How to use *skyscapeR*

Fabio Silva

2017-10-17

skyscapeR is a open source R package for data reduction, visualization and analysis in skyscape archaeology, archaeoastronomy and cultural astronomy. It is intended to become a fully-fledged, transparent and peer-reviewed package offering a robust set of quantitative methods while retaining simplicity of use.

It includes functions to transform horizontal (Az/Alt) to equatorial (Dec/RA) coordinates, create or download horizon profiles, plot them and overlay them with visible paths of common celestial objects/events for prehistoric or historic periods, as well as functions to test statistical significance and estimate stellar visibility and seasonality. It also includes the ability to easily construct azimuth polar plots and declination curvigrams. Future versions will add data reduction and likelihood-based model selection facilities, as well as an easy-to-use Graphical User Interface.

1. First Steps

Like all vignettes for *R* packages this is neither intended for those unexperienced in *R* nor a fully-fledged and detailed manual for *skyscapeR*. Neither will it be understandeable to those unfamiliar with the terminology and methodologies of cultural astronomy, archaeoastronomy or skyscape archaeology.

This document is an entry-point to the package's main functions and sets out workflows of what you can do with it. More detail, as well as more functions, can be found in the package's manual pages, acessible from within *R*.

If you are new to *R* then I recommend the online free short course Code School's TryR. For those who are already familiar with R I apologise in advance for any patronising bits (such as the next two sections). With some basic *R* skills, guidance from this vignette and autonomous exploration of the package's manual pages, anyone with prior training in skyscape archaeology can become a master *skyscapeR*.

Installation

The package requires that the latest version of *R* is installed first. See the R Project website for details. Also suggested is the installation of RStudio. With *R* installed, install package *devtools* by running:

```
install.packages('devtools')
```

The latest version of *skyscapeR*, and associated requirements, can then be downloaded and installed by running:

```
devtools::install_github('f-silva-archaeo/skyscapeR')
```

Upon succesful completion you should see a line saying `* DONE (skyscapeR)`. If this doesn't happen then there should be an error message explaining what went wrong.

Initialization

Every time you want to use the package within R it needs to be loaded. For this you need to type:

```
library(skyscapeR)
```

Datasets and help

The current version of *skyscapeR* comes with a couple of measurment datasets that can be used for learning and testing the package. These can be loaded by typing:

```
data(RugglesRSC)
```

or

```
data(RugglesCKR)
```

If you want to know more about these datasets then use the helpful `?` command which opens up the help page related to whatever you are asking help for. In this case, if you want to know more about the *RugglesRSC* dataset you can type in the console

```
?RugglesRSC
```

This opens the manual page for the dataset, or function. In *RStudio* this opens on the bottom-right pane by default.

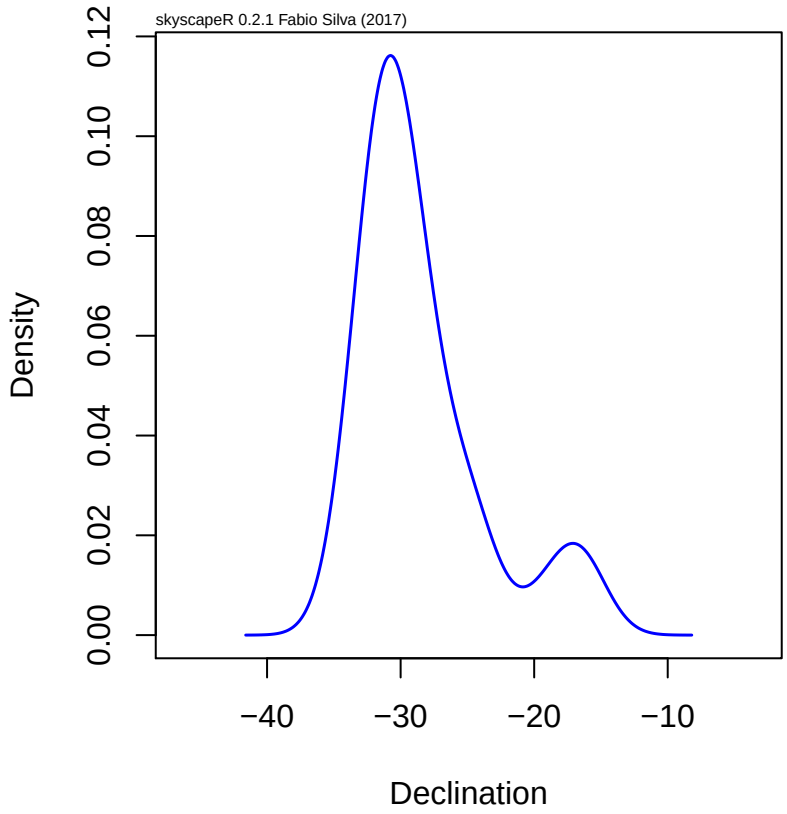
2. Hello, Curvigram!

Let's start by creating your first curvigram with *skyscapeR*. The function for this is aptly called `curvigram()`. Go ahead and do `?curvigram` right now to learn more about it, including an example at the bottom of the manual page, which we will now do:

```
data(RugglesRSC)
curv <- curvigram(RugglesRSC$Dec, 2)
```

This creates a curvigram based on the declination data in the *RugglesRSC* dataset, and using an uncertainty of 2^o for all measurements. You can visualize it by typing:

```
plotCurv(curv)
```



Be sure to check `?curvigram` and `?plotCurv` to see what other options are available.

Adding celestial objects to a curvigram

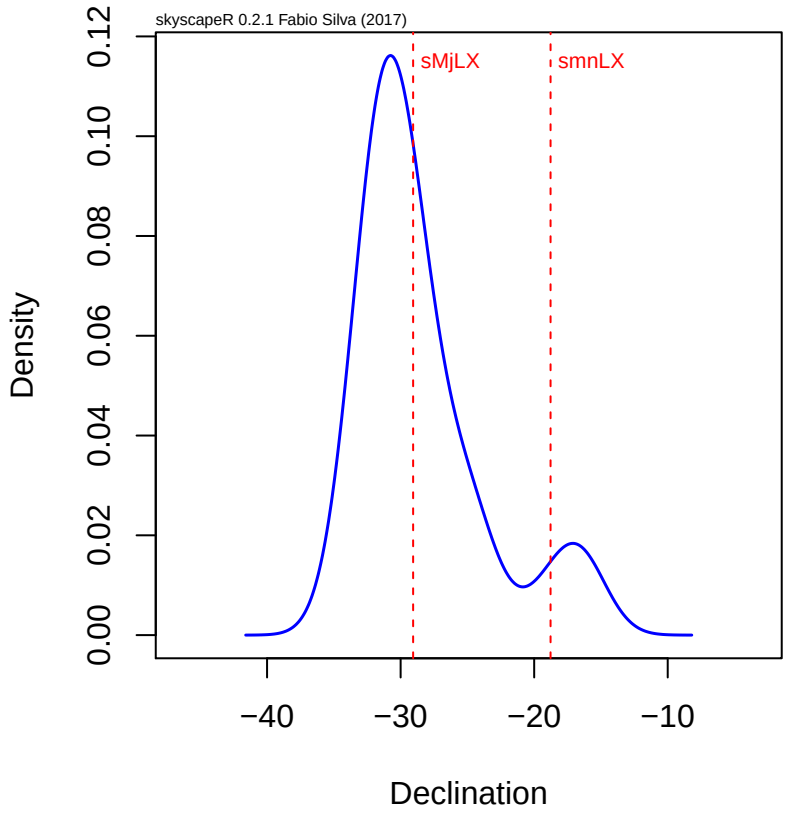
Let's add some celestial targets to the curvigram on order to compare them with those frequency peaks. We need to create a *skyscapeR.object* first. This is done with `sky.objects()`:

```
lunar <- sky.objects('moon', epoch=-2000, col='red', lty=2)
```

```
## LA04 data loaded
```

This creates an object that will include all standard lunar targets (currently only the lunar extremes), set for the year 1999 BCE (see section on epochs below), and these will be drawn in red colour with line type (lty) two, which is to say dashed (check out other types and options in `?par`). Then redo the curvigram with this object:

```
plotCurv(curv, lunar)
```



You can now see the southern major lunar extreme (sMjLX) and the southern minor lunar extreme (smnLX) declinations for the year 1999 BCE on the same plot.

3. Handling and Visualizing Fieldwork Data

skyscapeR has some built-in function to automatically process fieldwork data. Although the data reduction process can be done manually using other functions (such as `az2dec()`, `mag.dec()` or `sunAz()`) the entire process has been automated by using the `reduct.` functions.

Data reduction of compass measurements

Imagine that you've just returned from a day of fieldwork surveying a set of archaeological structures for their orientation with your compass and clinometer. You can input your measurements into R, along with the georeferences and measurement date, and use `reduct.compass()` to process it.

```
georef <- rbind(c(35.1,-7.1),c(35.1,-7),c(35.2,-7.1),c(35.1,-7.3))    # GPS data
azimuths <- c(93, 108, 105, 98)    # Compass measurements
altitudes <- c(2, 1.5, 0.5, 1)    # Clinometer measurements

data <- reduct.compass(loc=georef, mag.az=azimuths, date="2017/06/13", alt=altitudes)
```

```
## Altitude values found. Calculating declination...
```

```
data
```

```
##   Latitude Longitude Magnetic.Az       Date   Mag.Dec True.Az Altitude
## 1     35.1      35.1          93 2017/06/13   5.00640  98.006      2.0
## 2     35.2      35.1         108 2017/06/13   5.02004 113.020      1.5
```

```
## 3     -7.1      -7.0         105 2017/06/13 -11.47167  93.528      0.5
## 4     -7.1      -7.3          98 2017/06/13 -11.61256  86.387      1.0
##   Declination
## 1      -5.561
## 2     -17.960
## 3      -2.873
## 4       3.298
```

This has: (1) automatically retrieved the value of magnetic declinations for your locations and dates; (2) corrected azimuths to true; and (3) calculated the corresponding declination. The resulting *data.frame* object can be easily exported into a file for safe keeping using the standard *R* utils (such as `write.csv()`, for example).

If the altitude component is not included then the function will only calculate the true azimuth:

```
data2 <- reduct.compass(loc=georef, mag.az=azimuths, date="2017/06/13")
```

```
## No altitude values or horizon profile found. Declination values were not calculated.
```

```
data2
```

```
##   Latitude Longitude Magnetic.Az       Date   Mag.Dec True.Az
## 1     35.1      35.1          93 2017/06/13   5.00640  98.006
## 2     35.2      35.1         108 2017/06/13   5.02004 113.020
## 3     -7.1      -7.0         105 2017/06/13 -11.47167  93.528
## 4     -7.1      -7.3          98 2017/06/13 -11.61256  86.387
```

In such cases, if a horizon profile is given, the altitude can be automatically retrieved from it. See section on *Dealing with Horizons* below for examples.

Data reduction of theodolite measurements

On the other hand, if fieldwork consisted on theodolite/total station measurements using the sun-sight technique, the azimuth still need to be processed. For such precise measurements we recommend using function `ten()` to convert from deg-arcmin-arcsec to decimal point degree as follows:

```
georef <- c( ten(35,50,37.8), ten(14,34,6.4) )    # GPS data
az <- c( ten(298,24,10), ten(302,20,40))    # Theodolite H measurements
alt <- c( ten(1,32,10), ten(0,2,27))    # Theodolite V measurements
az.sun <- ten(327,29,50)    # The azimuth of the sun as measured at time
date <- "2016/02/20"
time <- "11:07:17"    # Time the sun measurement was taken
timezone <- "Europe/Malta"    # Timezone corresponding to time above

data <- reduct.theodolite(loc=georef, az, date, time, timezone, az.sun, alt)
```

```
## Altitude values found. Calculating declination...
```

```
data
```

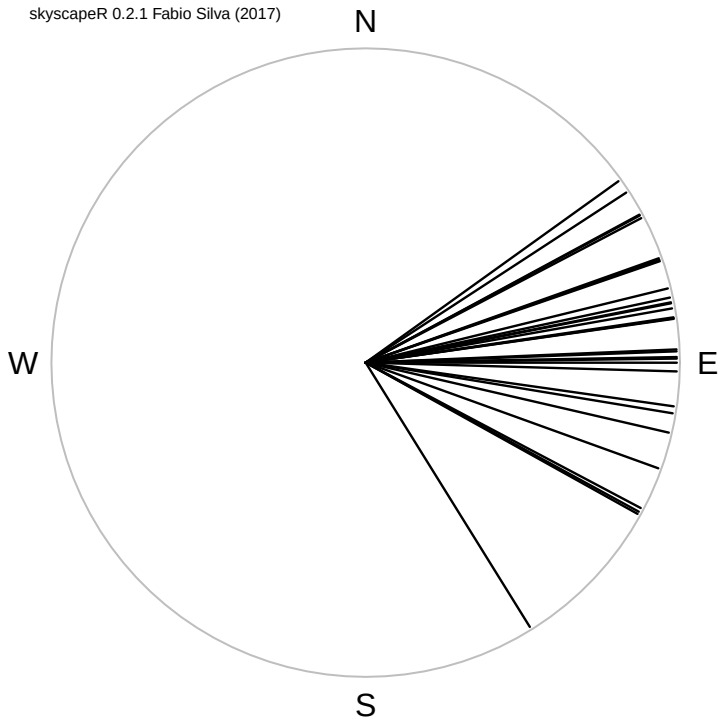
```
##   Latitude Longitude Uncorrected.Az           Date.Time   Sun.Az
## 1 35.84383  14.56844       298.4028 2016-02-20 11:07:17 157.7928
## 2 35.84383  14.56844       302.3444 2016-02-20 11:07:17 157.7928
##   Corrected.Az   Altitude Declination
## 1     128.6983 1.53611111   -29.64171
## 2     132.6400 0.04083333   -33.67346
```

Similarly to `reduct.compass()` if a horizon profile is given, the altitude can be automatically retrieved from it.

Plotting azimuths

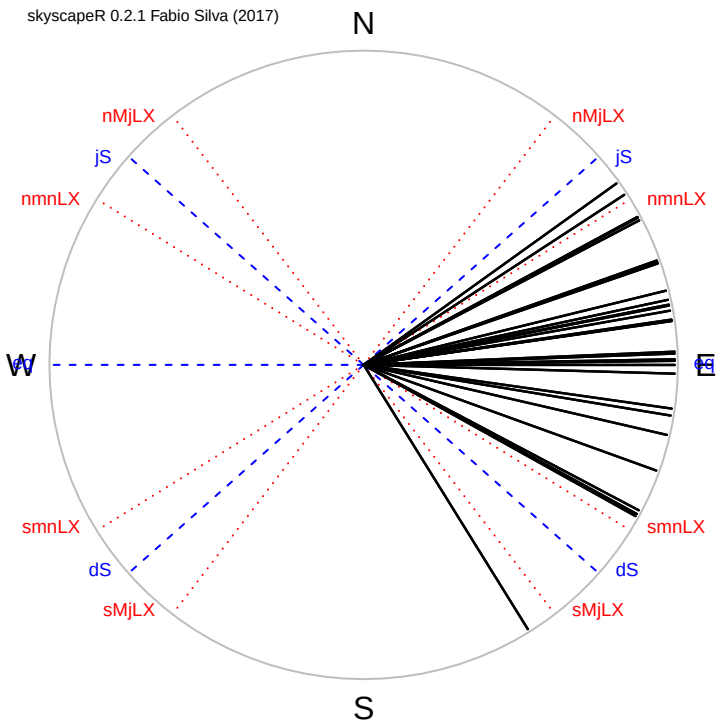
In general, the use of azimuths for analysis and visualization in *skyscapeR* is deprecated since azimuths are location-specific. It is preferable to convert all measurements to declinations and work with equatorial coordinates (see section below). However *it_skyscapeR* does include a much requested function to create a polar plot of azimuth values. The function is `plotAz()`:

```
az <- rnorm(30, 85, 20)     # This creates 30 random azimuths
plotAz(az)
```



You can use the same *skyscapeR.object* for this plot, but then you need to specify a single location (since azimuths are location-specific). At the moment the horizon altitude is assumed to be 0^{o} and flat as well.

```
sunandmoon <- sky.objects(c('sun','moon'), epoch=-4000, col=c('blue','red'), lty=c(2,3))
plotAz(az, obj=sunandmoon, loc=c(52,0))
```



Converting azimuth to declination

If you are looking for a mere horizontal to equatorial coordinate conversion, without all the extra automation that the `reduct.` functions provide you can use the `az2dec()` function:

```
dec <- az2dec(az, loc= c(35,-7), alt=0)
dec
```

```
##  [1] -23.8247286  -8.0032363   0.5116888  15.6050562 -44.5509754
##  [6]  28.1565521 -22.8650761   0.2802345  22.0033539  22.4374558
## [11]   6.3636637  10.9280152   9.5278326  -1.6556674 -23.4685873
## [16]   1.6645206  15.3600433  -6.9429693  15.1465993  22.5687721
## [21]   6.5188491 -10.9571954 -16.4961599   8.7986807   8.6579677
## [26]   6.2910556   1.3727137  -0.3449777   7.8716991  26.2295262
```

4. Dealing with Horizon Profiles

One of *skyscapeR*'s main missions is to handle horizon profile data. This is done through the creation of a *skyscapeR.hor* object which can then be plugged in to several functions of this package not only for automation, but also for the purposes of visualizing the visible paths of celestial objects.

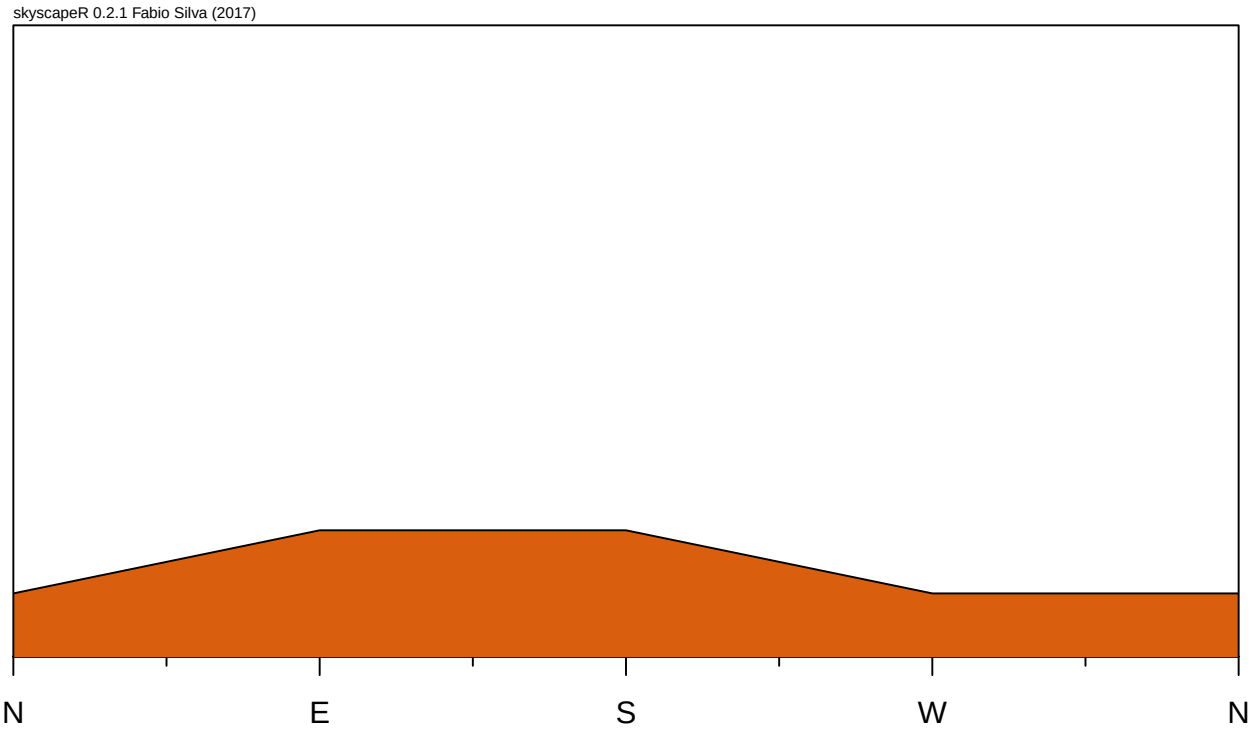
Creating or downloading profiles

If you have azimuth and altitude data a profile can be constructed as in the following example:

```
az <- c(0,90,180,270,360)
alt <- c(0,5,5,0,0)
georef = c(40.1, -8)
hor <- createHor(az, alt, loc=georef, name= 'Horizon Profile 1')
```

This can be visualized by simply typing:

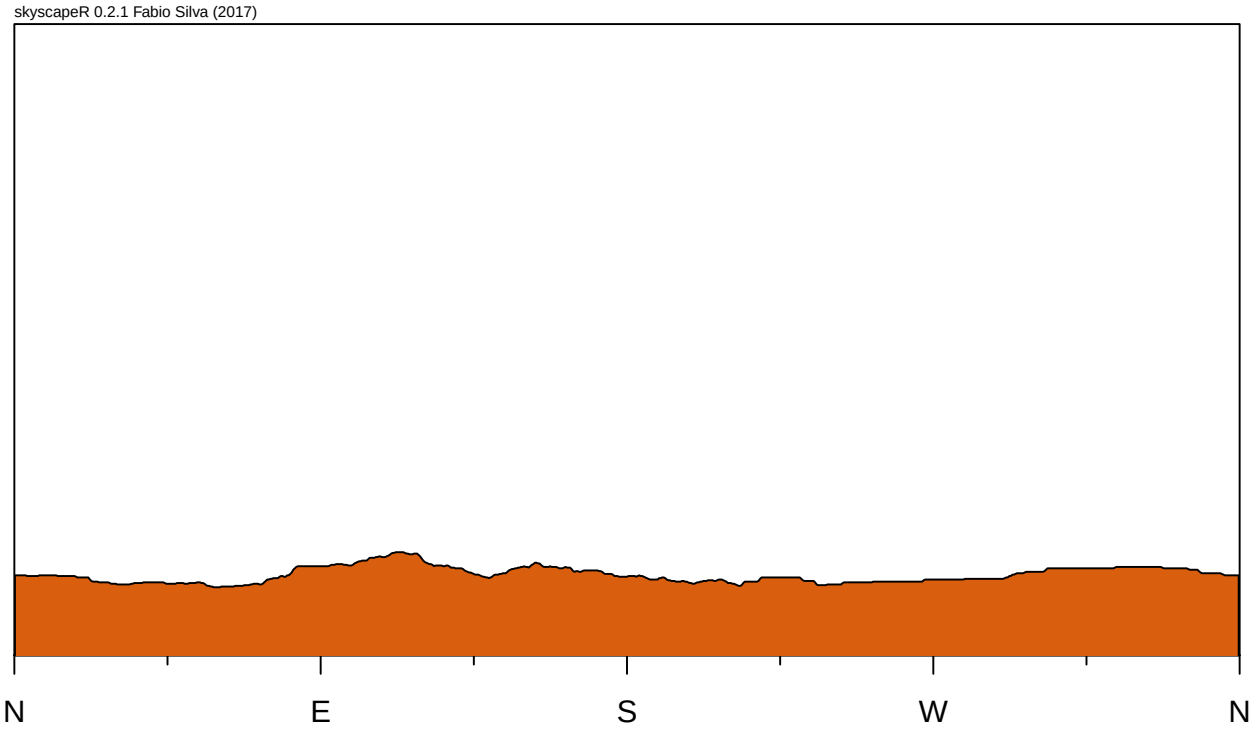
```
plotHor(hor)
```



Not the prettiest horizon you've ever seen, but that's what can be interpolated from the five datapoints given...

A prettier horizon profile can be downlaoded from the HeyWhatsThat (HWT) website. These profiles are based on the SRTM digital elevation models and, therefore are not always trustworthy (especially so if the horizon is near). Nevertheless they can be a great help for situations when a horizon altitude is impossible to measure on site. To do this, first create a horizon profile at the HWT website, then save the 8-digit ID code that is as the end of the permanent link given by HWT. For example, if the link is `https://www.heywhatsthat.com/?view=NML6GMSX`, save the bit after `view=` and use function `download.HWT()`:

```
hor <- download.HWT('NML6GMSX')
plotHor(hor)
```



Exporting profiles to *Stellarium*

One can then export these profiles into a format that *Stellarium* recognises:

```
exportHor(hor, name='Test', description='Test horizon export to Stellarium')
```

This creates a zip-file ready to be imported into *Stellarium*'s landscape feature.

Automating horizon altitude retrieval

With a horizon profile set up (or even many), you can retrieve the horizon altitude for any azimuth value by doing:

```
hor2alt(hor, az=90)
```

```
## [1] 2.06
```

```
hor2alt(hor, az=110)
```

```
## [1] 2.91
```

This automation can also be used in the data reduction functions by using a *skyscapeR.horizon* object rather than simple georeferences:

```
data <- reduct.compass(loc=hor, mag.az=azimuths, date="2017/06/13")
```

```
## Horizon profile found. Obtaining altitude values and calculating declination...
```

```
data
```

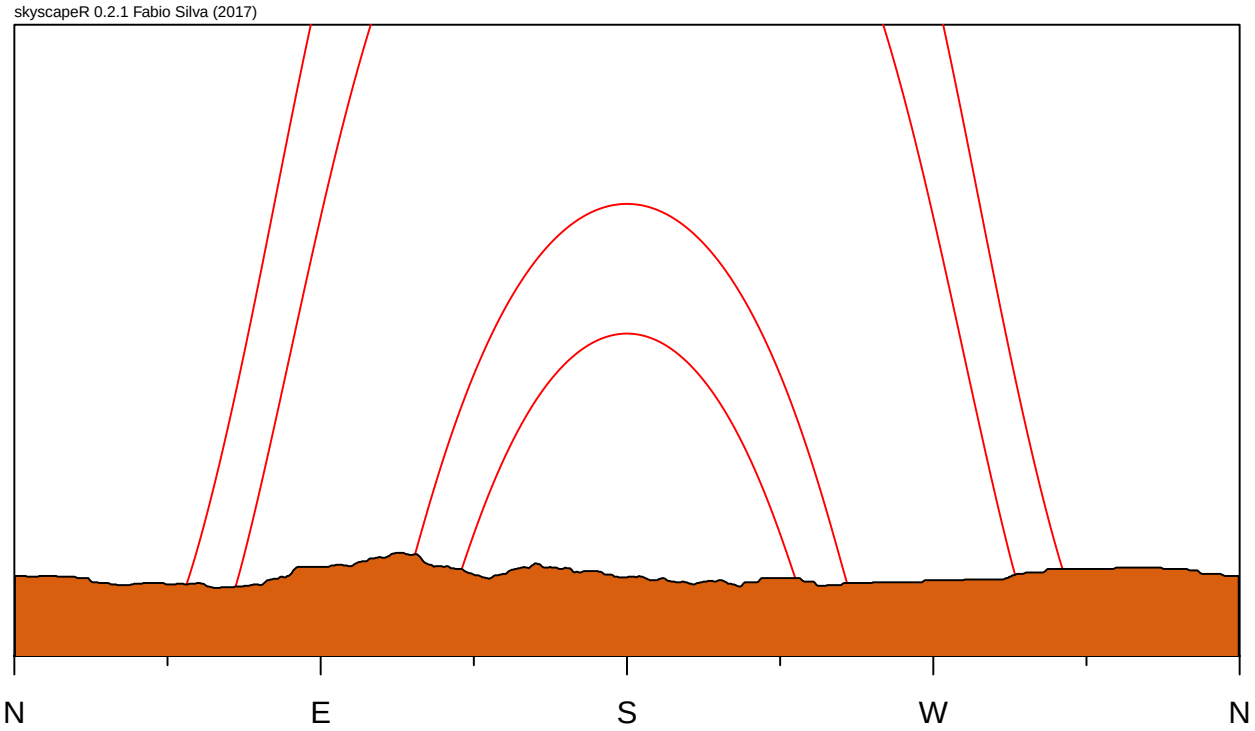
```
##   Latitude Longitude Magnetic.Az       Date   Mag.Dec True.Az Altitude
## 1 40.44385 -7.938178          93 2017/06/13 -2.118487  90.882     2.06
## 2 40.44385 -7.938178         108 2017/06/13 -2.118487 105.882     2.74
## 3 40.44385 -7.938178         105 2017/06/13 -2.118487 102.882     2.50
```

```
## 4 40.44385 -7.938178          98 2017/06/13 -2.118487  95.882     2.22
##   Declination
## 1       0.471
## 2     -10.366
## 3      -8.296
## 4      -3.215
```

Visualizing celestial object paths

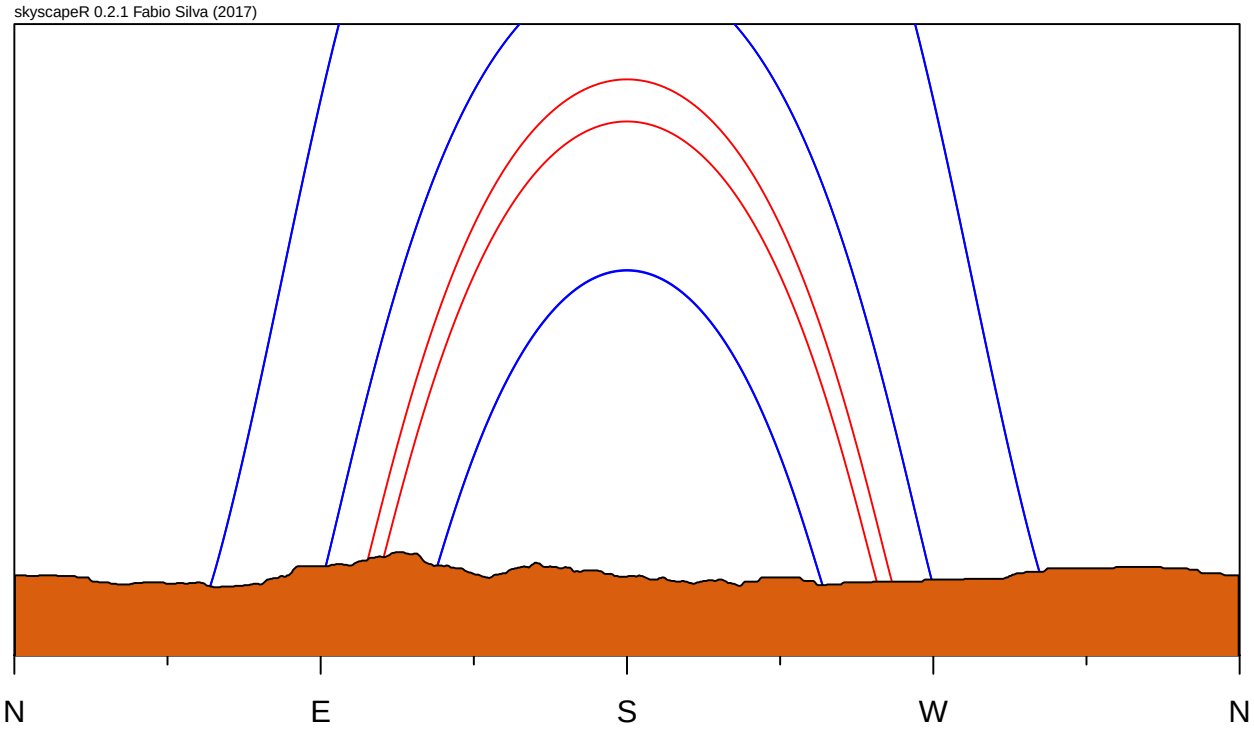
`plotHor()` can be used to display the visible paths of celestial objects chosen with `sky.objects()`. To use the one we created before for the curvigram just type:

```
plotHor(hor, obj=lunar)
```



Or create a new one, in this case using an epoch range for cases where we might have uncertainty in the age of the site, and including both solar and a stellar target:

```
aux <- sky.objects(names=c('sun', 'Aldebaran'), epoch=c(-4300,-3700), col=c('blue', 'red'))
plotHor(hor, obj=aux)
```



Stars

skyscapeR includes data on the brightest xx stars in the night sky. The data table is accessible via `data(stars)`. To pick a particular stars, maybe to see what declination it has, or had, you can do as follows:

```
ss <- star('Sirius')
ss
```

```
## $name
## [1] "Sirius"
##
## $constellation
## [1] "Cma"
##
## $colour
## [1] "White"
##
## $app.mag
## [1] -1.46
##
## $ra
## [1] 101.2885
##
## $dec
## [1] -16.71314
##
## $proper.motion
## [1]  -546.01 -1223.08
##
## $epoch
```

```
## [1] "J2000.0"
##
## attr(,"class")
## [1] "skyscapeR.star"
```

This shows the information for Sirius which is now loaded into object `ss`. Note the epoch the data is output, *J2000*, by default. To get coordinates for other epochs just do:

```
ss <- star('Sirius', year=-4000)
ss$dec
```

```
## [1] -26.57285
```

Star phases and seasonality

To estimate the phases, events and/or seasonality of stars the function `star.phases()` can be used. It works as follows, for the location of Cairo, an epoch of -3000, and a horizon altitude of 2º:

```
sp <- star.phases('Sirius', -3000, loc=c(30.0,31.2), alt.hor=2)
```

```
## Running calculations. This may take a while...
```

One can then check the star's phase, event date-range and seasons by typing:

```
sp$phase
```

```
## [1] "Arising and Lying Hidden"
```

```
sp$events
```

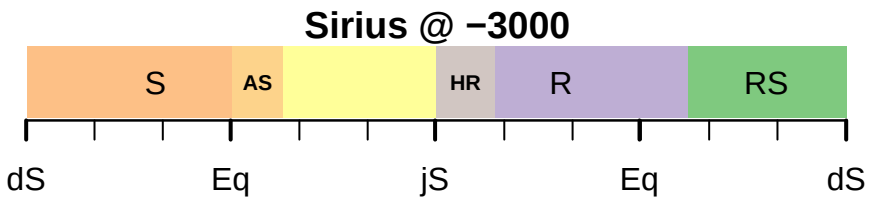
```
##                     [,1]
## Achronycal Setting "April 2 - April 24"
## Heliacal Rising    "July 2 - July 27"
```

```
sp$seasons
```

```
##                          [,1]
## Rising                   "July 2 - October 21"
## Rising and Setting       "October 20 - December 31"
## Setting                  "January 1 - April 24"
## Arising and Lying Hidden "April 25 - July 1"
```

The range of dates for events is a range of dates, given the parameters, where it is possible to observe the star and recognise the observation as the said event (note in particular the meaning of parameter `alt.rs` in the manual page for `star.phases()`). These can be visualized using:

```
plotPhases(sp)
```



Significance Testing for Curvigrams

You can test the statistical significance of an empirical curvigram by comparing it with the expectattion of a given null hypothesis. This technique can be used to output a *p-value* (either 1-tailed or 2-tailed) which is an establsihed measure of significance.

skyscapeR comes with a limited set of built-in null hypothesis, namely that of a random azimuthal orientation (`nh.Uniform()`), a random solar orientation (`nh.SolarRange()`), a random lunar orientation (`nh.LunarRange()`) and a random orientation to the Summer Full Moon (`nh.SummerFM()`). To demonstrate significance testing we will again use the Recumbent Stone Circle data. But first, one needs to choose one's null hypothesis. As usual, the help pages are essential to understand what parameter are required.

```
nullhyp <- nh.Uniform(c(57,2), alt=0)
```

Then one uses `sigTest()` to run the significance testing routine. This can take a while depending on your machine's resources. If it takes too long, try lowering the *nsims* parameter (though this brings a cost of resolution, see the manual page for this function).

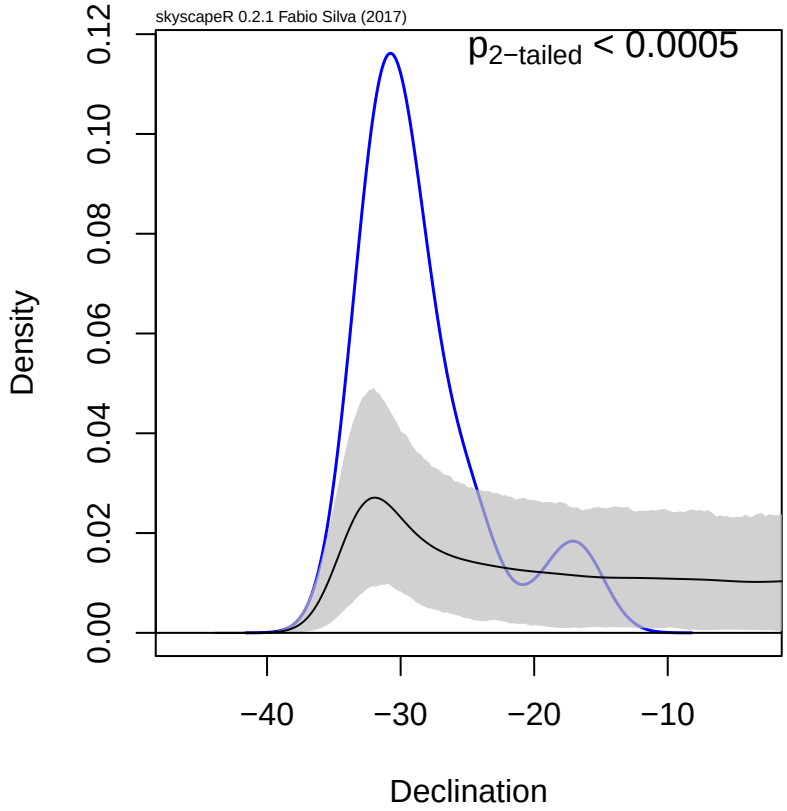
```
sg <- sigTest(curv, nullhyp)
```

```
## Running 2000 simulations. This may take a while...
```

```
## Performing a 2-tailed test at the 95% significance level.
```

One can then plot the curvigram again, but now with the results of the significance testing displayed. This adds the expectation around the null hypothesis (the grey shaded area) as well as the estimated overall p-value.

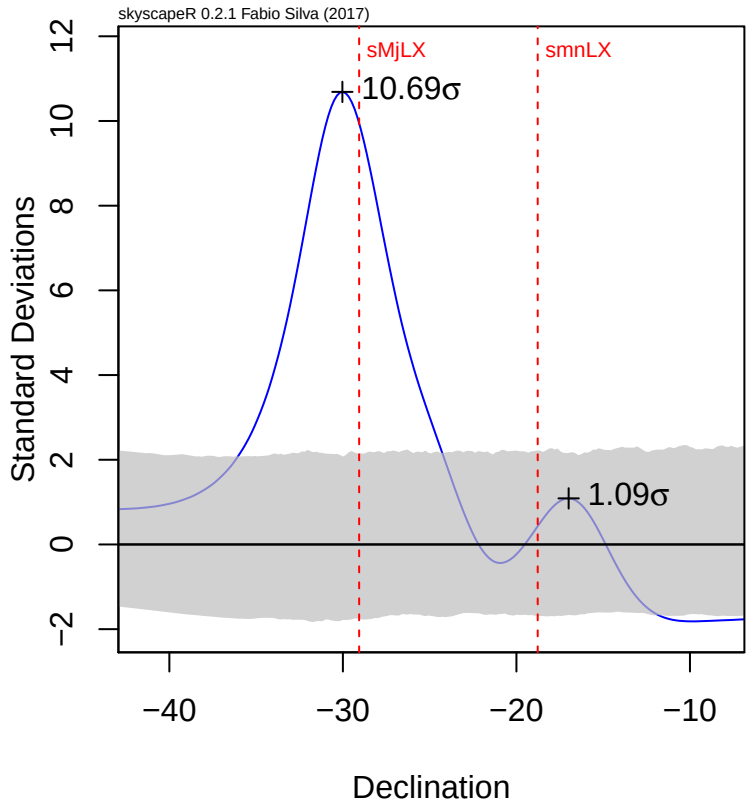
```
plotCurv(curv,signif=sg)
```



Only peaks that are outside the grey-shaded area can be considered significant. To quantify the significance of individual peaks one can plot the z-score transformed curvigram, which also highlights the number of

sigma deviations for each peak:

```
plotZscore(sg, lunar)
```



If one is not interested in ploting the results of the significance testing, but simply getting the values then you can retrieve them from the output of `sigTest()`:

```
sg$p.value
```

```
## [1] 0
```

```
sg$maxima
```

```
##             [,1]       [,2]
## dec    -30.02912 -16.992210
## zScore  10.68681   1.088228
```