Light-level geolocation analyses

Last edited on 2018-11-14

Contents

Preface **5**

Acknowledgements 6

License 6

1 Structure of the manual **7**

The datasets 7

Reproducing the analyses 8

2 Getting started **9**

3 Loading data **11**

4 Twilight Annotation **13**

Cleaning/Filtering twilight times 21

5 GeoLight **23**

Getting started 23

Calibration 25

Location estimation 28

Hill-Ekstrom calibration 30

Movement analysis 35

6 probGLS **43**

Getting started 43

Load additional data 46

Download remote sensed environmental data 52

Twilight error (Calibration) 52

Run the iterative algorithm 53

Plot results 54

7 SGAT **57**

8 FLightR **59**

9 Data repositories **61**

10 Your contribution **63**

References **69**

Preface

Note: The Manual is currently under development and content may not show up (ask Simeon if you need immediate access)!

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.

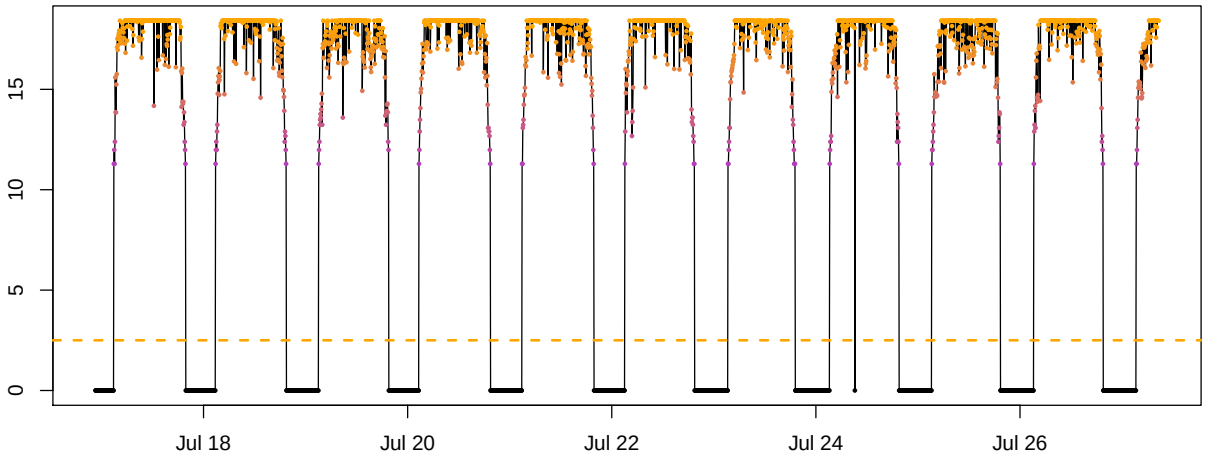
All tools discussed in this manual require as one of their inputs a data frame containing the times of sunrise and sunset (henceforth twilight events) for the duration of the study period. The twilight events are estimated based on a light-level threshold, which is the light value that separates day 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.

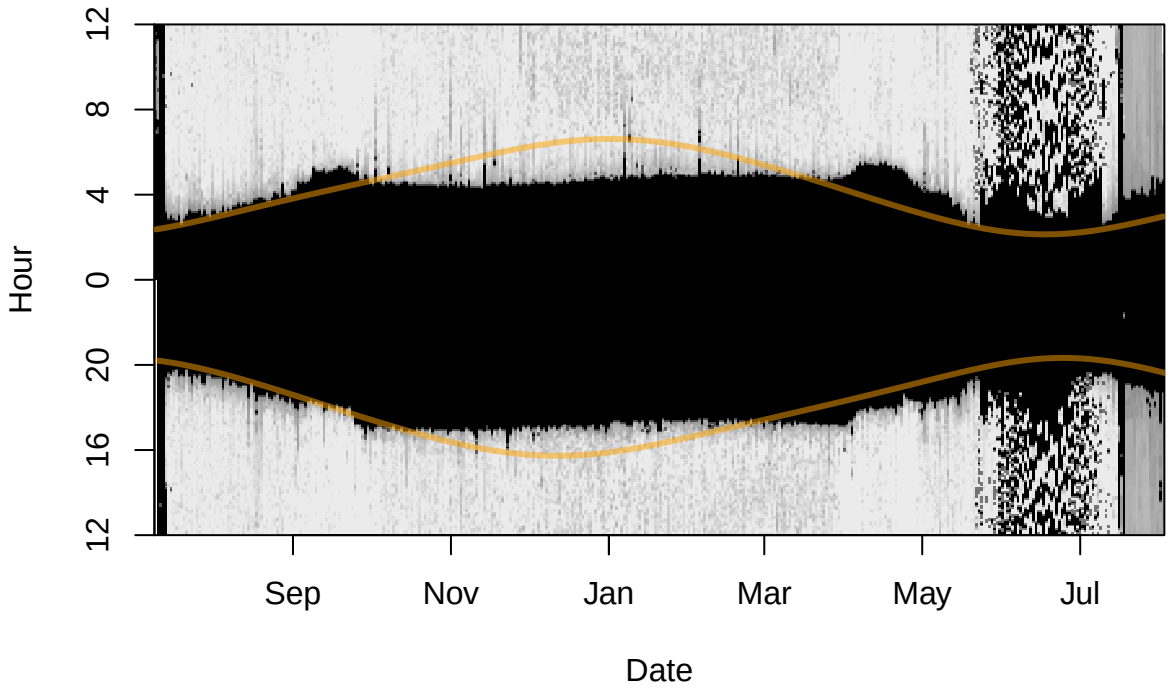
It is a good idea to plot (parts) of the dataset and see how the threshold fits into the light recordings:

```
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
```

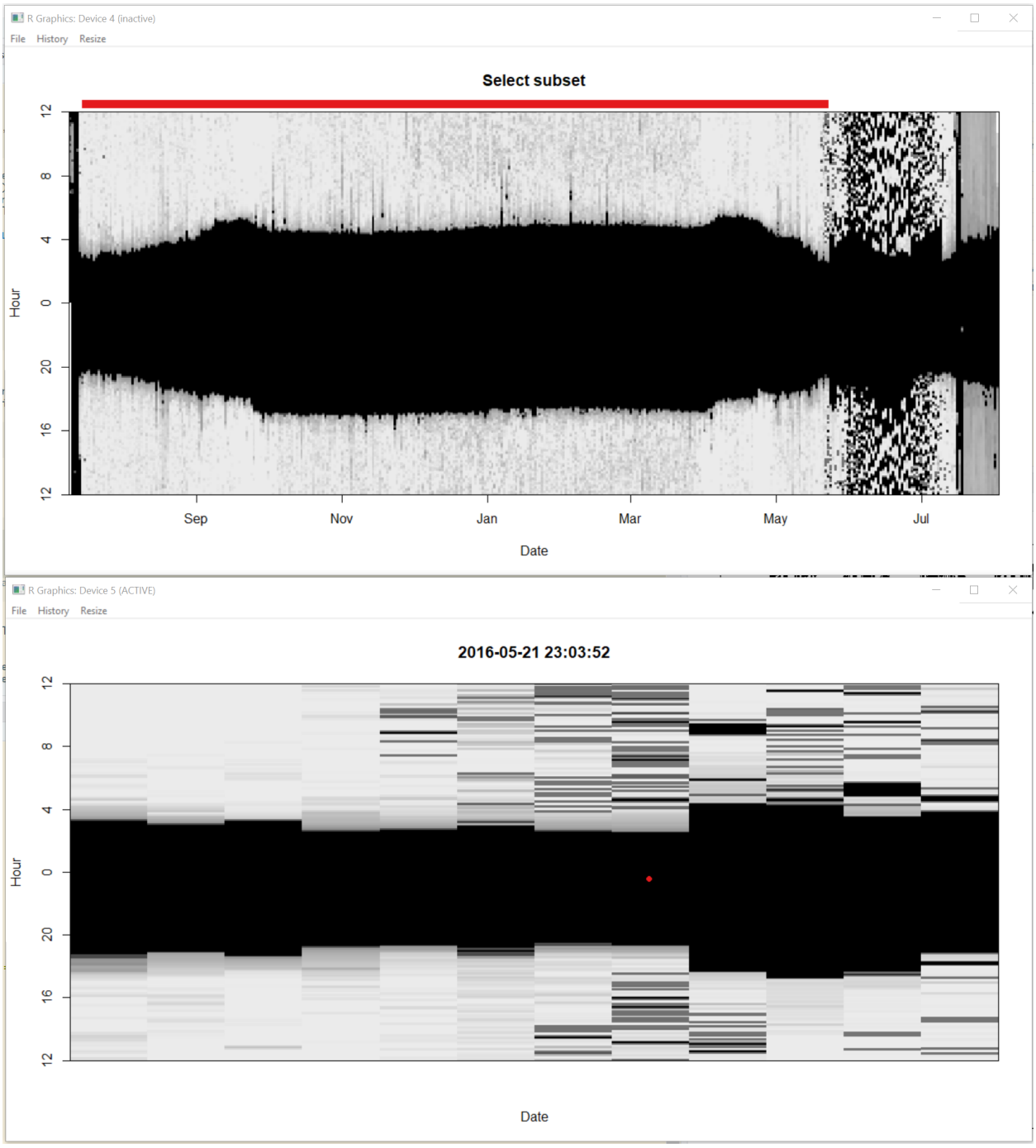
Below, we explain the major functionalities.

When you run,

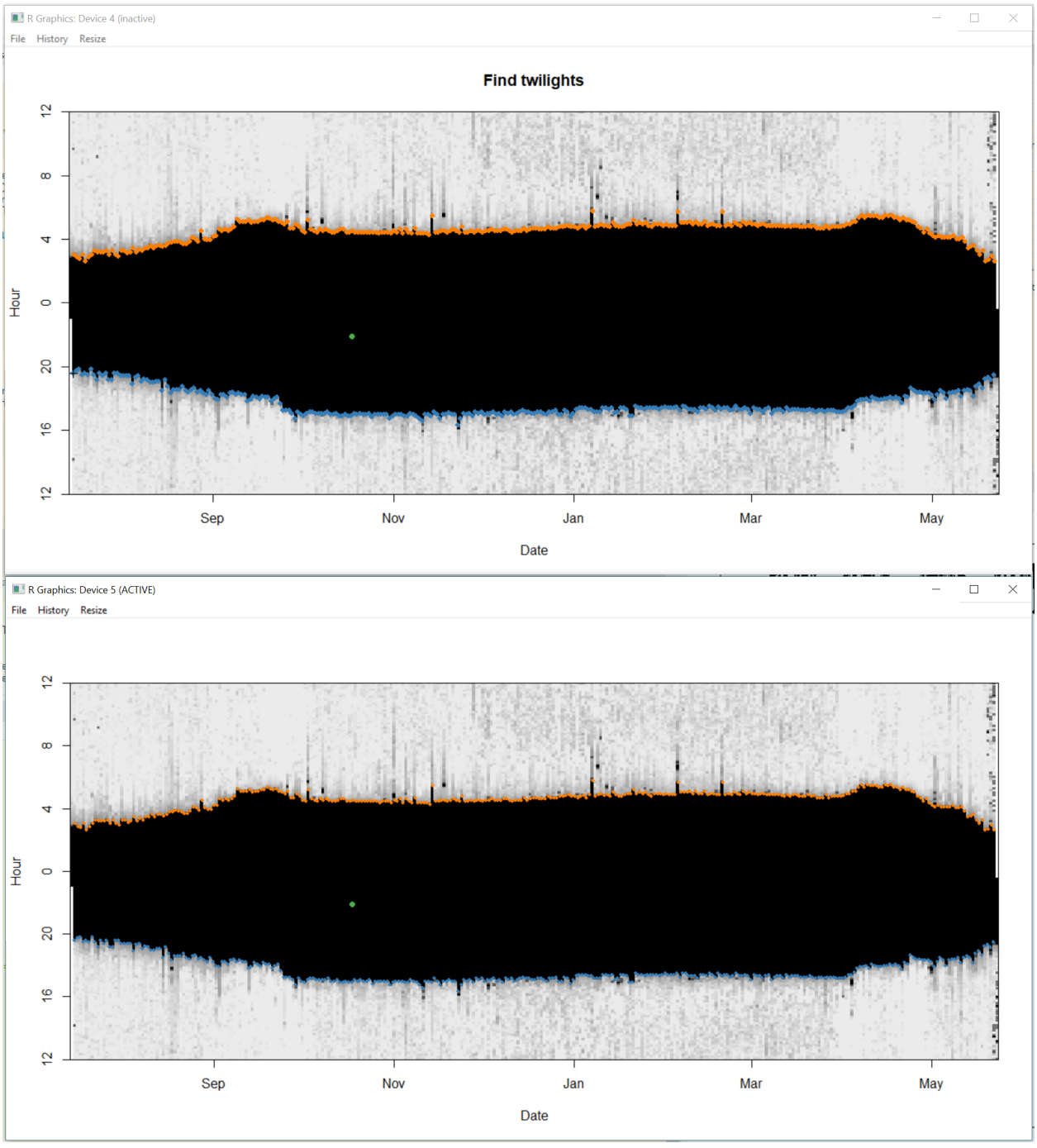
```
twl <- preprocessLight(raw,
  threshold = threshold,
  offset = offset,
  lmax = 20,          # max. light valu
  gr.Device = "x11") # MacOS version (and windows)
```

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.



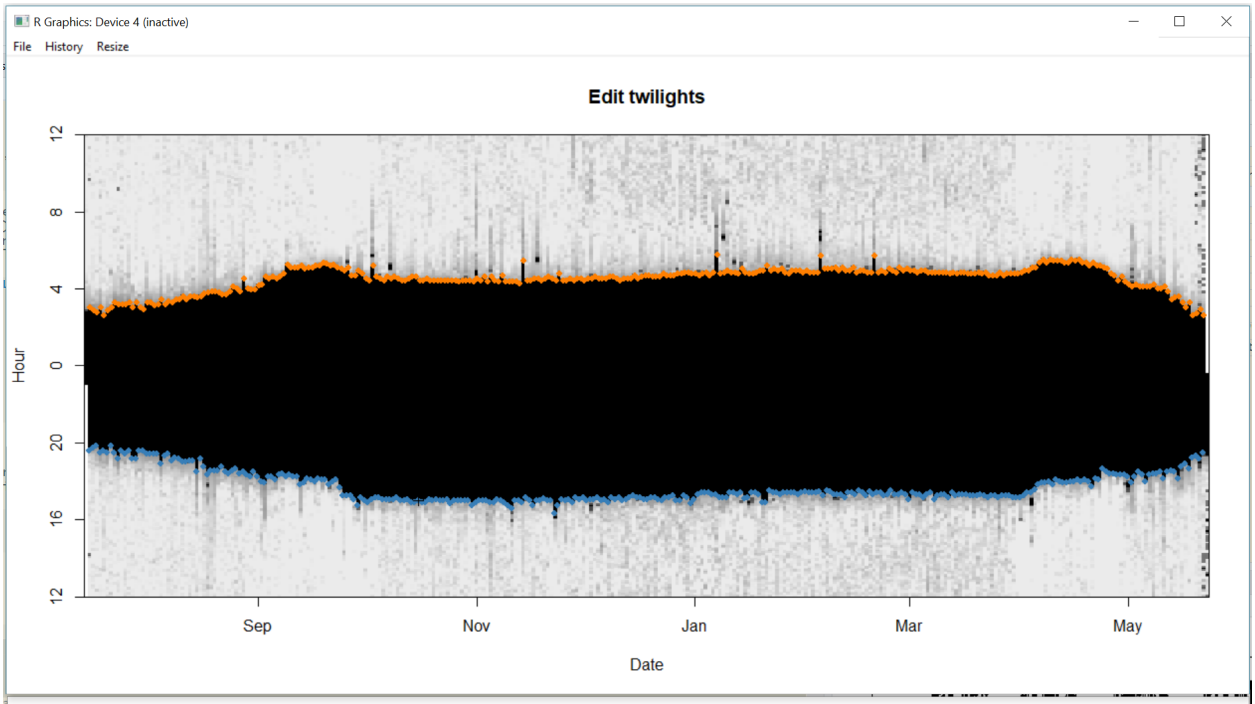
Step 2. click on the window entitled "Find twilights" and the second window will zoom in. All you need to do here is click in the dark part (in the zoomed in image i.e. the one not entitled "Find twilights") of the image and this will identify all the sunrises (orange) and sunsets (blue) based on the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

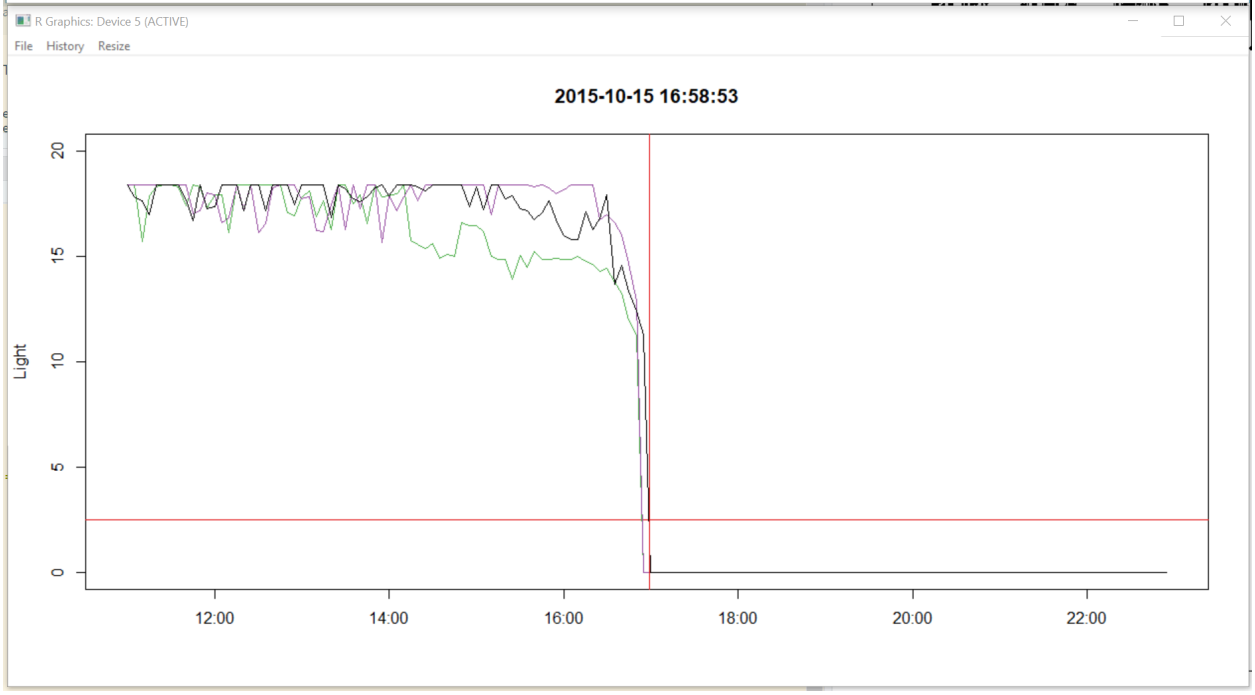


Step 3. This step is for adding or deleting points. If there are no missing data points, you can skip this step by pressing "a" on the keyboard. However, if you do want to add a point, you can click on the "Insert twilights" window to select a region of "the blob" that the second unintitled window will 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
```

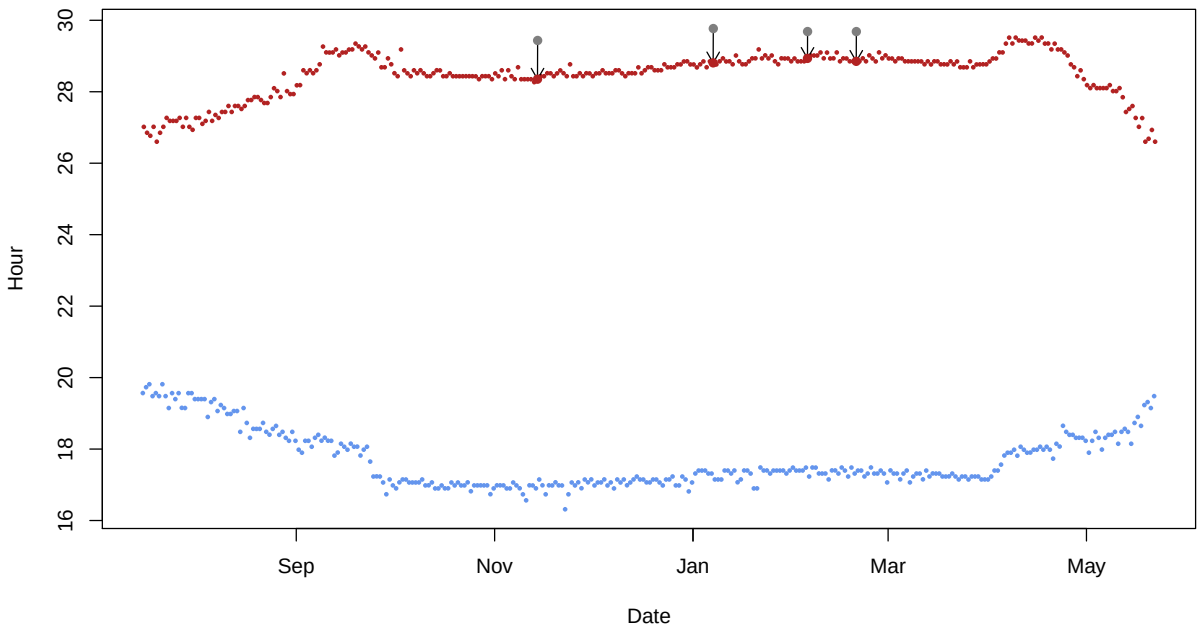
Cleaning/Filtering twilight times

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

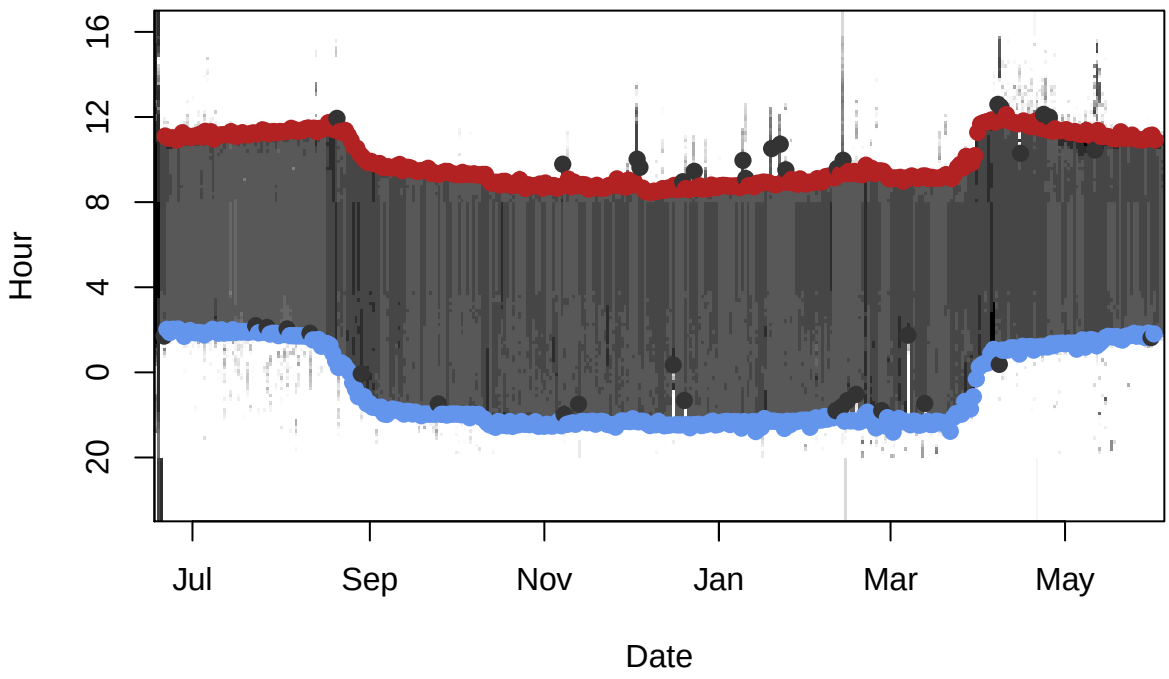
To illustrate the *GeoLight* analysis, we use the Purple martin dataset.

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)
```

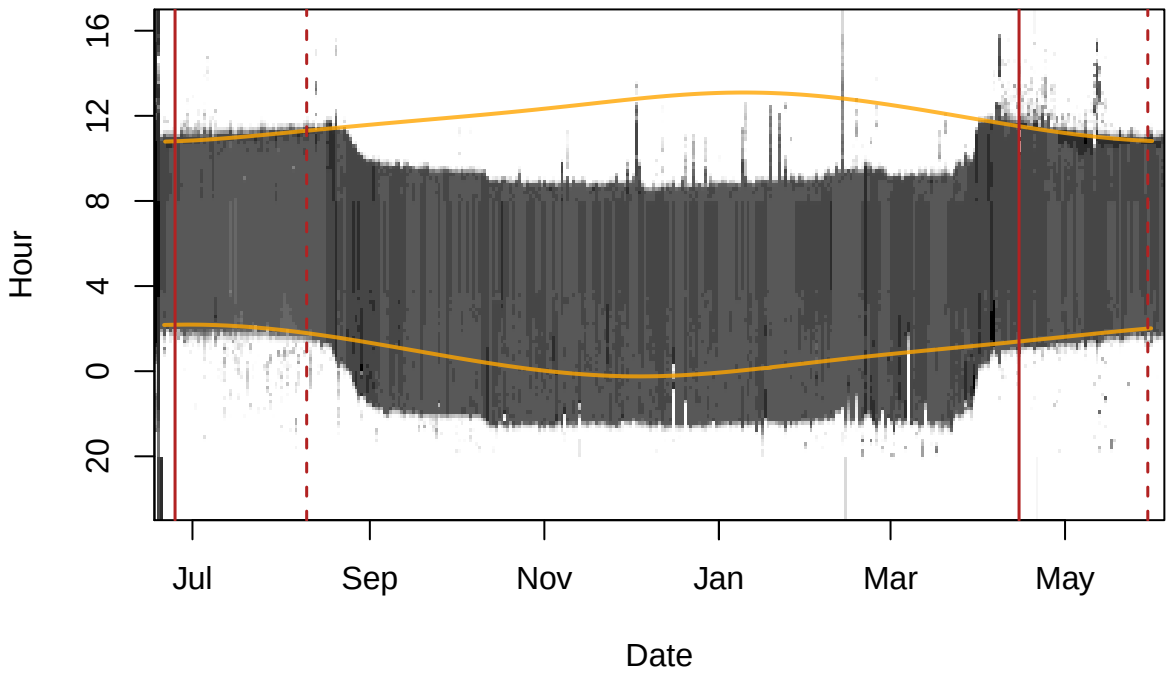
```
               tFirst             tSecond type
1 2011-06-21 11:06:22 2011-06-22 02:01:18    1
2 2011-06-22 02:01:18 2011-06-22 11:00:25    2
3 2011-06-22 11:00:25 2011-06-23 01:52:21    1
4 2011-06-23 01:52:21 2011-06-23 11:01:08    2
5 2011-06-23 11:01:08 2011-06-24 02:01:44    1
6 2011-06-24 02:01:44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



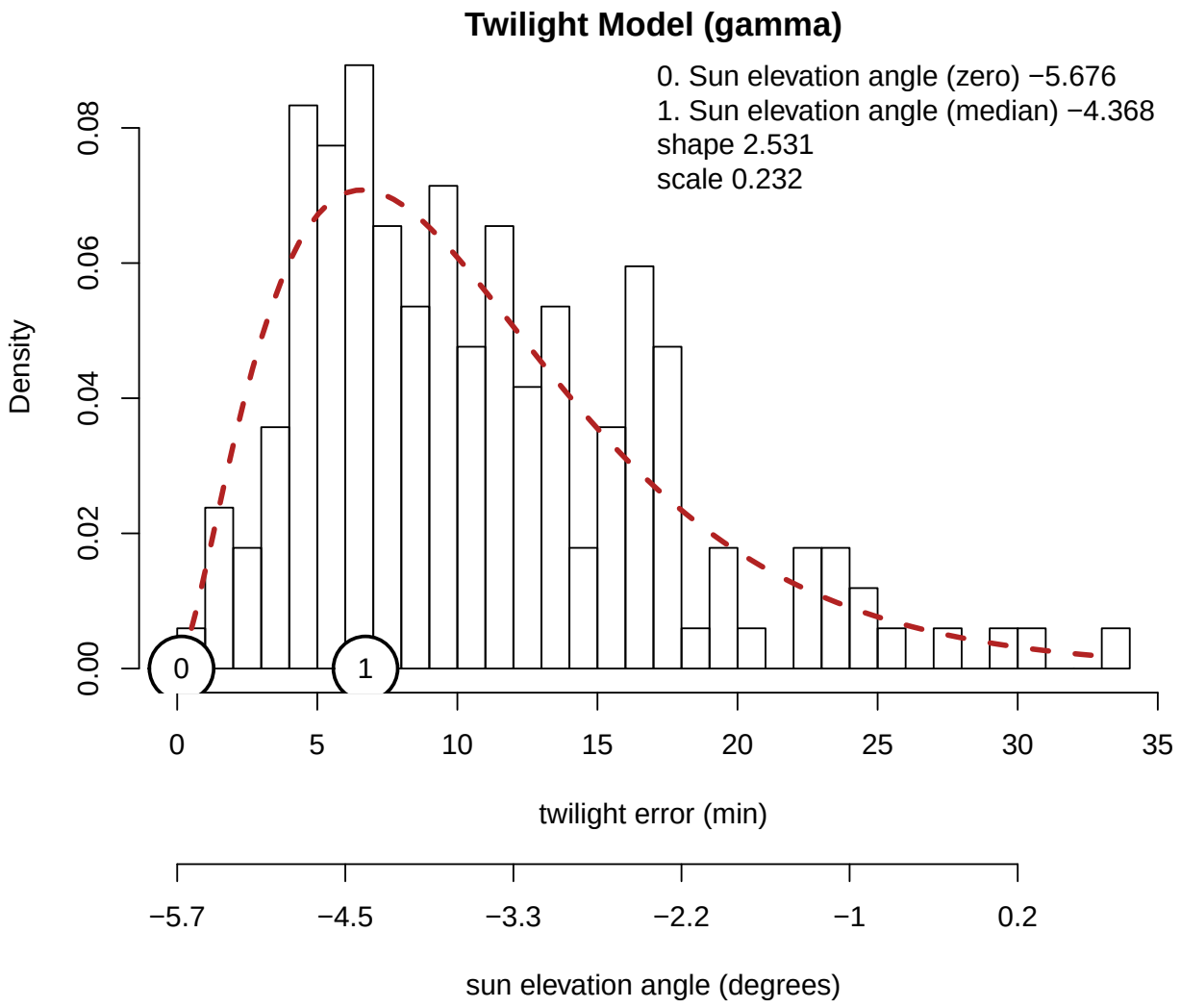
We can now subset the twilight table `twl.gl`.

```
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.

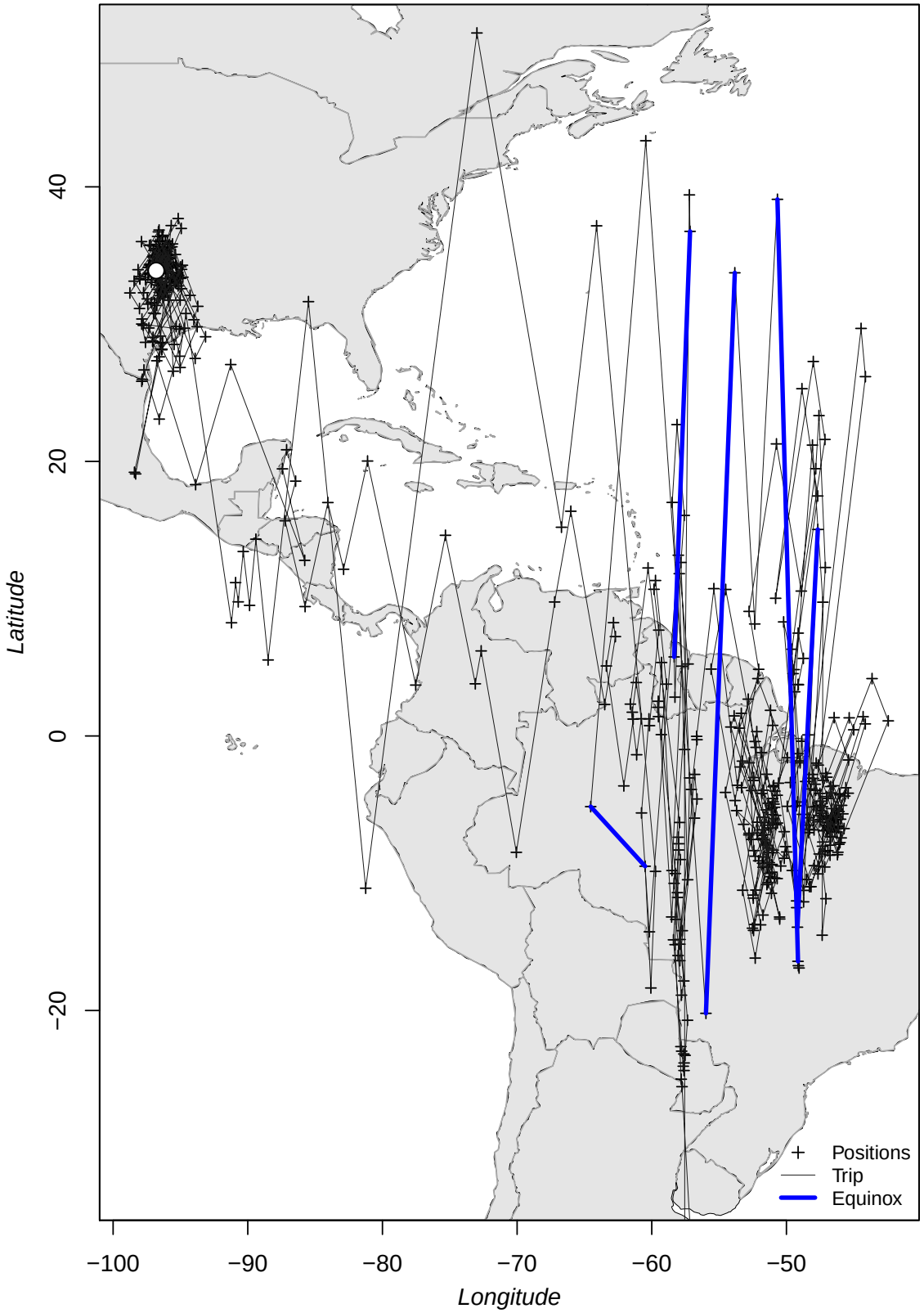
The graph produced by `getElevation` can help to judge whether your calibration data is sufficiently long and whether it is correct. A twilight error that fits the gamma distribution is a good sign, that you have enough data for your calibration. In case you have a few bars only, the fit is poor, and 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

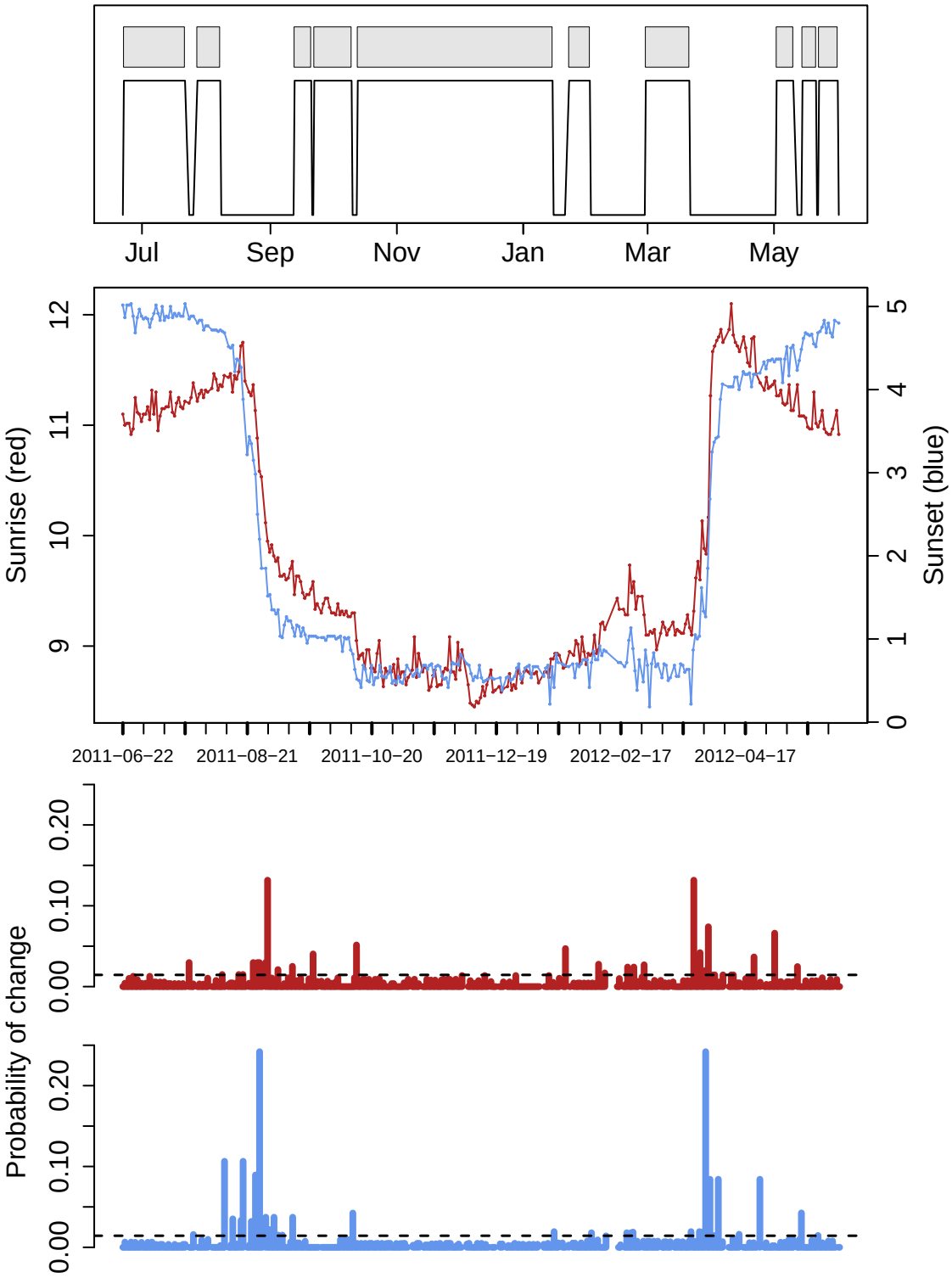
GeoLight has tools to make simple adjustments and refine the location estimates. First, we can use alternative calibration methods to see if the calibration at the known site is actually representative for the twilight error during the entire year. Assuming a fixed sun elevation angle is often not correct. 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)

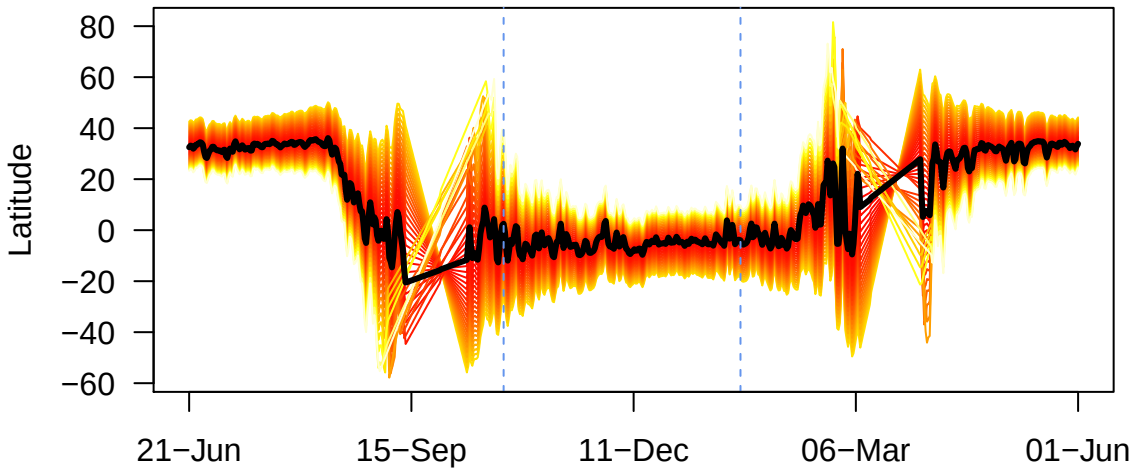
```
cL <- changeLight(twl = twl.gl, quantile = 0.8)
```

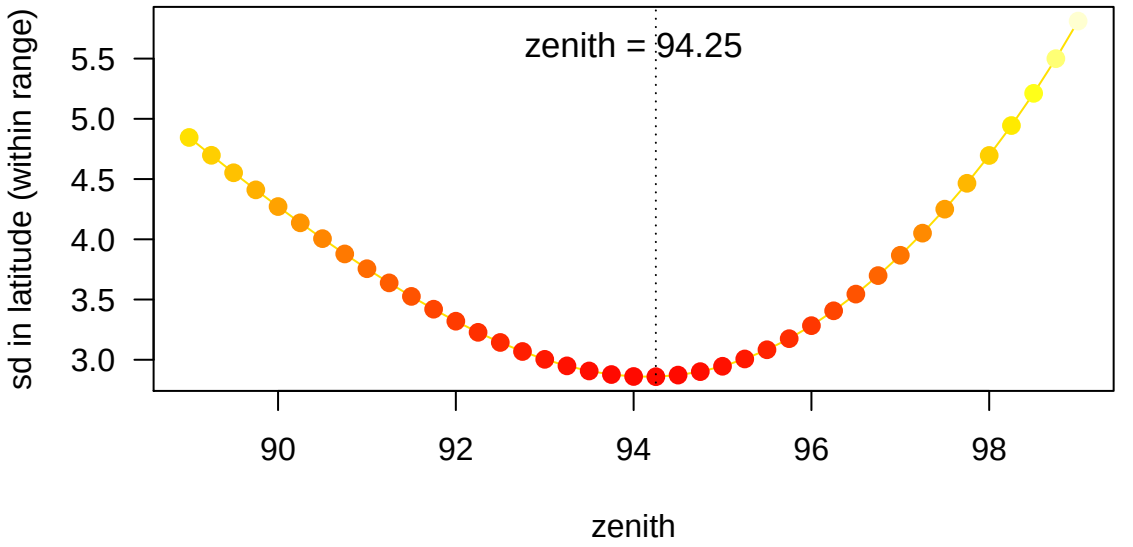


Besides other information, the `changeLight` function returns a vector with the sites `cL$site` that we can use to subset the twilight table for the e.g. longest period (in this case stationary period Nr. 5). **Important**, the function we use to run the *Hill-Ekstrom calibration*, `findHEZenith`, is 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

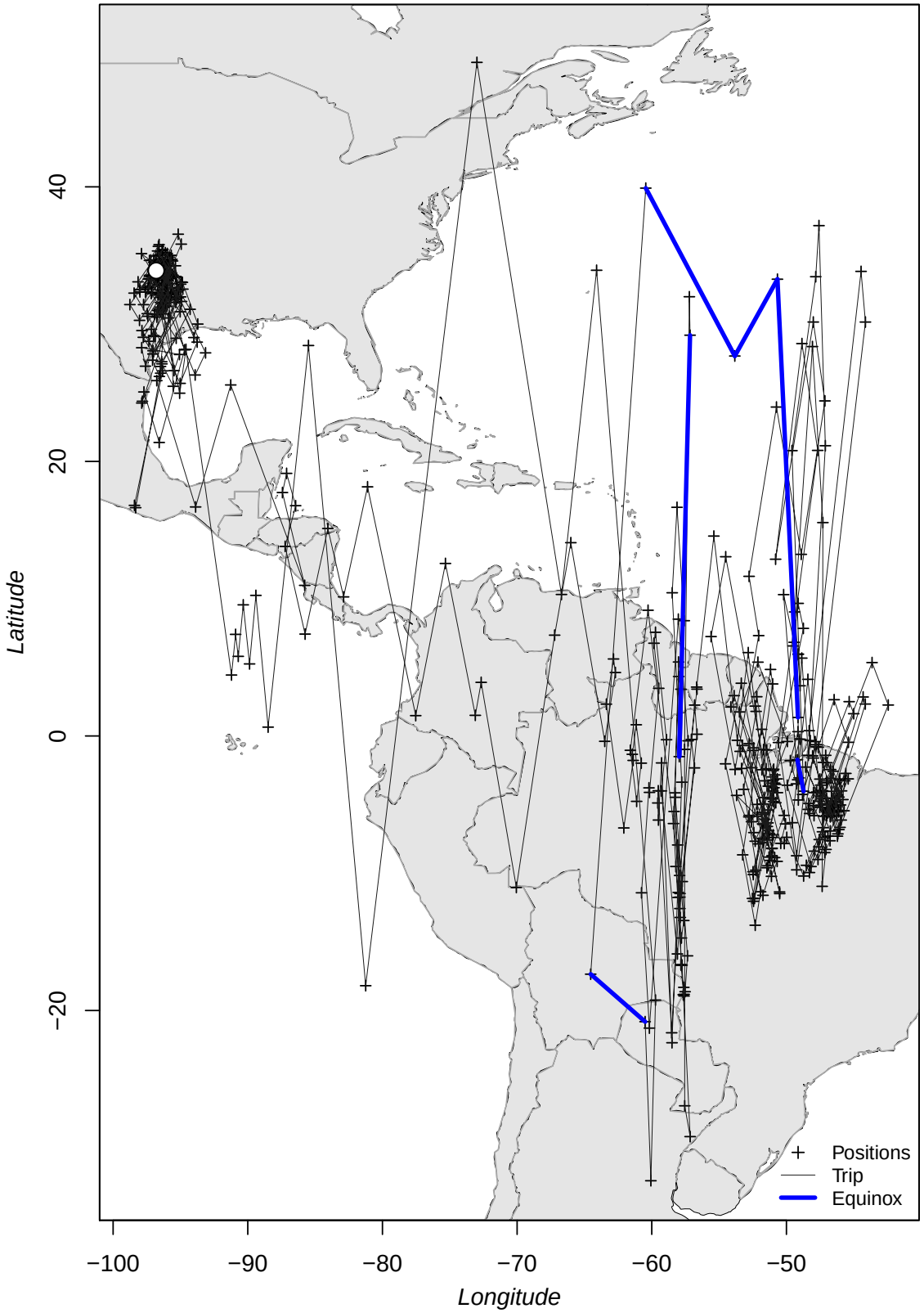
for the calibration period. This is not massive but still a significant difference probably explained by the non-breeding site being close to the equator with higher likelihood of clouds than in e.g. south of the US.

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 %)
```

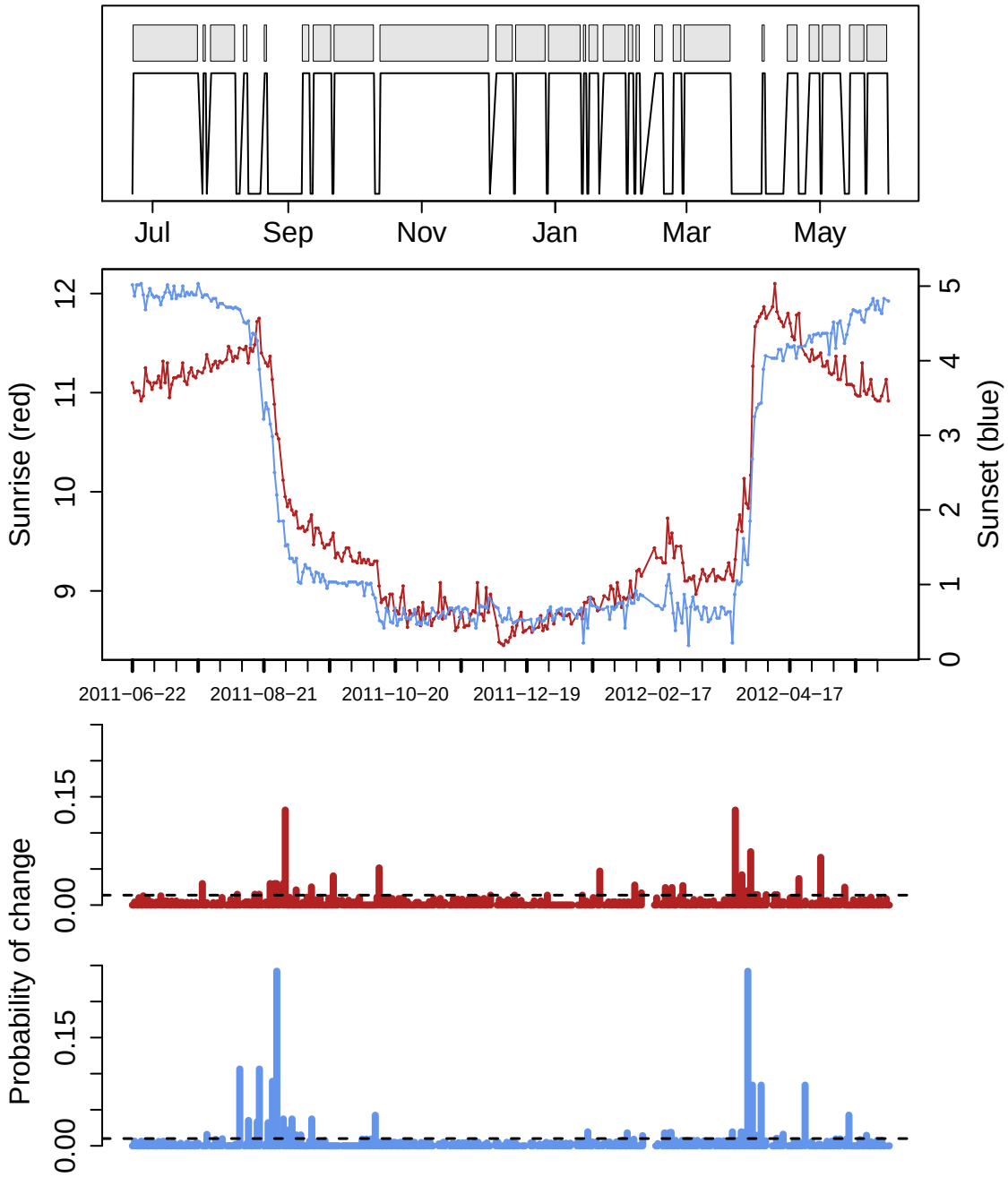
```
## using tripMap
tripMap(crds, xlim = c(-98.75, -42.4), ylim = c(-32, 50))
points(lon.calib, lat.calib, pch = 21, cex = 1.5, bg = "white") # adding the release location
```


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)
```



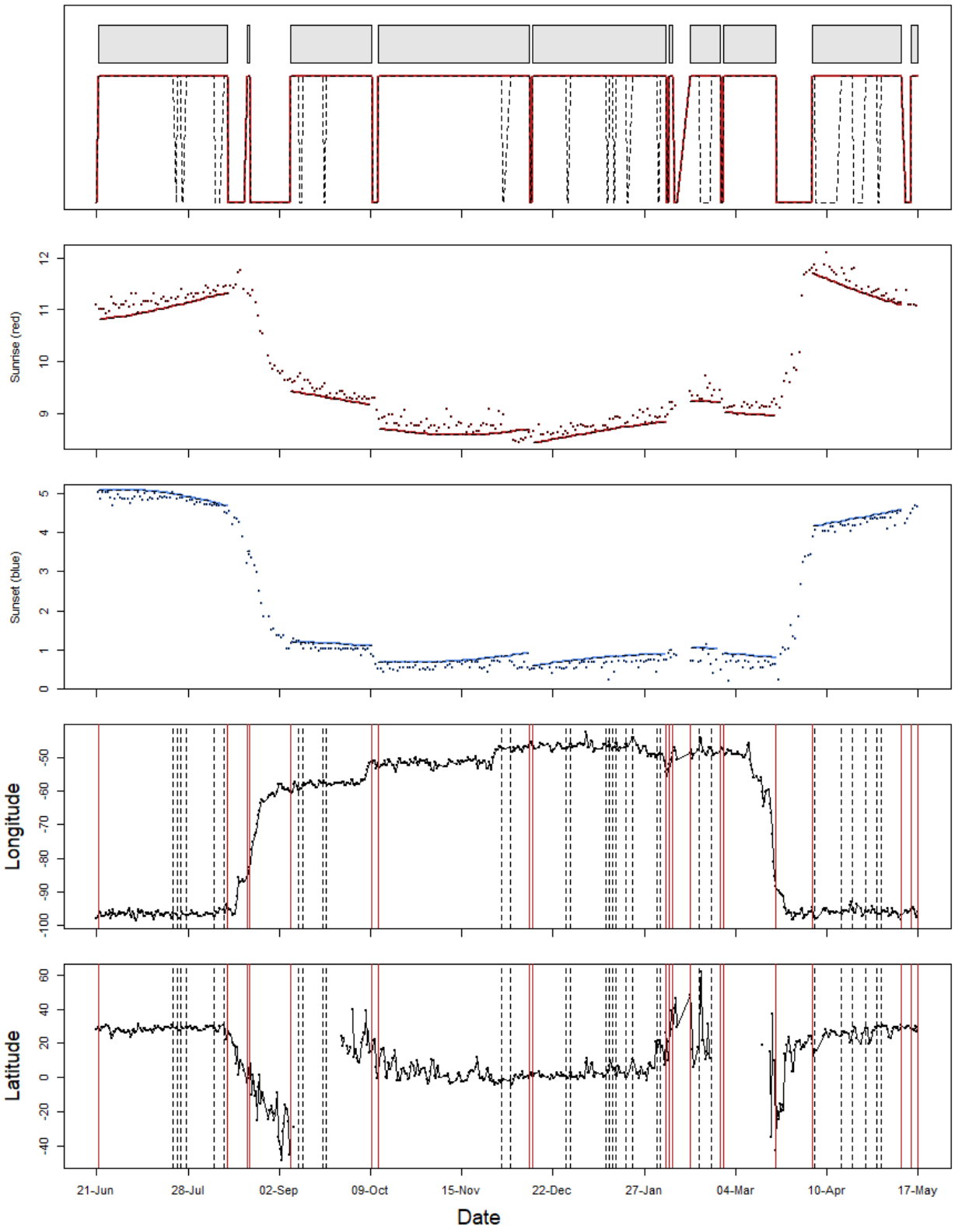
These settings seem to detect all changes that are obviously visible (and more). We can now use `mergeSites` to refine this selection and to merge consecutive sites if they are only separated by single large errors in the twilight times but are otherwise likely to be at the same site. `mergeSite` uses 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
```

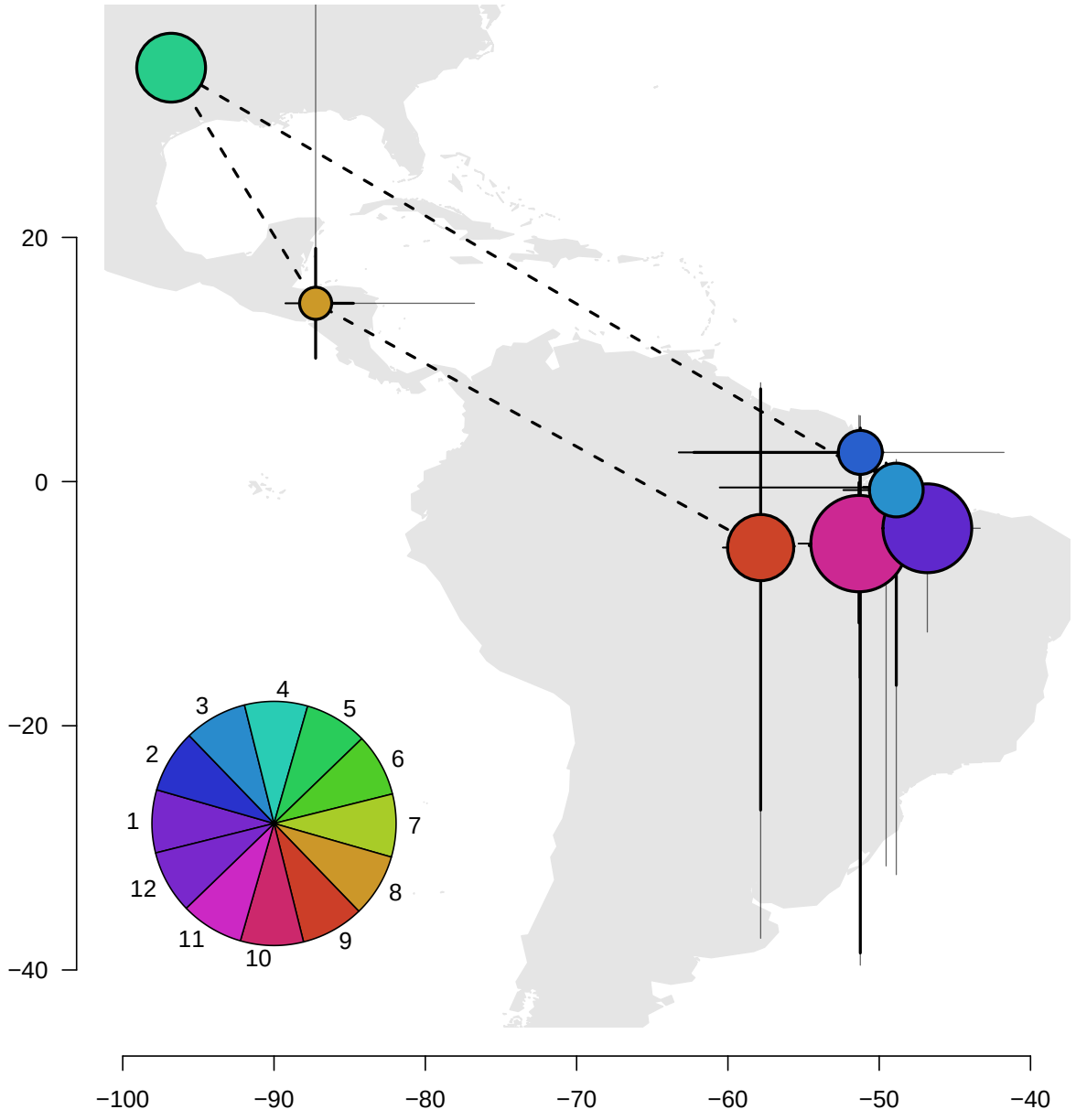


The plot created by `mergeSites2` is similar to what we know from `changeLight`. However, the top panel shows which of the original sites were merged (red lines around dotted lines), and the same is shown for the

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.

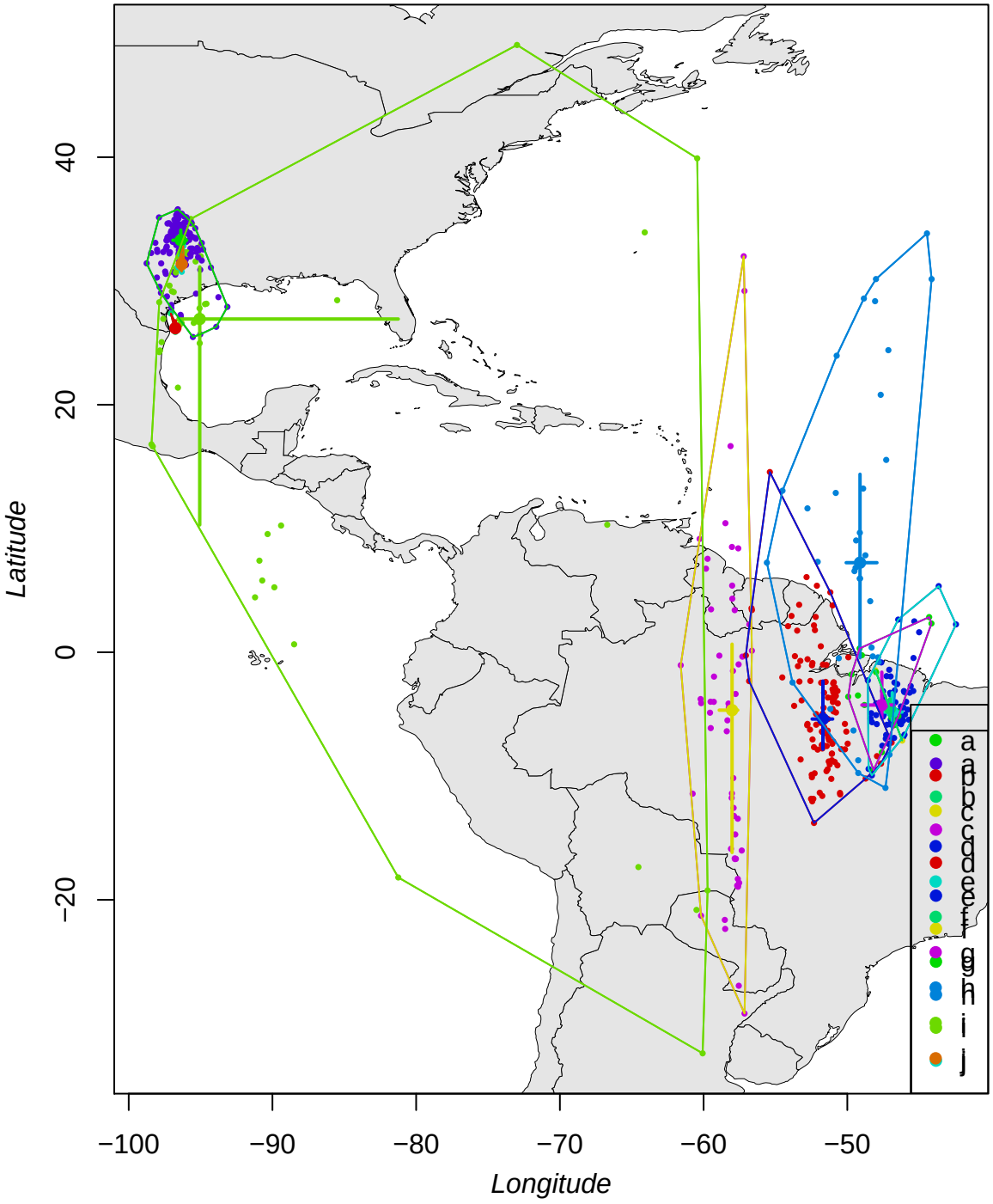
We can now plot the results, using the longitude and latitude estimates of `mergeSites2`.

```
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.

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.

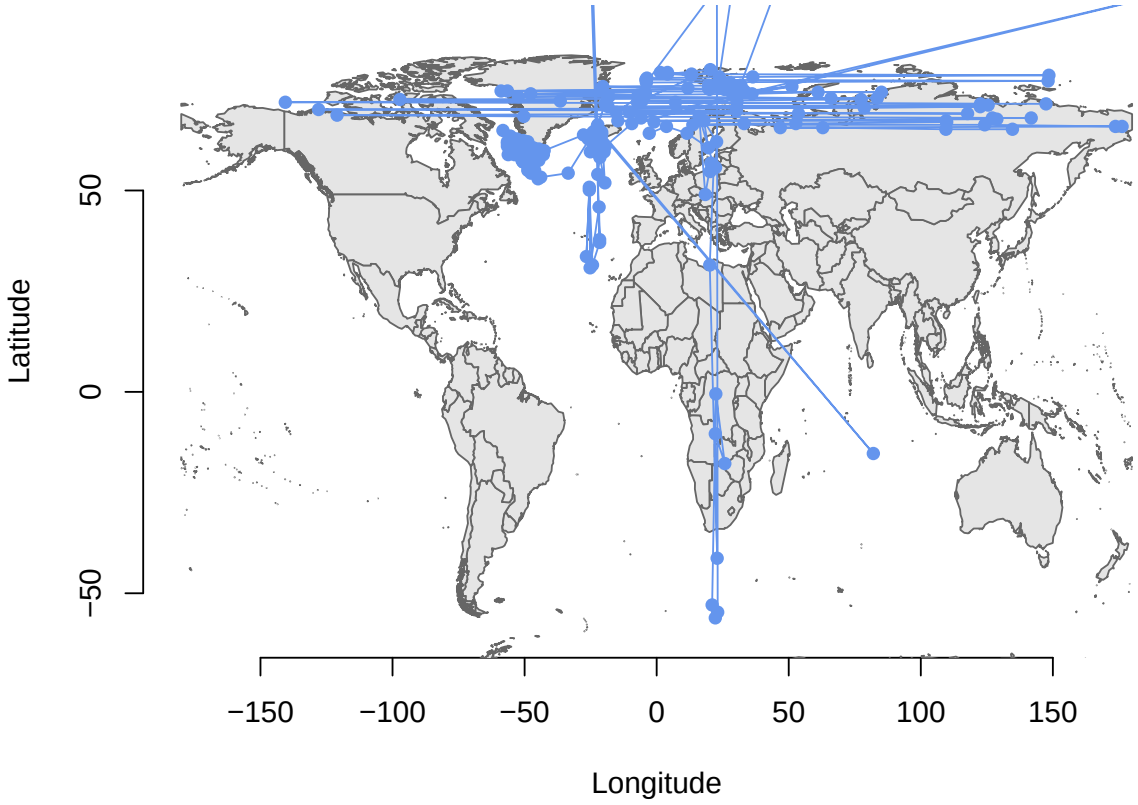
We first define the metadata and read in the raw recordings.

```
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")
```



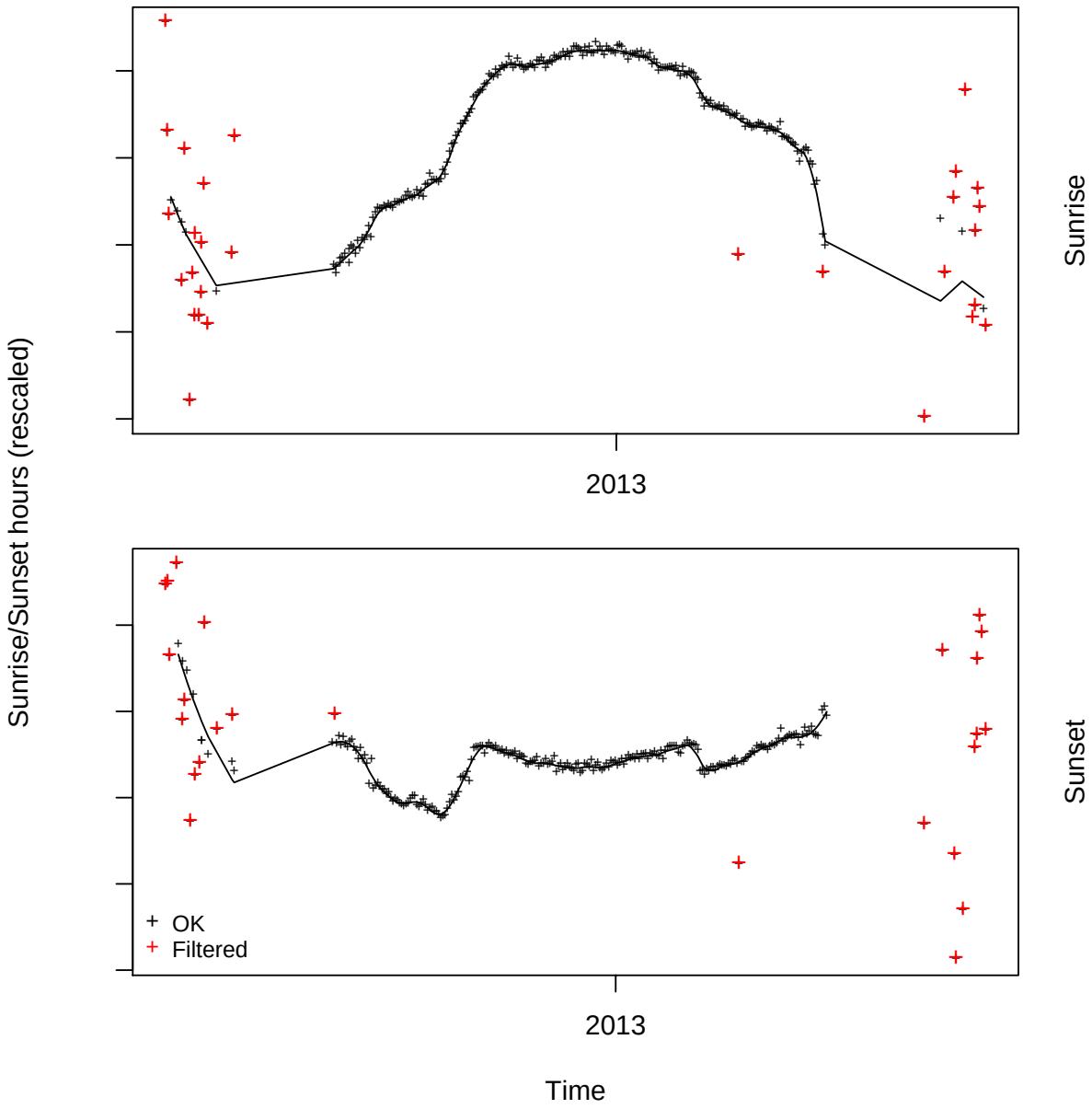
Therefore, we back calculate times of sunrise and sunset using this angle and positional data provided by the day log file.

```
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.

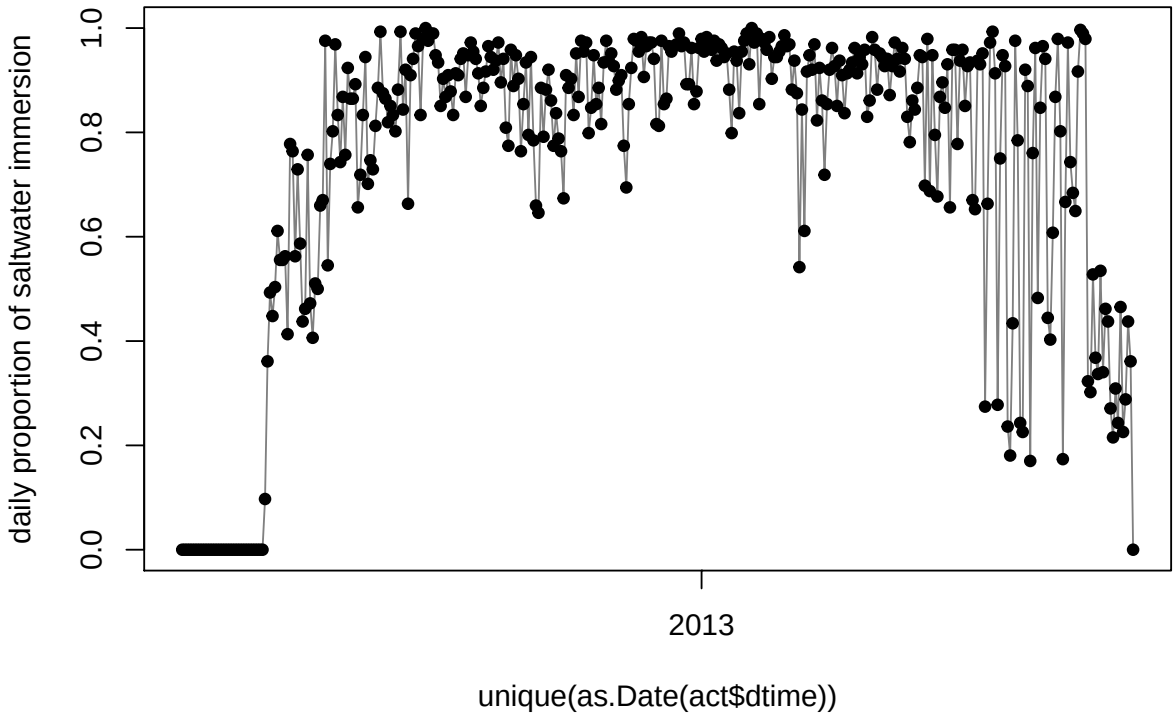
In this example the sampling rate is every 5 minutes. However, this may differ between the actual geolocator models.

```
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
```

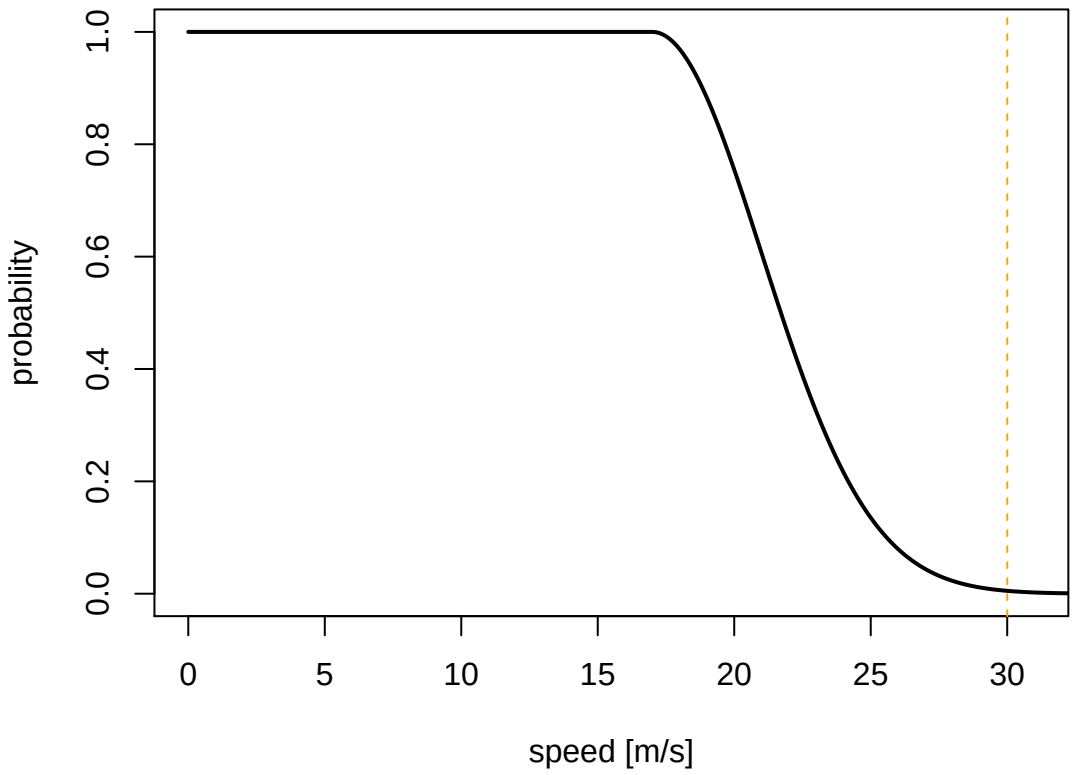
Using this auxiliary data, we can plot the daily proportion the logger has been submerged in saltwater.

```
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)
```



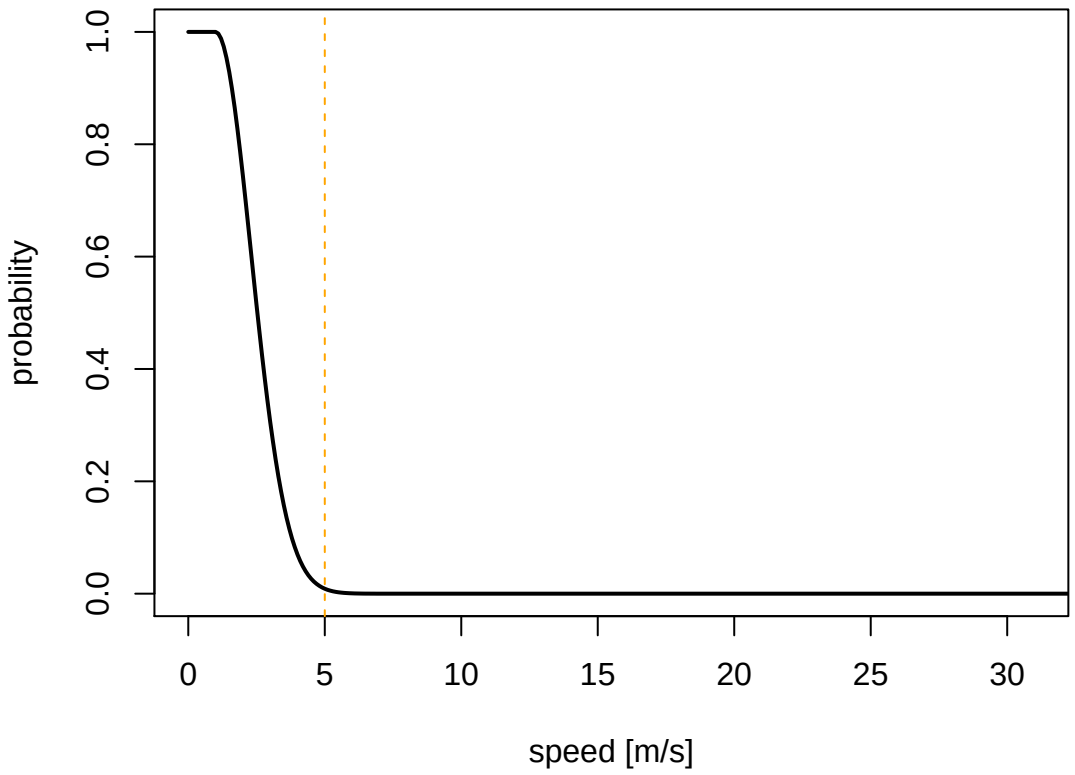
The here shown proportion, the logger is immersed in salt water each day can be used to estimate behaviour states of the individual tracked (e.g. sitting on water or flying). Additionally, the movement speed of a bird on or in water is not as high as when it is airborne, allowing us to define 2 speed 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.

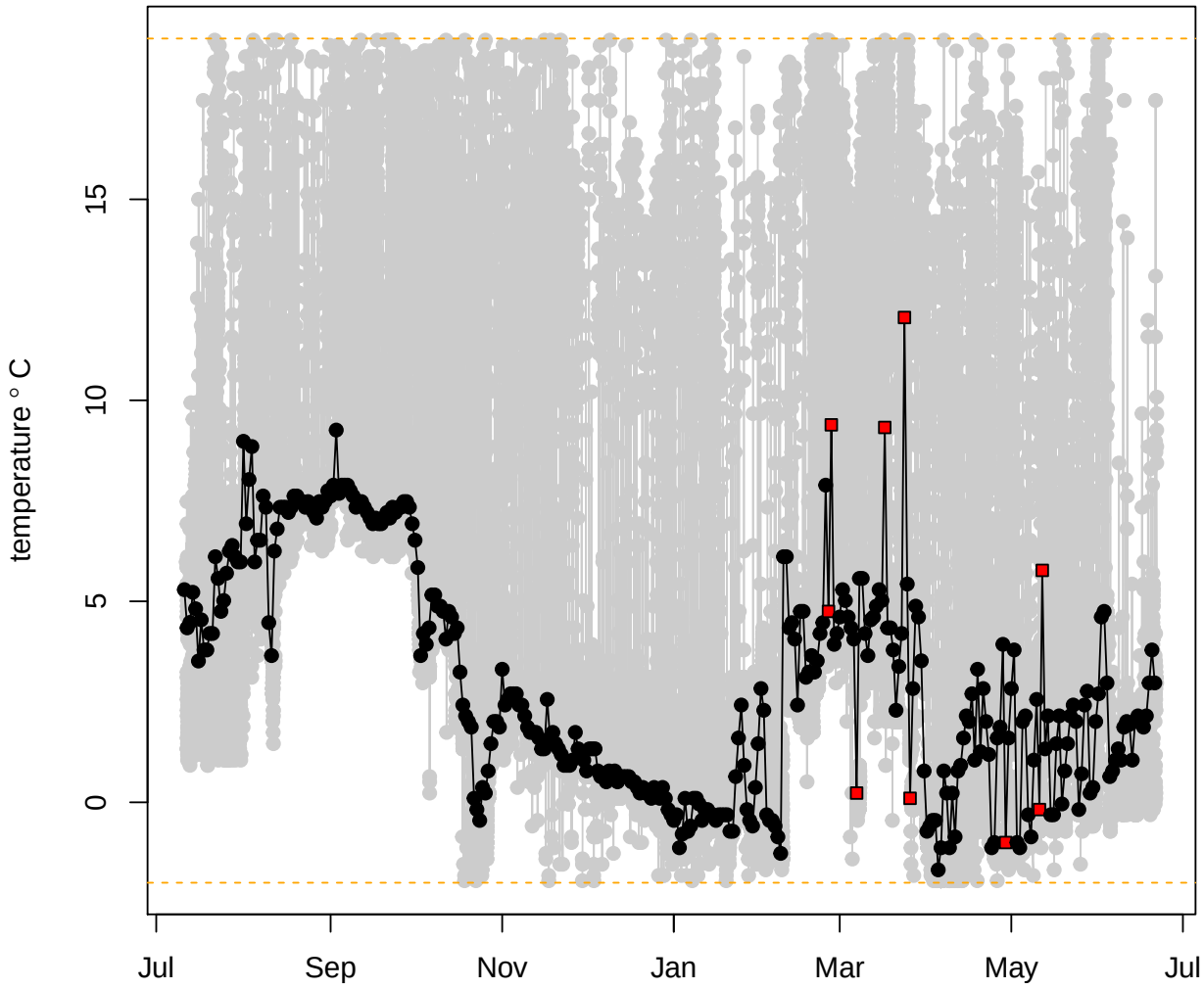
```
speed_wet  = c(1, 1.3, 5)

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

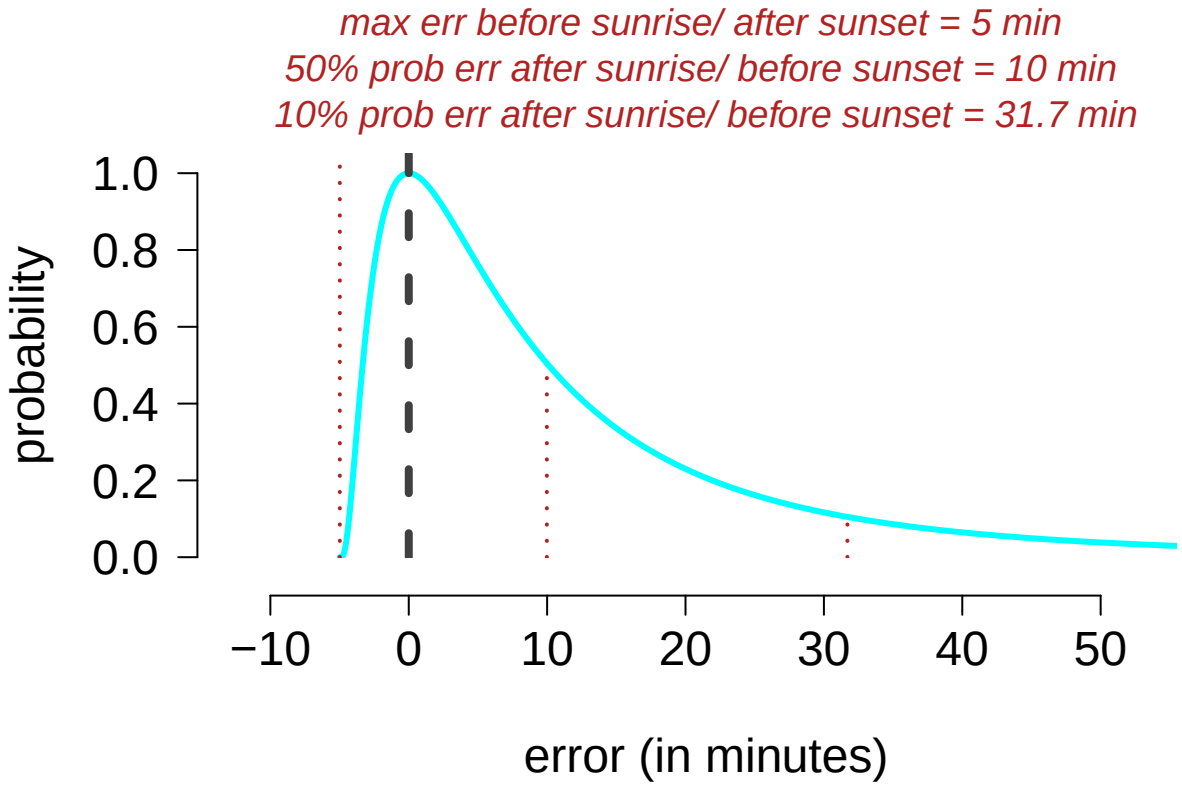
Now, we need remote sensed SST fields to fit the deduced SST values recorded by the logger with what was available in the environment. Here we use one of many remote sensed data products. The strength of this product is the long temporal time scale (1981 - now) coupled with a reasonable spatial resolution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

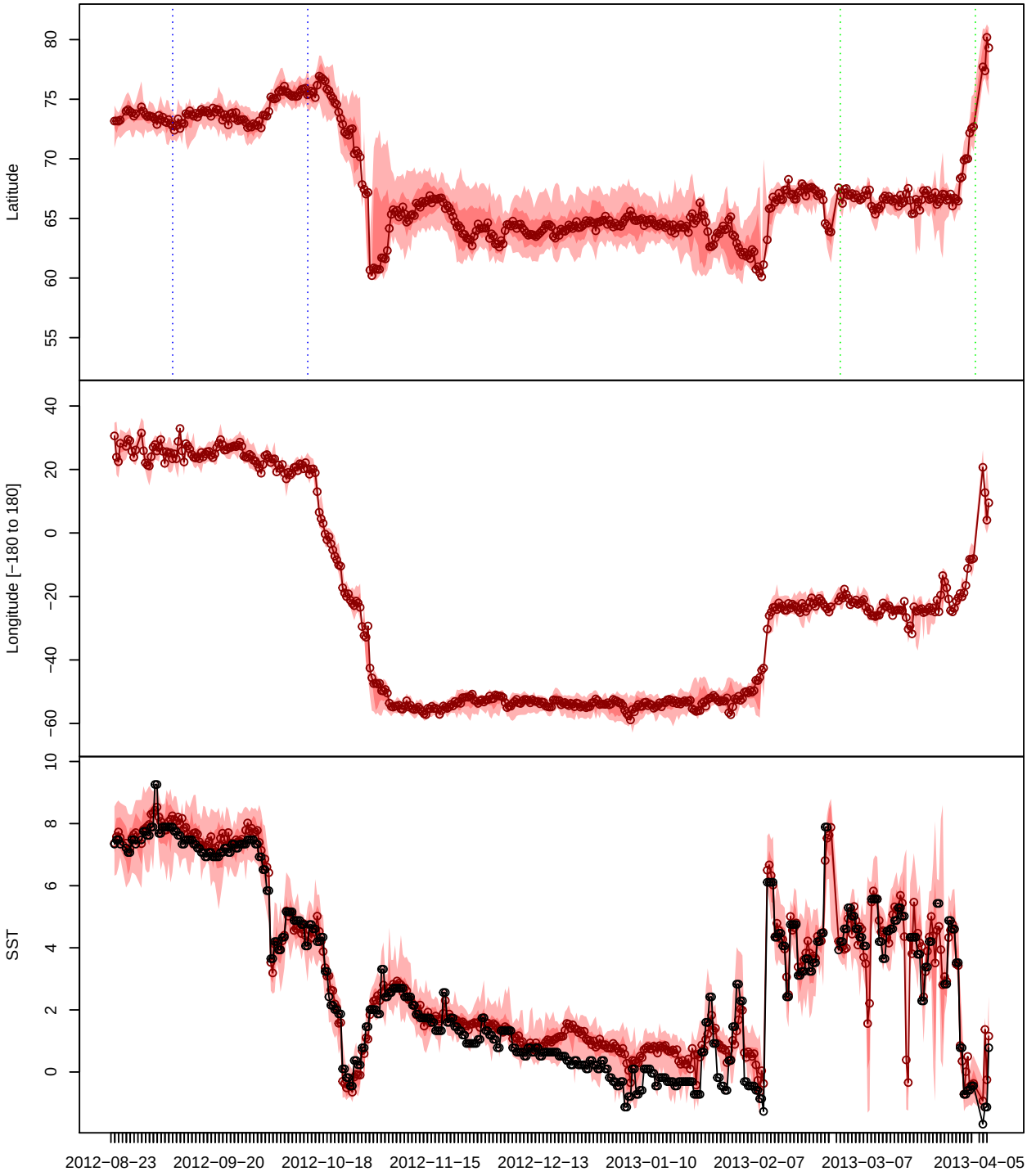
```
summary(pr)
```

```
                       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
```

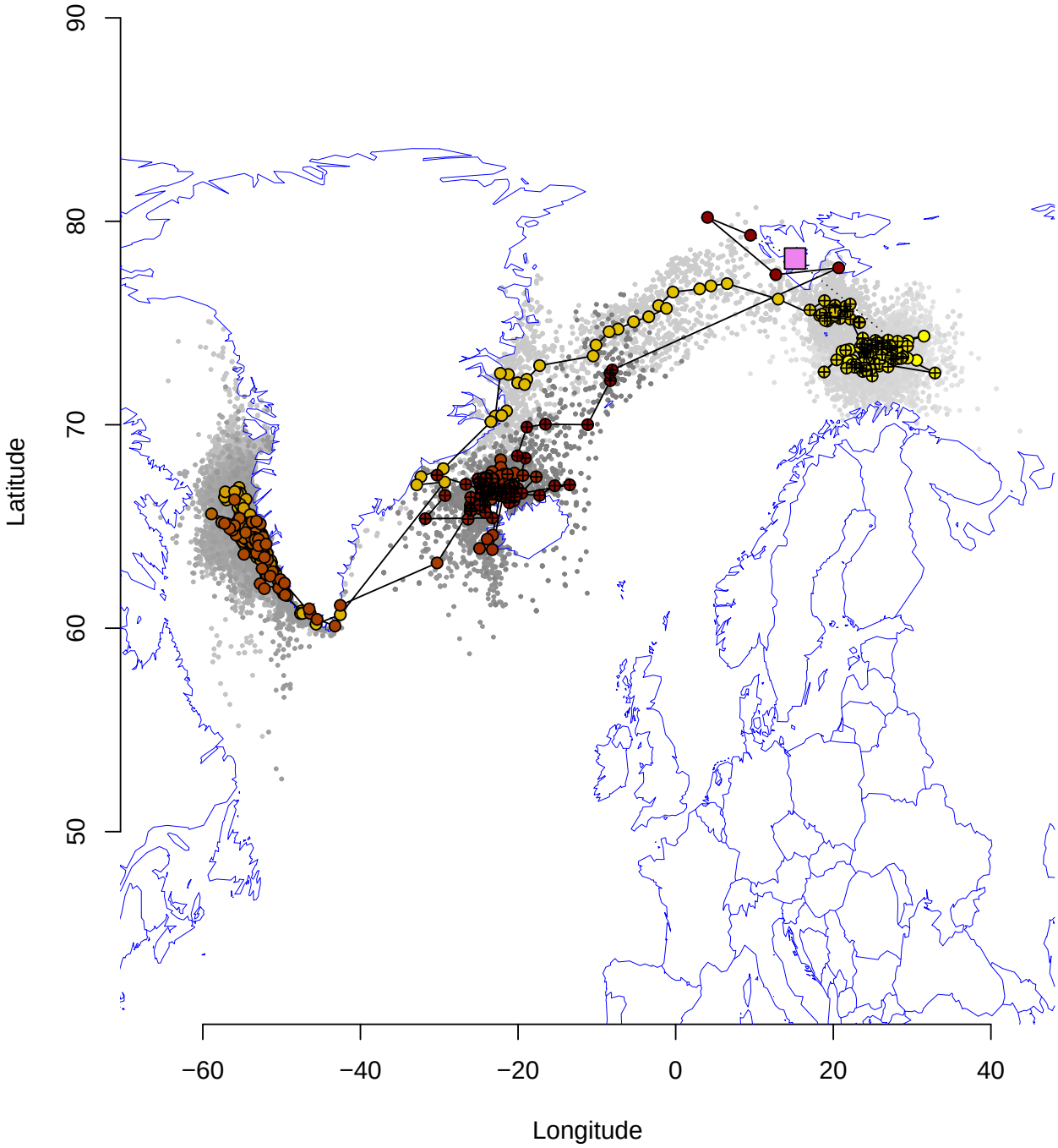
- The first item contains all 100 tracks computed by the algorithms.
- The second item contains the most probable track as geographic median at each step.
- The third item contains all generates particles at each step.
- The fourth item stores all input parameter.
- The last item is just the time it 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

...

Chapter 8

FLightR

...

Chapter 9

Data repositories

...

Chapter 10

Your contribution

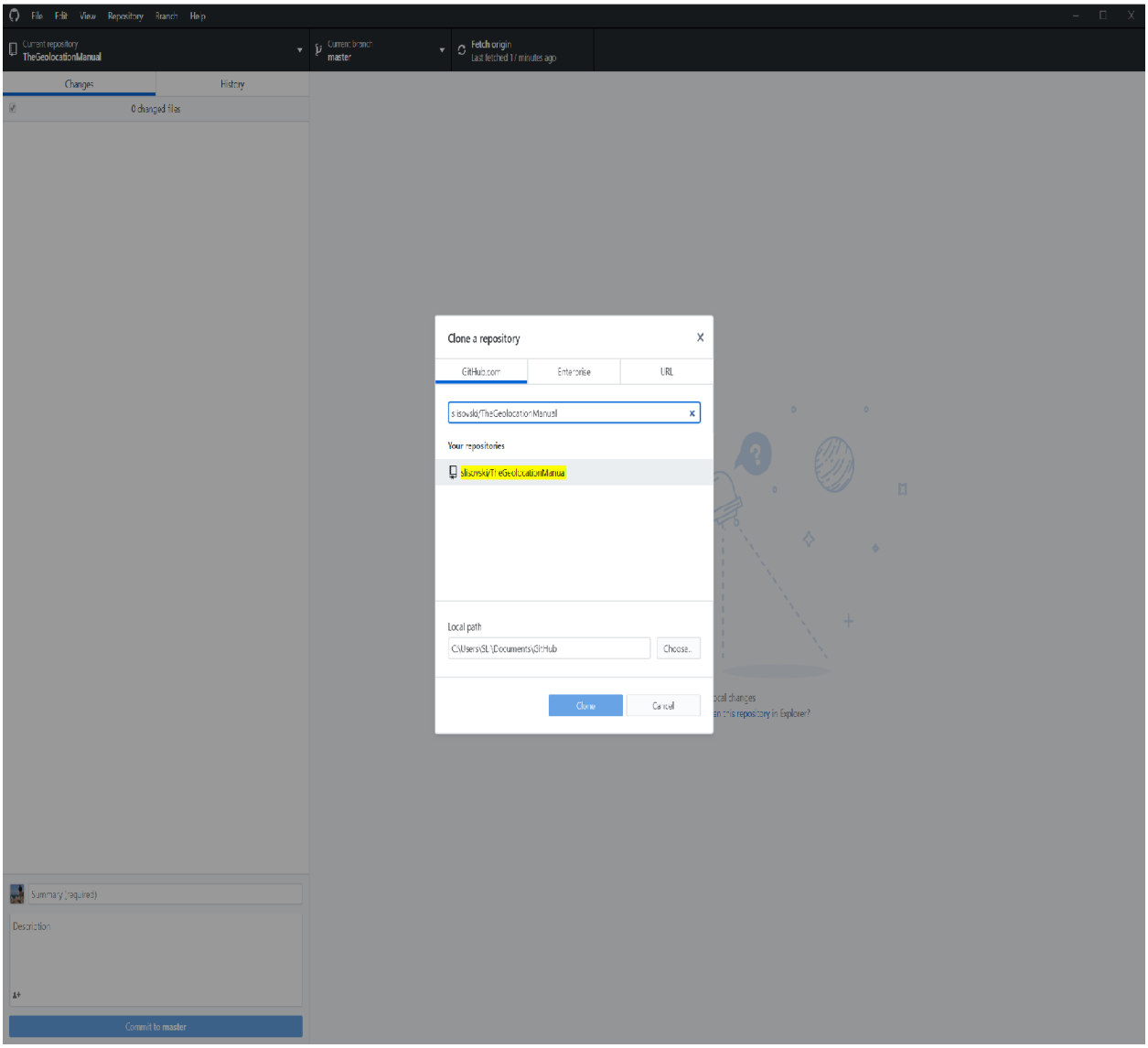
We do consider this document to be a community endeavour for which we have given a starting point. We apprechiate any contribution in terms of thoughs on the current version, changes and additions. Arguably, your expereince with geolocation is valuable for the community and by adding it to this manual 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.

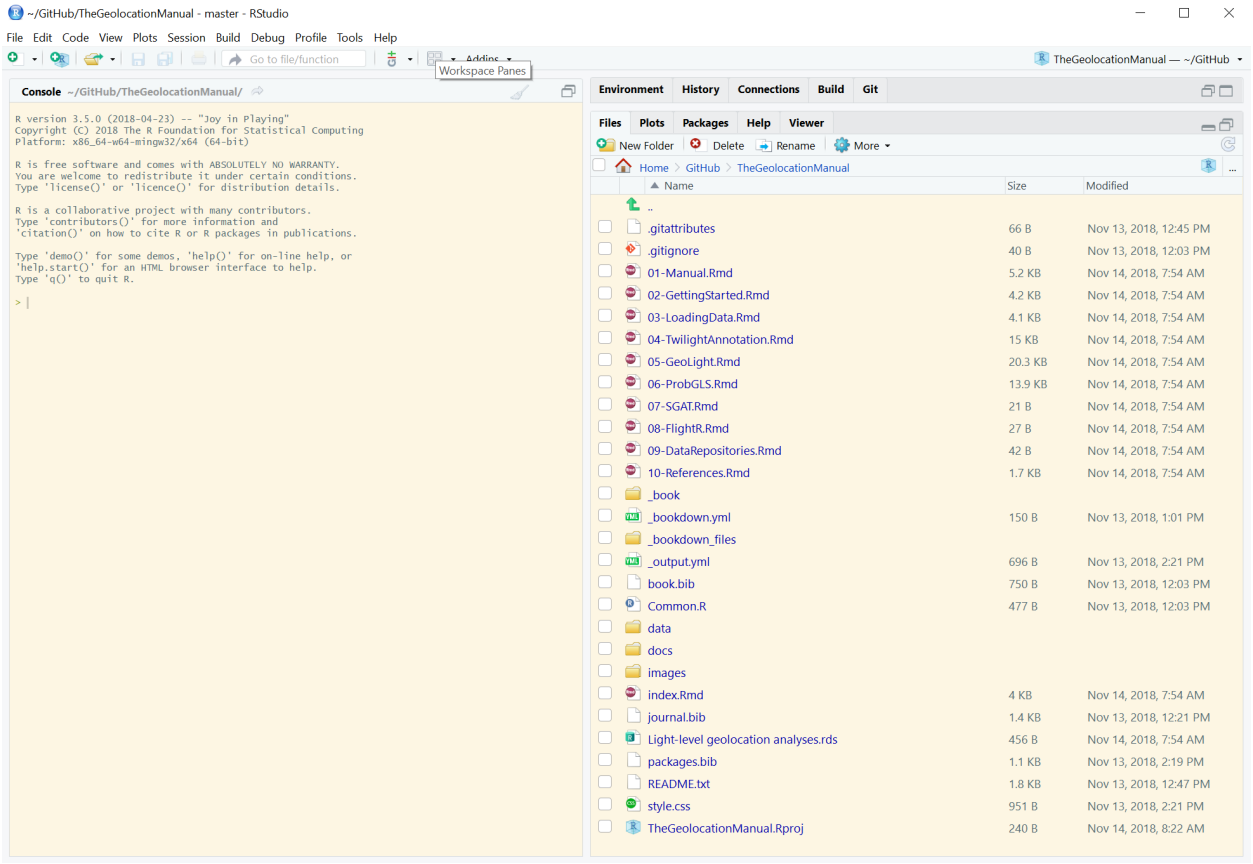
If you have not already created an account do so (www.github.com). And while RStudio has implemented a Git environment that allows direct communication with GitHub, we recommend to download the GitHub Desktop application if you are new to this world of version control. The application can be downloaded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

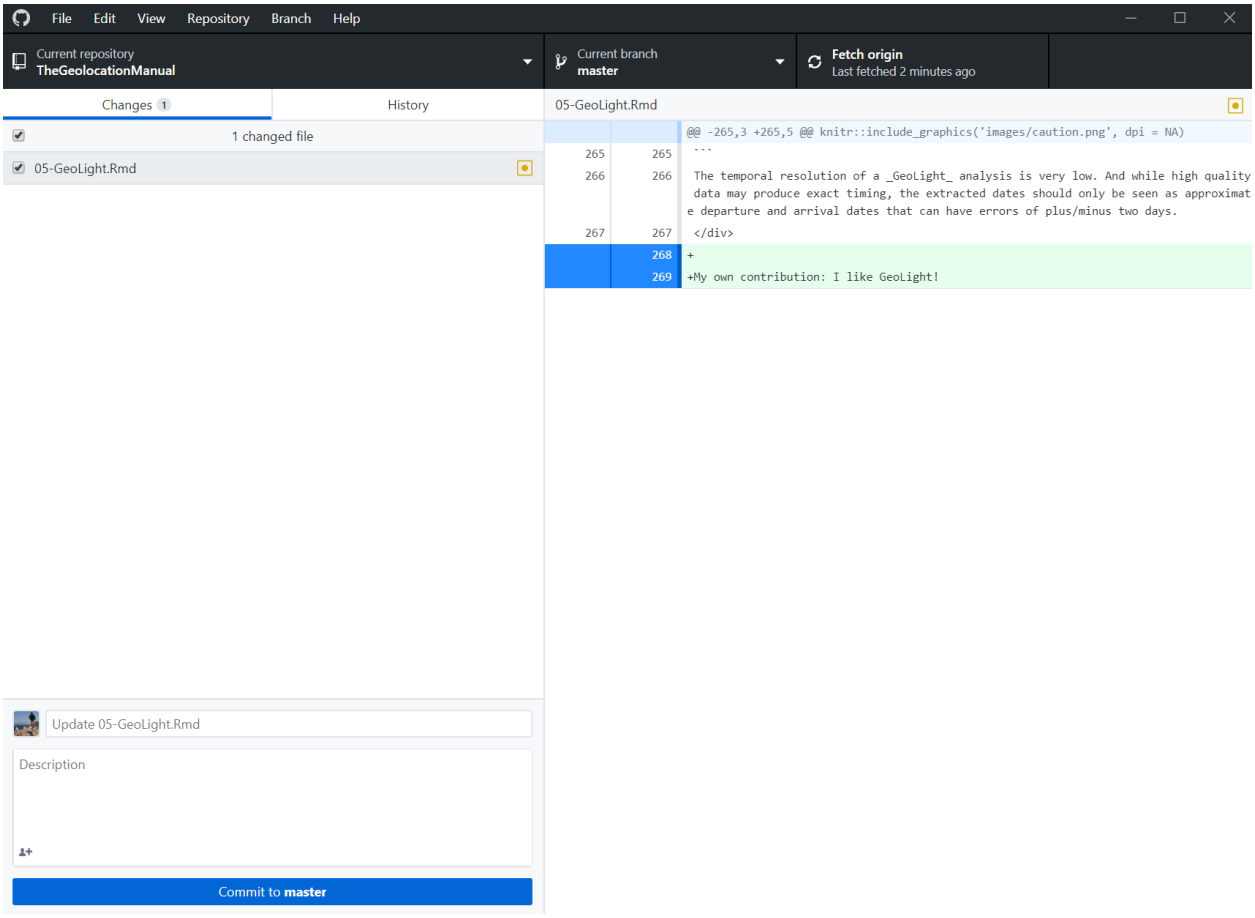


Navigate to the local folder that should now contain a subfolder called *TheGeolocationManual*. In this subfolder you will find a RStudio project file called "TheGeolocationManual.Rproj". Open this file with a double click. In RStudio you are now working within this folder and in the *Files* window,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

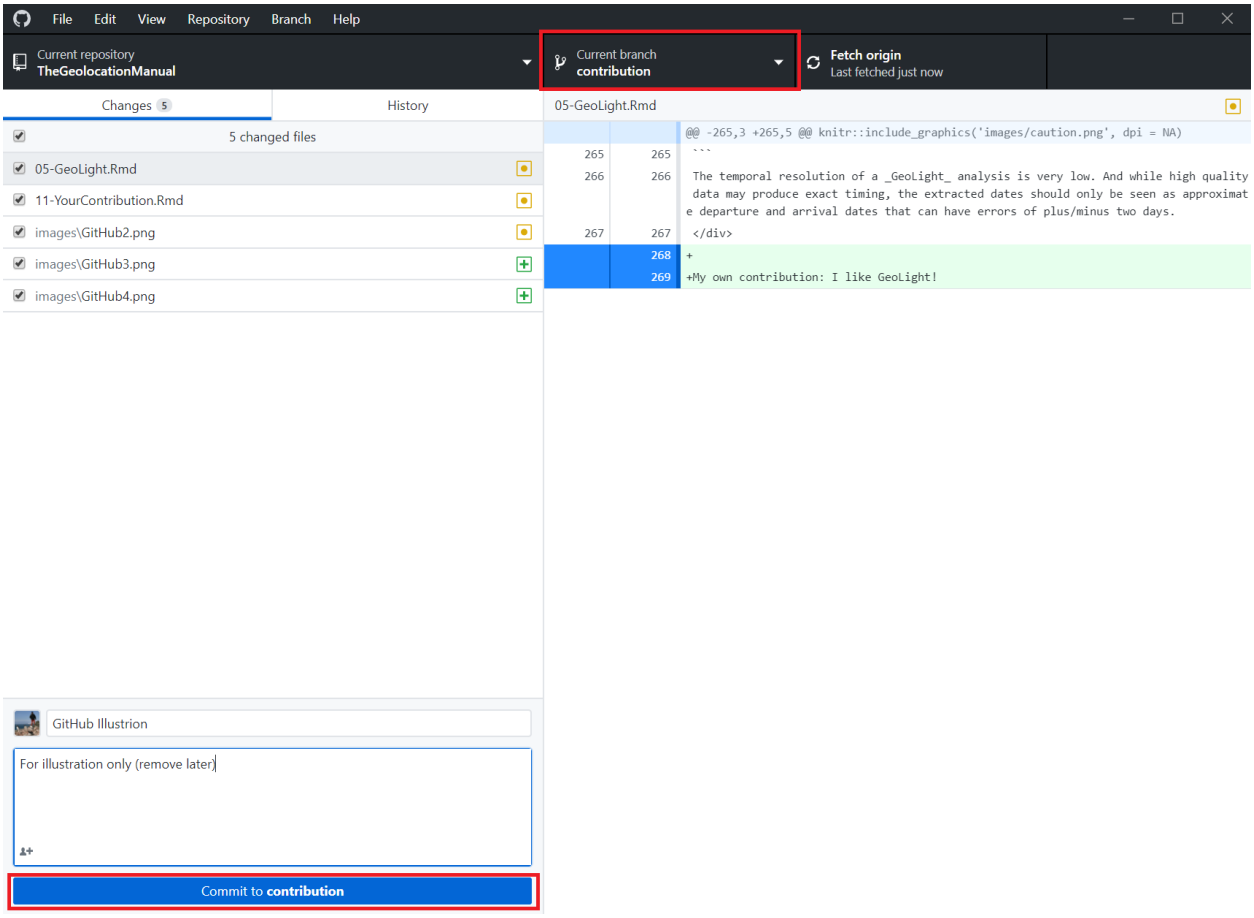


Make sure to *fetch* (pressing *Fetch origin*) before you start making any edits in RStudio. This gives you the newest version and avoids conflicts with edits from other users!

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.

Bibliography

Lisovski, S., Hewson, C. M., Klaassen, R. H. G., Korner-Nievergelt, F., Kristensen, M. W., and Hahn, S. (2012). Geolocation by light: accuracy and precision affected by environmental factors. *Methods in Ecology and Evolution*, 3(3):603–612.