Visualisation for bioacoustics and ecoacoustics in R

Ed Baker

2022-10-26

Contents

About

Bioacoustics and ecoacoustics are rapidly advancing multi-disciplinary fields of study that focus on how organisms communicate using sound, and the overall sound of a landscape (the soundscape). Despite the focus on sound, much of the communication of ideas, and even sounds, between researchers is done using graphical representations.

This should not come as a surprise, the printing press came centuries before the radio as a means for long distance communication, and ink on paper has a permanence that sounds would not achieve for a long time after the invention of writing. The current flourishing of these disciplines is driven as much by the low cost and ease of use of products such as AudioMoth and the decreasing cost of digitial storage and processing as by novel ideas.

Visualizations of acoustic data however are not going away - we are a predominantly visual species, and as ways of summarising acoustic data - or making the ultrasound tangible, they are powerful tools in the hands of the acoustician.

Chapter 1

Introduction

> "...while we take it for granted that sounds may be described visually, the convention is recent, is by no means universal and, as I will show, is in many ways dangerous and inappropriate."
>
> — Schafer [1977]

While Murray Schafer's *Tuning of the World* [Schafer, 1977] inspired many soundscape scientists, this view is of an earlier time, where the concept of multiple simultaneous streams of acoustic data being processed by a single individual was still an idea beyond the horizon. Individual sounds could be isolated and studied (as today they still are by bioacousticians interested in the behaviour of individual species). The scale of many contemporary ecoacoustics projects precludes an individual from listening to every minute that is recorded, the task no longer delegated to students but networks of machines.

Additionally, it is now useful to distinguish between two concepts that Schafer bought together under the concept of *notation*. Schafer used this term to bring together what is more typically known as notation – phonetics and musical notation – alongside visual representations of the physical properties of acoustic waves (amplitude, frequency, etc).

Historically both musical notation and phonemes have been used to describe the songs of various animals, however these methods do not scale to the entirety of the biological soundscape. There is after all, a great deal of the soundscape that is beyond the limits of human hearing, the infrasound, the ultrasound, and the quiet. All manner of information is gathered and shared by other species beyond the limits of our perception, and visualisation is the main tool by which we are able to interpret the entire soundscape. For all species that share it.

1.1 Basic acoustic terminology

1.2 Types of files

Even though the costs of digital storage have fallen significantly (Figure 1.1) the cost of file transfer and storage are still significant factors in many acoustics projects. For this reason it is sometimes necessary to compress the audio file in some way: discarding some data in exchange for cost reductions.

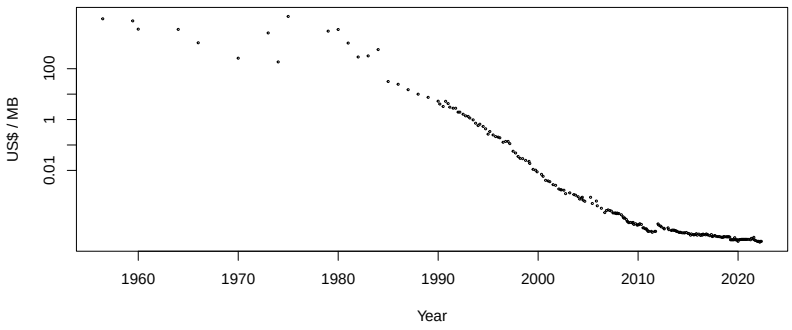


Figure 1.1: Storage costs per megabyte over time. Data from https://jcmit.net/diskprice.htm.

1.2.1 Waveform files

Waveform files are created by sampling the amplitude at a sensor at a constant rate, typically tens of thousands of times per second (Figure @ref{fig:wave-sampling}).

This is achieved using an analogue-to-digital converter (DAC) that converts the continuous variations in amplitude into a number of discrete levels that can be represented numerically (Figure 1.3. The number of discrete levels is called the bit depth.

1.2.2 zero-crossing

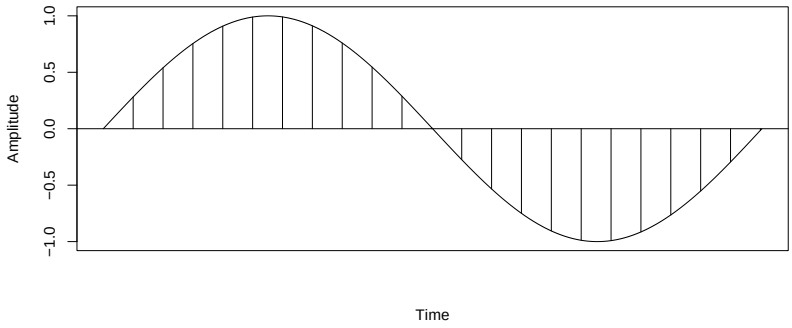


Figure 1.2: Sampling a waveform.

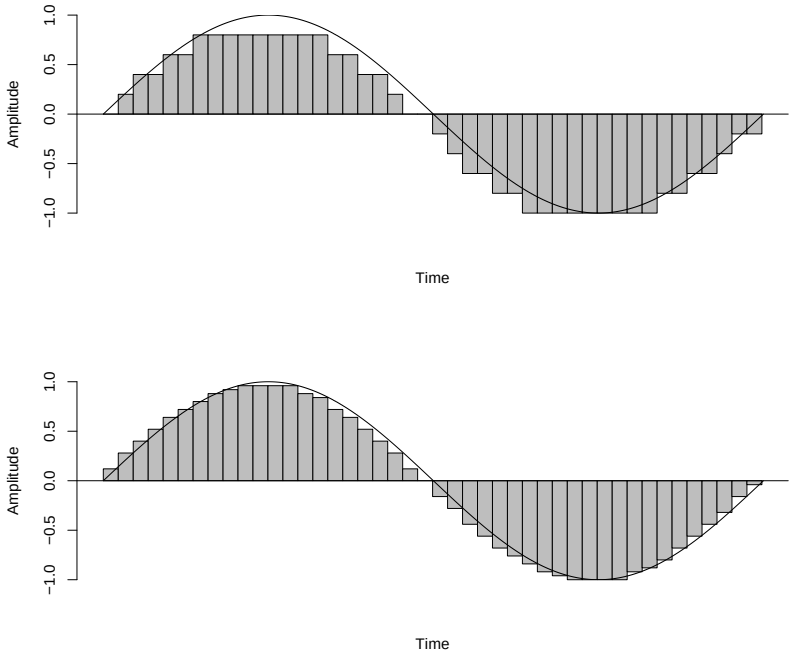


Figure 1.3: A sampled waveform with low bit depth (top) and higher bit depth(bottom).

Chapter 2

History

2.1 Descriptive acoustics

2.1.1 Phonemes and onomatopoeia

2.1.2 Musical notation

Typical musical notation shows broad similarities with a typical audio visualisation familiar to all bioacousticians and ecoacousticians - the frequency against time plot. Time proceeds in a strictly linear fashion from left to right, and frequency is represented rising from bottom to top.

2.2 Analytic acoustics

2.2.1 The 'big-three'

2.2.1.1 Amplitude vs Time

2.2.1.2 Ampltiude vs Frequency

2.2.1.3 Frequency vs Time

Chapter 3

Early visualisations - the analog years

3.1 CRTs

3.2 Print outs

Chapter 4

Static digital images

The three basic quantities of sound waves are time, frequency, and amplitude. The most commonly used visualisations of sounds are consequently combinations of these properties.

The `seewave` package [Sueur et al., 2008] is the easiest way to produce these plots, and we will use the `sheep` audio sample provided by `seewave` for the first plots.

```
library(seewave)
data(sheep, package="seewave")
```

4.1 The time domain

The *time domain* is used to describe acoustic analyses where the recorded waveform is not analysed for frequency. Conversion of a waveform to a spectrum of frequencies is a relatively costly computation and, before digital computers, hard to achieve with the precision and resolution we are accustomed to today.

4.1.1 Oscillograms

Digitally sampled audio is a series of amplitude measurements, sampled at a regular time time interval. Given that it now seems natural for time to be expressed on the x axis, increasing towards the right, the most basic plot we could recreate from a sampled file would be plotting amplitude values against time.

Figure 4.1 shows 50 samples from the `sheep` audio file plotted using the standard R `barplot()` function. This representation has the advantage that the individual bars ar a visual reminder that we are handling a sampled representation, as opposed to a continuously varying waveform. It is common however to create a

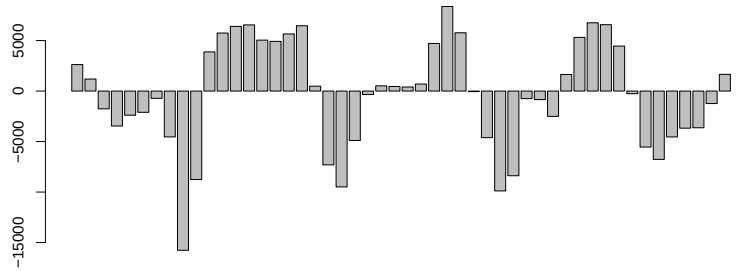


Figure 4.1: Plotting a waveform

visual representation closer to that of an actual wave (Figure @ref(fig:amplitude-lines shows how this is achieved using base R functions).

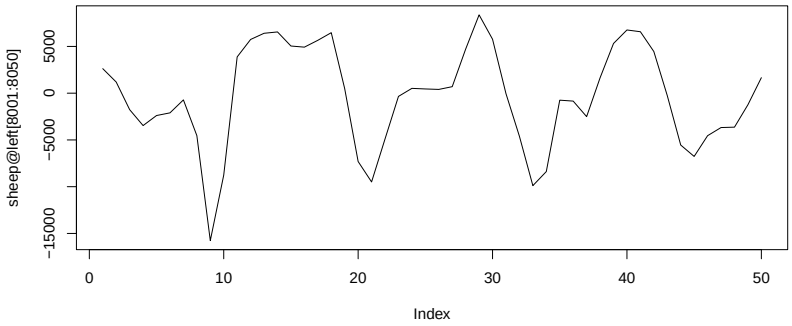


Figure 4.2: Plotting a waveform

The `oscillo()` function from `seewave` handles the labelling of axes for us, and is a very convenient solution. Figure 4.3 shows the resulting plot, an oscillogram, for the entire `sheep` file.

4.2 The frequency domain

The *frequency domain* covers analyses in which frequency data is extracted from the sampled waveform. This information is generally extracted using the *Fast Fourier Transform* algorithm.

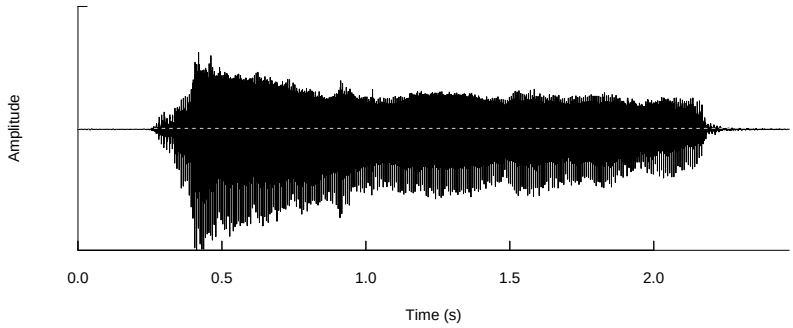


Figure 4.3: Oscillogram

4.2.1 Plotting a spectrum

A spectrum is a plot of amplitude against frequency.

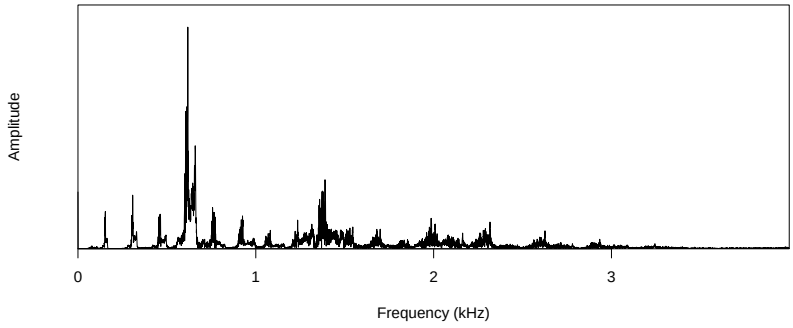


Figure 4.4: Spectrum

4.2.2 Plotting a spectrogram

4.2.2.1 Settings for a FFT

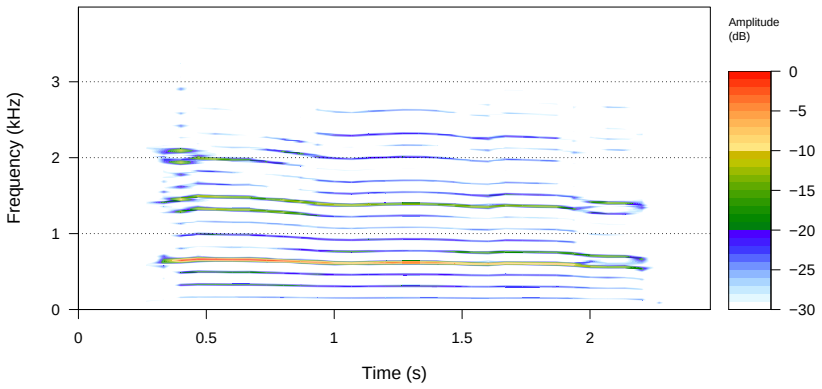


Figure 4.5: Spectrogram

Chapter 5

Dynamic digital visualisations

5.1 Video spectrograms

5.2 zcjs: Zero-crossing File Visualisation

The zcjs JavaScript library was originally developed for displaying zero-crossing audio files (1.2.2) on the web as part of the BioAcoustica project (https://bio.acousti.ca; Baker et al. [2015]). The `zcjs` package [Baker, 2022] imports this visualisation functionality into R.

The following code installs the `zcjs` package.

```
install.packages("devtools")
devtools::install_github("bioacoustica/zcjs-r")
```

To load the package:

```
library(zcjs)
```

The package comes with a demonstration file for testing the package's functionality.

5.2.1 Customising a zcjs plot

x-compress

Chapter 6

Representing Soundscapes

6.1 False Colour Index Spectrograms

Chapter 7

Patterns of activity

Lots of organismal activities are tied to the cycles of the day, and particularly in temperate zones, cycles of the year. These cycles bring regular fluctuations in light levels, day lengths, temperatures, and a host of other influences. Often these cycles interact, with the dawn chorus peaking in the early daylight hours, and it's timing and intensity fluctuating on a yearly cycle. This chapter looks at visualising these cycles, and additionally the effects of lunar cycles.

These plots are created using the SonicScrewdriver package [Baker, 2021] which in turn uses the suncalc package [Thieurmel and Elmarhraoui, 2019] to perform the required sun and moon position calculations. The Plotrix package [Lemon, 2006] is used for creating the visualisation. These packages can be installed as shown below.

```
install.packages(c("plotrix", "sonicscrewdriver"))
```

The SonicScrewdriver package must be loaded before constructing a visual.

```
library(sonicscrewdriver)
```

7.1 Daily Cycles

The use of the term *diel* for daily cycles has been contested by Broughton [1963] as being an incorrectly formed unnecessary neologism, it sees greater use (according to the online Oxford English Dictionary) than his suggested *nycthemeral*.

The design for these plots came from a desire to compare the dawn chorus at various locations around the UK, although they also offer great potential for comparing locations with greater longitudinal and/or latitudinal separation. The plots show the times of day, night, twilight (7.1.1), sunrise, sunset, nadir and solar noon. The day part of the plot shows the altitude (angle of the sun

above the horizon) throughout the day, with the maximum value representing the sun being directly overhead.

7.1.1 The Types of Twilight

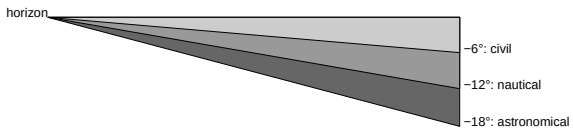


Figure 7.1: Different types of twilight as the sun sets below the horizon

7.1.1.1 Civil Twilight

Civil twilight occurs when the geometric centre of the sun (as seen from Earth) passes between 0° and 6° below the horizon. During this time it is normal for humans not to need the assistance of artifical light for everyday tasks.

7.1.1.2 Nautical Twilight

Nautical twilight occurs when the sun is between 6° and 12° below the horizon. During this time there is sufficient light to distinguish the horizon even without illumination from the moon (allowing determination of position at sea through star sightings).

7.1.1.3 Astronomical Twilight

When the sun is between 12° and 18° below the horizon many astronomical observations are possible even though some light from the sun is visible through the atmosphere. In urban areas with light pollution this is often considered to be a dark sky.

7.1.2 Diel Plots

As the times of the solar day are dependent both on the date and location these must be passed to the `dielPlot()` function.

```
dielPlot("2022-08-08", lat=53, lon=0.1)
```

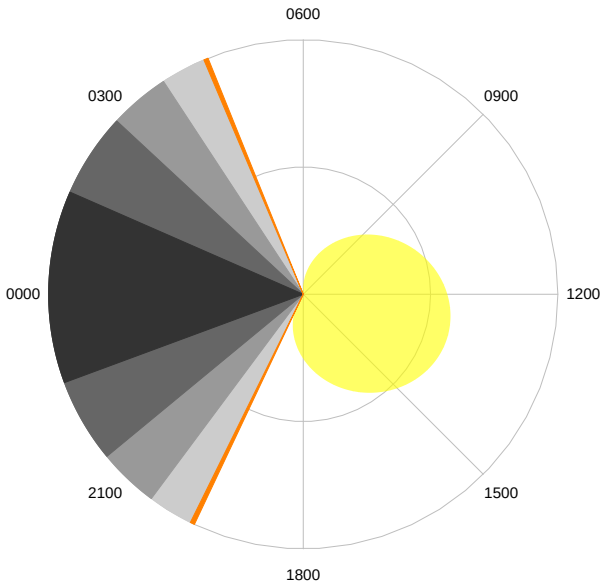


Figure 7.2: Example of a diel plot

7.1.2.1 Rotations of a `dielPlot()`

By default the information is plotted in the UTC timezone, so locations in other timezones will have an overall rotation.

```
par(mfrow=c(1,3))
dielPlot(Sys.Date(), lat=53, lon=-50)
dielPlot(Sys.Date(), lat=53, lon=-0)
dielPlot(Sys.Date(), lat=53, lon=50)
```

Plots can be made in any timezone by using the `rot` parameter to `dielPlot()` and the `tz()` function.

```
par(mfrow=c(1,3))
dielPlot(Sys.Date(), lat=53, lon=-50, rot=tz(3))
dielPlot(Sys.Date(), lat=53, lon=-0, rot=tz(3))
dielPlot(Sys.Date(), lat=53, lon=50, rot=tz(3))
```

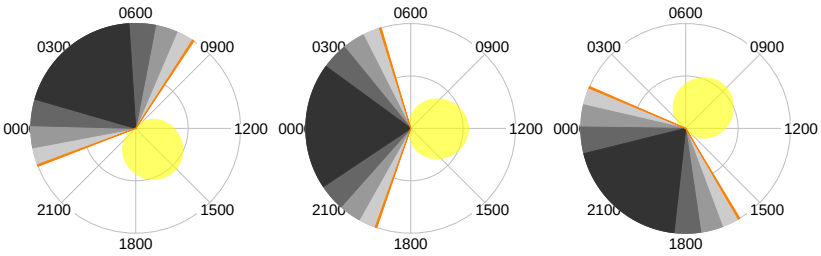


Figure 7.3: Diel plots in UTC.

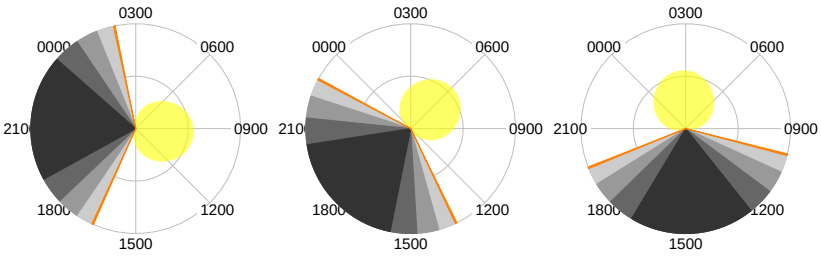


Figure 7.4: The same diel plots in UTC +3

By setting the `rot` parameter to `Solar Noon` it is possible to align the plots to solar noon. Notice that this rotates the plot labels.

```
par(mfrow=c(1,3))
dielPlot(Sys.Date(), lat=53, lon=-50, rot="Solar Noon")
dielPlot(Sys.Date(), lat=53, lon=-0, rot="Solar Noon")
dielPlot(Sys.Date(), lat=53, lon=50, rot="Solar Noon")
```

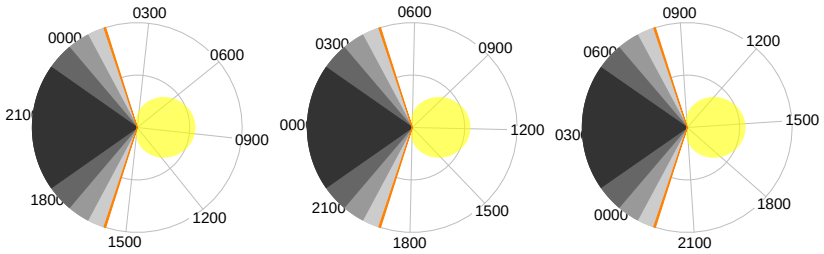


Figure 7.5: The same diel plots aligned to solar noon

7.1.2.2 Customising a `dielPlot()`

In addition to the `date`, `lat` and `lon` parameters to `dielPlot()` it is possible to make additional customisations to how the information is presented.

Legend

A legend can be added to the plot by setting `legend=TRUE`.

```
dielPlot("2022-08-08", lat=53, lon=0.1, legend=TRUE)
```

Plotting Components

The components that can be plotted are listed below. By default all are plotted except for `Solar Noon` and `Nadir`.

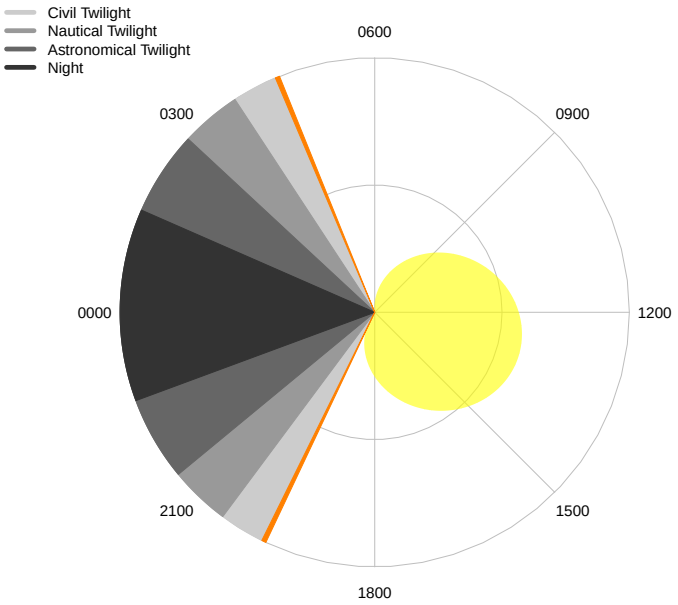
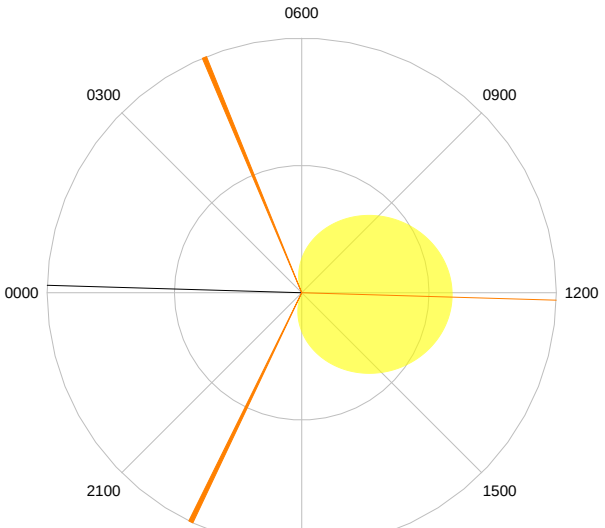


Figure 7.6: Example adding a legend to a diel plot

Name	Notes
Astronomical Twilight	
Nautical Twilight	
Civil Twilight	
Sunrise	
Solar Noon	The time when the sun is highest in the sky
Sunset	
Nadir	

The components that are plotted can be specified using the `plot` parameter.

```
components <- c("Sunrise", "Sunset", "Solar Noon", "Nadir")
dielPlot("2022-08-08", lat=53, lon=0.1, plot=components)
```



```
yearlyPlot(2022, lat=53, lon=0.1)
```

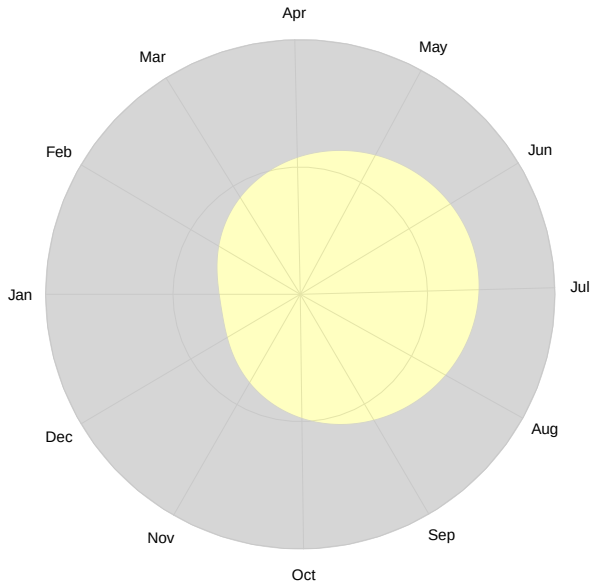


Figure 7.8: Example of a yearly plot

7.3 Lunar Cycles

7.4 Core and ring plots

These visualisations for cyclical data plot their information onto a circle with radius of two units. It is possible to limit the plot either to the centre of the circle (a 'core' plot) or to the edge (a 'ring plot'). These alternative forms may be more useful when these plots are used to visualise addition variables (7.7).

```
dielPlot("2022-08-08", lat=53, lon=0.1, limits=c(0,1))
```

```
dielPlot("2022-08-08", lat=53, lon=0.1, limits=c(1,2))
```

7.5 Behind the scenes

7.5.1 `radialPolygon()`

The majority of the plotting performed for cyclical plots in `SonicScrewdriver` is performed by the `radialPolygon()` function. This function can be used to plot

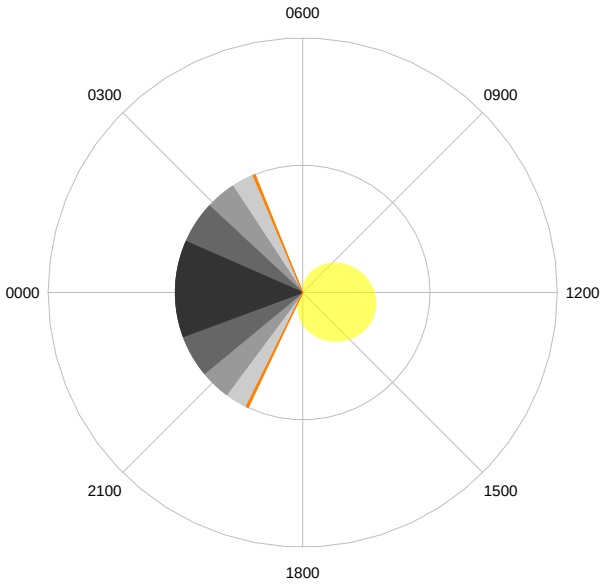


Figure 7.9: A 'core' diel plot.

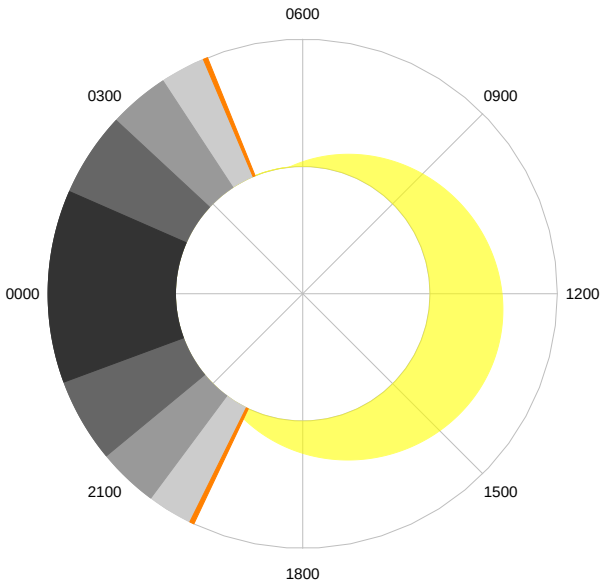


Figure 7.10: A 'ring' diel plot.

sectors, annuli, horizon plots and irregular polygons. It is used by the plotting functions such as `dielPlot()` as well as helper functions such as `dielRings()` to add data to cyclical plots.

For simple use cases, knowledge of the operation of `radialPolygon()` may not be needed. A number of helper functions cover the most common uses. However, an understanding of how this function works will allow for far greater customisation of cyclical plots than would otherwise be possible.

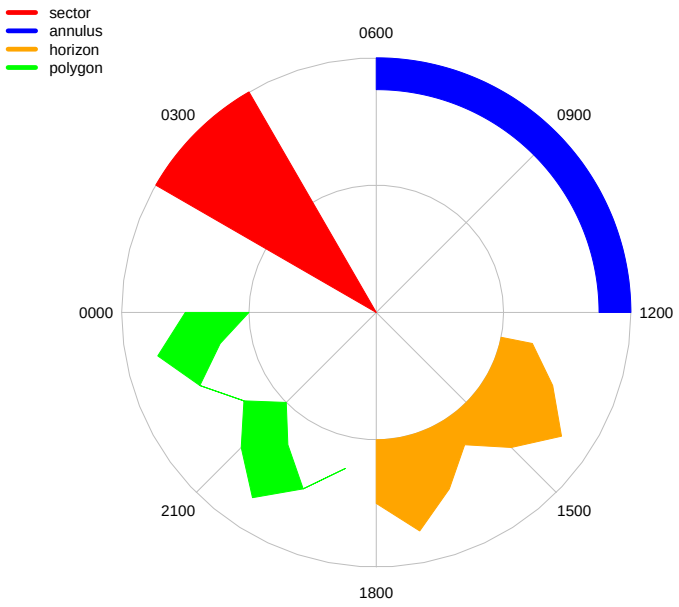


Figure 7.11: Types of radial polygon plot

The various types of plots are created by changing the angle and radius parameters to `radialPolygon()`.

```
radialPolygon(angle1, angle2, radius1, radius2, col)
```

7.5.1.1 Orientation

Unlike traditional polar plots, diel and yearly plots start their periods on the left hand horizontal, and proceed clockwise. This orientation is assumed by `radialPolygon()`, although it may be modified (e.g. the parameters `reverse=FALSE` and `rot=0` will plot using the standard conventions for polar coordinate systems.)

7.5.1.2 Sectors

A sector is a section of a circle defined by two radii and an arc between them. Sectors are used widely in the default settings of `dielPlot()` to plot the times of night and twilight.

```
emptyDiel()
radialPolygon(pi/6, 2*pi/3, 0, 2, col="red")
```

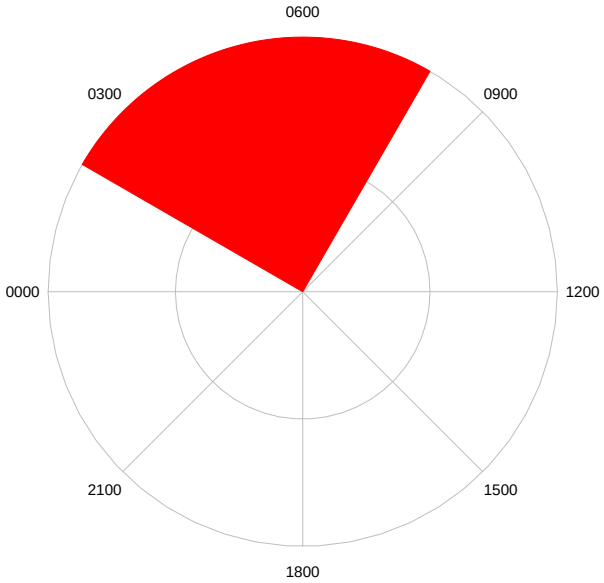


Figure 7.12: Plotting a sector

Reversing the angle arguments allows the complementary sector to be drawn.

```
emptyDiel()
radialPolygon(2*pi/3, pi/6, 0, 2, col="blue")
```

7.5.1.3 Annuli

An annulus is the region between two concentric circles. Annuli and annular sectors are generated by `radialPolygon()` when the parameter `radius` is greater than zero.

```
emptyDiel()
radialPolygon(0, 2*pi, 1.75, 2, col="blue")
radialPolygon(pi, 4*pi/3, 1, 1.5, col="red")
legend(
```

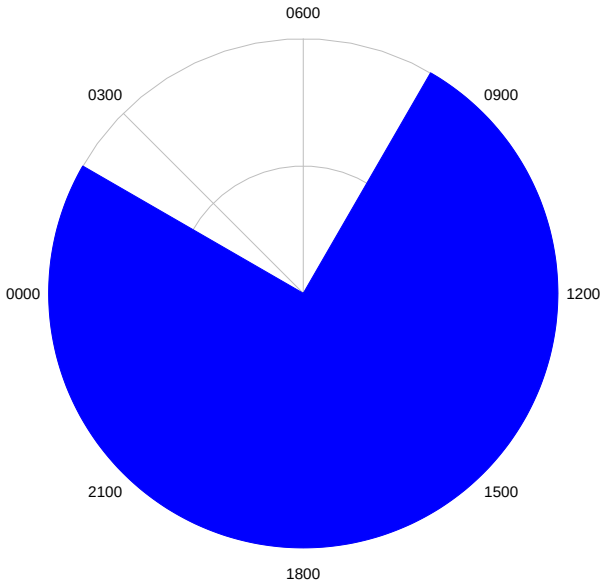


Figure 7.13: Plotting a sector

```
  -3,2.5,
  c("annulus", "annuluar sector"),
  col=c("blue", "red"),
  lty=1,
  lwd=5,
  bty = "n",
  cex = 1)
```

7.5.1.4 Horizons

Horizons have one circular edge, and one that represents data, they are named as they often resemble a landscape or cityscape horizon. The example below uses a generated sine pattern to form the data edge.

```
library(tuneR)

angles <- (0:200)*pi/200 + pi/2
values <- 0.05*sine(10, samp.rate=201)@left

emptyDiel()
radialPolygon(NA,angles,0.5,1+values)
```

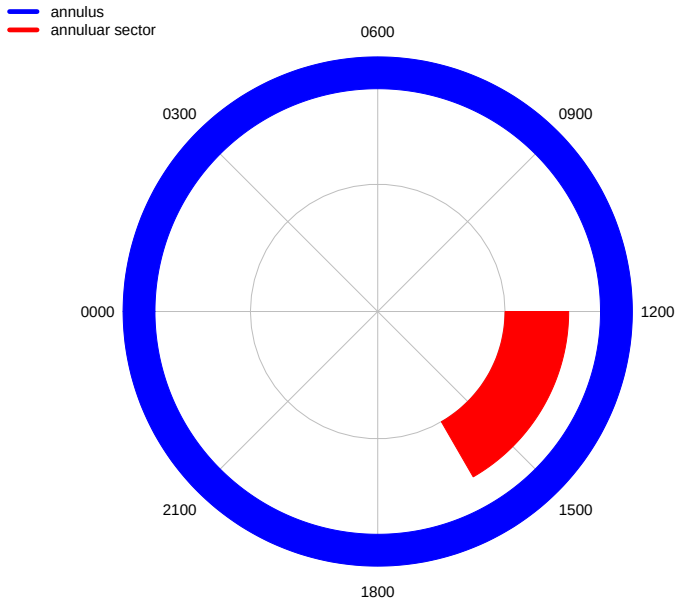


Figure 7.14: Plotting an annulus and an annular sector

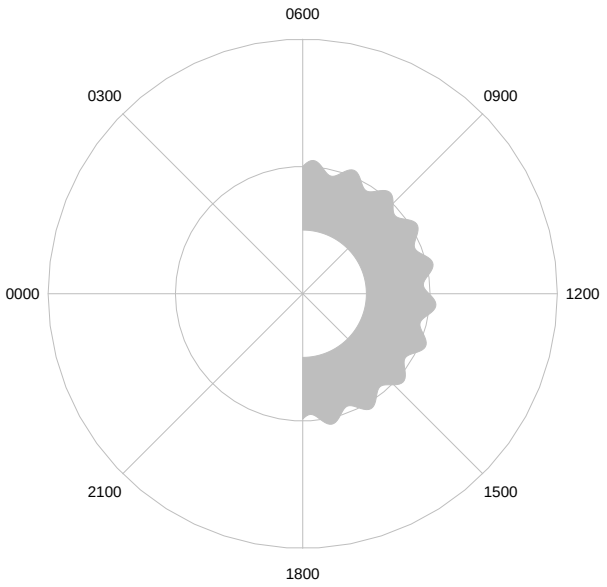


Figure 7.15: Plotting a horizon polygon.

Setting the first angle parameter to `NA` uses the range of the second to calculate the inner edge.

The inner edge can be used to show data by swapping the order of the angle and radius parameters.

```
library(tuneR)

angles <- (0:200)*pi/200 + pi/2
values <- 0.05*sine(10, samp.rate=201)@left

emptyDiel()
radialPolygon(angles,NA,1+values,2)
```

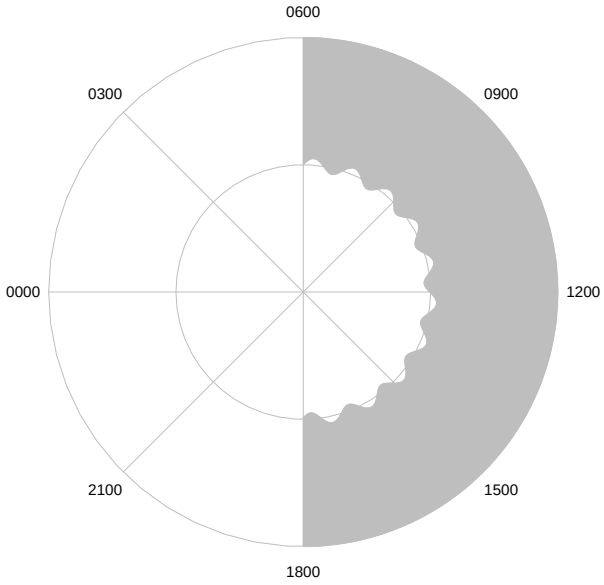


Figure 7.16: Plotting a horizon polygon.

The `yearlyPlot()` function uses two horizon plots, with a shared data edge.

7.5.1.5 Polygons

7.6 Empty plots

The uses of daily and yearly plots extend beyond linking data to earthly cycles. The following functions provide the basic coordinate system without any data plotted (these functions are used intenally by `dielPlot()` and `yearlyPlot()`

to establish their coordinate system).

7.6.1 emptyDiel()

```
emptyDiel()
```

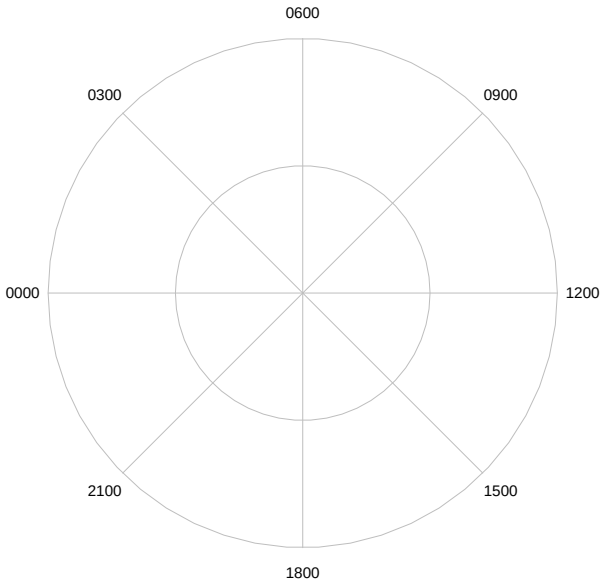


Figure 7.17: An empty dielPlot().

7.6.2 emptyYearly()

The `emptyYearly()` function takes an option `year` parameter that automatically adjusts the grid positions for leap years, although the visual effect is marginal.

```
emptyYearly(year=2000)
```

7.7 Adding data to the visualisation

7.7.1 Periodic data: rings

The ring functions (`dielRing()`,…) plot ring segments on top of a base cyclical plot. These rings are useful for showing typical periods of activity for a species, or events that happen continuously for a specified period of time.

By defaults the limits for the rings are `1,2` for use with a *core* type plot, but

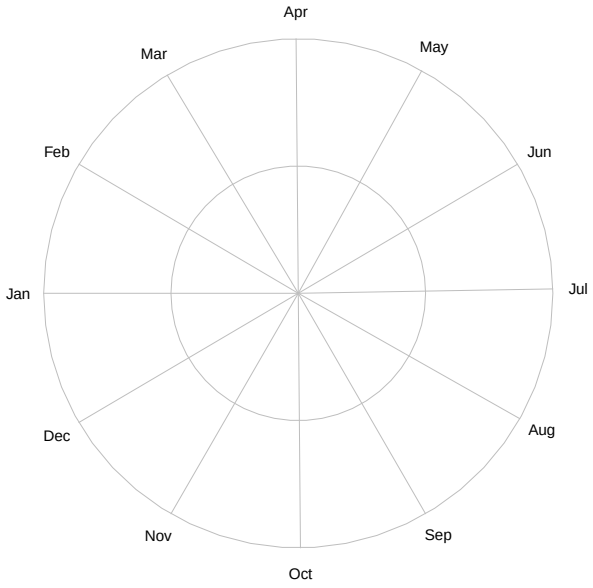


Figure 7.18: An empty yearlyPlot().

this can be changed by specifying the `limits` parameter to the ring function. Similarly, the plot legend may be removed with the paramater `legend=FALSE`.

7.7.1.1 `dielRings()`

```
names <- c("activity 1", "activity 2", "activity 3")
starts <- c("0600", "0900", "1500")
ends <- c("1200", "1700", "1900")
cols <- c("red", "green", "blue")

dielPlot("2022-08-08", lat=53, lon=0.1, limits=c(0,1))
dielRings(names, starts, ends, cols=cols)
```

```
names <- c("activity 1", "activity 2", "activity 3")
starts <- c("0600", "0900", "1500")
ends <- c("1200", "1700", "1900")
cols <- c("red", "green", "blue")

dielPlot("2022-08-08", lat=53, lon=0.1, limits=c(1,2))
dielRings(names, starts, ends, cols=cols, limits=c(0,1))
```

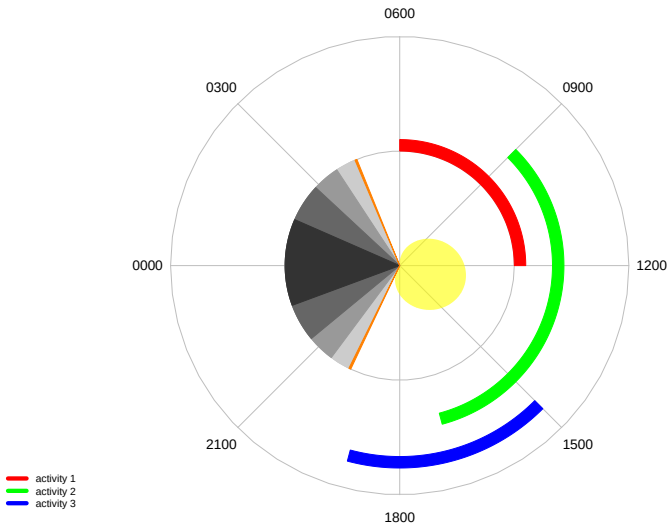


Figure 7.19: A 'core' diel plot with diel rings.

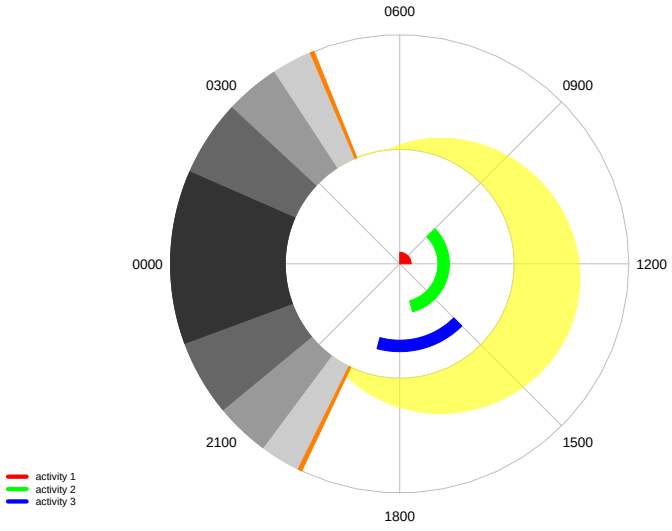


Figure 7.20: A 'ring' diel plot with diel rings.

7.7.2 Periodic data: horizons

In this example we will plot the average monthly minimum and maximum temperatures for Lyme Regis, UK onto a `yearlyPlot()`. This example introduces three small helper functions; `yearlyLabels()`, `yearlyPositions()` and `circularise()`.

7.7.2.1 `yearlyLabels()` and `yearlyPositions`

These two functions are closely related, and used internally by `SonicScrewdriveR` to label a yearly plot.

```
yearlyLabels()
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

The related `yearlyPositions()` gives angular positions for the data around the plot (in radians).

```
yearlyPositions(year=2022)
```

```
##  [1] 0.0000000 0.5336404 1.0156382 1.5492786 2.0657048 2.5993452 3.1157713
##  [8] 3.6494117 4.1830521 4.6994783 5.2331187 5.7495449
```

The temperature data we have for Lyme Regis is monthly, however we would like to plot the values at the middle of the respective month. For this we add the parameter `format="mid-month"` to get the appropriate radial angles.

```
yearlyPositions(year=2022, format="mid-months")
```

```
##  [1] 0.2668202 0.7746393 1.2824584 1.8074917 2.3325250 2.8575582 3.3825915
##  [8] 3.9162319 4.4412652 4.9662985 5.4913318 5.9733296
```

If we now plot this data we see that the output is not quite a complete ring or horizon data as we might have expected.

```
# Temperature data for Lyme Regis
t_min <- c(3, 2.7, 3.4, 5.2, 8.2, 11.2, 13.1, 13, 14.4, 9.2, 5.8, 3.8)
t_max <- c(7.9,8, 9.8, 12.4, 15.4, 18.2, 20.1, 19.5, 17.8, 14.4, 10.6, 8.4)

# Scale the data
sf <- max(t_max)
t_min <- t_min/sf
t_max <- t_max/sf

angles <- yearlyPositions(format="mid-months")

yearlyPlot(lat=50.7, lon=-2.9)
radialPolygon(angles, angles, t_min, t_max, col="orange")
```

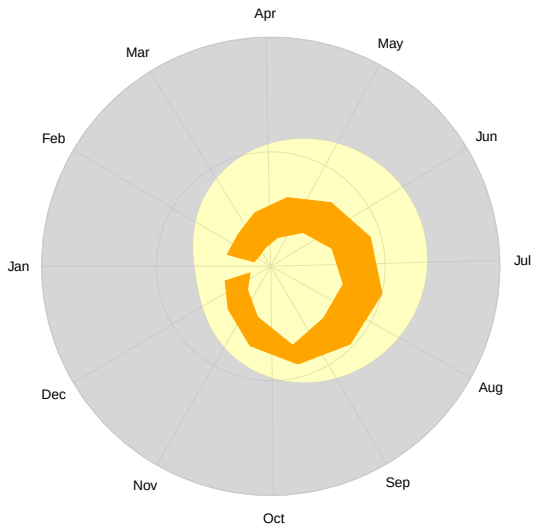


Figure 7.21: Horizon data plot without circularise().

7.7.2.2 `circularise()`

In order to join the horizons into a complete ring we can use the `circularise()` function on the angle and temperature vectors.

```
# Temperature data for Lyme Regis
t_min <- c(3, 2.7, 3.4, 5.2, 8.2, 11.2, 13.1, 13, 14.4, 9.2, 5.8, 3.8)
t_max <- c(7.9,8, 9.8, 12.4, 15.4, 18.2, 20.1, 19.5, 17.8, 14.4, 10.6, 8.4)

# Scale the data
sf <- max(t_max)
t_min <- t_min/sf
t_max <- t_max/sf

angles <- yearlyPositions(format="mid-months")

# Circularise
t_min <- circularise(t_min)
t_max <- circularise(t_max)
angles <- circularise(angles)

yearlyPlot(lat=50.7, lon=-2.9)
radialPolygon(angles, angles, t_min, t_max, col="orange")
```

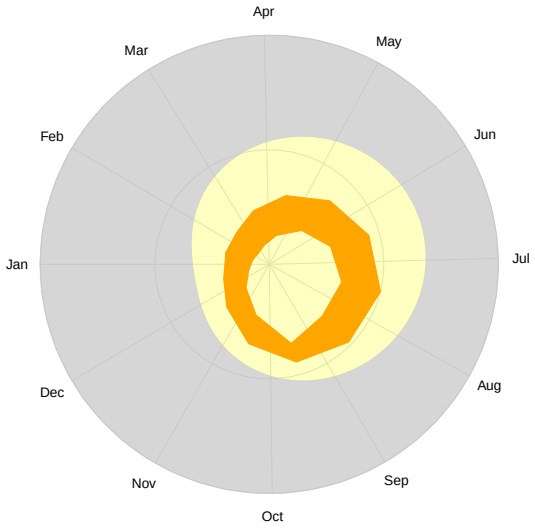


Figure 7.22: Horizon data plot with circularise().

7.7.3 Helper functions

7.7.3.1 `dielFraction()`

The `dielFraction()` function is used to convert a POSIX time, or a time in `HHMM` string format, into a radial fraction. This function is called by `dielPlot()` and `dielRings()`. By default the output is multiplied by 2 to output a position round a circle.

```
time <- Sys.time()
print(time)
```

```
## [1] "2022-10-26 16:00:32 BST"
```

```
frac <- dielFraction(time)
print(frac)
```

```
## [1] 4.191134
```

The raw fraction can be specified using the parameter `unit="fraction"`.

```
time <- Sys.time()
print(time)
```

```
## [1] "2022-10-26 16:00:32 BST"
```

```
frac <- dielFraction(time, unit="fraction")
print(frac)
```

```
## [1] 0.6670401
```

7.7.3.2 `yearlyFraction()`

Similarly to `dielFraction()`, `yearlyFraction()` by default returns a fraction of a cycle. It can take either a number representing a day of the year, or an object that can be coerced into `POSIXlt` format.

```
frac <- yearlyFraction(31, input="day")
print(frac)
```

```
## [1] 0.5336404
```

```
date <- Sys.Date()
print(date)
```

```
## [1] "2022-10-26"
```

```
frac <- yearlyFraction(date, input="POSIXlt")
print(frac)
```

```
## [1] 5.129833
```

7.8 Interactive Plots

These plots can be used to create Shiny apps

- shiny-diel is an example that shows diel plots for a number of locations, and can be animated using the play button under the date slider.

Chapter 8

Displaying annotations

Chapter 9

Shiny: Interactive Web Apps

Chapter 10

The Future

Chapter 11

Acknowledgements

For discussions around visualisation as part of the Urban Nature Project: Chris Raper, John Tweddle.

Bruce Miller provided valuable feedback during the development of the zcjs visualisation tools for zero-crossing files.

The `SonicScrewdriveR` package was initially developed as part of the Leverhulme Trust funded Automated Acoustic Observatories project at the University of York, and development has continued as part of the Urban Nature Project at the Natural History Museum.

Bibliography

E. Baker. *SonicScrewdriveR*, 2021. URL https://cran.r-project.org/package=sonicscrewdriver.

Ed Baker. *zcjs*, 2022. URL https://github.com/bioacoustica/zcjs-r.

Edward Baker, Ben W. Price, S. D. Rycroft, Jon Hill, and Vincent S Smith. Bioacoustica: A free and open repository and analysis platform for bioacoustics. *Database*, 2015(bav054), 2015.

W.B. Broughton. *Acoustic Behavior of Animals*, chapter Glossarial Index. Elsevier, 1963.

J Lemon. Plotrix: a package in the red light district of r. *R-News*, 6(4):8–12, 2006.

R. Murray Schafer. *Tuning of the World*. Random House, 1977.

J. Sueur, T. Aubin, and C. Simonis. Seewave: a free modular tool for sound analysis and synthesis. *Bioacoustics*, 18:213–226, 2008. URL https://www.tandfonline.com/doi/abs/10.1080/09524622.2008.9753600.

Benoit Thieurmel and Achraf Elmarhraoui. *suncalc: Compute Sun Position, Sunlight Phases, Moon Position and Lunar Phase*, 2019. URL https://CRAN.R-project.org/package=suncalc. R package version 0.5.0.