David Roman Garcia
UROP 2022
July 5, 2022

Cross Track Distance

Algorithm

Consider an arbitrary point B and a given great circle arc both on the surface of the same unit sphere centered at the origin. To calculate the shortest distance from each other along the sphere, let $\vec{P}$ be the vector going from the origin to B, and $\vec{Q}$ and $\vec{R}$ be the vectors from the origin to endpoints of the arc. We can then use the following algorithm:

$$\vec{S} = \hat{Q} \times \hat{R} \qquad \text{Find the normal to the great circle and call it } \vec{S}.$$

$$\hat{P} \cdot \vec{S} = |\vec{S}| \cos\beta \qquad \text{Project } \vec{P} \text{ onto } \vec{S}.$$

$$\cos\beta = \frac{\hat{P} \cdot \vec{S}}{|\vec{S}|} \qquad \text{Solve for } \cos\beta.$$

$$\cos\beta = \sin\alpha \qquad \text{Use right angle trigonometry.}$$

$$\alpha = \arcsin\left(\frac{\hat{P} \cdot \vec{S}}{|\vec{S}|}\right) \qquad \text{Solve for } \alpha.$$

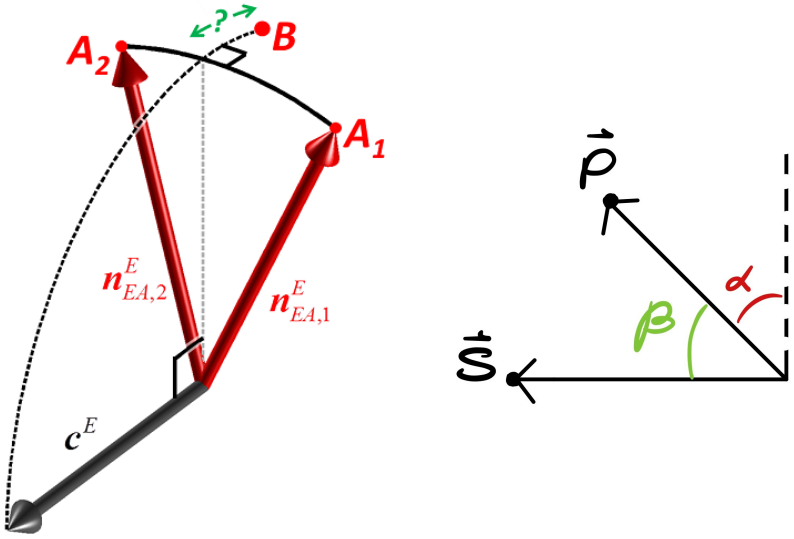

FIGURE 1. Great circle distance to a point.

Figure 1 shows two viewpoints. The left one considers the full 3D representation of the sphere and the points that lie on its surface. The right hand side shows its projection on the plane spanned by the normal of the arc segment joining $\vec{Q}$ and $\vec{R}$, and an arbitrary point on the surface whose coordinates are given by the vector $\vec{P}$.

See the last example on [1] for a sample code and [2] for an explanation of the method.

Implementation

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:


```

```
7  import numpy as np
8  import astropy.coordinates
9  import astropy.units as u
10 from astropy.coordinates import SkyCoord
11
12
13 # In[ ]:
14
15
16 def crosstrackdistance(lat1, long1, lat2, long2, lat3, long3):
17     c1 = SkyCoord(ra=long1*u.radian, dec=lat1*u.radian, frame='icrs')
18     c2 = SkyCoord(ra=long2*u.radian, dec=lat2*u.radian, frame='icrs')
19     c3 = SkyCoord(ra=long3*u.radian, dec=lat3*u.radian, frame='icrs')
20
21     c1.representation_type = 'cartesian'
22     c2.representation_type = 'cartesian'
23     c3.representation_type = 'cartesian'
24
25     P1 = c1.x.value, c1.y.value, c1.z.value
26     P2 = c2.x.value, c2.y.value, c2.z.value
27     P3 = c3.x.value, c3.y.value, c3.z.value
28
29     S = np.cross(P1,P2)
30     dot = np.dot(S,P3)
31
32     dist = np.arcsin(dot/np.linalg.norm(S))
33
34     return dist
```

Listing 1. Python 3 implementation of figure 1.

We have now made use of *SkyCoord* to switch between coordinate representations without an external function. This ensures that we are still working in the ICRS reference frame and adds robustness to the implementation. Since *SkyCoord* returns objects of type Quantity, we have used the value() function to strip the numerical magnitude from the object. This is not a problem given that we get cartesian unit vectors for all three points of interest on the sphere.

References

[1] Norwegian Defence Research Establishment. The n-vector page. https://www.ffi.no/en/research/n-vector/, 2010.
[2] L. Strous. Compute minimum distance between point and great arc on sphere. https://math.stackexchange.com/questions/337055/compute-minimum-distance-between-point-and-great-arc-on-sphere, 2018.