Spherical Distance Calculation Methods and Accuracy Comparison Postprint

Authors: Fan Dongwei; He Boliang; Li Changhua; Han Jun; Xu Yunfei; Cui Chenzhou

Date: 2018-06-22T00:00:00+00:00

Abstract

Spherical distance (angular separation) calculation is one of the most frequently used computations in astronomy and geography, serving as the foundation for target identification, cone search, cross-matching, and other methods. Mathematically, the distance between two points can be derived directly through spherical geometry, and numerous formulas of varying complexity have been established by predecessors. However, due to finite computational precision, rounding errors in numerical calculations can lead to deviations in formula results. This study investigates several commonly employed spherical distance calculation formulas, testing and comparing their accuracy, advantages, and disadvantages across different computational environments. Furthermore, it presents and compares the spherical distance calculation methods implemented in several widely used astronomical software packages and databases, aiming to assist astronomers in selecting the most appropriate calculation method for their specific needs.

Full Text

Research on Spherical Distance Computation Methods and Accuracy Comparison

Fan Dongwei, He Boliang, Li Changhua, Han Jun, Xu Yunfei, Cui Chenzhou
National Astronomical Observatories, Chinese Academy of Sciences, Beijing 100101, China

Abstract: Spherical distance (angular separation) calculation is one of the most commonly used computations in astronomy and geography, and forms the basis for methods such as object detection, cone search, and cross-matching. Mathematically, the distance between two points can be calculated directly through

spherical geometry, and researchers have derived multiple formulas of varying complexity. However, due to the limited precision of computers, rounding errors occur during numerical computation, leading to deviations in the calculated results. This paper examines several widely-used spherical distance calculation formulas, tests and compares their accuracy across different computational environments, and discusses their respective advantages and disadvantages. Additionally, we demonstrate and compare spherical distance calculation methods implemented in several commonly-used astronomical software packages and databases, with the aim of helping astronomers select the most suitable calculation method for their specific needs.

Keywords: Numerical computation; Spherical distance; Angular separation; Calculation accuracy

1 Spherical Geometry Methods

Spherical distance calculation can be derived directly from spherical geometry. Commonly used formulas include the Great-circle formula [1], the Haversine formula [2], etc. Additionally, Vincenty' s formula [3] has been applied in code libraries such as Astropy, but its complexity is too high; we briefly mention it in Section 3.1.

Let the equatorial coordinates of two points be (α_1, δ_1) and (α_2, δ_2). To calculate their spherical angular distance d, the Great-circle formula is given by equation (1); when the distance between two points is very small, some also use the simplified form (2) [2,3,4,5,6]; the Haversine formula is shown as (3).

Computers typically use the IEEE 754 binary floating-point arithmetic standard to process floating-point numbers, where single-precision 32-bit (float or single) values are accurate to 6-9 decimal places, and double-precision 64-bit (double) values are accurate to 15-17 decimal places [4]. This paper uses C++ programs to compare the accuracy of the three formulas, with results shown in Table 1 .

As can be seen from Table 1 , under single-precision, the Great-circle formula returns all zeros when the distance between two points is small, indicating severe rounding errors, while the error is also large when the distance is greater. Under double-precision, the Great-circle formula can return results, but with poorer accuracy than the Haversine formula. The simplified form of the Great-circle formula mostly achieves accuracy comparable to the Haversine formula. A common issue with these three formulas is that they may slightly exceed the true value, which can lead to missing sources when strict boundary values are applied. Therefore, in practical applications, it is necessary to consider slightly expanding the boundary values to include sources that might be missed due to computational precision limitations.

It is worth noting that the simplified form of the Great-circle formula produces very large errors near the poles. Although it has the lowest computational com-

plexity among the above formulas and achieves accuracy similar to Haversine outside the polar regions and for small distances, extreme caution should be exercised when using this formula in practice.

2 Cartesian Coordinate Method

In addition to solving based on the equatorial coordinate system, one can also use a Cartesian coordinate system to calculate the spherical distance between two points through three-dimensional coordinate vectors. Considering the celestial sphere with radius 1, the center of the celestial sphere is taken as the origin of the three-dimensional Cartesian coordinate system, the direction of the north celestial pole in the equatorial coordinate system serves as the z-axis, and the direction of zero right ascension on the celestial equatorial plane serves as the positive x-axis. Following the right-hand coordinate system, the axis intersecting the x-z plane is the y-axis. Thus, right ascension and declination can be converted to three-dimensional Cartesian coordinates (x, y, z), with the conversion relationships given by equations (4-6) [7].

The distance between two points can then be calculated directly using equation (8). This formula can be simply derived from the Haversine formula, where the part under the square root in the Haversine formula (equation (7)) is actually half the square of the three-dimensional Euclidean distance between the two points.

Regarding the computational accuracy of the Cartesian coordinate method, this paper again uses C++ programs in both single-precision and double-precision environments to compute the formula and compare the results with Haversine, with results shown in Table 2 . As can be seen from Table 2 , the accuracy of the Cartesian coordinate method is almost identical to that of Haversine, and sometimes even superior to Haversine (see the rows for 42,43).

The Cartesian coordinate method can be further optimized for distance comparisons, as shown in equation (9): let the Cartesian coordinates of two points be (x_1, y_1, z_1) and (x_2, y_2, z_2). Its advantage lies in the ability to precompute x, y, z values and the threshold (setting the maximum angular distance as θ_{max}), thereby avoiding the use of time-consuming trigonometric functions during actual distance calculations. Instead, it only requires three subtractions, three multiplications, two additions, and one comparison, significantly reducing computational load and making it highly suitable for large-scale data computation. Correspondingly, its disadvantage is the need for additional storage space to save the three values x, y, and z, which may become problematic in memory-constrained environments. Therefore, the actual choice of formula should still depend on data scale and computational environment. Additionally, there is another form of three-dimensional vector-based calculation method using normal vectors, but its computational process is more complex than described in this section; we will briefly describe it in Section 3.3.

3 Computation Libraries

Many astronomical software packages also include spherical distance calculation functions. This section selects several libraries that are relatively convenient to install and use, and are widely applied. We demonstrate their usage methods and conduct tests. Since it is difficult to distinguish whether single-precision or double-precision is actually used in these packages, this section will no longer differentiate between them, but instead directly compare them with double-precision Haversine results.

3.1 Astropy

Python has become very popular in the astronomical community for data processing, with Astropy [8] contributing significantly to its widespread adoption. To calculate spherical distance with Astropy, one needs to use astropy.units and astropy.coordinates.SkyCoord. The calling method is as follows, where d is the angular distance between two points (in degrees):

```
c1 = SkyCoord(ra1*units.rad, dec1*units.rad)
c2 = SkyCoord(ra2*units.rad, dec2*units.rad)
d = c1.separation(c2).deg
```

For fairness, the Haversine algorithm was reimplemented in 64-bit Python 3.6.3 on 64-bit Windows 10 and compared with Astropy' s distance calculation function. The comparison results are shown in Table 3 , demonstrating that the two methods are essentially consistent and equally matched.

Reading Astropy' s source code reveals that its distance calculation function separation actually calls the angular_separation(lon1, lat1, lon2, lat2) function in astropy/coordinates/angle_utilities.py [5], which implements Vincenty' s formula, shown as equation (10). This formula is more stable for distance calculations at all positions, but its computational complexity is also higher [9]. From the above comparison results, Astropy does not achieve higher precision than Haversine everywhere, as seen in the row for (180,0); of course, Astropy' s precision is also better than Haversine for (42,43). However, the differences between the two are very small and almost negligible. Based on the comparison results, Haversine is already quite accurate. Naturally, if higher computational stability is required, Vincenty' s formula can be considered.

3.2 HTM

The Hierarchical Triangular Mesh (HTM) [10] is a multi-level, recursive triangular partitioning method for the sphere. The publicly available software package for HTM [6] includes a C# code library and a SQL Server extension package. The file Spherical.Htm.Sql.cs contains the function fDistanceEq(ra1, dec1, ra2, dec2) for distance calculation. It should be noted that the return value of fDistanceEq is in arcminutes, not degrees. For HTM, this paper implements the Haversine function in C# and compares it with the fDistanceEq function from

SphericalHTM v3.1.2. As shown in Table 4 , the results are essentially the same. Examining the source code reveals that fDistanceEq also uses the Haversine formula for calculation, with only an additional conversion from degrees to arcminutes.

In addition to directly calling fDistanceEq in C# programs, the SQL Server database extension package provided by the HTM software suite also includes this function, making it very convenient to use in databases.

3.3 SLALIB and SOFA

SLALIB, the Starlink Library of Positional Astronomy Routines [7], is implemented in Fortran 77 with the goal of enabling astronomers to write accurate and reliable astrometric programs more easily and quickly. Although it is implemented in Fortran 77, a Python interface called pyslalib [8] has been developed, allowing one to obtain Fortran 77 computational precision in existing Python environments.

The slalib.sla_dsep in pyslalib directly corresponds to the sla_DSEP (A1, B1, A2, B2) spherical distance calculation function in SLALIB. We compared its results with the Haversine function in a 64-bit Python 3.5.2 environment on 64-bit Ubuntu 16.04.4, as shown in Table 5 . First, we can see that the Python Haversine program on Linux produces results consistent with Windows, demonstrating that Python 3 performs consistently across different operating systems. Second, the pyslalib calculation results are completely consistent with Haversine, except for slight differences in the (42,43) rows, and the precision is also very high.

Examining SLALIB' s source code reveals that it uses the three-dimensional vectors in the Cartesian coordinate system from Section 2 as computational units. However, unlike Section 2, SLALIB further converts them to normal vectors [9], namely $\mathbf{n_1}$ and $\mathbf{n_2}$, and then uses equation (11) [11] for calculation. The main advantage of using vectors is that vector algebra can replace some trigonometric calculations, providing better numerical stability, maintaining good accuracy, and showing no anomalies at boundary points (such as the north and south celestial poles, or the 0-360 degree boundary).

The iauSeps function [11] in the SOFA library [10], commonly used in astrometry, also employs equation (11). SOFA, which stands for Standards of Fundamental Astronomy, is an authoritative fundamental astronomy standard library established and maintained by the IAU, containing both ANSI C and Fortran 77 versions. The rightmost column of Table 5 lists the results calculated by a double-precision C++ program calling SOFA.

4 Databases

Relational databases all support SQL statements, so Haversine formula can be directly implemented using SQL statements for distance calculation across

different systems. Since each database uses slightly different mathematical functions and computational precision, this paper conducted tests on Microsoft SQL Server 2008, PostgreSQL 9.4.5, and MySQL 5.7.18. The test results are shown in Table 6 , from which we can see that the differences among the three databases are not significant, with MySQL retaining more precision by default.

In addition to the Haversine formula, using the Cartesian coordinate method in databases is actually more convenient. Databases have been heavily optimized for I/O and can better schedule data. Therefore, although the Cartesian coordinate method requires adding three fields for x, y, and z, it does not impose too much burden on the database, and can significantly reduce computational complexity, enabling faster calculation results for massive datasets.

4.1 PostgreSQL Database Plugins

In addition to directly using formulas in databases, various database extension plugins can also be used for related calculations, reducing programming effort for users. For example, PostgreSQL database has multiple plugins supporting spherical distance calculation, including PostGIS [12], pgsphere [13], Q3C [14], and H3C [15]. Among them, PostGIS is designed specifically for geographic information systems. Although it can also be used in astronomy, it requires converting units such as meters and kilometers back to radians, which is too cumbersome and inefficient. In contrast, pgsphere, Q3C [12], and H3C are designed specifically for astronomical queries and are more concise. The usage methods of three plugins are demonstrated below, and Table 7 compares the results of these three plugins. It should be noted that Q3C and H3C directly use degrees for calculation, while pgsphere requires conversion to radians first, then conversion back to degrees for the result.

```
SELECT a ra1, b dec1, c ra2, d dec2,
       q3c_dist(a, b, c, d) q3c,
       h3c_dist(a, b, c, d) h3c,
       DEGREES(spoint(RADIANS(a), RADIANS(b)) <-> spoint(RADIANS(c), RADIANS(d))) pg
FROM (VALUES (0.0, 0.0, 0.01, 0.0),
             (0.0, 0.0, 0.0, 0.01),
             (180.0, 0.0, 180.01, 0.0),
             (42.0, 43.0, 42.01, 43.0),
             (42.0, 43.0, 42.0, 43.01),
             (0.0, 90.0, 180.0, 89.99),
             (0.0, 0.0, 0.1, 0.0)) AS v(a, b, c, d)
```

From the results of the three plugins, the accuracy is roughly equivalent with no significant differences, and calculations at the poles also show no major deviations. Examining the source code reveals that the h3c_dist function uses the Haversine formula; the q3c_dist function uses a variant of Haversine; the pgsphere case is more complex, using the Great-circle formula for large distances and the Cartesian coordinate formula for small distances. Based on the

calculation results, all three plugins are mature and usable extensions. The specific choice depends on other requirements. For example, both Q3C and H3C provide join functions (q3c_join and h3c_join respectively), optimized for cross-matching two star catalogs. Meanwhile, pgsphere provides rich spherical geometric calculations, such as area calculations for spherical shapes and intersection/union operations for spherical shapes.

This paper has examined spherical distance calculation methods including the Great-circle formula, simplified Great-circle formula, and Haversine formula, and compared their results in single-precision and double-precision numerical computations. Building upon this, we extended the discussion to distance calculation methods based on three-dimensional Cartesian coordinates. The results show that the Great-circle formula suffers from severe rounding errors under single-precision, while the simplified Great-circle formula produces large errors near the poles or when the distance between two points is large. The Haversine formula and Cartesian coordinate method achieve similar, very high precision, while the Cartesian coordinate method requires less computation and is suitable for large-scale data calculations. Additionally, it should be noted that all these formulas may sometimes produce results slightly larger than the true value. Therefore, in practical applications, it may be necessary to set boundary values slightly larger to include sources that might be missed due to rounding errors.

Existing software packages such as Astropy, HTM, SLALIB, SOFA, and some database extensions also include spherical distance calculation modules. They employ different calculation methods and achieve very high result precision. When spherical distance calculation is needed, these libraries can be used directly, eliminating the need to implement overly complex formulas oneself, thus avoiding redundant work and potential errors due to incomplete consideration.

References

[1] ZHANG Hailong, YE Xinchen, LI Huijuan, et al. Astronomical Data Query and Release Review [J]. Astronomical Research and Technology, 2017, 14(02): 212-228.

[2] ZHANG Hailong, NIE Jun, ZHAO Qing, et al. Xinjiang Astronomical Observatory Data Center Custom Uploading Crossmatcher [J]. Astronomical Research and Technology, 2017, 14(03): 347-355.

[3] ZHANG Yanxia. Research on Automatic Classification Methods in Multi-wavelength Astrophysics [D]. Ph.D. Thesis, 2003.

[4] GAO Dan. Development of Massive Astronomy Data Federation System and Research of Data Mining Algorithms [D]. Ph.D. Thesis, 2008.

[5] ZHAO Qing. Research on High-Efficient Massive Data Oriented Astronomical Cross-Match [D]. Ph.D. Thesis, 2010.

[6] DU Peng. The Cross-match and Aggregation of Large-scale Astronomical

Catalogs Data [D]. M.D. Thesis, 2013.

[7] Gray J, Nieto-Santisteban M A, Szalay A S. The zones algorithm for finding points-near-a-point or cross-matching spatial datasets [J]. arXiv preprint cs/0701171, 2007.

[8] Robitaille T P, Tollerud E J, Greenfield P, et al. Astropy: A community Python package for astronomy [J]. Astronomy & Astrophysics, 2013, 558: A33.

[9] Vincenty T. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations [J]. Survey Review, 1975, 23(176): 88-93.

[10] Kunszt P Z, Szalay A S, Thakar A R. The hierarchical triangular mesh [M]//Mining the Sky. Springer, Berlin, Heidelberg, 2001: 631-637.

[11] Gade K. A non-singular horizontal position representation [J]. The Journal of Navigation, 2010, 63(3): 395-417.

[12] Koposov S, Bartunov O. Q3C, Quad Tree Cube—the new sky-indexing concept for huge astronomical catalogues and its realization for main astronomical queries (cone search and Xmatch) in open source database PostgreSQL [C]//Astronomical Data Analysis Software and Systems XV, 2006, 351: 735.

Note: Figure translations are in progress. See original paper for figures.