# Package 'celestial'

September 2, 2025

**Type** Package

**Title** Collection of Common Astronomical Conversion Routines and Functions

**Version** 1.5.8

**Date** 2025-09-02

**Description** Contains a number of common astronomy utility functions for cosmology and angular coordinates.

**License** GPL-3

**LazyData** TRUE

**Depends** R (>= 3.00), RANN, NISTunits, pracma

**Suggests** plotrix, igraph

**LinkingTo** Rcpp

**NeedsCompilation** yes

**Author** Aaron Robotham [aut, cre]

**Maintainer** Aaron Robotham <aaron.robotham@uwa.edu.au>

**Repository** CRAN

**Date/Publication** 2025-09-02 08:00:02 UTC

## Contents

---

`celestial-package` *Collection of Common Astronomical Conversion Routines and Functions*

---

## Description

Various functions for converting between commonly used coordinate systems in astronomy and making cosmological calculations.

## Details

| | |
|---|---|
| Package: | celestial |
| Type: | Package |
| Version: | 1.5.6 |
| Date: | 2023-11-23 |
| License: | GPL-3 |
| Depends: | R (>= 3.00), RANN, NISTunits, pracma |
| Suggest: | plotrix, igraph |

There are a number of functions included, but the most useful for astronomy conversions are the decimal degrees to DMS/HMS formats used at many telescopes: deg2dms, deg2hms, dms2deg, hms2deg. It also contains functions for various cosmological calculations (i.e. distance, volume and age for different cosmologies and redshifts).

## Author(s)

Aaron Robotham

Maintainer: Aaron Robotham <aaron.robotham@uwa.edu.au>

`car2sph` *Transforms 3D cartesian coordinates to spherical coordinates*

## Description

Transforms 3D cartesian coordinates to spherical coordinates. The user can choose to return the spherical coordinates in degrees or radians.

## Usage

```
car2sph(x, y, z, deg = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | x values, can also contain a matrix of x, y and z (in that order). |
| `y` | y values. |
| `z` | z values |
| `deg` | Should degrees be returned (default) or radians. |

## Details

This is a low level function that is used for plot transformations.

## Value

A data.frame is returned containing the columns long (longitude), lat (latitude) and radius.

## Author(s)

Aaron Robotham

## See Also

`sph2car`

## Examples

```
print(car2sph(x=1,y=1,z=0,deg=TRUE))
```

cosdist *Cosmological distance calculations*

## Description

These functions allow comoving, angular size and luminosity distances to be calculated for a given redshift, it can also return look back time. They use curvature correctly, calculated internally using the relation OmegaM+OmegaL+OmegaR+OmegaK=1, but by default they assume a flat Universe where only OmegaM needs to be specified and OmegaR=0 (so no radiation pressure at any epoch).

## Usage

```
cosdist(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1, wprime = 0,
age=FALSE, ref, error=FALSE)
cosdistz(z=1)
cosdistzeff(zref = 1, zem = 2)
cosdista(z=1)
cosdistCoDist(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1,
wprime = 0, ref)
cosdistLumDist(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1,
wprime = 0, ref)
cosdistAngDist(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1,
wprime = 0, ref)
cosdistAngDist12(z1=1, z2=2, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0,
w0 = -1, wprime = 0, ref)
cosdistCoDistTran(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1,
wprime = 0, ref)
cosdistCoDist12ang(z1=1, z2=2, ang=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, w0 = -1, wprime = 0, inunit='deg', ref)
cosdistLumDist12ang(z1=1, z2=2, ang=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, w0 = -1, wprime = 0, inunit='deg', ref)
cosdistAngDist12ang(z1=1, z2=2, ang=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, w0 = -1, wprime = 0, inunit='deg', ref)
cosdistzem12ang(z1=1, z2=2, ang=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, w0 = -1, wprime = 0, inunit='deg', ref)
cosdistzeff12ang(z1=1, z2=2, ang=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, w0 = -1, wprime = 0, inunit='deg', ref)
cosdistDistMod(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1,
wprime = 0, ref)
cosdistAngScale(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1,
wprime = 0, ref)
cosdistAngSize(z=1, Size=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1,
wprime=0, Dim=1, Dist='Co',  outunit='deg', ref)
cosdistAngArea(z=1, Size=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1,
wprime=0, Dim=2, Dist='Co',  outunit='deg2', ref)
cosdistCoVol(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1,
wprime = 0, ref)
```

```
cosdistHubTime(H0=100)
cosdistUniAgeNow(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1,
wprime = 0, ref)
cosdistUniAgeAtz(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1,
wprime = 0, ref)
cosdistTravelTime(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1,
wprime = 0, ref)
cosdistRelError(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1, wprime = 0,
ref)
cosdistCrit(z_lens=1, z_source=2, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0,
w0 = -1, wprime = 0, ref)
```

## Arguments

| | |
|---|---|
| z | Cosmological redshift, where z must be > -1 (can be a vector). |
| H0 | Hubble constant as defined at z=0 (default is H0=100 (km/s)/Mpc). |
| OmegaM | Omega Matter as defined at z=0 (default is 0.3). |
| OmegaL | Omega Lambda as defined at z=0 (default is for a flat Universe with OmegaL = 1-OmegaM-OmegaR = 0.7). |
| OmegaR | Omega Radiation as defined at z=0 (default is 0, but OmegaM/3400 is typical). |
| w0 | The value of dark energy equation of state as defined at z=0. See cosgrow for more details. |
| wprime | The evolution term that governs how the dark energy equation of state evolves with redshift. See cosgrow for more details. |
| age | Flag for cosdist function to return age or not- this slows calculation, so is by default turned off. |
| ref | The name of a reference cosmology to use, one of 137 / 737 / Planck / Planck13 / Planck15 / Planck18 / WMAP / WMAP9 / WMAP7 / WMAP5 / WMAP3 / WMAP1 / Millennium / GiggleZ. Planck = Planck18 and WMAP = WMAP9. The usage is case insensitive, so wmap9 is an allowed input. This overrides any other settings for H0, OmegaM and OmegaL. If OmegaR is missing from the reference set then it is inherited from the function input (0 by default). See cosref for details. |
| error | Flag for cosdist to calculate the relative error for distance/age values. |
| z1 | Redshift for object 1, where z1 must be > -1 (can be a vector) and less than z2. |
| z2 | Redshift for object 2, where z2 must be > -1 (can be a vector) and greater than z1. |
| zref | Redshift for the reference object, i.e. the object that we caste as the observer of another object at zem. |
| zem | Redshift for the emitting object, i.e. the object that we caste as being observed by another object at zref. |
| z_lens | Redshift, where z_lens must be > -1 (can be a vector) and z_lens < z_source. |
| z_source | Redshift, where z_source must be > -1 (can be a vector) and z_lens < z_source. |
| ang | The observed angular separation between object 1 and object 2 in degrees. |

| | |
|---|---|
| `Size` | The 1D size of the object (i.e. diameter or total length) in Mpc. Either comoving or angular, as specified by '`Dist`'. For cosdistAngArea this is always taken to be the diameter of either the projected or 3D object. The '`Size`' specified should be assuming the same cosmology as provided, so be careful with your '`H0`'! |
| `Dim` | Specifies whether the object being considered is 1D (a line) 2D (e.g. a face on galaxy) or 3D (e.g. dark matter halo). This makes a very small modification to the geometry used (tan of 1D/2D and sin for 3D), but is only noticeable for large structures at low redshifts. |
| `Dist` | Determines the distance type of the specified '`Size`', i.e. angular / physical distances (Ang) or with respect to comoving distances (Co). |
| `inunit` | The units of angular coordinate provided for '`ang`'. Allowed options are deg for degrees, amin for arc minutes, asec for arc seconds, and rad for radians. |
| `outunit` | For `cosdistAngSize` units of angular coordinate output. Allowed options are deg for degress (default), amin for arc minutes, asec for arc seconds, and rad for radians.<br>For `cosdistAngArea` units of angular area output. Allowed options are deg2 for square degrees (default), amin2 for square arc minutes, asec2 for square arc seconds and rad2 or sr for steradians. |

## Details

Functions are largely based on D. W. Hogg et al. 1999 and Wright et al. 2006.

Negative value of z> -1 are allowed, which produces future predictions based on present day cosmology.

cosdistAngDist12 is only available for OmegaK>=0.

## Value

`cosdist` Returns a data.frame (even if only 1 redshift if requested) with the following columns:

| | |
|---|---|
| z | Requested redshift |
| a | Universe expansion factor, as given by a=1/(1+z) |
| CoDist | Line-of-sight (i.e. radial) comoving distance in units of Mpc |
| LumDist | Luminosity distance in units of Mpc |
| AngDist | Angular diameter distance in units of Mpc |
| CoDistTran | Transverse comoving distance in units of Mpc |
| DistMod | The distance modulus used where AbsMag = ApMag - DistMod, and DistMod = 5log10(LumDist)+25 in units |
| AngScale | Physical projected scale of an object at z in units of kpc/arcsec |
| CoVol | Comoving volume of Universe within z in units of Gpc^3 |

If age=TRUE is set then additional age-related information is calculated for each z as extra columns:

| | |
|---|---|
| HubTime | Approximate Hubble age of the Universe in units of Gyrs |
| UniAgeNow | Age of the Universe now in units of Gyrs |

| | |
|---|---|
| UniAgeAtz | Age of the Universe at the specified redshift (z) in units of Gyrs |
| TravelTime | Light travel time from the specified redshift (AKA look back time) in units of Gyrs |

If error=TRUE is set then the relative error for distance/age values is calculated for each z as an extra column:

| | |
|---|---|
| RelError | Relative error of the distance/age integrals (this is the main source of error in the calculations) |

The outputs of the standalone functions are:

| | |
|---|---|
| `cosdistz` | Returns the input redshift (only included for clarity). |
| `cosdistzeff` | Returns the apparent redshift that the object at zref will observe the object at zem for the Universe age that zref is observed to have now. This is given by $(1 + zem)/(1 + zref)$. |
| `cosdista` | Returns the Universe expansion factor, as given by $a = 1/(1 + z)$. |
| `cosdistCoDist` | Returns the line-of-sight (i.e. radial) comoving distance in units of Mpc. For a flat Universe (OmegaK=0) this is exactly the samething as the transverse comoving distance, and by extension it is also the proper motion distance. |
| `cosdistLumDist` | Returns the luminosity distance in units of Mpc. |
| `cosdistAngDist` | Returns the angular diameter distance in units of Mpc. |
| `cosdistAngDist12` | Returns the radial angular diameter distance separation in units of Mpc between objects at '`z1`' and '`z2`' that have small angular separations on sky. |
| `cosdistCoDistTran` | Returns the transverse comoving distance in units of Mpc. This is equivilant to the proper motion distance for all values of Universe curvature (OmegaK !=0), and is the same thing as the line-of-sight comoving distance for a flat Universe (OmegaK=0). |
| `cosdistCoDist12ang` | Returns the total comoving distance in units of Mpc between objects at '`z1`' and '`z2`' with a separation '`ang`'. This works for curved cosmologies (i.e. OmegaK!=0) and for large radial and tangential separations. For small separations at a certain value of z for both objects the result is very similar to cosdistCoDistTran(z)*sin(ang*pi/180). This function was mostly extracted from Eqn 3.19 in Peacock (1999). |
| `cosdistLumDist12ang` | Returns the total luminosity distance in units of Mpc between objects at '`z1`' and '`z2`' with a separation '`ang`'. This is equal to cosdistCoDist12ang*(1+zeff), where zeff is the apparent redshift that the object at z1 will observe the object at z2 for the Universe age that z1 is observed to have now. See `cosdistCoDist12ang` for details. |
| `cosdistAngDist12ang` | Returns the total angular diameter distance in units of Mpc between objects at '`z1`' and '`z2`' with a separation '`ang`'. This is equal to cosdistCoDist12ang/(1+zeff), where zeff is the apparent redshift that the object at z1 will observe the object at z2 for the Universe age that z1 is observed to have now. See `cosdistCoDist12ang` for details. |

`cosdistzem12ang`
: Returns the apparent redshift that the object at 'z1' would observe the object at 'z2' to be for our current Universe age. See `cosdistCoDist12ang` for details.

`cosdistzeff12ang`
: Returns the apparent redshift that the object at 'z1' would observe the object at 'z2' to be for the Universe age that z1 is observed to have now. See `cosdistCoDist12ang` for details.

`cosdistDistMod`
: Returns the distance modulus used where AbsMag = ApMag - DistMod, and DistMod = 5log10(LumDist)+25 in units of mag.

`cosdistAngScale`
: Returns the physical projected scale of an object at z in units of kpc/arcsec.

`cosdistAngSize`
: Returns the angular size (length or diameter) of an object (by default in degrees).

`cosdistAngArea`
: Returns the angular area of an object (by default degrees^2), taking the specified Size to be the diameter.

`cosdistCoVol`
: Returns the comoving volume of Universe within z in units of Gpc^3.

`cosdistHubTime`
: Returns the approximate Hubble age of the Universe in units of Gyrs.

`cosdistUniAgeNow`
: Returns the age of the Universe now in units of Gyrs.

`cosdistUniAgeAtz`
: Returns the age of the Universe at the specified redshift (z) in units of Gyrs.

`cosdistTravelTime`
: Returns the light travel time from the specified redshift (AKA look back time) in units of Gyrs.

`cosdistRelError`
: Returns the relative error of the distance/age integrals (this is the main source of error in the calculations).

`cosdistCrit`
: Returns the critical surface mass density, SigmaC (see also `cosNFW`).

## Author(s)

Aaron Robotham

## References

Based on the equations in:

Davis T.M. & Lineweaver, Charles H., 2004, PASA, 21, 97

Hogg D.W., 1999, arXiv, 9905116

Liske J., 2000, MNRAS, 319, 557L

Peacock J.A., 1999, Cosmological Physics, Cambridge University Press

Wright E.L., 2006, PASP, 118, 1711

## See Also

`cosvol`, `cosmap`, `cosgrow`, `cosref`, `cosNFW`

## Examples

```
## Not run:
cosdist(0.3,70,age=TRUE)
cosdist(0.3,70,age=TRUE,ref='Planck')
cosdistz(0.3)
cosdista(0.3)
cosdistCoDist(0.3,70)
cosdistLumDist(0.3,70)
cosdistAngDist(0.3,70)
cosdistAngDist12(0.3,0.5,70)
cosdistCoDistTran(0.3,70)
cosdistCoDist12ang(0,2,10)
cosdistDistMod(0.3,70)
cosdistAngScale(0.3,70)
cosdistAngSize(0.3,1,70)
cosdistCoVol(0.3,70)
cosdistHubTime(70)
cosdistUniAgeNow(0.3,70)
cosdistUniAgeAtz(0.3,70)
cosdistTravelTime(0.3,70)
cosdistRelError(0.3)
cosdistCrit(0.3,0.5,70)
cosdistzeff(1,2)
cosdistzem12ang(1,2)
cosdistzeff12ang(1,2)

#A check of the comoving separation between objects function:

cosdistCoDistTran(2,OmegaM = 0.3, OmegaL=1)*sin(pi/180)
cosdistCoDist12ang(2,2,ang=1,OmegaM=0.3,OmegaL=1)

#Very close, however cosdistCoDist12ang lets us go further:

cosdistCoDist12ang(1,2,ang=10,OmegaM=0.3,OmegaL=1)
cosdistCoDist12ang(2,2,ang=180,OmegaM=0.3,OmegaL=1)

#The second number should be be the same as:

cosdistCoDist(2,OmegaM=0.3,OmegaL=1)*2

#Example 1 by John Peacock for EDS Universe (answer should be nearly 3):

cosdistzem12ang(3,4,56.4,H0=100,OmegaM=1,OmegaL=0)

#Example 2 by John Peacock for EDS Universe (answer should be nearly 2995 Mpc/h):

cosdistCoDist12ang(3,4,56.4,H0=100,OmegaM=1,OmegaL=0)

#Example 3 by John Peacock for Milne Universe (answer should be nearly 5294 Mpc/h):

cosdistCoDist12ang(3,4,56,H0=100,OmegaM=0,OmegaL=0)
```

```
#Example 4 by John Peacock for Milne Universe (answer should be nearly 4.846):

cosdistzeff12ang(3,4,56,H0=100,OmegaM=0,OmegaL=0)

#Example 5 by John Peacock for Milne Universe (answer should be nearly 364 Mpc/h):

cosdistAngDist12ang(3,4,56,H0=100,OmegaM=0,OmegaL=0)

#Nice plot of distance estimates:

redshifts=seq(0,3,by=0.01)
plot(redshifts, cosdistCoDist(redshifts, ref='planck'), type='l', col='darkgreen',
xlab='Redshift / z', ylab='Distance / Mpc')
lines(redshifts, cosdistLumDist(redshifts, ref='planck'), col='red')
lines(redshifts, cosdistAngDist(redshifts, ref='planck'), col='blue')
legend('topleft', legend=c('Comoving Distance', 'Luminosity Distance', 'Angular Diameter Distance'),
col=c('darkgreen', 'red', 'blue'),lty=1)

plot(redshifts, cosdistTravelTime(redshifts, ref='planck'), type='l',
xlab='Redshift / z', ylab='Light travel time / Yrs')

#Actual time example (Figure 1 of Davis & Lineweaver 2004)
zseq=10^seq(-2,6,len=1e3)-1
dists=cosdistCoDist(zseq, ref='737')*0.00326
times=cosdistTravelTime(zseq, ref='737')
plot(dists, times, type='l', xlab='Comoving Distance / Glyr',
ylab='Time / Gyr')
abline(v=0, h=0, lty=1)
abline(h=c(min(times), max(times)), lty=2)
abline(v=c(min(dists), max(dists)), lty=2)

#Conformal time example (Figure 1 of Davis & Lineweaver 2004):
#Mpc to Glyr conversion is 0.00326

zseq=10^seq(-2,6,len=1e3)-1
dists=cosdistCoDist(zseq, ref='737')*0.00326
plot(dists, dists, type='l',
xlab='Comoving Distance / Glyr', ylab='Conformal Time / Gyr')
abline(v=0, h=0, lty=1)
abline(h=c(min(dists), max(dists)), lty=2)
abline(v=c(min(dists), max(dists)), lty=2)

## End(Not run)
```

---

cosgrow *Cosmological growth and evolution calculations*

---

## Description

These functions allow various properties of the expansion of the Universe to be calculated: e.g. OmegaM/OmegaL/OmegaR/OmegaK for ay redshift, growth rate and growth factor, sigma8, and

RhoCrit and RhoMean. They use curvature correctly, calculated internally using the relation OmegaM+OmegaL+OmegaR+O
but by default they assume a flat Universe where only OmegaM needs to be specified and OmegaR=0
(so no radiation pressure at any epoch).

## Usage

```
cosgrow(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0 = -1, wprime = 0,
 Sigma8=0.8, fSigma8=FALSE, Dist='Co',
 Mass='Msun', ref)
 cosgrowz(z = 1)
 cosgrowa(z = 1)
cosgrowH(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0,
 ref)
 cosgrowCoVel(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1,
 wprime=0, ref)
 cosgrowPecVel(z=1, zob=1)
cosgrowOmegaM(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0, ref)
cosgrowOmegaL(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0, ref)
cosgrowOmegaR(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0, ref)
cosgrowOmegaK(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0, ref)
cosgrowDecelq(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0, ref)
 cosgrowEoSwDE(z=1, w0=-1, wprime=0)
 cosgrowRhoDE(z=1,w0=-1, wprime=0, rhoDE=1)
cosgrowFactor(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0, ref)
cosgrowRate(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0,
 Sigma8=0.8, fSigma8=FALSE, ref)
cosgrowSigma8(z=1, OmegaM=0.3,OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0,
 Sigma8=0.8, ref)
cosgrowFactorApprox(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0,
 ref)
cosgrowRateApprox(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0,
 Sigma8=0.8, fSigma8=FALSE, ref)
cosgrowSigma8Approx(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1, wprime=0,
 Sigma8=0.8, ref)
cosgrowRhoCrit(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1,
 wprime=0, Dist='Co', Mass='Msun', ref)
cosgrowRhoMean(z=1, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, w0=-1,
 wprime=0, Dist='Co', Mass='Msun', ref)
 cosgrowDeltaVir(z=1, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0, ref)
```

## Arguments

| | |
|---|---|
| z | Cosmological redshift, where z must be > -1 (can be a vector). |
| zob | Observed redshift, where z must be > -1 (can be a vector). Essentially a combination of the cosmological redshift 'z' and the peculiar velocity of the objects with respect to this. |
| H0 | Hubble constant as defined at z=0 (default is H0=100 (km/s)/Mpc). |
| OmegaM | Omega Matter as defined at z=0 (default is 0.3). |

| | |
|---|---|
| OmegaL | Omega Lambda as defined at z=0 (default is for a flat Universe with OmegaL = 1-OmegaM-OmegaR = 0.7). |
| OmegaR | Omega Radiation as defined at z=0 (default is 0, but ~OmegaM/3400 is typical). |
| w0 | The value of dark energy equation of state as defined at z=0. |
| wprime | The evolution term that governs how the dark energy equation of state evolves with redshift. |
| rhoDE | The z=0 reference energy density for dark energy. |
| Sigma8 | The value of Sigma8 to use if fsigma8=TRUE (by default this is a reasonable 0.8 for simplicity). |
| fSigma8 | Logical to express whether the growth rate of structure calculated by cosgrow, cosgrowRate or cosgrowRateApprox is given as f*Sigma8 (TRUE) or simply f (FALSE). This is useful for redshift space distortion comparisons (RSD), since RSD strictly measures f*Sigma8. |
| Dist | Determines the distance type, i.e. whether the Rho critical energy or mean mass densities are calculated with respect to angular / physical distances (Ang), with respect to comoving distances (Co) or with respect to physical metres (m). |
| Mass | Determines the mass type, i.e. whether Rho critical energy or mean mass densities are calculated with respect to solar masses (Msun) or with respect to kilograms (kg). |
| ref | The name of a reference cosmology to use, one of 137 / 737 / Planck / Planck13 / Planck15 / Planck18 / WMAP / WMAP9 / WMAP7 / WMAP5 / WMAP3 / WMAP1 / Millennium / GiggleZ. Planck = Planck18 and WMAP = WMAP9. The usage is case insensitive, so wmap9 is an allowed input. See cosref for details. This overrides any other settings for H0/ OmegaM and OmegaL. If ref=137 or ref=737 no specific Sigma8 is assumed, instead Sigma8 is set to whatever the input value is set to (by default this is 0.8). |

## Details

The above functions are heavily based on the equations in Hamilton A.J.S., 2001, MNRAS 322 419 and Lahav O., et al., 1991, MNRAS, 251, 136.

Negative value of z> -1 are allowed, which produces future predictions based on present day cosmology.

The approximation routines are generally accurate to sub 1 percent, and since they do not involve numerical integration they are substantially faster when computing large grids of numbers, i.e. they are recommended for plots, since the accuracy is sub the line width.

## Value

| | |
|---|---|
| cosgrow | Returns a data.frame (even if only 1 redshift if requested) with the following parameters evaluated at the respective redshift/s: |

| | |
|---|---|
| z | Requested redshift |
| a | Universe expansion factor, as given by a=1/(1+z) |
| H | Hubble expansion rate in units of (km/s)/Mpc |

| | |
|---|---|
| CoVel | Cosmological recession velocity in units of km/s |
| OmegaM | Omega Matter |
| OmegaL | Omega Lambda |
| OmegaR | Omega Radiation |
| OmegaK | Omega K(c)urvature |
| Decelq | Traditional deceleration parameter q |
| Factor | Exact growth factor (g, see cosgrowFactor below for details) |
| Rate | Exact growth rate (f or f*Sigma8, see cosgrowRate below for details) |
| Sigma8 | Power spectrum fluctuation amplitude on the scale 8 Mpc/z |
| RhoCrit | Critical energy density of the Universe at z, where $\rho_{crit} = (3.H(z)^2)/(8.\pi.G)$, in units of $M_\odot/Mpc^3$ |
| RhoMean | Mean mass density of the Universe at z, where $\rho_{mean} = \rho_{crit}.\Omega_M(z)$, in units of $M_\odot/Mpc^3$ |

The outputs of the standalone functions are:

| | |
|---|---|
| `cosgrowz` | Returns the input redshift (only included for clarity). |
| `cosgrowa` | Returns the Universe expansion factor, as given by a=1/(1+z). |
| `cosgrowH` | Returns the value of the Hubble expansion rate at z, in units of km/s/Mpc. |
| `cosgrowCoVel` | Returns the value of the cosmological recession velocity of the object in units of km/s. |
| `cosgrowPecVel` | Returns the value of the peculiar velocity of the object in units of km/s. |
| `cosgrowOmegaM` | Returns the value of Omega Matter at z. |
| `cosgrowOmegaL` | Returns the value of Omega Lambda at z. |
| `cosgrowOmegaR` | Returns the value of Omega Radiation at z. |
| `cosgrowOmegaK` | Returns the value of Omega K(c)urvature at z. |
| `cosgrowDecelq` | Returns the traditional deceleration parameter q, given by q=OmegaM/2+Omegar-OmegaL. |
| `cosgrowEoSwDE` | Returns w for the dark energy equation of state, where $P = w.\rho_{DE}.c^2$ and $w = w_0 + 2.w'.(1 - 1/(1+z))$, as described in Wright (2006). |
| `cosgrowRhoDE` | Returns the energy density for dark energy, given by $\rho_{DE}.e^{-6.w'.(1-1/(1+z))}/(1+z)^{-(3+3.w_0+6.w')}$, as described in Wright (2006).. |
| `cosgrowFactor` | Returns the exact value of the growth factor (typically referred to as 'g' in the astronomy literature), at z. This is defined such that it equals 1 at z=Inf and is less than 1 at lower z. |
| `cosgrowRate` | Returns either the true (typically referred to as 'f' in the astronomy literature) or RSD type (f*Sigma8) value of the growth rate of structure, at z. This is defined such that it equals 1 at z=Inf and is less than 1 at lower z. |
| `cosgrowSigma8` | Returns the power spectrum fluctuation amplitude on the scale 8 Mpc/z at z, and is unitless. |
| `cosgrowFactorApprox` | Returns the approximate value of the growth factor (typically referred to as 'g' in the astronomy literature), at z. This is defined such that it equals 1 at z=Inf and is less than 1 at lower z. |

`cosgrowRateApprox`
: Returns either the approximate true (typically referred to as 'f' in the astronomy literature) or approximate RSD type (f*Sigma8) value of the growth rate of structure, at z. This is defined such that it equals 1 at z=Inf and is less than 1 at lower z.

`cosgrowSigma8Approx`
: Returns the approximate power spectrum fluctuation amplitude on the scale 8 Mpc/z at z, and is unitless.

`cosgrowRhoCrit`
: Returns the critical energy density of the Universe at z, where $\rho_{crit} = (3.H(z)^2)/(8.\pi.G)$, in units of $M_\odot/Mpc^3$.

`cosgrowRhoMean`
: Returns the mean mass density of the Universe at z, where $\rho_{mean} = \rho_{crit}.\Omega_M(z)$, in units of $M_\odot/Mpc^3$.

`cosgrowDeltaVir`
: Returns the delta-critical virial radius overdensity criterion for a range of flat Universes with varying OmegaM. Taken from Eqn. 6 of Bryan & Norman (1998).

## Note

The difference between RhoCrit and RhoMean at z=0 is simply RhoMean=RhoCrit*OmegaM. Corrected for 1/(1+z)^3 RhoMean stays constant with redshift (as it should- to first order we do not gain or lose mass within a comoving volume).

The growth rate and growth factor does not make use of OmegaR in the cosgrow function, hence OmegaR cannot be provided in the individual functions. This is because correctly accounting for the effect of radiation pressure before the surface of last scattering (z~1100) on the growth rate of structure is highly complex, and beyond the scope of this package. In the case of cosgrow, even if OmegaR is specified it is, in effect, set to zero when making growth factor and rate calculations.

The evolution of the dark matter equation of state (w) is parameterised as described in Wright (2006).

It is important to remember that H is in physical units for both the numerator and denominator (i.e. 'proper' at a given redshift, so the units are km/s / pMpc). To ask the question "is the Universe accelerating?" is to really ask "is the expansion factor accelerating?". This requires the denominator to be in comoving units (so rescaling for proper distances today) and measuring the differential with time or redshift. You will only find an accelerating Universe when dividing H(z)/(1+z)! See the examples to see how we can find this location, and that it is consistent with the start of acceleration calculated from the decleration parameter (q) directly.

## Author(s)

Aaron Robotham

## References

Based on the equations in:

Bryan & Norman, 1998, ApJ, 495, 80

Davis T.M. & Lineweaver, Charles H., 2004, PASA, 21, 97

Davis T.M. & Scrimgeour M.I., 2014, MNRAS, 442, 1117

Hamilton A.J.S., 2001, MNRAS 322 419

Lahav O., et al., 1991, MNRAS, 251, 136

Peacock J.A., 1999, Cosmological Physics, Cambridge University Press

Wright E.L., 2006, PASP, 118, 1711

## See Also

cosvol, cosmap, cosdist, cosref, coshalo

## Examples

```
cosgrow(0.3)
cosgrow(0.3,ref='Planck')
cosgrowz(0.3)
cosgrowa(0.3)
cosgrowH(0.3)
cosgrowCoVel(0.3)
cosgrowPecVel(0.3,0.31)
cosgrowOmegaM(0.3)
cosgrowOmegaL(0.3)
cosgrowOmegaK(0.3)
sum(cosgrowOmegaM(0.3)+cosgrowOmegaL(0.3)+cosgrowOmegaK(0.3)) #Still 1.
cosgrowDecelq(0.3)
cosgrowEoSwDE(0.3)
cosgrowFactor(0.3)
cosgrowFactorApprox(0.3) #Approximation better than 1% for reasonable cosmologies.
cosgrowRate(0.3)
cosgrowRateApprox(0.3) #Approximation better than 1% for reasonable cosmologies.
cosgrowRhoCrit(0.3)
cosgrowRhoMean(0.3)
cosgrowRhoMean(0)-cosgrowRhoMean(2,Dist='Ang')/(1+2)^3 #Mass is conserved in co-vol
cosgrowRhoMean(0)-cosgrowRhoMean(10,Dist='Co') #Mass is conserved in co-vol

# Various recessional velocities (see Figure 2 of Davis & Lineweaver 2004):

plot(10^seq(-1,4,by=0.01), cosgrowCoVel(10^seq(-1,4,by=0.01), ref='planck')
/299792.458, type='l', log='x', xlab='z', ylab='Cosmological Recession Velocity / c')
lines(10^seq(-1,4,by=0.01), cosgrowPecVel(0,10^seq(-1,4,by=0.01))/299792.458, col='red')
lines(10^seq(-1,4,by=0.01), 10^seq(-1,4,by=0.01), col='blue')
abline(h=1,v=1.5,lty=2)
legend('topleft', legend=c('GR', 'SR', 'Approx (cz)', 'Superluminal'), lty=c(1,1,1,2),
col=c('black','red','blue','black'))

# Comparison of the various energy densities that make up the Universe for Planck 2013:

plot(cosdistUniAgeAtz(10^seq(-3,4.9,by=0.1), ref='Planck')*1e9,
cosgrowRhoCrit(10^seq(-3,4.9,by=0.1), ref='Planck', Dist='m', Mass='kg')*
cosgrowOmegaR(10^seq(-3,4.9,by=0.1), ref='Planck'), type='l',log='xy',
xlab='Years since Universe formed', ylab=expression('Energy Density'*(kg/m^3)))

lines(cosdistUniAgeAtz(10^seq(-3,4.9,by=0.1), ref='Planck')*1e9,
```

```
cosgrowRhoCrit(10^seq(-3,4.9,by=0.1), ref='Planck', Dist='m', Mass='kg')*
cosgrowOmegaM(10^seq(-3,4.9,by=0.1), ref='Planck'), col='red')

lines(cosdistUniAgeAtz(10^seq(-3,4.9,by=0.1), ref='Planck')*1e9,
cosgrowRhoCrit(10^seq(-3,4.9,by=0.1), ref='Planck', Dist='m', Mass='kg')*
cosgrowOmegaL(10^seq(-3,4.9,by=0.1), ref='Planck'), col='blue')

abline(v=cosdistUniAgeAtz(0.33,ref='Planck')*1e9,lty=2) # Matter = Vacuum
abline(v=cosdistUniAgeAtz(3391,ref='Planck')*1e9,lty=2) # Matter = Radiation

legend('topright', legend=c('Radiation Energy Density', 'Matter Energy Density',
'Vacuum Energy Density'), lty=1, col=c('black','red','blue'))

# Where's the acceleration?
plot(seq(0,2,len=1e3),cosgrowH(seq(0,2,len=1e3)),type='l',xlab='z',
ylab='H(z) / km/s / pMpc')
# There it is!
plot(seq(0,2,len=1e3),cosgrowH(seq(0,2,len=1e3))/(1+seq(0,2,len=1e3)),
type='l',xlab='z',ylab='H(z) / km/s / cMpc')
#When does it start accelerating?
accel.loc=which.min(abs(cosgrowDecelq(seq(0,2,len=1e3))))
abline(v=seq(0,2,len=1e3)[accel.loc],lty=2)
```

---

coshalo *Virial halo conversion functions*

---

## Description

All 6 Virial parameter conversion functions. Each can map precisely to the other as a one paramter function.

## Usage

```
coshaloMvirToSigma(Mvir=1e+12, z=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, Rho='crit', Dist='Co', DeltaVir=200, Munit=1, Lunit=1e6, Vunit=1e3, Dim=3, ref)
coshaloSigmaToMvir(Sigma=230, z=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, Rho='crit', Dist='Co', DeltaVir=200, Munit=1, Lunit=1e6, Vunit=1e3, Dim=3, ref)
coshaloMvirToRvir(Mvir=1e12, z=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, Rho='crit', Dist='Co', DeltaVir=200, Munit=1, Lunit=1e6, Vunit=1e3, Dim=3, ref)
coshaloRvirToMvir(Rvir=162.635, z=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, Rho='crit', Dist='Co', DeltaVir=200, Munit=1, Lunit=1e6, Vunit=1e3, Dim=3, ref)
coshaloSigmaToRvir(Sigma=230, z=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, Rho='crit', Dist='Co', DeltaVir=200, Munit=1, Lunit=1e6, Vunit=1e3, Dim=3, ref)
coshaloRvirToSigma(Rvir=162.635, z=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, Rho='crit', Dist='Co', DeltaVir=200, Munit=1, Lunit=1e6, Vunit=1e3, Dim=3, ref)
coshaloSigmaToTvir(Sigma=230, Vunit=1e3, Tunit='K', Type='halo', Dim=3)
```

## Arguments

| | |
|---|---|
| Mvir | Mass within virial radius in units of 'Munit'. |
| Sigma | Velocity dispersion (3D) within virial radius in units of 'Vunit'. For coshaloSigmaToTvir the Sigma input should be the virial Sigma which can be found by setting DeltaVir='get' in the the other coshalo functions. |
| Rvir | Virial radius (3D) in units of 'Lunit'. |
| z | Cosmological redshift, where z must be > -1 (can be a vector). |
| H0 | Hubble constant as defined at z=0 (default is H0=100 (km/s)/Mpc). |
| OmegaM | Omega Matter as defined at z=0 (default is 0.3). |
| OmegaL | Omega Lambda as defined at z=0 (default is for a flat Universe with OmegaL = 1-OmegaM = 0.7). |
| OmegaR | Omega Radiation as defined at z=0 (default is 0, but OmegaM/3400 is typical). |
| Rho | Set whether the critical energy density is used (crit) or the mean mass density (mean). |
| Dist | Determines the distance type, i.e. whether the Rho critical energy or mean mass densities are calculated with respect to angular / physical distances (Ang) or with respect to comoving distances (Co). In effect this means Rvir values are either angular / physical (Ang) or comoving (Co). It does not affect Mvir <-> Sigma conversions, but does affect Mvir <-> Rvir and Rvir <-> Sigma. |
| DeltaVir | Desired overdensity of the halo with respect to Rho. If set to 'get' it will estimate the required DeltaVir for a virial collapse using the cosgrowDeltaVir function. |
| Munit | Base mass unit in multiples of Msun. |
| Lunit | Base length unit in multiples of parsecs. |
| Vunit | Base velocity unit in multiples of m/s. |
| Type | Specify the 'halo' or 'gas' temperature to be computed. |
| Tunit | Specify the output temperature to be Kelvin ('K'), 'eV' or 'keV'. |
| Dim | The dimensional type for the halo (either the 2 or 3). 3 (default) means quantities are intrinsic 3D values. 2 means quantities are for projected systems (i.e. radius and velocity dispersion are compressed). From comparisons to simulations (so NFW, c~5 halos) Rvir[proj]=Rvir[3D]/1.37 and Sigma[proj]=Sigma[3D]/sqrt(3). The former has dependence on the halo profile (so is approximate), whereas the latter is a dimensionality argument that should hold for any virialised system. Note that for projected systems Sigma is measured along one dimension: the line-of-site. |
| ref | The name of a reference cosmology to use, one of 137 / 737 / Planck / Planck13 / Planck15 / Planck18 / WMAP / WMAP9 / WMAP7 / WMAP5 / WMAP3 / WMAP1 / Millennium / GiggleZ. Planck = Planck18 and WMAP = WMAP9. The usage is case insensitive, so wmap9 is an allowed input. See cosref for details. This overrides any other settings for H0/ OmegaM and OmegaL. |

## Details

These functions allow for various analytic conversions between the 3 major properties related to virial radius: the mass, velocity dispresion and size. The default properties calculate properties for 1e12 Msun halos and assume masses in Msun, velocities in km/s and distances in Kpc.

## Value

`coshaloMvirToSigma`
: Outputs approximate velocity dispersion (in units of Vunit) given mass (this is exactly the escape velocity at Rvir).

`coshaloSigmaToMvir`
: Outputs mass (in units of Munit) given velocity dispersion.

`coshaloMvirToRvir`
: Outputs radius (in units of Lunit) given mass.

`coshaloRvirToMvir`
: Outputs mass (in units of Munit) given radius.

`coshaloSigmaToRvir`
: Outputs radius (in units of Lunit) given velocity dispersion.

`coshaloRvirToSigma`
: Outputs approximate velocity dispersion (in units of Vunit) given radius (this is exactly the escape velocity at Rvir).

`coshaloSigmaToTvir`
: Output temperture (in units of Tunit) given velocity dispersion. Based on Eqns. 3/7/8/9 of Bryan & Norman (1998).

## Author(s)

Aaron Robotham, Chris Power

## References

coshaloSigmaToTvir based on the equations in:

Bryan & Norman, 1998, ApJ, 495, 80

## See Also

cosvol, cosmap, cosdist, cosgrow, cosNFW

## Examples

```
coshaloMvirToSigma(1e13) # Velocity in km/s
coshaloMvirToSigma(1e13, Vunit=1) # Velocity in m/s
coshaloSigmaToMvir(coshaloMvirToSigma(1e13, Vunit=1),Vunit=1)
coshaloMvirToRvir(1e13) #Radius in kpc
coshaloSigmaToRvir(coshaloMvirToSigma(1e13, Vunit=1),Vunit=1)

#Some sanity checks

rho_crit200=cosgrowRhoCrit(z=0)*200 #200 times rho critical at z=0
rho_mean200=cosgrowRhoMean(z=0)*200 #200 times rho mean at z=0
#For a 10^12 Msun/h halo, the radius in Mpc/h where the contained density equals rho_crit*200
rad_crit200=(1e12/rho_crit200*3/4/pi)^(1/3)
coshaloMvirToRvir(1e12,Lunit=1e6)-rad_crit200 # ~0 as expected
#For a 10^12 Msun/h halo, the radius in Mpc/h where the contained density equals rho_crit*200
rad_mean200=(1e12/rho_mean200*3/4/pi)^(1/3) # ~0 as expected
coshaloMvirToRvir(1e12,Lunit=1e6,Rho='mean')-rad_mean200
```

cosmap *Cosmological Mapping Functions*

## Description

Functions for mapping from one arbitrary cosmological parameter to another. This includes the provision of a generic interpolation function and another exact value lookup.

## Usage

```
cosmapval(val=50, cosparam="CoVol", H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0,
w0=-1, wprime=0, Sigma8=0.8, fSigma8=FALSE, zrange=c(-0.99,100), res=100, iter=8,
out='cos', degen='lo', ref)
cosmapfunc(cosparamx="CoVol", cosparamy="z", H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR,
OmegaR=0, w0=-1, wprime=0, Sigma8=0.8, fSigma8=FALSE, zrange=c(0,20), step='z', res=100,
degen='lo', ref)
```

## Arguments

| | |
|---|---|
| val | The value/s to be mapped from parameter cosparamx to parameter cosparamy (this can be a vector or a single number). |
| cosparam | Cosmological parameter, must be one of: z, a, CoDist, LumDist, CoDistTran, DistMod, AngScale, CoVol, UniAgeAtz, TravelTime (see cosdist help for further description of these) H, CoVel, OmegaM, OmegaL, OmegaK, Factor, Rate, RhoCrit (see cosgrow help for further description of these). Note that AngDist and AngSize are not an option for cosparam due to degenerate redshift solutions causing problems with the approxfun mapping. |
| cosparamx | Cosmological parameter, must be one of: z, a, CoDist, LumDist, CoDistTran, DistMod, AngScale, CoVol, UniAgeAtz, TravelTime (see cosdist help for further description of these) H, CoVel, OmegaM, OmegaL, OmegaK, Factor, Rate, RhoCrit (see cosgrow help for further description of these). Note that AngDist and AngSize are not an option for cosparamx due to degenerate redshift solutions causing problems with the approxfun mapping. |
| cosparamy | Cosmological parameter, must be one of: z, a, CoDist, LumDist, AngDist, CoDistTran, DistMod, AngScale, CoVol, UniAgeAtz, TravelTime (see cosdist help for further description of these) H, CoVel, OmegaM, OmegaL, OmegaK, Factor, Rate, RhoCrit (see cosgrow help for further description of these). |
| H0 | Hubble constant as defined at z=0 (default is H0=100 (km/s)/Mpc). |
| OmegaM | Omega Matter as defined at z=0 (default is 0.3). |
| OmegaL | Omega Lambda as defined at z=0 (default is for a flat Universe with OmegaL = 1-OmegaM = 0.7). |
| OmegaR | Omega Radiation as defined at z=0 (default is 0, but OmegaM/3400 is typical). |
| w0 | The value of dark energy equation of state as defined at z=0. See cosgrow for more details. |

| | |
|---|---|
| wprime | The evolution term that governs how the dark energy equation of state evolves with redshift. See cosgrow for more details. |
| Sigma8 | The value of Sigma8 as defined at z=0 to use if fsigma8=TRUE (by default this is a reasonable 0.8 for simplicity). |
| fSigma8 | Logical to express whether the growth rate of structure calculated by cosgrow, cosgrowRate or cosgrowRateApprox is given as f*Sigma8 (TRUE) or simply f (FALSE). This is useful for redshift space distortion comparisons (RSD), since RSD strictly measures f*Sigma8. |
| zrange | Lower and upper z limits that the approxfun mapping is generated over (increase range if default is not sufficient, and decrease if it is wasteful, i.e. the possible redshift window is known to be quite narrow). |
| step | The type of stepping used. Allowed values are 'z' (uniform stepping in z), 'logz' (uniform stepping in log10(1+z) and expansion factor 'a' (uniform stepping in a=1/(1+z)). Default is z. For mappings using time (UniAgeNow, UniAgeAtz, TravelTime) or comoving quantities (CoDist, CoDistTran, CoVol) or distance modulus (DistMod) 'a' or 'logz' map the numeric range more uniformly. This is because a and log10(1+z) are approximately linear in light travel time (positive and negative correlation respectively), and typically they have better behaviour than stepping uniformly in z directly. |
| res | The resolution of steps. Larger numbers will be more accurate, but will be slower to compute. |
| iter | The number of iterations to make when calculating the exact location of a given cosmological parameter when using cosmapval. |
| out | Either out='cos', in which case the output is a data.frame containing the output of cosdist and cosgrow for the specified input vaue, or out='z', in which case the output is a vector of the corresponding redshift (z) values. |
| degen | In cases where solutions are degenerate (multiple y solutions for a single x), this specifies whether to calculate the lower y solution (degen='lo'), or the higher y solutions (degen='hi'). |
| ref | The name of a reference cosmology to use, one of 137 / 737 / Planck / Planck13 / Planck15 / Planck18 / WMAP / WMAP9 / WMAP7 / WMAP5 / WMAP3 / WMAP1 / Millennium / GiggleZ. Planck = Planck18 and WMAP = WMAP9. The usage is case insensitive, so wmap9 is an allowed input. See cosref for details. This overrides any other settings for H0/ OmegaM and OmegaL. If ref=137 or ref=737 no specific Sigma8 is assumed, instead Sigma8 is set to whatever the input value is set to (by default this is 0.8). |

## Details

The default zrange and res should be sufficient for most reasonable cosmologies if the approximate redshift location of the region to be mapped is entirely unknown.

Predictions into the future are possible if val is set to negative (distance and volume parameters) or below their present day value (age and growth parameters). However, many potential values are outside of the asymptotic limits, e.g. using the default 737 cosmology H is tending to 83.666, i.e. it will fail if you request H=83 but work if your resuest H=84.

The default res and iter for cosmapval is appropriate for most mappings with -0.99 < z < 100 using a fiducial 737 cosmology. If this proves insufficient (this should be obvious from error column) then increase both of these. Overall accuracy goes as res^iter.

## Value

If out='cos', cosmapval contains the concatenation of the cosdist (with age=TRUE and error=TRUE) and cosgrow functions for parameter 'cosparam' at value 'val'. The 'z' and 'a' columns are only included once (from the output of cosdist). See cosdist and cosgrow for information on the cosdist and cosgrow outputs. If out='z', then cosmapval merely returns the corresponding redshifts.

The cosmapval output (when out='cos') includes an additional final column named 'MapError' which gives the approximate relative error of the values returned compared to the desired lookup location. Smaller is obviously better, but at the cost of computational time.

cosmapfunc uses base R approxfun to map cosparamx onto cosparamy between zrange[1] and zrange[2] in uniform steps of expansion factor (a=1/(1+z)). cosmofunc returns the output function created by approxfun.

## Author(s)

Aaron Robotham

## References

Based on the equations in:

Hogg D.W., 1999, arXiv, 9905116

Wright E.L., 2006, PASP, 118, 1711

## See Also

cosdist, cosvol, cosgrow

## Examples

```
## Not run:
tempfunc=cosmapfunc('CoVol', 'UniAgeAtz')
tempfunc(50)

cosmapval(50:60, 'CoVol')

#A future prediction:

cosmapval(59, 'H', H0=70)

## End(Not run)
```

Cosmology Reference Sets

*Cosmology parameter data included in celestial package.*

## Description

cosref: Cosmology H0 / OmegaM / OmegaL / OmegaR (via OmegaM/zeq) and Sigma8 parameters taken from Planck (13/15/18), WMAP (1/3/5/9), Millennium Simulation and GiggleZ. Not all of these exist for each source, so NA values are used in these cases.

For Planck we use the second column of the main cosmology table, which does no use external data.

## Usage

```
data(cosref)
```

## Details

The included data is a table of the following cosmological parameters:

| Ref | H0 | OmegaM | OmegaL | OmegaR | Sigma8 |
|---|---|---|---|---|---|
| 737 | 70.0 | 0.300 | 0.700 | NA | NA |
| 137 | 100.0 | 0.300 | 0.700 | NA | NA |
| Planck | 68.4 | 0.301 | 0.699 | 8.985075e-05 | 0.793 |
| Planck18 | 68.4 | 0.301 | 0.699 | 8.985075e-05 | 0.793 |
| Planck15 | 67.8 | 0.308 | 0.692 | 9.150327e-05 | 0.815 |
| Planck13 | 67.3 | 0.315 | 0.685 | 9.289295e-05 | 0.829 |
| WMAP | 69.7 | 0.288 | 0.712 | 8.780488e-05 | 0.817 |
| WMAP9 | 69.7 | 0.288 | 0.712 | 8.780488e-05 | 0.817 |
| WMAP7 | 70.4 | 0.275 | 0.725 | 8.569648e-05 | 0.816 |
| WMAP5 | 70.5 | 0.274 | 0.726 | 8.45679e-05 | 0.812 |
| WMAP3 | 70.4 | 0.268 | 0.732 | NA | 0.776 |
| WMAP1 | 72.0 | 0.290 | 0.710 | NA | 0.900 |
| Millennium | 73.0 | 0.250 | 0.750 | NA | 0.900 |
| GiggleZ | 70.5 | 0.273 | 0.727 | NA | 0.812 |

## Author(s)

Aaron Robotham

## References

| Name | Full Reference | arXiv Refence |
|---|---|---|
| 737 | Simplified concordance cosomology | NA |
| 137 | Simplified concordance cosomology | NA |

| | | |
|---|---|---|
| Planck 18 | Planck Collaboration, 2018, arXiv, 1807.06209 | arxiv:1807.06209 |
| Planck 15 | Planck Collaboration, 2015, A&A, 594, 13 | arxiv:1502.01589 |
| Planck 13 | Planck Collaboration, 2014, A&A, 571, 16 | arXiv:1303.5076v3 |
| WMAP9 | Hinshaw G., et al., 2013, ApJS, 208, 19 | arXiv:1212.5226v3 |
| WMAP7 | Komatsu E., et al., 2010, ApJS, 192, 18 | arXiv:1001.4538v3 |
| WMAP5 | Komatsu E., et al., 2009, ApJS, 180, 306 | arXiv:0803.0547v2 |
| WMAP3 | Spergel D. N., et al., 2007, ApJS, 170, 377 | arXiv:astro-ph/0603449v2 |
| WMAP1 | Spergel D. N., et al., 2003, ApJS, 148, 175 | arXiv:astro-ph/0302209v3 |
| Millennium | Springel V., et al., 2005, Nature, 435, 629 | arXiv:astro-ph/0504097v2 |
| GiggleZ | Poole G. B., et al., 2015, MNRAS, 449, 1454 | arXiv:1407.0390v1 |

## See Also

cosvol, cosmap, cosdist, cosgrow

## Examples

```
data(cosref)
cosref[cosref[,'Ref']=='Planck',]
```

---

cosNFW *Navarro Frenk and White profile*

---

## Description

Density and total mass values for Navaro Frenk and White (NFW) profiles

## Usage

```
cosNFW(Rad=0, Rho0=2.412e15, Rs=0.03253)
cosNFWmass_c(Rho0=2.412e15, Rs=0.03253, c=5, Munit = 1, Lunit = 1e+06)
cosNFWmass_Rmax(Rho0=2.412e15, Rs=0.03253, Rmax=0.16265, Munit = 1, Lunit = 1e+06)
cosNFWvcirc(Rad = 0.16264, Mvir = 1e+12, c = 5, f = Inf, z = 0, H0 = 100, OmegaM = 0.3,
OmegaL = 1 - OmegaM - OmegaR, OmegaR = 0, Rho = "crit", Dist = "Co", DeltaVir = 200,
Munit = 1, Lunit = 1e+06, Vunit = 1000, ref)
cosNFWvesc(Rad = 0.16264, Mvir = 1e+12, c = 5, f = Inf, z = 0, H0 = 100, OmegaM = 0.3,
OmegaL = 1 - OmegaM - OmegaR, OmegaR = 0, Rho = "crit", Dist = "Co", DeltaVir = 200,
Munit = 1, Lunit = 1e+06, Vunit = 1000, ref)
cosNFWsigma(Rad=0.03253, Rs=0.03253, c=5, z = 0, H0 = 100, OmegaM = 0.3,
OmegaL = 1-OmegaM-OmegaR, OmegaR=0, Rho = "crit", DeltaVir = 200, Munit = 1,
Lunit = 1e+06, Vunit = 1000, ref)
cosNFWsigma_mean(Rad=0.03253, Rs=0.03253, c=5, z = 0, H0 = 100, OmegaM = 0.3,
OmegaL = 1-OmegaM-OmegaR, OmegaR=0, Rho = "crit", DeltaVir = 200, Munit = 1,
Lunit = 1e+06, Vunit = 1000, ref)
```

```
cosNFWgamma(Rad=0.03253, Rs=0.03253, c=5, SigmaC=1, z = 0, H0 = 100,
OmegaM = 0.3, OmegaL = 1-OmegaM-OmegaR, OmegaR=0, Rho = "crit", DeltaVir = 200,
Munit = 1, Lunit = 1e+06, Vunit = 1000, ref)
cosNFWduffym2c(M=2e12, z = 0, H0 = 100, OmegaM = 0.3, OmegaL = 1-OmegaM-OmegaR,
OmegaR=0, Rho = "crit", A=6.71, B=-0.091, C=-0.44, Munit = 1, ref)
```

## Arguments

| | |
|---|---|
| Mvir | Mass within virial radius in units of 'Munit'. |
| Rad | Radius at which to calculate output in units of 'Lunit'. Either this is a 3D radius (cosNFW) or a projected 2D radius (cosNFWsigma/cosNFWsigma_mean). |
| Rho0 | The normalising factor. |
| Rs | The NFW profile scale radius, where Rs=Rmax/c, in units of 'Munit'. |
| c | The NFW profile concentration parameter, where c=Rmax/Rs. |
| f | The NFW profile truncation radius in units of 'Rmax'. |
| Rmax | The NFW profile Rmax parameter, where Rmax=Rs*c, in units of 'Lunit'. |
| SigmaC | The critical surface mass density (when SigmaC=1 we compute the excess surface density / ESD). See cosdistCrit for general computation given source and lens redshifts. |
| M | The halo mass required for computing the Duffy (2008) mass to concentration conversion in units of 'Munit'. Here the halo mass required for input is the 200 times overdense with respect to critical variation. |
| z | Cosmological redshift, where z must be > -1 (can be a vector). |
| H0 | Hubble constant as defined at z=0 (default is H0=100 (km/s)/Mpc). |
| OmegaM | Omega Matter as defined at z=0 (default is 0.3). |
| OmegaL | Omega Lambda as defined at z=0 (default is for a flat Universe with OmegaL = 1-OmegaM = 0.7). |
| OmegaR | Omega Radiation as defined at z=0 (default is 0, but OmegaM/3400 is typical). |
| Rho | Set whether the critical energy density is used (crit) or the mean mass density (mean). |
| Dist | Determines the distance type, i.e. whether the Rho critical energy or mean mass densities are calculated with respect to angular / physical distances (Ang) or with respect to comoving distances (Co). In effect this means Rvir values are either angular / physical (Ang) or comoving (Co). It does not affect Mvir <-> Sigma conversions, but does affect Mvir <-> Rvir and Rvir <-> Sigma. |
| DeltaVir | Desired overdensity of the halo with respect to Rho. |
| Munit | Base mass unit in multiples of Msun. |
| Lunit | Base length unit in multiples of parsecs. |
| Vunit | Base velocity unit in multiples of m/s. |
| A | Parameter used for Duffy mass to concentration relation. |
| B | Parameter used for Duffy mass to concentration relation. |
| C | Parameter used for Duffy mass to concentration relation. |

`ref` The name of a reference cosmology to use, one of 137 / 737 / Planck / Planck13 / Planck15 / Planck18 / WMAP / WMAP9 / WMAP7 / WMAP5 / WMAP3 / WMAP1 / Millennium / GiggleZ. Planck = Planck18 and WMAP = WMAP9. The usage is case insensitive, so wmap9 is an allowed input. See `cosref` for details. This overrides any other settings for H0/ OmegaM and OmegaL.

## Details

These functions calculate various aspects of the NFW profile.

## Value

`cosNFW` Returns the instantaneous NFW profile density.

`cosNFWmass_c` Returns the total mass given Rs and c in Msun/h.

`cosNFWmass_Rmax` Returns the total mass given Rs and Rmax in Msun/h.

`cosNFWvcirc` Returns the circular Keplarian orbit velocity for a given radius assuming an NFW halo potential.

`cosNFWvesc` Returns the minimum escape (or unbinding) velocity for a given radius assuming an NFW halo potential.

`cosNFWsigma` Returns the line-of-sight surface mass density at Rad (Eqn. 11 of Wright & Brainerd, 2000).

`cosNFWsigma_mean` Returns the means surface mass density within Rad (Eqn. 13 of Wright & Brainerd, 2000).

`cosNFWgamma` Returns the radial dependence of the weak lensing shear (Eqn. 12 of Wright & Brainerd, 2000).

`cosNFWduffym2c` Returns the Duffy et al (2008) predicted concentration for a given halo mass.

## Author(s)

Aaron Robotham

## References

Duffy A.R., et al., 2008, MNRAS, 390L

Navarro J.F., Frenk C.S., White Simon D.M., 1996, ApJ, 462

Wright C.O. & Brainerd T.G., 2000, ApJ, 534

## See Also

`cosvol`, `cosmap`, `cosdist`, `cosgrow`, `coshalo`

## Examples

```
#What difference do we see if we use the rad_mean200 radius rather than rad_crit200

rad_crit200=coshaloMvirToRvir(1e12,Lunit=1e6)
rad_mean200=coshaloMvirToRvir(1e12,Lunit=1e6,Rho='mean')
cosNFWmass_Rmax(Rmax=rad_crit200) #By construction we should get ~10^12 Msun/h
cosNFWmass_Rmax(Rmax=rad_mean200) #For the same profile this is a factor 1.31 larger

#Shear checks:

plot(10^seq(-2,2,by=0.1), cosNFWgamma(10^seq(-2,2,by=0.1),Rs=0.2,c=10), type='l',
log='xy', xlab='R/Rs', ylab='ESD')
legend('topright', legend=c('Rs=0.2','c=10'))

#How do critical, mean 200 and 500 masses evolve with redshift? Let's see:

zseq=10^seq(-2, 1, by=0.1)
con=seq(2, 20, by=0.01)
concol=rainbow(length(con), start=0, end=5/6)
rad_crit200=coshaloMvirToRvir(1, z=zseq, Rho='crit', DeltaVir=200, ref='Planck15')
rad_crit500=coshaloMvirToRvir(1, z=zseq, Rho='crit', DeltaVir=500, ref='Planck15')
rad_mean200=coshaloMvirToRvir(1, z=zseq, Rho='mean', DeltaVir=200, ref='Planck15')
rad_mean500=coshaloMvirToRvir(1, z=zseq, Rho='mean', DeltaVir=500, ref='Planck15')
rad_vir=coshaloMvirToRvir(1, z=zseq, Rho='crit', DeltaVir='get', ref='Planck15')

plot(1, 1, type='n', xlim=c(0.01,10), ylim=c(0.8,1.55), xlab='Redshift',
ylab='M200c / M500c', log='x')
for(i in 1:length(con)){
lines(zseq, cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_crit200)/
cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_crit500), col=concol[i])
}

plot(1, 1, type='n', xlim=c(0.01,10), ylim=c(0.8,1.55), xlab='Redshift',
ylab='M200m / M500m', log='x')
for(i in 1:length(con)){
lines(zseq, cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_mean200)/
cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_mean500), col=concol[i])
}

plot(1, 1, type='n', xlim=c(0.01,10), ylim=c(0.8,1.55), xlab='Redshift',
ylab='M200m / M200c',log='x')
for(i in 1:length(con)){
lines(zseq, cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_mean200)/
cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_crit200), col=concol[i])
}

plot(1, 1, type='n', xlim=c(0.01,10), ylim=c(0.8,1.55), xlab='Redshift',
ylab='M500m / M500c', log='x')
for(i in 1:length(con)){
lines(zseq, cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_mean500)/
cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_crit500), col=concol[i])
}
```

```
plot(1, 1, type='n', xlim=c(0.01,10), ylim=c(0.8,1.55), xlab='Redshift',
ylab='Mvir / M200c',log='x')
for(i in 1:length(con)){
lines(zseq, cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_vir)/
cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_crit200), col=concol[i])
}

plot(1, 1, type='n', xlim=c(0.01,10), ylim=c(0.8,1.55), xlab='Redshift',
ylab='Mvir / M200m',log='x')
for(i in 1:length(con)){
lines(zseq, cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_vir)/
cosNFWmass_Rmax(Rho0=1, Rs=rad_crit200[1]/con[i], Rmax=rad_mean200), col=concol[i])
}

plot(zseq, rad_crit200/rad_crit500, type='l', xlim=c(0.01,10), ylim=c(0.8,1.55),
xlab='Redshift', ylab='R200 / R500', log='x')

plot(zseq, rad_mean200/rad_crit200, type='l', xlim=c(0.01,10), ylim=c(0.8,1.55),
xlab='Redshift', ylab='Rm / Rc', log='x')

plot(zseq, rad_vir/rad_crit200, type='l', xlim=c(0.01,10), ylim=c(0.8,1.55),
xlab='Redshift', ylab='Rvir / R200c', log='x')

plot(zseq, rad_vir/rad_mean200, type='l', xlim=c(0.01,10), ylim=c(0.8,1.55),
xlab='Redshift', ylab='Rvir / R200c', log='x')

#R200m and R200c go either side of Rvir, so by cosmic conspiracy the mean is nearly flat:

plot(zseq, 2*rad_vir/(rad_mean200+rad_crit200), type='l', xlim=c(0.01,10),
ylim=c(0.8,1.55), xlab='Redshift', ylab='2Rvir / (R200c+R200m)', log='x')

#To check Vcirc and Vesc for a 10^12 Msun halo:

plot(0:400, cosNFWvcirc(0:400,f=1,Lunit=1e3), type='l', lty=1, xlab='R / kpc',
ylab='V / km/s', ylim=c(0,500))
lines(0:400, cosNFWvesc(0:400,f=1,Lunit=1e3), lty=2)
legend('topright', legend=c('Vel-Circ','Vel-Escape'), lty=c(1,2))
abline(v=coshaloMvirToRvir(Lunit=1e3), lty=3)
```

---

cosorb *Orbital functions*

---

## Description

A variety of obital analysis functions. These are useful for setting up initial conditions for merging systems etc.

## Usage

```
cosorbVisViva(mass=1e12, Rad=162.635, SemiMajRad=162.635, Munit=1, Lunit=1e3, Vunit=1)
cosorbFreeFall(M1=1e12, M2=1, Rad=162.635, Munit=1, Lunit=1e3, Vunit=1, Tunit=1e9)
cosorbRocheRad(M1=1e12, M2=1e10, Size=35.03865, Rfac=2.44)
cosorbRocheSize(M1=1e12, M2=1e10, Rad=396.8294, Rfac=2.44)

potential_part(part, eval = NULL, mass = 1, soft = 1, Munit = 1, Lunit = 1000,
  Vunit = 1)
kinetic_part(vel, mass=NULL, Vunit = 1)
```

## Arguments

| | |
|---|---|
| mass | Mass in units of 'Munit'. |
| M1 | Mass of primary body in units of 'Munit'. |
| M2 | Mass of secondary body in units of 'Munit'. |
| Rad | Separation between bodies in units of 'Lunit' (for cosorbRocheSize this is in arbitrary units). |
| part | Cartesian particle positions [x,y,z] in units of 'Lunit'. |
| eval | Cartesian evalution positions [x,y,z] in units of 'Lunit'. If this is not provided then the potentials are evaluated at the positions of 'part'. |
| vel | Cartesian particle velocities [vx,vy,vz] in units of 'Vunit'. |
| soft | Numeric softening in units of 'Lunit'. |
| SemiMajRad | The semi major radius of the orbit (a > 0 for ellipses, a = Rad for circles, 1/a = 0 for parabolas, and a < 0 for hyperbolas). |
| Size | The size radius of the secondary object. Inside of this radius the object is bound to the secondary, outside of this radius the object is stripped by the primary. |
| Rfac | The Roche factor. Approximately taken to be 2.44, but in reality it varies depending on the shape of the potentials etc. |
| Munit | Base mass unit in multiples of Msun. |
| Lunit | Base length unit in multiples of parsecs. |
| Vunit | Base velocity unit in multiples of km/s. |
| Tunit | Base time unit in multiples of years. |

## Details

These functions allow for various analytic conversions between the 3 major properties related to virial radius: the mass, velocity dispresion and size. The default properties calculate properties for 1e12 Msun halos and assume masses in Msun, velocities in km/s and distances in Kpc.

## Value

cosorbVisViva function gives the required velocity in units of Vunit to create the specified orbit.

cosorbFreeFall function gives the free fall time to static initial velocity separated bodies.

cosorbRocheRad function gives the orbital radius at which the secondary will become stripped within a specified bound radius.

cosorbRocheSize function gives the limiting bound radius of the secondary for a given system.

potential_part functions gives the total gravitational potential either at the positions of particles provided ('part') or at the requested evaluation positions ('eval').

kinetic_part functions gives the total kinetic energy of particles provided ('vel')

## Author(s)

Aaron Robotham, Chris Power

## See Also

cosvol, cosmap, cosdist, cosgrow

## Examples

```
cosorbVisViva(M=1e15, Rad=1, Lunit=1e6)
cosorbFreeFall(M1=1e15, M2=1, Rad=1, Lunit=1e6)
cosorbRocheRad(M1=1e12, M2=1e12, Size=162.635, Rfac=2.44)
cosorbRocheSize(M1=1e12, M2=1e12, Rad=396.8294, Rfac=2.44)
```

---

# cosvar *Driver & Robotham (2010) cosmic variance calculator*

---

## Description

The main cosmic variance calculator function taken from Driver & Robotham (2010). cosvarcar is an interface to the Cartesian coordinate version, whilst cosvarsph is a utility interface to give approximate cosmic variance for astronomy survey regions (usually defined by RA, Dec and redshift limits).

## Usage

```
cosvarcar(aside = 50, bside = 50, cside = 50, regions = 1)
cosvarsph(long = c(129, 141), lat = c(-2, 3), zmax = 1, zmin = 0, regions = 1,
inunit='deg', sep=":")
cosvararea(area=60, zmax=1, zmin=0, regions=1, inunit='deg2')
```

## Arguments

| | |
|---|---|
| aside | The aside (shortest projected side) of the Cartesian box, must be defined using 737 cosmology. |
| bside | The bside (longest projects side) of the Cartesian box, must be defined using 737 cosmology. |
| cside | The cside (radial side) of the Cartesian box, must be defined using 737 cosmology. |
| regions | How many well separated regions of this size will there be? The geometry provided is just for a single region, i.e. we reduce the single region CV by 1/sqrt(regions). |
| long | Upper and lower longitude (RA) limits of interest in units of inunit. If of length 1 then the number specified is assumed to be the upper limit and the lower limit is set to 0. |
| lat | Upper and lower latitude (Dec) limits of interest in units of inunit. If of length 1 then the number specified is assumed to be the upper limit and the lower limit is set to 0. |
| zmax | Maximum redshift of comoving cone. |
| zmin | Minimum redshift of comoving cone. |

| | |
|---|---|
| cosvarsph | The units of angular coordinate provided for long and lat (see skyarea). |
| cosvararea | The units of angular area provided (see cosvol). |

| | |
|---|---|
| inunit | |
| sep | When inunit='sex', sep defines the type of separator used for the HMS and DMS strings (i.e. H:M:S and D:M:S would be sep=':', which is the default). See hms2deg and dms2deg for more details. |
| area | Sky area in units of innunit (default is square degrees) |

## Details

These functions use the empircally motivated cosmic variance percentage formula provided in Driver & Robotham (2010) Eqn 4.

cosvarsph is a 'best effort' approximation of the comoving box subtended by the specified spherical coordinates using the following conversions:

CoDistLow = cosdistCoDist(z=zmin,H0=70,OmegaM=0.3)

CoDistHigh = cosdistCoDist(z=zmax,H0=70,OmegaM=0.3)

cside=CoDistHigh-CoDistLow

area=skyarea(long = long, lat = lat, inunit = inunit, outunit='deg2')[1]

volume=cosvol(area=area, zmax = zmax, zmin=zmin, H0 = 70, OmegaM = 0.3, inunit='deg2')[1]

aside=cos(mean(lat)*pi/180)*(abs(diff(long))/360)*(CoDistLow+cside/2)

bside=(abs(diff(long))/180)*(CoDistLow+cside/2)

scale=sqrt(volume*1e9/(aside*bside*cside))

aside=aside*scale

bside=bside*scale

return(cosvarcar(aside=aside, bside=bside, cside=cside, subsets=subsets))

cosvararea is a simplifed version of cosvarsph, where the assumption is that aside=bside (so the aspect ratio on the sky is 1:1).

## Value

The output is the approximate percentage cosmic (or sample) variance that is expected for the volume specified.

## Note

Many people get upset at the term 'cosmic variance' and prefer 'sample variance'. Whilst I am sympathetic to the argument, more astronomers are familiar with the former term.

These cosmic variance estimates are defined using SDSS at z~0.1, caveats abound at higher redshifts, but these numbers should serve as a reasonably conservative (i.e. pessimistic) upper limit.

## Author(s)

Aaron Robotham and Simon Driver

## References

Driver S.P. & Robotham A.S.G., 2010, MNRAS, 407, 2131

## See Also

cosvol, skyarea

## Examples

```
#Approximate CV of the GAMA equatorial regions:
cosvarsph(long=12, lat=5, zmax=0.5)*1/sqrt(3)
#Or using the GAMA sexigesimal coordinates (should be the same):
cosvarsph(long = c('11:36:0','12:24:0'), lat = c('-2:0:0','3:0:0'), zmax=0.5,
inunit='sex')*1/sqrt(3)
#Approximate CV of the SDSS:
cosvarsph(long=150, lat=100, zmax=0.3)
```

`cosvol` *Cosmological volume calculator*

## Description

Given the sky area, two redshifts and the cosmology, this function calculates the comoving volume.

## Usage

```
cosvol(area=60, zmax=1, zmin=0, H0=100, OmegaM=0.3, OmegaL=1-OmegaM-OmegaR, OmegaR=0,
w0=-1, wprime=0, inunit = "deg2", ref)
```

## Arguments

| | |
|---|---|
| `area` | Sky area in units of innunit (default is square degrees) |
| `zmax` | Maximum cosmological redshift of comoving cone. |
| `zmin` | Minimum cosmological redshift of comoving cone. |
| `H0` | Hubble constant as defined at z=0 (default is H0=100 (km/s)/Mpc) |
| `OmegaM` | Omega Matter as defined at z=0 (default is 0.3). |
| `OmegaL` | Omega Lambda as defined at z=0 (default is for a flat Universe with OmegaL = 1-OmegaM-OmegaR = 0.7). |
| `OmegaR` | Omega Radiation as defined at z=0 (default is 0, but OmegaM/3400 is typical). |
| `w0` | The value of dark energy equation of state as defined at z=0. See `cosgrow` for more details. |
| `wprime` | The evolution term that governs how the dark energy equation of state evolves with redshift. See `cosgrow` for more details. |
| `inunit` | The units of angular area provided. Allowed options are deg2 for square degrees, amin2 for square arc minutes, asec2 for square arc seconds and rad2 or sr for steradians. |
| `ref` | The name of a reference cosmology to use, one of 137 / 737 / Planck / Planck13 / Planck15 / Planck18 / WMAP / WMAP9 / WMAP7 / WMAP5 / WMAP3 / WMAP1 / Millennium / GiggleZ. Planck = Planck18 and WMAP = WMAP9. The usage is case insensitive, so wmap9 is an allowed input. See `cosref` for details. This overrides any other settings for H0/ OmegaM and OmegaL. |

## Value

A 3 element vector. The first element (voltot) specifies the comoving volume of the requested cone segment in Gpc^3, the second element (volmeanz) specifies the mean redshift when mass is uniformly distributed in the volume, the third element (volmedz) specifies the median redshift when mass is uniformly distributed in the volume.

## Author(s)

Aaron Robotham

## References

Based on the equations in:

Hogg D.W., 1999, arXiv, 9905116

Wright E.L., 2006, PASP, 118, 1711

## See Also

cosdist,skyarea, cosmap, cosgrow

## Examples

```
#Approximate volume of the GAMA survey (area given in skyarea example, zmax is approx
#limit of main galaxy sample):
TotalGAMAvol=cosvol(293.82,0.6)[1]
print(paste('The GAMA survey volume is ~',round(TotalGAMAvol,2),'Gpc^3'))

#Approximate volume of SDSS (area given for DR7, zmax is approx limit of main galaxy sample):
TotalSDSSvol=cosvol(8423,0.3)[1]
print(paste('The SDSS survey volume is ~',round(TotalSDSSvol,2),'Gpc^3'))

#Change of reference cosmology
cosvol(293.82,0.6,ref='Planck')
```

---

# deg2dms *Convert decimal degrees to dms format.*

---

## Description

Convert decimal degrees to dms (degrees, minutes, seconds) format. This is probably most useful for declination conversion, since dms is fairly standard method of presenting declination coordinates. The decimal degrees=d+m/60+s/3600. Degrees should range from -90 to +90.

## Usage

```
deg2dms(deg, type='mat', sep=':', digits=2)
```

## Arguments

| | |
|---|---|
| deg | The decimal degrees you are converting. All deg values should be -90<=deg<=90 |
| type | The output type desired. If 'mat' then the output is a 3 column data.frame where column 1 is the degree, column 2 is the minutes and column 3 is the seconds. If 'cat' then the output is a single vector of strings where the separator is defined by the 'sep' argument. |
| sep | Defines the type of separator used when type='cat'. Any value other than 'DMS' and 'dms' is used for all separations, so the default ':' would produce an output like 3:34:45.5. If set to 'dms' or 'DMS' then the output is of the format 3d34m45.5s and 3D34M45.5s resepctively. |

| | |
|---|---|
| digits | The digits to print for angular seconds. See formatC for details on how digits is parsed. |

## Value

A data.frame with the columns degrees, minutes and seconds if type='mat'. If type='cat' then a vector of strings with separators defined by the 'sep' argument.

## Author(s)

Aaron Robotham

## See Also

dms2deg

## Examples

```
print(deg2dms(12.345))
print(deg2dms(12.345,type='cat',sep=':'))
print(deg2dms(12.345,type='cat',sep='dms'))
print(deg2dms(12.345,type='cat',sep='DMS'))
```

---

# deg2hms *Convert decimal degrees to hms format.*

---

## Description

Convert decimal degrees to hms (hours, minutes, seconds) format. This is probably most useful for right-ascension (RA) conversion, since hms is fairly standard method of presenting RA coordinates. The decimal degrees=15*h+15*m/60+15*s/3600 (i.e. there are 24 hours in 360 degrees). Degrees should range from 0 to 360.

## Usage

```
deg2hms(deg, type='mat', sep=':', digits=2)
```

## Arguments

| | |
|---|---|
| deg | The decimal degrees you are converting. All deg values should be 0<=d<=360. |
| type | The output type desired. If 'mat' then the output is a 3 column data.frame where column 1 is the degree, column 2 is the minutes and column 3 is the seconds. If 'cat' then the output is a single vector of strings where the separator is defined by the 'sep' argument. |
| sep | Defines the type of separator used when type='cat'. Any value other than 'DMS' and 'dms' is used for all separations, so the default ':' would produce an output like 3:34:45.5. If set to 'hms' or 'HMS' then the output is of the format 3h34m45.5s and 3H34M45.5s resepctively. |

digits — The digits to print for angular seconds. See formatC for details on how digits is parsed.

## Value

A data.frame with the columns degrees, minutes and seconds if type='mat'. If type='cat' then a vector of strings with separators defined by the 'sep' argument.

## Author(s)

Aaron Robotham

## See Also

hms2deg

## Examples

```
deg2hms(123.456)
deg2hms(123.456,type='cat',sep=':')
deg2hms(123.456,type='cat',sep='hms')
deg2hms(123.456,type='cat',sep='HMS')
```

---

# dms2deg *Convert DMS to degrees format.*

---

## Description

Convert DMS (degrees, minutes, seconds) to degrees format. This is probably most useful for declination conversion, since dms is fairly standard method of presenting declination coordinates. The decimal degrees=d+m/60+s/3600. Degrees should range from -90 to +90. Degrees and minutes should be integer and seconds can be decimal.

## Usage

```
dms2deg(d,m,s,sign='d',sep=':')
```

## Arguments

d — The integer number of degrees you are converting. If it is not integer then the floor of the number is taken. This can contain the sign of the declination when sign='d', but must be all positive if the sign argument is specified (this is required if d contains any 0s, see below). If sign is specified, all d values should be 0<=d<=90, otherwise d values should be -90<=d<=90.

m — The integer number of minutes you are converting. If it is not integer then the floor of the number is taken. All m values should be 0<=m<60.

s — The decimal number of seconds you are converting. All s values should be 0<=s<60.

| | |
|---|---|
| sign | The sign of the declination. The default 'd' inherits the sign of the d argument. This is ambiguous when d is 0 since the sign of +/-0 is taken to be 0. If d contains any 0s, you must supply a vector of the same length as d with +ve or -ve values (e.g. +/- 1), the sign of these value will be taken as the sign for the declination. |
| sep | Defines the type of separator used when 'd' is a vector of strings. Any value other than 'DMS' and 'dms' is used for all separations, so the default ':' would be for an input like 3:34:45.5. If set to 'dms' or 'DMS' then the output is of the format 3d34m45.5s and 3D34M45.5s resepctively. |

## Value

A value of decimal degrees.

## Author(s)

Aaron Robotham

## See Also

deg2dms

## Examples

```
print(dms2deg(70,45,19,-1))
print(dms2deg('-70:45:19'))
print(dms2deg('-70d45m19s',sep='dms'))
print(dms2deg(c('-70D45M19S','3D5M15S'),sep='DMS'))
```

---

# eq2gal *Convert Equtorial to Galactic Coordinates*

---

## Description

Simple conversions accurate to about 0.1 arcseconds between J2000 Equatorial and MW Galactic.

## Usage

```
eq2gal(RA, Dec, pole_RA = 192.859508, pole_Dec = 27.128336, eta = 32.932)
gal2eq(gal_long, gal_lat, pole_RA = 192.859508, pole_Dec = 27.128336, eta = 32.932)
```

## Arguments

| | |
|---|---|
| RA | Numeric vector; Right Ascension in degrees (J2000 system). For convenience this can also be a two column matrix / data.frame containing 'RA' and 'Dec' columns. |
| Dec | Numeric vector; Declination in degrees (J2000 system). |

| | |
|---|---|
| gal_long | Numeric vector; Galactic Longitude in degrees (J2000 system). For convenience this can also be a two column matrix / data.frame containing 'gal_long' and 'gal_lat' columns. |
| gal_lat | Numeric vector; Galactic Latitude in degrees (J2000 system). |
| pole_RA | Numeric scalar; Right Ascension of Galactic Pole in degrees (J2000 system). |
| pole_Dec | Numeric scalar; Declination of Galactic Pole in degrees (J2000 system). |
| eta | Numeric scalar; Tilt of the Earth's axis in degrees (J2000 system). |

## Details

Approximate simple conversions, accurate usully to 0.1 arcseconds

## Value

eq2gal returns the Galactic longitude and latitude in degrees (two column output). gal2eq returns the Equatorial RA and Dec in degress (two column output).

## Author(s)

Aaron Robotham

## References

"Practical astronomy with your calculator" (Peter Duffett-Smith)

## See Also

sph2car,car2sph

## Examples

```
# Notice the conversion anbiguity at the poles

gal2eq(eq2gal(seq(0,360, len=19), seq(-90,90, len=19)))
```

---

# getpixscale *Get Pixel Scale*

---

## Description

Given a FITSio of astro header, calculate the image pixel scale.

## Usage

```
getpixscale(header, CD1_1 = 1, CD1_2 = 0, CD2_1 = 0, CD2_2 = 1)
```

## Arguments

| | |
|---|---|
| header | Full FITS header in table or vector format. Legal table format headers are provided by the read.fitshdr function or the 'hdr' list output of read.fits in the astro package; the 'hdr' output of readFITS in the FITSio package or the 'header' output of magcutoutWCS. Missing header keywords are printed out and other header option arguments are used in these cases. See xy2radec. |
| CD1_1 | FITS header CD1_1 for the Tan Gnomonic projection system. Change in RA-Tan in degrees along x-Axis. |
| CD1_2 | FITS header CD1_2 for the Tan Gnomonic projection system. Change in RA-Tan in degrees along y-Axis. |
| CD2_1 | FITS header CD2_1 for the Tan Gnomonic projection system. Change in Dec-Tan in degrees along x-Axis. |
| CD2_2 | FITS header CD2_2 for the Tan Gnomonic projection system. Change in Dec-Tan in degrees along y-Axis. |

## Details

In most cases users will simply provide a valid header to find the WCS, but you can enter the 'CD' values explicitly. Calculating the pixel scale from the latter is almost trivial, but the option is there for the curious/lazy.

## Value

Numeric scalar; the image pixscale in asec/pixel (so typically a value of 0.1-0.5 for modern survey instruments).

## Author(s)

Aaron Robotham

## Examples

```
## Not run:
#The answer should be almost exactly 0.2 asec/pixel:

#Using FITSio and ProFit packages
image = readFITS(system.file("extdata", 'KiDS/G266035fitim.fits', package="ProFit"))
getpixscale(image$hdr)
#Using astro package
image = read.fits(system.file("extdata", 'KiDS/G266035fitim.fits', package="ProFit"))
getpixscale(image$hdr[[1]])

## End(Not run)
```

`hms2deg` *Convert hms to degrees format.*

## Description

Convert hms (hours, minutes, seconds) to degrees format. This is probably most useful for right ascension (RA) conversion, since hms is fairly standard method of presenting RA coordinates. The decimal degrees=15*h+15*m/60+15*s/3600. Should range between 0 and 24 hours. Hours and minutes should be integer and seconds can be decimal.

## Usage

```
hms2deg(h,m,s,sep=':')
```

## Arguments

| | |
|---|---|
| `h` | The integer number of hours you are converting. If it is not integer then the floor of the number is taken. All m values should be 0<=h<=24. |
| `m` | The integer number of minutes you are converting. If it is not integer then the floor of the number is taken. All m values should be 0<=m<60. |
| `s` | The decimal number of seconds you are converting. All s values should be 0<=s<60. |
| `sep` | Defines the type of separator used when 'h' is a vector of strings. Any value other than 'HMS' and 'hms' is used for all separations, so the default ':' would be for an input like 3:34:45.5. If set to 'hms' or 'HMS' then the output is of the format 3h34m45.5s and 3H34M45.5s resepctively. |

## Value

A value of decimal degrees.

## Author(s)

Aaron Robotham

## See Also

`deg2hms`

## Examples

```
hms2deg(12,10,36)
hms2deg('12:10:36')
hms2deg('12h10m36s',sep='hms')
hms2deg(c('12H10M36S','3H4M10S'),sep='HMS')
```

## IAUID *IAU name creator.*

### Description

Creates IAU legal names for objects given coordinates, name and epoch.

### Usage

```
IAUID(ra, dec, name = "GAMA", epoch = "J")
```

### Arguments

| | |
|---|---|
| ra | Right Ascension in decimal degrees. |
| dec | Declination in decimal degrees. |
| name | Name to be appended to IAU designation as a string. |
| epoch | Epoch, i.e. 'J' (default) or 'B'. Enter as a string. |

### Value

Text string that outputs an IAU legal name for an object.

### Author(s)

Aaron Robotham

### Examples

```
IAUID(123.45,67.89,'GAMA','J')
```

## planck *Planck's Law and Related Functions*

### Description

Functions related to Planck's Law of thermal radiation.

## Usage

```
cosplanckLawRadFreq(nu,Temp=2.725)
cosplanckLawRadWave(lambda,Temp=2.725)
cosplanckLawEnFreq(nu,Temp=2.725)
cosplanckLawEnWave(lambda,Temp=2.725)
cosplanckLawRadFreqN(nu,Temp=2.725)
cosplanckLawRadWaveN(lambda,Temp=2.725)
cosplanckPeakFreq(Temp=2.725)
cosplanckPeakWave(Temp=2.725)
cosplanckSBLawRad(Temp=2.725)
cosplanckSBLawRad_sr(Temp=2.725)
cosplanckSBLawEn(Temp=2.725)
cosplanckLawRadPhotEnAv(Temp=2.725)
cosplanckLawRadPhotN(Temp=2.725)
cosplanckCMBTemp(z,Temp=2.725)
```

## Arguments

| | |
|---|---|
| nu | The frequency of radiation in Hertz (Hz). |
| lambda | The wavelength of radiation in metres (m). |
| Temp | The absolute temperature of the system in Kelvin (K). |
| z | Redshift, where z must be > -1 (can be a vector). |

## Details

The functions with Rad in the name are related the spectral radiance form of Planck's Law (typically designated I or B), whilst those with En are related to the spectral energy density form of Planck's Law (u), where $u = 4\pi I/c$.

To calculate the number of photons in a mode we simply use $E = h\nu = hc/\lambda$.

Below h is the Planck constant, $k_B$ is the Boltzmann constant, c is the speed-of-light in a vacuum and $\sigma$ is the Stefan-Boltzmann constant.

cosplanckLawRadFreq is the spectral radiance per unit frequency version of Planck's Law, defined as:

$$B_\nu(\nu, T) = I_\nu(\nu, T) = \frac{2h\nu^3}{c^2} \frac{1}{e^{h\nu/k_B T} - 1}$$

cosplanckLawRadWave is the spectral radiance per unit wavelength version of Planck's Law, defined as:

$$B_\lambda(\lambda, T) = I_\lambda(\lambda, T) = \frac{2hc^2}{\lambda^5} \frac{1}{e^{hc/\lambda k_B T} - 1}$$

cosplanckLawRadFreqN is the number of photons per unit frequency, defined as:

$$B_\nu(\nu, T) = I_\nu(\nu, T) = \frac{2\nu^2}{c^2} \frac{1}{e^{h\nu/k_B T} - 1}$$

`cosplanckLawRadWaveN` is the number of photons per unit wavelength, defined as:

$$B_\lambda(\lambda, T) = I_\lambda(\lambda, T) = \frac{2c}{\lambda^4} \frac{1}{e^{hc/\lambda k_B T} - 1}$$

`cosplanckLawEnFreq` is the spectral energy density per unit frequency version of Planck's Law, defined as:

$$u_\nu(\nu, T) = \frac{8\pi h\nu^3}{c^3} \frac{1}{e^{h\nu/k_B T} - 1}$$

`cosplanckLawEnWave` is the spectral energy density per unit wavelength version of Planck's Law, defined as:

$$u_\lambda(\lambda, T) = \frac{8\pi hc}{\lambda^5} \frac{1}{e^{hc/\lambda k_B T} - 1}$$

`cosplanckPeakFreq` gives the location in frequency of the peak of $I_\nu(\nu, T)$, defined as:

$$\nu_{peak} = 2.821 k_B T$$

`cosplanckPeakWave` gives the location in wavelength of the peak of $I_\lambda(\lambda, T)$, defined as:

$$\lambda_{peak} = 4.965 k_B T$$

`cosplanckSBLawRad` gives the emissive power (or radiant exitance) version of the Stefan-Boltzmann Law, defined as:

$$j^* = \sigma T^4$$

`cosplanckSBLawRad_sr` gives the spectral radiance version of the Stefan-Boltzmann Law, defined as:

$$L = \sigma T^4 / \pi$$

`cosplanckSBLawEn` gives the energy density version of the Stefan-Boltzmann Law, defined as:

$$\epsilon = 4\sigma T^4 / c$$

Notice that $j^*$ and L merely differ by a factor of $\pi$, i.e. L is per steradian.

`cosplanckLawRadPhotEnAv` gives the average energy of the emitted black body photon, defined as:

$$< E_{phot} >= 3.729282 \times 10^{-23} T$$

`cosplanckLawRadPhotN` gives the total number of photons produced by black body per metre squared per second per steradian, defined as:

$$N_{phot} = 1.5205 \times 10^{15} T^3 / \pi$$

Various confidence building sanity checks of how to use these functions are given in the Examples below.

## Value

Planck's Law in terms of spectral radiance:

`cosplanckLawRadFreq`
: The power per steradian per metre squared per unit frequency for a black body ($W.sr^{-1}.m^{-2}.Hz^{-1}$).

`cosplanckLawRadWave`
: The power per steradian per metre squared per unit wavelength for a black body ($W.sr^{-1}.m^{-2}.m^{-1}$).

Planck's Law in terms of spectral energy density:

`cosplanckLawEnFreq`
: The energy per metre cubed per unit frequency for a black body ($J.m^{-3}.Hz^{-1}$).

`cosplanckLawEnWave`
: The energy per metre cubed per unit wavelength for a black body ($J.m^{-3}.m^{-1}$).

Photon counts:

`cosplanckLawRadFreqN`
: The number of photons per steradian per metre squared per second per unit frequency for a black body ($photons.sr^{-1}.m^{-2}.s^{-1}.Hz^{-1}$).

`cosplanckLawRadWaveN`
: The number of photonsper steradian per metre squared per second per unit wavelength for a black body ($photons.sr^{-1}.m^{-2}.s^{-1}.m^{-1}$).

Peak locations (via Wien's displacement law):

`cosplanckPeakFreq`
: The frequency location of the radiation peak for a black body as found in `cosplanckLawRadFreq`.

`cosplanckPeakWave`
: The wavelength location of the radiation peak for a black body as found in `cosplanckLawRadWave`.

Stefan-Boltzmann Law:

`cosplanckSBLawRad`
: Total energy radiated per metre squared per second across all wavelengths for a black body ($W.m^{-2}$). This is the emissive power version of the Stefan-Boltzmann Law.

`cosplanckSBLawRad_sr`
: Total energy radiated per metre squared per second per steradian across all wavelengths for a black body ($W.m^{-2}.sr^{-1}$). This is the radiance version of the Stefan-Boltzmann Law.

`cosplanckSBLawEn`
: Total energy per metre cubed across all wavelengths for a black body ($J.m^{-3}$). This is the energy density version of the Stefan-Boltzmann Law.

Photon properties:

`cosplanckLawRadPhotEnAv`
: Average black body photon energy (J).

`cosplanckLawRadPhotN`
: Total number of photons produced by black body per metre squared per second per steradian ($m^{-2}.s^{-1}.sr^{-1}$).

Cosmic Microwave Background:

`cosplanckCMBTemp`
: The temperaure of the CMB at redshift z.

## Author(s)

Aaron Robotham

## References

Marr J.M., Wilkin F.P., 2012, AmJPh, 80, 399

## See Also

cosgrow

## Examples

```
#Classic example for different temperature stars:

waveseq=10^seq(-7,-5,by=0.01)
plot(waveseq, cosplanckLawRadWave(waveseq,5000),
log='x', type='l', xlab=expression(Wavelength / m),
ylab=expression('Spectral Radiance' / W*sr^{-1}*m^{-2}*m^{-1}), col='blue')
lines(waveseq, cosplanckLawRadWave(waveseq,4000), col='green')
lines(waveseq, cosplanckLawRadWave(waveseq,3000), col='red')
legend('topright', legend=c('3000K','4000K','5000K'), col=c('red','green','blue'), lty=1)

#CMB now:

plot(10^seq(9,12,by=0.01), cosplanckLawRadFreq(10^seq(9,12,by=0.01)),
log='x', type='l', xlab=expression(Frequency / Hz),
ylab=expression('Spectral Radiance' / W*sr^{-1}*m^{-2}*Hz^{-1}))
abline(v=cosplanckPeakFreq(),lty=2)

plot(10^seq(-4,-1,by=0.01), cosplanckLawRadWave(10^seq(-4,-1,by=0.01)),
log='x', type='l', xlab=expression(Wavelength / m),
ylab=expression('Spectral Radiance' / W*sr^{-1}*m^{-2}*m^{-1}))
abline(v=cosplanckPeakWave(),lty=2)

#CMB at surface of last scattering:

TempLastScat=cosplanckCMBTemp(1100) #Note this is still much cooler than our Sun!

plot(10^seq(12,15,by=0.01), cosplanckLawRadFreq(10^seq(12,15,by=0.01),TempLastScat),
```

```
log='x', type='l', xlab=expression(Frequency / Hz),
ylab=expression('Spectral Radiance' / W*sr^{-1}*m^{-2}*Hz^{-1}))
abline(v=cosplanckPeakFreq(TempLastScat),lty=2)

plot(10^seq(-7,-4,by=0.01), cosplanckLawRadWave(10^seq(-7,-4,by=0.01),TempLastScat),
log='x', type='l', xlab=expression(Wavelength / m),
ylab=expression('Spectral Radiance' / W*sr^{-1}*m^{-2}*m^{-1}))
abline(v=cosplanckPeakWave(TempLastScat),lty=2)

#Exact number of photons produced by black body:

cosplanckLawRadPhotN()

#We can get pretty much the correct answer through direct integration, i.e.:

integrate(cosplanckLawRadFreqN,1e8,1e12)
integrate(cosplanckLawRadWaveN,1e-4,1e-1)

#Stefan-Boltzmann Law:

cosplanckSBLawRad_sr()

#We can get (almost, some rounding is off) the same answer by multiplying
#the total number of photons produced by a black body per metre squared per
#second per steradian by the average photon energy:

cosplanckLawRadPhotEnAv()*cosplanckLawRadPhotN()
```

---

`Sky Coordinate Matching` *Sky matching*

---

## Description

These functions allows the user to match a reference set of sky coordinates against a comparison set of sky coordinates. The match radius can be varied per source (all matches per source are given within this radius), and mutual best matches are also extracted. `coordmatch` should be used for finding multiple matches and `coordmatchsing` should be used when trying to find matches around a single source. `internalclean` is a utility function that will remove closely duplicated objects via some 'tiebreak' criterion, and is probably only of interest to advanced users trying to clean catalogues that were produced from overlapping frames.

## Usage

```
coordmatch(coordref, coordcompare, rad = 2, inunitref = "deg", inunitcompare = "deg",
  radunit = "asec", sep = ":", kstart = 10, ignoreexact = FALSE, ignoreinternal = FALSE,
  matchextra = FALSE, smallapprox = FALSE, jitter = FALSE, jamount = 1e-12, jseed = 666)
```

```
coordmatchsing(RAref, Decref, coordcompare, rad = 2, inunitref = "deg",
  inunitcompare = "deg", radunit = 'asec', sep = ":", ignoreexact = FALSE,
  smallapprox = FALSE)

internalclean(RA, Dec, rad = 2, tiebreak, decreasing = FALSE, Nmatch = 'all',
  iter = FALSE, group = FALSE, ...)

group_links(links, grouptype = 'list', selfgroup = FALSE, return_groupinfo = FALSE,
  return_linkslist = FALSE)

group_graph(links, selfgroup=FALSE)
```

## Arguments

| | |
|---|---|
| coordref | For coordmatch this is the reference dataset, i.e. you want to find matches for each object in this catalogue. A minimum two column matrix or data.frame, where column one is the RA and column two the Dec. See 'matchextra'. |
| coordcompare | The comparison dataset, i.e. you want to find objects in this catalogue that match locations in coordref. A minimum two column matrix or data.frame, where column one is the RA and column two the Dec. If 'coordcompare' is not provided then it is set to 'coordref' automatically. Since this means the user is doing a single table internal match 'ignoreinternal' is automatically set to TRUE (but this can be overridden). See 'matchextra'. |
| RAref | For coordmatchsing this is the reference RA for the single object of interest. |
| Decref | For coordmatchsing this is the reference Dec for the single object of interest. |
| RA | For internalclean this is a vector of right ascensions for internal cleaning. If 'RA' is a two column structure then the second column is taken to be 'Dec'. |
| Dec | For internalclean this is a vector of declinations for internal cleaning. If 'RA' is a two column structure then the second column is taken to be 'Dec'. |
| inunitref | The units of angular coordinate provided for coordref / RAref / Decref. Allowed options are deg for degress, rad for radians and sex for sexigesimal (i.e. HMS for RA and DMS for Deg). |
| inunitcompare | The units of angular coordinate provided for coordcompare. Allowed options are deg for degress, rad for radians and sex for sexigesimal (i.e. HMS for RA and DMS for Deg). |
| radunit | The unit type for the radius specified. Allowed options are deg for degress, amin for arc minutes, asec for arc seconds and rad for radians. |
| sep | If inunitref, inunitcompare or inunit is set to 'sex' then sep defines the separation type as detailed in hms2deg and dms2deg. |
| kstart | The number of matching nodes to attempt initial. The code iterates until all matches within the specified radius (rad) have been found, but it works faster if the kstart is close to the maximum number of matches for any coordref object. |
| ignoreexact | Should exact matches be ignored in the output? If TRUE then 0 separation ID matches are set to 0 and the separation is NA. This might be helpful when matching the same table against itself, where you have no interest in finding object matches with respect to themselves. |

`ignoreinternal` Should identical row matches be ignored in the output? If TRUE then exact row ID matches are set to 0 and the separation is NA. The bestmatch output will ignore these trivial matchesw also. This only makes sense if '`coordref`' and '`coordcompare`' are the same table and you are trying to do an internal table match where you do not want the trivial result of rows matching to themselves. Automatically switches to TRUE if '`coordcompare`' is not provided.

`matchextra` Should extra columns in '`coordref`' and '`coordcompare`' be used as part of the N-D match? Extra columns beyond the requried RA and Dec can be provided and these will be used as part of the N-D match. The meaning of '`rad`' in this case is not trivial of course since the match is done within a hyper-sphere. When the extra columns have the same value '`rad`' can still be interpretted as an angular coordinate match. These extra columns should be appropriately scaled, e.g. you might want to make a 2 arcsec match with an extra magnitude column. In this case even if two objects sit on top of each other on sky, they cannot differ by more than 2 mag in flux to be a match.

`smallapprox` Should the small angle approximation of asin(a/b) = a/b be used? If TRUE then some computations may be much faster, since asin is an expensive computation to make for lots of near matches.

`jitter` Logical; should a very small jitter be applied to the '`coordcompare`' data? This should be negligibly small, but does make the output slightly variable depending on the seed. If '`jitter`' is not explicitly set and '`ignoreinternal`' is TRUE (which is the case if only one catalogue is supplied) and '`ignoreexact`' is FALSE then '`jitter`' is automatically switched to TRUE. See `jitter`.

`jamount` Numeric scaler; max amount of jitter in radians (passed to '`amount`' in `jitter`). The default is less than a millionth of an asec, so there should be no circumstances where this affects real results (that is massively sub Event Horizon Telescope resolution even).

`jseed` Integer scalar; random seed to use for jitter.

`rad` The matching radius to use. If this is length one then the same radius is used for all objects, otherwise it must be the same length as the number of rows in coordref.

`tiebreak` For `internalclean` this is a vector of values to determine the preferred source, e.g. something like magnitude of distance to the centre of the origin frame. By default smaller values are considered better, but this can be flipped by setting '`decreasing`'=TRUE. If '`tiebreak`' is not provided then the first source that appears is considered the better object in the cleaned catalogue.

`decreasing` Determines whether smaller ('`decreasing`'=FALSE) or larger ('`decreasing`'=TRUE) '`tiebreak`' values are considered preferable.

`Nmatch` Character scalar or integer vector; if default 'all' then cleaning is done for any number of matches and no match objects (unambiguous) are passed through. If a vector then the number of matches has to exist in the '`Nmatch`' vector to get passed out. This is useful if we only want cleaned objects in overlap regions where we expect a certain number of matches. Note '`Nmatch`'=1 means two objects match- the base reference and 1 other etc.

`iter` Logical; should `internalclean` by run until no matches within the '`rad`' exist? The reason you might want to do this is to break degeneracies in chains where all

links are within the search 'rad' but the extremes are outside of this. In that situation you might want to keep sub-groups based on their preferred 'tiebreak' (in which case use 'iter' = FALSE) or you might want to fully resolve the whole complexes down to guarantee nothing matches within the 'rad' (in which case use 'iter' = TRUE). The latter is the only solution that guarantees no sources are left that match within 'rad'. See Examples.

group — Logical; this goes a step further than 'iter' and cretes full FoF groups. See Examples.

links — Integer Matrix; the two column link associations that we wish to group together. These do not need to be symmetric and can be non-contiguous, they must be integers though.

grouptype — Characeter scalar; the type of group output, options are 'list': a list of the groups; or 'DF': a data.frame, where column 1 is the linkID and column 2 the groupID.

selfgroup — Logical; should 1-1 self-groupings be ignored (i.e. 2 matching to 2)?

return_groupinfo — Logical; should basic group info (Ngroup, Nlist, IDmin, IDmax) be returned as a data.frame? Note Nlist is only present if 'return_linkslist' is also TRUE.

return_linkslist — Logical; should the links that form the groups be returned?

... — Other arguments to be passed to coordmatch.

## Details

For coordmatch the main matching is done using nn2 that comes as part of the RANN package. coordmatch adds a large amount of sky coordinate oriented functionality beyond the simple implementation of nn2. For single object matches coordmatchsing should be used since it is substantially faster in this regime (making use of direct dot products).

'ignoreexact' is more strict in a sense since all objects exactly matching are ignored, whereas with 'ignoreinternal' only identical row IDs are interpretted as being the same object. This setting can be useful in internalclean (via ...) too, since having exact matches and internal matches can create ambiguous cleaning.

## Value

The output of coordmatch is a list containing:

ID — The full matrix of matching IDs. The rows are ordered identically to 'coordref', and the ID value is the row position in 'coordcompare' for the match.

sep — The full matrix of matching separations in the same units as 'radunit'. The rows are ordered identically to 'coordref', and the sep value is the separation for each matrix location in the ID list object.

Nmatch — Nmatch is a vector giving the total number of matches for each 'coordref' row.

bestmatch — A three column data.frame giving the best matching IDs. Only objects with at least one match are listed. Column 1 (refID) gives the row position from 'coordref' and column 2 (compareID) gives the corresponding best matching row position in 'coordcompare'. Column 3 (sep) gives the separation between the matched ref and compare positions in the same units as radunit.

The output of coordmatchsing is a list containing:

| | |
|---|---|
| ID | The full vector of matching IDs. The ID values are the row positions in 'coordcompare' for the match. |
| sep | The full vector of matching separations in the same units as 'radunit'. The sep value is the separation for each vector location in the ID list object. |
| Nmatch | Total number of matches within the specified radius. |
| bestmatch | The best matching ID, where the ID value is the row position in 'coordcompare' for the match. |

The output of internalclean is a vector containing IDs of the rows to keep to achieve the cleaned catalogue.

The output of group_links depends on the return parameters. Minimally it is a list of the FoF groups. Can also be a list structure containing:

| | |
|---|---|
| group | The list of the FoF groups (if 'grouptype' = 'list') or data.frame containing linkID and groupID (if 'grouptype' = 'DF'). |
| groupinfo | The table of basic group information when 'return_groupinfo' is TRUE. The data.frame will contain Ngroup, Nlist, IDmin, IDmax, though note Nlist is only present if 'return_linkslist' is also TRUE. |
| linkslist | The list of the links that form the respective groups. |

The output of group_graph is the data.table containing linkID and groupID

## Author(s)

Aaron Robotham

## See Also

hms2deg, dms2deg, sph2car

## Examples

```
set.seed(666)

#Here we make objects in a virtual 1 square degree region

mocksky=cbind(runif(1e3), runif(1e3))

#Now we match to find all objects within an arc minute, ignoring self matches

mockmatches=coordmatch(mocksky, mocksky, ignoreexact=TRUE, rad=1, radunit='amin')

#Now we match to find all objects with varying match radii, ignoring self matches

mockmatchesvary=coordmatch(mocksky, mocksky, ignoreexact=TRUE, rad=seq(0,1,length=1e3),
radunit='amin')
```

```
#We can do this also by using the internal table match mode:

mockmatchesvary2=coordmatch(mocksky, rad=seq(0,1,length=1e3), radunit='amin')

#Check that this looks the same (should be identical with all zeroes):

summary(mockmatchesvary$bestmatch-mockmatchesvary2$bestmatch)

#Internal matching can be complicated:

coords = cbind(c(-8:8, 11:14),c(-8:8, 11:14))*0.0001
plot(coords)

#Note the search radius is such two adjacent objects can be associated:

library(plotrix)
points(0,0,pch=4)
draw.circle(0,0,radius=1.2/3600)
draw.circle(8e-4,8e-4,radius=1.2/3600)

(internalclean(coords, rad=1.2))
(internalclean(coords, rad=1.2, iter=TRUE))
(internalclean(coords, rad=1.2, group=TRUE))

#Basic Fof groups:
set.seed(666)
links = cbind(sample(40,20,replace=TRUE), sample(40,20,replace=TRUE))
print(links)

group_links(links, return_groupinfo=TRUE)

group_graph(links)
```

---

`skyarea` *Exact angular area calculator*

---

## Description

This function takes a survey geometry defined by RA (long) and Dec (latitude) limits and calculates the exact angular area covered.

## Usage

```
skyarea(long = c(129, 141), lat = c(-2, 3), inunit = "deg", outunit = "deg2", sep=":")
```

## Arguments

| | |
|---|---|
| `long` | Upper and lower longitude (RA) limits of interest in units of inunit. If of length 1 then the number specified is assumed to be the upper limit and the lower limit is set to 0. |

| | |
|---|---|
| lat | Upper and lower latitude (Dec) limits of interest in units of inunit. If of length 1 then the number specified is assumed to be the upper limit and the lower limit is set to 0. |
| inunit | The units of angular coordinate provided. Allowed options are deg for degress, amin for arc minutes, asec for arc seconds, rad for radians and sex for sexigesimal (i.e. HMS for RA and DMS for Deg). |
| outunit | The units of angular area desired. Allowed options are deg2 for square degrees, amin2 for square arc minutes, asec2 for square arc seconds and rad2 or sr for steradians. |
| sep | When inunit='sex', sep defines the type of separator used for the HMS and DMS strings (i.e. H:M:S and D:M:S would be sep=':', which is the default). See hms2deg and dms2deg for more details. |

## Value

Two value vector. First value is the sky area covered in units of outunit (named area), second value is the fraction of the celestial sphere covered by the specified geometry (named areafrac).

## Author(s)

Aaron Robotham

## See Also

cosvol, hms2deg, dms2deg

## Examples

```
#The GAMA survey areas:
G02area=skyarea(c(30.2,38.8),c(-10.25,-3.72))
G09area=skyarea(c(129,141),c(-2,3))
G12area=skyarea(c(174,186),c(-3,2))
G15area=skyarea(c(211.5,223.5),c(-2,3))
G23area=skyarea(c(338.1,351.9),c(-35,-30))

#Total GAMA survey area:
TotalGAMAarea=G02area+G09area+G12area+G15area+G23area
paste('The GAMA survey area is',round(TotalGAMAarea['area'],2),'sq. deg.')

#Future TACs note: this is less than 1% of the sky ;-)
paste('The GAMA survey area is',round(TotalGAMAarea['areafrac']*100,2),'% of the sky')
```

skyproj *Tan Gnomonic and Sine Orthographic Projection System WCS Solver Functions*

## Description

Converts RA/Dec (degrees) to x/y (pixels) position using the Tan Gnomonic or Sine Orthographic projection systems, and vice-versa. Translations adapted from: http://mathworld.wolfram.com/GnomonicProjection.html and http://mathworld.wolfram.com/OrthographicProjection.html.

## Usage

```
radec2xy(RA, Dec, header, CRVAL1 = 0, CRVAL2 = 0, CRPIX1 = 0, CRPIX2 = 0, CD1_1 = 1,
CD1_2 = 0, CD2_1 = 0, CD2_2 = 1, CTYPE1 = 'RA--TAN', CTYPE2 = 'DEC--TAN')
xy2radec(x, y, header, CRVAL1 = 0, CRVAL2 = 0, CRPIX1 = 0, CRPIX2 = 0, CD1_1 = 1,
CD1_2 = 0, CD2_1 = 0, CD2_2 = 1, CTYPE1 = 'RA--TAN', CTYPE2 = 'DEC--TAN')
getimlim(x, y, header, CRVAL1 = 0, CRVAL2 = 0, CRPIX1 = 0, CRPIX2 = 0, CD1_1 = 1,
CD1_2 = 0, CD2_1 = 0, CD2_2 = 1, CTYPE1 = 'RA--TAN', CTYPE2 = 'DEC--TAN')
```

## Arguments

| | |
|---|---|
| RA | Vector or matrix; target right ascension in degrees. If matrix then the first column will be used as RA and the second column as Dec. |
| Dec | Vector; target declination in degrees. Ignored if 'RA' is a matrix. |
| x | Vector or matrix; target x-pixel. If Matrix then the first column will be used as the x-axis and the second column as y-axis. For getimlim this can be an image matrix or an image and header combination. |
| y | Vector; target y-pixel. Ignored if 'x' is a matrix. |
| CRVAL1 | FITS header CRVAL1 for the 'CTYPE1' projection system. This is the RA in degrees at the location of 'CRPIX1'. |
| CRVAL2 | FITS header CRVAL2 for the 'CTYPE2' projection system. This is the Dec in degrees at the location of 'CRPIX2'. |
| CRPIX1 | FITS header CRPIX1 for the 'CTYPE1' projection system. This is the x pixel value at the location of 'CRVAL1'. |
| CRPIX2 | FITS header CRPIX2 for the 'CTYPE2' projection system. This is the y pixel value at the location of 'CRVAL2'. |
| CD1_1 | FITS header CD1_1 for the 'CTYPE1' projection system. Change in 'CTYPE1' in degrees along x-Axis. |
| CD1_2 | FITS header CD1_2 for the 'CTYPE1' projection system. Change in 'CTYPE1' in degrees along y-Axis. |
| CD2_1 | FITS header CD2_1 for the 'CTYPE2' projection system. Change in 'CTYPE2' in degrees along x-Axis. |
| CD2_2 | FITS header CD2_2 for the 'CTYPE2' projection system. Change in 'CTYPE2' in degrees along y-Axis. |

| | |
|---|---|
| CTYPE1 | The RA projection system type. Either 'RA–TAN' for Tan Gnomonic (default), or 'RA–SIN' for Sine Orthographic. 'RA–NCP' is approximated by Sine Orthographic with a warning. Over-ridden by the FITS header. |
| CTYPE2 | The DEC projection system type. Either 'DEC–TAN' for Tan Gnomonic (default), or 'DEC–SIN' for Sine Orthographic. 'DEC–NCP' is approximated by Sine Orthographic with a warning. Over-ridden by the FITS header. |
| header | Full FITS header in table or vector format. Legal table format headers are provided by the `read.fitshdr` function or the 'hdr' list output of `read.fits` in the `astro` package). Also the 'hdr' output of `readFITS` in the `FITSio` package provides legal vector format inputs. If a header is provided then key words will be taken from here as a priority. Missing header keywords are printed out and other header option arguments are used in these cases. |

## Details

These functions encode the standard FITS Tan Gnomonic and Sine Orthographic projection systems for solving an image WCS (covering most moden imaging and radio data). They do not deal with higher order polynomial distortion terms.

## Value

| | |
|---|---|
| radec2xy | Returns a two column matrix with columns x and y. |
| xy2radec | Returns a two column matrix with columns RA and Dec (in degrees). |
| getimlim | Returns a list of RA and Dec limits for the target image ('RAlims' and 'Declims', both with 2 elements for lower and upper limits). |

## Author(s)

Aaron Robotham

## References

http://mathworld.wolfram.com/GnomonicProjection.html http://mathworld.wolfram.com/OrthographicProjection.html

## See Also

deg2dms, deg2hms, dms2deg, hms2deg

## Examples

```
#A simple example:

radec2xy(10, 20)
xy2radec(radec2xy(10, 20))
xy2radec(radec2xy(10, 20, CTYPE1='RA--SIN', CTYPE2='DEC--SIN'),
CTYPE1='RA--SIN',CTYPE2='DEC--SIN')

#A more complicated example, where we transform and rotate large amounts:
```

```
exdata_start=expand.grid(1:10,21:30)
plot(exdata_start)
exradec=radec2xy(exdata_start, CRVAL1=20, CRPIX1=100, CRVAL2=30, CRPIX2=130, CD1_1=0.1,
CD1_2=-0.05, CD2_1=0.05, CD2_2=0.1)
plot(exradec)
exdata_end=xy2radec(exradec, CRVAL1=20, CRPIX1=100, CRVAL2=30, CRPIX2=130, CD1_1=0.1,
CD1_2=-0.05, CD2_1=0.05, CD2_2=0.1)
plot(exdata_start,cex=2)
points(exdata_end,col='red')

#The residuals should be very small (in the noice of double precision arithmetic):

plot(density(exdata_start[,1]-exdata_end[,1]))
lines(density(exdata_start[,2]-exdata_end[,2]),col='red')
```

---

sph2car *Transforms 3D spherical coordinates to cartesian coordinates*

---

## Description

Transforms 3D spherical coordinates to cartesian coordinates. The user can choose to input the spherical coordinates in degrees or radians.

## Usage

```
sph2car(long, lat, radius = 1, deg = TRUE)
```

## Arguments

| | |
|---|---|
| long | Longitude values, can also contain a matrix of long, lat and radius (in that order). |
| lat | Latitude values. |
| radius | Radius values. |
| deg | Specifies if input is in degrees (default) or radians. |

## Details

This is a low level function that is used for plot transformations.

## Value

A data.frame is returned containing the columns x, y and z.

## Author(s)

Aaron Robotham

## See Also

coordmatch

## Examples

```
print(sph2car(45,0,sqrt(2),deg=TRUE))
```

# Index