

# Package ‘insol’

October 13, 2022

**Type** Package

**Title** Solar Radiation

**Version** 1.2.2

**Date** 2021-02-10

**Author** Javier G. Corripio

**Maintainer** Javier G. Corripio <jgc@meteoexploration.com>

**Depends** methods

**Imports** raster

**Suggests** datasets,graphics,rgl,stats

**Description** Functions to compute insolation on complex terrain.

**License** GPL-2

**URL** <https://meteoexploration.com/R/insol/index.html>

**LazyLoad** yes

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2021-02-10 15:30:05 UTC

## R topics documented:

insol-package . . . . .	2
aspect . . . . .	3
cgrad . . . . .	4
daydoy . . . . .	7
daylength . . . . .	8
declination . . . . .	9
degrees . . . . .	10
doshade . . . . .	11
doyday . . . . .	13
eqtime . . . . .	14
GCdistance . . . . .	15

hillshading . . . . .	16
hourangle . . . . .	18
insolation . . . . .	19
JD . . . . .	22
JDymd . . . . .	24
meteoandes . . . . .	25
normalvector . . . . .	26
p2rho . . . . .	27
radians . . . . .	28
rh2sh . . . . .	29
slope . . . . .	30
sunpos . . . . .	31
sunr . . . . .	32
sunvector . . . . .	33
wvapsat . . . . .	34
z2p . . . . .	35

<b>Index</b>	<b>36</b>
--------------	-----------

---

insol-package	<i>Solar Radiation</i>
---------------	------------------------

---

## Description

Calculates insolation and cast shadows on tilted and irregular surfaces, computes atmospheric transmittance and related parameters such as: Earth radius vector, declination, sunset and sunrise, daylength, equation of time, vector in the direction of the sun, vector normal to surface, and some atmospheric physics.

## Details

Package:	insol
Type:	Package
Version:	1.2
Date:	2019-01-16
License:	GPL-2
LazyLoad:	yes

## Author(s)

Javier G. Corripio

Maintainer: Javier G. Corripio <jgc@meteoexploration.com>

Additional information: <https://meteoexploration.com/R/insol/>

## References

- Bird, R. E. and Hulstrom, R. L. (1981a) Review, evaluation and improvements of direct irradiance models, *Trans. ASME J. Solar Energy Eng.* 103, 182-192.
- Bird, R. E. and Hulstrom, R. L. (1981b) *A simplified clear sky model for direct and diffuse insolation on horizontal surfaces*, Technical Report SERI/TR-642-761, Solar Research Institute, Golden, Colorado.
- Iqbal, M. (1983) *An Introduction to Solar Radiation*, Academic Press, Toronto.
- Bourges, B.: 1985, Improvement in solar declination computation, *Solar Energy* 35(4), 367-369.
- Corripio, J. G.: 2003, Vectorial algebra algorithms for calculating terrain parameters from DEMs and the position of the sun for solar radiation modelling in mountainous terrain, *International Journal of Geographical Information Science* 17(1), 1-23.
- Danby, J. M. Eqn. 6.16.4 in *Fundamentals of Celestial Mechanics*, 2nd ed. Richmond, VA: Willmann-Bell, p. 207, 1988.
- Meeus, J. 1999. *Astronomical Algorithms*. Willmann-Bell, Richmond, Virginia, USA.
- Reda, I. and Andreas, A. 2003. *Solar Position Algorithm for Solar Radiation Applications*. 55 pp.; NREL Report No. TP-560-34302, Revised January 2008. <https://www.nrel.gov/docs/fy08osti/34302.pdf>
- <https://www.esrl.noaa.gov/gmd/grad/solcalc/calcdetails.html>

---

aspect

*Aspect or orientation of the slope*

---

## Description

Calculates the aspect of every grid cell in a digital elevation model (DEM) from the output of cellgradient, which is a set of unit vectors normal to every grid cell in the DEM.

## Usage

```
aspect(cgrad, degrees = FALSE)
```

## Arguments

cgrad	A 3D array of dimensions [nrow(dem), ncol(dem), 3], where the third dimensions are the x, y, z component of the unit vectors normal to the surface of the DEM grid cells.
degrees	Logical. If FALSE, returns radians, if TRUE, returns degrees.

## Details

Uses atan2() to compute the orientation within the range  $[0, 2\pi]$

## Value

Aspect or orientation of the slope.

**See Also**

[slope](#), [cgrad](#)

**Examples**

```
# Create a west-east facing ramp
slpwe = rep(1,10) %o% c(1:5,4:1)
# calculate the aspect at every node or grid cell (it should be 270 or 90 degrees):
cgr = cgrad(slpwe,1)
aspect(cgr,degrees=TRUE)

## Not run: ## raster package display nicer than image and handles projections:
# Calculate the aspect of a rough mountain area in the pyrenees
zipfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/dempyrenees.asc.zip",zipfile)
header = read.table(unz(zipfile,'dempyrenees.asc'),nrows=6)
dem = read.table(unz(zipfile,'dempyrenees.asc'),skip=6)
dem = as.matrix(dem)
unlink(zipfile)
cellsize = header[5,2]
aspectdem = aspect(cgrad(dem,cellsize),degrees=TRUE)
image(t(aspectdem[nrow(aspectdem):1,]),col=grey(1:100/100))

require(raster)
demfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/dempyrenees.tif",demfile)
dem = raster(demfile)
aspectdem = aspect(cgrad(dem),degrees=TRUE)
aspectdem = raster(aspectdem,crs=projection(dem))
extent(aspectdem) = extent(dem)
plot(aspectdem,col=grey(1:100/100))
unlink(demfile)

## End(Not run)
```

---

cgrad

*Normal vector to every grid cell in a DEM*

---

**Description**

Computes a unit vector normal to every grid cell in a digital elevation model.

**Usage**

```
cgrad(dem, dlx, dly = dlx, cArea = FALSE)
```

**Arguments**

dem	Digital Elevation Model, either a matrix or a Raster Layer.
d1x	Resolution along X axis (longitude).
d1y	Resolution along Y axis (latitude).
cArea	Logical, if TRUE returns the surface area of every grid cell instead of the gradient.

**Details**

The normal vector to the grid cell is the cross product of vectors along the sides of the polygon that form the grid cell. By definition the length of this vector is equal to the area of the polygon.

**Value**

Returns a 3D matrix with the 2 first dimensions as input dem and the 3rd dimension of value 3 corresponding to the x, y, z coordinates of a unit vector perpendicular to every grid cell. If cArea is TRUE, the result is a 2D matrix with the surface area of every grid cell.

**warning**

d1x and d1y are assumed to be constant over the extension of the DEM, therefore the projection should not be lat.long. In this case the resolution is a constant angle, and the equivalent distance on the surface changes with latitude, giving incorrect results.

**Note**

The returned information for every cell is contained by the node at the upperleft corner and the last column and row are undefined. The values given for the last column and row are a replication of the previous column and row.

**Author(s)**

Javier G. Corripio

**References**

Corripio, J. G.: 2003, Vectorial algebra algorithms for calculating terrain parameters from DEMs and the position of the sun for solar radiation modelling in mountainous terrain, *International Journal of Geographical Information Science* 17(1), 1-23.

**See Also**

[aspect](#), [slope](#),

## Examples

```

## visualize x, y z components of vector normal to a DEM representing a regular pyramid
m = matrix(0,nrow=100,ncol=100)
for (i in 1:100){ for (j in 1:100){
m[i,j]=50-max(abs(i-50),abs(j-50)) }}
grdm = cgrad(m,1)
xcomponent = grdm[,1]
ycomponent = grdm[,2]
zcomponent = grdm[,3]
image(t(xcomponent[nrow(xcomponent):1,]) ,col=grey(1:10/10))
image(t(ycomponent[nrow(ycomponent):1,]) ,col=grey(1:10/10))
image(t(zcomponent[nrow(zcomponent):1,]) ,col=grey(1:10/10))

## Not run:
## Surface area of every grid cell in a mountain region
## Steep slopes correspond to larger surface area per grid cell
zipfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/dempyrenees.asc.zip",zipfile)
header = read.table(unz(zipfile,'dempyrenees.asc'),nrows=6)
dem = read.table(unz(zipfile,'dempyrenees.asc'),skip=6)
dem = as.matrix(dem)
unlink(zipfile)
cellsize = header[5,2]
grdarea = cgrad(dem,cellsize,cArea=TRUE)
image(t(grdarea[nrow(grdarea):1,]),col=grey(100:1/100))

## plot the relationship between surface slope and surface area per grid cell:
slpg = slope(cgrad(dem,cellsize),degrees=TRUE)
plot(slpg,grdarea)

## This would be a linear relationship in 2D,
## but in a DEM the slope of the grid cell depends on 4 points in 3D
plot(tan(radians(slpg)),grdarea)

## Use raster for better display and keep the dem projection
require(raster)
demfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/dempyrenees.tif",demfile)
dem = raster(demfile)
plot(dem)
grd = cgrad(dem)
grdarea = cgrad(dem,cArea=TRUE)
rgrdarea = raster(grdarea,crs=projection(dem))
extent(rgrdarea) = extent(dem)
plot(rgrdarea,col = grey(100:1/100))
contour(dem,col='sienna1',lwd=.5,nlevels=30,add=TRUE)
unlink(demfile)

## End(Not run)

```

---

daydoy	<i>Dates to day of the year</i>
--------	---------------------------------

---

**Description**

Returns day of the year for given dates.

**Usage**

```
## S4 method for signature 'numeric'  
daydoy(x, month, day)
```

```
## S4 method for signature 'POSIXct'  
daydoy(x)
```

**Arguments**

x	Year, four digits format, or an object of class POSIXct with no extra arguments
month	Month number.
day	Day of the month.

**Value**

Day of the year [1:366].

**See Also**

[ISOdate](#)

**Examples**

```
daydoy(2019, 2, 27:29)  
daydoy(ISOdate(2019, 2, 27:29))
```

---

daylength	<i>Length of daylight</i>
-----------	---------------------------

---

**Description**

Compute duration of day light for a given latitude and Julian Day.

**Usage**

```
daylength(lat, long, jd, tmz)
```

**Arguments**

lat	Latitude in degrees and decimal fraction.
long	Longitude in degrees and decimal fraction.
jd	Julian Day.
tmz	Timezone, west of Greenwich is negative.

**Details**

It considers sunrise and sunset as the time when the center of the sun pass above or below the horizon, it does not take into account limb, summer time, atmospheric refraction or twilight.

**Value**

sunrise	Time of sunrise.
sunset	Time of sunset.
daylen	Duration of daylight in hours and decimal fraction.

It returns NA for sunrise and sunset during the polar night.

**Note**

You may like to double check at: <https://www.esrl.noaa.gov/gmd/grad/solcalc/azel.html>

**Author(s)**

Javier G. Corripio

**References**

Corripio, J. G.: 2003, Vectorial algebra algorithms for calculating terrain parameters from DEMs and the position of the sun for solar radiation modelling in mountainous terrain, *International Journal of Geographical Information Science* 17(1), 1-23.

**See Also**

[declination](#), [eqtime](#)



**Examples**

```
daylength(47, 11, JDymd(2019, 1, 1, 12), 1)
daylength(c(47, 75), 11, 2456282, 1)

# Daylength for the whole 2019 year
jd2019=JD(seq(ISOdate(2019, 1, 1), ISOdate(2019, 12, 31), by='day'))
plot(daylength(47, 11, jd2019, 1)[, 3], xlab='Day of the year', ylab='day length [h]', ylim=c(0, 24))
```

---

declination	<i>Declination</i>
-------------	--------------------

---

**Description**

Computes the declination of the Sun for a given Julian Day.

**Usage**

```
declination(jd)
```

**Arguments**

jd                    Julian Day.

**Value**

Declination in degrees and decimal fraction.

**Author(s)**

Javier G. Corripio

**References**

<https://www.esrl.noaa.gov/gmd/grad/solcalc/calcdetails.html>

Meeus, J. 1999. *Astronomical Algorithms*. Willmann-Bell, Richmond, Virginia, USA.

Reda, I. and Andreas, A. 2003. *Solar Position Algorithm for Solar Radiation Applications*. 55 pp.; NREL Report No. TP-560-34302, Revised January 2008. <https://www.nrel.gov/docs/fy08osti/34302.pdf>

**Examples**

```
declination(JDymd(2019,1,1))

jdays = JD(ISOdate(2019,1:12,21))
declination(jdays)

## Plot daily changes in declination from 2018 to 2020
jdays=JD(seq(ISOdate(2018,1,1),ISOdate(2020,12,31),by='day'))
plot(declination(jdays),xlab='days from 2018-01-01',ylab='declination')
```

---

degrees

*Radians to degrees*

---

**Description**

Accessory function to transform radians into degrees.

**Usage**

```
degrees(radian)
```

**Arguments**

radian            Angle in radians and decimal fraction.

**Value**

Angle in degrees.

**See Also**

[radians](#)

**Examples**

```
degrees(seq(0,2*pi,pi/2))
```

---

doshade	<i>Cast shadows</i>
---------	---------------------

---

### Description

Calculates cast shadows over matrix or Raster Layer DEM for a given illumination direction.

### Usage

```
doshade(dem, sv, dl = 0, sombra = dem)
```

### Arguments

dem	Digital elevation model, a matrix or RasterLayer representing terrain elevation on a regular grid.
sv	Unit vector in the direction of the sun.
dl	Grid spacing. Not needed if dem is a Raster Layer.
sombra	Returned matrix or Raster Layer, no input needed for this argument.

### Details

doshade calls a fortran routine that scans the DEM in lines parallel the sun direction. It compares the projection of grid cells on a plane perpendicular to the sun to determine whether they are in the sun or in the shadow of a previous cell. See Figure 6 of reference for more details.

### Value

Return an object of the same class the the input dem (either a matrix o a Raster Layer), with values 0 for shaded or 1 for not shaded.

### Author(s)

Javier G. Corripio

### References

Corripio, J. G.: 2003, Vectorial algebra algorithms for calculating terrain parameters from DEMs and the position of the sun for solar radiation modelling in mountainous terrain, *International Journal of Geographical Information Science* 17(1), 1-23.

### Examples

```
# define the sun vector: northwest at 15 degrees elevation
sv = normalvector(75,315)

## create a pyramid 100 units by side and 50 nunits tall
m = matrix(0,nrow=100,ncol=100)
```

```

for (i in 1:100){ for (j in 1:100){
m[i,j] = 50-max(abs(i-50),abs(j-50)) }}

## place it on a large flat expanse
mm = matrix(0,nrow=500,ncol=500)
mm[201:300,201:300] = m

## calculate and plot the cast shadows from the sun
sh = doshade(mm,sv,1)
image(t(sh[nrow(sh):1,]),col=grey(1:100/100))
contour(mm,add=TRUE,col='sienna1',nlevels=25)
## (mm is symmetrical, no need to rotate as for shadows)

## Not run:
## plot cast shadows on mountain terrain, sun at NW, 25 degrees elevation
zipfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/dempyrenees.asc.zip",zipfile)
header = read.table(unz(zipfile,'dempyrenees.asc'),nrows=6)
dem = read.table(unz(zipfile,'dempyrenees.asc'),skip=6)
dem = as.matrix(dem)
unlink(zipfile)
cellsize=header[5,2]
sv = normalvector(65,315)
sh = doshade(dem,sv,cellsize)
image(t(sh[nrow(sh):1,]),col=grey(1:100/100))

## add intensity of illumination in this case sun at NW 45 degrees elevation
sv = normalvector(45,315)
grd = cgrad(dem,cellsize)
hsh = grd[,1]*sv[1]+grd[,2]*sv[2]+grd[,3]*sv[3]
## remove negative incidence angles (self shading)
hsh = (hsh+abs(hsh))/2
sh = doshade(dem,sv,cellsize)
hshsh = hsh*sh
image(t(hshsh[nrow(sh):1,]),col=grey(1:100/100))

## plot cast shadows on mountain terrain using raster
sv = normalvector(65,315)
require(raster)
demfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/dempyrenees.tif",demfile)
dem = raster(demfile)
sh = doshade(dem,sv)
plot(sh,col=grey(0:1),legend=FALSE)
contour(dem,add=TRUE,col='sienna1',lwd=.5,nlevels=50)

## add intensity of illumination in this case sun at NW 45 degrees elevation
sv = normalvector(45,315)
grd = cgrad(dem)
hsh = grd[,1]*sv[1]+grd[,2]*sv[2]+grd[,3]*sv[3]
## remove negative incidence angles (self shading)
hsh = (hsh+abs(hsh))/2

```

```
## convert back to raster as dem and add shadows
hsh = raster(hsh,crs=projection(dem))
extent(hsh) = extent(dem)
sh = doshade(dem,sv)
plot(hsh*sh,col=grey(1:100/100),legend=FALSE)
unlink(demfile)

## End(Not run)
```

---

doyday

*Day of the year to date*

---

### Description

Returns the date for given days of the year.

### Usage

```
doyday(year, doy)
```

### Arguments

year	Year, four digits format. It can have a decimal fraction if day is omitted.
doy	Day of the year [1:366].

### Value

returns an object of class POSIXlt.

### Author(s)

Javier G. Corripio

### See Also

[as.POSIXlt](#)

### Examples

```
doyday(2019,58:65)
```

```
doyday(2019.5)
```

---

eqtime

*Equation of time*

---

### Description

Computes the equation of time for a given Julian Day.

### Usage

```
eqtime(jd)
```

### Arguments

jd                    Julian Day.

### Value

Equation of time in minutes.

### Author(s)

Javier G. Corripio

### References

<https://www.esrl.noaa.gov/gmd/grad/solcalc/calcdetails.html>

Meeus, J. 1999. *Astronomical Algorithms*. Willmann-Bell, Richmond, Virginia, USA.

Reda, I. and Andreas, A. 2003. *Solar Position Algorithm for Solar Radiation Applications*. 55 pp.; NREL Report No. TP-560-34302, Revised January 2008. <https://www.nrel.gov/docs/fy08osti/34302.pdf>

### Examples

```
# plot the equation of time for 2013 at daily intervals
jdays = seq(ISOdate(2013,1,1),ISOdate(2013,12,31),by='day')
jd = JD(jdays)
plot(eqtime(jd))
abline(h=0,col=8)

# Analema
plot(eqtime(jd),declination(jd))

# Analema from Greenwich Observatory
latGwch = 51.4791
x = 180+eqtime(jd)*15/60
y = 90-latGwch+declination(jd)
plot(x,y,ylim=c(0,90),xlab=expression(paste('Azimuth (' ,degree,')')),
ylab=expression(paste('Elevation (' ,degree,')')))
```

```

## Add the solstices and equinoxes (nearest day, see Meeus ch. 26 for more precision)
decl = declination(jd)
wintersolstice = which(decl==min(decl))
summersolstice = which(decl==max(decl))
## spring equinox: when declination becomes zero in the first part of the year
springeqx = uniroot(declination, jd[c(1,180)])$root
springeqx = daydoy(JD(springeqx, inv=TRUE))
autumeqx = uniroot(declination, jd[c(180,360)])$root
autumeqx = daydoy(JD(autumeqx, inv=TRUE))
nodeseqx = c(springeqx, summersolstice, autumeqx, wintersolstice)
points(x[nodeseqx], y[nodeseqx], pch=19, col=3)
abline(h=c(90-latGwch, 90-latGwch+max(decl), 90-latGwch+min(decl)), col=8)

```

---

GCdistance

*Great circle distance*


---

### Description

Great circle or geodesic distance.

### Usage

```
GCdistance(lat1, lon1, lat2, lon2)
```

### Arguments

lat1	Latitude of points of origin.
lon1	Longitude of points of origin.
lat2	Latitude of points of destination.
lon2	Longitude of points of destination.

### Value

Distance between origin and destination points in metres.

### Author(s)

Javier G. Corripio

### References

<https://edwilliams.org/avform.htm>,  
<http://mathworld.wolfram.com/GreatCircle.html>.

**Examples**

```

GCdistance(0,0,0,180)*2

# distance between the center of US states
data(state)
ddd = matrix(nrow=50,ncol=50,dimnames=list(state.name,state.name))
for (i in 1:50){
  for (j in 1:50){
    distij = GCdistance(state.center$y[i],state.center$x[i],
state.center$y[j],state.center$x[j])
    # round to miles
    ddd[i,j]=round(distij/1609.344,2)
  }
}
# format and print results for the 10 firsts states
as.dist(ddd[1:10,1:10])

```

---

hillshading

*Intensity of illumination over a surface*


---

**Description**

Computes the intensity of illumination over a surface, such as a DEM, according to the position of the sun.

**Usage**

```
hillshading(cgrad, sv)
```

**Arguments**

cgrad	3D array ( nrow, ncol, 3) of unit vectors normal to surface grid cells. The output of cgrad().
sv	unit vector in the direction of the sun.

**Details**

The intensity of the sun beams falling on a surface are proportional to the cosine of the angle between the sun vector and the vector normal to the surface, which in this case is the dot product between cgrad and sv.

**Value**

A matrix of illumination intensity values. Negative values are the equivalent of self shading and are set to zero.



**Author(s)**

Javier G. Corripio

**References**

Corripio, J. G.: 2003, Vectorial algebra algorithms for calculating terrain parameters from DEMs and the position of the sun for solar radiation modelling in mountainous terrain, *International Journal of Geographical Information Science* 17(1), 1-23.

**See Also**

[cgrad](#), [insolation](#), [sunvector](#)

**Examples**

```
## Not run:
## From the "Obligatory Mathematical surface" in persp3d {rgl}
## this shows the cast shadows clearly, but there is some interference with rgl internal
## lit parameters
require(rgl)
x = seq(-10, 10, length= 300)
y = x
f = function(x,y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
z = outer(x, y, f)
z[is.na(z)] = 1
zgr = cgrad(z,2/30)
sv = normalvector(55,315)
hsh = zgr[,1]*sv[1]+zgr[,2]*sv[2]+zgr[,3]*sv[3]
hsh = (hsh+abs(hsh))/2
sh = doshade(z,sv,2/30)
hshsh = hsh*sh
hshsh = t(hshsh[nrow(hshsh):1,])
open3d()
rgl.light(viewpoint.rel = F, ambient = "#FFFFFF", diffuse = "#FFFFFF", specular = "#000000")
persp3d(x, y, z, col=grey(hshsh))

## End(Not run)

## Hillshading on mountain terrain, sun at NW, 35 degrees elevation
zipfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/dempyrenees.asc.zip",zipfile)
header = read.table(unz(zipfile,'dempyrenees.asc'),nrows=6)
dem = read.table(unz(zipfile,'dempyrenees.asc'),skip=6)
dem = as.matrix(dem)
unlink(zipfile)
cellsize = header[5,2]
sv = normalvector(55,315)
grd = cgrad(dem,cellsize)
hsh = grd[,1]*sv[1]+grd[,2]*sv[2]+grd[,3]*sv[3]
## remove negative incidence angles (self shading)
hsh = (hsh+abs(hsh))/2
sh = doshade(dem,sv,cellsize)
```

```

hshsh = hsh*sh
image(t(hshsh[nrow(sh):1,]),col=grey(1:100/100))

## Not run:
## Hillshading on mountain terrain, sun at NW, 45 degrees elevation. Using raster
sv = normalvector(45,315)
require(raster)
demfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/dempyrenees.tif",demfile)
dem = raster(demfile)
grd = cgrad(dem)
hsh = grd[,1]*sv[1]+grd[,2]*sv[2]+grd[,3]*sv[3]
## remove negative incidence angles (self shading)
hsh = (hsh+abs(hsh))/2
hsh = raster(hsh,crs=projection(dem))
extent(hsh) = extent(dem)
plot(hsh,col=grey(1:100/100),legend=FALSE)
unlink(demfile)

## End(Not run)

```

---

hourangle

*Hour angle*


---

### Description

Hour angle, internal function for solar position.

### Usage

```
hourangle(jd, longitude, timezone)
```

### Arguments

jd	Julian Day.
longitude	Longitude.
timezone	Timezone in hours, west of Greenwich is negative.

### Value

Hour angle

---

insolation	<i>Direct and diffuse solar radiation.</i>
------------	--

---

**Description**

Computes direct and diffuse solar irradiance perpendicular to the beam, for a given zenith angle, Julian Day, altitude and atmospheric conditions.

**Usage**

```
insolation(zenith, jd, height, visibility, RH, tempK, O3, alphag)
```

**Arguments**

zenith	Zenith angle in degrees.
jd	Julian Day.
height	Altitude above sea level.
visibility	Visibility [km].
RH	Relative humidity [%].
tempK	Air temperature [K].
O3	Ozone thickness [cm].
alphag	Albedo of the surrounding terrain [0 to 1].

**Details**

See <https://disc.gsfc.nasa.gov/datasets?page=1&source=AURA%200MI> for ozone data.

**Value**

Returns a two column matrix of irradiance values. The first column is direct radiation, the second is diffuse radiation.

**Author(s)**

Javier G. Corripio

**References**

- Bird, R. E. and Hulstrom, R. L. (1981a) Review, evaluation and improvements of direct irradiance models, *Trans. ASME J. Solar Energy Eng.* 103, 182-192.
- Bird, R. E. and Hulstrom, R. L. (1981b) *A simplified clear sky model for direct and diffuse insolation on horizontal surfaces*, Technical Report SERI/TR-642-761, Solar Research Institute, Golden, Colorado.
- Iqbal, M. (1983) *An Introduction to Solar Radiation*, Academic Press, Toronto.

**See Also**

[doshade](#), [hillshading](#), [sunvector](#)

**Examples**

```

insolation(30,2458656,3200,28,60,278.15,0.3,0.2)
insolation(30,JDymd(2019,6,21),3200,28,60,278.15,0.3,0.2)

# Compare measured and modelled insolation

# load data from automatic weather station in the Andes
data(meteoandes)

# Get zenith angle for every time step
meteodate = as.POSIXct(strptime(paste(meteoandes$year,meteoandes$doy,
meteoandes$hh,meteoandes$mm),format="%Y %j %H %M"))
metjd = JD(meteodate)

sunv = sunvector(metjd,-33.695,-70.0033,-3)
zenith = sunpos(sunv)[,2]

# Compute direct and diffuse beam irradiance
Idirdif = insolation(zenith,metjd,4640,90,
meteoandes$RH,meteoandes$Tair+273.15,0.3,0.55)

# modify for angle of incidence on horizontal surface (pyranometer)
cos_inc_sfc = sunv%%as.vector(normalvector(0,0)) ## or sum(sunv*normalvector(0,0))

# set to zero values with no incident light
cos_inc_sfc[cos_inc_sfc<0] = 0

# Add direct and diffuse simulated radiation on horizontal surface
Isim = Idirdif[,1] * cos_inc_sfc + Idirdif[,2]

# plot the measured insolation
plot(meteodate,meteoandes$pyra1,'l',col=2)

# add a shaded polygon corresponding to 10% accuracy in the measurements
polygon(c(meteodate, rev(meteodate)), c(meteoandes$pyra1*(1+0.1),
rev(meteoandes$pyra1*(1-0.1))),col = "#ff000033", border = NA)

# add the simulated insolation
lines(meteodate,Isim,col=4)

#check if discrepancy is due to station tilted 10 degrees north
cos_inc_sfc = sunv%%as.vector(normalvector(15,355)) ## or sum(sunv*normalvector(15,355))
cos_inc_sfc[cos_inc_sfc<0] = 0
Isim = Idirdif[,1] * cos_inc_sfc + Idirdif[,2]
plot(meteodate,meteoandes$pyra1,'l',col=2)
polygon(c(meteodate, rev(meteodate)), c(meteoandes$pyra1*(1+0.1),
rev(meteoandes$pyra1*(1-0.1))),col = "#ff000033", border = NA)

```

```

lines(meteodate,Isim,col=4)

# We measured that diffuse reflected solar radiation from the surrounding mountains
# covered in snow could be up to 10% of total incoming radiation.
# There is one hour of shadows early in the morning (not simulated)
# Add 10% diffuse reflected radiation
lines(meteodate,1.1*Isim,col=3)

## Calculate insolation on the island of La Palma, Spain on the 21.03.2013
## reduced resolution DEM from SRTM, https://www2.jpl.nasa.gov/srtm/
zipfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/demlapalma.asc.zip",zipfile)
header = read.table(unz(zipfile,'demlapalma.asc'),nrows=6)
dem = read.table(unz(zipfile,'demlapalma.asc'),skip=6)
dem = as.matrix(dem)
unlink(zipfile)
cellsize = header[5,2]
cgr = cgrad(dem,cellsize)
height = 750
visibility = 30
RH = 80
tempK = 288
tmz = 0
year = 2013
month = 3
day = 21
timeh = 12
jd = JDymd(year,month,day,hour=timeh)
Iglobal = array(0,dim=dim(dem))
deltat = 0.5
lat = 28.135
lon = -17.247
dayl = daylength(lat,lon,jd,0)
for (srs in seq(dayl[1],dayl[2],deltat)){
jd = JDymd(year,month,day,hour=srs)
sv = sunvector(jd,lat,lon,tmz)
hsh = hillshading(cgr,sv)
sh = doshade(dem,sv,cellsize)
zenith = sunpos(sv)[2]
Idirdif = insolation(zenith,jd,height,visibility,RH,tempK,0.2,0.15)
## direct radiation modified by terrain + diffuse irradiation (skyviewfactor ignored)
## values in J/m^2
Iglobal = Iglobal + (Idirdif[,1] * hsh + Idirdif[,2] )*3600*deltat
}

## dispaly results
image(t(Iglobal[nrow(Iglobal):1,]),col=grey(1:100/100))
contour(t(dem[nrow(dem):1,]),lwd=.5,col='sienna1',add=TRUE,levels=seq(0,2500,500))
contour(t(dem[nrow(dem):1,]),lwd=.25,col='sienna1',add=TRUE,levels=seq(0,2500,50),drawlabels=FALSE)

```

```

## Not run:
## The same using raster
require(raster)
demfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/demlapalma.tif",demfile)
dem = raster(demfile)
plot(dem)
cgr = cgrad(dem)
demmm = raster:::as.matrix(dem)
dl = res(dem)[1]
## Isolation at 30 min interval over the length of the day
## RH and temp would change over the dy, here we use a constant value for simplicity
height = 750
visibility = 30
RH = 80
tempK = 288
tmz = 0
year = 2013
month = 3
day = 21
timeh = 12
jd = JDymd(year,month,day,hour=timeh)
Iglobal = array(0,dim=dim(demmm))
deltat = 0.5
lat = 28.135
lon = -17.247
dayl = daylength(lat,lon,jd,0)
for (srs in seq(dayl[1],dayl[2],deltat)){
  jd = JDymd(year,month,day,hour=srs)
  sv = sunvector(jd,lat,lon,tmz)
  hsh = hillshading(cgr,sv)
  sh = doshade(demmm,sv,dl)
  zenith = sunpos(sv)[2]
  Idirdif = insolation(zenith,jd,height,visibility,RH,tempK,0.2,0.15)
  ## direct radiation modified by terrain + diffuse irradiation (skyviewfactor ignored)
  ## values in J/m^2
  Iglobal = Iglobal + (Idirdif[,1] * hsh + Idirdif[,2] )*3600*deltat
}
## rasterize to plot nicely
Iglobal = raster(Iglobal,crs=projection(dem))
extent(Iglobal) = extent(dem)
plot(Iglobal*1e-6,col=grey(1:100/100),
      legend.args = list(text=expression(paste('Insolation MJ ',m^-2)), side=4,line=2.5))
contour(dem,lwd = 0.5,col='sienna1',add=TRUE,levels=seq(0,2500,500))
contour(dem,lwd = 0.25,col='sienna1',add=TRUE,levels=seq(0,2500,50),drawlabels=FALSE)
unlink(demfile)

## End(Not run)

```

**Description**

Computes Julian Day from dates as POSIXct object.

**Usage**

```
JD(x, inverse=FALSE)
```

**Arguments**

x	POSIXct object.
inverse	Logical. If false (default) returns the Julian Days corresponding to given dates. If TRUE returns the date corresponding to input Julian days

**Details**

Class "POSIXct" represents the (signed) number of seconds since the beginning of 1970 (in the UTC timezone) as a numeric vector, and Julian Day is the number of days since January 1, 4713 BCE at noon UTC, so the Julian Day is calculated as `numeric(POSIXct)+2440587.5` days.

**Value**

Julian Day

**Note**

You may like to double check the results here:

<https://ssd.jpl.nasa.gov/tc.cgi>

To get correct values it is recommended to increase the number of digits to display: `options(digits=12)`

**Author(s)**

Javier G. Corripio

**See Also**

[JDymd](#)

**Examples**

```
JD(Sys.time())  
JD(seq(ISOdate(2019, 1, 21), ISOdate(2019, 12, 21), by='month'))
```

---

JDymd

*Julian Day from yyyy, mm, dd*

---

### Description

Computes Julian Day from a given date.

### Usage

```
JDymd(year, month, day, hour=12, minute=0, sec=0)
```

### Arguments

year	numeric year
month	1-12: number of the month.
day	1-31: day of the month.
hour	0-23: hour of the day.
minute	0-59: minutes.
sec	0-59: seconds.

### Value

Julian Day, or number of days since January 1, 4713 BCE at noon UTC.

### Warning

This simplification is only valid between 1901 and 2099. To get correct values it is recommended to increase the number of digits to display: options(digits=12)

### Note

You may like to double check the results here:  
<https://ssd.jpl.nasa.gov/tc.cgi>

### Author(s)

Javier G. Corripio

### References

Danby, J. M. Eqn. 6.16.4 in *Fundamentals of Celestial Mechanics*, 2nd ed. Richmond, VA: Willmann-Bell, p. 207, 1988.

### See Also

[JD](#)



**Examples**

```
JDymd(2019,3,20,12)

print(paste('Number of days since the beginning of the century (1/1/2001):',
  JD(Sys.time())-JDymd(2001,1,1,0)))
```

---

meteoandes	<i>Mountain meteorological data</i>
------------	-------------------------------------

---

**Description**

Meteorological data from an automatic weather station in the Central Andes of Chile.

**Usage**

```
data(meteoandes)
```

**Format**

A data frame with 1152 observations on the following 10 variables.

year Year  
doy Day of the year  
hh hour  
mm minute  
Tair Air temperature, grades centigrade  
pyra1 Incoming solar short-wave radiation  $\text{Wm}^{-2}$   
pyra2 Reflected solar short-wave radiation  $\text{Wm}^{-2}$   
windspeed Wind speed,  $\text{ms}^{-1}$   
winddir Wind direction, degrees  
RH Relative humidity %

**Source**

Measured by the author on Loma Larga Glacier, -33.6917, -70.0, 4640 m a.s.l. January 2001.

**References**

Corripio, J. G. and Purves, R. S.: 2005, Surface energy balance of high altitude glaciers in the Central Andes: the effect of snow penitentes, in C. de Jong, D. Collins and R. Ranzi (eds), *Climate and Hydrology in Mountain Areas*, Wiley, London, chapter 3, pp. 15-27.

**Examples**

```

data(meteoandes)
str(meteoandes)

# plot the 2 pyranometers measurements
# one facing up: incident insolation, one facing down: reflected insolation

meteodate = strptime(paste(meteoandes$year,meteoandes$doy,meteoandes$hh,meteoandes$mm),
format="%Y %j %H %M",tz="America/Santiago")
plot(meteodate,meteoandes$pyra1,'l',col=2,xlab='Date',
ylab=expression(paste('Solar radiation [ ',Wm^-2,' ]')),
main='Insolation at Loma Larga glacier')
lines(meteodate,meteoandes$pyra2,col=4)

```

---

normalvector	<i>Vector normal to surface</i>
--------------	---------------------------------

---

**Description**

Calculates a unit vector normal to a surface defined by slope inclination and slope orientation.

**Usage**

```
normalvector(slope, aspect)
```

**Arguments**

slope	slope inclination in degrees.
aspect	slope orientation in degrees.

**Value**

Vector normal to surface, matrix of [x, y, z] coordinates.

**Author(s)**

Javier G. Corripio

**References**

Corripio, J. G.: 2003, Vectorial algebra algorithms for calculating terrain parameters from DEMs and the position of the sun for solar radiation modelling in mountainous terrain, *International Journal of Geographical Information Science* 17(1), 1-23.

**Examples**

```
# horizontal surface
normalvector(0,0)

# surface 45 degrees south
normalvector(45,180)

# range of surfaces 45 degrees E,SE,S,SW,W
normalvector(45,seq(90,270,45))

# Angle of incidence of the sun on a tilted surface 15 degrees south on March at Davos
jd = JD(seq(ISOdate(2019,3,20,0),ISOdate(2019,3,20,23),by="hour"))
degrees(acos(sunvector(jd,46.813,9.844,1) %*% as.vector(normalvector(15,180))))
```

---

p2rho

*Air pressure to density*


---

**Description**

Calculates air density for a given pressure, temperature and relative humidity.

**Usage**

```
p2rho(Pz, TempK, RH)
```

**Arguments**

Pz	Air pressure in hPa
TempK	Air temperature in K
RH	Relative humidity (%)

**Value**

Air density ( $\text{kgm}^{-3}$ )

**Author(s)**

Javier G. Corripio

**References**

Brutsaert, W.: 1982, *Evaporation into the atmosphere : theory, history, and applications*, Reidel, Dordrecht. 1984 edition.

**See Also**

[wvapsat](#)

**Examples**

```
p2rho(1013, 288, 60)

# plot density vertical profile

z = seq(0, 10000, 100)
press = z2p(z)
Tair = 288-0.0065*z
par(mar=c(5.1, 4.5, 4.1, 2.1)) # increase left margin for label
plot(z,p2rho(press,Tair,50),ty='l',xlab='Altitude',
ylab=expression(paste('Air density [ kg ', m^-3, ' ]')))
```

---

radians

*Degrees to radians*

---

**Description**

Accessory function to transform degrees into radians.

**Usage**

```
radians(degree)
```

**Arguments**

degree            Angle in degrees and decimal fraction.

**Value**

Angle in radians.

**See Also**

[degrees](#)

**Examples**

```
radians(seq(0,360,90))
```

---

rh2sh	<i>Relative humidity to specific humidity</i>
-------	---

---

**Description**

Computes specific humidity from given relative humidity, temperature and pressure.

**Usage**

```
rh2sh(RH, tempk, Pz, ice)
```

**Arguments**

RH	Relative humidity (%).
tempk	Air temperature in K
Pz	Air pressure in hPa
ice	Whether over water or ice surface (0,1).

**Value**

Specific humidity (kg/kg).

**Author(s)**

Javier G. Corripio

**References**

Brutsaert, W.: 1982, *Evaporation into the atmosphere : theory, history, and applications*, Reidel, Dordrecht. 1984 edition.

**See Also**

[wvapsat](#)

**Examples**

```
plot(250:300-273.15,rh2sh(50, 250:300, 1013, 0),xlab='Temperature [C]',  
ylab='specific humidity',  
main='Specific humidity for RH=0.5 and varying temperature')
```

---

slope *Slope of grid cells in a DEM*

---

### Description

Calculates the slope of every grid cell in a digital elevation model (DEM) from the output of `cgrad`, which is a set of unit vectors normal to every grid cell.

### Usage

```
slope(cgrad, degrees = FALSE)
```

### Arguments

<code>cgrad</code>	A 3D array of dimensions <code>nrow(dem)</code> , <code>ncol(dem)</code> , 3, where the third dimensions are the x, y z component of the unit vectors normal to the surface of the DEM grid cells.
<code>degrees</code>	Logical. If FALSE, returns radians, if TRUE, returns degrees.

### Value

A matrix of slope values for all grid cells.

### See Also

[aspect](#), [cgrad](#)

### Examples

```
# Calculate the slope of a rough mountain area in the pyrines
zipfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/dempyrenees.asc.zip", zipfile)
header = read.table(unz(zipfile, 'dempyrenees.asc'), nrows=6)
dem = read.table(unz(zipfile, 'dempyrenees.asc'), skip=6)
dem = as.matrix(dem)
unlink(zipfile)
cellsize = header[5,2]
slopedem = slope(cgrad(dem, cellsize), degrees=TRUE)
image(t(slopedem[nrow(slopedem):1,]), col=grey(100:1/100))
```

```
## similar but using raster
## Not run:
require(raster)
demfile = tempfile()
download.file("https://meteoexploration.com/R/insol/data/dempyrenees.tif", demfile)
dem = raster(demfile)
slopedem = slope(cgrad(dem), degrees=TRUE)
slopedem = raster(slopedem, crs=projection(dem))
```

```
extent(slopedem) = extent(dem)
plot(slopedem,col = grey(100:1/100))
unlink(demfile)

## End(Not run)
```

---

sunpos

*Azimuth and zenith of the Sun*

---

### Description

Returns a matrix of azimuth and zenith angles of the sun given the unit vectors from the observer to the direction of the sun.

### Usage

```
sunpos(sunv)
```

### Arguments

sunv                    coordinates x, y, z of the unit vector in the direction of the sun.

### Value

A matrix of azimuth and zenith angles.

### Author(s)

Javier G. Corripio

### References

Corripio, J. G.: 2003, Vectorial algebra algorithms for calculating terrain parameters from DEMs and the position of the sun for solar radiation modelling in mountainous terrain, *International Journal of Geographical Information Science* 17(1), 1-23.

### See Also

[sunvector](#),

### Examples

```
## Julian Day hourly intervals at spring equinox
jd = JD(seq(ISOdate(2019,3,20,0),ISOdate(2019,3,20,23),by="hour"))

## sun position
sp = sunpos(sunvector(jd,46.813,9.844,1))

## daylight zenith<=90
```

```

sp = sp[which(sp[,2]<=90),]

## Plot the apparent solar path at Davos on the spring equinox
ramp = colorRamp(c("red", "orange", "yellow"))
crmp = c(rgb(ramp(seq(1/6,1,1/6)), max = 255),rgb(ramp(seq(1,1/6,-1/6)), max = 255))
plot(sp[,1],90-sp[,2],xlab='Azimuth',
ylab='Elevation',main='Apparent solar path at Davos on the spring equinox',
pch=20,col=crmp,cex=(300-sp[,2])/90)

## Not run:
require(plotrix)
polar.plot(90-sp[,2],sp[,1],start=90,clockwise=TRUE,rp.type='s',
point.symbols=20,point.col=2,cex=2,radial.lim=c(0,90),
main='Apparent solar path at Davos on the spring equinox')

## End(Not run)

```

---

sunr

*Earth radius vector*


---

### Description

Calculates the Earth radius vector.

### Usage

```
sunr(jd)
```

### Arguments

jd                    Julian Day

### Value

Earth Radius Vector in Astronomical Units (AU). This is used to modify the solar constant as a function of the earth-sun distance.

### Author(s)

Javier G. Corripio

### References

<https://www.esrl.noaa.gov/gmd/grad/solcalc/calcdetails.html>

Meeus, J. 1999. *Astronomical Algorithms*. Willmann-Bell, Richmond, Virginia, USA.

Reda, I. and Andreas, A. 2003. *Solar Position Algorithm for Solar Radiation Applications*. 55 pp.; NREL Report No. TP-560-34302, Revised January 2008. <https://www.nrel.gov/docs/fy08osti/34302.pdf>



## Examples

```
# plot the variation of the earth radius vector over the next year
days_nexty = seq(Sys.time(),Sys.time()+86400*365,by='day')
plot(days_nexty,sunr(JD(days_nexty)),xlab='Date',ylab='Earth Radius Vector [AU]')
abline(h=1,col=8)
```

---

sunvector

*Solar vector*

---

## Description

Calculates a unit vector in the direction of the sun from the observer position.

## Usage

```
sunvector(jd, latitude, longitude, timezone)
```

## Arguments

jd	Julian Day and decimal fraction.
latitude	Latitude of observer in degrees and decimal fraction.
longitude	Longitude of observer in degrees and decimal fraction.
timezone	Time zone, west is negative.

## Details

To calculate the sunvector to the nearest hour, give the Julian Day with a precision better than 1/24; to approximate it to the nearest minute use decimal fractions smaller than 1/(24\*60), and so on.

## Value

3 column matrix with the x, y, z coordinates of the sun vector.

## Author(s)

Javier G. Corripio

## References

Corripio, J. G.: 2003, Vectorial algebra algorithms for calculating terrain parameters from DEMs and the position of the sun for solar radiation modelling in mountainous terrain, *International Journal of Geographical Information Science* 17(1), 1-23.

## See Also

[sunpos](#)

### Examples

```
# Current solar vector at Greenwich observatory
sunvector(JD(Sys.time()),51.4778,-0.0017,0)

juneday = JD(seq(ISOdate(2019,6,21,0),ISOdate(2019,6,21,23,30),by='30 min'))
## Not run:
# Path of the sun over Greenwich in summer
require(scatterplot3d)
scatterplot3d(sunvector(juneday,51.4778,-0.0017,0),
ylim=c(-1,1),zlim=c(0,1),pch=8,color='orange')

## End(Not run)
# print values
options(digits=12) # make sure decimals are printed
sunvector(juneday,51.4778,-0.0017,0)
```

---

wvapsat

*Saturation pressure of water vapor*

---

### Description

Computes the saturation pressure of water vapour in air over water or ice.

### Usage

```
wvapsat(tempk, ice)
```

### Arguments

tempk	Air temperature [K].
ice	Over water or ice [0,1].

### Value

Partial pressure of water vapour [hPa].

### Author(s)

Javier G. Corripio

### References

Lowe, P. R.: 1977, An approximating polynomial for the computation of saturation vapor pressure, *Journal of Applied Meteorology* 16, 100-103.

**Examples**

```
## Plot the differences saturation pressure over water and over ice
plot(wvapsat(250:300), xlab='Temperature', ylab='saturation vapour pressure [hPa]')

Tair = 223:273
plot(Tair,wvapsat(Tair),ty='l',lwd=2,col=4,xlab='Temperature',
ylab='saturation vapour pressure [hPa]')
lines(Tair,wvapsat(Tair,ice=1),col=8)
legend('topleft',c('saturation pressure over water','saturation pressure over ice'),
col=c(4,8),lwd=2)
```

z2p

*Altitude to pressure***Description**

Computes air pressure for a given altitude according to the standard atmosphere.

**Usage**

```
z2p(z, P0 = 101325, T0 = 288.15)
```

**Arguments**

z	altitude above sea level in metres [0:10000].
P0	Pressure at sea level.
T0	Temperature at sea level.

**Value**

Pressure in hPa.

**Author(s)**

Javier G. Corripio

**References**

U.S. NOAA: 1976, *U.S. standard atmosphere, 1976*, NOAA-S/T; 76-1562, U.S. National Oceanic and Atmospheric Administration, National Aeronautics and Space Administration, United States Air Force, Washington. 227 pp.

**Examples**

```
# Plot pressure from sea level to the top of Mt. Everest
plot(z<-0:8848,z2p(z),'l',xlab='Altitude [m]',ylab='Pressure [hPa]')
```

# Index

- \* **datasets**
  - meteoandes, [25](#)
- \* **package**
  - insol-package, [2](#)
- as.POSIXlt, [13](#)
- aspect, [3](#), [5](#), [30](#)
- cgrad, [4](#), [4](#), [17](#), [30](#)
- daydoy, [7](#)
- daydoy,missing-method (daydoy), [7](#)
- daydoy,numeric-method (daydoy), [7](#)
- daydoy,POSIXct-method (daydoy), [7](#)
- daylength, [8](#)
- declination, [8](#), [9](#)
- degrees, [10](#), [28](#)
- doshade, [11](#), [20](#)
- doyday, [13](#)
- eqtime, [8](#), [14](#)
- GCdistance, [15](#)
- hillshading, [16](#), [20](#)
- hourangle, [18](#)
- insol (insol-package), [2](#)
- insol-package, [2](#)
- insolation, [17](#), [19](#)
- ISOdate, [7](#)
- JD, [22](#), [24](#)
- JDymd, [23](#), [24](#)
- meteoandes, [25](#)
- normalvector, [26](#)
- p2rho, [27](#)
- radians, [10](#), [28](#)
- rh2sh, [29](#)
- slope, [4](#), [5](#), [30](#)
- sunpos, [31](#), [33](#)
- sunr, [32](#)
- sunvector, [17](#), [20](#), [31](#), [33](#)
- wvapsat, [27](#), [29](#), [34](#)
- z2p, [35](#)