# How to use *skyscapeR*

*Fabio Silva*

*2017-10-15*

*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.

## First Steps

Like all vignettes for *R* packages this is neither intended for those unexperienced in *R* nor a fully-fledged manual for *skyscapeR* (that will come later with v.1.0.0). Nor is it going to 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. 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 manual pages for individual functions, 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:

```r
library(skyscapeR)
```

## Data Sets and Help

The current version of *skyscapeR* comes with a couple of 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.

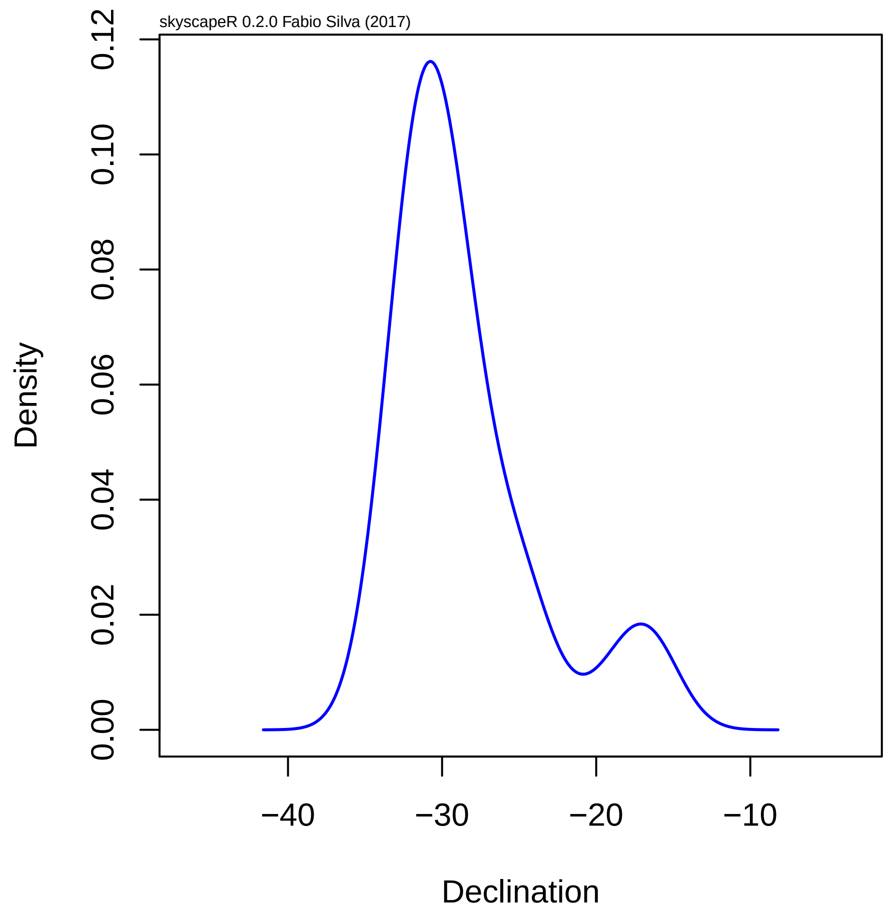
## Hello, Curvigram!

Let's now do 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:

```r
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 2º for all measurements. You can visualize it by typing:

```r
plotCurv(curv)
```



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

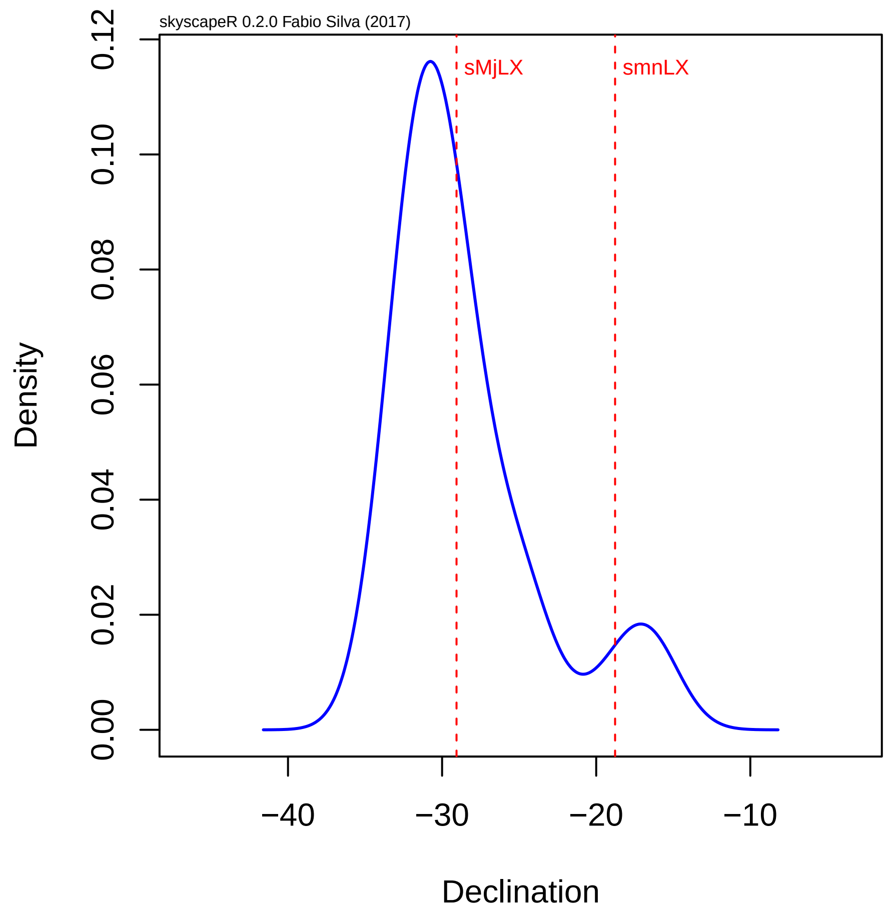
### Adding Celestial Objects to the mix

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()`:

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

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:

```r
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.

## Handling Fieldwork Data

### Data Reduction for Compass Measurements

```
reduction.comp()
```

### Data Reduction for Theodolite Measurements

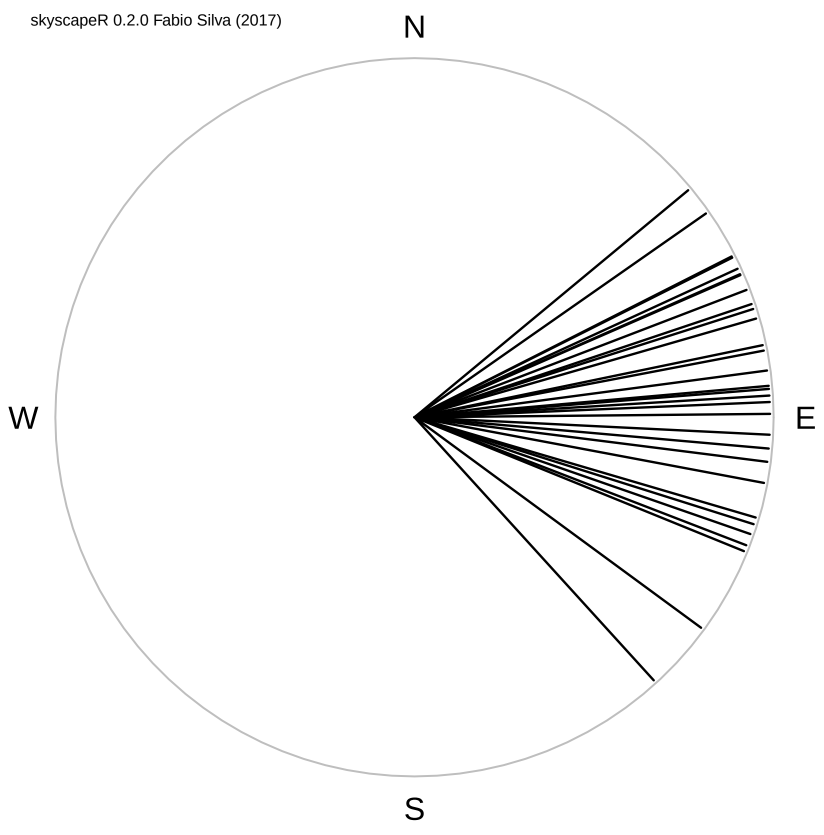
```
reduction.theod()
```

### 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()`:

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

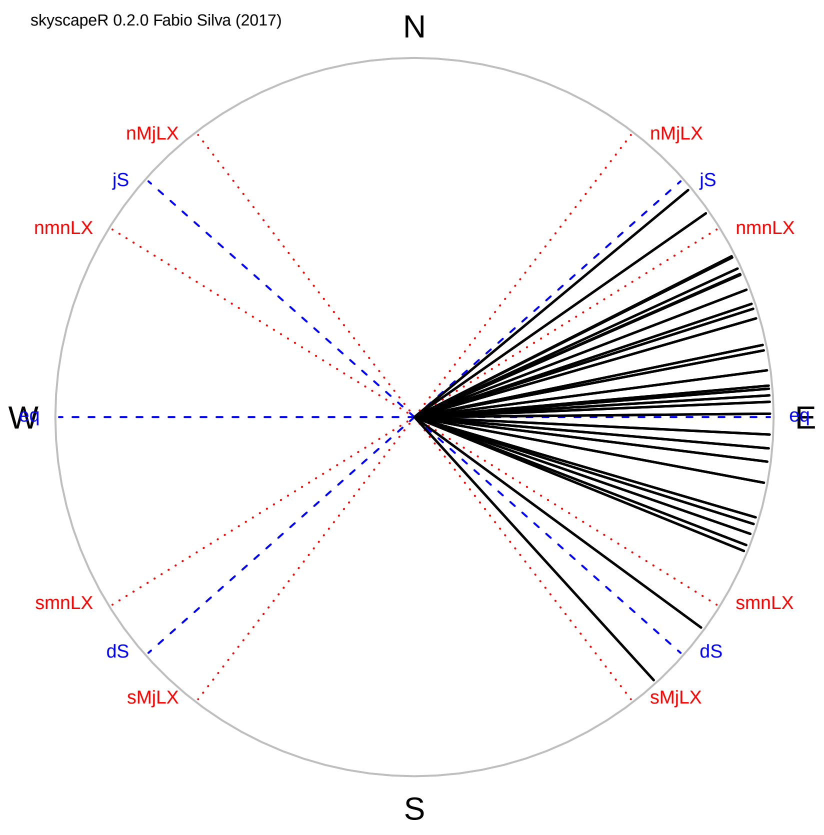
```
## Warning in par(oldpar): graphical parameter "cin" cannot be set

## Warning in par(oldpar): graphical parameter "cra" cannot be set

## Warning in par(oldpar): graphical parameter "csi" cannot be set

## Warning in par(oldpar): graphical parameter "cxy" cannot be set

## Warning in par(oldpar): graphical parameter "din" cannot be set

## Warning in par(oldpar): graphical parameter "page" cannot be set
```



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 0º and flat as well.

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

```
## Warning in par(oldpar): graphical parameter "cin" cannot be set

## Warning in par(oldpar): graphical parameter "cra" cannot be set

## Warning in par(oldpar): graphical parameter "csi" cannot be set

## Warning in par(oldpar): graphical parameter "cxy" cannot be set
## Warning in par(oldpar): graphical parameter "din" cannot be set

## Warning in par(oldpar): graphical parameter "page" cannot be set
```



### Converting Azimuths to Declination

```
az2dec()
```

## Dealing with Horizons

## 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)
```

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

```
sp$phase
```

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

```
sp$events
```

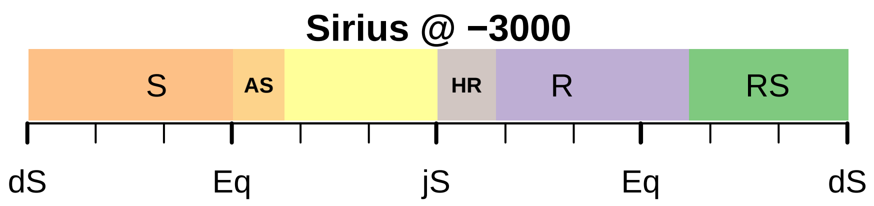
```
##                     [,1]
## Achronycal Setting "April 2 - April 24"
## Heliacal Rising    "July 2 - July 27"
```

```
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)
```

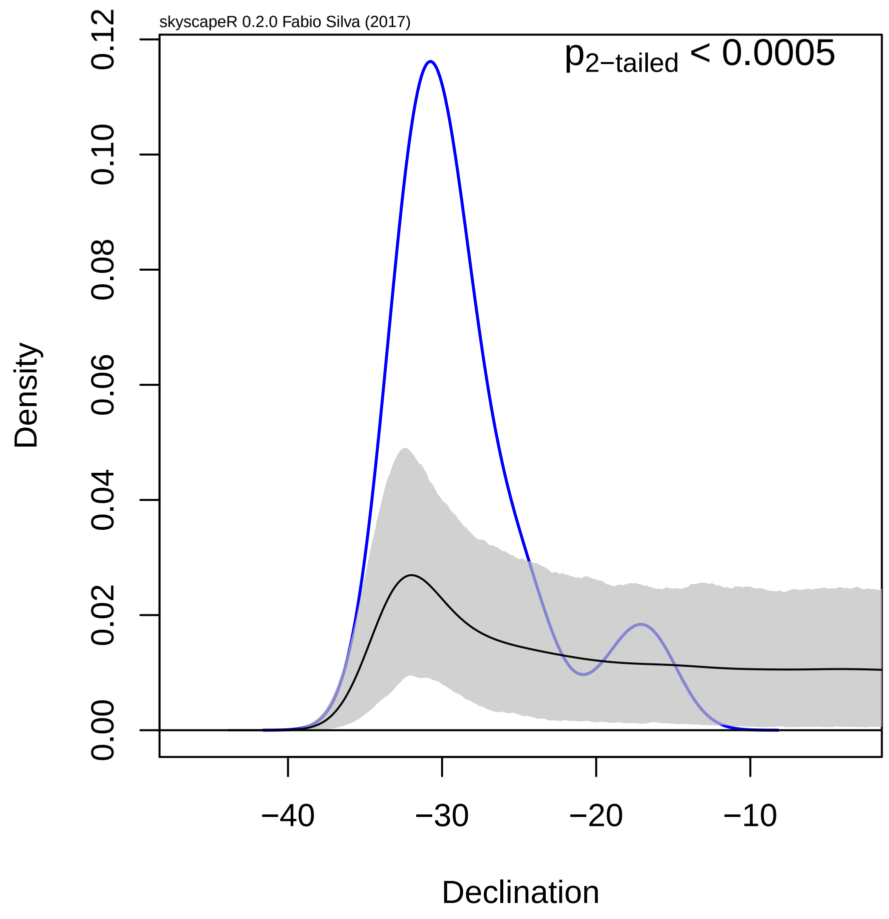
Then one uses `sigTest()` to run the significance testing routine. This can take a while depending on your machine's resources. If it takes too long, try lowering the *nsims* parameter (though this brings a cost of resolution, see the manual page for this function).

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

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

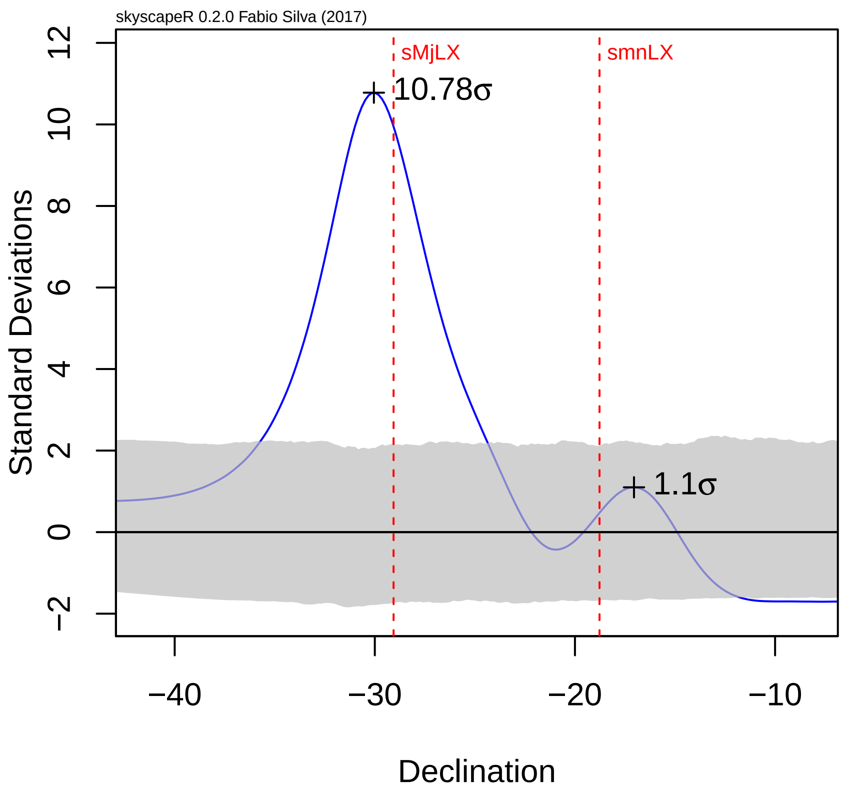
One can then plot the curvigram again, but now with the results of the significance testing displayed. This adds the expectation around the null hypothesis (the grey shaded area) as well as the estimated overall p-value.

```
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    -30.04751 -17.050186
## zScore  10.77827   1.099903
```

## Where next for ***skyscapeR*** ?

## References