TJ-boundary-characterization-PA-NA.nb

(Development of a module to characterize a plate boundary at a triple junction, given an appropriate Euler vector)

Coded in *Mathematica* v. 11.2.0 by Vince Cronin
February 5, 2019

Introduction

The purpose of this *Mathematica* notebook is to build and test a module to quantitatively describe a plate boundary at a triple junction, given an appropriate Euler vector and the assumption that Earth is spherical.

Assumptions

We assume that Earth is a sphere of unit radius in all but one part of this analysis. Of course, Earth is not really a sphere. Earth has an irregular surface with deep ocean basins and tall mountains. The average radius of Earth is around 6,371.01±0.02 km (Yoder, 1995, p. 8). The deepest part of the ocean basins, called the Challenger Deep in the Mariana Trench, is ~11.0 km below sea level (that is, ~0.2% of Earth's mean radius), and the summit of Mt. Everest is ~8.848 km above sea level (~0.1% of Earth's mean radius). Earth's polar radius (6356.7530 km) differs from its mean equatorial radius (6378.1363 km) by only ~21 km, so Earth's shape is different from a sphere by about 1 part in 300 or about 0.3 % (*e.g.,* Cazenave, 1995, p. 36).

Assuming that Earth is a sphere of unit radius makes the mathematics we use in this problem much simpler than if we used either an ellipsoid or the geoid. For just one example, the radius of a sphere is the same measured at any point on the sphere's surface, and this is not true for either the reference ellipsoid or the geoid. Although Earth is a little wider at its equator than along its spin axis, we do not generate serious errors in our calculations by assuming Earth's radius is the same in all directions -- that it is a sphere.

When we say that Earth is assumed to have unit radius, that means that we set Earth's radius to 1 unit of length. The one part of this analysis in which we must use the actual size of the Earth in kilometers is in the determination of the circumferencial distance between the reference point

and the observed pointt.

Specify the input data

Input only decimal degrees (*i.e.,* not degrees, minutes, seconds). The sign convention follows: north latitude is positive, south latitude is negative, east longitude is positive, west longitude is negative.
The variables **tjLat** and **tjLong** are the latitude and longitude of the triple junction.

```
tjLat = 40.352;
```

```
tjLong = -124.632;
```

The variables **boundLat** and **boundLong** are the latitude and longitude of a typical point along the plate boundary near the triple junction.

```
boundLat = 38.681;
```

```
boundLong = -123.414;
```

The variables **leftPlate** and **rightPlate** are the names of the plates to the left and right of the boundary, respectively, as viewed from the triple junction. Because these are text values, they must be enclosed in quotation marks, as in “Pacific”.

```
leftPlate = "North America";
```

```
rightPlate = "Pacific";
```

```
refPlate = "Pacific";
```

```
movingPlate = "North America";
```

The variables **poleLat** and **poleLong** are the latitude and longitude of the Euler pole, and **omegaPole** is the angular velocity around that pole, expressed in °/Myr. A positive value indicates a right-hand or anticlockwise rotation around the pole, as observed from the pole toward Earth’s center.

```
poleLat = 49.30;
```

```
poleLong = -76.01;
```

```
omegaPole = 0.791;
```

The variable **radiusEarth** is the radius of an assumed-spherical Earth. A value of 6371.02 is usually used, and comes from Yoder (1995).

```
radiusEarth = 6371.02;
```

Operate on the input data

User-defined functions

The function **makeVector** computes the unit location vector of a given point on an assumed-spherical Earth, given that point's latitude and longitude. The x coordinate is equal to **Cos**[latitude]*__Cos__[longitude]. The y coordinate is equal to **Cos**[latitude]*__Sin__[longitude]. The z coordinate is equal to **Sin**[latitude]. The input latitude and longitude values must be in decimal degrees (*i.e.,* not degrees, minutes, seconds). The forms "*lat* **Degree**" and "*long* **Degree**" in the square brackets invokes the built-in *Mathematica* function **Degree** to convert the input data from degrees to radians, as required for the built-in *Mathematica* trigonometric functions (*e.g.*, **Sin**, **Cos**, **Tan**, **ArcSin**, **ArcCos**, **ArcTan**).

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

The function **unitVector** finds the corresponding vector with a length of 1 (*i.e.,* unit length) given a vector of arbitrary length. It uses a built-in *Mathematica* function **Norm**, which returns the length of the input vector. Given a vector **A** with components $\{a_x, a_y, a_z\}$, the length or *norm* of **A** is equal to

$$\mathbf{Norm}[\mathbf{A}] = \sqrt{(a_x)^2 + (a_y)^2 + (a_z)^2}\,.$$

The unit-vector that corresponds to **A** has components $\{(a_x/\text{Norm}[\mathbf{A}]), (a_y/\text{Norm}[\mathbf{A}]), (a_z/\text{Norm}[\mathbf{A}])\}$.

```
unitVector[a_] := {a[[1]] / Norm[a], a[[2]] / Norm[a], a[[3]] / Norm[a]};
```

The built-in *Mathematica* function **VectorAngle** provides the angle between two vectors, in radians. Recall that 360° = $(2\,\pi)$ radians, so to convert from radians to degrees, multiply the radian value by $(180/\pi)$.

```
tangVelocityV2[refPtVect_, poleVect_, angVel_, radEarth_] :=
  Module[{circEarth, northPole, θ1, θ2, θ3, θ4, normVect1, normVect2, normVect3,
    distKm, bearing, speed, direction, answer}, circEarth = 2 * π * radEarth;
   northPole = {0, 0, 1};
   normVect1 = Cross[northPole, refPtVect];
   normVect2 = Cross[poleVect, refPtVect];
   θ1 = VectorAngle[refPtVect, poleVect];
   distKm = (θ1 / (2 * π)) * circEarth;
   θ2 = (180 / π) * (VectorAngle[normVect1, normVect2]);
   θ3 = (180 / π) * (VectorAngle[normVect1, poleVect]);
   normVect3 = Cross[refPtVect, normVect1];
   θ4 = (180 / π) * (VectorAngle[normVect3, poleVect]);
   bearing = If[((θ3 == 90) ∨ (θ3 == 180)),
     If[(θ4 <= 90), 0, 180], If[(θ3 > 90), (360 - θ2), θ2]];
   speed = (angVel * Sin[θ1]) * (circEarth / 360);
   direction = If[(bearing ≤ 270), bearing + 90, (bearing - 270)];
   answer = {direction, speed};
   answer];

distAzFromVectorsV2[refPtVect_, otherPtVect_, radEarth_] :=
  Module[{circEarth, northPole, θ1, θ2, θ3, θ4, normVect1, normVect2,
    normVect3, distKm, bearing, answer}, circEarth = 2 * π * radEarth;
   northPole = {0, 0, 1};
   normVect1 = Cross[northPole, refPtVect];
   normVect2 = Cross[otherPtVect, refPtVect];
   θ1 = VectorAngle[refPtVect, otherPtVect];
   distKm = (θ1 / (2 * π)) * circEarth;
   θ2 = (180 / π) * (VectorAngle[normVect1, normVect2]);
   θ3 = (180 / π) * (VectorAngle[normVect1, otherPtVect]);
   normVect3 = Cross[refPtVect, normVect1];
   θ4 = (180 / π) * (VectorAngle[normVect3, otherPtVect]);
   bearing = If[((θ3 == 90) ∨ (θ3 == 180)),
     If[(θ4 <= 90), 0, 180], If[(θ3 > 90), (360 - θ2), θ2]];
   answer = {distKm, bearing};
   answer];
```

```
bearingFromVectorsV2[refPtVect_, otherPtVect_] :=
  Module[{circEarth, northPole, θ1, θ2, θ3, θ4, normVect1, normVect2,
    normVect3, distKm, bearing, answer}, northPole = {0, 0, 1};
   normVect1 = Cross[northPole, refPtVect];
   normVect2 = Cross[otherPtVect, refPtVect];
   θ1 = (180 / π) * (VectorAngle[normVect1, normVect2]);
   θ2 = (180 / π) * (VectorAngle[normVect1, otherPtVect]);
   normVect3 = Cross[refPtVect, normVect1];
   θ3 = (180 / π) * (VectorAngle[normVect3, otherPtVect]);
   bearing = If[((θ2 == 90) ∨ (θ2 == 180)),
     If[(θ3 <= 90), 0, 180], If[(θ2 > 90), (360 - θ1), θ1]];
   answer = bearing;
   answer];
```

```
tjBoundaryCharacteristics[tjLat_, tjLong_, boundLat_,
    boundLong_, poleLat_, poleLong_, omegaPole_, leftPlate_,
    rightPlate_, refPlate_, movingPlate_, radiusEarth_] :=
  Module[{circEarth, tjVect, boundVect, poleVect, alpha, boundaryType,
    tangVel, boundaryBearing, refSide, movingSide, epCrossTJ,
    boundCrossTJ, tjCrossBoundCrossTJ, beta, dirRelMotion, purity,
    delta, boundParallelSpeed, shearSense, boundNormalSpeed,
    acrossBoundary, answer}, circEarth = 2 * π * radiusEarth;
   tjVect = {Cos[tjLat Degree] Cos[tjLong Degree],
     Cos[tjLat Degree] Sin[tjLong Degree], Sin[tjLat Degree]};
   boundVect = {Cos[boundLat Degree] Cos[boundLong Degree],
     Cos[boundLat Degree] Sin[boundLong Degree], Sin[boundLat Degree]};
   poleVect = {Cos[poleLat Degree] Cos[poleLong Degree],
     Cos[poleLat Degree] Sin[poleLong Degree], Sin[poleLat Degree]};
   alpha = VectorAngle[Cross[tjVect, boundVect],
      (Cross[tjVect, poleVect])] / Degree;
   boundaryType = If[(alpha ≥ 75) ∧ (alpha ≤ 105), "transform",
     If[(alpha ≤ 15) ∧ (alpha ≥ 165), "convergent/divergent", "oblique"]];
   tangVel = tangVelocityV2[tjVect, poleVect, omegaPole, radiusEarth];
   boundaryBearing = bearingFromVectorsV2[tjVect, boundVect];
   refSide = If[(refPlate == leftPlate), "leftSide", "rightSide"];
   movingSide = If[(refPlate == leftPlate), "rightSide", "leftSide"];
   epCrossTJ = Cross[poleVect, tjVect];
   boundCrossTJ = Cross[boundVect, tjVect];
   tjCrossBoundCrossTJ = Cross[tjVect, boundCrossTJ];
   beta = VectorAngle[epCrossTJ, boundCrossTJ] / Degree;
   dirRelMotion = If[(beta < 90) ∧ (beta > 0), "rightSide",
     If[(beta > 90) ∧ (beta < 180), "leftSide", "boundaryParallel"]];
   purity = If[(beta == 0) ∨ (beta == 180), "boundaryNormal",
     If[(beta == 90), "boundaryParallel", "partlyOblique"]];
   delta = VectorAngle[tjCrossBoundCrossTJ, epCrossTJ] / Degree;
   boundParallelSpeed = tangVel[[2]] * Abs[Cos[delta Degree]];
   shearSense = If[(delta < 90) ∧ (movingSide == "leftSide"), "rightLateral",
     If[(delta < 90) ∧ (movingSide == "rightSide"), "leftLateral", If[(delta >
           90) ∧ (movingSide == "leftSide"), "leftLateral", If[(delta > 90) ∧
         (movingSide == "rightSide"), "rightLateral", "boundaryNormal"]]]];
   boundNormalSpeed = tangVel[[2]] * Sin[delta Degree];
   acrossBoundary = If[(refSide == dirRelMotion), "convergent",
     If[(movingSide == dirRelMotion), "divergent", dirRelMotion]];
   answer = {alpha, boundaryBearing, boundaryType, tangVel, purity,
     boundParallelSpeed, shearSense, boundNormalSpeed, acrossBoundary};
   answer];
```

Computation

```
results = tjBoundaryCharacteristics[tjLat,
   tjLong, boundLat, boundLong, poleLat, poleLong, omegaPole,
   leftPlate, rightPlate, refPlate, movingPlate, radiusEarth];
```

Input Summary

{tjLat, tjLong}

{40.352, -124.632}

{boundLat, boundLong}

{38.681, -123.414}

{poleLat, poleLong}

{49.3, -76.01}

omegaPole

0.791

{leftPlate, rightPlate}

{North America, Pacific}

{refPlate, movingPlate}

{Pacific, North America}

radiusEarth

6371.02

Output

The value **results[[1]]** is the angle (in degrees) between (1) the bearing from the triple junction along the boundary to (2) the bearing from the triple junction to the Euler pole.

results[[1]]

91.6599

The value **results[[2]]** is the bearing (azimuth) from the triple junction to the selected point along the plate boundary.

```
results[[2]]
```

```
150.261
```

The value **results[[3]]** is a brief description of the type of plate boundary: convergent, divergent, transform, or oblique.

```
results[[3]]
```

```
transform
```

The value **results[[4]]** is a 2-value list in which the first value is the direction of relative plate motion at the selected point along the boundary, and the second value is the tangential speed in mm/yr.

```
results[[4]]
```

```
{148.601, 50.4212}
```

The value **results[[5]]** contains information about whether the boundary is purely convergent, purely divergent, or purely transform in its relative motion.

```
results[[5]]
```

```
partlyOblique
```

The value **results[[6]]** is the value for the instantaneous speed of the plate, relative to the triple junction, in the direction parallel to the boundary.

```
results[[6]]
```

```
50.4
```

The value **results[[7]]** contains information about whether the boundary has a left-lateral or right-lateral sense of shear along the boundary.

```
results[[7]]
```

```
rightLateral
```

The value **results[[8]]** is the value for the instantaneous speed of the plate, relative to the triple junction, in the direction perpendicular to the boundary..

```
results[[8]]
```

```
1.46055
```

The value **results[[9]]** contains information about whether the boundary has a divergent or convergent sense of motion at the triple junction..

```
results[[9]]
```

```
divergent
```

References

Cazenave, A., 1995, Geoid, topography and distribution of landforms, *in* Ahrens, T.J., [editor], **Global Earth Physics -- A Handbook of Physical Constants**: American Geophysical Union Reference Shelf 1, p. 32-39, ISBN 0-87590-851-9.

Davis, H.F., and Snider, A.D., 1987, **Introduction to Vector Analysis** [5th edition]: Boston, Massachusets, Allyn and Bacon, 365 p., ISBN 0-205-10263-8.

DeMets, C., Gordon, R.G., and Argus, D.F., 2010, Geologically current plate motions: Geophysical Journal International, v. 181, p. 1-80, doi:10.1111/j.1365-246X.2009.04491.x

Yoder, C.F., 1995, Astrometric and geodetic properties of Earth and the solar system, *in* Ahrens, T.J., [editor], **Global Earth Physics -- A Handbook of Physical Constants**: American Geophysical Union Reference Shelf 1, p. 1-31, ISBN 0-87590-851-9.