Light-level geolocation analyses

Last edited on 2018-11-15

Contents

Preface

This manual is part of the following publication and has been written by the same group of authors:

Simeon Lisovski, Silke Bauer, Martins Briedis, Kiran Danjahl-Adams, Sarah Davidson, Christoph Meier, Lykke Pedersen, Julia, Karagicheva, Benjamin Merkel, Janne Ouwehand, Michael T. Hallworth, Eldar Rakhimberdiev, Michael Sumner, Caz Taylor, Simon Wotherspoon, Eli Bridge (201X) The Nuts and Bolts of Light-Level Geolocation Analyses. Journal X:xxx-xxx.

Geolocation by light is a method of animal tracking that uses small, light-detecting data loggers (referred to as geolocators) to determine the locations of animals based on the light environment they move through. Technological and fieldwork issues aside, effective use of light level geolocation requires translation of a time series of light levels into geographical locations. Geographical locations that are derived from light-level

data are subject to error which directly arises from noise in the light-level data, i.e. unpredictable shading of the light sensor due to weather or the habitat (Lisovski et al. 2012). Although light-level geolocation has provided a wealth of new insights into the annual movements of hundreds of bird species and other taxa, researchers struggle with the analytical steps that are needed to obtain location estimates, interpret them, present their results, and document what they have done.

This manual has been written by some of the leading experts in geolocator analysis and is based on material created for several international training workshops. It offers code and experience that we have accumulated over the last decade, and we hope that this collection of analysis using different open source software tools (R packages) helps both newcomers and experienced users of light-level geolocation.

Acknowledgements

We want to acknowledge all people that have been involved in the development of geolocator tools as well as all participants of the many international geolocator workshops. Furthermore, we like to acknowledge Steffen Hahn and Felix Liechti organisewho organised a first workshop of the analysis of geolocator data from songbirds back in 2011. This workshop has been financially supported by the Swiss Ornithological Institute and the Swiss National Science Foundation. The National Centre for Ecological Analysis and Synthesis (NCEAS) has supported two meetings with experts in geolocator analysis in 2012 ans 2013 and many of the tools that are discussed in this manual were kick started at these meetings. We want to thank James Fox from Migrate Technology Ltd. as well as the US National Science Foundation for contiouing financial support to develop tools and organise workshops.

License

Chapter 1

Structure of the manual

This manual should allow users with limited knowledge in R coding to perform a state-of-the-art analysis of geolocator data. Thus, we start with the very basics of loading packages and data 3 Starting with the initial data editing steps, which we call twilight annotation 4, we provide instructions on how to use several prominent analysis packages, illustrate the general analysis workflow using example data, and provide some recommendations for how to visualize and present results. We do not cover every available analysis package but focus on what we percieve to be the most frequently used tools, which are GeoLight 5, probGLS 6, SGAT 7 and FLightR 8. The manual concludes with a section on data repositories such as Movebank that allows storing and shring geolocator tracks 9.

The datasets

To illustrate the capabilities of the different packages, discuss the potential pitfalls, and provide some recommendations, we will use raw geolocator data from four individuals of different species. All used tag data and the results as well as the code for the analyses has been uploaded onto Movebank unter study: xxxx.

TagID	Species	Folder	Tag type
M034	Red-backed Shrike	LanCol	Integio (Migrate Technology Ltd.)
14SA	European bee-eater	MerApi	PAM (Swiss Ornithological Institute)
PasCir01	Purple martin	PasCir	Custom (by Eli Bridge)
2655	Brünnich's guillemot	UriLom	Lotek

Although all of these tag types record light values over time, they differ in some key details. First, tags often differ in the frequency at which they write/log data. Many tags collect a reading every minute and store the maximal light value every 5 or 10 minutes. Other may store a maximum every 2 minutes. The tag that yielded the Purple martin data set, averaged 1min readings every 10min instead of taking a maximum. These four tags also differ in their sensitivity and how they record light levels. Some tags are sensitive only at low light levels and quickly "max out" when they experience a lot of light. As such, their light-levels do not have units and are simply an index of light intensity. The Integio tags can record unique light values for all natural light levels on earth, and they store lux values that range from 0 to ~70,000. Depending on the tag type, you may have to perform some preliminary steps such as log-transforming your data or time shifting light values for sunsets (we will provide details while working on the specific datasets).

Reproducing the analyses

This manual contains code that can be copy pasted into an R script and executed to reproduce the results. In order to do so, you need to download the raw data as well as annotated twilight files used in this manual. The data need to be in a specific structure of folders and we do recommend you have a similar structure for your own analysis. During the processing of the data we save intermediate steps that allow us to step into the next analysis step without going through all initial and often time consuming parts. Having your raw data and your results in a well structured fomr, becomes especially important if you run analyses for many tags of the same or different species. It is also recommended that you create a single R script for each analysis (e.g. for each individual and each analysis using different tools). For example, you can name the R scripts using the tag id and the tool e.g. `14SA_SGAT.R`. Since this manual is dealing with tags from different species, the following structure with sub-folders per speces (first three letters of the genus name and the species name) is setup within the main folder (called *data*):

- RawData
  - LanCol
  - MerApi
  - PasCir
- Results
  - LanCol
  - MerApi
  - PasCir
- RCode
  - LanCol
  - MerApi
  - PasCir

You can download the folders with the raw data as well as the annotaded twilight files directly via R and extract into a *data* folder.

```
url <- "https://github.com/slisovski/TheGeolocationManual/raw/master/download/data.zip"

temp <- tempfile()
download.file(url, temp)
unzip(temp, exdir = "data")
```

We also recommend using R Studio and creating a project (File -> NewProject). Alternatively, you can set the working directory using the `setwd` function. With the *data* folder in your project folder (or more in general in your working directory) you should be able to run the code provided in this manual.

We also recommend to use *R Studio* and to create a project (File -> NewProject). Save the project file into the existing *Data* folder. This makes sure that *Data* is your working directory and it will remain the working directory even if the folder moves around on your drive. Alternatively, you can set the working directory using the `setwd` function. With the suggested folder structure and the raw data and the annotaded twiligth files you should be able to run the code provided in this manual.

Chapter 2

Getting started

To analyse light-level geolocator data in R we need a couple of R packages as well as functions that allow to run our code. We created a package called *GeoLocTools* that contains functions that are not nessesarily associated to a certain package but are used in this manual. Importantly the package can also run a check on you system (function: *setupGeolocation()*), detecting packages that are already on your computer and installs the missing tools directly from CRAN or GitHub.

The package requires *devtools* (install if nessesary using the *install.packages()* function). With *devtools* on your system, you are able to download and built as well as install R packages directly from GitHub (e.g. *GeoLocTools*).

```
library(devtools)
install_github("SLisovski/GeoLocTools")
```

You should now be able to load the package and run the `setupGeolocation()` function. We recommend to include this line at the beginning of each script you create for a geolocator analysis. Also check (every now and then), if there is a new version of *GeoLocTools* available. And if that is the case, re-install the package using the same code you used for initial installation.

```
library(GeoLocTools)
setupGeolocation()
```

if you see "You are all set!" in your console, the function ran succesfully and you are able to proceed.

Amongst dependencies, the following geolocator specific packages are loaded by this function:

- twGeos
- GeoLight
- probGLS
- SGAT
- FLightR

- **What the $#@%!#!!!** Although the *GeoLocTools* should make things much easier, it is quite common for problems to arise when setting up your environment. A few frequent and frustrating issues are:
- **Outdated version of R**. If you are not running the latest (or at least a recent) version of R, then some of the packages might not be compatible. Use *sessionInfo()* to see what version of R you are running. You can ususally track down the latest version of R at the R project webpage: www.r-project.org.

Note that you may have to reinstall all of your packages when you get a new version of R. So expect to spend a few minutes on the update.)

- **Missing libraries**. Some packages require that you have specific sofware libraries installed an accessible on your system. if you get a message like "configure: error: geos-config not found or not executable," you may be missing a library. Dealing with these issues may require some use of the Bash or Unix shell to install or locate a library. You can often find instructions for intalling new libraries by searching the internet, but if you do not feel comfortable installing stuff with the command line or you do not have permission to do so, you will probably need to seek some assistance from someone with IT credentials.
- **Typos**. Probably the most common error in R arises simply from typos. Even published scripts or manuals like these may contain small typos that prevent your script from running.

Chapter 3

Loading data

The first step is to load your raw data into R. Different geolocator types (e.g. from different manufacturers or different series) provide raw data in different formats. And while there are functions available to read a whole range of formats, you may have to either write your own function, use simple read text utilities or get in touch with the package managers to write code that fits your format if it is not yet implemented.

The most frequently used geolocators provide files with the extension `.lux` (Migrate Technology Ltd), `.lig` (BAS, Biotrack) or `.glf` (Swiss Ornithological Institute). The functions `readMTlux`, `ligTrans` and `glfTrans` allows you to read these files. The documentations of the different packages may help to provide information on how to read other files (e.g. ?GeoLight). In most cases the raw data is stored in a text file that can also be read in to R using the base function `read.table()`.

A short note on ***naming and saving of data files*** (final results and intermediate steps): We have already discussed, that it makes sense to have a certain folder structure for the analysis of geolocators. It not only helps to keep track of all files and analysis, but most importantly it allows to run the same code for saving and reading of data once you defined a set of metadata information.

With the suggested data structure, we can then define metadata information on the individual, the species, the deployment location, and define the sub-folder for saving and extracting data files.

```
ID <- "14SA"
Species <- "MerApi"

lon.calib <- 11.96
lat.calib <- 51.32

wd <- "data"
```

By using the above metadata we can use the `paste0` command to include this information in reading and writing of files.

```
raw <- glfTrans(paste0(wd, "/RawData/", Species, "/", ID, ".glf"))
  names(raw) <- c("Date", "Light")
  raw$Light  <- log(raw$Light+0.0001) + abs(min(log(raw$Light+0.0001)))
head(raw)
```

```
                 Date Light
1 2015-07-10 00:00:00     0
2 2015-07-10 00:05:00     0
3 2015-07-10 00:10:00     0
4 2015-07-10 00:15:00     0
5 2015-07-10 00:20:00     0
6 2015-07-10 00:25:00     0
```

In this case it is required log transform the light data. In addition, we add a small value since the night readings are sometimes smaller than zero, values that cannot be log transformed.

Adding to the confusion of different raw data types, the read functions also provide different output. However, the most important columns are,

1. Date
2. Light

and these columns need to be in a specific format with Date being a `POSIXc` class and Light being `numeric` integers. Check if the structure of your data follows the required format with the function `str`. If not adjust Date format with `as.POSIXct(raw$Date, tz = "GMT")`.

```
str(raw)
```

```
'data.frame':   112161 obs. of  2 variables:
 $ Date : POSIXct, format: "2015-07-10 00:00:00" "2015-07-10 00:05:00" ...
 $ Light: num  0 0 0 0 0 0 0 0 0 0 ...
```

Do I need to log-transform my raw light measurements?

Log-transformation of the light intensities is helpful to visualise and inspect the data and for the twilight annotation process. It allows to focus at the low light values while seeing the whole light curve and thus makes sense for the tags that measure the full light spectrum (e.g. tags from Migrate Technology Ltd. and from the Swiss Ornithological Institute). If you proceed to analyse your data with FLightR, where you need the raw light intensities, there is no need to back-transform you light data as FLightR will do that automatically.

Chapter 4

Twilight Annotation

There are a few options for how to define and edit twilights.

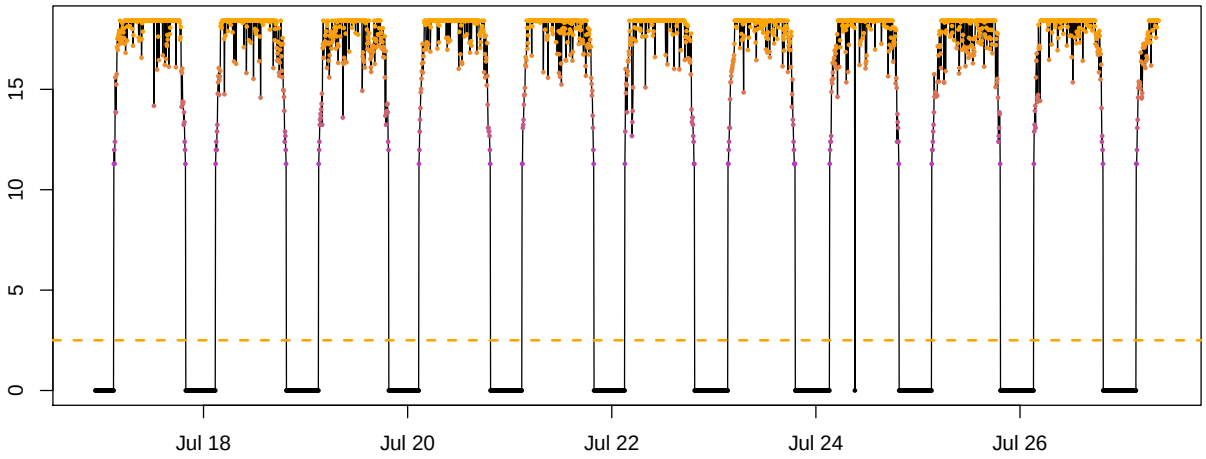
All tools discussed in this manual require as one of their inputs a data frame containing the times of sunrise and sunset (henceforth twilight events) for the duration of the study period. The twilight events are estimated based on a light-level threshold, which is the light value that separates day from night - values above the threshold indicate the sun has risen and values below the threshold value indicate the sun has set. There are a few options for how to generate the twilight data. `twilightCalc` is one function that allows transitions to be defined and is part of the GeoLight package. Given the much better realisation of this process in TwGeos, we will not discuss the GeoLight version of defining twilights. `TwGeos` provides an easier to use and more interactive process that is called `preprocessLight`. An important input, besides the raw data, is a pre-defined light intensity threshold value.

How do I choose the right threshold?

How do I know which threshold to use: You should choose the lowest value that is consistently above any noise in the nighttime light levels. Here, we use a threshold of 2.5, that is above any nighttime noise. However, this value is tag and species specific. For forest interior, ground dwelling species a lower threshold may be helpful, especially if there isn't much 'noise' during the night. A threshold of 1 may be appropriate for such species.

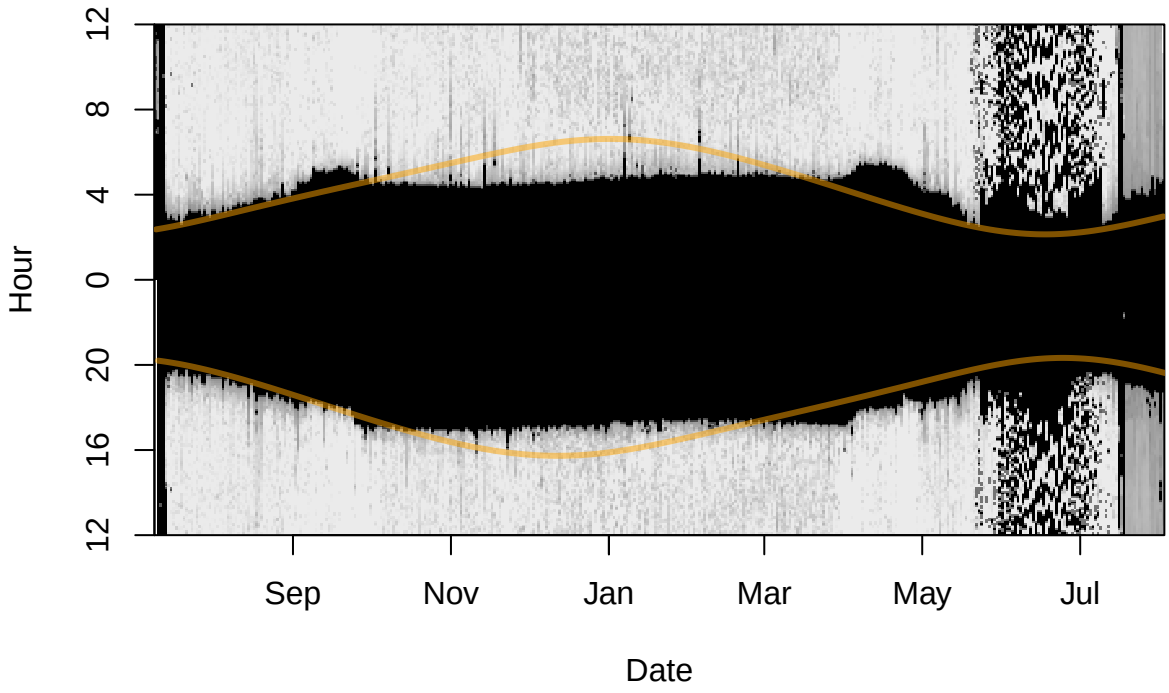
It is a good idea to plot (parts) of the dataset and see how the threshold fits into the light recordings:

```
threshold <- 2.5

col = colorRampPalette(c('black',"purple",'orange'))(50)[as.numeric(cut(raw[2000:5000,2],breaks = 50))]

par(mfrow = c(1, 1), mar = c(2, 2, 2, 2) )
with(raw[2000:5000,], plot(Date, Light, type = "o", pch=16,  col = col, cex = 0.5))
abline(h=threshold, col="orange", lty = 2, lwd = 2)
```



Another useful plot can be created using lightImage; In the resulting figure, each day is represented by a thin horizontal line that plots the light values as grayscale pixels (dark = low light and white = maximum light) in order from bottom to top. a A light image allows you to visualize an entire data set at once, and easily spot discrepancies in light to dark transitions. Additionally, you can add the sunrise and sunset times of the deployment or retrieval locaitons (using `addTwilightLine`). This may help to spot inconsistncies in the dataset, e.g.: *time shifts - resulting in a good overlap of twilight times at the beginning but a systematic shift between expected and recorded twilight times.* false time zone - if the predicted sunrise and sunset times are shifted up- or downwards it is highly likely that your raw data is not recorded (or has been transformed) in GMT (or UTC). Check with producer or data provider. Furthermore, the lines can help to identify the approximate timing of departure and arrival to the known deployment or retrieval site and this may help to identify calibration periods that are required in the next steps of the analysis.

```
offset <- 12 # adjusts the y-axis to put night (dark shades) in the middle

lightImage( tagdata = raw,
  offset = offset,
  zlim = c(0, 20))

tsimageDeploymentLines(raw$Date, lon = lon.calib, lat = lat.calib,
                       offset = offset, lwd = 3, col = adjustcolor("orange", alpha.f = 0.5))
```



Depending on the tag type, geolocator data are automatically adjusted for clock drift by the manufacturer, or, can be easily corrected by comparing the internal device time and real time when data is downloaded. For practical reasons, clock drift in geolocators is assumed to occur at a constant rate. If geolocator data are affected by clock drift the longitude estimates during stationary periods will drift continuously in one direction. In case the tag had stopped recording before data download or the internal time stamp is obviously incorrect, clock drift can be adjusted during the process of locations estimation. In short, an estimated clock drift is added to the twilight data and longitudinal positions are (re)calculated, e.g. using a best-guess sun elevation angle. Clock drift is adequately corrected for, if the slope of a linear regression between longitude and time during stationary periods is zero, showing that there is no directional changes in longitude over time anymore. Latitude estimates are negligibly affected due to the small difference in shifting sunrise and sunset times within the same day.

In the next step, we want to define daily sunrise and sunset times. `preprocessLight` is an interactive function for editing light data and deriving these twilight times Note: if you are working on a Mac you must install Quartz first (https://www.xquartz.org) and then set gr.Device to "x11" in the function. If you are working with a virtual machine, the function may not work at all. Detailed instructions of how to complete the interactive process can be found by running the following code:

```
?preprocessLight
```

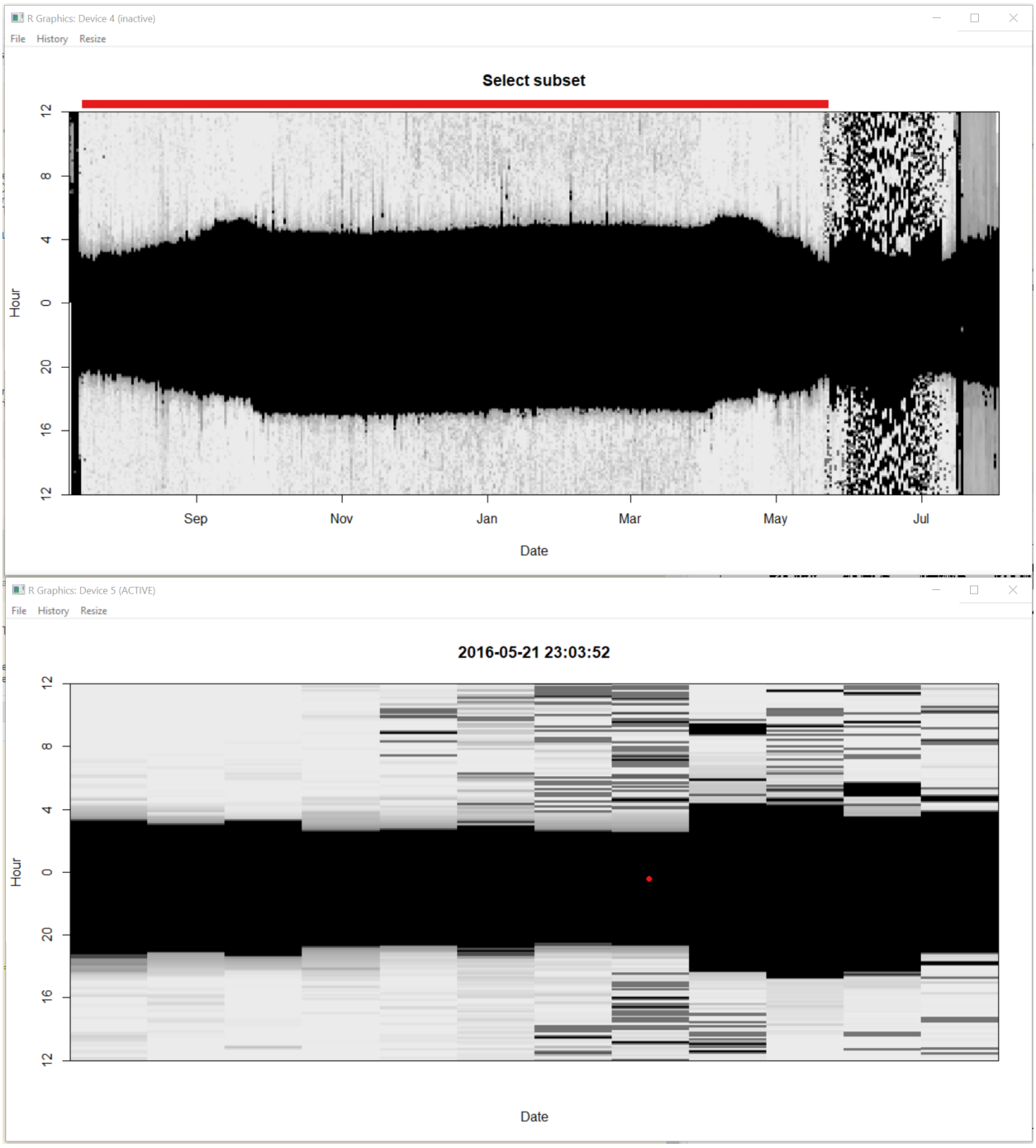
Below, we explain the major functionalities.

When you run,

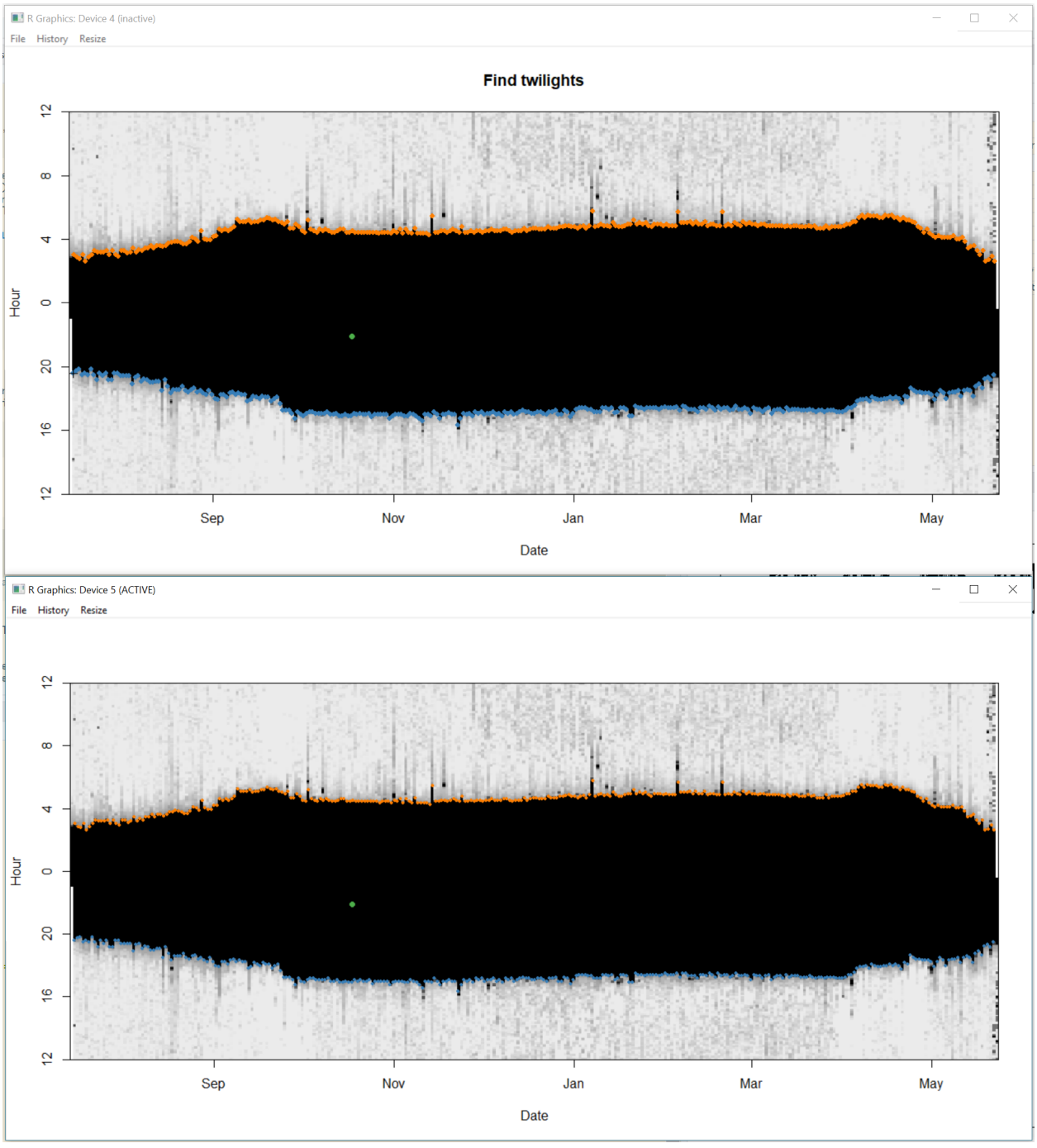
```
twl <- preprocessLight(raw,
  threshold = threshold,
  offset = offset,
  lmax = 20,          # max. light valu
  gr.Device = "x11") # MacOS version (and windows)
```

two windows will appear. Move them so they are not on top of each other and you can see both. They should look like a big black blob. This identifies the "nightime" period over time. The top of the blob shows all the sunrises and the bottom of blob shows all the sunsets. You can note for instance that the days get longer (and thus the nights shorter) at the end of the time series, because the blob gets thinner. You may even note changes in the light image that relate to changes in activity patterns or breeding behavior.

Step 1. Click on the window entitled "Select subset". With the left mouse button choose where you want the start of the dataset to be, and right mouse button to choose the end. You will notice that the red bar at the top moves and that the second window zooms into that time period. Select when you want your time series to start and end. This allows you to ignore for instance periods of nesting. Once you are happy with the start and end of the timeseries press "a" on the keyboard to accept and move to next step.



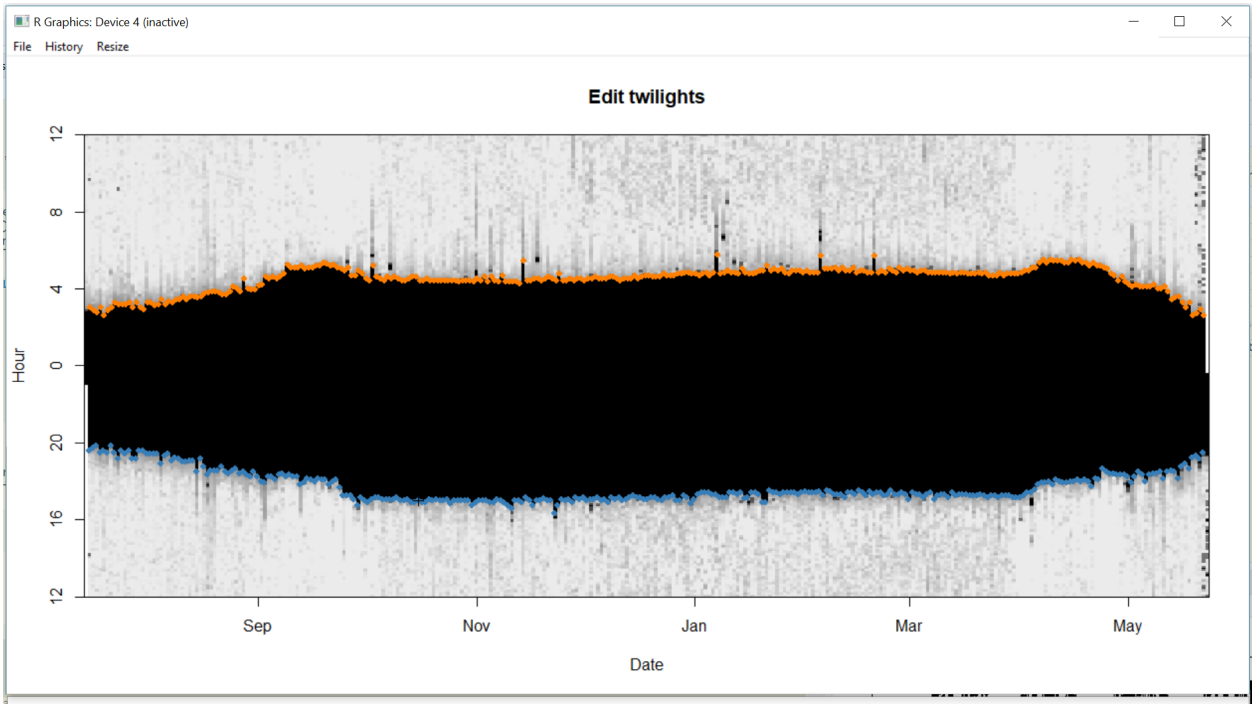
Step 2. click on the window entitled "Find twilights" and the second window will zoom in. All you need to do here is click in the dark part (in the zoomed in image i.e. the one not entitled "Find twilights") of the image and this will identify all the sunrises (orange) and sunsets (blue) based on the threshold defined in the previous section. Press "a" on the keyboard to accept and move to next step.

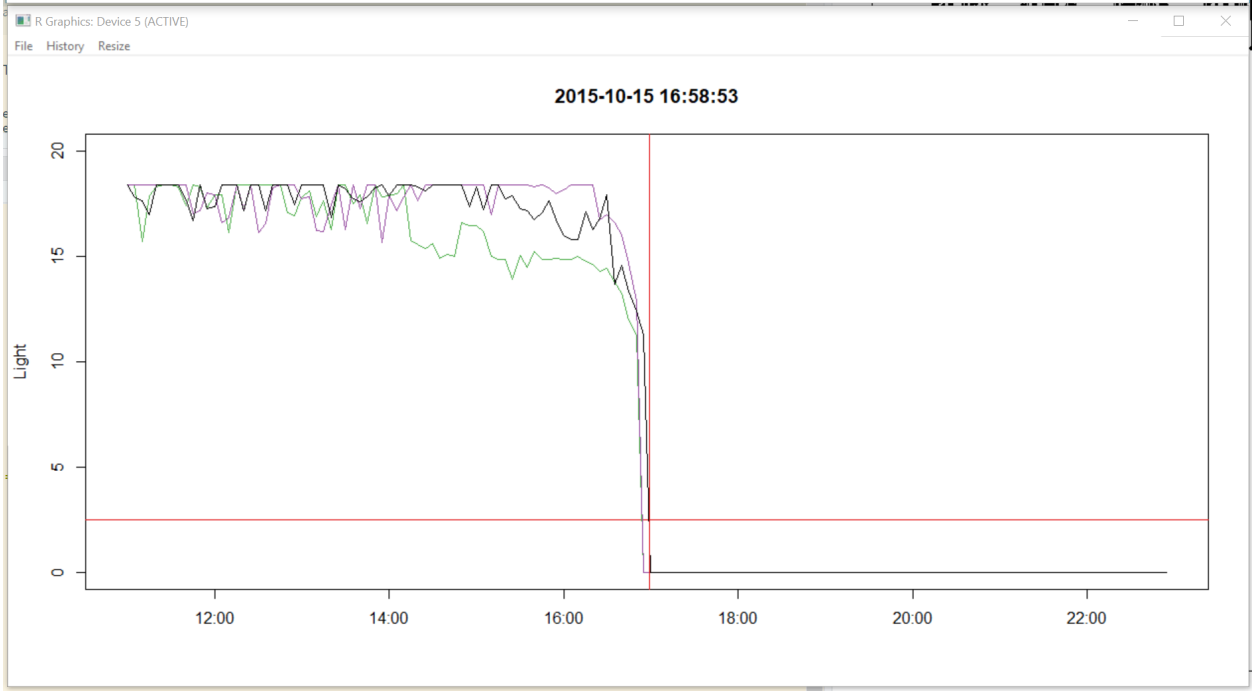


Step 3. This step is for adding or deleting points. If there are no missing data points, you can skip this step by pressing "a" on the keyboard. However, if you do want to add a point, you can click on the "Insert twilights" window to select a region of "the blob" that the second unintitled window will zoom into. In the zoomed window, use left mouse click to add a sunrise, and right mouse click to add a sunset. You can use "u" on the keyboard to undo any changes, and "d" to delete any points which are extra. Press "a" to move to next step.

Step 4. This step allows you to find points which have been miss-classified (often because the bird was in the shade or in a burrow) and to move the respective sunrise or sunset to where it should be. Choose a point by clicking on it in the "edit twilights" window and the other window will display the sunrise (or sunset) from

the previous and next days (purple and green) relative to the current sunrise or sunset (in black). Thus if the black line shows a much earlier sunset or later sunrise than the purple and green ones, it is likely badly classified. . You can then left click at the point where you want the day to start and press “a” to accept and move the sunrise or sunset. You will notice the red line then moves. Do this for as many points as necessary.





Then close the windows with “q”.

How important is it to edit twilights?

If you have no a priori reason and criteria to strongly edit twilight events, it is generally better to be a bit conservative with editing. This prevents that data are changed into an unwanted direction, e.g. erroneously removing good data points (amidst shading events), or informative events such as strong movements. Also the criteria to edit or remove badly classified twilights will be different depending on the method you use to infer locations. For curve methods, similarity in the shape of the curve around sunrise or sunset is most important, while for threshold methods the similarity in the sunrise and sunset events itself is important.

Have a look at the output

```
head(twl)
```

```
             Twilight  Rise Deleted Marker Inserted           Twilight3
1 2015-07-15 19:34:02 FALSE   FALSE      0    FALSE 2015-07-15 19:34:02
2 2015-07-16 03:01:00  TRUE   FALSE      0    FALSE 2015-07-16 03:01:00
3 2015-07-16 19:43:53 FALSE   FALSE      0    FALSE 2015-07-16 19:43:53
4 2015-07-17 02:51:06  TRUE   FALSE      0    FALSE 2015-07-17 02:51:06
5 2015-07-17 19:48:53 FALSE   FALSE      0    FALSE 2015-07-17 19:48:53
6 2015-07-18 02:46:06  TRUE   FALSE      0    FALSE 2015-07-18 02:46:06
  Marker3
1       0
2       0
3       0
4       0
5       0
6       0
```

The output contains the following important information:

- **Twilight**
- The date and time of the sunrise/sunset events
- **Rise**
- whether the Twilight is a sunrise (TRUE) or a sunset (FALSE)
- **Deleted**
- whether you marked this twilight with a "d", that means it is still in the file and can/should be exluded later on.
- Marker (see detailed description in `?preprocessLight`)
- Inserted (whether this Twilight was manually inserted)
- Twilight3 (the original Twilight. Only different to Twilight if you edited the timing)

Other processes like `twilightCalc` or the software `TAGS` produce different outputs but it is preferred to get them into this format (at least with the columns `Twilight` and `Rise`), since you can go ahead with any analysis you want using these two columns (***note: do not save these two columns only, since the other information is important to reproduce your analysis***).

Save the output file as a .csv file, so that you never have to do this step again.

To save this file we use the metadata variables that were defined above:

```
write.csv(twl, paste0(wd, "/Results/", Species, "/", ID, "_twl.csv"), row.names = F)
```

This can later be loaded using the following code (note, that you have to define the class type `POSIXC` for the date):

```
twl <- read.csv(paste0(wd, "/Results/", Species, "/", ID, "_twl.csv"))
twl$Twilight <- as.POSIXct(twl$Twilight, tz = "GMT") # get the Twilight times back into the POSIX. clas
```

The result of this first part that is **independent** of which package/analysis will be used next is the twilight file that should at least look like (can have more columns):

```
head(twl[,c(1,2)])
```

```
             Twilight  Rise
1 2015-07-15 19:34:02 FALSE
2 2015-07-16 03:01:00  TRUE
3 2015-07-16 19:43:53 FALSE
4 2015-07-17 02:51:06  TRUE
5 2015-07-17 19:48:53 FALSE
6 2015-07-18 02:46:06  TRUE
```

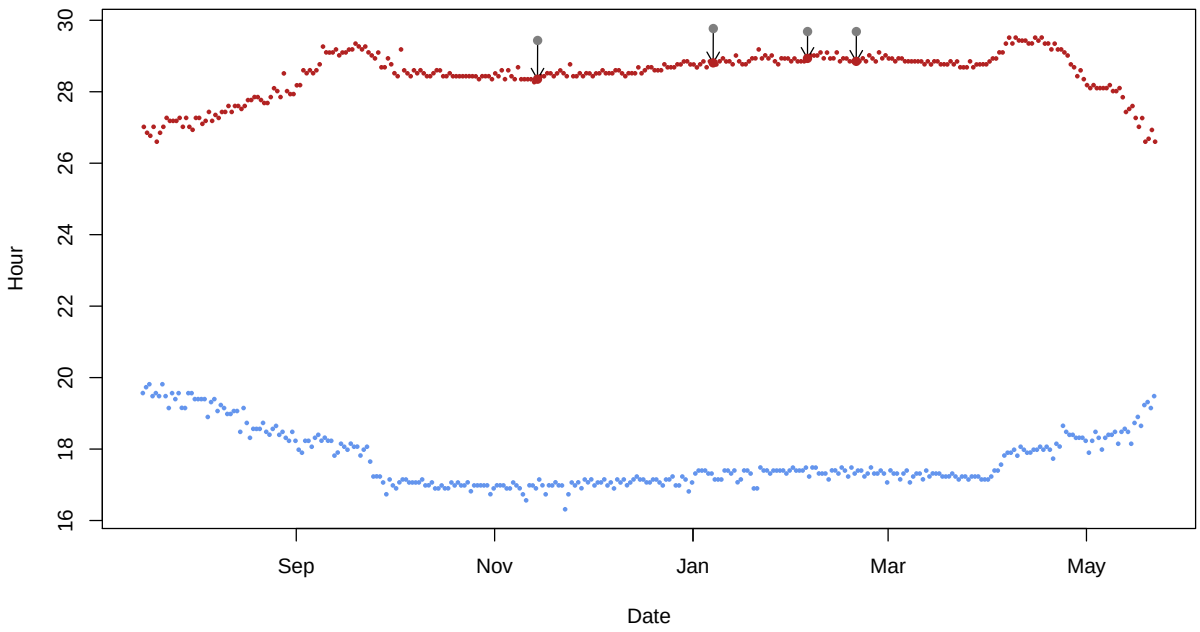
Cleaning/Filtering twilight times

Automated filtering of twilight times should be handled carefully. There is no *perfect* function that cleans your twilight file. However, `twilightEdit` can help to filter and remove (mark them as deleted) outliers (e.g. false twilights). The filtering and removing of twilight times is based on a set of rules:

1) if a twilight time is e.g. 45 minutes (**outlier.mins**) different to its surrounding twilight times, and these sourrounding twilight times are within a certain range of minutes (**stationary.mins**), then the twilight times will be adjusted to the median of the surrounding twilights.
2) if a twilight time is e.g. 45 minutes (**outlier.mins**) different to its surrounding twilight times, but the surrounding twilight times are more variable then you would expect them to be if they were recorded during stationary behavior, then the twilight time will be marked as deleted.

The argument **windows** defines the number of twilight times surrounding the twilight in focus (e.g. same as in conventional moving window methods).

```
twl <- twilightEdit(twilights = twl,
                    offset = offset,
                    window = 4,           # two days before and two days after
                    outlier.mins = 45,    # difference in mins
                    stationary.mins = 25, # are the other surrounding twilights within 25 mins of one a
                    plot = TRUE)
```



In this particular case and with the parameters, four twilight times have been corrected. Based on the output, you can also exclude them for further analysis. While you can also save the output file, we recommend archiving the twilight file from above and redo the `twilightEdit` after reading in the archived twilight file from above.

This method helps to adjust and remove twilight times that are either outliers or false twilights given a set of rules. While subjective to a certain degree as well as reproducible, the method may not be able to detect all false twilight times and may even remove correct entries during fast migration periods.

Chapter 5

GeoLight

GeoLight uses the threshold method to estimate simple discrete locations per set of twilight events. It was developed in 2012 with the goal of being a complete, quick, and reproducible method of geolocator analysis Lisovski & Hahn 2012. Over time, *GeoLight* has been further developed and functions such as `mergeSites`, `mergeSites2` and `siteEstiamte` were added. These functions make use of a very simple movement analysis that aims to separate periods of movement from periods of residency by finding changes in the recorded sunrise and sunset times. Investigating entire stationary periods and estimating a single location (e.g., for all sunrise and sunset times during the major non-breeding period when the bird was stationary) using a optimization procedure (maximum likelihood) we can both, refine location estimates and estimate credible intervals around the most likely location. Even with these new functions, *GeoLight* is still a tool that uses simple principles and requires low computing capacity. It also allows for a quick analysis (that should be thoroughly checked if used for publications) and is thus an analysis tool by itself but can also be used as an initial step before going into the more sophisticated and complex approaches like SGAT or FLightR.

Getting started

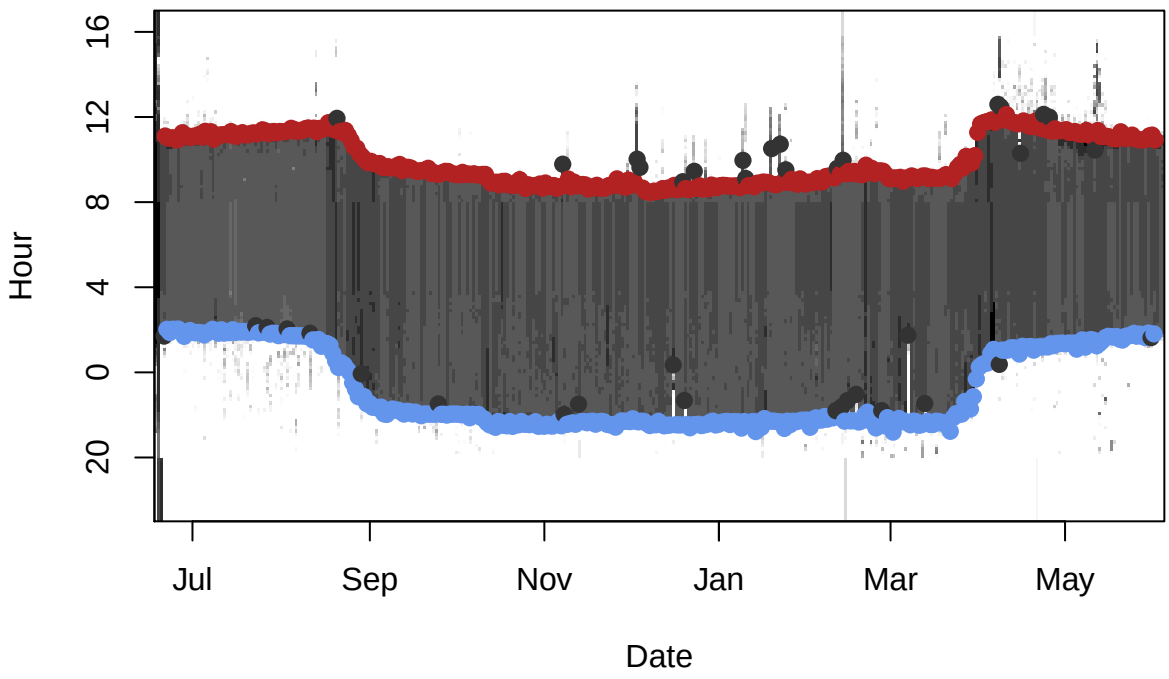
To illustrate the *GeoLight* analysis, we use the Purple martin dataset.

We first define the metadata and read in the raw recordings. We skip the twilight definition process but read in the file that has been generated using `preprocessLight`.

```
Species <- "PasCir"
ID      <- "PasCir01"

lat.calib <-  33.9
lon.calib <- -96.8

wd <- "data"

raw <- readLig(paste0(wd, "/RawData/", Species, "/", ID, ".lig"))
  raw$Light <- log(raw$Light)

twl <- read.csv(paste0(wd, "/Results/", Species, "/", ID, "_twl.csv"))
  twl$Twilight <- as.POSIXct(twl$Twilight, tz = "GMT")
```

Let's have a look at the dataset using the `lightImage` function from *TwGeos*.

```
offset <- 17 # adjusts the y-axis to put night (dark shades) in the middle

lightImage( tagdata = raw,
            offset = offset,
            zlim = c(0, 4))

tsimagePoints(twl$Twilight, offset = offset, pch = 16, cex = 1.2,
              col = ifelse(twl$Deleted, "grey20", ifelse(twl$Rise, "firebrick", "cornflowerblue")))
```



As you can see, there are many twilight events that are marked as deleted. Therefore, we have to subset out twilight table.

```
twl <- subset(twl, !Deleted) # only rows that are not marked as deleted.
```

GeoLight requires a certain input format of the twilight table that differs from the ouput of e.g. `preprocessLight` or *TAGS*. We need to have rows that always have a set of twilight times, e.g. sunrise and sunset of a day or sunset and sunrise of a night. The first columns (and the first twilight) is called *tFirst*, the second *tSecond* and the third column *type* defines whether it is a day (1) or a night (2). The function `export2GeoLight` can transform the table from above (twl) into the required format.

```
twl.gl  <- export2GeoLight(twl)
head(twl.gl)
```

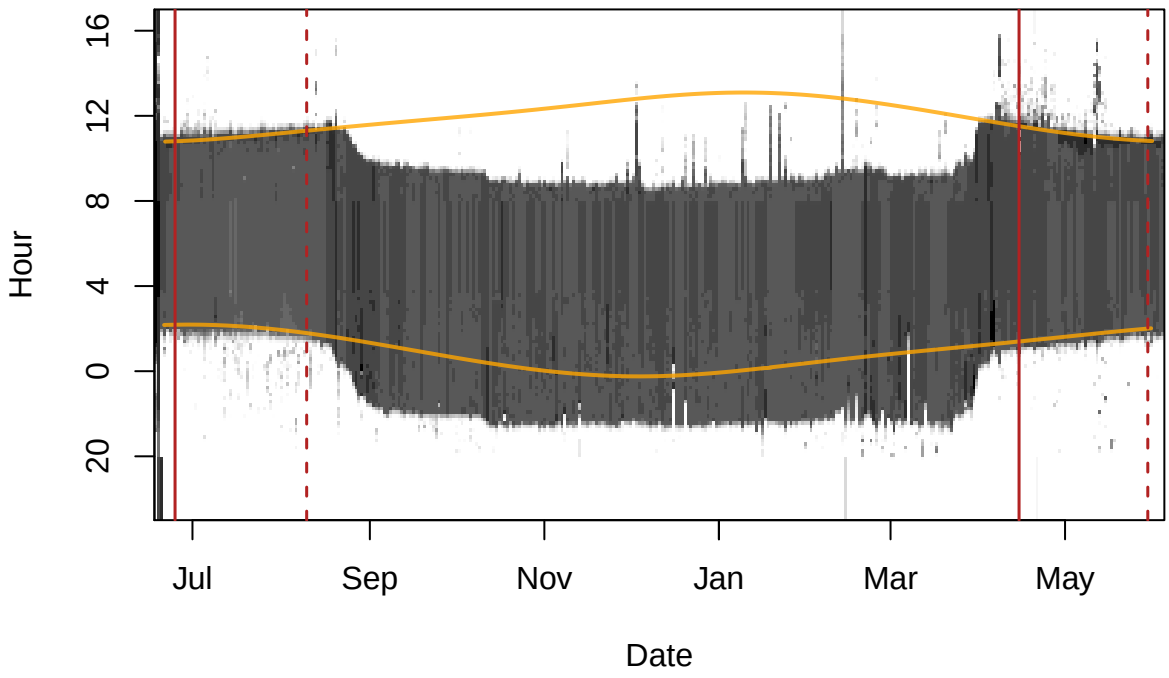
```
               tFirst             tSecond type
1 2011-06-21 11:06:22 2011-06-22 02:01:18    1
2 2011-06-22 02:01:18 2011-06-22 11:00:25    2
3 2011-06-22 11:00:25 2011-06-23 01:52:21    1
4 2011-06-23 01:52:21 2011-06-23 11:01:08    2
5 2011-06-23 11:01:08 2011-06-24 02:01:44    1
6 2011-06-24 02:01:44 2011-06-24 11:01:08    2
```

Calibration

From the image above, we see that there are two clear periods when the birds has been at the release/recapture site. We can use either one period or both. Given that calibration is best if we use as many twilight times as possible, we here use both periods. We again use the `ligthImage` and plot the sunrise/sunset curve of the deployment site. Then we play with the dates and add lines until we are satisfied with the calibration periods.

It is most important to not include any periods when the birds has not been at the known location since that can totally screw the calibration. Thus, it is preferred have a buffer to the departure/arrival.

```
lightImage( tagdata = raw,
            offset = offset,
            zlim = c(0, 4))

tsimageDeploymentLines(twl$Twilight, lon.calib, lat.calib, offset = offset,
                       lwd = 2, col = adjustcolor("orange", alpha.f = 0.8))

tm1 <- c(as.POSIXct("2011-06-25"), as.POSIXct("2011-08-10"))
tm2 <- c(as.POSIXct("2012-04-15"), as.POSIXct("2012-05-30"))

abline(v = tm1, lty = c(1,2), col = "firebrick", lwd = 1.5)
abline(v = tm2, lty = c(1,2), col = "firebrick", lwd = 1.5)
```



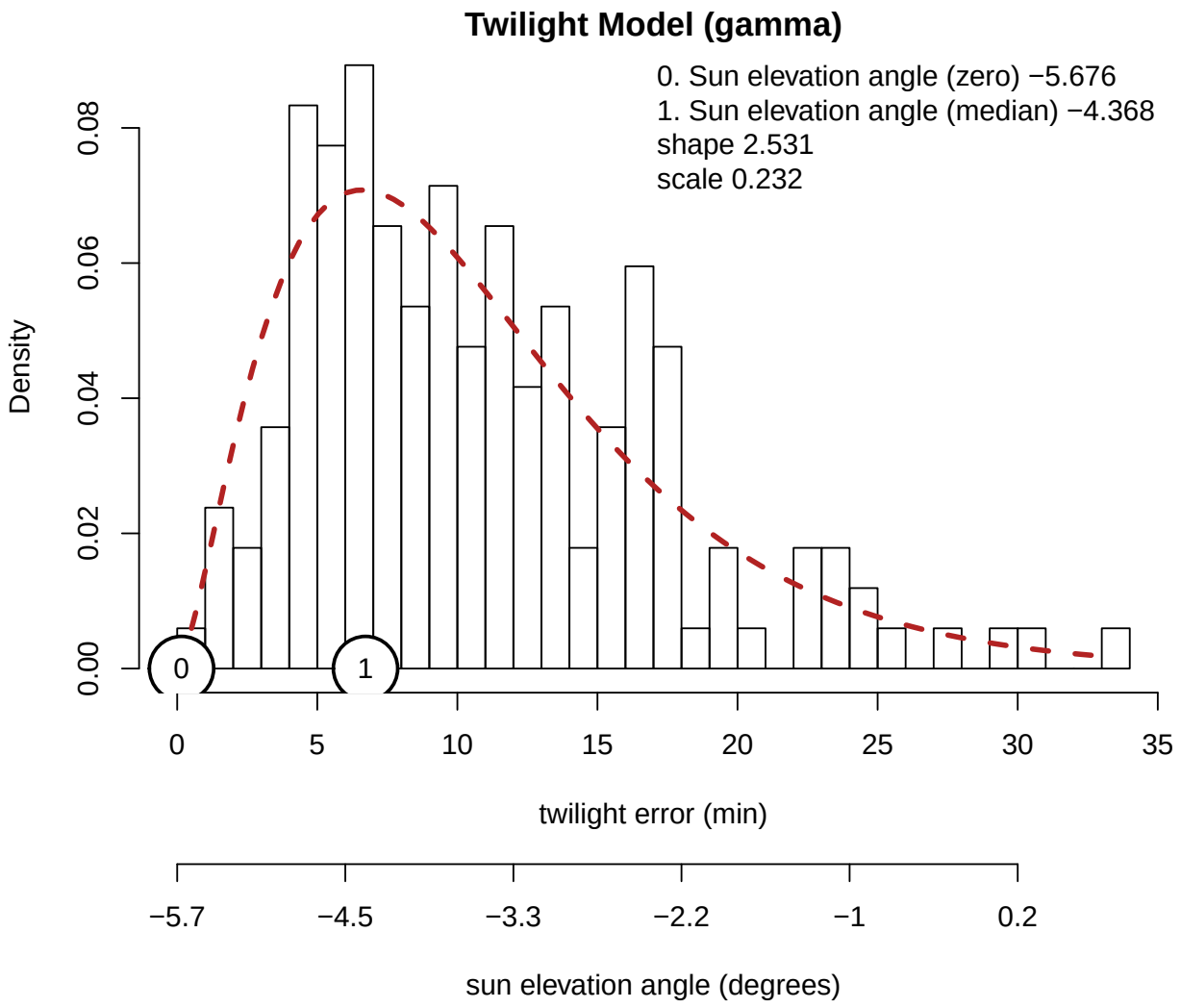
We can now subset the twilight table `twl.gl`.

```
d.calib <- subset(twl.gl, (tFirst>=tm1[1] & tSecond<=tm1[2]) |
                          (tFirst>=tm2[1] & tSecond<=tm2[2]))
```

The method behind the calibration function `getElevation` in *GeoLight* has changed over time and is now based in the error distribution (the variation) of the detected twilight times during the calibration period.

```
gE      <- getElevation(twl = d.calib, known.coord = c(lon.calib, lat.calib),
                        method = "gamma")
gE
```

```
        a1         e0      shape      scale
93.8281938 -5.6759414  2.5305391  0.2317587
```



The figure above represents a nice calibration curve; the twilight error indicating the deviation from the true twilight events in minutes follows quite nicely a gamma distribution (the red dotted line). The function provides four numbers. The first one is the reference sun elevation angle (the round dot with the 1) that can be used to calculate the threshold locations. This reference angle is based on the median of the twilight error distribution, minimizing the accuracy of the location estimates (not the precision that is affected by the variability in twilight events). The second value in the output is the sun elevation angle that defines the zero deviation and thus the lowest sun elevation angle a twilight could be detected. This sun elevation angle is important in the `mergeSites2` function but is also used in the e.g. *SGAT* analysis.

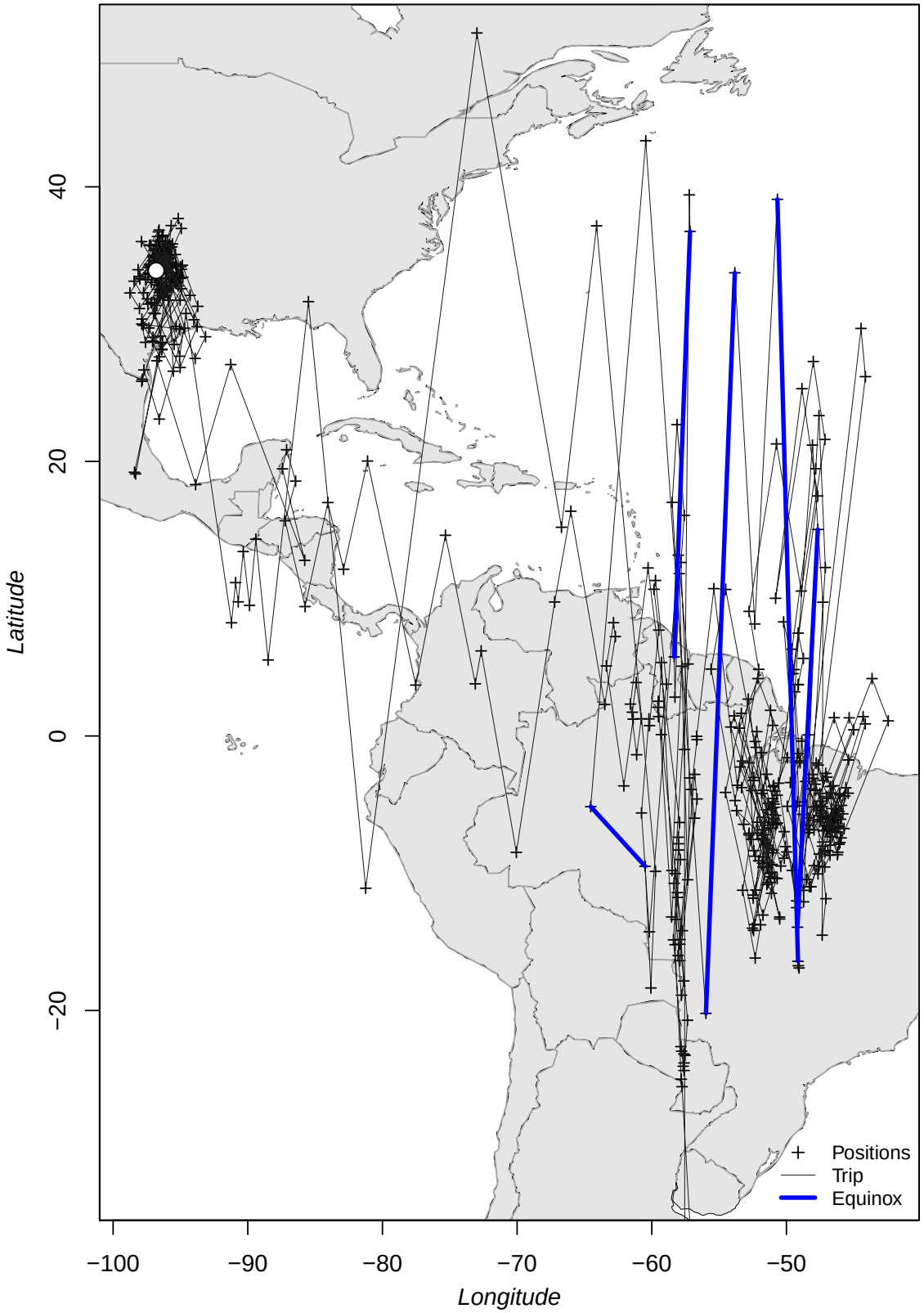
The graph produced by `getElevation` can help to judge whether your calibration data is sufficiently long and whether it is correct. A twilight error that fits the gamma distribution is a good sign, that you have enough data for your calibration. In case you have a few bars only, the fit is poor, and your time series is probably too short. You can also use a log-normal error distribution (method = "log-norm"), in some cases this results in a better fit. Additionally, if you see a few values at zero deviation and a big gap between them and the next junk of data points you have most likely included a period that was not recorded at the known

location or you have falsely defined twilight events that are earlier/later than the logger is able to detect light (go back to the twilight annotation and have a thorough look at the data).

Location estimation

We can now calculate the locations and have a fist look at a map by either using the implemented `tripMap` function or by simply plotting the locations and adding a map.

```
crds <- coord(twl.gl, degElevation = 90-gE[1], note = FALSE)

## using tripMap
tripMap(crds, xlim = c(-98.75, -42.4), ylim = c(-32, 50))
points(lon.calib, lat.calib, pch = 21, cex = 1.5, bg = "white") # adding the release location

## using the plot option (you need to remove the hash in front of the code)
# plot(crds, type = "n") # sets the extent
# plot(wrld_simpl, col = "grey90", border = "grey50", add = T) # adds the map from maptools
# points(crds, pch = 21, cex = 0.5, bg = "white", type = "o")
# points(lon.calib, lat.calib, pch = 21, cex = 1.5, bg = "firebrick") # adding the release location
```


Latitude
40
20
0
−20
−100
−90
−80
−70
−60
−50
Longitude
Positions
Trip
Equinox

ven the crude location estimates of the simple threshold method provide useful information and we get a feeling of the track, the major non-breeding sites and potentially the stopover locations. We also the huge jumps, notably during migration that is most likely influenced by the equinox. However, large north-south jumps are also an indication of rapid east-west movements.

Hill-Ekstrom calibration

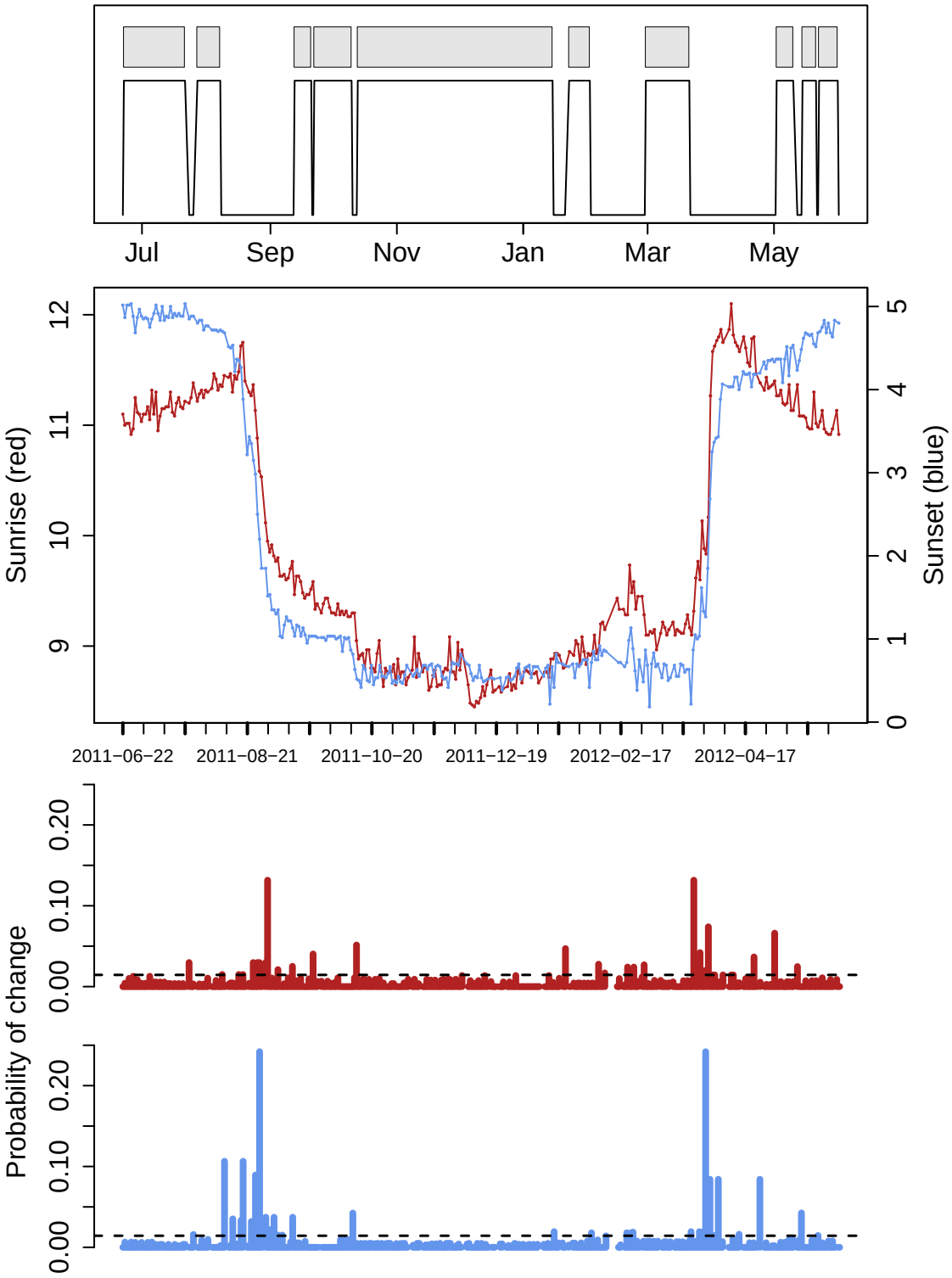
GeoLight has tools to make simple adjustments and refine the location estimates. First, we can use alternative calibration methods to see if the calibration at the known site is actually representative for the twilight error during the entire year. Assuming a fixed sun elevation angle is often not correct. Birds may behave very different during the breeding season and my use different habitat (more open or more in the vegetation). There is a good chance that this is unknown and that we cannot investigate if that is indeed the case and we have therefore make the assumption that it is the same as during the calibration period (simply assuming that is different without evidence from data is not recommended). However, in some cases we can use the s called *Hill-Ekstrom calibration* to estimate a reference sun elevation angle from stationary periods at unknown location. The *Hill-Ekstrom calibration*, is based on the theory that the precision in latitude estimates of a stationary period is highest (lowest variation) if the correct sun elevation angle has been chosen (see Lisovski et al. 2012 for more details). We can thus define stationary periods and estimate latitude using a range of sun elevation angles and see which one result in the lowest variation in latitudes.

GeoLight offers a tool to distinguish between periods of movement and periods of residency. The `changeLight` function uses the twilight times (not the location estimates) that are unaffected by e.g. the equinox and searches for sudden changes that indicate changes in the location of the animal.

If we are simply interested in a long stationary period outside the deployment/release location we can use very conservative parameters, e.g. low values for the quantile. This means that we except changes with a relatively low likelihood (e.g. 0.75).

Play with the settings and you will see how this changes the separation of periods (upper panel)

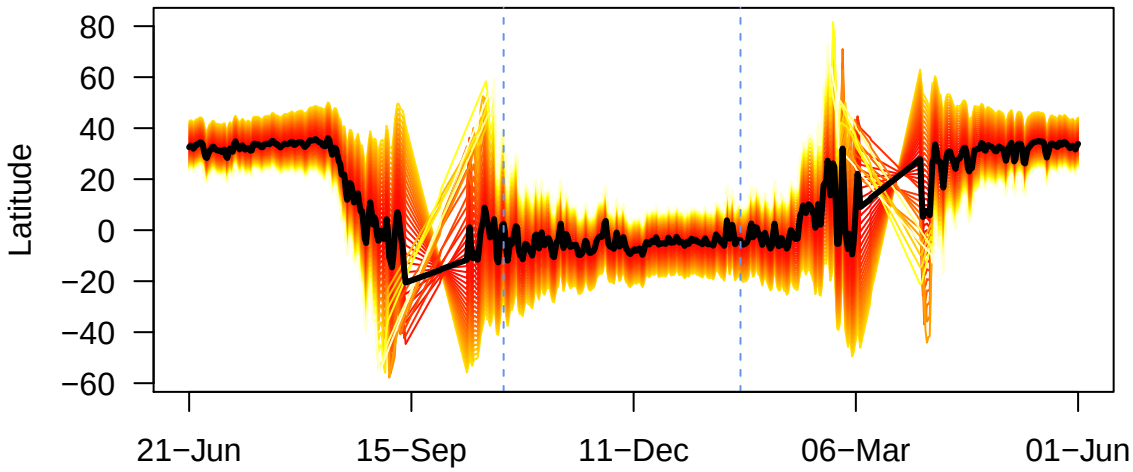
```
cL <- changeLight(twl = twl.gl, quantile = 0.8)
```

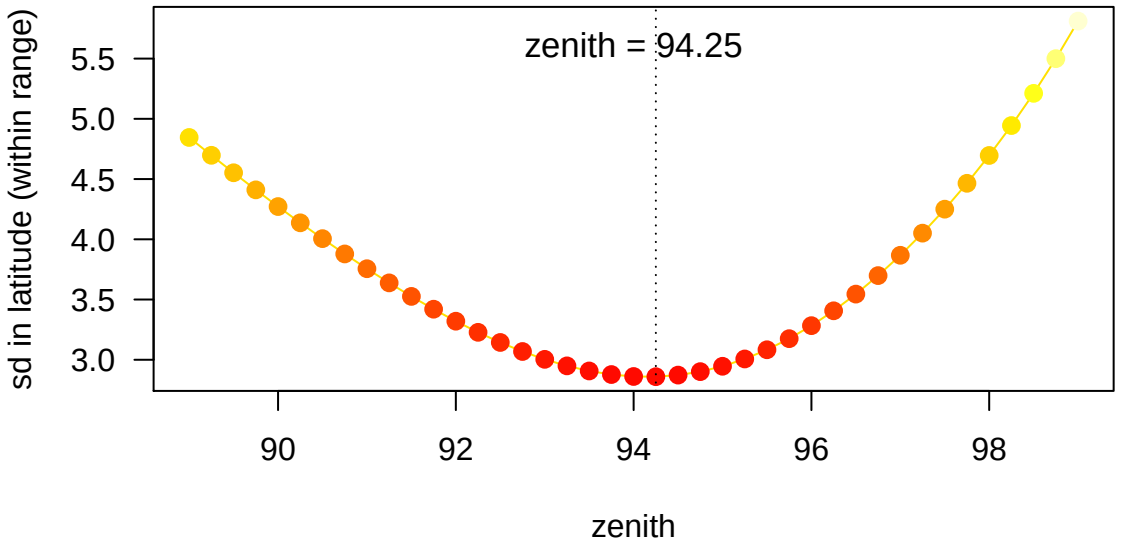


Besides other information, the `changeLight` function returns a vector with the sites `cL$site` that we can use to subset the twilight table for the e.g. longest period (in this case stationary period Nr. 5). **Important**, the function we use to run the *Hill-Ekstrom calibration*, `findHEZenith`, is from the *TwGeos* package and requires the twilight table with all twilight in one column called Twilight, and the information on whether it is a sunrise or a sunset in a second column called Rise, e.g. the output from the `preprocessLight` function.

Using the output of the change-point analysis we can define the start and the end of the long stationary period. It is however recommended to reduce the stationary period by a couple of days at each side. This makes sure that potential movement that can be mis-identified at the transition between real movements and stopover behavior are not part of the analysis.

```
StartEnd <- range(which(twl$Twilight>=(min(twl.gl$tFirst[cL$site==5])+5*24*60*60) &
                        twl$Twilight<=(max(twl.gl$tFirst[cL$site==5])+5*24*60*60)))

HE <- findHEZenith(twl, range = StartEnd)
```





In the plots above, you see the calculated latitudes of the entire tracking period. The latitudes have been calculated using a while range of sun elevation angels. The lower graph has the most important information; the standard deviation of the latitudes from the selected stationary period over the used zenith angle. In this case there is a clear minima (a sign that the *Hill-Ekstrom calibration* is working) at 94.25 degrees. We will discuss the issue of having different references for the sun elevation angle (e.g. sun elevation angle vs. zenith angle) in the SGAT section. Here, we simply transfer the zenith to sun elevation angle: 94.25 = -4.25. That means, that the optimal sun elevation angle for this period is 0.5 degrees higher than the one we estimated

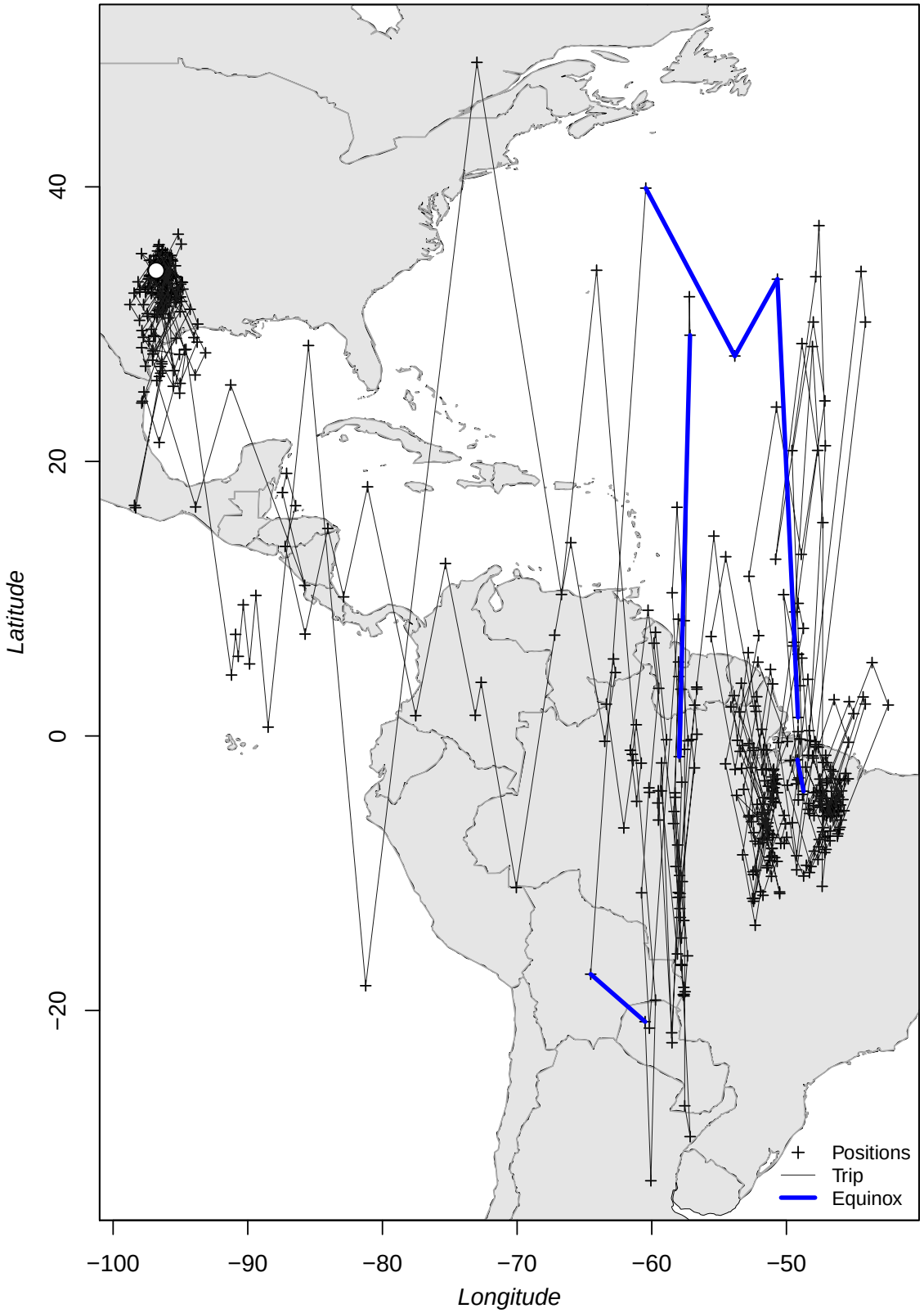
for the calibration period. This is not massive but still a significant difference probably explained by the non-breeding site being close to the equator with higher likelihood of clouds than in e.g. south of the US.

So, let's use this reference sun elevation angle to calculate the locations.

```
crds <- coord(twl.gl, degElevation = 90-HE)
```

```
Note: Out of 592 twilight pairs, the calculation of 25 latitudes failed (4 %)
```

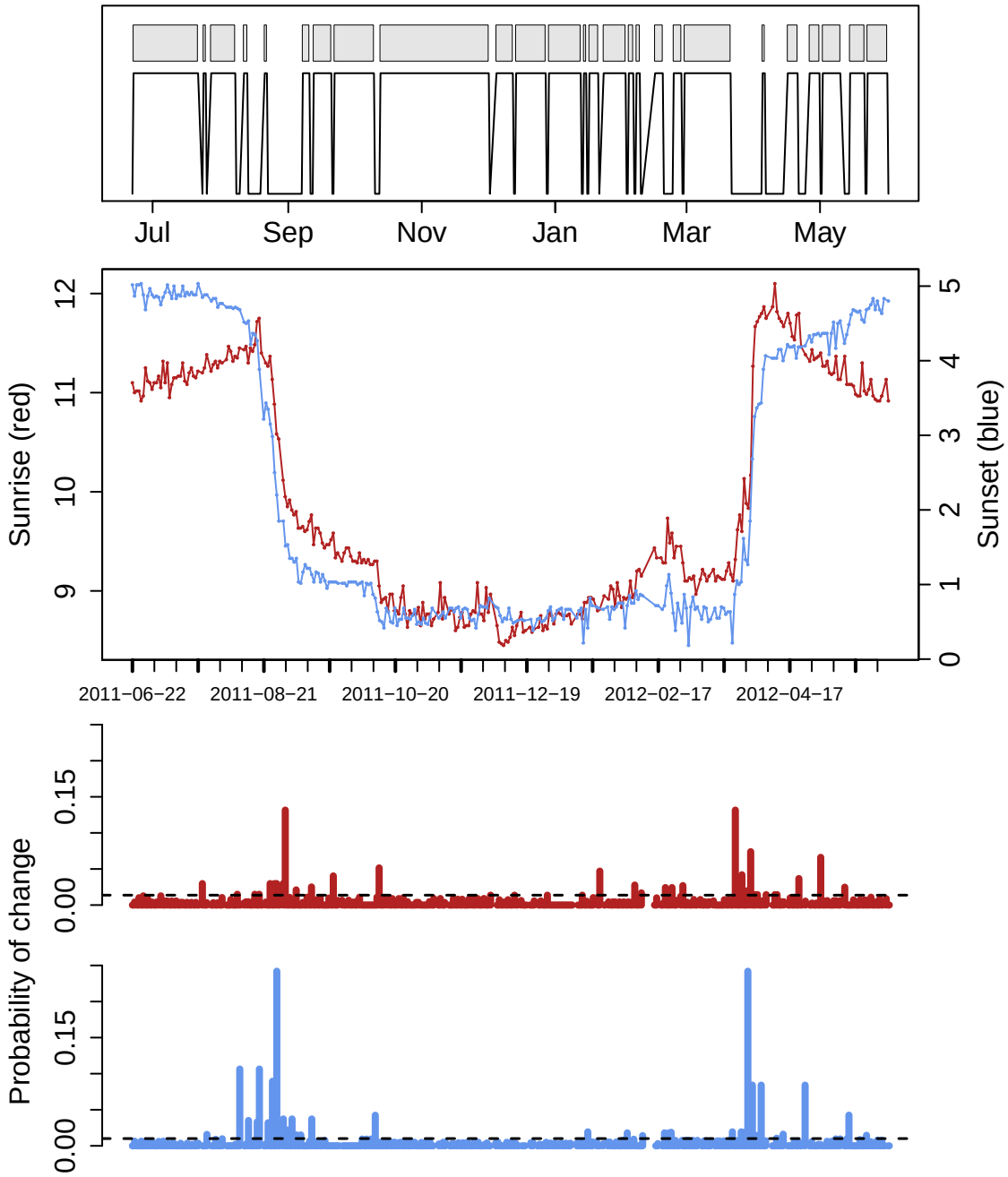
```
## using tripMap
tripMap(crds, xlim = c(-98.75, -42.4), ylim = c(-32, 50))
points(lon.calib, lat.calib, pch = 21, cex = 1.5, bg = "white") # adding the release location
```


Latitude
40
20
0
−20
−100
−90
−80
−70
−60
−50
Longitude
Positions
Trip
Equinox

Movement analysis

We have now put sufficient effort into the calibration and are ready to continue with the movement analysis. We again use the `changeLight` function but it is very important to adjust the settings until you are satisfied, and you are sure that you used a quantile value and a certain number of days (minimum days to define a stationary period), that results in a good separation between movement and resident behavior. Most importantly, you want to make sure that all movement are detected. Don't worry of the analysis detect too many movements, we will deal with that later. The likelihood of changes (lower barplots) are relative to the biggest change. Thus, if you have strong outliers in your twilight table (false twilight times) the analysis is biased towards these outliers and you should go back to the twilight annotation process and remove strong outliers. In some cases the likelihood of changes and the variability in twilight events is very different between sunrise and sunset, if that is the case define a threshold for sunrise and sunset separately (see `?changeLight` for details).

```
cL <- changeLight(twl = twl.gl, quantile = 0.78, days = 1)
```



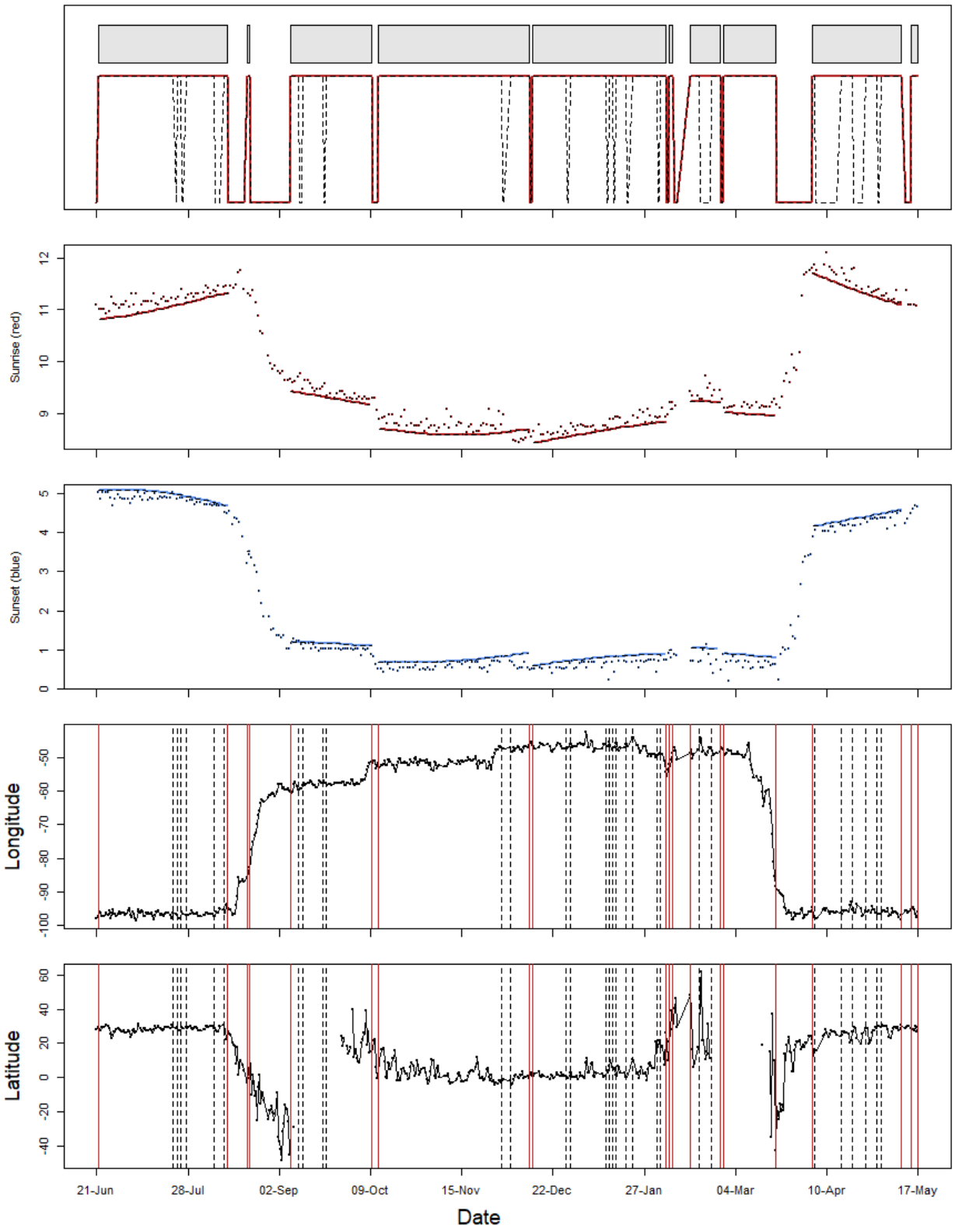
These settings seem to detect all changes that are obviously visible (and more). We can now use `mergeSites` to refine this selection and to merge consecutive sites if they are only separated by single large errors in the twilight times but are otherwise likely to be at the same site. `mergeSite` uses a maximum likelihood fit to optimize longitude and latitude for each stationary period. However, the error term is modeled using a Gaussian distribution and that is certainly wrong. However, the analysis often returns good results and even good estimates of the most likely location and the credible intervals. The `mergeSites2` function is a further development and uses the correct assumptions (it also replaces the function `siteEstimation`). We can use the twilight error parameters and the reference angle that we have calculated using the `getElevation` function. The `mergeSites2` function evaluates the likelihood surface for each stationary period, starting from the first one and compares the most likely location and the 95% credible interval with the most likely location and the credible intervals of the next stationary site. If the most likely locations are smaller than

the `distThreshold` and the 95% credible intervals overlap, the site is merged and the process starts again from the new merged period to the next period. Additionally, we can use a simple masking option to prevent locations from beeing on land or at sea. The function requires some calculation power and can take several minutes to complete.

The function returns the updated vector of the sites as well as the locations estimated using the maximum likelihood approach.

...

```
## may take several minutes to complete
mS <- mergeSites2(twl = twl.gl, site = cL$site,
                  distThreshold = 500,
                  degElevation = gE[2]-0.75,          # the HE corrected zero sun elevation angle
                  alpha = gE[3:4], method = "gamma", # parameters and model of the twilight errro
                  mask = "land")                      # mask option
```

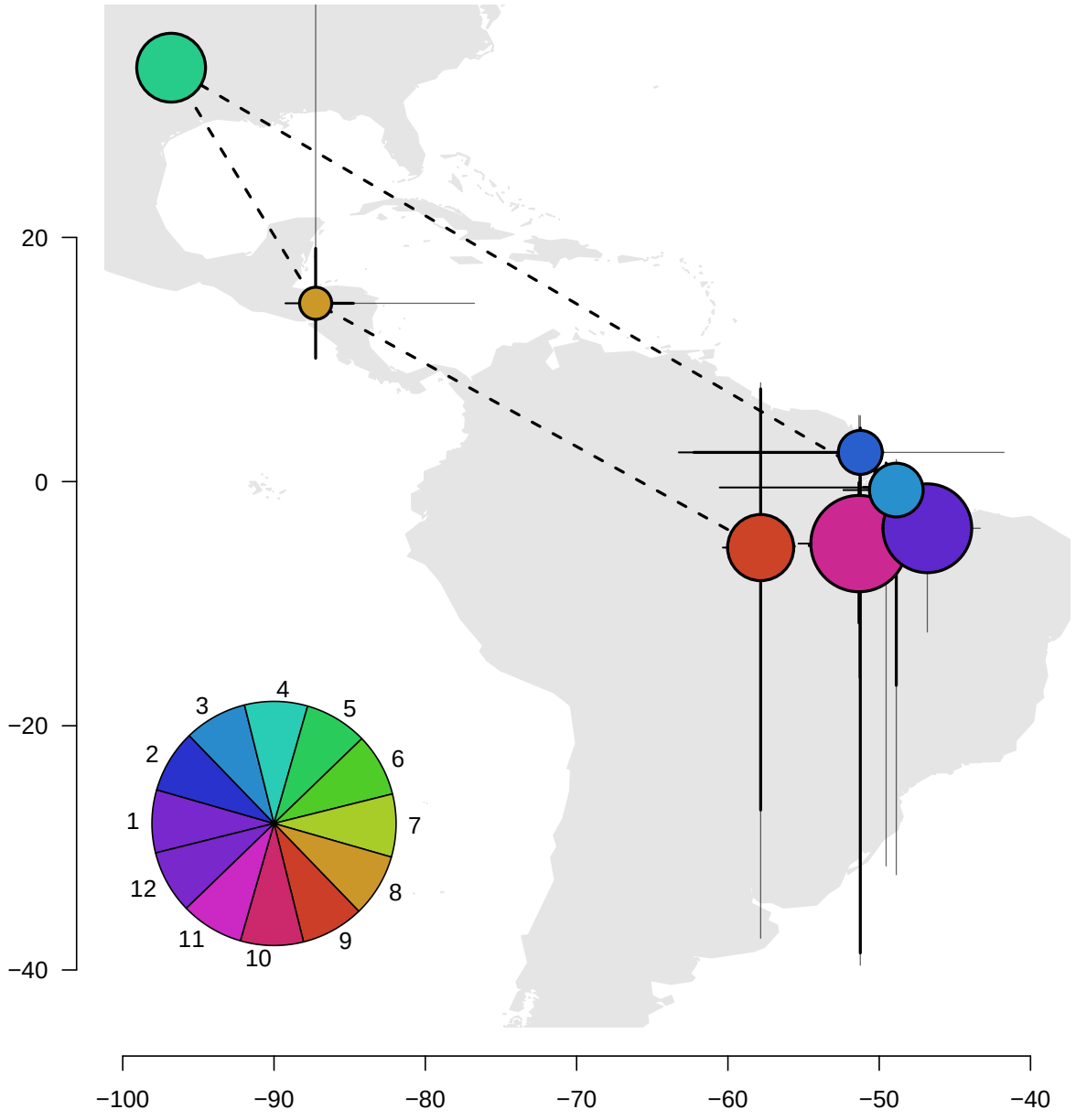


The plot created by `mergeSites2` is similar to what we know from `changeLight`. However, the top panel shows which of the original sites were merged (red lines around dotted lines), and the same is shown for the

longitude and latitude estimates in the two bottom panels. The middle panel shows the sunrise and sunset times recorded by the geolocator and the fitted sunrise and sunset times (lines) of the most likely location.

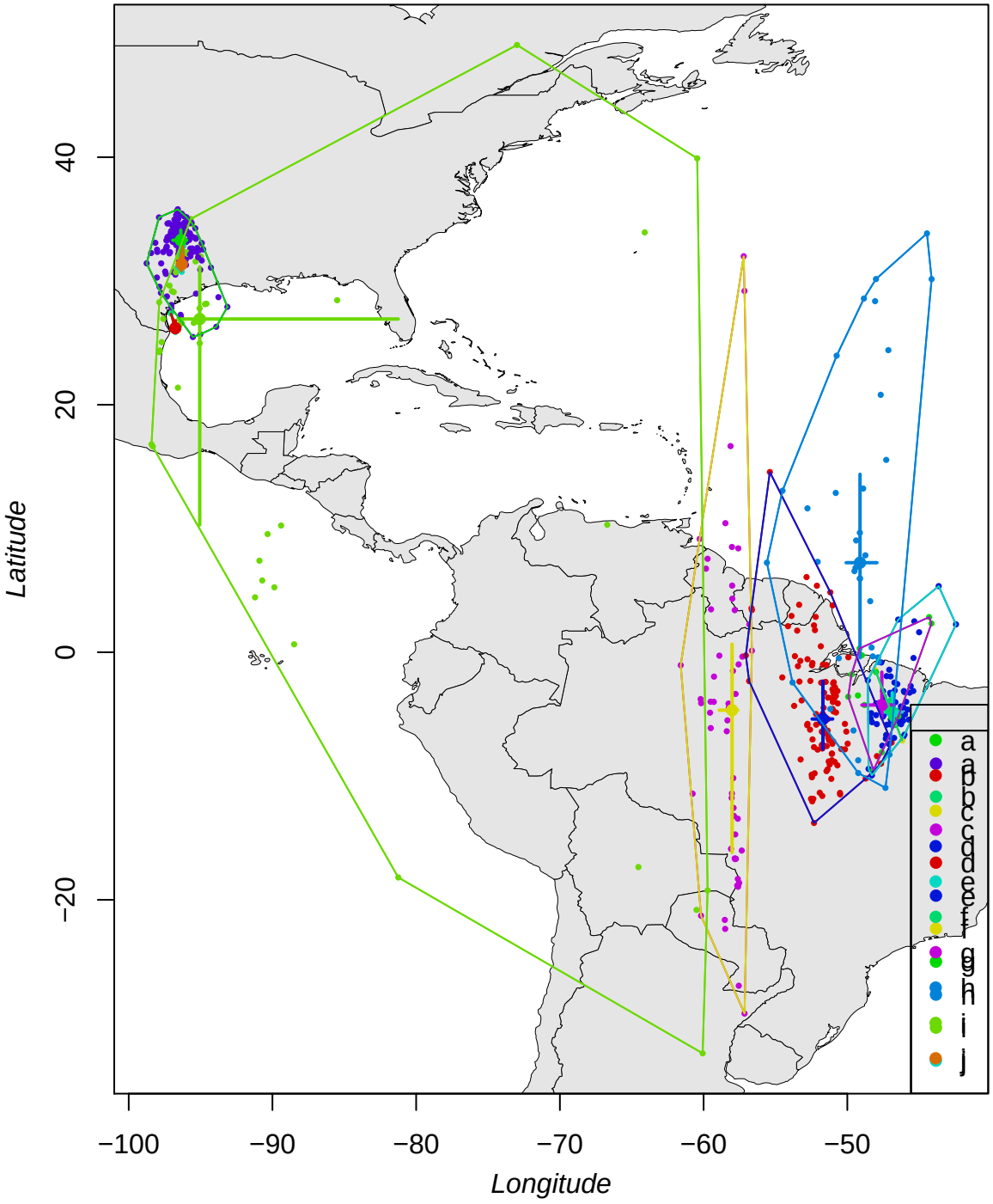
We can now plot the results, using the longitude and latitude estimates of `mergeSites2`.

```
data(wrld_simpl)
## create a color scale for stationary sites dependent on data
Seasonal_palette <- grDevices::colorRampPalette(grDevices::hsv(1 - ((1:365) + (365/4))%%365/365, s = 0.8
                                                space = "Lab")


### replace first and last estimate with the deployment/retrieval location
sm <- mS$summary
  sm[c(1,nrow(sm)), 2:3] <- matrix(c(lon.calib, lat.calib), ncol = 2, nrow = 2, byrow = T)
  sm[c(1,nrow(sm)), -c(1:3)] <- NA

day  <- as.POSIXlt(aggregate(mS$twl$tFirst[mS$site>0], by = list(mS$site[mS$site>0]), FUN = median)$x,
                   origin = "1970-01-01", tz  ="GMT")$yday
stp  <- as.numeric(aggregate(mS$twl$tFirst[mS$site>0], by = list(mS$site[mS$site>0]), FUN = function(x)
                             difftime(x[length(x)],x[1], units = "days"))$x)

## scale point of stationary periods according to the time spent on the site
cexf <- approxfun(range(stp), c(3, 9), rule = 3)


plot(NA, xlim = range(sm[,c(2,4:7)], na.rm = TRUE)+c(-2,2), ylim = range(sm[,c(3,8:10)], na.rm = T)+c(-1
     type = "n", bty = "n", mgp = c(3,2,1), xlab = "", ylab = "", las = 1)
plot(wrld_simpl, add = T, col = "grey90", border = "grey90")

lines(sm[,2], sm[,3], lwd = 2, lty = 2)
arrows(sm[,2], sm[,8], sm[,2], sm[,11], lwd = 0.8, length = 0, col = adjustcolor("black", alpha.f = 0.6)
arrows(sm[,2], sm[,9], sm[,2], sm[,10], lwd = 2, length = 0)

arrows(sm[,4], sm[,3], sm[,7], sm[,3], lwd = 0.8, length = 0, col = adjustcolor("black", alpha.f = 0.6))
arrows(sm[,5], sm[,3], sm[,6], sm[,3], lwd = 2, length = 0)

arrows(sm[,4], sm[,3], sm[,5], sm[,3], lwd = 1.2, length = 0)
points(sm[,2], sm[,3], pch = 21, cex = c(1, cexf(stp[-c(1, length(stp))]), 1), bg = Seasonal_palette(365
text(sm[,2], sm[,3], 1:nrow(sm),
     col = c("transparent", rep(0, nrow(sm)-2), "transparent"))
mapplots::add.pie(x = -90, y = -28, z = rep(1, 12), radius = 10, col = Seasonal_palette(12), init.angle
```



In comparison, we can use the `sites` vector and plot all location estimates grouped into the sites using the `siteMap` function.

```
siteMap(crds, site = mS$site, type = "points", xlim = range(crds[,1], na.rm = T), ylim = range(crds[,2]
  siteMap(crds, site = mS$site, type = "cross", add = TRUE)
```



Finally, we can extract the migration schedule:

```
schedule(mS$twl$tFirst, mS$twl$tSecond, site = mS$site)
```

```
   Site              Arrival           Departure
```

```
1     a                <NA> 2011-08-13 06:24:43
2     b 2011-08-21 05:46:22 2011-08-22 05:51:30
3     c 2011-09-07 15:55:15 2011-10-10 03:35:25
4     d 2011-10-13 03:12:52 2011-12-12 15:03:45
5     e 2011-12-14 03:05:24 2012-02-05 15:20:55
6     f 2012-02-07 03:33:51 2012-02-08 15:33:01
7     g 2012-02-15 15:35:00 2012-02-27 15:28:11
8     h 2012-02-29 03:29:28 2012-03-21 03:24:29
9     i 2012-04-04 18:21:08 2012-05-09 18:20:37
10    j 2012-05-14 18:21:36                <NA>
```

The temporal resolution of a *GeoLight* analysis is very low. And while high quality data may produce exact timing, the extracted dates should only be seen as approximate departure and arrival dates that can have errors of plus/minus two days.

My own contribution: I like GeoLight!

Chapter 6

probGLS

ProbGLS is an intuitive framework to compute locations from twilight events collected by geolocators from different manufacturers. The procedure uses an iterative forward step selection, weighting each possible position using a set of parameters that can be specifically selected for each analysis.

Getting started

To illustrate the *probGLS* analysis we use the Brünnich's guillemot (*Uria lomvia*) dataset which was collected as part of the SEATRACK project (http://www.seapop.no/en/seatrack). *probGLS* was developed mainly for the marine realm. Here, remote sensed data are often available to help location estimation. In contrast, solar angle calibration is often challenging as many species breed at high latitudes (i.e. no twilight events during breeding) and stationary periods are not as easily definable.

The tag used here has been developed by *Lotek*. No raw light intensities are stored by the logger. Instead, they estimate locations with onboard algorithms and summarises these outputs in their day log file. Additionally, these loggers record saltwater immersion every 5 minutes as well as ambient temperature at the same rate.

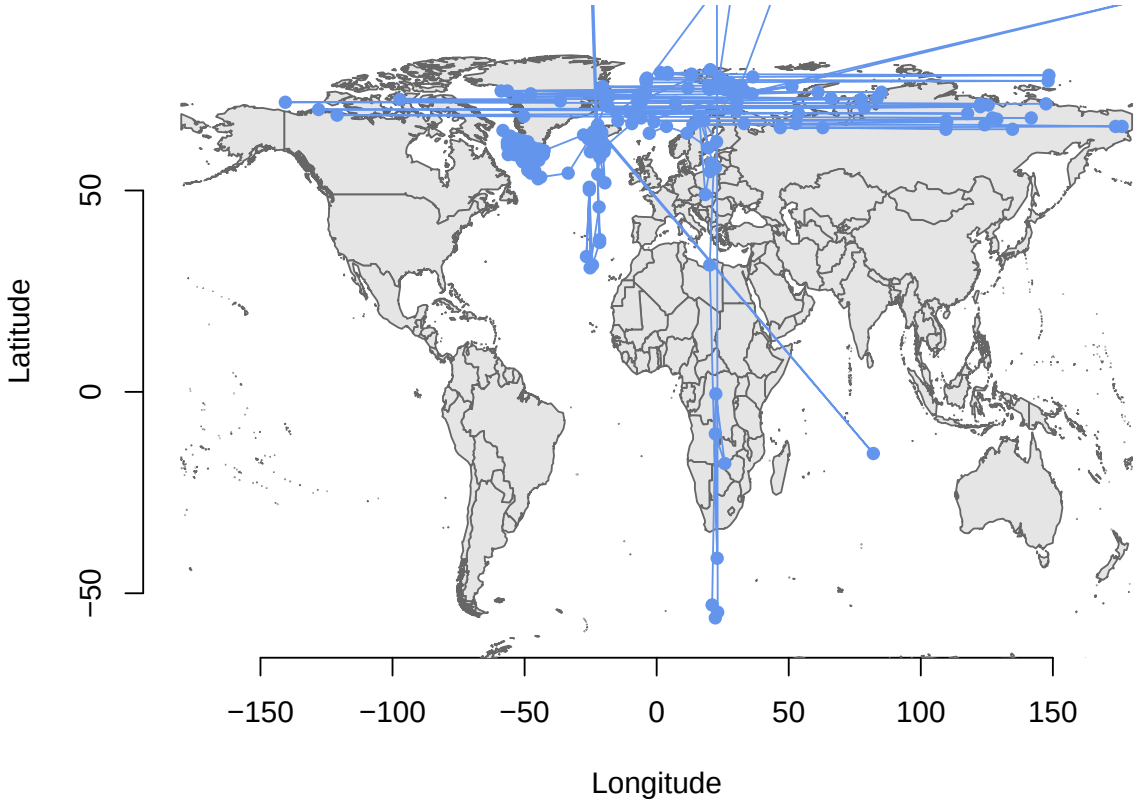
We first define the metadata and read in the raw recordings.

```
Species <- "UriLom"
ID <- "2655"

lat.calib <- 15.15
lon.calib <- 78.17

# define deployement and retrieval dates
start    <- as.Date("2012-07-11")
end      <- as.Date("2013-05-01")

wd <- "data"

raw       <- read.csv(paste0(wd, "/RawData/", Species, "/", ID, ".csv"),sep=",", header = T,row.names =
raw$TimeS <- as.Date(strptime(raw$TimeS,"%d/%m/%y"))
head(raw, 2)
```

```
       TimeS Sunrise Sunset TFLatN TFLatS TFNoonN TFNoonS SST1 SST1Depth
1 2012-06-06                   100  100.0   00:00   00:00   45      2000
2 2012-06-07   13:45  29:19     53   52.9   20:54   21:15   45      2000
  SST1Time TFLatErrN TFLatErrS TFLonErrN TFLonErrS MinIntTemp WetDryChange
1        0      38.7       2.1  1.00e+06       3.7      20.08            1
2        0      15.8      12.7  2.31e+01      19.5      16.90            1
  MaxPress  TRLon TRLat TFLonN TFLonS  X
1        0  200.0 100.0  179.7  179.7 NA
2        0 -143.2  40.6 -133.7 -139.0 NA
```

TRLon and *TRLat* are calculated using the threshold method and a hard coded solar angle of -3.44 degrees. Hence, if this angle does not mirror the actual solar angle, the latitudinal component of the data will of course be highly biased.

```
# exclude data outside deployment period
raw2  <- raw[raw$TimeS > start & raw$TimeS < end,]

# Plot threshold method derived location provided by the onboard algorithm
data(wrld_simpl)
plot(raw2$TRLon,raw2$TRLat ,type = "n", ylab="Latitude", xlab="Longitude",
     xlim = c(-180, 180), ylim = c(-60, 90), bty = "n")
plot(wrld_simpl, col = "grey90", border = "grey40", add = T)
points(raw2$TRLon,raw2$TRLat, pch=16, col="cornflowerblue", type = "o")
```



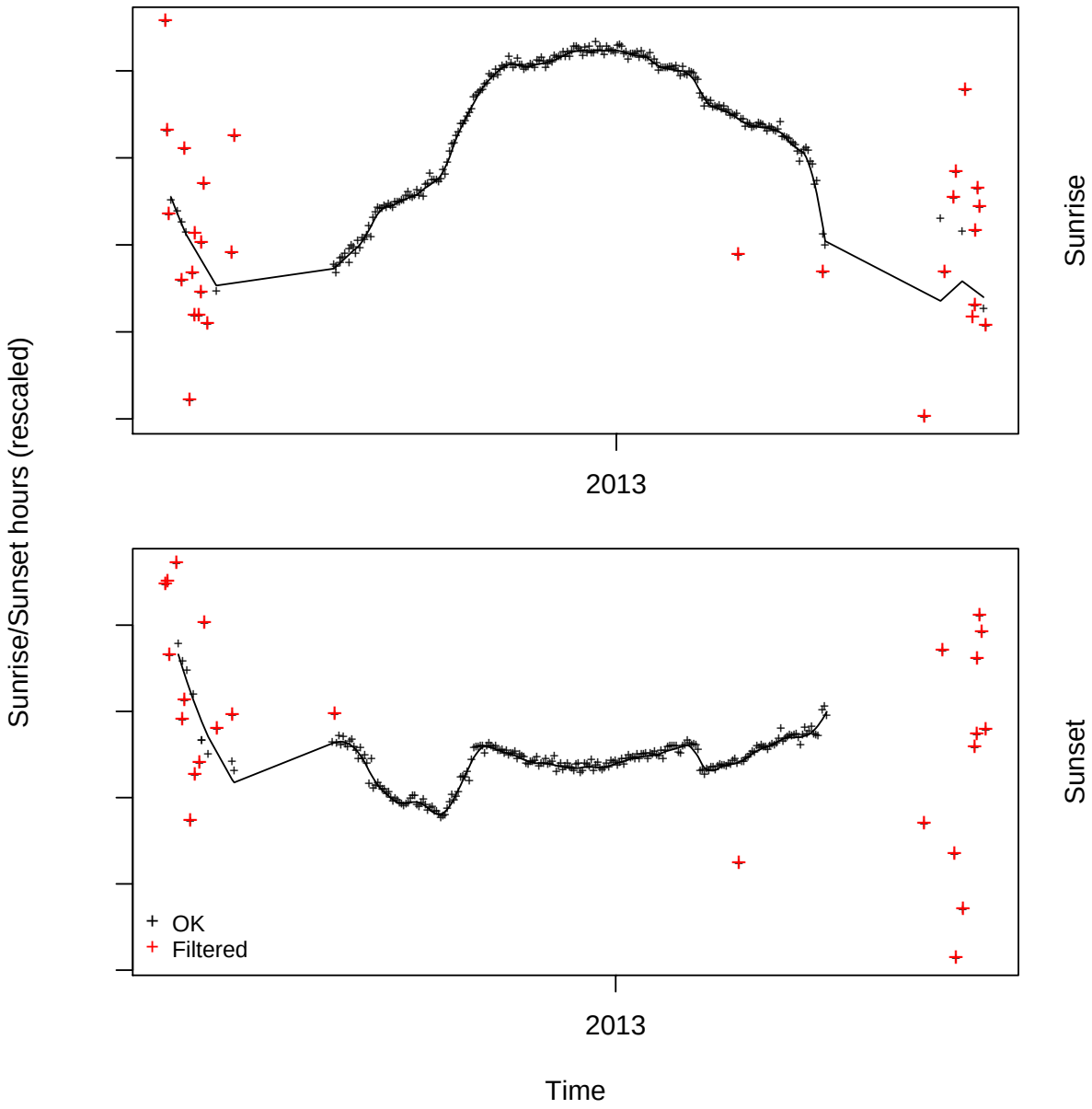
Therefore, we back calculate times of sunrise and sunset using this angle and positional data provided by the day log file.

```
trn <- lotek_to_dataframe(date    = raw$TimeS,
                          sunrise = as.character(raw$Sunrise),
                          sunset  = as.character(raw$Sunset),
                          TRLat   = raw$TRLat,
                          TRLon   = raw$TRLon)
trn <- trn[!is.na(trn$tSecond),]
head(trn,2)
```

```
               tFirst             tSecond type
1 2012-06-07 13:45:42 2012-06-08 05:18:12    1
2 2012-06-08 05:18:12 2012-06-08 07:29:15    2
```

Now we can have a look at these calculated twilight times to see how noisy they are using the `loessFilter` function from *GeoLight*.

```
trn$keep <- loessFilter(trn, k = 3, plot = T)
```



In this example the data seems to be quite clean. However, we can see spurious twilight times during summer as well as a gap of no data whatsoever until about August and from April onwards. Reason for this pattern is the fact that the study colony is located at 78 North. Hence, it is experiencing constant day light (i.e. midnight sun) from April until the end of August. This makes it difficult to calibrate the solar angle based on a known location as this species is breeding from June to the beginning of August.

To get around this obstacle we use additional data recorded by the logger to be able to estimate a probable movement track without calibrating the solar angle.

Load additional data

Saltwater immersion data*

We use 2 additional data streams recorded by the logger. These are salt water immersion (aka wet dry) and immersion temperature.

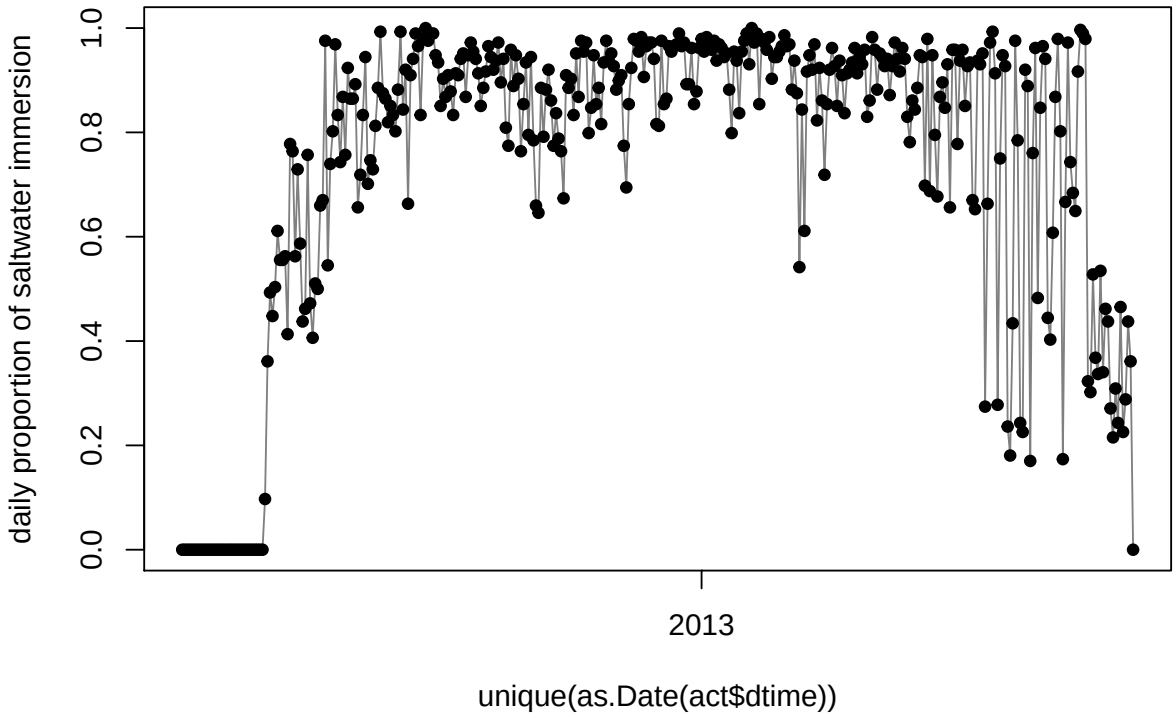
In this example the sampling rate is every 5 minutes. However, this may differ between the actual geolocator models.

```
data <-  read.csv(paste0(wd, "/RawData/", Species, "/", ID, "_Basic Log.csv"))

## datetime object needs to be in POSIXct, UTC time zone and must be called 'dtime'
data$dtime     <- as.POSIXct(strptime(data[,1], format = "%H:%M:%S %d/%m/%y"),tz='UTC')
data           <- data[!is.na(data$dtime),]
act            <- data
## wet dry data column must be called 'wetdry'
act$wetdry    <- 1-act$WetDryState
act <- subset(act, select = c(dtime,wetdry))

head(act)
```

```
                dtime wetdry
1 2012-06-08 13:44:59      0
2 2012-06-08 13:49:59      0
3 2012-06-08 13:54:59      0
4 2012-06-08 13:59:59      0
5 2012-06-08 14:04:59      0
6 2012-06-08 14:09:59      0
```

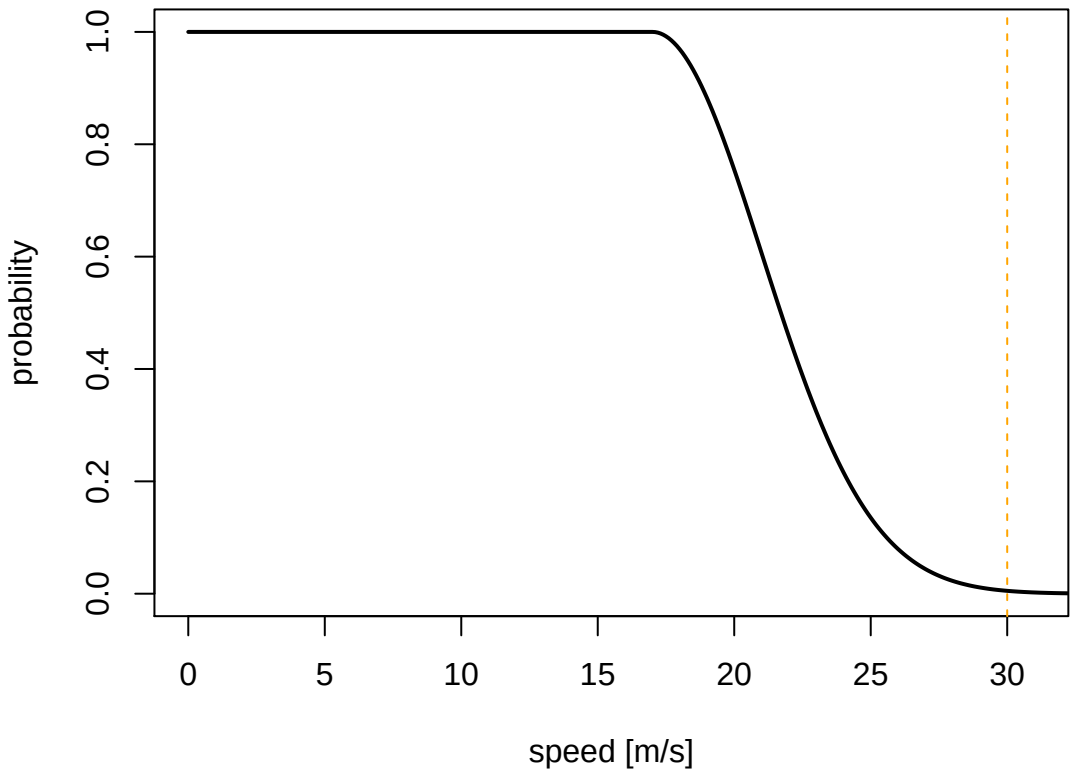
Using this auxiliary data, we can plot the daily proportion the logger has been submerged in saltwater.

```
plot(unique(as.Date(act$dtime)), tapply(act$wetdry,as.Date(act$dtime),sum)/288,
     type="l",col=grey(0.5),ylab="daily proportion of saltwater immersion")
points(unique(as.Date(act$dtime)),tapply(act$wetdry,as.Date(act$dtime),sum)/288,
     pch=19,cex=0.8)
```



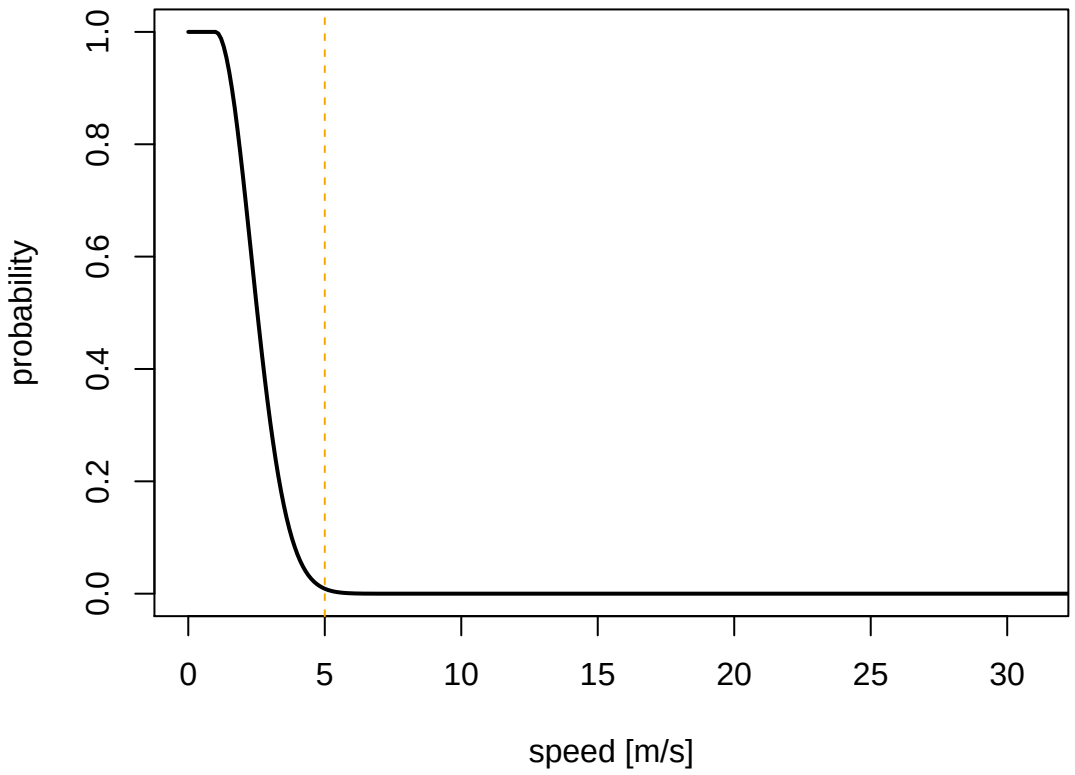
The here shown proportion, the logger is immersed in salt water each day can be used to estimate behaviour states of the individual tracked (e.g. sitting on water or flying). Additionally, the movement speed of a bird on or in water is not as high as when it is airborne, allowing us to define 2 speed distributions: One for when the logger is dry and one for when the logger is wet. The first should be informed by the ecology of the study species. In our example we can use speed estimates from GPS tracking studies to define a speed distribution.

```
speed_dry  = c(17, 4, 30)

sd <- data.frame(speed=seq(0,35,0.1),prob=dnorm(seq(0,35,0.1),speed_dry[1],speed_dry[2]))
sd$prob <- sd$prob/max(sd$prob)
sd$prob[sd$speed<speed_dry[1]] <- 1
plot(sd,type="l",xlim=c(0,31),xlab="speed [m/s]",ylab="probability",lwd=2)
abline(v=speed_dry[3],lty=2,col="orange")
```



The movement speed of an animal when wet can be defined using current speeds from ocean currents in the part of the globe you assume the animal to be in. In our example this is the North Atlantic current and the fast East Greenland current.

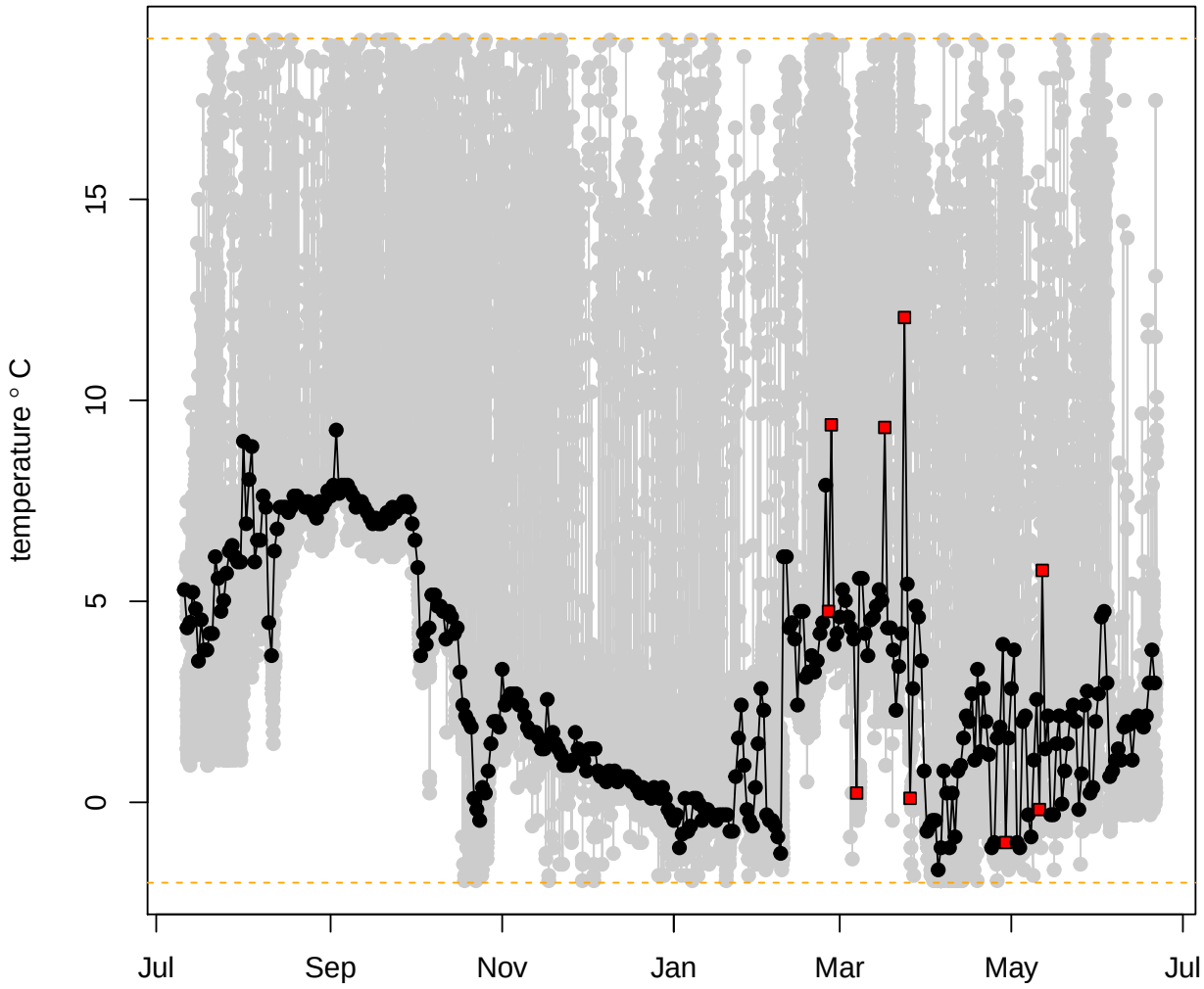
```
speed_wet  = c(1, 1.3, 5)

sd <- data.frame(speed=seq(0,35,0.1),prob=dnorm(seq(0,35,0.1),speed_wet[1],speed_wet[2]))
sd$prob <- sd$prob/max(sd$prob)
sd$prob[sd$speed<speed_wet[1]] <- 1
plot(sd,type="l",xlim=c(0,31),xlab="speed [m/s]",ylab="probability",lwd=2)
abline(v=speed_wet[3],lty=2,col="orange")
```



Immersion temperature data

In this example the sampling rate is every 5 minutes. This differs between logger models and brands.

```
td <- data

## only keep temperature values when the logger was immersed in salt water
## and if the 3 previous readings have been recorded while immersed in sea water as well
td$WetDryState.before   <- c(NA,head(td$WetDryState,-1))
td$WetDryState.before.2 <- c(NA,NA,head(td$WetDryState,-2))
td$WetDryState.before.3 <- c(NA,NA,NA,head(td$WetDryState,-3))
td           <- td[td$WetDryState         ==0 &
                   td$WetDryState.before  ==0 &
                   td$WetDryState.before.2==0 &
                   td$WetDryState.before.3==0,]

## determine daily SST value recorded by the logger
sst           <- sst_deduction(datetime = td$dtime, temp = td$IntTemp, temp.range = c(-2,19))
abline(h=c(-2,19),lty=2,col="orange")
```



In grey you can see the temperature values recorded by the tag. In black is the estimated daily sea surface temperature (SST) values determined from the algorithm. All red data points are labeled as "remove". Here, data is conservativly removed rather than smoothed out or interpolated as temperature can often shift rather dramatically, especially when an individual is utilizing an oceanographic front. For removed values no SST data will be used to improve the location estimation algorithm.

```
head(sst)
```

```
                date   SST SST.remove
2012-07-11 2012-07-11 5.290      FALSE
2012-07-12 2012-07-12 4.340      FALSE
2012-07-13 2012-07-13 4.475      FALSE
2012-07-14 2012-07-14 5.225      FALSE
2012-07-15 2012-07-15 4.815      FALSE
2012-07-16 2012-07-16 3.515      FALSE
```

Download remote sensed environmental data

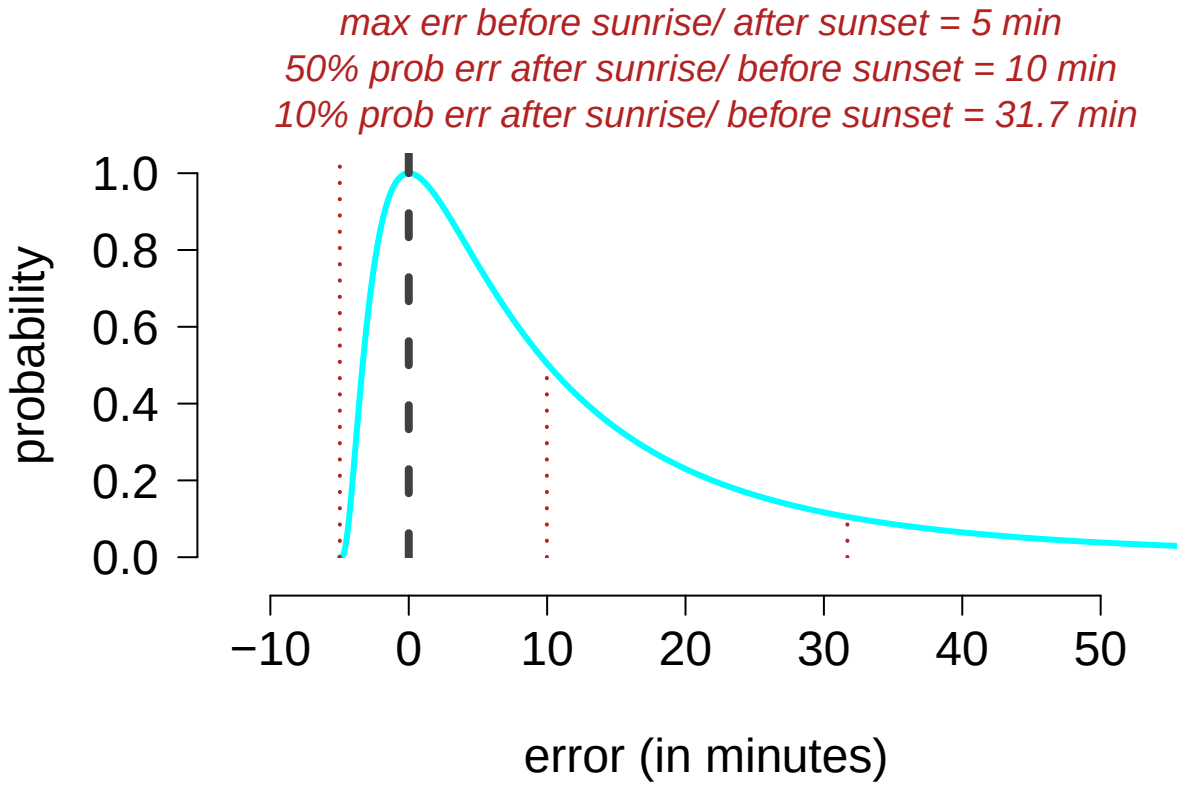
Now, we need remote sensed SST fields to fit the deduced SST values recorded by the logger with what was available in the environment. Here we use one of many remote sensed data products. The strength of this product is the long temporal time scale (1981 - now) coupled with a reasonable spatial resolution of 0.25 x 0.25 degrees.

```
# download environmental data ----

# download yearly NetCDF files for (replace YEAR with appropriate number):
# daily mean SST                   -> 'sst.day.mean.YEAR.nc'
# daily SST error                  -> 'sst.day.err.YEAR.nc'
# daily mean sea ice concentration -> 'icec.day.mean.YEAR.nc'

# from:
# https://www.esrl.noaa.gov/psd/data/gridded/data.noaa.oisst.v2.highres.html
# and place all in the same folder (here we use a AuxiliaryData in the RawData folder of the species)

# Also, download the land mask file: 'lsmask.oisst.v2.nc' from the same directory
# and place it in the same folder as all the other NetCDF files
```

Twilight error (Calibration)

Here we estimate the assumed error around a twilight event as log-normal distribution. The parameters used here are chosen as they resemble the twilight error structure of open habitat species.

```
tw <- twilight_error_estimation(shape = 2.49, scale = 0.94, delay = 0)
```



Run the iterative algorithm

Finally, all pieces are in place and we can run the iterative algorithm.

```
pr   <- prob_algorithm(trn                          = trn,
                       sensor                       = sst[sst$SST.remove==F,],
                       act                          = act,
                       tagging.date                 = start,
                       retrieval.date               = end,
                       loess.quartile               = NULL,
                       tagging.location             = c(lon.calib, lat.calib),
                       particle.number              = 500,
                       iteration.number             = 100,
                       sunrise.sd                   = tw,
                       sunset.sd                    = tw,
                       range.solar                  = c(-7,-1),
                       speed.wet                    = speed_wet,
                       speed.dry                    = speed_dry,
                       sst.sd                       = 0.5,
                       max.sst.diff                 = 3,
                       boundary.box                 = c(-90,120,40,90),
                       days.around.spring.equinox   = c(21,14),
                       days.around.fall.equinox     = c(14,21),
                       ice.conc.cutoff              = 0.9,
```

```
                      land.mask                 = T,   # The track is assumed not to be on land
                      med.sea                   = F,   # if T the track cannot enter the Mediterrane
                      black.sea                 = F,   # if T the track cannot enter the Black Sea
                      baltic.sea                = F,   # if T the track cannot enter the Baltic Sea
                      caspian.sea               = F,   # if T the track cannot enter the Caspian Sea
                      east.west.comp            = F,   # if true use the east west compensation (see
                      wetdry.resolution         = 300, # in seconds, i.e. 5 minutes = 300 seconds
                      NOAA.OI.location          = paste0(wd, "/RawData/", Species, "/AuxiliaryData")

## loess.quartile  - if it is not NULL then the GeoLight loessFilter function is used previous to runni
## particle.number - number of particles generated at each step
## range.solar     - range of solar angles assumed
## sst.sd          - accuracy of the logger temperature sensor
## max.sst.diff    - maximum discrepancy allowed between recorded and remote sensed SST
## boundary.box    - spatial extent
## ice.conc.cutoff - the animal is not assumed to enter pixels with more than 90% sea ice concentration
```

The data object created is a list containing 5 parts:

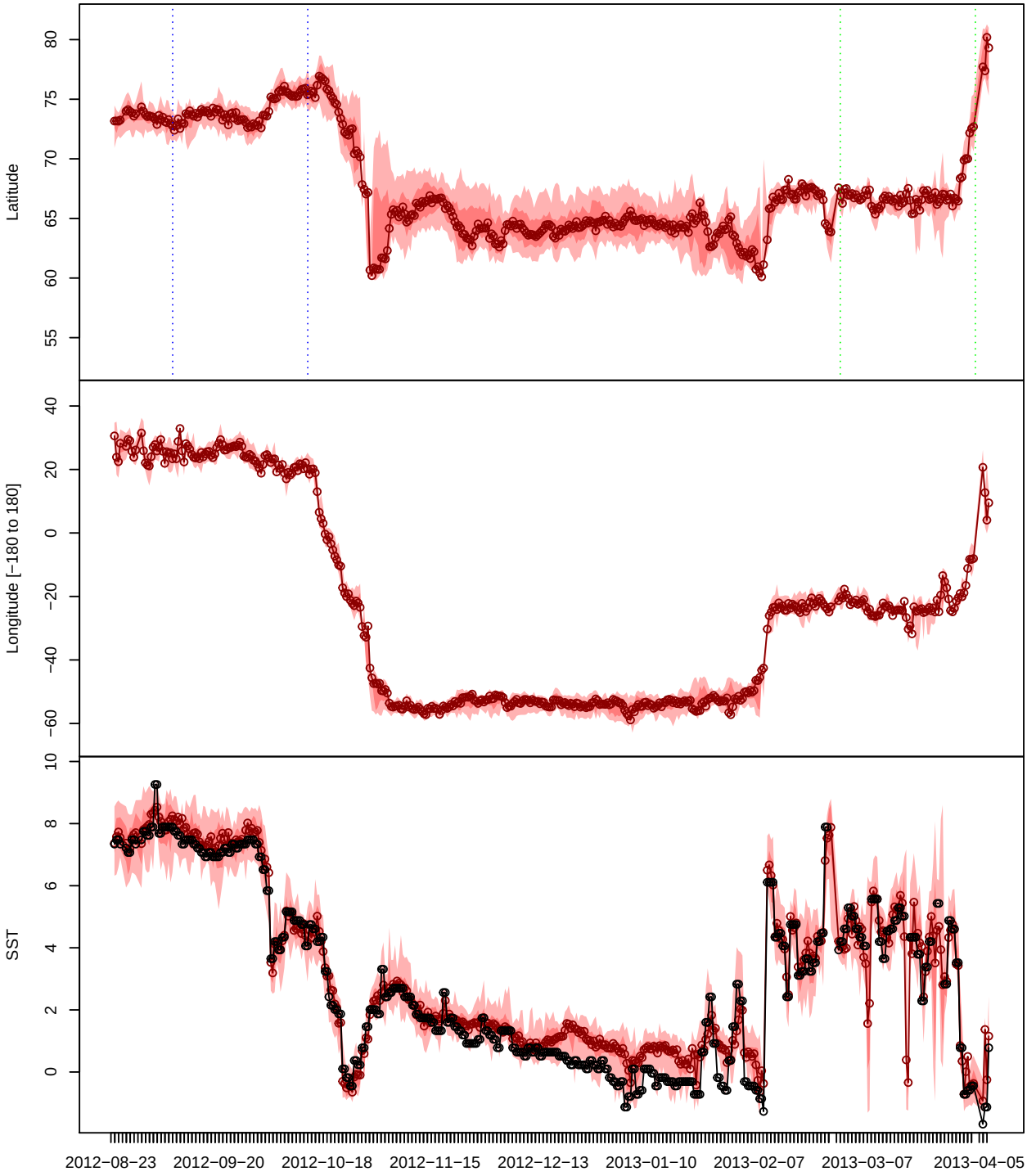
```
summary(pr)
```

```
                       Length Class                   Mode
all tracks              43767 SpatialPointsDataFrame S4
most probable track       442 SpatialPointsDataFrame S4
all possible particles 221966 SpatialPointsDataFrame S4
input parameters            2 data.frame             list
model run time              1 difftime               numeric
```

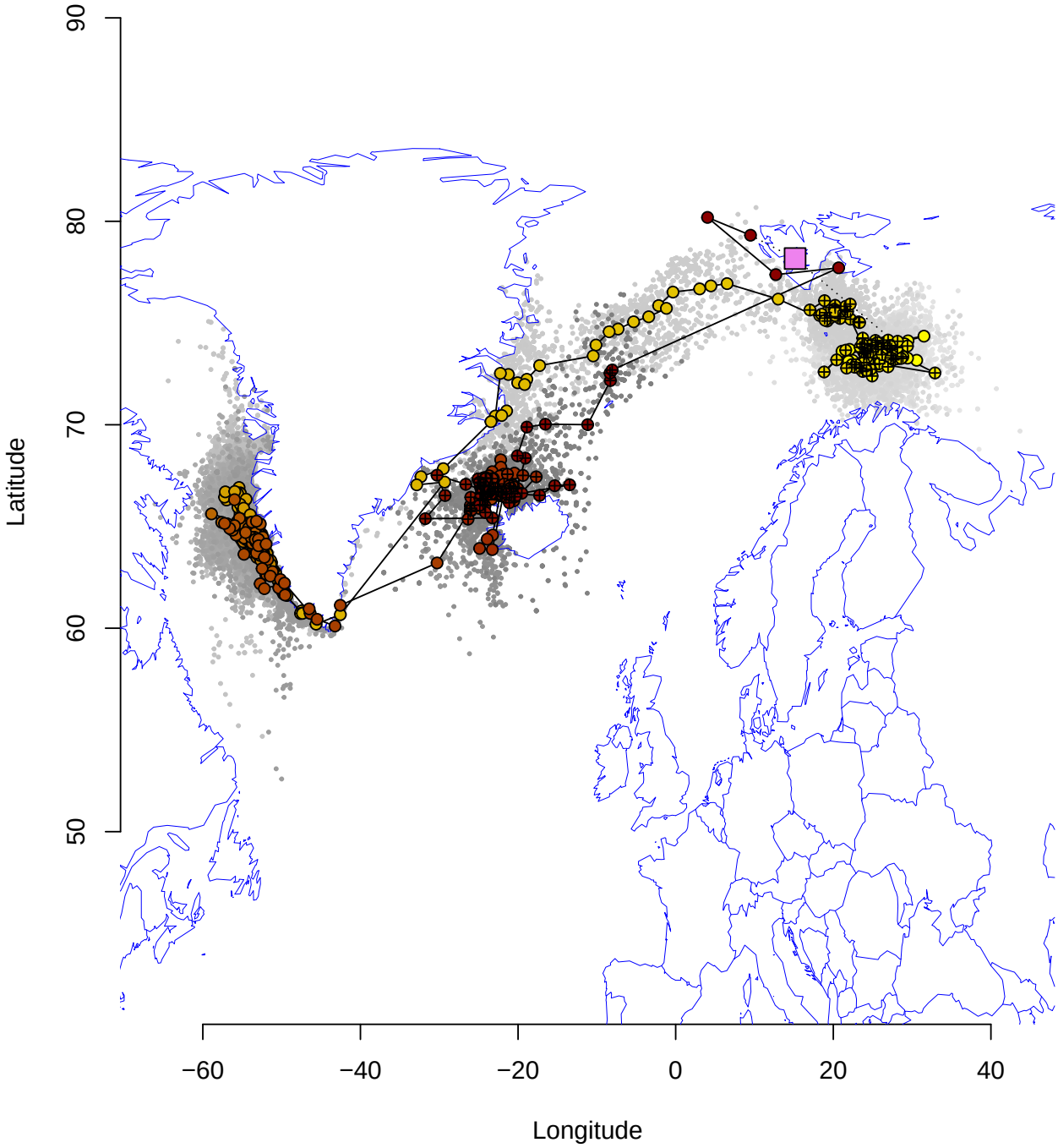
- The first item contains all 100 tracks computed by the algorithms.
- The second item contains the most probable track as geographic median at each step.
- The third item contains all generates particles at each step.
- The fourth item stores all input parameter.
- The last item is just the time it took to run the algorithm (17 minutes in this example).

Plot results

```
# plot lat, lon, SST vs time ----
plot_timeline(pr,degElevation = NULL)
```




```
# plot lon vs lat map ----
plot_map(pr)
```



Here, the most probable track is displayed from yellow to dark red with its uncertainty in grey (light to dark). The colony location is visualised at violet square. Locations estimated around the equinox periods are marked with a cross.

The median track as well as its associated uncertainty can now be used in further analyses.

Chapter 7

SGAT

The package *SGAT* (TAGS backwards) is based in the principles that haven been developed for the *tripEstimation* package that has now been deprecated by *SGAT*. The biggest difference between these two packages is the possibility to use twilight events to run the mode. *tripEstimation* was based solely on the curve method. However, *SGAT* has additional capabilities that we will discuss in the workflow below. Here, we highlight the *groupModel*, and the *twilightFree* model as recent developments with great potential.

In general, *SGAT* implements two models - **Estelle** and **Stella**. Both models can be setup using threshold based twilight events or twilight period (*curve method*), but impose different constraints on the movement of the tag. Stella estimates the locations $x_1, x_2, \ldots, x_n$ of the tag at the observed times of twilight $t_1, t_2, \ldots, t_n$ assuming the great circle distance between any two successive locations x_i and x_{i+1} follows a given distribution. Estelle also considers intermediate locations $z_1, z_2, \ldots, z_{n-1}$, where z_i is the location of the tag at an arbitrary time τ_i between twilights $t_i < \tau_i t_{i+1}$, and assumes the great circle distance along any dog-leg path x_i, z_i, x_{i+1} follows a given distribution.

Both models estimate location based on observed times of twilight. More precisely, let $t = (t_1, t_2, \ldots, t_n)$ denote the observed times of twilight, let $x = (x_1, x_2, \ldots, x_n)$ denote the corresponding locations of the tag at these times, and let $r = (r_1, r_2, \ldots, r_n)$ be indicators of whether each twilight is a sunrise or sunset. Let $\hat{t}_i(x)$ denote the true time at which the twilight corresponding to t_i occurs at location x.

Both models assume that

$$t_i \sim F(\hat{t}_i(x_i); r_i, \alpha)$$

for some known distribution F dependent upon a vector of (known) parameters α.

The two models differ in the way they represent the motion of the tag between successive twilights.

Let $d = (d_1, d_2, \ldots, d_{n-1}) = D(x)$ denote the vector of great circle distances d_i between successive locations x_i and x_{i+1}. Stella assumes the joint distribution of these distances is

$$d \sim G(\beta)$$

for some known distribution G dependent upon a vector of (known) parameters β. This package implements a less general form of model in which it is assumed the d_i are independently distributed

$$d_i \sim G_i(\beta).$$

Together, the twilight and behavioural models define the likelihood for the model. If $p(x_i)$ denote independent priors for the locations then the posterior distribution for x under the Stella model can be written

$$p(x \mid t, r, \alpha, \beta) \propto \left(\prod_{i=1}^{n} f(t_i \mid \hat{t}_i(x_i), r_i, \alpha) \right) \times g(D(x)|\beta) \times \left(\prod_{i=1}^{n} p(x_i). \right)$$

Similarly, if $p(z_i)$ denote independent priors for the intermediate points z_i, then the posterior under the Estelle model can be written

$$p(x, z \mid t, r, \alpha, \beta) \propto \left(\prod_{i=1}^{n} f(t_i \mid \hat{t}_i(x_i), r_i, \alpha) \right) \times g(D(x, z)|\beta) \times \left(\prod_{i=1}^{n} p(x_i) \right) \times \left(\prod_{i=1}^{n-1} p(z_i) \right).$$

In both cases a sequence of samples from the posterior can be drawn by standard MCMC techniques.

Getting started

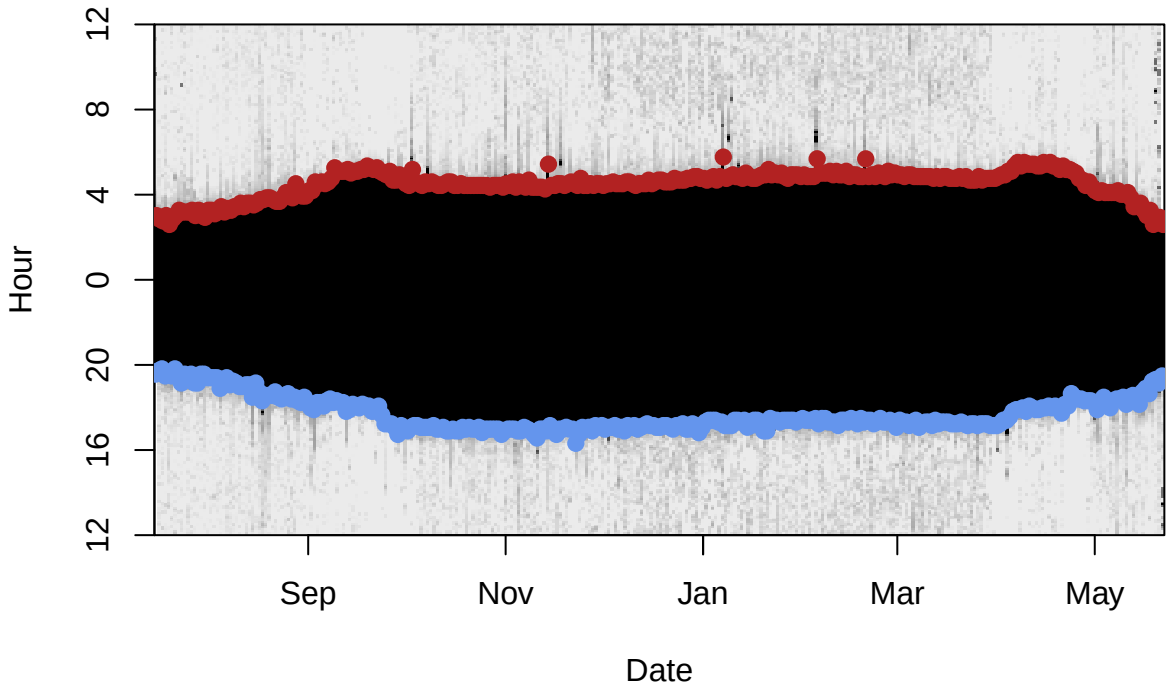
To illustrate the *SGAT* analysis, we use the European bee-eater dataset. The light intensities were recorded by a geolocator from the Swiss Ornithological Institute, measuring light every minute writing the mean of every 5 measurements. Thus, we do not need ot adjust the sunset times.

We first define the metadata and read in the raw recordings. We skip the twilight definition process but read in the twilight file that has been generated using `preprocessLight`. Note: it is required to retransform the Twilight column into the `POSIXc`format.

```
ID <- "14SA"
Species <- "MerApi"

lon.calib <- 11.96
lat.calib <- 51.32

wd <- "data"

raw <- glfTrans(paste0(wd, "/RawData/", Species, "/", ID, ".glf"))
  names(raw) <- c("Date", "Light")
  raw$Light  <- log(raw$Light+0.0001) + abs(min(log(raw$Light+0.0001)))

twl <- read.csv(paste0(wd, "/Results/", Species, "/", ID, "_twl.csv"))
  twl$Twilight <- as.POSIXct(twl$Twilight, tz = "UTC")
  twl <- twl[!twl$Deleted,]

raw <- subset(raw, Date>=min(twl$Twilight) & Date<=max(twl$Twilight)) # clipping raw data to relevant es
```

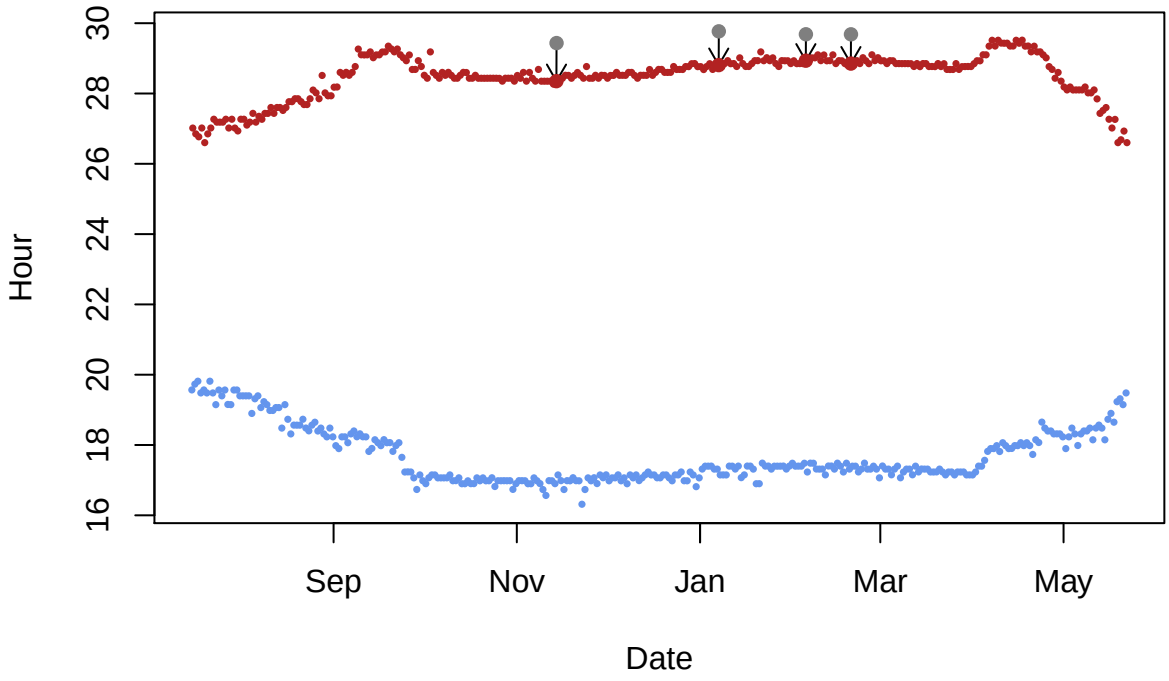
We can have a look into the data using the `lightImage` function from the `TwGeos` package:

```
offset <- 12 # adjusts the y-axis to put night (dark shades) in the middle

lightImage( tagdata = raw,
            offset = offset,
            zlim = c(0, 20))

tsimagePoints(twl$Twilight, offset = offset, pch = 16, cex = 1.2,
              col = ifelse(twl$Rise, "firebrick", "cornflowerblue"))
```



There are some sunrises and sunsets that have been misclassified, so we can use the `twlightEdit` function to move these to where they should be.

```
twl <- twilightEdit(twilights = twl,
                    offset = offset,
                    window = 4,           # two days before and two days after
                    outlier.mins = 45,    # difference in mins
                    stationary.mins = 25, # are the other surrounding twilights within 25 mins of one a
                    plot = TRUE)
```



It's usually best to do this step manually. See the Twilight annotation page for more information.

Calibration

Calibration for the *SGAT* process is similar to the calibration performed in the *GeoLight* analysis. Both, the *zero* and the *median* sun elevation angles, as well as the parameters for the error distribution of the twilight times is crucial for the analysis. **However**, while we use sun elevation angles in *GeoLight* we need the **zenith** angle in *SGAT*. The difference is trivial; sun elevation angle refers to the deviation of the sun relative to the horizon, whereas the zenith angle refers to the deviation from the zenith. Thus, civil twilight is defined as the time when the sun elevation angle is -6 degrees which equals a zenith angle of 96 degrees.

The simple conversion of sun elevation angle to zenith angle is:

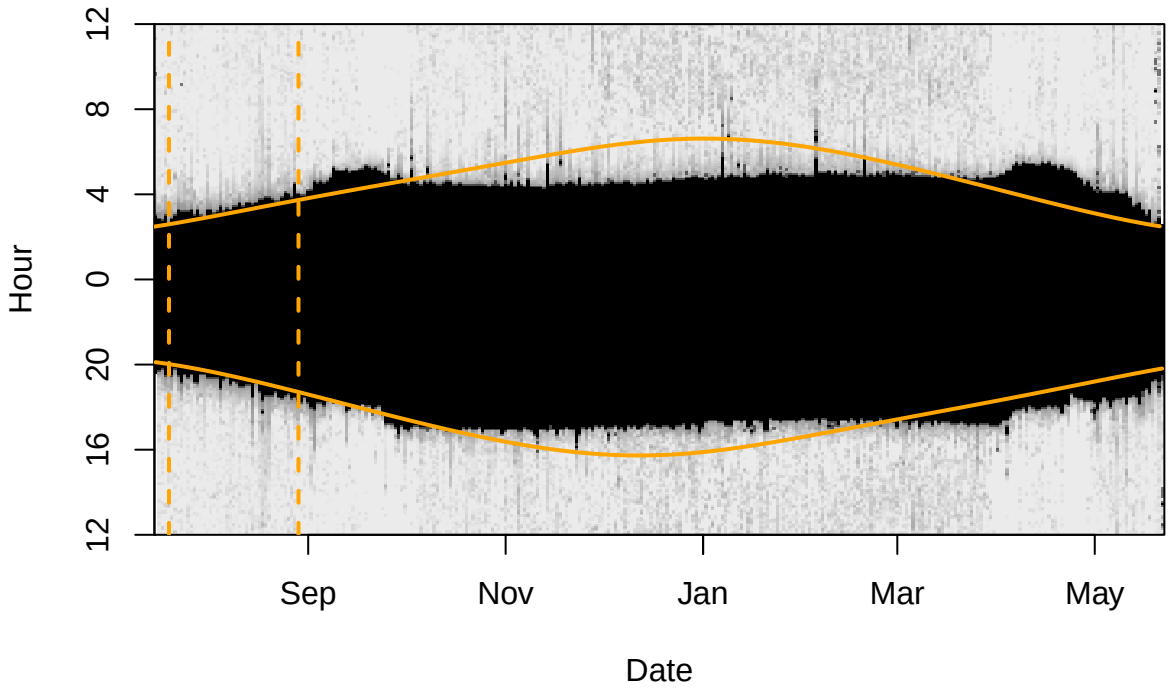
$$zenith = 90 - sunelevationangle$$

There are multiple ways to define the time period for calibration. Best is to know when the individual left the deployment site and if there where a couple of weeks between deployment and departure. In many instances the departure date (or the arrival to the retrieval site) is unknown. The `lightImage` together with the `tsimageDeploymentLine` can help to define suitable period (the right time period can be optimized by changing the date in the `tm.calib` vector and plotting the lines over and over again until you are sure that

you have selected the beginning and the end of the calibration period). Again, the longer the period the better, but periods that are influenced by e.g. breeding in nest boxes or by movements should be excluded.

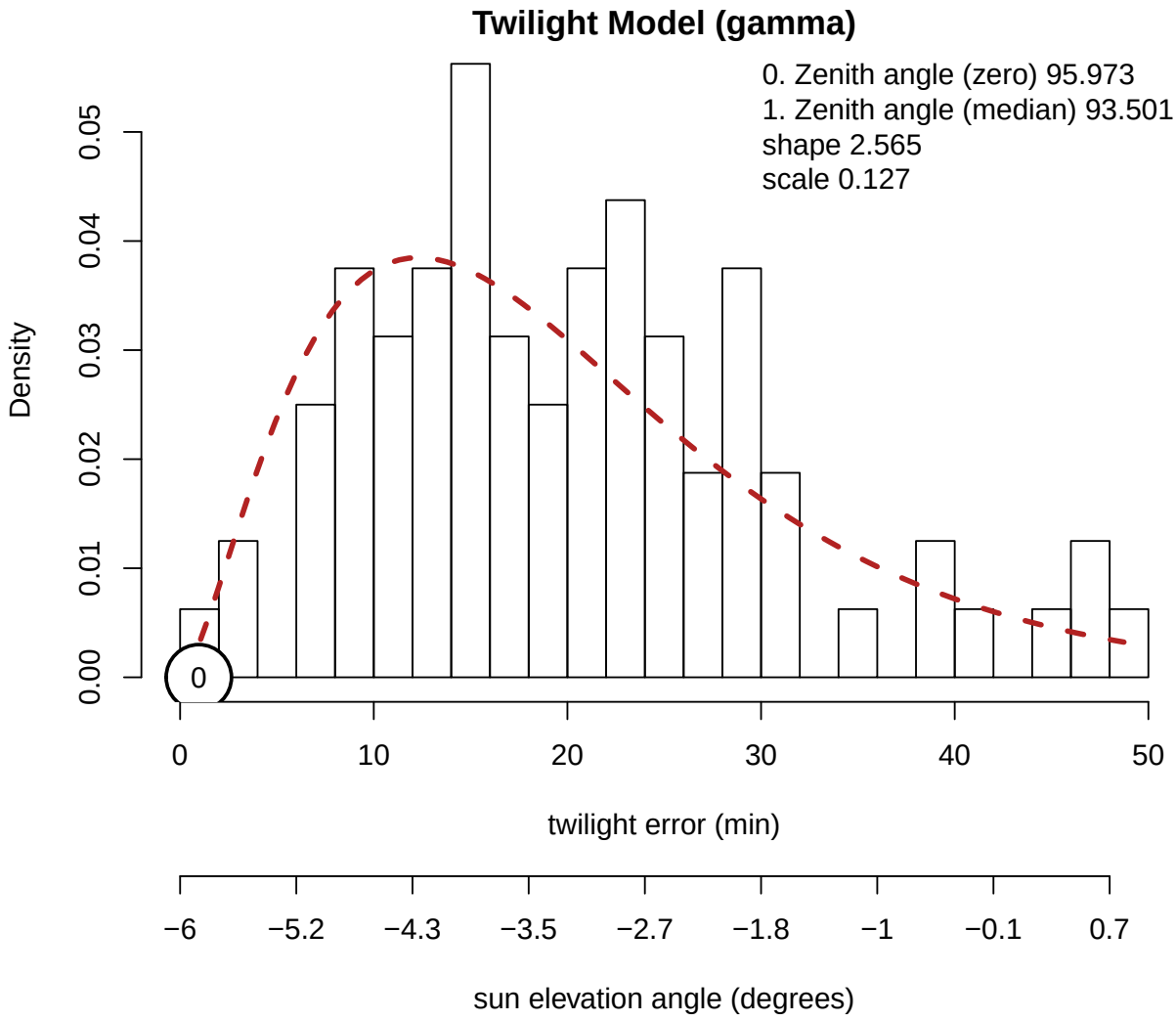
More specifically, `lightImage` visually presents night (in black) and day (white) throughout the year. This allows us to see when changes in night length occur and thus when the bird has moved. Based on this, we can identify when the bird left the deployment site and manually specify these for `tm.calib` .

```
lightImage( tagdata = raw,
            offset = offset,
            zlim = c(0, 20))

tsimageDeploymentLines(twl$Twilight, lon.calib, lat.calib, offset, lwd = 2, col = "orange")

tm.calib <- as.POSIXct(c("2015-07-20", "2015-08-29"), tz = "UTC")
abline(v = tm.calib, lwd = 2, lty = 2, col = "orange")

d_calib <- subset(twl, Twilight>=tm.calib[1] & Twilight<=tm.calib[2])
```



Using the calibration subset of the *twl* table we can perform the calibration:

```
calib <- thresholdCalibration(d_calib$Twilight, d_calib$Rise, lon.calib, lat.calib, method = "gamma")
```



This is how a calibration time series should look like. Based in theory it should follow a gamma or a log-normal distribution (both can be used in *SGAT*). What we can see, is that the recorded twilight times most frequently deviation approx. 12 minutes. However, deviations of up to 50 minutes have been recorded. For the following analysis, we need the zenith angle for both the zero deviation (0, and second number in return vector e.g. `calib[2]`) and the most frequent *median* deviation (1, and the first number in the return vector e.g. `calib[1]`). Additionally we need the parameters of the error distribution (`alpha` parameters, e.g. `calib[3:4]`).

```
zenith  <- calib[1]
zenith0 <- calib[2]

alpha <- calib[3:4]
```

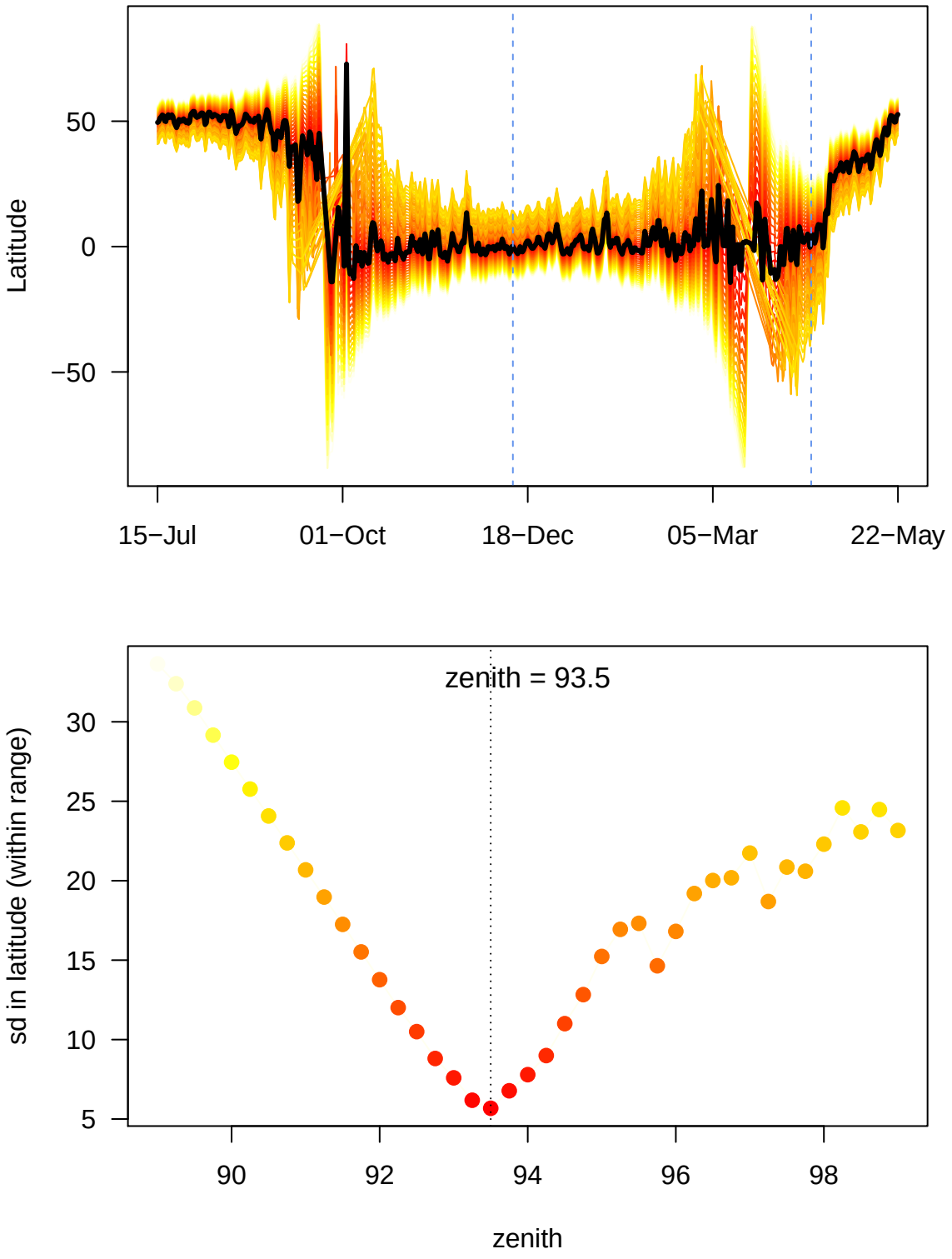
Alternative calibration

For the bee-eaters and many other species, the breeding season is often also when the loggers are deployed but is a very special period because the birds use different habitats and show different behaviors compared to the rest of the annual cycle. For instance, bee-eaters use burrows during the breeding season, but not during the rest of the year. This is of course suboptimal for calibration since it would lead to good estimates for the breeding grounds when we know the exact location, and biased estimates of sunrise and sunset for the rest of the year. We can therefore try and estimate an alternative zenith angle based in the Hill-Ekstrom theory that the right zenith angle should lead to the lowest variance in latitude estimates (i.e. flattest) during stationary periods. And the latter is most pronounced around the equinox. The following bits of code draw a basic path and then compare different zeniths to find the one with the lowest variation. It then uses that new zenith with the least sd in the threshold model.

In the `findHEZenith`function, the `tol` argument defines how many locations should be linearly interpolated around the equinox. Large values lead to larger periods with interpolated values. For this type of calibration, it makes sense to play with this value but in general it is recommended to set it to a low value (e.g. 0.08). If the tracked individual has been stationary during the time of the equinox this period provides the best data for the Hill-Ekstrom calibration.

```
startDate <- "2015-12-12"
endDate   <- "2016-04-15"

start = min(which(as.Date(twl$Twilight) == startDate))
end = max(which(as.Date(twl$Twilight) == endDate))

(zenith_sd <- findHEZenith(twl, tol=0.01, range=c(start,end)))
```

```
[1] 93.5
```

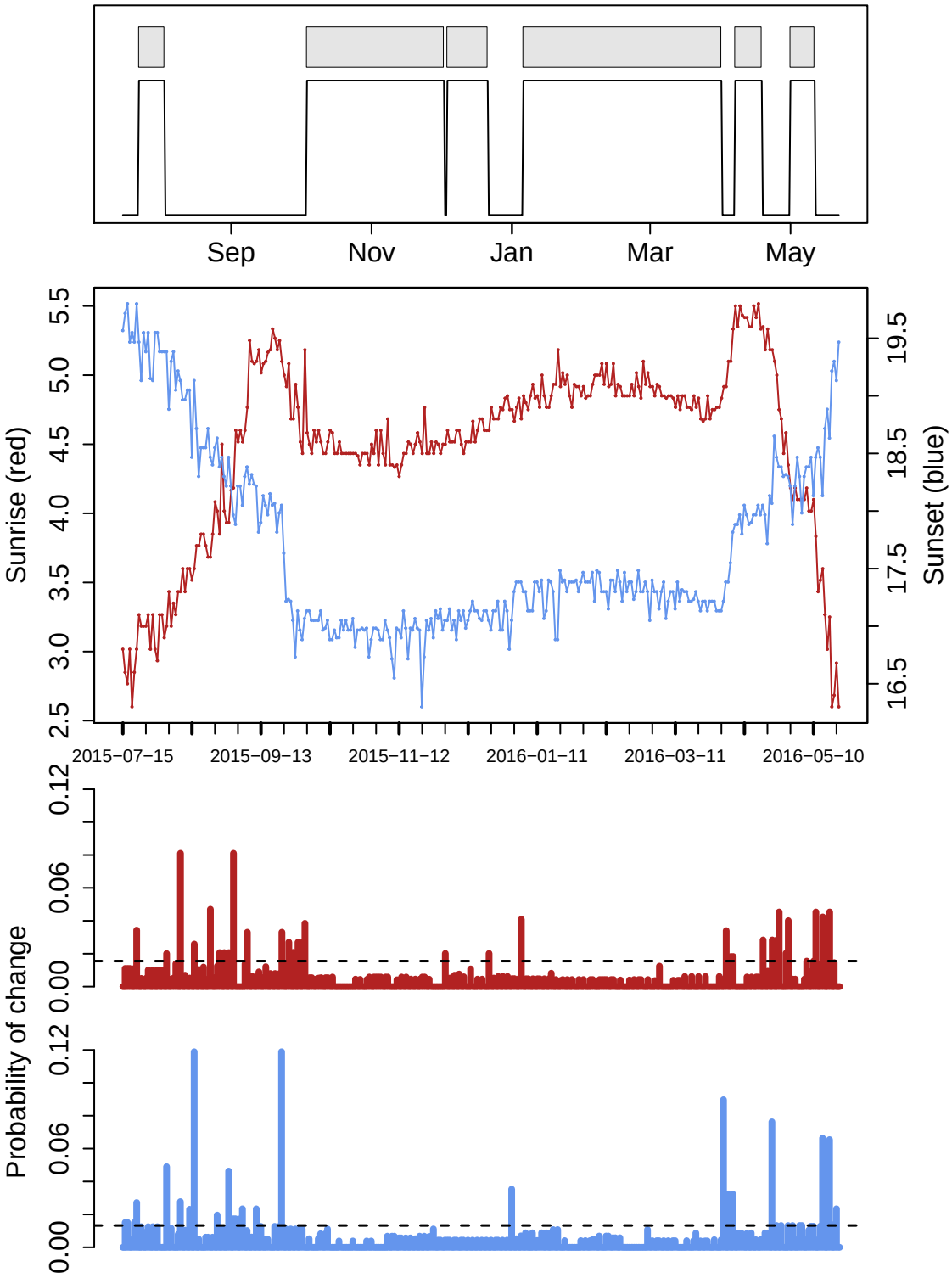


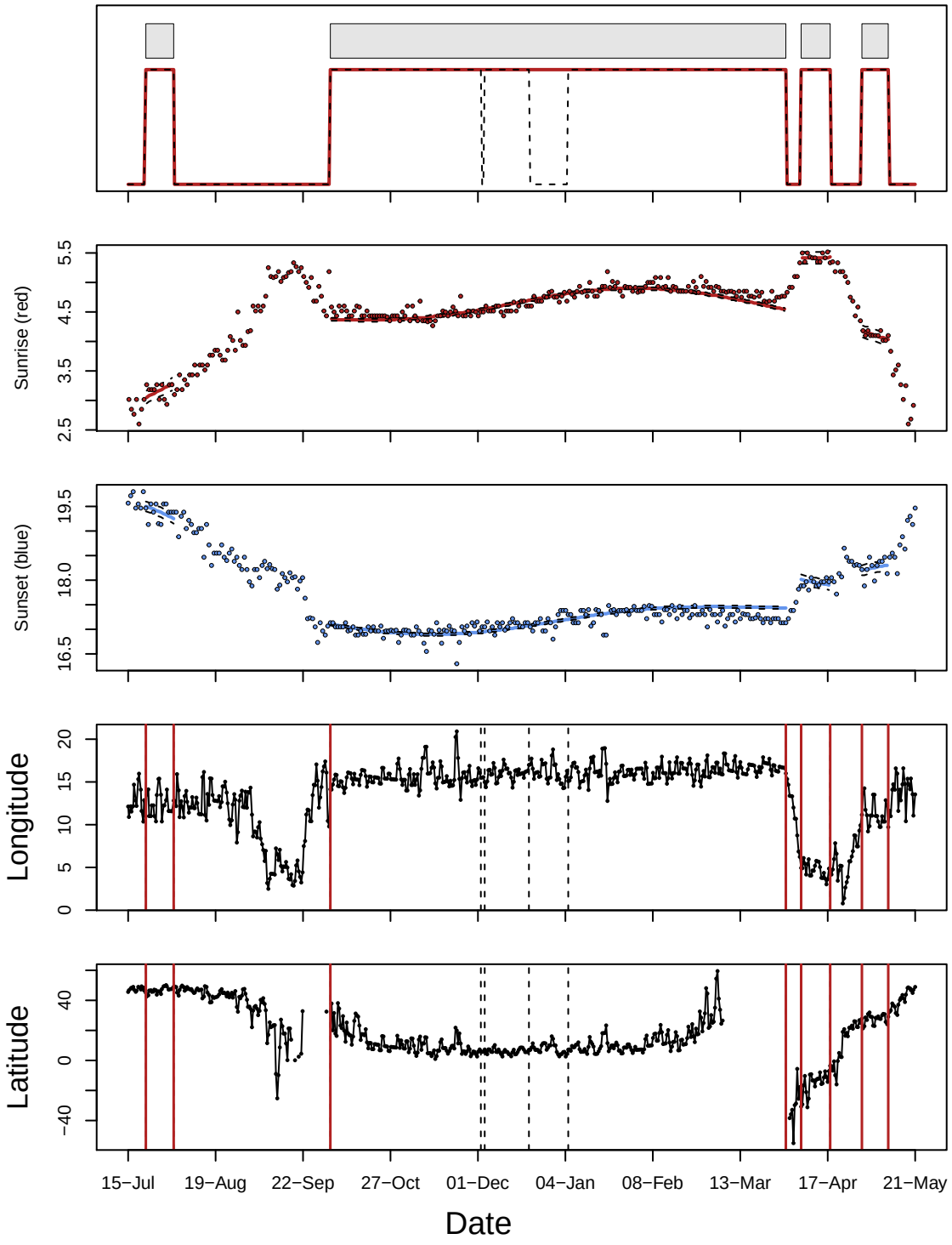
The top panel shows the entire path (latitude) using different zenith angles with the black line indicating the latitude estimates with the smallest variation within the specified range (in between the two blue dashed lines). One needs to be quite sure that the individual did not move during this period. The lower pane shows the actual variation in latitudes across a range of zenith angles. It is good if one can see a clear minimum in this curve.

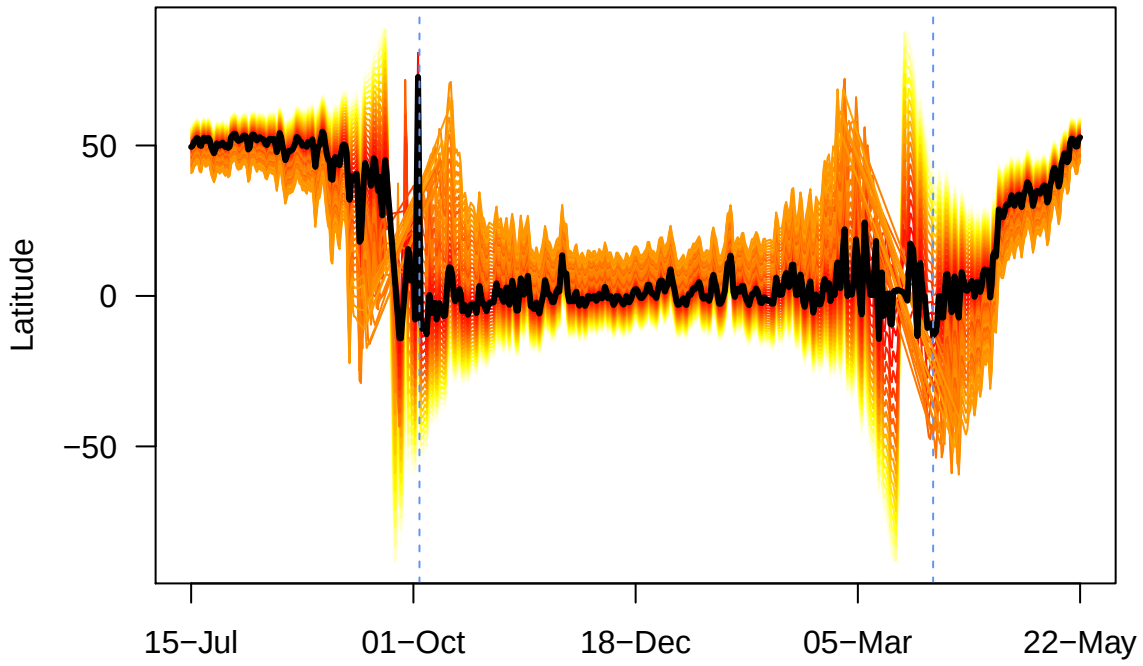
Play around with the range. For instance, look what happens when the `endDate` is changed to "2016-01-15". This is not what you want - there is no clear u-shape in the bottom panel and the latitude during stationary non-breeding period in the top panel is very curved, not flat. In such cases, it's important to increase the range to cover some of the equinox period which is the most noisy. In some cases it can even be worth using the `mergeSites` function from the `GeoLight` package to find stationary sites to use in the Hill-Ekstrom calibration. Here's an example below of how this can be done.

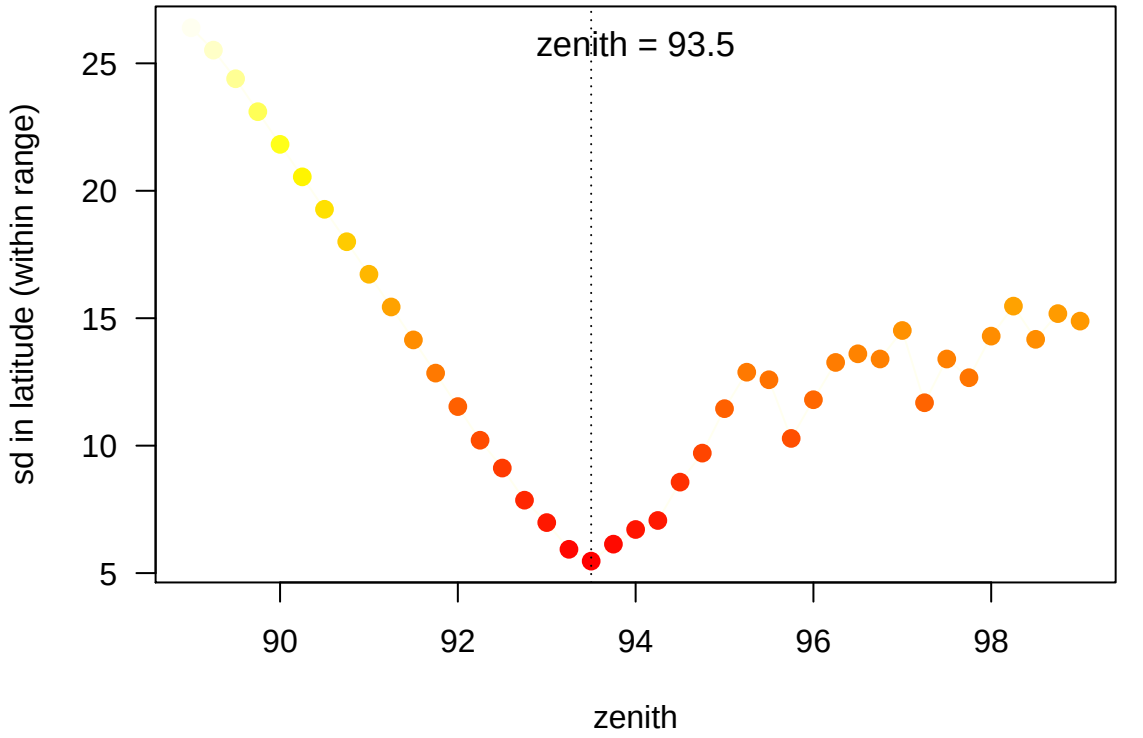
```
#convert to geolight format
geo_twl <- export2GeoLight(twl)

# this is just to find places where birds have been for a long time, would not use these parameters for
cL <- changeLight(twl=geo_twl, quantile=0.8, summary = F, days = 10, plot = T)
# merge site helps to put sites together that are separated by single outliers.
mS <- mergeSites(twl = geo_twl, site = cL$site, degElevation = 90-zenith0, distThreshold = 500)

#specify which site is the stationary one
site           <- mS$site[mS$site>0] # get rid of movement periods
stationarySite <- which(table(site) == max(table(site))) # find the site where bird is the longest

#find the dates that the bird arrives and leaves this stationary site
start <- min(which(mS$site == stationarySite))
end   <- max(which(mS$site == stationarySite))

(zenith_sd <- findHEZenith(twl, tol=0.01, range=c(start,end)))
```

```
[1] 93.5
```


Sep
Nov
Jan
Mar
May
Sunrise (red)
Sunset (blue)
2.5
3.0
3.5
4.0
4.5
5.0
5.5
16.5
17.5
18.5
19.5
2015−07−15
2015−09−13
2015−11−12
2016−01−11
2016−03−11
2016−05−10
Probability of change
0.00
0.06
0.12


Sunrise (red)
2.5
3.5
4.5
5.5
Sunset (blue)
16.5
18.0
19.5
Longitude
0
5
10
15
20
Latitude
−40
0
40
15−Jul
19−Aug
22−Sep
27−Oct
01−Dec
04−Jan
08−Feb
13−Mar
17−Apr
21−May
Date





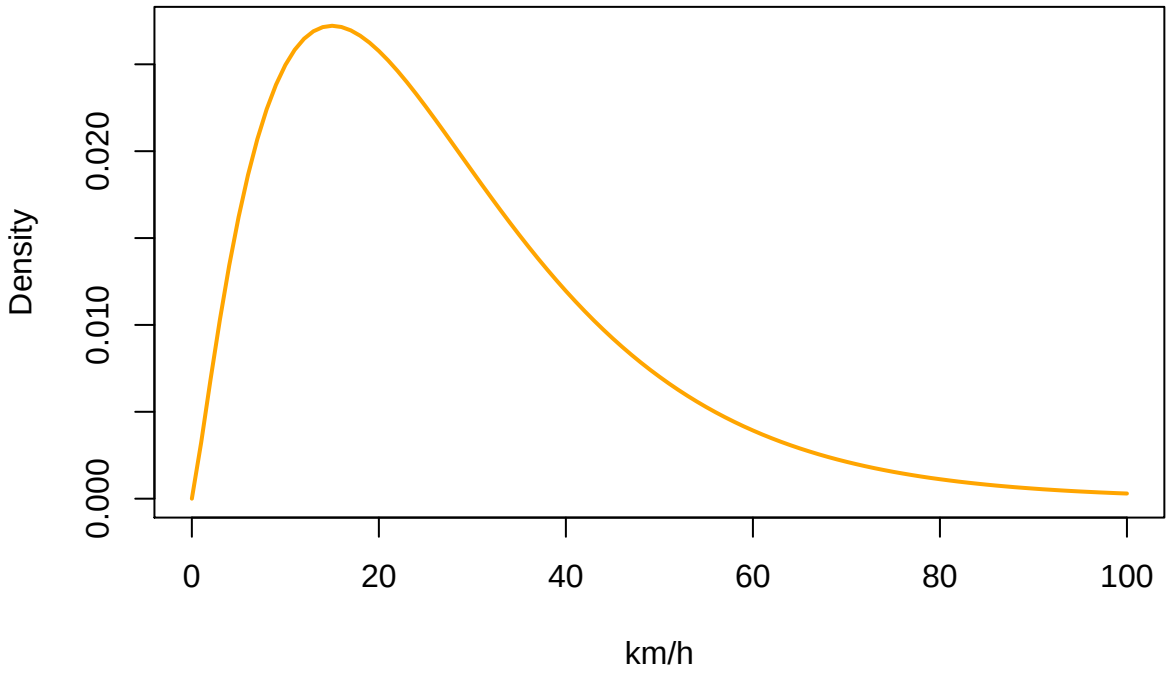
In this case, there is no real difference between the two calibrations. If a difference will be detected (>0.5 degrees), one should consider adjusting the zenith angles calculated from the breeding site.

```
zenith  <- zenith + abs(zenith0-zenith_sd)
zenith0 <- zenith_sd
```

Movement Model

We also have to generate some parameters for a basic movement model. We need to provide a mean and standard deviation for a gamma distribution of flight speeds that get applied to each day of the analysis period. We typically want short (near zero) distance flights to be common and long distance flights to be relatively rare. So both mean and distribution should be small.

```
beta  <- c(2.2, 0.08)
matplot(0:100, dgamma(0:100, beta[1], beta[2]),
        type = "l", col = "orange",lty = 1,lwd = 2,ylab = "Density", xlab = "km/h")
```

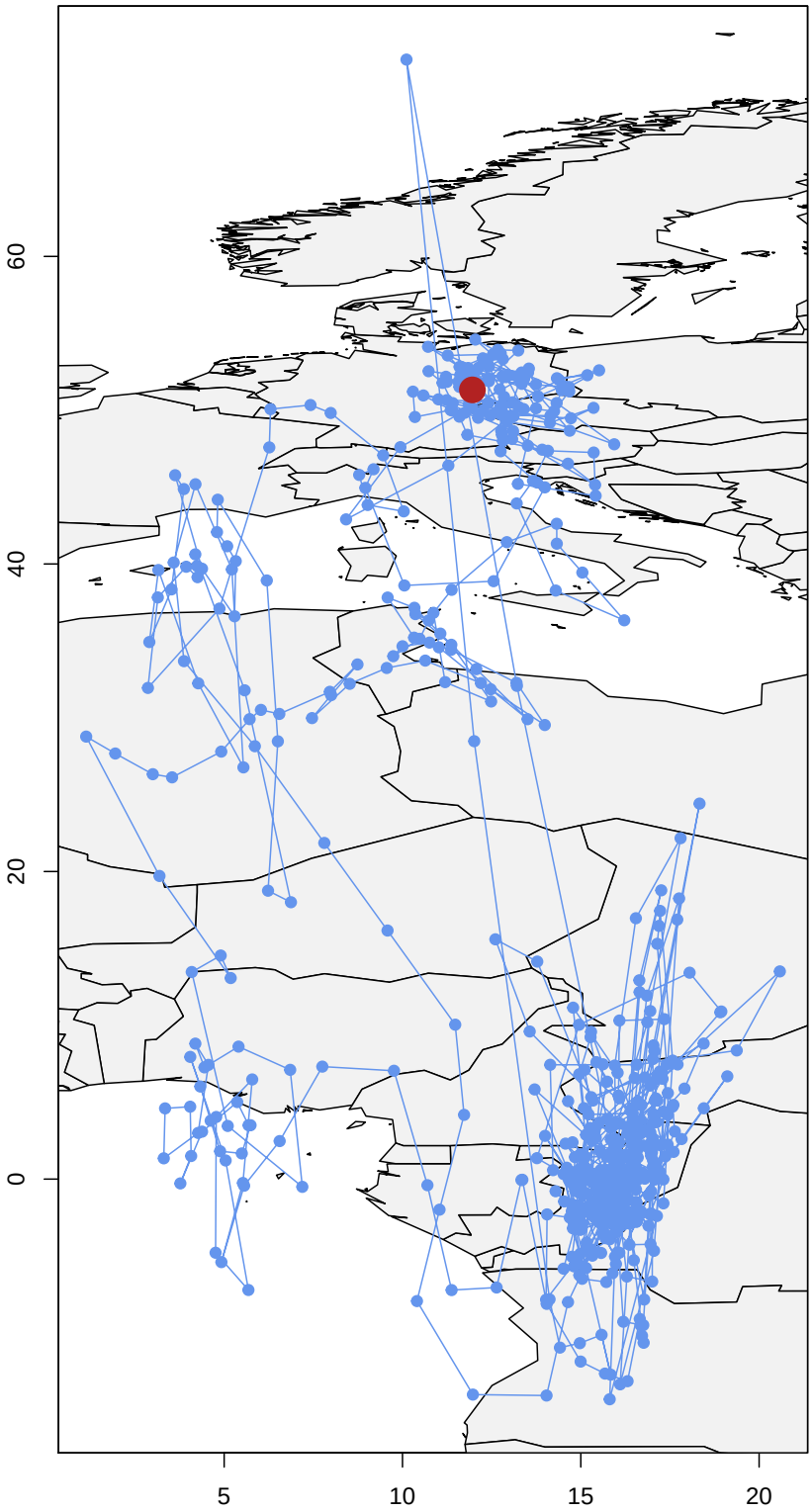


If you have a species which moves very slowly, you can have `beta = c(1,0.08)` whereas if you have a species which does moves quickly e.g. bar-tailed godwit, a larger distribution e.g. `beta = c(2.2,0.06)` might be more appropriate. Note that having a broader distribution is always better as it does not restrict the species movements. The best is to start large and then move to something narrower if the end model doesn't fit the data

Initial path

Now we need to get an initial path for the MCMC simulation as well as the midpoints between each consecutive location estimate.

```
path <- thresholdPath(twl$Twilight, twl$Rise, zenith = zenith0, tol=0.01)

x0 <- path$x
z0 <- trackMidpts(x0)

data(wrld_simpl)
plot(x0, type = "n", xlab = "", ylab = "")
plot(wrld_simpl, col = "grey95", add = T)

points(path$x, pch=19, col="cornflowerblue", type = "o")
points(lon.calib, lat.calib, pch = 16, cex = 2.5, col = "firebrick")
box()
```


60
40
20
0
5
10
15
20

Play around with `tol`. You'll notice that with e.g. `tol=0.18` you start getting straight lines. This is because tol is used to interpolate over the equinox period. A smaller tol is always better as it reduces interpolation. For an anaylsis, always start with a low tol and only increase if the model cannot deal with the noise in the data (creates impossible solutions which do not allow convergence - for instance having a bird)

Define known locations

For many tracks we know at least one location - the starting point at the deployment site. We can set this location and the sampler in the MCMC simulation will be instructed to keep these locations fixed. In this case we also know that the bird flew back to the same location, and that the geolocator was still measuring light when this happened, then we can also fix the last couple of twilight times. Theoretically, if a bird was observed during the year, any twilight time can be fixed to the location that is known.

```
fixedx <- rep(F, nrow(x0))
fixedx[1:2] <- T # first two location estimates

fixedx[(nrow(x0) - 1):nrow(x0)] <- T # last two location estimates

x0[fixedx, 1] <- lon.calib
x0[fixedx, 2] <- lat.calib

z0 <- trackMidpts(x0) # we need to update the z0 locations
```

Land mask

A land mask can be quite simple, e.g. differences in the probability of occurance between land and ocean, or highly complex, e.g. including elevation and temperature etc. Here we use a simple land-sea mask that can be created using the function `earthseaMask` below. This is something that can be customised for purpose, but for the time being we assume that bee eaters are more likely to pend time flying on land than at sea.

```
earthseaMask <- function(xlim, ylim, n = 2, pacific=FALSE) {

  if (pacific) { wrld_simpl <- nowrapRecenter(wrld_simpl, avoidGEOS = TRUE)}

  # create empty raster with desired resolution
  r = raster(nrows = n * diff(ylim), ncols = n * diff(xlim), xmn = xlim[1],
             xmx = xlim[2], ymn = ylim[1], ymx = ylim[2], crs = proj4string(wrld_simpl))

  # create a raster for the stationary period, in this case by giving land a value of 1 and sea NA
  mask = cover(rasterize(elide(wrld_simpl, shift = c(-360, 0)), r, 1, silent = TRUE),
             rasterize(wrld_simpl, r, 1, silent = TRUE),
             rasterize(elide(wrld_simpl,shift = c(360, 0)), r, 1, silent = TRUE))

  xbin = seq(xmin(mask),xmax(mask),length=ncol(mask)+1)
  ybin = seq(ymin(mask),ymax(mask),length=nrow(mask)+1)
```

```
  function(p) mask[cbind(.bincode(p[,2],ybin),.bincode(p[,1],xbin))]
}
```

This function constructs a gridded representation of the world's land masses for the region delimited by xlim and ylim with a resolution of n cells per degree and creates a look-up function that returns NA for locations that fall outside the extent of the grid, otherwise it returns TRUE or FALSE depending whether the point corresponds to land or sea.

```
xlim <- range(x0[,1]+c(-5,5))
ylim <- range(x0[,2]+c(-5,5))

mask <- earthseaMask(xlim, ylim, n = 1)
```

The location estimates derived by the following Estelle model can effectively excluded from the land by imposing a prior on the x (and z) locations so that locations on the land have a vanishingly small probability of occurrence. The prior is defined on the log scale. Here, we don't want to exclude them but give location estimates on land a higher prior.

```
## Define the log prior for x and z
log.prior <- function(p) {
    f <- mask(p)
    ifelse(f | is.na(f), log(2), log(1))
}
```

The Estelle Model

Now, we are ready to specify a model (we only use the Estelle) for the analysis. Below we specify a few key parameters.

1. `twilight` = twilight times that we determined above.
2. `rise` = a logical vector sunrise = TRUE - this is calculated at the same time when you define twilights.
3. `twilight.model` = the distribution type for the difference between observed twilight and expected twilight.
4. `alpha` = the shape of the twilight.model distribution
5. `beta` = the movement model parameter
6. `logp.x` and `logp.z` = constraints set on the x and z (intermediate) positions. This is where you set the constraints for land
7. `x0` = initial values for the birds path (x positions)
8. `z0` = initial values for the birds path (z positions)
9. `zenith` = the zenith angle to be used. This can take a single value (no change in zenith throughout the year) or a vector of `nrow(twl)` if you want to use different zenith angles.

10. `fixedx` = a vector telling the model which locations need to be estimated because positions are unknown.

First, we define a model with a `ModifiedLogNormal`twilight model. This is a more relaxed model that helps to get better starting values for the tuning and the final run.

```
model <- thresholdModel(twilight = twl$Twilight,
                        rise = twl$Rise,
                        twilight.model = "ModifiedGamma",
                        alpha = alpha,
                        beta = beta,
                        logp.x = log.prior, logp.z = log.prior,
                        x0 = x0,
                        z0 = z0,
                        zenith = zenith,
                        fixedx = fixedx)
```

We also need to define the error distribution around each location. We set that using a multivariate normal distribution. Then we can fit the model:

```
proposal.x <- mvnorm(S=diag(c(0.0025,0.0025)),n=nlocation(x0))
proposal.z <- mvnorm(S=diag(c(0.0025,0.0025)),n=nlocation(z0))

fit <- estelleMetropolis(model, proposal.x, proposal.z, iters = 1000, thin = 20)
```

Tuning the proposals

Once the chain meets the positivity constraint, the next step is to tune the proposal distributions. The model and proposals are redefined using the last set of locations from the previous run to initialize.

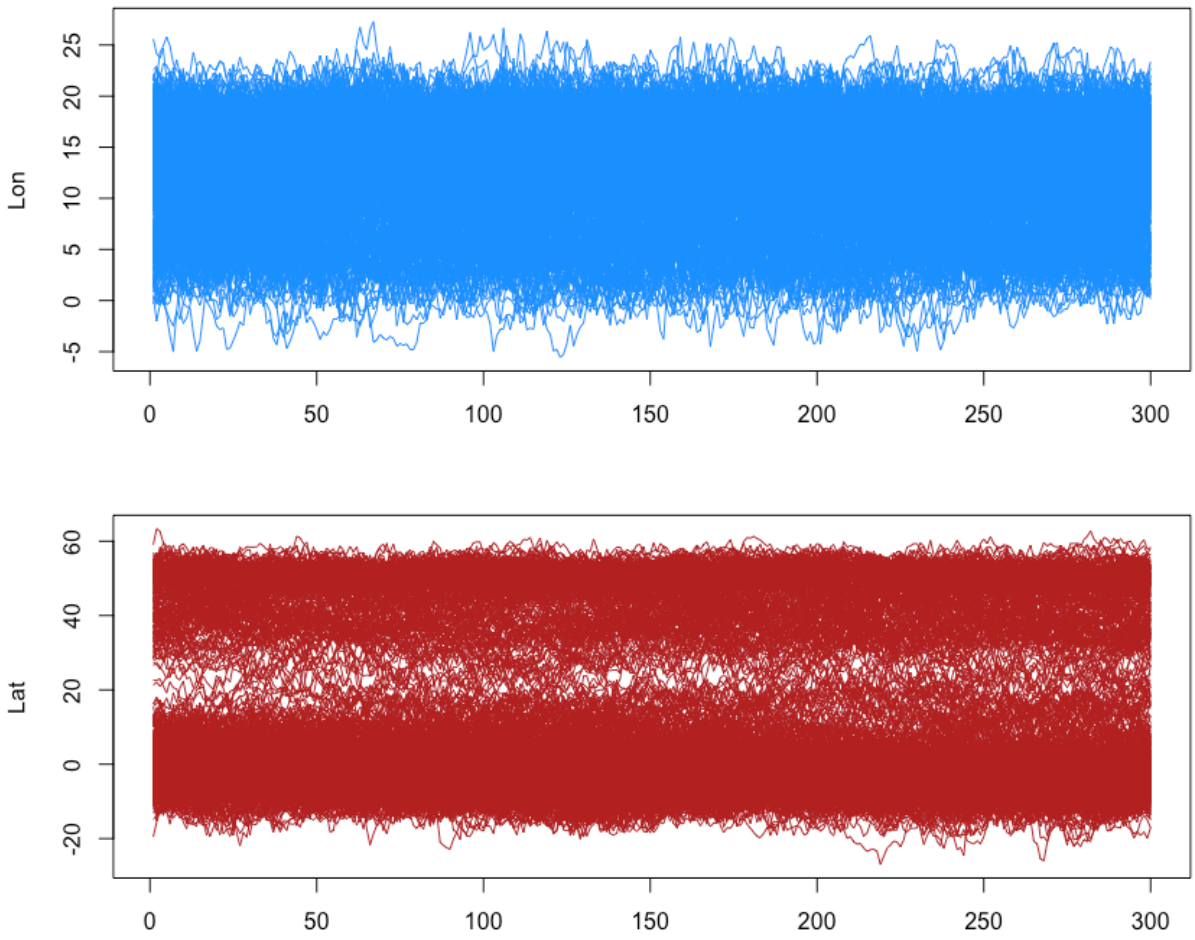
```
x0 <- chainLast(fit$x)
z0 <- chainLast(fit$z)

model <- thresholdModel(twilight = twl$Twilight,
                        rise = twl$Rise,
                        twilight.model = "Gamma",
                        alpha = alpha,
                        beta = beta,
                        logp.x = log.prior, logp.z = log.prior,
                        x0 = x0,
                        z0 = z0,
                        zenith = zenith,
                        fixedx = fixedx)

x.proposal <- mvnorm(S = diag(c(0.005, 0.005)), n = nrow(twl))
z.proposal <- mvnorm(S = diag(c(0.005, 0.005)), n = nrow(twl) - 1)
```

A number of short runs are conducted to tune the proposals. At the end of each run, new proposal distributions are defined based on the dispersion observed in the previous run.

```
for (k in 1:3) {
    fit <- estelleMetropolis(model, x.proposal, z.proposal, x0 = chainLast(fit$x),
        z0 = chainLast(fit$z), iters = 300, thin = 20)

    x.proposal <- mvnorm(chainCov(fit$x), s = 0.2)
    z.proposal <- mvnorm(chainCov(fit$z), s = 0.2)
}
```

The samples drawn through this process need to be examined to ensure the chain mixes adequately

```
opar <- par(mfrow = c(2, 1), mar = c(3, 5, 2, 1) + 0.1)
matplot(t(fit$x[[1]][!fixedx, 1, ]), type = "l", lty = 1, col = "dodgerblue", ylab = "Lon")
matplot(t(fit$x[[1]][!fixedx, 2, ]), type = "l", lty = 1, col = "firebrick", ylab = "Lat")
par(opar)
```



Final run

Once the proposals are tuned, a larger final sample is drawn.

```
x.proposal <- mvnorm(chainCov(fit$x), s = 0.25)
z.proposal <- mvnorm(chainCov(fit$z), s = 0.25)

fit <- estelleMetropolis(model, x.proposal, z.proposal, x0 = chainLast(fit$x),
                         z0 = chainLast(fit$z), iters = 1000, thin = 20)
```

Summarize the results

`locationSummary` provides the median tracks and percentiles based on the MCMC Chains from the final run.

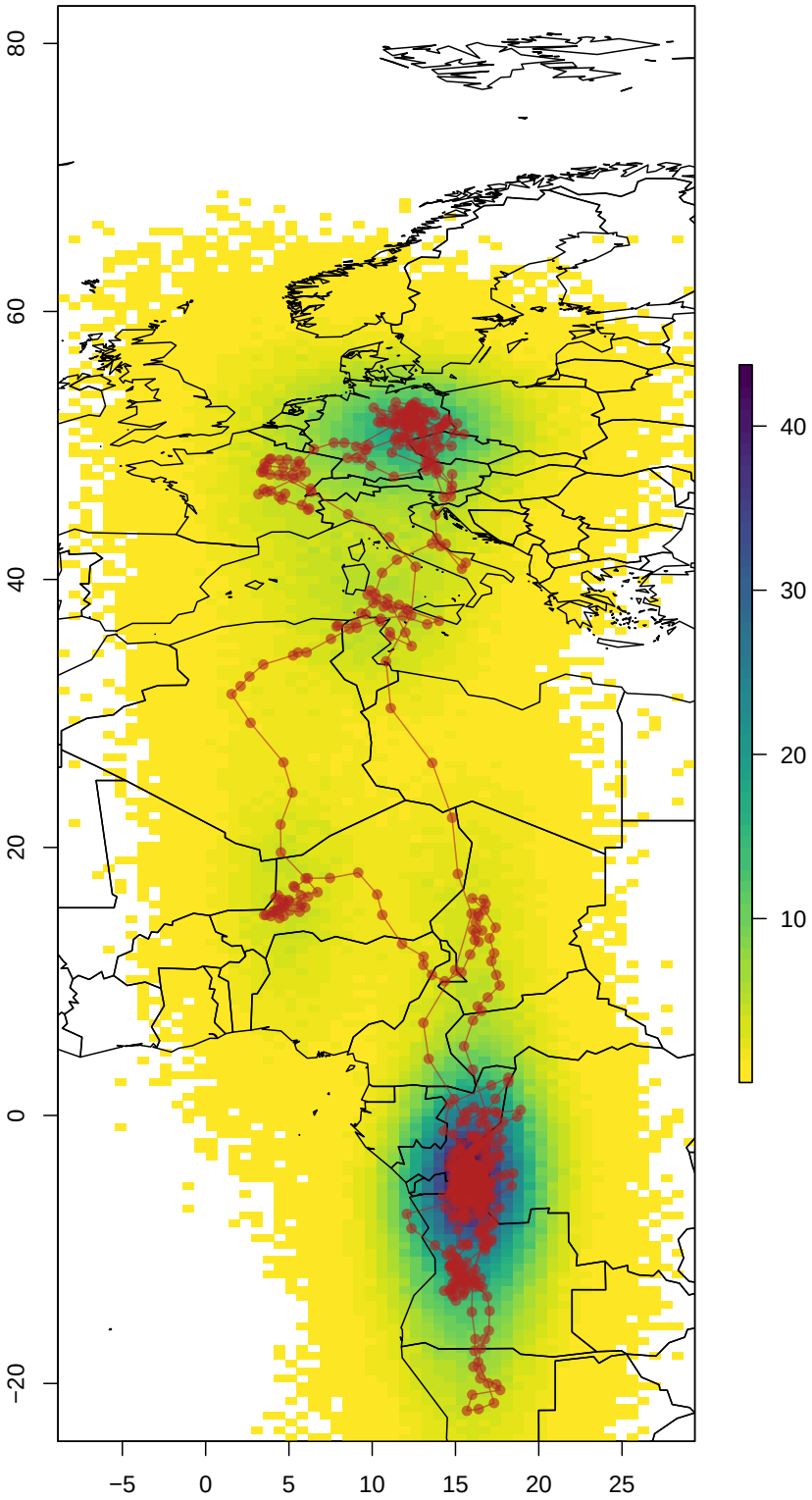
```
sm <- locationSummary(fit$z, time=fit$model$time)
head(sm)
```

```
                Time1               Time2 Lon.mean   Lon.sd  Lon.50%
1 2015-07-15 19:34:02 2015-07-16 03:01:00 11.92681 1.842167 11.98949
2 2015-07-16 03:01:00 2015-07-16 19:43:53 12.02179 4.388854 11.97274
3 2015-07-16 19:43:53 2015-07-17 02:51:06 12.13850 3.103504 12.18128
4 2015-07-17 02:51:06 2015-07-17 19:48:53 12.54457 4.906193 12.29718
5 2015-07-17 19:48:53 2015-07-18 02:46:06 12.76144 2.948357 12.80829
6 2015-07-18 02:46:06 2015-07-18 19:28:53 13.05053 4.467219 13.06342
  Lon.2.5% Lon.97.5% Lat.mean   Lat.sd  Lat.50% Lat.2.5% Lat.97.5%
1 8.273525  15.81978 51.36540 1.058634 51.42250 49.29978  53.30306
2 3.303923  20.88697 51.33786 2.627901 51.42393 46.03879  56.50354
3 5.785892  18.06353 51.25457 2.036834 51.24177 47.27354  55.53580
4 3.134997  22.77443 51.55448 3.030207 51.51746 45.72056  57.66152
5 7.077072  18.69472 51.79386 1.873417 51.79962 48.17154  55.36501
6 4.242315  22.28707 50.73941 2.855003 50.77695 45.04796  56.31163
```

Plotting the results

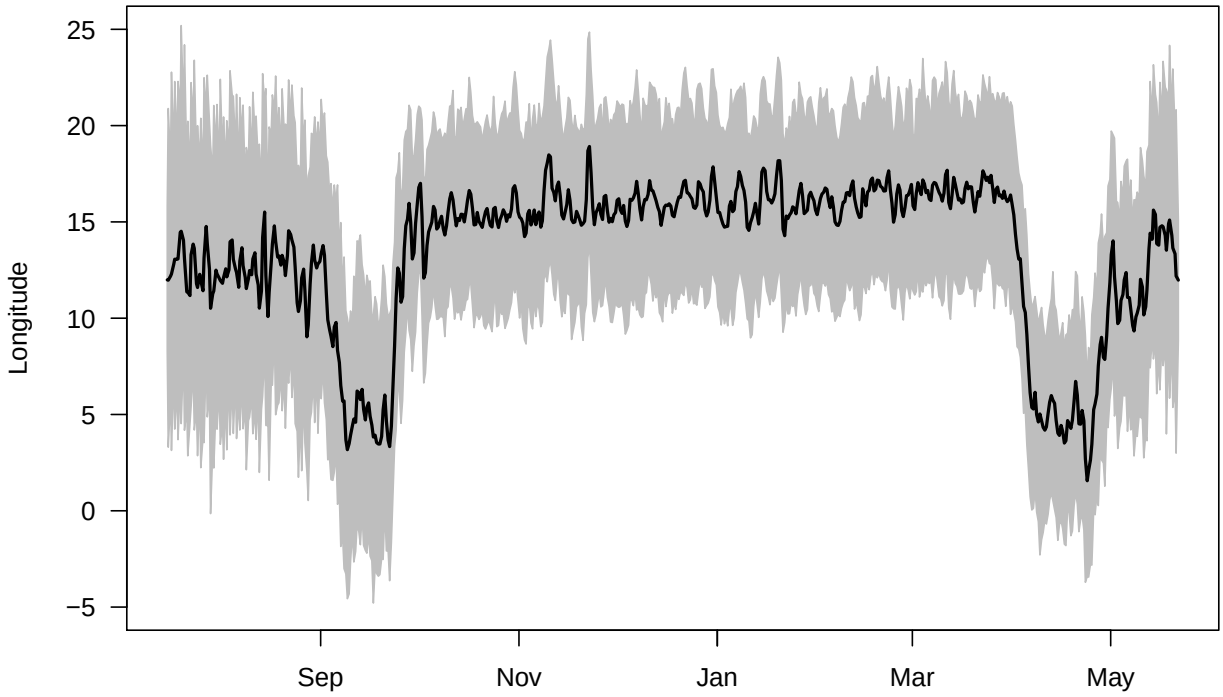
The results can be presented in many ways, here‘s just a quick one.

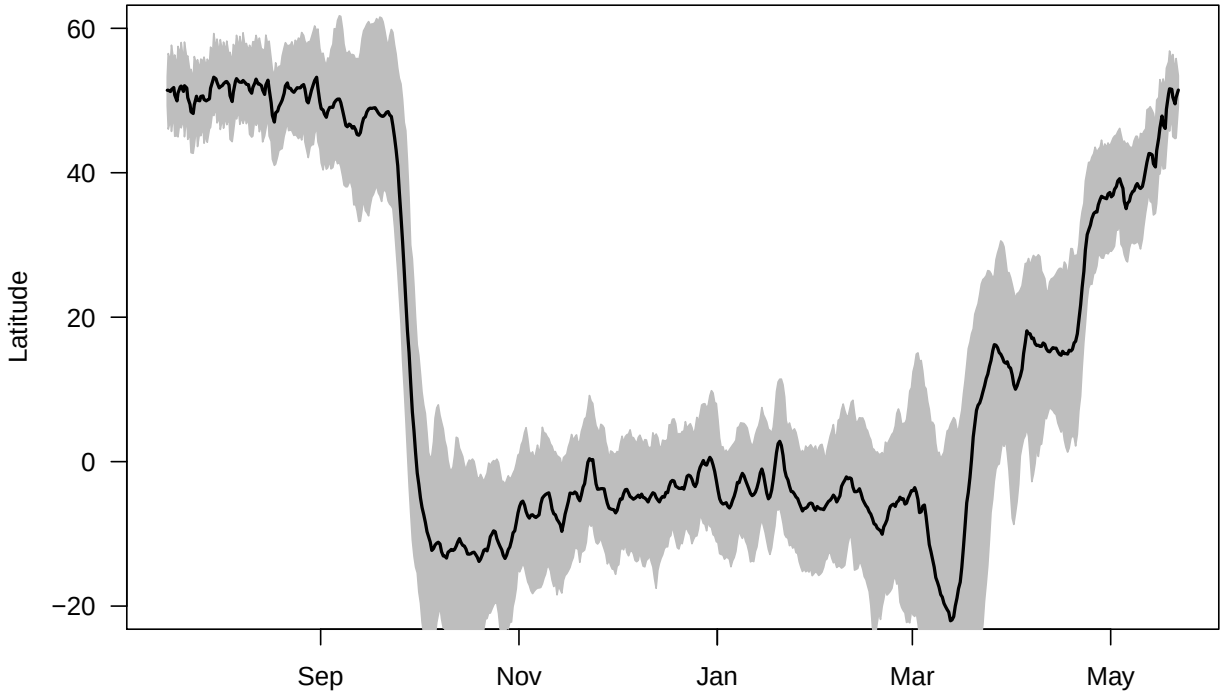
```
# empty raster of the extent
r <- raster(nrows = 2 * diff(ylim), ncols = 2 * diff(xlim), xmn = xlim[1]-5,
            xmx = xlim[2]+5, ymn = ylim[1]-5, ymx = ylim[2]+5, crs = proj4string(wrld_simpl))

s <- slices(type = "intermediate", breaks = "week", mcmc = fit, grid = r)
sk <- slice(s, sliceIndices(s))

plot(sk, useRaster = F,col = rev(viridis::viridis(50)))
plot(wrld_simpl, xlim=xlim, ylim=ylim,add = T, bg = adjustcolor("black",alpha=0.1))

lines(sm[,"Lon.50%"], sm[,"Lat.50%"], col = adjustcolor("firebrick", alpha.f = 0.6), type = "o", pch =
```


80
60
40
20
0
−20
−5
0
5
10
15
20
25
40
30
20
10

Additionally, we can plot the Longitudes and Latitudes separately with their confidence intervals.

```
par(mfrow=c(2,1),mar=c(4,4,1,1))
plot(sm$Time1, sm$"Lon.50%", ylab = "Longitude", xlab = "", yaxt = "n", type = "n", ylim = c(-5, 25))
axis(2, las = 2)
polygon(x=c(sm$Time1,rev(sm$Time1)), y=c(sm$`Lon.2.5%`,rev(sm$`Lon.97.5%`)), border="gray", col="gray")
lines(sm$Time1,sm$"Lon.50%", lwd = 2)

plot(sm$Time1,sm$"Lat.50%", type="n", ylab = "Latitude", xlab = "", yaxt = "n", ylim = c(-20,60))
axis(2, las = 2)
polygon(x=c(sm$Time1,rev(sm$Time1)), y=c(sm$`Lat.2.5%`,rev(sm$`Lat.97.5%`)), border="gray", col="gray")
lines(sm$Time1,sm$"Lat.50%", lwd = 2)
```


Longitude
25
20
15
10
5
0
−5
Sep
Nov
Jan
Mar
May


Latitude
60
40
20
0
−20
Sep
Nov
Jan
Mar
May

Saving the Results

We want to save the summary file as well as the MCMC chains in case we want to summarize them differently in the future. We also need the chains to make maps with a density distribution or similar presentations of the results.

```
write.csv(sm,
          paste0(wd, "/Results/", Species, "/", ID, "_SGATSummary.csv"),
          row.names = F)

save(fit,
     file = paste0(wd, "/Results/", Species, "/", ID, "_SGATfit.Rdata"),
     compress = T)
```

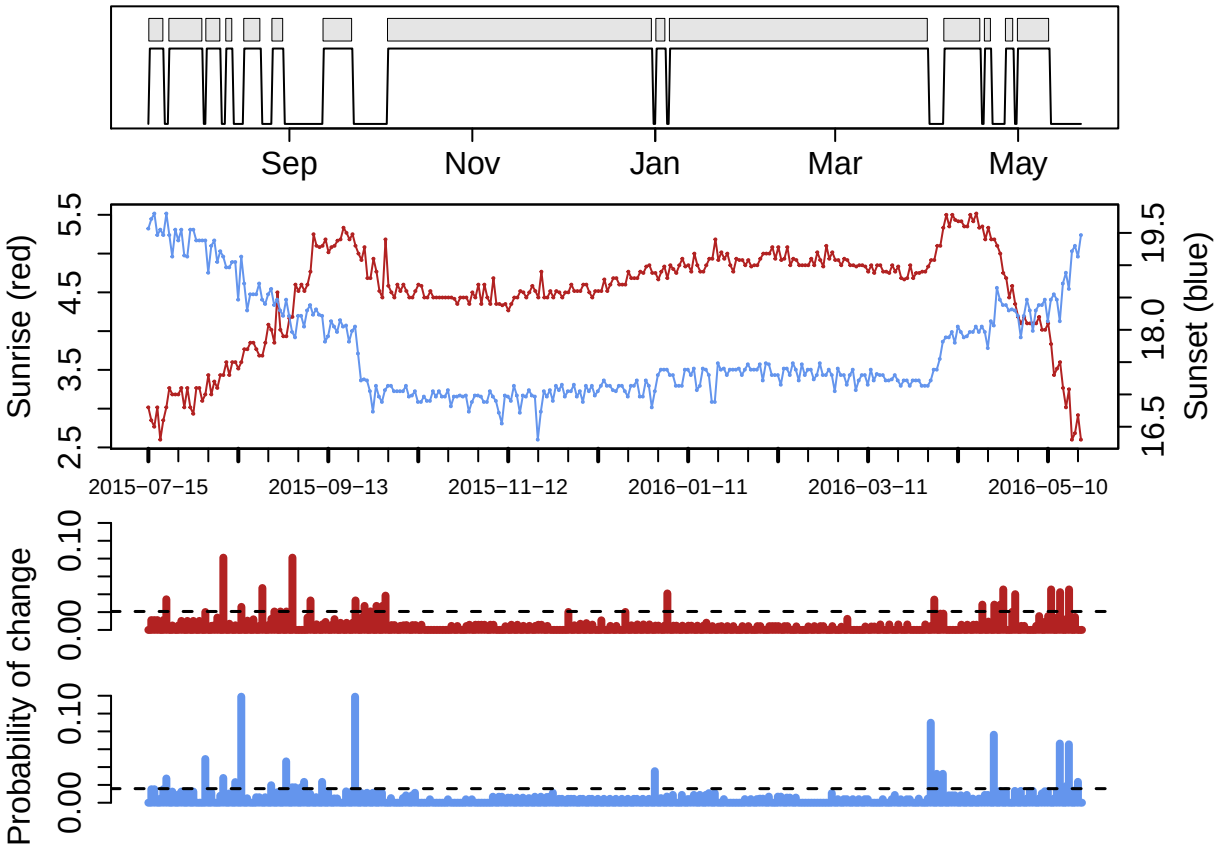
The Groupe Model

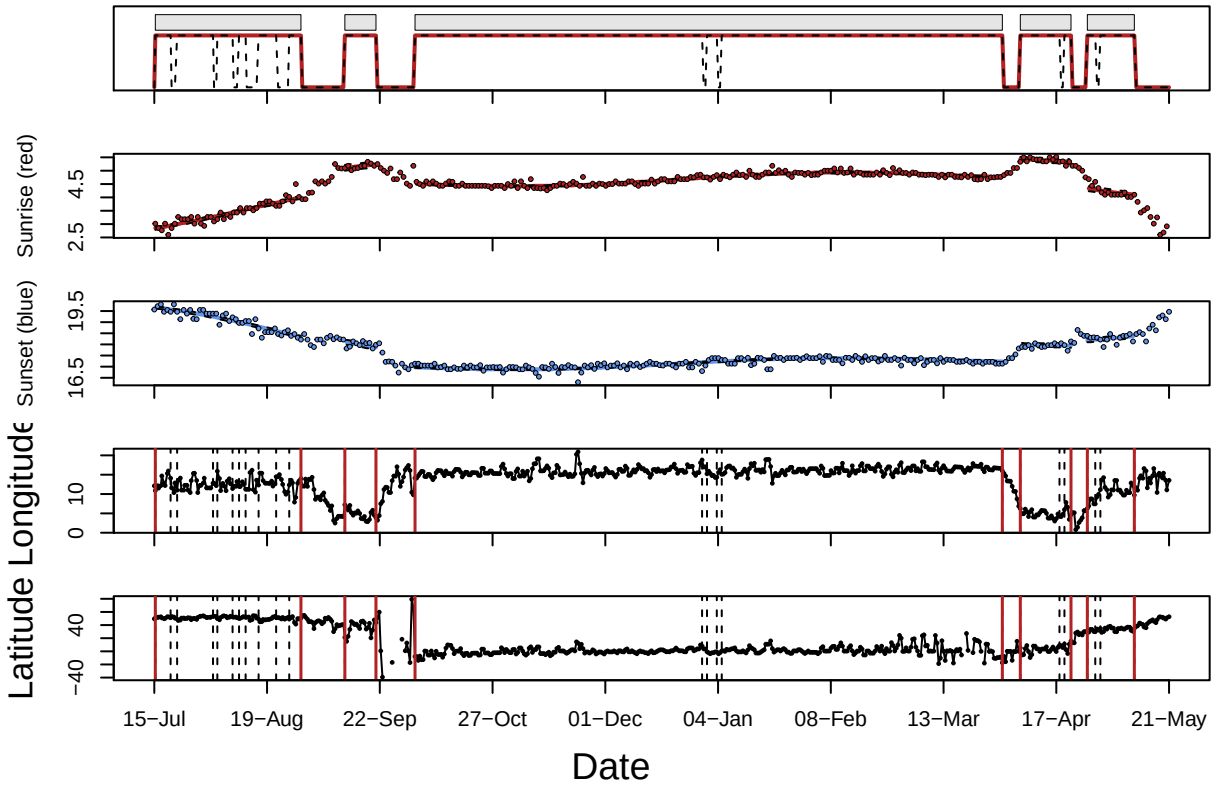
The group model is a special case of the estelle model discussed above. It allows group twilight times together which are then treated as a set of twilight times recorded at one single location. The location is thus the best spatial representation of a group of sunrise and sunset times.

To realise the grouping one could use the changepoint analyses from `GeoLight` that separates periods of residency from periods of movement based in changes in sunrise and sunset times.

We start with the twl file that needs reformatting to match the `GeoLight` requirements.

```
geo_twl <- export2GeoLight(twl)

# Often it is necessary to play around with quantile and days
# quantile defines how many stopovers there are. the higher, the fewer there are
# days indicates the duration of the stopovers
cL <- changeLight(twl=geo_twl, quantile=0.86, summary = F, days = 2, plot = T)

# merge site helps to put sites together that are separated by single outliers.
mS <- mergeSites(twl = geo_twl, site = cL$site, degElevation = 90-zenith0, distThreshold = 500)
```


Sep
Nov
Jan
Mar
May
Sunrise (red)
2.5
3.5
4.5
5.5
Sunset (blue)
16.5
18.0
19.5
2015−07−15
2015−09−13
2015−11−12
2016−01−11
2016−03−11
2016−05−10
Probability of change
0.00
0.10
0.00
0.10



Play around with `distThreshold` in `mergeSites`, and `quantile` and `days` in `changeLight` and see how results change. It can help to look at how latitudes are classed by `mergeSites`. If there are large changes in longitude within the same stationary site, then it is worth reducing the `quantile` to allow more movement or increasing the `distThreshold`. Overall, for a SGAT grouped model, it's best to allow a lot of movement and only have stopovers that are certain classed as stopovers.

The plot shows the sites that have been identified and merged (red line in top panes represents the merged sites). See GeoLight for more information on this analysis.

We know have to back transfer the twilight table and create a group vector with TRUE and FALSE according to which twilights to merge.

```
twl.rev <- data.frame(Twilight = as.POSIXct(geo_twl[,1], geo_twl[,2]),
                      Rise     = c(ifelse(geo_twl[,3]==1, TRUE, FALSE), ifelse(geo_twl[,3]==1, FALSE, TR
                      Site     = rep(mS$site,2))
twl.rev <- subset(twl.rev, !duplicated(Twilight), sort = Twilight)

grouped <- rep(FALSE, nrow(twl.rev))
grouped[twl.rev$Site>0] <- TRUE
grouped[c(1:3, (length(grouped)-2):length(grouped))] <- TRUE
```

```
# Create a vector which indicates which numbers sites as 11112344444567888889
g <- makeGroups(grouped)

# Add data to twl file
twl$group <- c(g, g[length(g)])


# Add behavior vector
behaviour <- c()
for (i in 1:max(g)){
  behaviour<- c(behaviour, which(g==i)[1])
}
stationary <- grouped[behaviour]
sitenum <- cumsum(stationary==T)
sitenum[stationary==F] <- 0
```

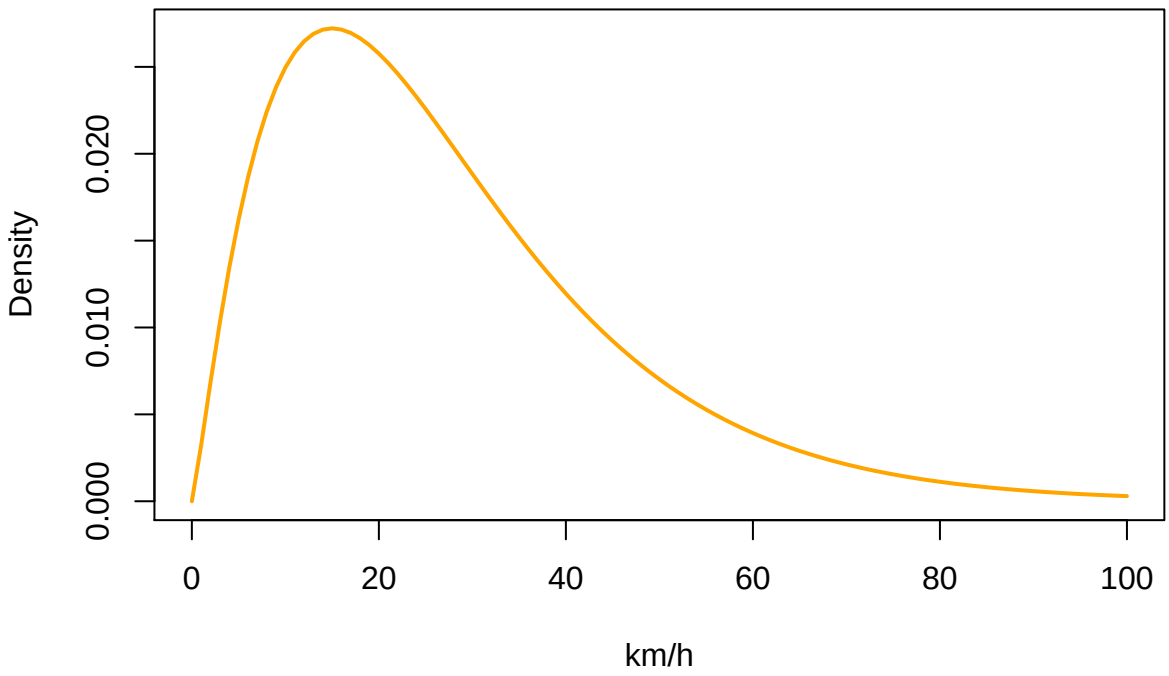
Initiate the model

The initial path as well as the fixed vector needs to be slightly different, e.g. only one value for a group of twilights.

```
x0 <- cbind(tapply(path$x[,1],twl$group,median),
            tapply(path$x[,2],twl$group,median))


fixedx <- rep_len(FALSE, length.out = nrow(x0))
fixedx[1] <- TRUE
fixedx[c(1, length(fixedx))] <- TRUE

x0[fixedx,1] <- lon.calib
x0[fixedx,2] <- lat.calib

z0 <- trackMidpts(x0)
```

For the movement model we also use different parameters since those should now only reflect the speeds during active flight.

```
beta  <- c(2.2, 0.08)
matplot(0:100, dgamma(0:100, beta[1], beta[2]),
        type = "l", col = "orange",lty = 1,lwd = 2,ylab = "Density", xlab = "km/h")
```



Land mask for group model

Now that we know when birds are stationary and when they are not, we can change the mask to ensure that when birds are stationary, they are on land, and that when they are moving/migrating, they can go anywhere. We can therefore create two rasters, one for movement and one for stationary periods which we can then access using an index derived from `stationary`.

```
earthseaMask <- function(xlim, ylim, n = 2, pacific=FALSE, index) {

  if (pacific) { wrld_simpl <- nowrapRecenter(wrld_simpl, avoidGEOS = TRUE)}

  # create empty raster with desired resolution
  r = raster(nrows = n * diff(ylim), ncols = n * diff(xlim), xmn = xlim[1],
             xmx = xlim[2], ymn = ylim[1], ymx = ylim[2], crs = proj4string(wrld_simpl))

  # create a raster for the stationary period, in this case by giving land a value of 1
  rs = cover(rasterize(elide(wrld_simpl, shift = c(-360, 0)), r, 1, silent = TRUE),
             rasterize(wrld_simpl, r, 1, silent = TRUE),
             rasterize(elide(wrld_simpl,shift = c(360, 0)), r, 1, silent = TRUE))

  # make the movement raster the same resolution as the stationary raster, but allow the bird to go any
  rm = rs; rm[] = 1

  # stack the movement and stationary rasters on top of each other
  mask = stack(rs, rm)
```

```
  xbin = seq(xmin(mask),xmax(mask),length=ncol(mask)+1)
  ybin = seq(ymin(mask),ymax(mask),length=nrow(mask)+1)
  mask = as.array(mask)[nrow(mask):1,,sort(unique(index)),drop=FALSE]

  function(p) mask[cbind(.bincode(p[,2],ybin),.bincode(p[,1],xbin), index)]
}
```

We can then create the mask in a similar manner to before, but now with an index which we derive from `stationary`:

```
xlim <- range(x0[,1]+c(-5,5))
ylim <- range(x0[,2]+c(-5,5))

index = ifelse(stationary, 1, 2)

mask <- earthseaMask(xlim, ylim, n = 1, index=index)
```

The location estimates derived by the following Estelle model can effectively excluded from the land by imposing a prior on the x (and z) locations so that locations on sea are highly unlikely during the stationary period. The prior is defined on the log scale. Here, we do want to exclude them but give location estimates on land a higher prior.

```
## Define the log prior for x and z
logp <- function(p) {
    f <- mask(p)
    ifelse(f | is.na(f), -1000, log(1))
}
```

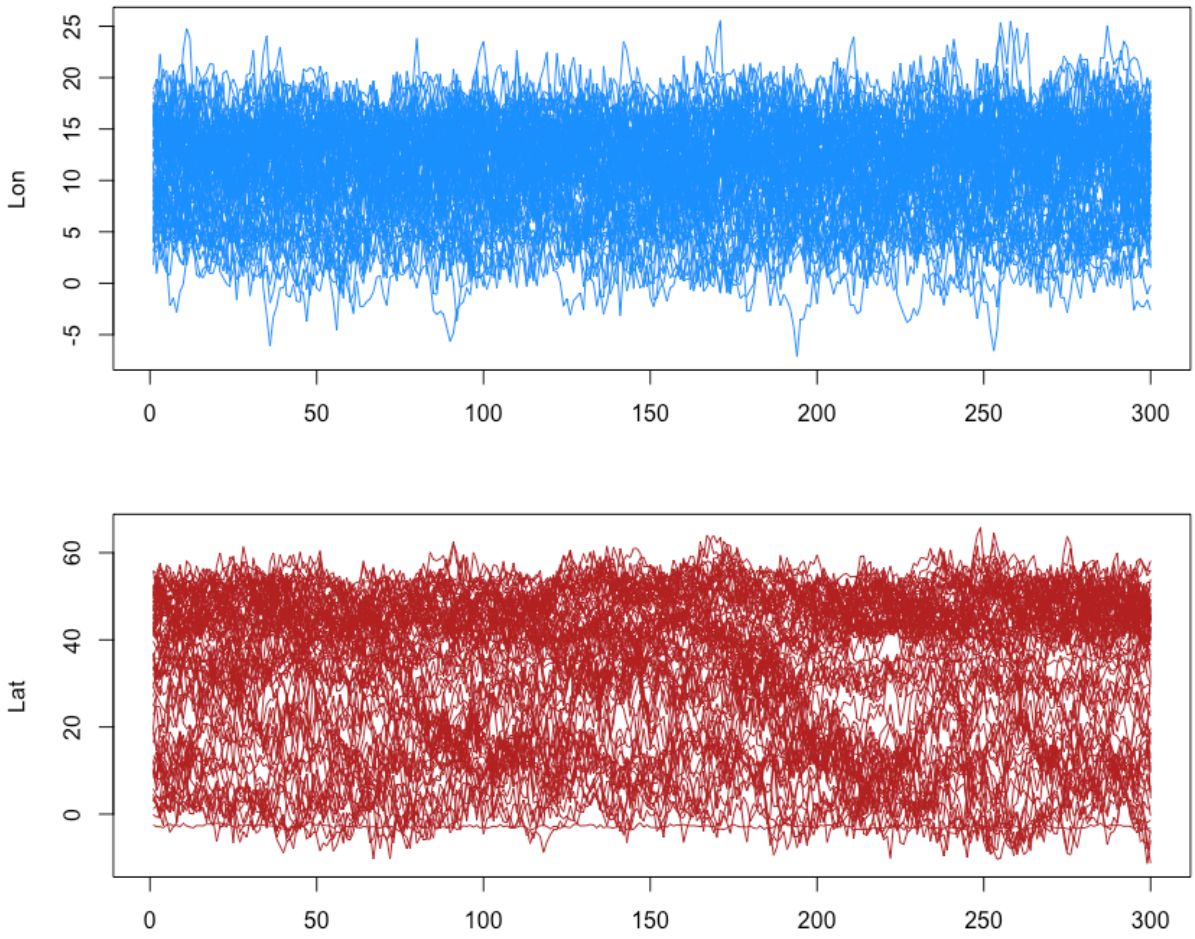
The Estelle Model

Now we can define the model (again a relaxed model first).

```
model <- groupedThresholdModel(twl$Twilight,
                               twl$Rise,
                               group = twl$group, #This is the group vector for each time the bird was
                               twilight.model = "ModifiedGamma",
                               alpha = alpha,
                               beta =  beta,
                               x0 = x0, # meadian point for each greoup (defined by twl$group)
                               z0 = z0, # middle points between the x0 points
                               zenith = zenith,
                               logp.x = logp, # land sea mask
                               fixedx = fixedx)


# define the error shape
x.proposal <- mvnorm(S = diag(c(0.005, 0.005)), n = nrow(x0))
z.proposal <- mvnorm(S = diag(c(0.005, 0.005)), n = nrow(z0))

# Fit the model
fit <- estelleMetropolis(model, x.proposal, z.proposal, iters = 1000, thin = 20)
```

Tuning

```
# use output from last run
x0 <- chainLast(fit$x)
z0 <- chainLast(fit$z)

model <- groupedThresholdModel(twl$Twilight,
                               twl$Rise,
                               group = twl$group,
                               twilight.model = "Gamma",
                               alpha = alpha,
                               beta =  beta,
                               x0 = x0, z0 = z0,
                               logp.x = logp,
                               missing=twl$Missing,
                               zenith = zenith,
                               fixedx = fixedx)

for (k in 1:3) {
  x.proposal <- mvnorm(chainCov(fit$x), s = 0.3)
  z.proposal <- mvnorm(chainCov(fit$z), s = 0.3)
  fit <- estelleMetropolis(model, x.proposal, z.proposal, x0 = chainLast(fit$x),
                           z0 = chainLast(fit$z), iters = 300, thin = 20)
}

## Check if chains mix
opar <- par(mfrow = c(2, 1), mar = c(3, 5, 2, 1) + 0.1)
matplot(t(fit$x[[1]][!fixedx, 1, ]), type = "l", lty = 1, col = "dodgerblue", ylab = "Lon")
matplot(t(fit$x[[1]][!fixedx, 2, ]), type = "l", lty = 1, col = "firebrick", ylab = "Lat")
par(opar)
```



Final run

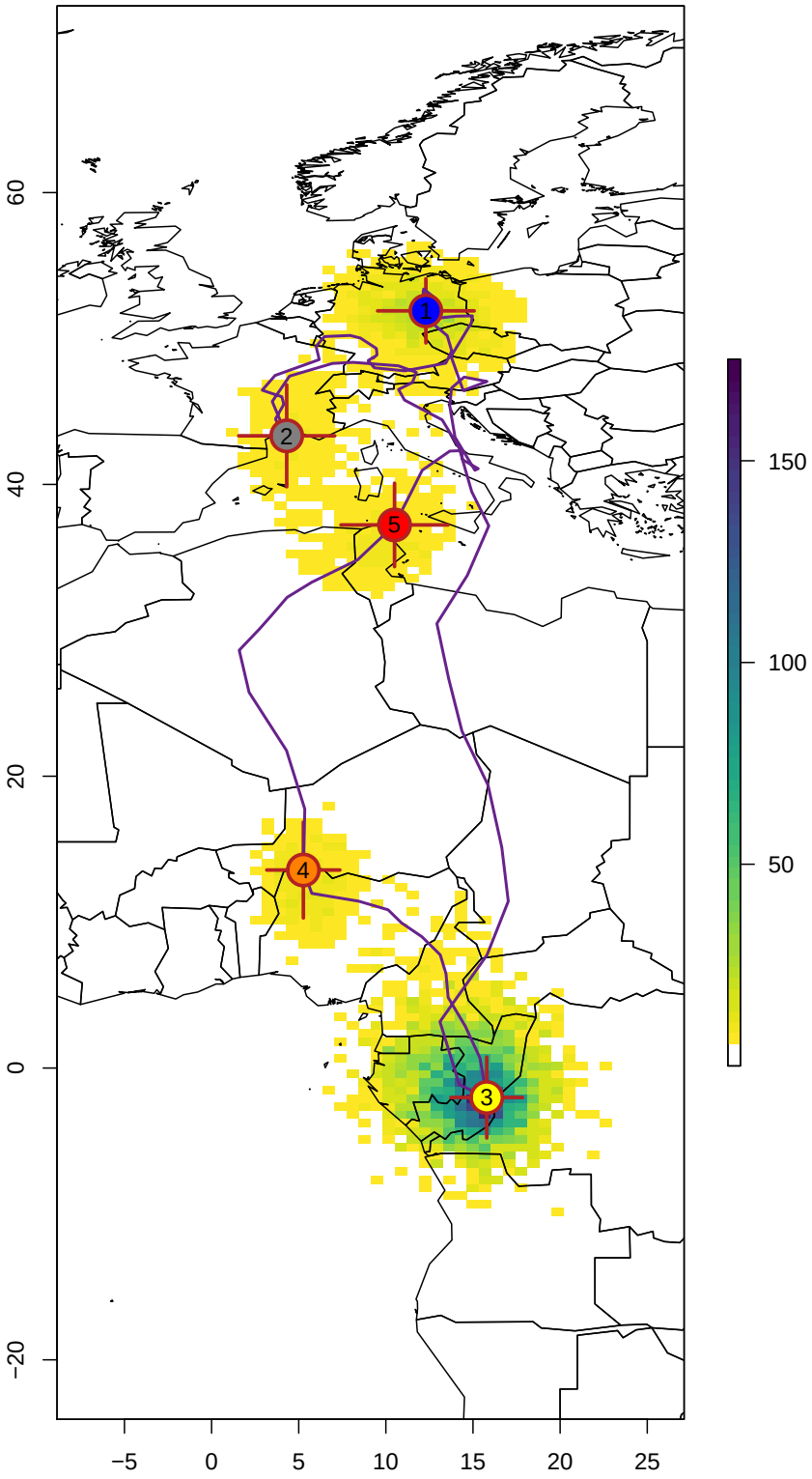
```
x.proposal <- mvnorm(chainCov(fit$x), s = 0.3)
z.proposal <- mvnorm(chainCov(fit$z), s = 0.3)

fit <- estelleMetropolis(model, x.proposal, z.proposal, x0 = chainLast(fit$x),
                         z0 = chainLast(fit$z), iters = 2000, thin = 20, chain = 1)
```

Summarize the results

```
sm <- locationSummary(fit$z, time=fit$model$time)
```

Plotting the results

```
colours <- c("black",colorRampPalette(c("blue","yellow","red"))(max(twl.rev$Site)))
data(wrld_simpl)

# empty raster of the extent
r <- raster(nrows = 2 * diff(ylim), ncols = 2 * diff(xlim), xmn = xlim[1]-5,
            xmx = xlim[2]+5, ymn = ylim[1]-5, ymx = ylim[2]+5, crs = proj4string(wrld_simpl))

s <- slices(type = "intermediate", breaks = "week", mcmc = fit, grid = r)
sk <- slice(s, sliceIndices(s))

plot(sk, useRaster = F,col = c("transparent", rev(viridis::viridis(50))))
plot(wrld_simpl, xlim=xlim, ylim=ylim,add = T, bg = adjustcolor("black",alpha=0.1))

with(sm[sitenum>0,], arrows(`Lon.50%`, `Lat.50%`+`Lat.sd`, `Lon.50%`, `Lat.50%`-`Lat.sd`, length = 0, lw
with(sm[sitenum>0,], arrows(`Lon.50%`+`Lon.sd`, `Lat.50%`, `Lon.50%`-`Lon.sd`, `Lat.50%`, length = 0, lw
lines(sm[,"Lon.50%"], sm[,"Lat.50%"], col = "darkorchid4", lwd = 2)

points(sm[,"Lon.50%"], sm[,"Lat.50%"], pch=21, bg=colours[sitenum+1],
       cex = ifelse(sitenum>0, 3, 0), col = "firebrick", lwd = 2.5)

points(sm[,"Lon.50%"], sm[,"Lat.50%"], pch=as.character(sitenum),
       cex = ifelse(sitenum>0, 1, 0))
```


60
40
20
0
−20
−5
0
5
10
15
20
25
1
2
3
4
5
150
100
50

Saving the Results

We again want to save the summary file as well as the MCMC Chains in case we want to summarize them differently in the future. We also need the chains to make maps with a density distribution or similar presentations of the results.

```
write.csv(sm,
          paste0(wd, "/Results/", Species, "/", ID, "_SGATGroupSummary.csv"),
          row.names = F)

save(fit,
     file = paste0(wd, "/Results/", Species, "/", ID, "_SGATGroupfit.RData"),
     compress = T)
```

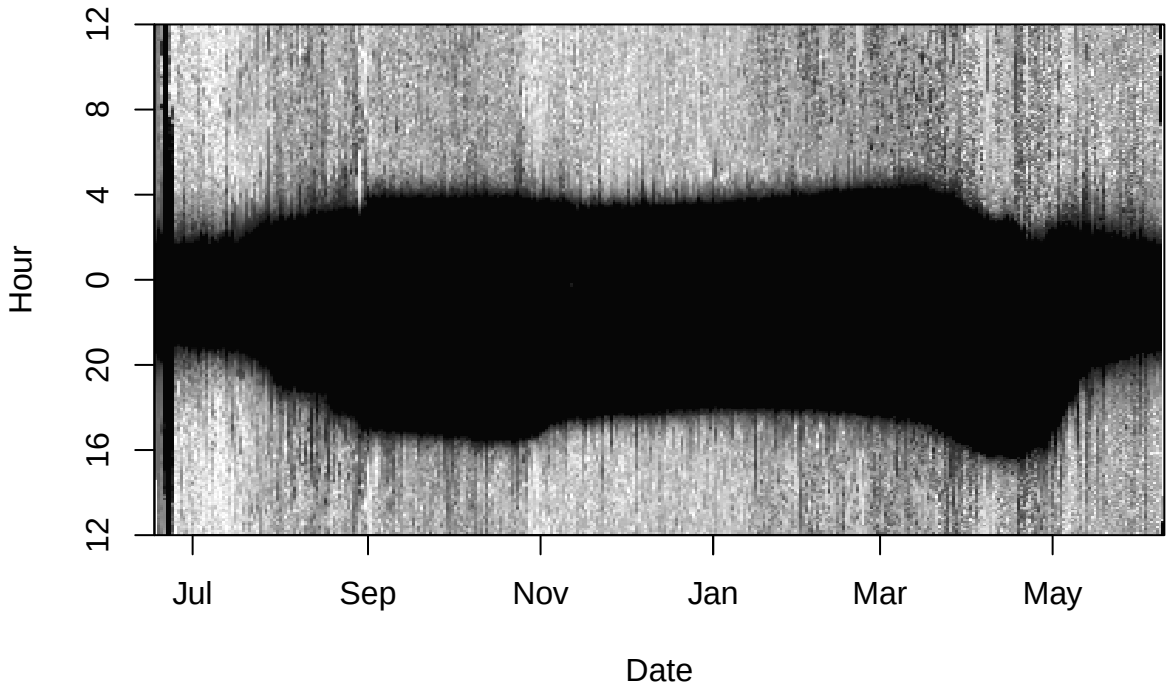
Chapter 8

FLightR

Getting started

We first define the metadata and read in the raw recordings. We then use the `ligthImage` to get a first look at the data.

```
ID <- "M034"
wd <- "Data"
Species <- "LanCol"

lon.calib <- 12.33
lat.calib <- 55.98

raw <- readMTlux(paste0(wd, "/RawData/", Species, "/", ID, ".lux"))
  names(raw) <- c("Date", "Light")
  raw$Light  <- log(raw$Light+0.0001) + abs(min(log(raw$Light+0.0001)))

offset <- 12 # adjusts the y-axis to put night (dark shades) in the middle

lightImage( tagdata = raw,
            offset = offset,
            zlim = c(0, 10))
```



We skip the twilight annotation that can be done using the discussed tools (e.g. `preprocessLight`). In this case we use a twiligth file that produced with the online platform *TAGS*. *FLightR* works efficiently with the output of *TAGS*, which are CSV files containing the following fields:

- `datetime` – date and time in ISO 8601 format e.g. 2013-06-16T00:00:11.000Z;
- `light` – light value measured by tag;
- `twilight` – assigned by the software numeric indication of whether the record belongs to sunrise (1), sunset (2) or none of those (0);
- `excluded` – indication of whether a twilight was excluded during manual inspection (logical, `TRUE | FALSE`);
- `interp` - indication of whether the light value at twilight was interpolated (logical, `TRUE | FALSE`). The fields `excluded` and `interp` may have values of `TRUE` only for `twilight > 0`. The online service http://tags.animalmigration.org saves data in the TAGS format. In the R packages GeoLight and BAStag or twGeos, the annotated twilight data need to be exported to TAGS, for which the functions in the FLightR (`GeoLight2TAGS`, `BAStag2TAGS` or `twGeos2TAGS`) can be used.

The function `get.tags.data` reads comma separated file in the TAGS format, detects the tag type, checks whether the light data are log-transformed, transforms them back from the log scale if needed and creates an object, containing

- the recorded light data,
- the detected twilight events,
- the light level data at the moment of each determined sunrise and sunset and around them (24 fixes before and 24 after it events into an object of two lists
- technical parameters of the tag, added automatically, unless preset by the user.

The finction works with all the common tag types: mk tags (produced by British Antarctic Survey, Lotek, and Migrate Technology Ltd.), Intigeo tags (Migrate technology Ltd.) and GDL tags (Swiss Ornithological Institute).

```
FLightR.data<-get.tags.data(paste0(wd, "/Results/", Species, "/", ID, "_twl.csv"))
```

```
Detected Intigeo_Mode_1 tag
Data found to be logtransformed
tag saved data every 300 seconds, and is assumed to measure data every 60 seconds, and write down max
```

Calibration

Geolocators measure light levels with different precision, and calibration is needed to establish the relationship between the observed and the expected light levels for each device. This relationship is depicted by the calibration parameters (slopes), calculated using the data recorded in known (calibration) geographic positions, e.g. where the animal was tagged, recaptured or observed. FLightR uses a 'template fit' for calibration Ekstrom 2004, 2007. For each tag it finds the linear (on a log-log scale) relationship between the light levels measured in known locations and the theoretical light levels, estimated from current sun angle in these locations with the deterministic equation developed by Ekstrom Rakhimberdiev et al. 2015.

To calculate the calibration parameters user needs to create a data frame where the geographic coordinates of the calibration location, and the start and end dates of the calibration period, i. e. the period of residence in the known location, are specified:

- `calibration.start` (POSIXct format)
- `calibration.stop` (POSIXct format)
- `lon` (numeric)
- `lat` (numeric)

The data frame contains as many rows as many distinct calibration periods the track contains.

```
Calibration.periods<-data.frame(
        calibration.start=as.POSIXct(c(NA, '2015-05-15')),
        calibration.stop=as.POSIXct(c('2014-07-15', NA)),
        lon=lon.calib, lat=lat.calib)
            # use c() also for the geographic coordinates,
            # if you have more than one calibration location
            # (e. g.,  lon=c(5.43, 6.00), lat=c(52.93,52.94))
print(Calibration.periods)
```

```
  calibration.start calibration.stop   lon   lat
1              <NA>       2014-07-15 12.33 55.98
2        2015-05-15             <NA> 12.33 55.98
```

Please carefully look at the data.frame you have created!

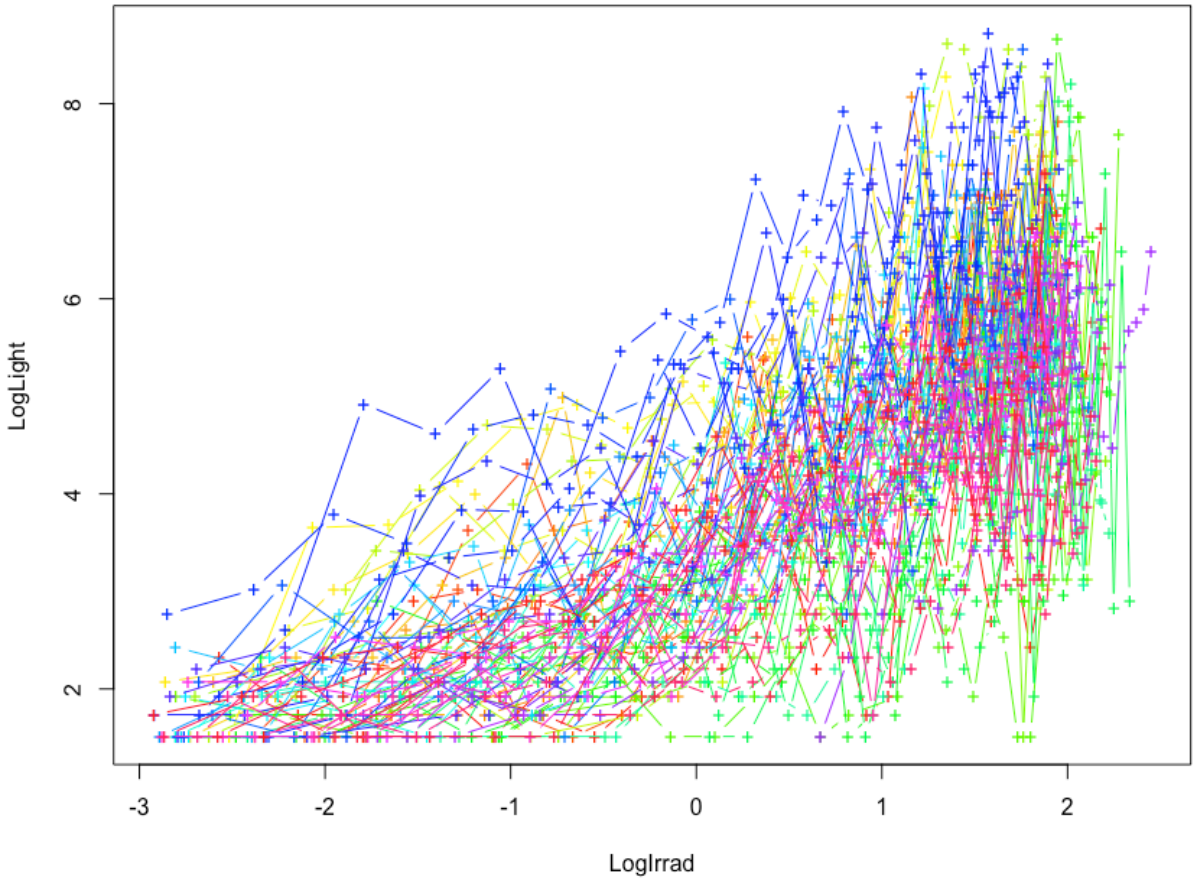
The first column should contain dates of the calibration start and the second - ends of the calibration periods.

In this example, we have two calibration periods in the same location, at the beginning and at the end of the track. This is a common case, as the birds are often recaptured at the same location, where they were tagged.

When multiple calibration locations are available, each of them has to be processed with the function `plot_slopes_by_location`. In this case, in the `Calibration periods` data frame, each row should refer to one calibration period. Compiling the data frame with multiple calibration locations, use `c()` also for the geographic coordinates (e. g., `lon=c(5.43, 6.00), lat=c(52.93,52.94)`).

A `Calibration` object is compiled with the function`make.calibration` from the created `Calibration periods` data frame and the `FLightR.data` object.

```
Calibration<-make.calibration(FLightR.data, Calibration.periods, model.ageing=TRUE, plot.final = T)
```

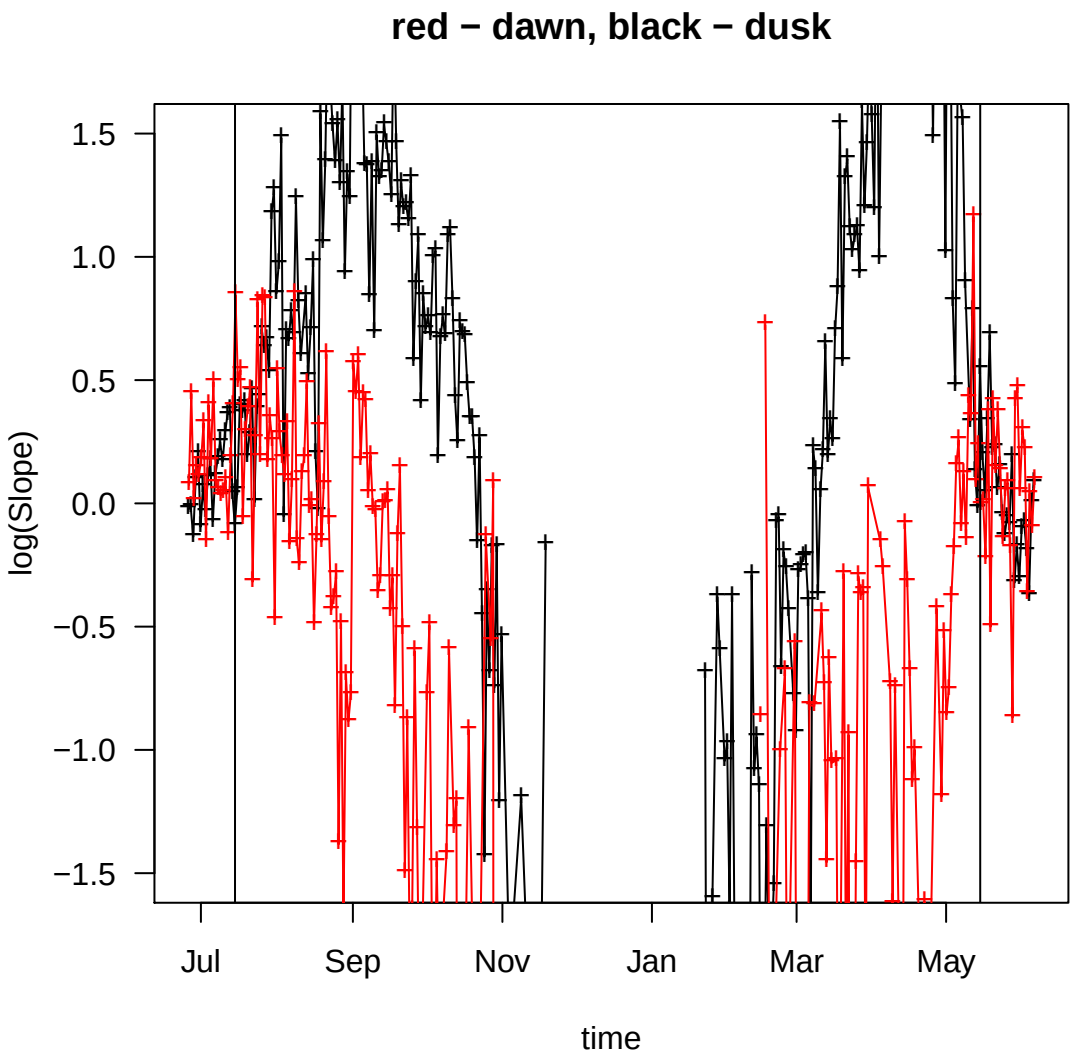


This object contains all the calibration parameters for the tag, and it will be further used for calculation of geographic positions across the track. When there are more than one calibration periods, the parameter `model.ageing` can be set `TRUE` to account for the tag ageing. In this case, the calibration parameters are calculated, based on the assumption that the calibration slope changes linearly with time. Alternatively, one can use one very long calibration period to try estimating ageing of the tag. We would recommend to have at least two months in a known location to be able to do that.

Find calibration periods for a known calibration location

The exact period of a tagged animal's stay in a known location is usually unknown, but it can be derived from the data. For this, calibration slopes for the sunset and sunrise of each day of the tracking period are calculated, based on the assumption that the tag remained in the same known position all the time. The slopes are calculated and plotted with the function `plot_slopes_by_location`.

```
plot_slopes_by_location(Proc.data=FLightR.data, location=c(12.23, 55.98), ylim=c(-1.5, 1.5))

abline(v=as.POSIXct("2015-05-15")) # end of first calibration period
abline(v=as.POSIXct("2014-07-15")) # start of the second calibration period
```



Looking at the plot, we can define the time periods, during which the tag resided in the calibration location (recall, that we assume that the tag remained in this location all the time). Because calibration slopes reflect the adequacy of the light level measured by the device, they vary little, in time and between sunsets and sunrises, as long as the tagged animal stays in the calibration location, but become apparently diverse, when it moves away from it. Both patterns are clearly distinguishable at the plot.

Play with `abline()` to find the proper boundaries for the calibration.

With the current tag it is a bit complicated to see the end of the calibration period and also the first arrival back to the calibration site. To solve the problem we have made two runs - first with the obcvious short calibration period:

```
Calibration.periods<-data.frame(
        calibration.start=as.POSIXct(c(NA)),
        calibration.stop=as.POSIXct(c("2014-07-07")),
        lon=12.23, lat=55.98)
```

We ran the full estimation procedure with based on this short calibration. Looked at bird departure and arrival dates from and to breeding site (with simple `plot_lon_lat(Result)`), used these dates to create new calibration periods and to run the final analysis. If you will do the same, please note, that you should select edges of the calibration period on a safe side - at least 24 hours before the migration.

Find a calibration location for a known calibration period

It may happen that an animal was tagged in the High Arctic under polar day conditions or that it moved far away from the capture site immedialtly after tagging and the roof-top calibration data are not available. Even in such cases it is still possibe to obtain calibration parameters for a resident period at unknown location. FLightR approach to this problem is similar to Hill-Ekstrom calibration Lisovski et al. 2012b implemented in GeoLight Lisovksi et al. 2012a. If bird is assumed to be resident at some period one can try:

```
# ~ 15 min run time
Location<-find.stationary.location(FLightR.data, '2014-09-05', '2014-10-07',
                                   initial.coords=c(25, -10))
```

After 15 minutes the function will return geographic coordinates of the location for which the range of errors in slopes is minimal. User has to provide the initial coordinates, which should be within a few thousand kilometers from the hypothetical real location.

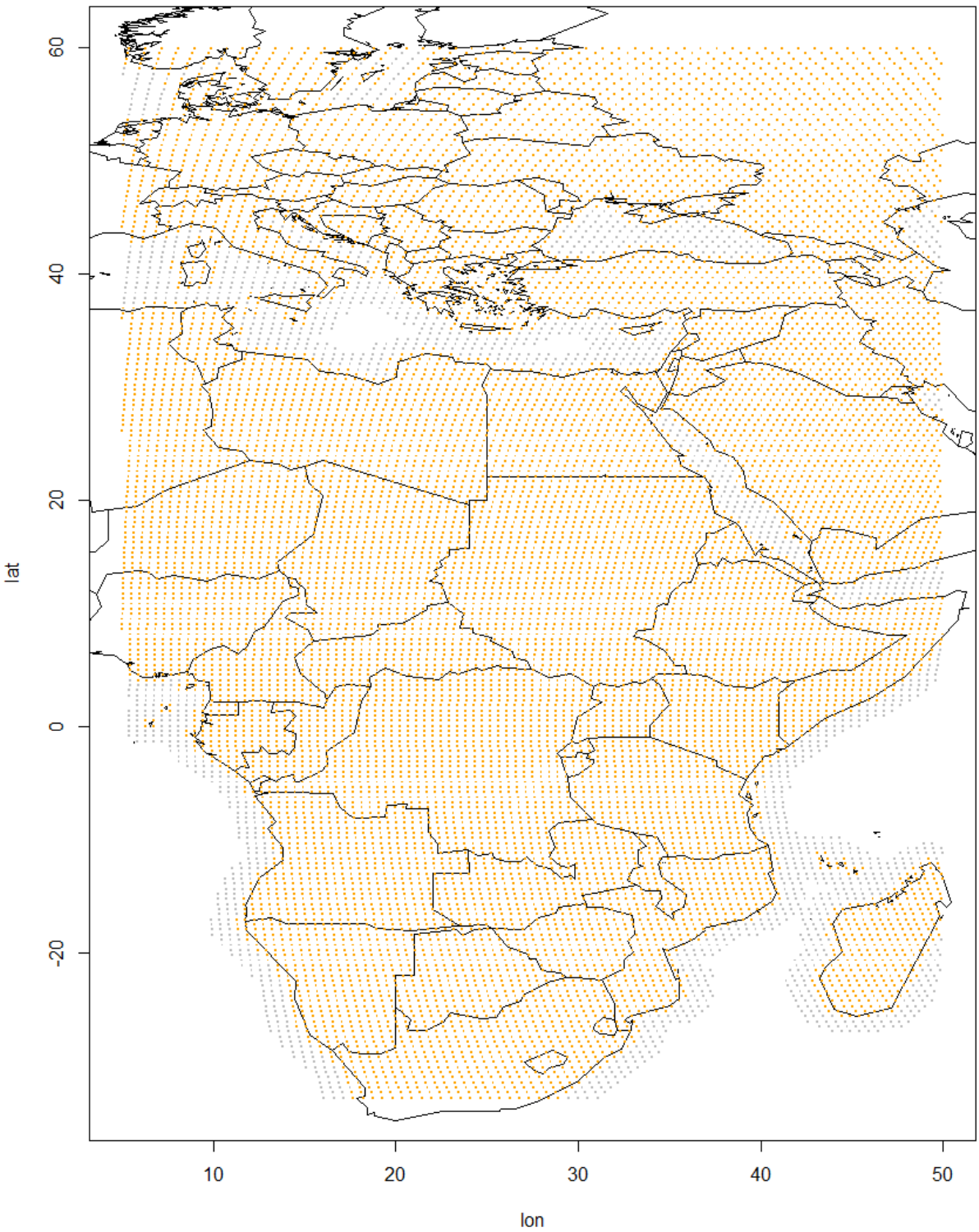
Note that this function is experimental and does not always produce meaningful results . Try it from different initial locations, and check whether it gets to the same result. >/div>

Assign spatial extent

Before running the model you will have to make a scientific guess on where you expect your bird to fly and assign a spatial grid (50 X 50 km) with user-defined boundaries, within which your model will calculate probabilities of presence of the birds. The wider you set the grid boundaries,the longer will take the model to run. If you have set the boundaries too narrow, you will see it in the output map: the defined location points will bounce close to the grid boundaries (as in our example XXXX). In this case you will have to extend the grid and re-run the model.
To set up the grid use the function `make.grid` and define the boundaries: `left`, `right`. `bottom` and `top`. It is possible that your tagged animal cannot occur or stay between two subsequent twilights over

particular areas, for example, over open water if it is a landbird or deep inland if it is a marine animal. In this case we recommend to apply additional parameters `distance.from.land.allowed.to.use` and `distance.from.land.allowed.to.stay`. Such restrictions will minimize the area processed and this way facilitate the analysis. Each of the parameters require a vector of two numbers: the minimal and the maximal distances (in km) from shoreline, at which the animal is allowed to occur/stay. The numbers can be negative. For instance, distance.from.land.allowed.to.stay=-30, will not allow your bird to stay further than 30 km inland from the shore. In our example we allow our shrike to fly over the water, but not further than 200 km offshore, and to be statuionary within 50 km from the land.

```
Grid <- make.grid(left=5, bottom=-33, right=50, top=60,
   distance.from.land.allowed.to.use=c(-Inf, 200),
   distance.from.land.allowed.to.stay=c(-Inf, 50))
```



The resulting `Grid` is a matrix with the columns: `lon` (longitude), `lat` (latitude) and `Stay` (probability of stay). Grid cells that presumably cannot be visited by the animal are excluded from the data, while the

locations at which an animal cannot be stationary are given a low probability of stay. The function produces a visual representation of the created grid with orange poins where the bird is allowed to be stationary and grey, where the bird can fly but not stay. Have a look at the figure and make sure it is correct. Using masks can side track model estimation to the local minima, and we recommend to initially run the model without the mask, enable the mask for the second run and visually compare the results to see if the model has converged to similar tracks.

Prepare the model for run

At this stage, all the objects, created at earlier steps: the light data with the detected twilight events (`FLightR.data`), the spatial parameters (`Grid`), geographic coordinates of the initial location, where the tracking has started (`start`), and the calibration parameters (`Calibration`), are to be incorporated in one complex object , which will be used in the main run. Use the function `make.prerun.object` for the compilation. Using this function you can also change the priors for the movement model. For example we set `M.mean` parameter to 750 km, because we know that shrikes migrate fast and also because we have explored the results and figured out that average migration distance was actually 750 km.

```
# ~ 15 min run time
 all.in <- make.prerun.object(FLightR.data, Grid, start=c(lon.calib, lat.calib),
                              Calibration=Calibration, M.mean=750)
```

This is the first object we want to save. It contains the twiligth file, the calibration information as well as the likelihood surfaces for each twilight.

```
save(all.in, file = paste0(wd, "/Results/", Species, "/", ID, "_FlightRCalib.RData"))
```

Particle filter run

Once you have got the `run.particle.filter` running, expect your computer to be busy for about 45 minutes. During this time it will generate the model output (we will call it "Results") containing: * a table of positions at each twilight (`$Results$Quantiles`), * statistics of the positions (mean, median values and credible intervals), * a table of parameters of the movement model (`$Results$Movement.results`), * posterior distribution of the parameters at every twilight (`$Results$Points.rle`) and * at every transition between twilights (`$Results$Transitions.rle`).

Within `run.particle.filter`, the following parameters can be defined: * `nParticles` - number of particles (we recommend to use 1e4 for test and 1e6 for the analysis); * `threads` - amount of parallel threads to use for the run default is -1 that means all available except one; * `known.last` - TRUE if you know that in the end of the logging period tag occurred in a known place (FALSE is the default option); * `check.outliers` – FALSE by default. Set it to TRUE if you wish on-the-fly outliers detection, highly recommended if the results have strong outliers. * `b` - the maximum distance allowed to be travelled between two subsequent twilights.

```
nParticles=1e6
Result<-run.particle.filter(all.in, threads=-1,
            nParticles=nParticles, known.last=TRUE,
                precision.sd=25, check.outliers=F,
            b=1700)
```

Again we want to save the Result.

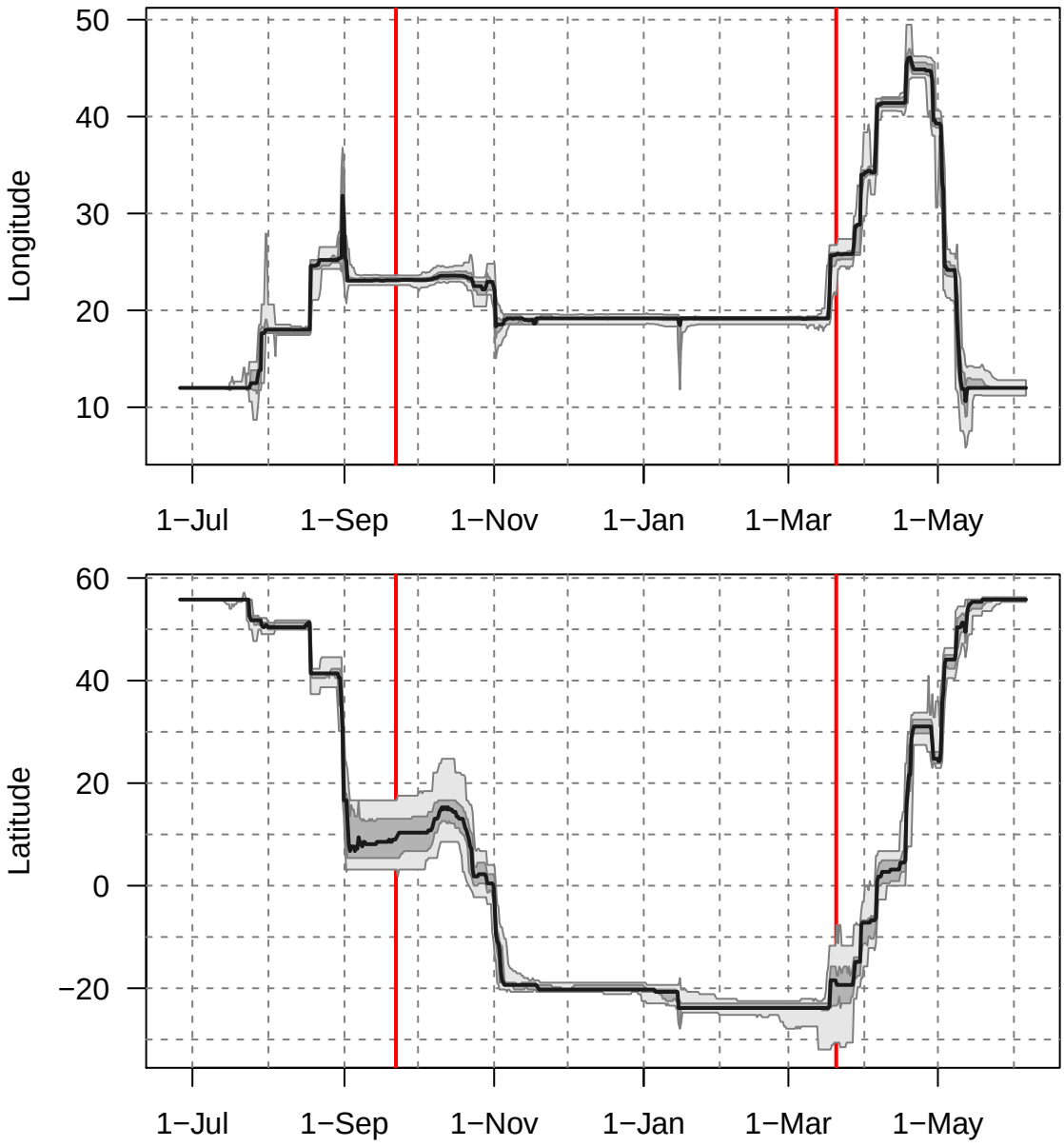
```
save(Result, file = paste0(wd, "/Results/", Species, "/", ID, "_FLightRResult.RData"))
```

Exploration of results

First look - plot longitude and latitude

After the results have been saved the first thing to do is to plot change of longitude and latitude over time. This is a very important graph.

```
plot_lon_lat(Result)
```



With this graph, please, carefully check whether: 1. Your track does not approach boundaries of the spatial grid you defined (you ave to keep those in mind assessing the graph). If this is the case, change spatial extend and rerun the analysis. 2. There are no sudden jumps that have no uncertainty measurements around them. These are likely outliers that should be excluded beforehands. 3. Presented by the model estiamates of time-periods spent by the bird at the calibration site match calibration periods assigned before the model run. If they do not, correct the calibration periods, redo the calibration and rerun the model. 4. There are no S-shape patterns in estimated latitude around equinox. Such patterns point at either incorrect calibration or very poor data. Try to improve your calibration periods or concider taking calibration data from another tag taken from the same species.

Summary

Repeatability of output

Note that however many times you will re-run the model with the same input parameters, you will never get the same output. Please, ***do not panic***, as stochasticity of probabilities is an inherent feature of Bayesian analysis. Also, if you look closer at your results, you will notice that the outputs vary within each others' standard errors.

Defining stopovers

FLightR is honest defining stopovers, which may seem a disappointing feature, but it makes us thinking. In general, geolocation analysis is all about how likely the bird was present in specific location at specific moment and how likely it was moving between two subsequent locations. It is not a problem when you work with long-distance migrants, such as bar-tailed godwit Rakhimberdiev et al. 2016, since it has very apparent stopover phases between apparent migration-flight bouts. However, many bird species migrate in little hops and literary stop "at every bush". Here you can get rather messy results, due to the overlapping probability distributions. In any case, we recommend to play with the stopover probabilities by tweaking the cut-off probability parameter of 1stationary.migration.summary‘. Recall that for each location and time moment the model generates a probability that the bird was moving to the next location. The cut-off probability is the minimal threshold probability of movement, at which we assume that the bird was actually moving. Depending on the assigned cut-off probability parameter, the model will consider the bird more or less likely to be stationary. In birds migrating in long leaps with few stops, defined stopovers are usually rather robust. However, in hopping migrants, probabilities of stopovers may vary, depending on the chosen cut-off probability parameter, which is a bad news if describing stopovers is the prior goal of your study. But, frankly speaking, what is stopover and does it exist in migrants moving in short hops? Presenting your results, you may arbitrarily choose the most adequate (repeatable) output and mention the cut-off probability, with which it was obtained. However, if you want to describe stopovers explicitly, you might want to present outputs obtained with several cut-off probability parameters and speculate about the variation. Most likely, some of your stopovers will be defined repeatedly with a range of cut-off parameters being assigned, and some will be less robust. Our shrike had rather distinct migration/stopover pattern. However, we get a slightly more stopovers with cut-off probabilities of 0.1 and 0.2.

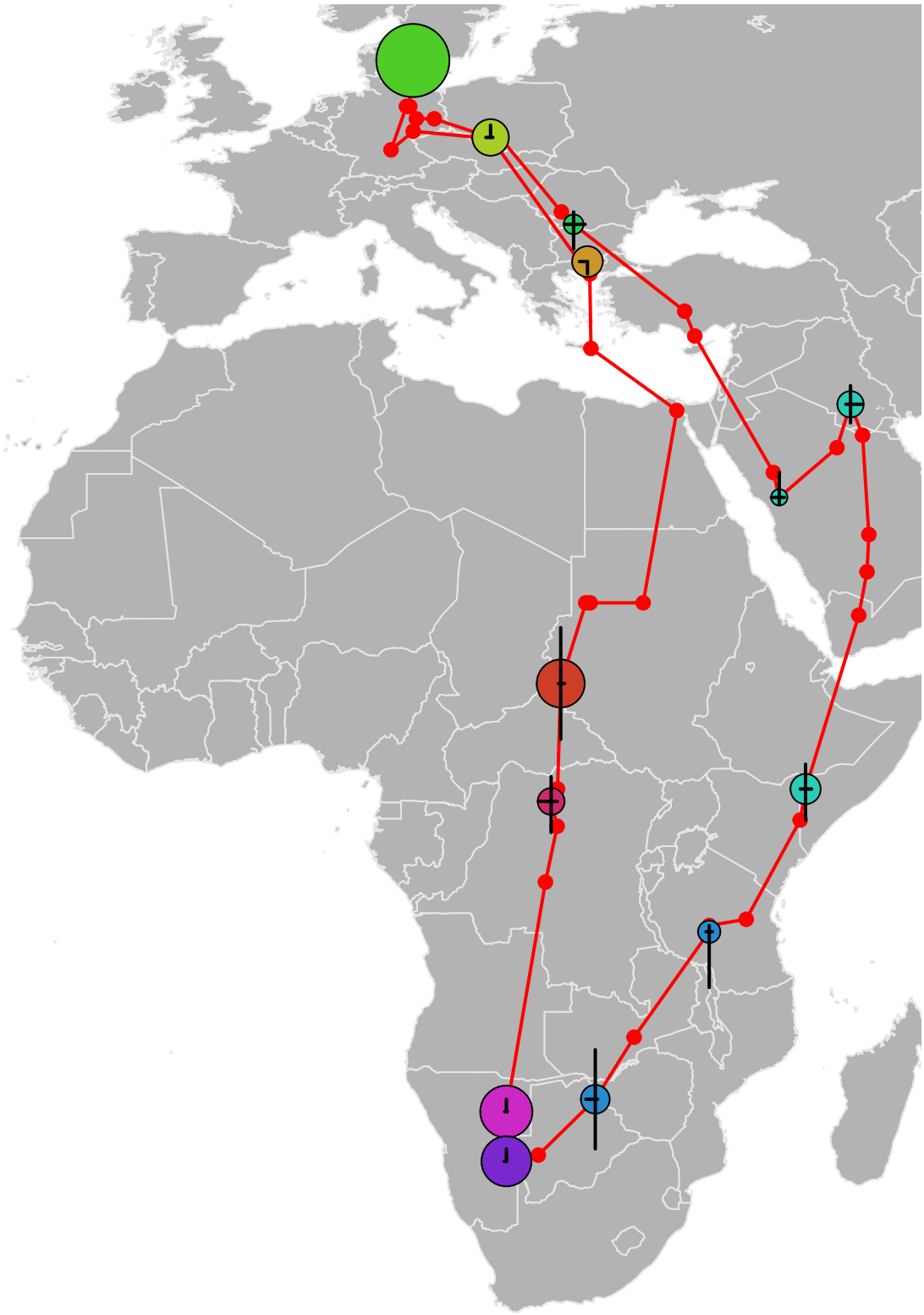
To distinguish between the phases of movement and stationarity use the function `stationary.migration.summary` helps . The process usually takes around 15 minutes of computer time.

```
Summary<-stationary.migration.summary(Result, prob.cutoff = 0.2)
```

To visualise results of this function we will plot them on a map.

```
# Now we want to plot the detected stationary periods on a map
Summary$Stationary.periods$stopover_duration<-as.numeric(difftime(Summary$Stationary.periods$Departure.
# Now I want to select the periods which were >=2 days and
Main_stopovers<-Summary$Stationary.periods[is.na(Summary$Stationary.periods$stopover_duration) | Summary
# delete breeding season
Main_stopovers<-Main_stopovers[-which(is.na(Main_stopovers$stopover_duration)),]

Coords2plot<-cbind(Result$Results$Quantiles$Medianlat, Result$Results$Quantiles$Medianlon)

for (i in 1:nrow(Summary$Potential_stat_periods)) {
  Coords2plot[Summary$Potential_stat_periods[i,1]:
    Summary$Potential_stat_periods[i,2],1] =
```

```
    Summary$Stationary.periods$Medianlat[i]

  Coords2plot[Summary$Potential_stat_periods[i,1]:
    Summary$Potential_stat_periods[i,2],2] =
    Summary$Stationary.periods$Medianlon[i]
}
Coords2plot<-Coords2plot[!duplicated(Coords2plot),]

#pdf('FLightR_shrike_migration_with_stopovers.pdf', width=6, height=9)
par(mar=c(0,0,0,0))
map('worldHires', ylim=c(-35, 60), xlim=c(-20, 50), col=grey(0.7),
    fill=TRUE, border=grey(0.9), mar=rep(0.5, 4), myborder=0)

lines(Coords2plot[,1]~Coords2plot[,2], col='red', lwd=2)
points(Coords2plot[,1]~Coords2plot[,2], ,lwd=2, col='red', pch=19)

# Here we assign the colours to represent time of the year
Seasonal_palette<-grDevices::colorRampPalette(grDevices::hsv(1-((1:365)+(365/4))%%365/365,
                                              s=0.8, v=0.8), space="Lab")
Seasonal_colors<-Seasonal_palette(12)

Main_stopovers$Main_month<-as.numeric(format(Main_stopovers$Arrival.Q.50+
                                             Main_stopovers$stopover_duration/2, format='%m'))

points(Main_stopovers$Medianlat~Main_stopovers$Medianlon, pch=21,
       cex=log(as.numeric(Main_stopovers$stopover_duration)),
       bg=Seasonal_colors[Main_stopovers$Main_month])

# Now, for each of these points we plot the uncertainties
# Horizontal
segments(y0=Main_stopovers$Medianlat, x0=Main_stopovers$FstQu.lon,
         x1=Main_stopovers$TrdQu.lon, lwd=2)
# Vertical
segments(x0=Main_stopovers$Medianlon, y0=Main_stopovers$FstQu.lat,
         y1=Main_stopovers$TrdQu.lat, lwd=2)

# and we also need to add breeding site here:
points(x=12.23, y=55.98, cex=6, pch=21 , bg=Seasonal_colors[6])
# dev.off()
```



Getting specific output

The function `find.times.distribution` derives the time at which an animal arrived or departed from the area and provides the measure of its uncertainty. First, select grid points of interest. For example in the current data we are interested in the date when our bird left from the breding grounds and when it was back. We will make a boundary at 55.5&deg latitude:

```
Index<-which(Result$Spatial$Grid[,2]>55)
```

Estimate probabilities of occurrence within the area at each twilight:

```
Arrivals.breeding<-find.times.distribution(Result,Index)
print(Arrivals.breeding)
```

```
                Q.025               Q.25               Q.50
1 2014-07-22 22:32:04 2014-07-23 17:27:14 2014-07-24 03:20:51
2 2015-05-12 01:22:59 2015-05-13 00:21:03 2015-05-14 13:54:11
                Q.75               Q.975
1 2014-07-24 14:32:33 2014-07-25 17:36:17
2 2015-05-20 13:58:56 2015-05-26 06:00:36
```

Visualisation of the results

Plot a simple map

Plot a map with the most probable positions, i.e. combinations of the most probable latitude and longitude for each twilight:

```
try(map.FLightR.ggmap(Result, zoom=3, save = FALSE))
```

We use `try` here to ease automated rendering. Just exclude it for your own runs. We also recommend a very nice feature of this function: `zoom='auto'` that will try finding optimal zoom for your output.

Plot utilization distribution

Plot space utilisation distribution for the wintering range:

```
try(tmp <- plot_util_distr(Result,
    dates=data.frame(as.POSIXct('2014-12-01'), as.POSIXct('2015-01-31')),
    add.scale.bar=TRUE, percentiles=0.5, zoom=7, save = FALSE))
```

we use `try` here to ease automated rendering. Just exclude it for your own runs. We also recommend a very nice feature of this function: `zoom='auto'` that will try finding optimal zoom for your output. >/div>

Chapter 9

Data repositories

...

Chapter 10

Your contribution

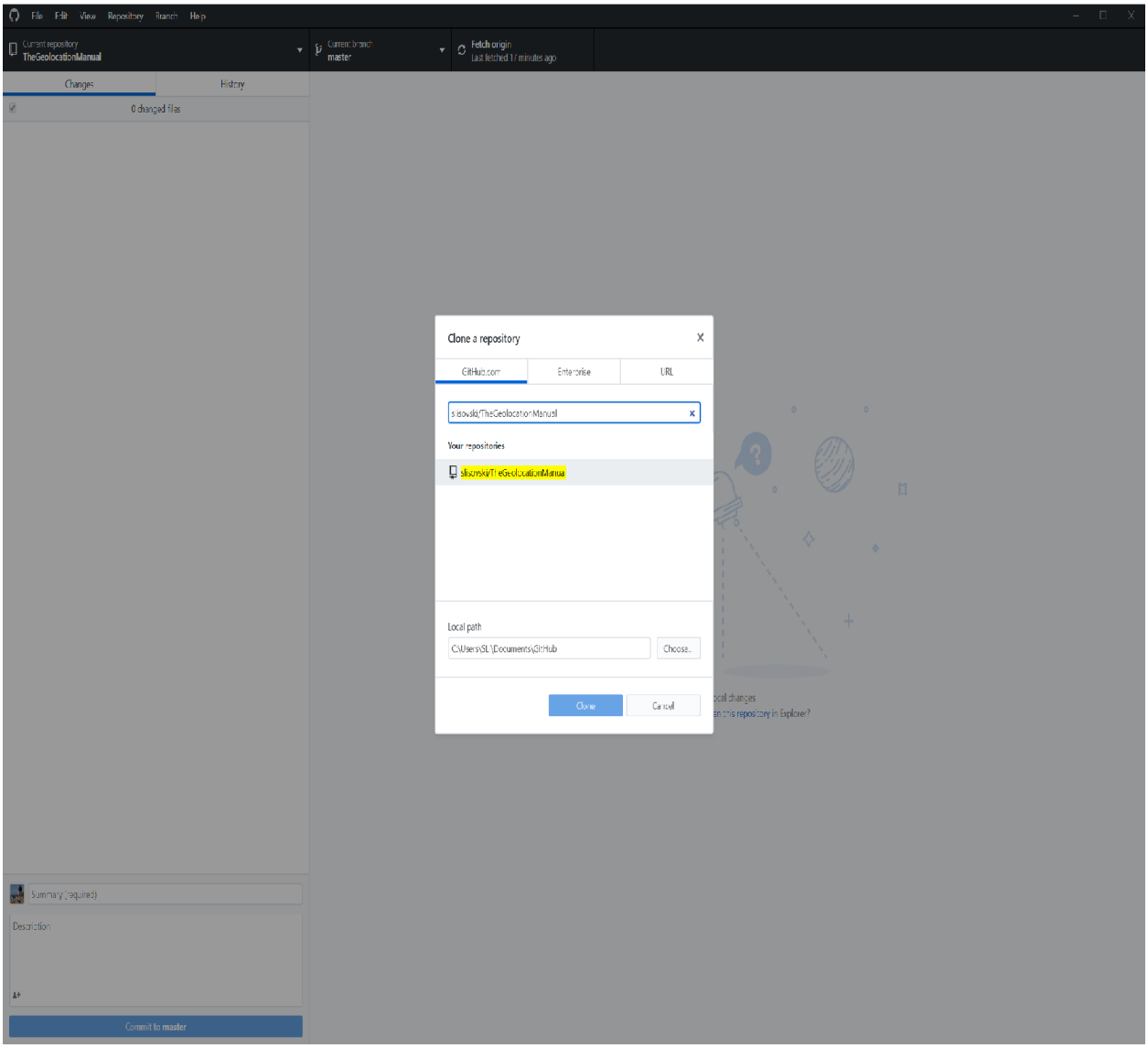
We do consider this document to be a community endeavour for which we have given a starting point. We apprechiate any contribution in terms of thoughs on the current version, changes and additions. Arguably, your expereince with geolocation is valuable for the community and by adding it to this manual it can help users significantly. Fortunately, the GitHub and RStudio environemnts make it (relatively) easy to give access to the source code of this manual and to *push* your changes directly into the online version

It may seem combersome at first but it is worth the effort to get familiar with GitHub and make use of the massive potential of version control and developing as well as publishing code, such as this manual.

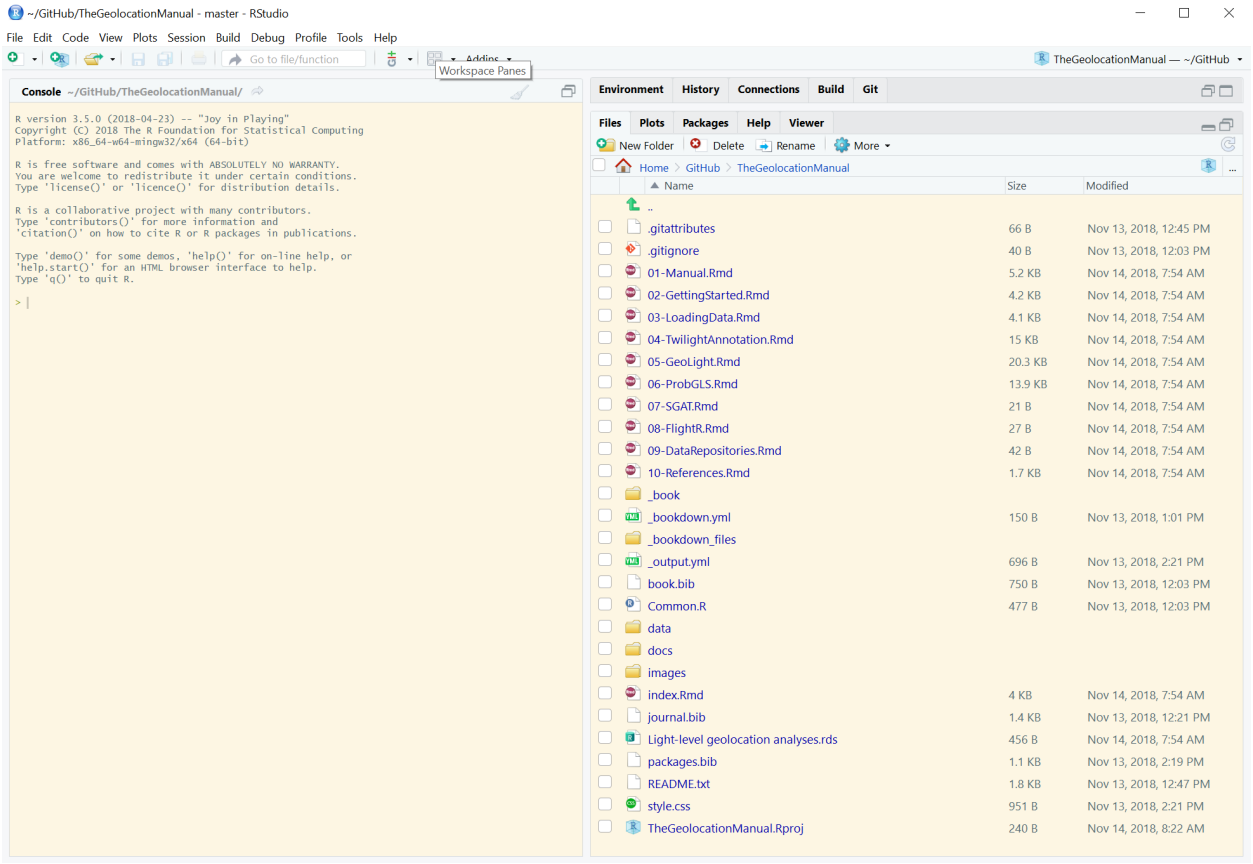
If you have not already created an account do so (www.github.com). And while RStudio has implemented a Git environment that allows direct communication with GitHub, we recommend to download the GitHub Desktop application if you are new to this world of version control. The application can be downloaded from: https://desktop.github.com/.

Once you have installed and loged into your account you can *clone* repositories that are already on GitHub or *create* new repositories via the file menu.

To get started with *The Manual*, choose *clone a repository*, define the local path where the data should be saved and type in *slisovski/TheGeolocationManual* and press *Clone*:

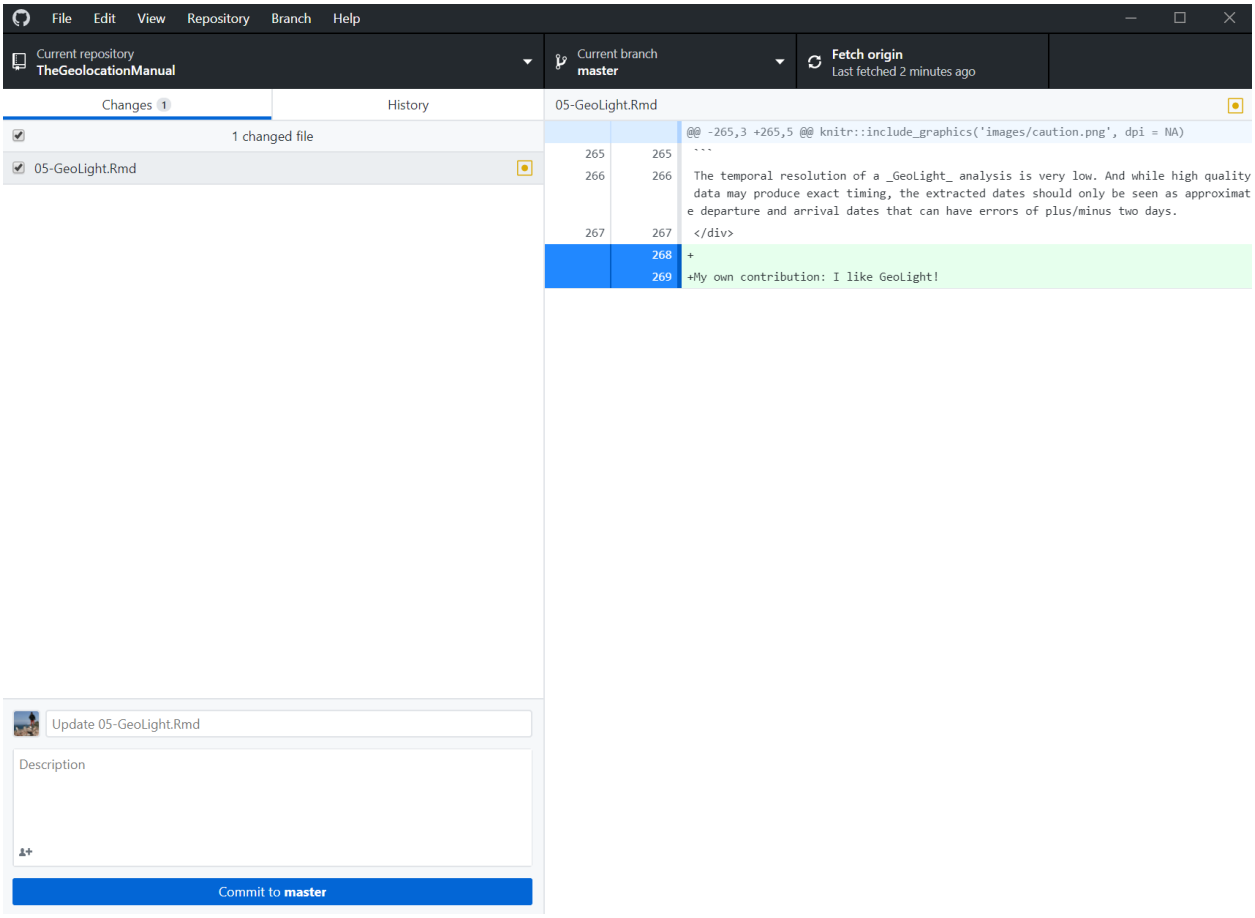


Navigate to the local folder that should now contain a subfolder called *TheGeolocationManual*. In this subfolder you will find a RStudio project file called "TheGeolocationManual.Rproj". Open this file with a double click. In RStudio you are now working within this folder and in the *Files* window, you will find all the files that are part of *The Manual*.



The important files are the **.Rmd** files that contain all the code and text of *The Manual*. If you intent to edit something in the e.g. GeoLight section, open the *05-GeoLight.Rmd* file and start editing. Make sure to save changes once you are done.

Back in the GitHub Application you can see all changes you have made that are different to the version your *fetched* from GitHub:

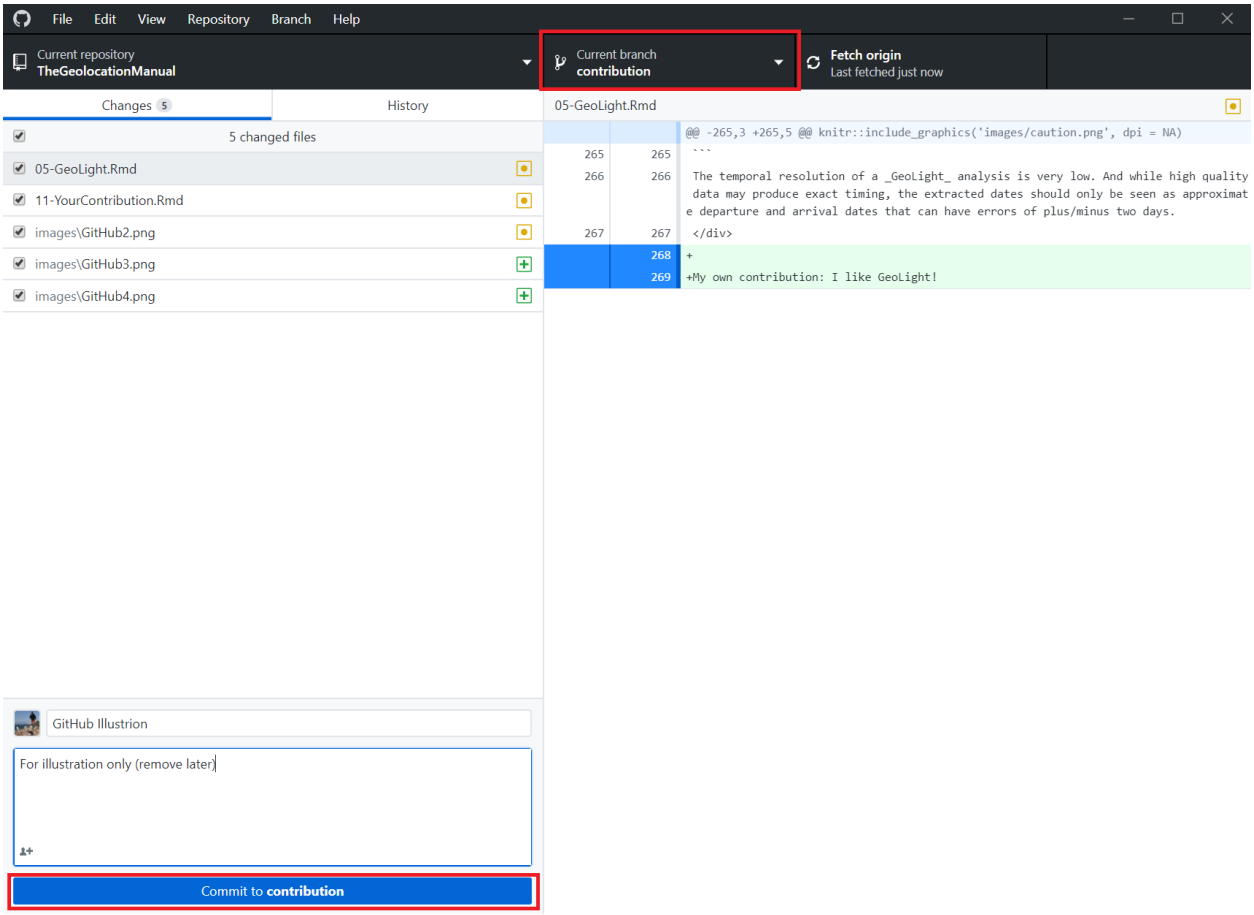


Make sure to *fetch* (pressing *Fetch origin*) before you start making any edits in RStudio. This gives you the newest version and avoids conflicts with edits from other users!

It is important(required) to provide a comment for your commit (your changes). Please use somthing that makes sense and allows others to have a rough idea what your changes entail. Feel free to provide more information in the *Decription* such as "I have removed an obvious bug in line xx" or, "I have added text in the calibration section, please review!". Next, press commit to **contribution**.

GitHub allows to have different *branches*. This is a creat invention that is immensely usefull for such endavours; we can keep the current online version of *The Manual* untouched but make edits etc. to the code. Once we are happy with new edits and maybe with a complete new version we can *merge* the branches and update the *master* branch and thereby the online version. This makes sure that the online version is always running and that changes can be reviewed and revised before going online.

To make sure that your changes will be submitted to the **contribution** branch select the correct branch in the menu:



The press *Commit to* ***contribution*** and if your want this commit to go online press *Push origin*.

Commit only logges the changes into the version control file on your computer, you have to *Push origin* to make sure that it is submitted to GitHub and visible for all other users!

References

- Ekstrom, P. (2004). An advance in geolocation by light. Memoirs of the National Insitute of Polar Research, Special Issue, 58, 210–226.
- Ekstrom, P. (2007). Error measures for template-fit geolocation based on light. Deep Sea Research Part II: Topical Studies in Oceanography, 54, 392–403.
- Lisovski, S., Hahn, S. (2012a). GeoLight - processing and analysing light-based geolocator data in R. Methods in Ecology and Evolution, 3, 1055–1059.
- Lisovski, S., Hewson, C.M., Klaassen, R.H.G., Korner-Nievergelt, F., Kristensen, M.W. & Hahn, S. (2012b). Geolocation by light: accuracy and precision affected by environmental factors. Methods in Ecology and Evolution, 3, 603–612.
- Pedersen L., et. al.
- Rakhimberdiev, E., Winkler, D.W., Bridge, E., Seavy, N.E., Sheldon, D., Piersma, T. & Saveliev, A. (2015). A hidden Markov model for reconstructing animal paths from solar geolocation loggers using templates for light intensity. Movement Ecology, 3, 25.
- Rakhimberdiev, E., Senner, N. R., Verhoeven, M. A., Winkler, D. W., Bouten, W. and Piersma T. (2016) Comparing inferences of solar geolocation data against high-precision GPS data: annual movements of a double-tagged Black-Tailed Godwit. Journal of Avian Biology 47: 589-596.
- Rakhimberdiev, E., Saveliev, A., Piersma, T., & Karagicheva, J. (2017). FLightR: An R package for reconstructing animal paths from solar geolocation loggers. Methods in Ecology and Evolution, 8(11), 1482-1487.