate boundary in California and Baja Mexico. Use these data to estimate the location of the instantaneous pole of relative motion between the Pacific and North American Plates, employing what you learned in sections 6.4 and 6.5 above.

6.6 Finite (total displacement) pole from displaced points on adjacent plates

This is easy, and we will start with an example for which we know the answer. We will imagine a pair of adjacent plates in which one plate has rotated 4° away from the other around the north pole. Two reference points on the western plate are called point 1w and point 2w, and the corresponding two reference points on the eastern plate are called point 1e and point 2e. Their current locations, after the 4° rotation, are given below:

```
In[100]:= latPt1w = 30;
In[101]:= longPt1w = -22;
In[102]:= latPt1e = 30;
In[103]:= longPt1e = -18;
In[104]:= latPt2w = 40;
In[105]:= longPt2w = -22;
In[106]:= latPt2e = 40;
In[107]:= longPt2e = -18;
```

The locations of points 1w and 1e were chosen because the great-circle arc at longitude -20 (20°W longitude) is half-way between these two points. Every point along that great-circle arc is an equal distance from points 1w and 1e, and that great-circle arc passes through the rotational axis of the 2-plate system which is, in this case, coincident with the north pole.

Similarly, the locations of points 2w and 2e were chosen because the great-circle arc at longitude -10 is half-way between these two points and passes through the rotational axis. In fact, the great-circle arc that is half-way between points 1w and 1e (longitude -20) intersects the great-circle arc that is half-way between points 2w and 2e (longitude -10) at the rotational pole (the north pole). So we already know that the rotational pole has Cartesian coordinates of {0, 0, 1} corresponding to geographic coordinates (latitude, longitude) of {90°, undefined). The longitude of the north pole is undefined because all longitudes merge at that point.

We start by converting the input geographic coordinates to the Cartesian geographic coordinate system

```
In[108]:= pt1w = convert2Cart[latPt1w, longPt1w];
In[109]:= pt2w = convert2Cart[latPt2w, longPt2w];
In[110]:= pt1e = convert2Cart[latPt1e, longPt1e];
In[111]:= pt2e = convert2Cart[latPt2e, longPt2e];
```


We find the unit vector **meanPt1** that is exactly half-way between point 1w and point 1e in the **pt1w**-**pt1e** plane. We do this by finding the unit vector of the sum of **pt1w** and **pt1e**. In a similar manner, we find unit vector **meanPt2**.

```
In[112]:= meanPt1 = N[unitVect3D[(pt1w + pt1e) / 2]];
In[113]:= meanPt2 = N[unitVect3D[(pt2w + pt2e) / 2]];
```

The unit vector that is normal to the great-circle arc between point 1w and point 1e is called **normalPt1**. Vector **normalPt1** is found by taking the unit vector of the result of **meanPt1** × (**pt1w** × **pt1e**). Similarly, vector **normalPt1** is found by taking the unit vector of the result of **meanPt2** × (**pt2w** × **pt2e**).

```
In[114]:= normalPt1 = unitVect3D[Cross[meanPt1, Cross[pt1w, pt1e]]];
In[115]:= normalPt2 = unitVect3D[Cross[meanPt2, Cross[pt2w, pt2e]]];
```

The unit vector to the rotation pole, rotPole, is computed by finding the unit vector of the cross product **normalPt1** × **normalPt2**.

```
In[116]:= rotPole = unitVect3D[Cross[normalPt1, normalPt2]]
Out[116]= {0.780224, -0.283978, 0.557321}
In[117]:= rotPoleGeog = findGeogCoord[rotPole]
Out[117]= {33.8707, -20.}
In[118]:= rotAntipoleGeog = findGeogCoord[-rotPole]
Out[118]= {-33.8707, 160.}
```

The result of this process is that we discovered that the rotational pole has a latitude of 90° and a longitude that is undefined (arbitrarily set to 0), which is the location of the north pole.

The angle of rotation in this example is probably obvious to you, but let's work through the problem to gain insight that we will need later. The angle of rotation is the same as the dihedral angle between two planes: the plane defined by vectors pt1e and rotPole, and the plane defined by pt1w and rotPole. The angle between these two planes is the angle between the vectors normal to each of the two planes.

```
In[119]:= normalPt1ePole = Cross[pt1e, rotPole];
In[120]:= normalPt1wPole = Cross[pt1w, rotPole];
In[121]:= rotAnglePt1 = VectorAngle[normalPt1ePole, normalPt1wPole] (180 / π);
In[122]:= N[%]
Out[122]= 48.425
```

We can (an generally must) do the same thing for point 2, although if the distance between point 1e and point 2e is exactly the same as the distance between point 1w and point 2w, we will get the same answer. In the more general case in which those distances are a little bit different, it is best to measure the two angles and take their average.

```
In[123]:= normalPt2ePole = Cross[pt2e, rotPole];
In[124]:= normalPt2wPole = Cross[pt2w, rotPole];
In[125]:= rotAnglePt2 = VectorAngle[normalPt2ePole, normalPt2wPole] (180 / π);
In[126]:= N[%]
Out[126]= 28.0489
In[127]:= rotAngle = Mean[{rotAnglePt1, rotAnglePt2}];
In[128]:= N[rotAngle]
Out[128]= 38.237
```

6.7 Boiling out the fat, leaving the meat, part 2

Now we will boil-out the fat in the preceding code, and find the pole associated with two other sets of points on two other plates. We don't

know where the pole is to begin with, but you can be confident that it is not the north pole. We will assign the following names to our reference points on imaginary plates a and b: point 1a and 1b, and point 2a and 2b.

Input

```
In[129]:= latPt1a = 30;
In[130]:= longPt1a = -5;
In[131]:= latPt1b = 31.4427;
In[132]:= longPt1b = -1.0257;
In[133]:= latPt2a = 10;
In[134]:= longPt2a = -25;
In[135]:= latPt2b = 10.5863;
In[136]:= longPt2b = -20.136;
```

Computation

```
In[137]:= pt1a = convert2Cart[latPt1a, longPt1a];
In[138]:= pt2a = convert2Cart[latPt2a, longPt2a];
In[139]:= pt1b = convert2Cart[latPt1b, longPt1b];
In[140]:= pt2b = convert2Cart[latPt2b, longPt2b];
In[141]:= meanPt1 = N[unitVect3D[pt1a + pt1b]];
In[142]:= meanPt2 = N[unitVect3D[pt2a + pt2b]];
In[143]:= normalPt1 = unitVect3D[Cross[meanPt1, Cross[pt1a, pt1b]]];
In[144]:= normalPt2 = unitVect3D[Cross[meanPt2, Cross[pt2a, pt2b]]];
In[145]:= rotPole = unitVect3D[Cross[normalPt1, normalPt2]];
In[146]:= rotPoleGeog = findGeogCoord[rotPole];
In[147]:= rotAntipoleGeog = findGeogCoord[-rotPole];
In[148]:= rotAngle1 = VectorAngle[Cross[pt1a, rotPole], Cross[pt1b, rotPole]] (180 / π);
In[149]:= rotAngle2 = VectorAngle[Cross[pt2a, rotPole], Cross[pt2b, rotPole]] (180 / π);
In[150]:= rotAngle = ((rotAngle1 + rotAngle2) / 2);
```

Output

```
In[151]:= N[rotPole]
Out[151]= {-0.371882, 0.260396, -0.89101}
In[152]:= N[rotPoleGeog]
Out[152]= {-63.0004, 145.}
In[153]:= N[rotAntipoleGeog]
Out[153]= {63.0004, -35.0001}
In[154]:= N[rotAngle]
Out[154]= 5.99993
```


Given the input data, the pole and antipole of rotation are located at latitude 63°S, longitude 145°E, and latitude 63°N, longitude 35°W. The rotation angle was 32°.

Assignment. Read Bullard, Everett and Smith (1965), Moulin and others (2010) and Seton and others (2012).

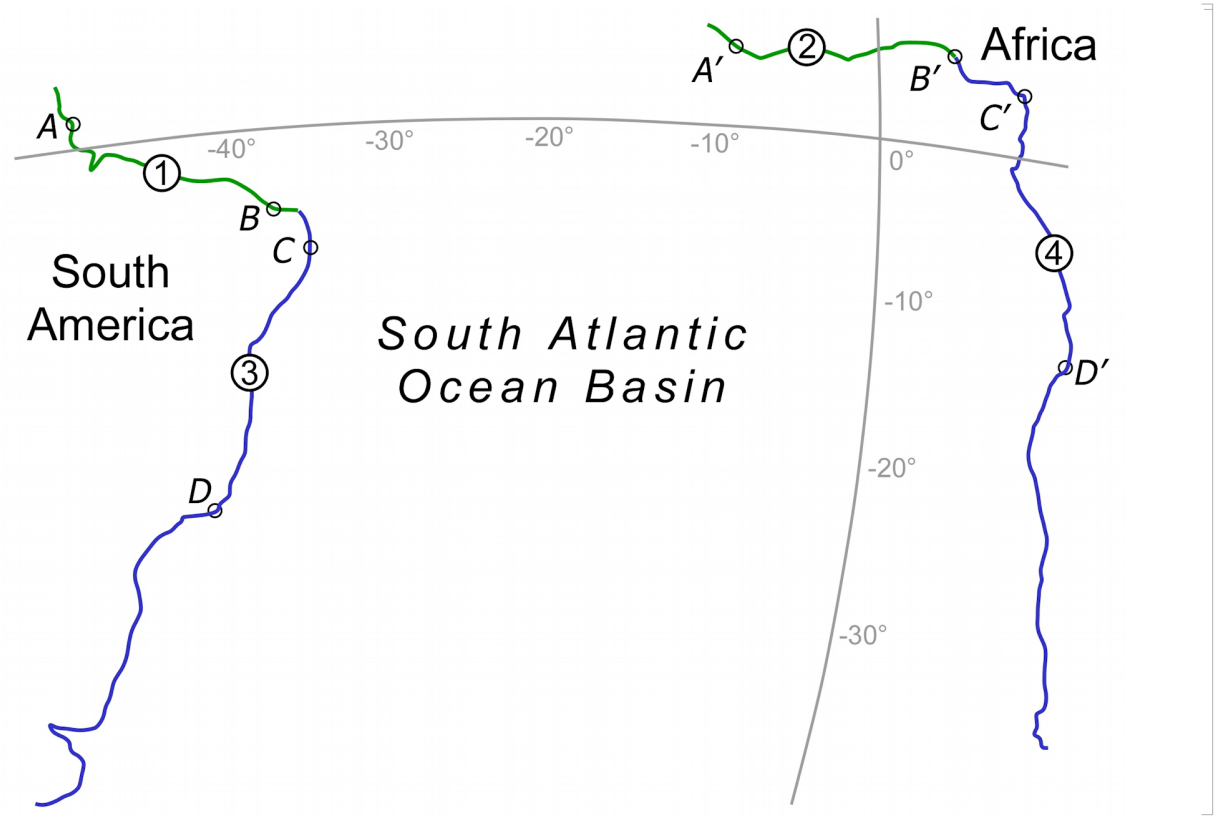


Figure 6-2. Selected reference points along the coastlines across the South Atlantic Ocean Basin, along the South American and African continents. Data are from Seton and others, 2012.

It can be said that the fit of Africa and South America is the earliest plate-kinematic problem, dating from before Wegener and his continental-drift ideas. Jim Everett was the first to employ a computational analysis to find a best-fit solution (Bullard *et al.*, 1965). His solution employed the 500-fathom (914 m) isobath around the continents to approximate the boundary between continental crust and oceanic crust, and he assumed that both continents have been rigid (without internal deformation) since the time of initial breakup. Many other kinematic analyses of this system have followed (*e.g.*, Moulin et al., 2010; Scotese, 2008; Seton et al., 2012).

Exercise 6.2 Determine total-opening poles for two segments of the boundaries of the South American and African plates, associated with the opening of the South Atlantic Ocean Basin (Figure 6.2). The reason the boundaries have been divided into two segments is that internal deformation of one or both of the continents has changed the shape of their initial boundaries, and so dividing the problem in two is a first-order attempt to accommodate this complication. We will call the northern boundary of the South American continent *boundary segment 1*, the corresponding boundary on Africa will be *boundary segment 2*, the southern boundary of South America will be *boundary segment 3*, and the southern boundary of Africa will be *boundary segment 4* (Figure 6.2).

For simplicity, we use the present-day coastline as a first-order proxy for the boundary between the continental and oceanic crust. The present-day continental-oceanic crust boundary is ill defined, and is distorted by extension during rifting, so the coastline will be sufficient for our pedagogical purposes. We identify a point (point *A*) toward the west end of boundary 1 that was immediately adjacent to a point (point *A'*) near the west end of boundary 2 prior to continental breakup. Then we identify another point (point *B*) near the east end of boundary 1, and the corresponding point (point *B'*) on boundary 2 such that the distance *A*-*B* is the same as *A'*-*B'*. We do the same for boundary segments 3 and 4, identifying points *C*, *C'*, *D* and *D'*. These paris of points were derived from the Africa-South America reconstruction map produced using *GPlates* software. *GPlates* is freeware made available by Dietmar Müller and his clever nerdy friends at the University of Sydney. Here are the present-day locations of our reference points:

A lat 1.59° long -49.87° *A'* lat 5.14° long -9.23°
B lat -5.03° long -37.04° *B'* lat 5.87° long 5.23°

C lat -7.67° long -34.92° *C'* lat -3.9112° long 9.7791°
D lat -22.85° long -42.19° *D'* lat -12.3471° long 13.5758°

Write a code in *Mathematica* that will determine the pole of rotation and rotational angle that will bring points *A* and *B* into coincidence with points *A'* and *B'*, respectively. Then find the pole of rotation and rotational angle for points *C* to *C'* and *D* to *D'*. Compare your results with published estimates of the total opening pole for the South Atlantic Ocean Basin (*e.g.*, Bullard and others, 1965, or more recent estimates).

6.8 Solutions involving more than two pairs of points

Surely, you can go back to section 6.5 and determine how to handle cases involving more than two pairs of points! You end up with multiple pole solutions, and then find the mean pole and the 95% confidence angle around that mean using Fisher statistics or some other appropriate statistical tool.

Exercise 6.3 Given the following reference points:
A lat 1.59° long -49.87° *A'* lat 5.14° long -9.23°
B lat -5.03° long -37.04° *B'* lat 5.87° long 5.23°
C lat -7.67° long -34.92° *C'* lat -3.9112° long 9.7791°
D lat -22.85° long -42.19° *D'* lat -12.3471° long 13.5758°
write a code in *Mathematica* that will determine the pole of rotation and rotational angle for the following combinations: A-A' and B-B', C-C' and D-D', A-A' and C-C', B-B' and D-D', A-A' and D-D'. Determine the mean rotational pole and the radius of the 95% confidence-interval cone around that mean of all 5 solutions. Also find the mean rotational angle and its 1-σ (one standard deviation) uncertainty. Compare your results with published estimates of the total opening pole for the South Atlantic Ocean Basin (*e.g.*, Bullard et al., 1965, or more recent estimates).

6.9 Importing input data from an Excel spreadsheet

It is often useful to operate on data that are contained in a file that is separate from the *Mathematica* notebook. *Mathematica* has the ability to input many types of data from many different sources, and can also export data to many different types of files. Much more information about inporting and exporting is available to you by going to the virtual book accessible through the Help drop-down menu on the *Mathematica* menu bar. In this chapter, we will import our input data from a Microsoft Excel spreadsheet.

Making a data file in Excel

The input data that we will use in this chapter is an Excel spreadsheet (.xls file) in which each record (that is, each horizontal row) has 2 values/columns. The first column contains the latitude of a point in degrees (range of latitude is -90° at the south pole to 90° at the north pole), and the second column has the longitude of the point in degrees (range of longitude is -180° to 180°, with west longitudes as negative numbers). Four Excel files called SAtlBound1.xls, SAtlBound2.xls, SAtlBound3.xls, and SAtlBound14xls have been made available to you, or can be downloaded from

http://CroninProjects.org/Vince/PlateKinematics/KinematicsPrimer/SAtlBound1.xls
http://CroninProjects.org/Vince/PlateKinematics/KinematicsPrimer/SAtlBound2.xls
http://CroninProjects.org/Vince/PlateKinematics/KinematicsPrimer/SAtlBound3.xls
http://CroninProjects.org/Vince/PlateKinematics/KinematicsPrimer/SAtlBound4.xls

Finding the Excel data file on your computer

Each user will need to ensure that the path to the input data file is correctly specified in the first input line of this notebook in the blue box below. For example, a correct specification for the file "SOAM-COnorth.xls" located on the desktop of Vince Cronin's office iMac computer would look like this:

```
mydata = Import["/Users/vince_cronin/Desktop/SAtlBound1.xls"];
```

On Vince Cronin's MacBook Pro, the code to find the Excel file would be

```
mydata = Import["/Users/vincecronin/Desktop/SAtlBound1.xls"];
```

and the specification for the file "otherPlate.xls" located in the Kinematics directory on the C drive of Cronin's Windows computer would look like this:

```
mydata=Import["C:\Kinematics\SAtlBound1.xls"];
```

In the code immediately following this text, be certain that the path to the input data file is correctly specified.

In[155]:=
```
mydata = Import["/Users/vince/Desktop/SAtlBound1.xls"];
```

Let's take a look at **myData**

In[156]:=
```
mydata
```

Out[156]=
```
{{{2.88, -50.77}, {2.68, -50.67}, {2.5537, -50.8184}, {2.287, -50.6927},
  {2.0225, -50.5464}, {1.8684, -50.3932}, {1.8615, -50.1966},
  {1.7856, -49.9634}, {1.5878, -49.8697}, {1.3393, -49.8903}, {1.1506, -49.936},
  {1.0011, -50.1669}, {0.8216, -50.3109}, {0.6605, -50.5006},
  {0.3682, -50.6355}, {0.1772, -50.8847}, {-0.0137, -50.9807},
  {-0.2232, -51.2253}, {-0.4442, -51.3648}, {-0.5845, -51.4836},
  {-0.5615, -51.223}, {-0.4142, -51.0379}, {-0.1634, -50.9373},
  {0.0343, -50.743}, {-0.0663, -50.5052}, {-0.2992, -50.6835},
  {-0.5178, -50.8207}, {-0.6513, -50.9464}, {-0.6582, -50.7521},
  {-0.9366, -50.7201}, {-0.7548, -50.727}, {-0.4994, -50.6218},
  {-0.2601, -50.5486}, {-0.145, -50.2972}, {-0.2209, -49.9474},
  {-0.2255, -49.7325}, {-0.2462, -49.5451}, {-0.1703, -49.3005},
  {-0.1519, -48.9759}, {-0.2762, -48.7084}, {-0.3015, -48.5232},
  {-0.4142, -48.3312}, {-0.5891, -48.4432}, {-0.8423, -48.4889},
  {-1.0264, -48.5461}, {-1.2104, -48.6947}, {-1.4865, -48.8044},
  {-1.5555, -49.0467}, {-1.6591, -49.3279}, {-1.7764, -49.5657},
  {-1.7948, -49.8514}, {-1.7741, -50.0343}, {-1.8661, -50.2766},
  {-1.919, -50.4664}, {-1.9627, -50.2515}, {-1.8937, -49.9931},
  {-1.9397, -49.7691}, {-1.873, -49.5359}, {-1.7327, -49.2525},
  {-1.9144, -49.3073}, {-2.1789, -49.3805}, {-2.3721, -49.4765},
  {-2.2341, -49.257}, {-1.9903, -49.1885}, {-1.7856, -49.049},
  {-1.5717, -48.8136}, {-1.4267, -48.5712}, {-1.2427, -48.3746},
  {-0.9274, -48.1895}, {-0.7433, -48.0203}, {-0.6007, -47.8397},
  {-0.626, -47.5585}, {-0.64, -47.37}, {-0.69, -47.09}, {-0.83, -46.86},
  {-1., -46.51}, {-1.03, -46.33}, {-1.03, -46.15}, {-1.14, -45.97},
  {-1.22, -45.76}, {-1.27, -45.57}, {-1.46, -45.33}, {-1.65, -45.3},
  {-1.57, -45.06}, {-1.52, -44.86}, {-1.68, -44.67}, {-1.89, -44.52},
  {-2.08, -44.55}, {-2.23, -44.33}, {-2.41, -44.29}, {-2.65, -44.46},
  {-2.73, -44.25}, {-2.62, -43.98}, {-2.54, -43.82}, {-2.37, -43.54},
  {-2.51, -43.39}, {-2.53, -43.15}, {-2.6, -42.94}, {-2.7, -42.59},
  {-2.78, -42.36}, {-2.85, -42.15}, {-2.91, -41.87}, {-2.93, -41.58},
  {-2.96, -41.16}, {-2.95, -40.95}, {-2.93, -40.58}, {-2.93, -40.34},
  {-2.93, -40.15}, {-2.95, -39.9}, {-3.03, -39.68}, {-3.13, -39.46},
  {-3.3, -39.23}, {-3.46, -38.98}, {-3.65, -38.71}, {-3.77, -38.55},
  {-3.9, -38.38}, {-4.09, -38.2}, {-4.26, -38.03}, {-4.44, -37.85},
  {-4.6, -37.6}, {-4.74, -37.33}, {-4.9, -37.17}, {-5.03, -37.04}}}
```

Each pair of lat/long values is enclosed within a set of curly brackets, and the list of these several lat/long datapairs is enclosed in not one but two curly brackets. Those double curly brackets are going to cause us problems further down in the code, so we need to use the built-in *Mathematica* function **Flatten** to remove one of the dimensions of this array -- that is, to get rid of one set of curly brackets that enclose the dataset. We will call this new and improved datafile **inData**.


```
In[157]:= inData = Flatten[mydata, 1];
```

Another way of evaluating whether we have a datafile that has the correct form is to use the built-in *Mathematica* function **Dimensions**. We want to work with a file of geographic (lat/long) coordinates that has two dimensions, as indicated by two values in the list generated by the following:

```
In[158]:= Dimensions[mydata]
Out[158]= {1, 123, 2}
```

We found that **Dimensions[mydata]** has three dimensions (3 values inside the curly brackets}. After flattening mydata to form inData, **Dimensions[inData]** has the desired two dimensions.

```
In[159]:= Dimensions[inData]
Out[159]= {123, 2}
```

Now, we need to convert the data of **inData** to Cartesian geographic coordinates, using the user-defined function **convert2Cart** that we developed in an earlier chapter, and the built-in *Mathematica* function **Table**. We specify the latitude of the record in row **i** using **inData[[i,1]]**, and the longitude is **inData[[i,2]]**. The total number of rows or records in the datafile is given by **Length[inData]**.

```
In[160]:= inDataCart =
            Table[convert2Cart[inData[[i, 1]], inData[[i, 2]]], {i, 1, Length[inData]}];
```

Let's take a look at the first four records/rows of **inDataCart** using the built-in *Mathematica* function **Take** and arranged in a nice matrix using the built-in function **MatrixForm**.

```
In[161]:= MatrixForm[Take[inDataCart, 4]]
Out[161]//MatrixForm=
          ⎛ 0.631636  -0.773635  0.0502443 ⎞
          ⎜ 0.633093  -0.772662  0.0467578 ⎟
          ⎜ 0.631153  -0.774378  0.0445557 ⎟
          ⎝ 0.632975  -0.773143  0.0399051 ⎠
```

As expected, each of the four rows contains three values, corresponding to the Cartesian coordinates of the individual points.

6.10 Importing and plotting the coastline data

We learned how to plot some data on a sphere in the previous chapter. Let's now do the same, using data stored externally in Excel spreadsheets. The data we will import are sets of geographic locations for points along the four boundary sections shown in Figure 6.2. Remember that you will have to adjust the path that the **Import** function acts on, to match your own computing environment.

In the code immediately following this text, be certain that the path to the input data file is correctly specified.

```
In[162]:= geogData1 = Import["/Users/vince/Desktop/SAtlBound1.xls"];
In[163]:= geogData2 = Import["/Users/vince/Desktop/SAtlBound2.xls"];
In[164]:= geogData3 = Import["/Users/vince/Desktop/SAtlBound3.xls"];
In[165]:= geogData4 = Import["/Users/vince/Desktop/SAtlBound4.xls"];
```

We use the built-in *Mathematica* function **Flatten** to change the input datasets so that they have the right number of dimensions.

```
In[166]:= inData1 = Flatten[geogData1, 1];
In[167]:= inData2 = Flatten[geogData2, 1];
In[168]:= inData3 = Flatten[geogData3, 1];
In[169]:= inData4 = Flatten[geogData4, 1];
```

We use the user-defined function **convert2Cart** to convert the input datasets into Cartesian coordinates, and save the results in a set of four tables.

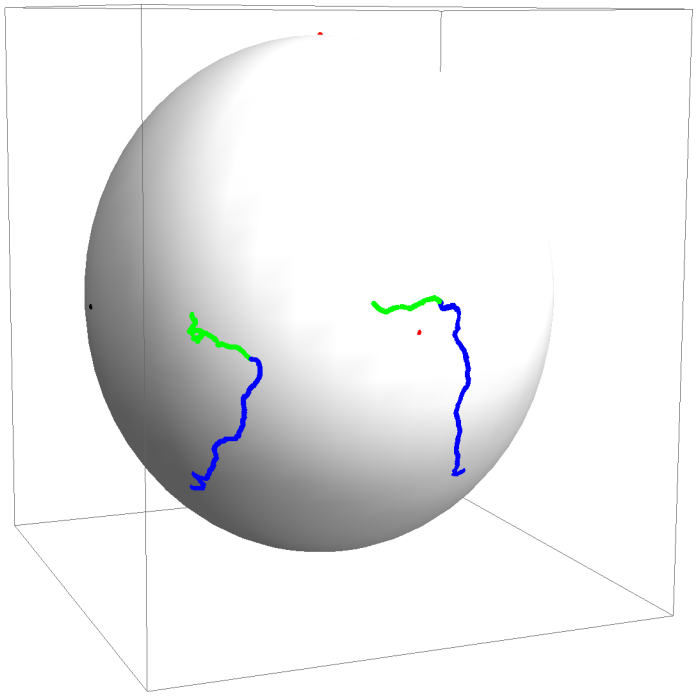

```
In[170]:= inDataCart1 =
            Table[convert2Cart[inData1[[i, 1]], inData1[[i, 2]]], {i, 1, Length[inData1]}];
In[171]:= inDataCart2 =
            Table[convert2Cart[inData2[[i, 1]], inData2[[i, 2]]], {i, 1, Length[inData2]}];
In[172]:= inDataCart3 =
            Table[convert2Cart[inData3[[i, 1]], inData3[[i, 2]]], {i, 1, Length[inData3]}];
In[173]:= inDataCart4 =
            Table[convert2Cart[inData4[[i, 1]], inData4[[i, 2]]], {i, 1, Length[inData4]}];
```

And now we plot the data on a sphere, as we did in Chapter 5. To make life simpler for you, I have concocted a module called **fourColorDotSphere** that plots points in four colors on a sphere. You will need to collect all of the vectors that you want to be represented as a given color into one table. Let's use the same color scheme as in Figure 6.1, and have all of the points along boundaries 1 and 2 be colored green, and all of the points along boundaries 3 and 4 colored blue. To help us orient the illustration, we will put red dots at the ends of the positive *X*, *Y* and *Z* axes, and black dots at the negative ends of those axes. The colors available in *Mathematica* by name are red, green, blue, black, white, gray, cyan, magenta, yellow, brown orange pink and purple. There is also light red, light green, and so on. More choices exist, and are explained in the Wolfram Language Guide (http://reference.wolfram.com/language/guide/Colors.html).

We will use the built-in *Mathematica* function **Join** to combine the lists of Cartesian coordinates by desired color.

```
In[174]:= greenDots = Join[inDataCart1, inDataCart2];
In[175]:= blueDots = Join[inDataCart3, inDataCart4];
In[176]:= redDots = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
In[177]:= blackDots = {{-1, 0, 0}, {0, -1, 0}, {0, 0, -1}};
In[178]:= fourColorDotSphere[greenDots, blueDots, redDots, blackDots]
```

Out[178]=



Exercise 6.4 In the Exercise 6.2, we determined two total-opening poles for the South Atlantic Ocean Basin: one for boundary sections 1 and 2 and the other for boundary sections 3 and 4.

In this exercise, we will move boundary sections 2 and 4 around their respective total-opening poles to bring them next to bondaries 1 and 3. We will use the results you obtained in Exercise 6.2. Four Excel files are provided online

(CroninProjects.org/Vince/PlateKinematics/KinematicsPrimer/Ex6-2.html) that contain points along the north and south coastlines of South America and Africa, with data taken from the estimated continent-ocean boundaries of the present-day continents as compiled by Seton and others (2012). Write a code in *Mathematica* that will rotate all of the points in boundaries 2 and 4 around the respective total poles to coincide with boundaries 1 and 3, respectively, and use the module fourColorDotSphere to illustrate those rotations in a single graphic. That is, illustrate the total rotation of African continental crust as observed from South American continental crust. In so doing, boundaries 1 and 3 will not change their spatial relationship with each other -- you will hold South America to be rigid -- but boundaries 2 and 4 might change their spatial relationship with each other.

Make all of the points along boundaries 1 and 3 (the South American coastline) blue, and the points along boundaries 2 and 4 (the displaced African coastline) green.

How might you determine the total motion of boundary 2 relative to boundary 4 during the opening of the South Atlantic?

6.11 References and relevant texts

Batschelet, E., 1981, Circular Statistics in Biology: London, Academic Press, 371 p.

Borradaile, G., 2003, Statistics of Earth Science Data, Their Distribution in Time, Space, and Orientation: Berlin, Springer-Verlag, 351 p.

Bullard, E.C., Everett, J.E., and Smith, A.G., 1965, The fit of the continents around the Atlantic, in Blackett, P.M.S., Bullard, E.C., and Runcorn, S.K., [editors], A symposium on continental drift: Philosophical Transactions of the Royal Society, London, series A, v. 258, p. 41-51.

Condie, K.C., and Pease, V., editors, 2008, When did plate tectonics begin on planet Earth: Geological Society of America, Special Paper 440, 294 p., ISBN 978-0-8137-2440-9.

Cronin, V.S., 2008, Finding the mean and 95 % confidence interval of a set of strike - and - dip or lineation data: Environmental and Engineering Geoscience, v.14, no.2, p.113 - 119.

Fisher, N.I., Lewis, T., and Embleton, B.J.J., 1987, Statistical Analysis of Spherical Data: Cambridge, UK, Cambridge University Press, 329 p.

Fisher, R.A., 1953, Dispersion on a sphere: Proceedings Royal Society, London, v. A217, no. 1130. p. 295-305.

Lancelot, Y., Larson, R.L., et al., 1990. Proceedings of the Ocean Drilling Program, Initial Reports, Leg 129: College Station, TX (Ocean Drilling Program) doi:10.2973/odp.proc.ir.129.1990

Lancelot, Y., R.L. Larson and ODP Leg 129 Scientific Party, 1990, Ancient Crust on the Pacific Plate, Nature, V.345, 112.

Lancelot, Y., R.L. Larson and ODP Leg 129 Scientific Party, 1990, Jurassic oceanic crust and sediments in the Pacific, at last, Geotimes, June 1990, 25-26.

Mardia, K.V., 1972, Statistics of Directional Data: London, Academic Press, 357 p.

McKenzie, D. & Sclater, J. G., 1971 The evolution of the Indian Ocean since the late Cretaceous. Geophys. J. R. astr. Soc. 25, 437-528.

Moulin, M., Aslanian, D., and Unternehr, P., 2010, A new starting point for the South and Equatorial Atlantic Ocean: Earth-Science Reviews, v. 98, p. 1-37, doi: 10.1016/j.earscirev.2009.08.001

Scotese, C.R., 2008, The fit of Africa and South America (105 - 145 Ma): unpublished research, PALEOMAP Project, Irving, Texas, 8 p.

Seton, M., Müller, R.D., Zahirovic, S., Gaina, C., Torsvik, T., Shephard, G., Talsma, A., Gurnis, M., Turner, M., Maus, S., and Chandler, M., 2012, Global continental and ocean basin reconstructions since 200 Ma: Earth-Science Reviews, v. 113, p. 212-270.

Useful Sites and Software

Data and other resources for exercises 6.2 through 6.4, accessible via http://CroninProjects.org/Vince/PlateKinematics/KinematicsPrimer/Ex6-2.html

GPlates software, accessible via http://www.gplates.org

PALEOMAP Project, accessible via http://www.scotese.com

Placa3D software, accessible via http://wwz.ifremer.fr/drogm/Produits-et-services/ Logiciels-gratuits/Logiciel-Placa3D

