Documentation and report of Astrophysics with Artificial Intelligence(Astropy and AstroML) – Astronomical Coordinates with astropy

Donghun Kim

* **All the information is based on and originated from ‘astropy.org’, ‘atsroml.org’, and ‘wikipedia.org’**

<1> The fundamental knowledge to utilize astropy and astroML for astrophysics

1. Coordinates system.

There are some coordinate systems such as Cartesian coordinate system, cylindrical coordinate system, and spherical coordinate system. Those coordinates are related each other ones and also can be alternated to clarify the object position or calculate something which are changed depending on the coordinate system. Those coordinate systems have each own defined units and also each units can be converted along the process of alternation of coordinate system.

* The sort of the coordinate systems commonly utilized in astronomical community.

1. Horizontal system

This coordinate system is employing the observer’s local horizon as the fundamental plane to measure the location of the observed star. This coordinate system can be expressed with ‘Altitude’ and ‘Azimuth’. The altitude is the angle which is measured by observer from local horizon to observed celestial body, and the azimuth is the angle between the projected vector and the reference vector on the reference plane which is measured from observer’s point to interest point projected perpendicularly onto a reference plane. In addition to it, instead of the altitude, it is possible to utilize the zenith distance which is the angle comparable with the value which is subtracted the altitude (elevation) from the 90 degree.



(Image from: https://en.wikipedia.org/wiki/Horizontal\_coordinate\_system)

1. Equatorial system.

This coordinate system can be implemented in spherical or rectangular coordinates and these ones are defined by the origin at the center of the Earth, the fundamental plane which is projected the Earth’s equator onto the celestial sphere, which is forming the celestial equator, primary direction toward the vernal equinox, and right-handed convention (anti-clockwise direction), which means that the coordinates increase along the northward and the eastward around the fundamental plane.



(Image from: <https://en.wikipedia.org/wiki/Equatorial_coordinate_system>)

1. Ecliptic system

This coordinate system is generally utilized in Solar System to represent the object’s apparent position, orbit, and pole orientation by employing the ecliptic and vernal equinox. This is because almost of the planets and small bodies in Solar system has orbits which are inclined to the ecliptic. The center of this coordinate can be set as one of Sun or Earth. This can be implemented in spherical or rectangular coordinates. However, the significant point is that the celestial equator and the ecliptic are slowly alternated because of the perturbing forces, which means that the primary direction is influenced, so this is not fixed. The slow motion of the Earth, known as axial precession, causes alternation of the coordinate system, which changes the coordinate system westward about the poles of the ecliptic slowly, but continuously. The completion of one circuit of this phenomenon is about 26,000 years. To consider the fixed position in space when the position is given in ecliptic coordinate system, this motion requires the specification of the equinox of a particular date, known as ‘epoch’. The most general methods are:

1. Mean equinox of a standard epoch

This is a fixed standard direction. This allows the various positions which are set at various dates to be compared.

1. Mean equinox of date

This utilizes the ecliptic of date and mean equator and those intersections. The ecliptic of date is the ecliptic in its position at date and the mean equator is rotating by precession to its position at date, but frees from the small sporadic oscillations of nutation.

1. True Equinox of date

Similar with the ‘Mean equinox of date’, this also employs the ecliptic of date and the equator, but this makes use of the true equator. In other words, in contrast with the Mean equinox of date, this equator includes the nutation. This is actual intersection of the two planes considered all motions and particular moment.



(Image from: <https://en.wikipedia.org/wiki/Ecliptic_coordinate_system>)

1. Galactic coordinate system

This coordinate system utilizes the two elements as the measurement elements, which are the ‘galactic longitude’ and ‘galactic latitude’ to find the target celestial body’s position. This system is generally employed to search the celestial body’s position and distribution. This also utilizes the degree, arcminute(arc-min), and arcsecond(arc-sec) to express the unit values of galactic longitude and latitude.

1. Galactic longitude

This measures the angular distance of the object eastward along the galactic equator from the galactic center.

1. Galactic latitude

This measures the angle of the object northward of the galactic equator.

This coordinate system also utilizes Sun as the center of the coordinate, the primary direction aligned with the approximate center of the Milky Way, and the fundamental plane parallel to an approximation of the galactic plane as the measure elements. Furthermore, this uses the right-handed convention, which means that it has positive value toward the north and east in the fundamental plane.



(Image from: https://en.wikipedia.org/wiki/Galactic\_coordinate\_system)

1. Super-galactic coordinate system

This coordinate system is the spherical coordinate system in the equator is the super-galactic plane. The characteristics of this are that the zero point of this coordinate system is where the super-galactic plane intersects with the galactic plane. Because of that the plane is observed from Earth, the plane passes Earth. Conventionally, super-galactic latitude is generally abbreviated SGB and longitude as SGL which analogies to ‘b’ and ‘l’ which are conventionally utilized in galactic coordinate system. Therefore, the zero point can be expressed SGB = 0°, SGL = 0°, lx = 137.37°, and bx = 0°. To be more specific, the north super-galactic pole (SGB = 90°) is laid at lz = 47.37° and bz = +6.32° at galactic coordinate system.



(image from: https://en.wikipedia.org/wiki/Supergalactic\_coordinate\_system)

1. Conversion of the various coordinate systems.

To begin with, each coordinate system has different units to express and calculate the object’s position.

|  |  |  |
| --- | --- | --- |
| Coordinate system | Symbol | Mean |
| Horizontal coordinate system | A  h | azimuth  altitude |
| Equatorial coordinate system | α  δ  ω | right ascension  declination  hour angle |
| Ecliptic coordinate system | *λ*  *β* | ecliptic longitude  ecliptic latitude |
| Galactic coordinate system | *l*  *b* | galactic longitude  galactic latitude |
| Miscellaneous | *λ*o  *ϕ*o  *ε*  *θ*L  *θ*G | observer’s longitude  observer’s latitude  obliquity of the ecliptic(about  local sidereal time  Greenwich sidereal time |

1. Hour angle and right ascension

h = *θ*L – *α = θ*G + *λ*o – *α*

*α = θ*L – h = *θ*G + *λ*o – h

1. Equatorial and ecliptic

The classical equations of the longitudinal coordinate which are stemmed from Spherical trigonometry are

cos(β)sin(λ) = cos(δ)sin(α)cos(ε) + sin(δ)sin(ε)

cos(β)cos(λ) = cos(δ)cos(α)

sin(β) = sin(δ)cos(ε) – cos(δ)sin(ε)sin(α)

If divide the first equation with second one, the result equation is

tan(λ) =

However, this division is unclear because the period of tan is 180° (*π)*, but the ones of sin and cos are the 360° (2*π*).

The rotation matrix equivalent over above equations is that:

In addition to,

cos(δ)sin(α) = cos(β)sin(λ)cos(ε) – sin(β)sin(ε)

cos(δ)cos(α) = cos(β)cos(λ)

sin(δ) = sin(β)cos(ε) + cos(β)sin(ε)sin(λ)

tan(α) =

Likewise, the rotation matrix equivalent is

1. Equatorial and horizontal

The azimuth (A) is measured from the south point and alternated along the western. Zenith distance is the angular distance along the great circle from the zenith to a celestial object and complementary angle of the altitude which can be expressed as 90° - a.

cos(a)sin(A) = cos(δ)sin(h)

cos(a)cos(A) = cos(δ)cos(h)sin(*ϕ*o) – sin(δ)cos(*ϕ*o)

sin(a) = sin(*ϕ*o)sin(δ) + cos(*ϕ*o)cos(δ)cos(h)

Divide the first equation by second one.

tan(A) =

To evaluate the A, it is necessary to resolve the tan(A) and, in this process, it is needed to make use of the ‘two-argument arctangent’ to avoid the ambiguity of the arctangent. This calculates the arctangent of y /x and accounts for the quadrant which is computed. The azimuth is measured from south and opening positive to the west.

A = -arctan(x, y)

(x = -sin(*ϕ*o)cos(δ)cos(h) + cos(*ϕ*o)sin(δ),

y = cos(δ)sin(h) )

If the value of A is produced a negative one, it is possible to convert it to positive one by adding 360°. The equation and the rotation matrix equivalent are

cos(δ)sin(h) = cos(a)sin(A)

cos(δ)cos(h) = sin(a)cos(*ϕ*o) + cos(a)cos(A)sin(*ϕ*o)

sin(δ) = sin(*ϕ*o)sin(a) – cos(*ϕ*o)cos(a)cos(A)

tan(h) =

Comparable with the all above process, to calculate the value ‘h’ in the tan(h), utilize the two-argument arctangent.

h = arctan(x, y)

(x = sin(*ϕ*o)cos(a)cos(A) + cos(*ϕ*o)sin(a),

y = cos(a)sin(A) )

The rotation matrix equivalent is

1. Equatorial and galactic

The equations converting the equatorial coordinates to the galactic coordinates are

cos(lNCP – l)cos(b) = sin(δ)cos(δG) – cos(δ)sin(δG)cos(α – αG)

sin(lNCP – l)cos(b) = cos(δ)sin(α – αG)

sin(b) = sin(δ)sin(δG) + cos(δ)cos(δG)cos(α – αG)

(αG, δG : the equatorial coordinates of the North Galactic Pole,

lNCP : the Galactic longitude of the North Celestial Pole)

The values of the various in above equations about ‘J2000.0’, known as Epoch’, as an example,

αG : 192.85948°, δG : 27.12825°, lNCP : 122.93192°

For other positions, it is prerequisite to be precessed to their position at J2000.0 before applying above formulae. For instance, applying that over the B2000.0 is

sin(α – αG)cos(δ) = cos(b)sin(lNCP – l)

cos(α – αG)cos(δ) = sin(b)cos(δG) – cos(b)sin(δG)cos(lNCP – l)

sin(δ) = sin(b)sin(δG) + cos(b)cos(δG)cos(lNCP – l)

1. Angular distance

The angular distance is also known as angular separation, apparent distance and apparent separation, which is the angle between two sightlines or two points objects as viewed from observer.

Presuppose that there are two astronomical objects A and B located on the surface of the sphere which seen from the center of the sphere and observed from the Earth, and O the observer on Earth which is located at the center of celestial sphere. Each object has the units of celestial coordinates, right ascension (RA) and declination (dec), which units are restrained like (αA, αB) ∈ [0, 2π] and (δA, δB) ∈ [-π/2, π/2].

The scalar product, also known as dot product, of OA and OB is

OA∙OB = R2 cos(θ)

= nA∙nB = cos(θ)

The unitary vectors in the 3-dimension coordinate (x, y, z) are

nA = , nB =

Therefore, the equation of scalar product of OA and OB can be calculated like

nA∙nB = cos(δA)cos(αA)cos(δB)cos(αB) + cos(δA)sin(αA)cos(δB)sin(αB) + sin(δA)sin(δB) = cos(θ)

If compute above equation about the ‘θ’, the equation is

θ = cos-1[sin(δA)sin(δB) + cos(δA)cos(δB)cos(αA - αB)]

If the angular distance is small enough, the equation can be simplified. In other words, if the objects are close in the sky like following cases: the stars in a telescope field of view, binary stars, or the satellites of the giant planets of the solar system, it is possible to employ the small-angle approximation. If the angel is θ ≪ 1 [rad], αA - αB ≪ 1, or δA - δB ≪ 1, it is enough to consider the angle is small to utilize the approximation.

cos(θ) ≈ 1 –

≈ sin(δA)sin(δB) + cos(δA)cos(δB) [1 – ]

≈ cos(δA - δB) – cos(δA)cos(δB)

≈ 1 – – cos(δA)cos(δB)

∴ cos(δA)cos(δB) ≈ cos2(δA)

∴ θ ≈

When it comes to the planar approximation, for applying of this, it also requires the condition which the angular distance is small enough. If consider that detector imaging a small sky field which has the α (right ascension) which parallels to the meridian with y-axis and δ (declination) which parallels along with the x-axis, the angular separation is

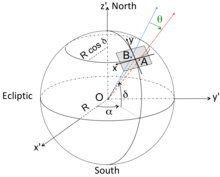
θ ≈

(δx = (αA – αB) cos(δA),

δy = δA – δB)

In above equation, the y-axis is equal with the declination, but the x-axis is right ascension which is modulated by cos(δA). This is because, in the section of sphere, the radius (R) at delination (latitude) δ is

R’ = Rcos(δA)



(image from: https://en.wikipedia.org/wiki/Angular\_distance)

1. Signal-to-noise ratio

The signal-to-noise ratio (SNR) is the proportion of the power of a signal to the power of background noise.

SNR =

(SNR: signal-to-noise ratio,

ps: average signal power,

pn: average noise power)

In above equation, each average power (P) is measured at equivalent points and within comparable bandwidth.

The SNR for random noise N is alternated, depending on whether the signal is a constant (s) or a random variable (S) like

SNR = =

(E: expected value)

The denominator is its variance, the square of tis standard deviation σN when the noise has expected value of zero.

Each value of signal and noise should be measured as equal method and points, and the root mean squares (RMS) can be employed as alternative way.

SNR = =

(A: root mean square amplitude)

The unit value of SNR, noise, and signal can have decibel [dB] one.

P­s [dB] = 10 log10(Ps)

Pn [dB] = 10 log10(Pn)

SNR [dB] = 10 log10 = 10 log10(SNR)

If utilize the quotient rule for logarithms, it is possible to express the SNR equation more specific.

10 log10 = 10 log10(Ps) – 10 log10(Pn)

∴ SNR [dB] = Ps [dB] - Pn [dB]

In the above equation, the values are measured as P which has the power as the unit value and SNR is pure number. However, if the values of variables which are the components of the SNR ratio formula –signal and noise- are measured in amplitude scale –volts or amperes-, the equation which makes use of the root mean squares is necessary.

SNR [dB] = 10 log10 = 20 log10 = (As [dB] - An [dB])

The average power of an AC (alternating current) signal is defined as the multiplication of rms values of voltage with current, which is also calculated by resistive with rms value of voltage times or with current.

P = Vrms Irms = =

To be more specific, the moment power supply which is the AC power transmits to the circuit is equivalent with the product of current with applied voltage. The moment power supply equation is

P = iΔv = Imaxsin(ωt) ΔVmaxsin(ωt + ø)

= ImaxΔVmaxsin2(ωt)sin(ωt + ø)

If employ to the trigonometric identity, sin(ωt + ø) = sin(ωt)cos(ø) + cos(ωt)sin(ø) and sin(ωt)cos(ωt) = ,

P = Imax ΔVmaxsin2(ωt)cos(ø) + Imax ΔVmaxsin(ωt)cos(ωt)sin(ø),

∴ Pavg =

<2> Developing the programs

(<https://learn.astropy.org/rst-tutorials/1-Coordinates-Intro.html?highlight=filtertutorials#exercises>)

First of all, it is essential requisite to import the packages for making the program.

(Code)

**import** matplotlib **as** mpl

**import** matplotlib.pyplot **as** plt

**%**matplotlib inline

**import** numpy **as** np

**from** astropy **import** units **as** u

**from** astropy.coordinates **import** **(**SkyCoord**,** Distance**,** Galactic, EarthLocation**,** AltAz**)**

**import** astropy.coordinates **as** coord

**from** astropy.io **import** fits

**from** astropy.table **import** QTable

**from** astropy.utils.data **import** download\_file

**from** astroquery.gaia **import** Gaia

Gaia**.**ROW\_LIMIT **=** **10000** *# Set the row limit for returned data*

The ‘Gaia.ROW\_LIMIT’ allows the data to be returned within limited range.

(Output)

Created TAP**+** **(**v20200428**.1)** **-** Connection**:**

Host**:** gea**.**esac**.**esa**.**int

Use HTTPS**:** **True**

Port**:** **443**

SSL Port**:** **443**

Created TAP**+** **(**v20200428**.1)** **-** Connection**:**

Host**:** gea**.**esac**.**esa**.**int

Use HTTPS**:** **True**

Port**:** **443**

SSL Port**:** **443**

Utilize open star cluster NGC188 as an example to create the SkyCoord object, which allows to query and find the stars which are the members of the cluster. In this assumption, the sky coordinate of the cluster has the (12.11, 85.26) degrees in the ICRS coordinate system. Therefore, by passing these values to the initializer of the method (SkyCoord) with the proper unit values, it is possible to create the object which can denote the components with each unit values in that coordinate system.

(Code)

ngc188\_center **=** SkyCoord**(12.11\***u**.**deg**,** **85.26\***u**.**deg**)**

ngc188\_center **=** SkyCoord**(12.11\***u**.**deg**,** **85.26\***u**.**deg**,** frame**=**'icrs'**)**

(Output)

**<**SkyCoord **(**ICRS**):** **(**ra**,** dec**)** **in** deg

**(12.11,** **85.26)>**

**<**SkyCoord **(**ICRS**):** **(**ra**,** dec**)** **in** deg

**(12.11,** **85.26)>**

Above input and output show that the default unit value of the ‘SkyCoord’ is the ICRS coordinate frame and if the values are entered, the result variable is adjusted to have proper unit value in ICRS coordinate system. In above codes, the variable ‘ngc188\_center’ has (12.11, 85.26) value which is the longitude and latitude values on Earth, with ‘deg’ as a unit, so this is alternated as ‘ra(right ascension)’ and ‘dec(declination)’.

If the parameters which are entered in method ‘SkyCoord’ are the ‘hms’ (hour-minute-second) and ‘dms’ (degrees-minute-second), this is also alternated to have the proper unit value comparable with the set coordinate frame. For instance,

(Code)

SkyCoord**(**'00h48m26.4s'**,** '85d15m36s'**,** frame**=**'icrs'**)**

SkyCoord**(**'00:48:26.4 85:15:36'**,** unit**=(**u**.**hour**,** u**.**deg**),**

frame**=**'icrs'**)**

(Output)

**<**SkyCoord **(**ICRS**):** **(**ra**,** dec**)** **in** deg

**(12.11,** **85.26)>**

**<**SkyCoord **(**ICRS**):** **(**ra**,** dec**)** **in** deg

**(12.11,** **85.26)>**

The exact coordinate of the astronomical object ‘NGC 188’ can be checked by utilizing the method ‘SkyCoord.from\_name()’

(Code)

ngc188\_center **=** SkyCoord**.**from\_name**(**'NGC 188'**)**

(Output)

**<**SkyCoord **(**ICRS**):** **(**ra**,** dec**)** **in** deg

**(12.10833333,** **85.255)>**

Another example of that

(Code)

a0620\_center = SkyCoord.from\_name('A0620')

a0620\_center = SkyCoord('08h05m43.20s','45d40m58.0s', frame**=**'icrs')

a0620\_center = SkyCoord('08:05:43.20', '45:40:58.0', unit**=**(u.hour, u.deg),

frame**=**'icrs')

(Output)

**<**SkyCoord **(**ICRS**):** **(**ra**,** dec**)** **in** deg

**(121.43,** **45.68277778)>**

About the example object ‘NGC 188’, the output values represent the components having ra(Right ascension) and dec(declination) as the unit values. Because, in the parameter of the ‘SkyCoord’, the key ‘frame’ is the ‘icrs’, those components are returned specialized values if print that with each unit values.

(Code)

ngc188\_center**.**ra**,** ngc188\_center**.**dec

(Output)

**(<**Longitude **12.10833333** deg**>,** **<**Latitude **85.255** deg**>)**

This shows that if the key ‘frame’ has the ‘icrs’ value, the result of returned components ‘ra’ and ‘dec’ are alternated longitude and latitude values and this can be verified by checking each sort with method ‘type()’ like

(Code)

type**(**ngc188\_center**.**ra**),** type**(**ngc188\_center**.**dec**)**

(Output)

**(**astropy**.**coordinates**.**angles**.**Longitude**,** astropy**.**coordinates**.**angles**.**Latitude**)**

The way to retrieve the different values in each nonidentical coordinate system is employing the ‘Quantitiy.to()’ method which offered by astropy.

(Code)

**(**ngc188\_center**.**ra**.**to**(**u**.**hourangle**),**

ngc188\_center**.**ra**.**to**(**u**.**radian**),**

ngc188\_center**.**ra**.**to**(**u**.**degree**))**

**or**

**(**ngc188\_center**.**ra**.**hour**,**

ngc188\_center**.**ra**.**radian**,**

ngc188\_center**.**ra**.**degree**)**

(Output)

**(<**Longitude **0.80722222** hourangle**>,**

**<**Longitude **0.21133028** rad**>,**

**<**Longitude **12.10833333** deg**>)**

**or**

**(0.8072222220000002,** **0.2113302835374691,** **12.10833333)**

Moreover, it is possible to return that value separated form in hourangle and degree excepting the radian because the former units are composed with (hour, minute, second) and (degree, minute, second) forms, but the latter’s one only has the radian unit in right ascension (ra) – hourangle: (0h48m26s) and degree: (12d06m30s), but radian: (0.21133rad).

(Code)

ngc188\_center**.**ra**.**to\_string**(**unit**=**u**.**hourangle**,** sep**=**':'**,** pad**=True)**

ngc188\_center**.**ra**.**to\_string**(**unit**=**u**.**degree**,** sep**=**':'**,** pad**=True)**

(Output)

'00:48:26'

'12:06:30'

Bring the ‘Gaia Archive’ to retrieve the coordinates of stars and to query the databases. For this process, there are two ways- one of them is utilizing the Internet and use the package and another one is loading that locally. This process helps to find the stars which are around the position of the center of NGC 188 and to distinguish that whether that are the members of that cluster or not.

(Code)

(using the Internet)

job **=** Gaia**.**cone\_search\_async**(**ngc188\_center**,** radius**=0.5\***u**.**deg**)**

ngc188\_table **=** job**.**get\_results**()**

*# only keep stars brighter than G=19 magnitude*

ngc188\_table **=** ngc188\_table**[**ngc188\_table**[**'phot\_g\_mean\_mag'**]** **<** **19\***u**.**mag**]**

(Output)

INFO**:** Query finished**.** **[**astroquery**.**utils**.**tap**.**core**]**

(Code)

cols **=** **[**'source\_id'**,**

'ra'**,**

'dec'**,**

'parallax'**,**

'parallax\_error'**,**

'pmra'**,**

'pmdec'**,**

'radial\_velocity'**,**

'phot\_g\_mean\_mag'**,**

'phot\_bp\_mean\_mag'**,**

'phot\_rp\_mean\_mag'**]**

ngc188\_table**[**cols**].**write**(**'gaia\_results.fits'**,** overwrite**=True)**

(Code)

(without the Internet connection)

ngc188\_table **=** QTable**.**read**(**'gaia\_results.fits'**)**

Those two ways are the comparable methods to load the data, query databases, and create the table ‘ngc188\_table’. By checking the length of above variables, it is possible to confirm that two methods are the same.

(Code)

len**(**ngc188\_table**)**

(Output)

**4938**

About 5000 stars are included in table ‘ngc188\_table’ from Gaia Data Release 2 catalog around the position of the ‘ngc188\_center’ and, in the database, the ICRS coordinate are given as column names ‘ra’ and ‘dec’.

(Code)

ngc188\_table**[**'ra'**]**

ngc188\_table**[**'dec'**]**

(Output)

ra : [12.09832966 12.14694473 12.05460389 ... 6.15840244 11.71137094 10.89179821] deg

dec : [85.25443128 85.25475081 85.25636896 ... 85.32562027 85.74769043 84.77264889] deg

The output values have degree as the unit because the database Gaia Archive data tables has the associated units and those values are read by employing the ‘QTable’.

Likewise above processes which inputs the values as the parameters of the ‘SkyCoord’ method (longitude and latitude), these output lists can be inserted ones of that.

(Code)

ngc188\_gaia\_coords **=** SkyCoord**(**ngc188\_table**[**'ra'**],** ngc188\_table**[**'dec'**])**

ngc188\_gaia\_coords

(Output)

**<**SkyCoord **(**ICRS**):** **(**ra**,** dec**)** **in** deg

**[(12.09832966,** **85.25443128),** **(12.14694473,** **85.25475081),**

**(12.05460389,** **85.25636896),** **...,** **(** **6.15840244,** **85.32562027),**

**(11.71137094,** **85.74769043),** **(10.89179821,** **84.77264889)]>**

This means that the variable ‘ngc188\_gaia\_coords’ has the 4938 number of stars’ positional information of right ascension and declination.

Call the object which contains the results of Gaia query and calculate the angular separation between each resulting star in such an object and the coordinate of the cluster for NGC 188 by utilizing the method ‘SkyCoord.separation()’. As a result, the list composed with the values of angular distance (also known as angular separation, apparent distance, or apparent separation).

(Code)

ngc188\_gaia\_coords**.**separation**(**ngc188\_center**)**

(Output)

[**0d00m03.6149s 0d00m11.5336s 0d00m16.74s ... 0d29m36.1677s 0d29m37.2132s 0d29m37.5986s**]

The distance is useful information with angular positions to represent the full 3-dimension coordinate. The distance of the example object NGC 188 is the 1.96 [kpc] (6.0479 x 1013 [km]). The dataset Gaia has the information of parallaxes of the stars around the NGC 188. From the parallax(ϖ), the distance(d) can be computed like, d ≈ 1/ϖ. However, this equation is approximate result, so it really works when the parallax error is smaller relating with the parallax.

The first step to retrieve the distance utilizing the parallax is filtering out the stars which have low signal-to-noise parallaxes (refer the theoretical formula ‘4. Signal-to-noise’).

(Code)

*# insert the distance*

gc188\_center\_3d **=** SkyCoord**(12.11\***u**.**deg**,** **85.26\***u**.**deg**,**

distance**=1.96\***u**.**kpc**)**

*# filter out the stars having low signal-to-noise parallaxes*

parallax\_snr **=** ngc188\_table**[**'parallax'**]** **/** ngc188\_table**[**'parallax\_error'**]**

ngc188\_table\_3d **=** ngc188\_table**[**parallax\_snr **>** **10]**

len**(**ngc188\_table\_3d**)**

(Output)

**2053**

The catalog of the stars queried from Gaia in above codes has the parallax information in milliarcsecond unit, which means that from this query, distance object can be created. Then, passing the distance values from distance object to the initializer allows to represent the 3-dimension positions of all stars of the Gaia.

(Code)

*# get the parallax information*

gaia\_dist **=** Distance**(**parallax**=**ngc188\_table\_3d**[**'parallax'**])**

*# pass the distance object as a parameter of the SkyCoord initializer*

ngc188\_coords\_3d **=** SkyCoord**(**ra**=**ngc188\_table\_3d**[**'ra'**],**

dec**=**ngc188\_table\_3d**[**'dec'**],**

distance**=**gaia\_dist**)**

ngc188\_coords\_3d

(Output)

**<**SkyCoord **(**ICRS**):** **(**ra**,** dec**,** distance**)** **in** **(**deg**,** deg**,** pc**)**

**[(12.09832966,** **85.25443128,** **1109.14781668),**

**(12.12843989,** **85.26076889,** **1835.66340989),**

**(12.07705917,** **85.24631009,** **1974.19410857),** **...,**

**(** **6.13894786,** **85.25954928,** **2532.64969255),**

**(** **6.15840244,** **85.32562027,** **1795.07434822),**

**(11.71137094,** **85.74769043,** **367.13489233)]>**

In contrast to the ‘ngc188\_gaia\_coords’, this includes the distance values in parsec [pc] unit.

Visualize above values by utilizing the matplotlib.

(Code)

fig**,** ax **=** plt**.**subplots**(**figsize**=(6.5,** **5.2),**

constrained\_layout**=True)**

cs **=** ax**.**scatter**(**ngc188\_coords\_3d**.**ra**.**degree**,**

ngc188\_coords\_3d**.**dec**.**degree**,**

c**=**ngc188\_coords\_3d**.**distance**.**kpc**,**

s**=5,** vmin**=1.5,** vmax**=2.5,** cmap**=**'twilight'**)**

cb **=** fig**.**colorbar**(**cs**)**

cb**.**set\_label**(**f'distance [{u**.**kpc:latex\_inline}]'**)**

ax**.**set\_xlabel**(**'RA [deg]'**)**

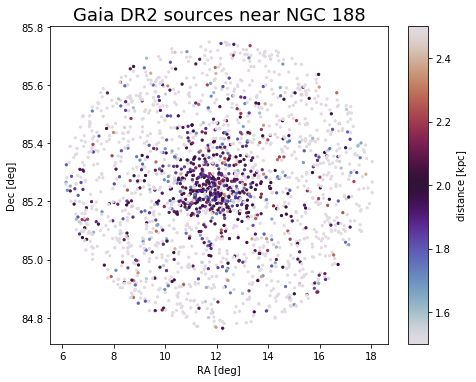
ax**.**set\_ylabel**(**'Dec [deg]'**)**

ax**.**set\_title**(**'Gaia DR2 sources near NGC 188'**,** fontsize**=18)**

(Output)

Text**(0.5,** **1.0,** 'Gaia DR2 sources near NGC 188'**)**

(Plot)



Calculate the 3D separation (distance) between all of the Gaia sources and the cluster center.

(Code)

sep3d **=** ngc188\_coords\_3d**.**separation\_3d**(**ngc188\_center\_3d**)**

sep3d

(Output)

**[850.85219575 124.33660295 14.20219631 ... 572.97077442 165.71210705**

**1592.88153948]**

Selecting the candidate members which have different specific value with above ones is possible. For example, if select the candidate members which are 50pc and 100pc away from the cluster center, that is select all stars within 50pc and 100pc of the cluster center:

(Code)

*# within 50pc*

ngc188\_3d\_mask **=** sep3d **<** **50\***u**.**pc

ngc188\_3d\_mask**.**sum**()**

*# within 100pc*

ngc188\_3d\_mask **=** sep3d **<** **100\***u**.**pc

ngc188\_3d\_mask**.**sum**()**

(Output)

**193**

**358**

The number of the stars within 50pc from the cluster center is 193 and 358 regarding 100pc out of 2053 numbers of stars.

(<https://learn.astropy.org/rst-tutorials/2-Coordinates-Transforms>)

Set the variable by passing the information data into the SkyCoord initializer.

(Code)

c **=** SkyCoord**(**ra**=15.9932\***u**.**deg**,** dec**=-10.52351344\***u**.**deg**)**

print**(**c**.**ra**.**hourangle**)**

print**(**c**.**to\_string**(**'hmsdms'**))**

print**(**c**.**dec**.**to\_string**(**sep**=**':'**,** precision**=5))**

(Output)

**1.0662133333333335**

**01**h03m58**.368**s **-10**d31m24**.6484**s

**-10:31:24.64838**

To convert or to represent the information of coordinate to other frame such as Cartesian, cylindrical, or spherical, utilize the ‘SkyCoord.represent\_as(‘name’)’ or ‘SkyCoord.represent\_as(coord.name)’.

(Code)

c**.**represent\_as**(**'cartesian'**)**

c**.**represent\_as**(**'spherical'**)**

c**.**represent\_as**(**'cylindrical'**)**

**or**

c**.**represent\_as**(**coord**.**CartesianRepresentation**)**

c**.**represent\_as**(**coord**.)**

c**.**represent\_as**(**coord**.)**

(Output)

**<**CartesianRepresentation **(**x**,** y**,** z**)** **[**dimensionless**]**

**(0.94512547,** **0.27088898,** **-0.18263903)>**

(<https://learn.astropy.org/rst-tutorials/3-Coordinates-Velocities.html?highlight=filtertutorials>)

(<https://learn.astropy.org/rst-tutorials/4-Coordinates-Crossmatch.html?highlight=filtertutorials>)