

# Using R tools for analysis of primary biodiversity data provided by SBDI

Debora Arlt and Alejandro Ruete for the Swedish Biodiversity Data Infrastructure

2021-05-31

## Contents

<b>Introduction</b>	<b>1</b>
R and Mirroreum . . . . .	2
SBDI4R - an R to search an access data . . . . .	2
Other packages needed . . . . .	4
Your collaboration is appreciated . . . . .	4
<b>1 Example with fish data from SERS</b>	<b>5</b>
1.1 Plotting data on a map . . . . .	6
1.2 Temporal summary . . . . .	7
1.3 Species summary . . . . .	8
1.4 Spatial biodiversity analysis . . . . .	9
<b>2 Example with opportunistic data on Dragonflies</b>	<b>12</b>
2.1 Name searching . . . . .	12
2.2 Filter the search to get the observations . . . . .	13
2.3 Quality and fit-for-use check . . . . .	14

## Introduction

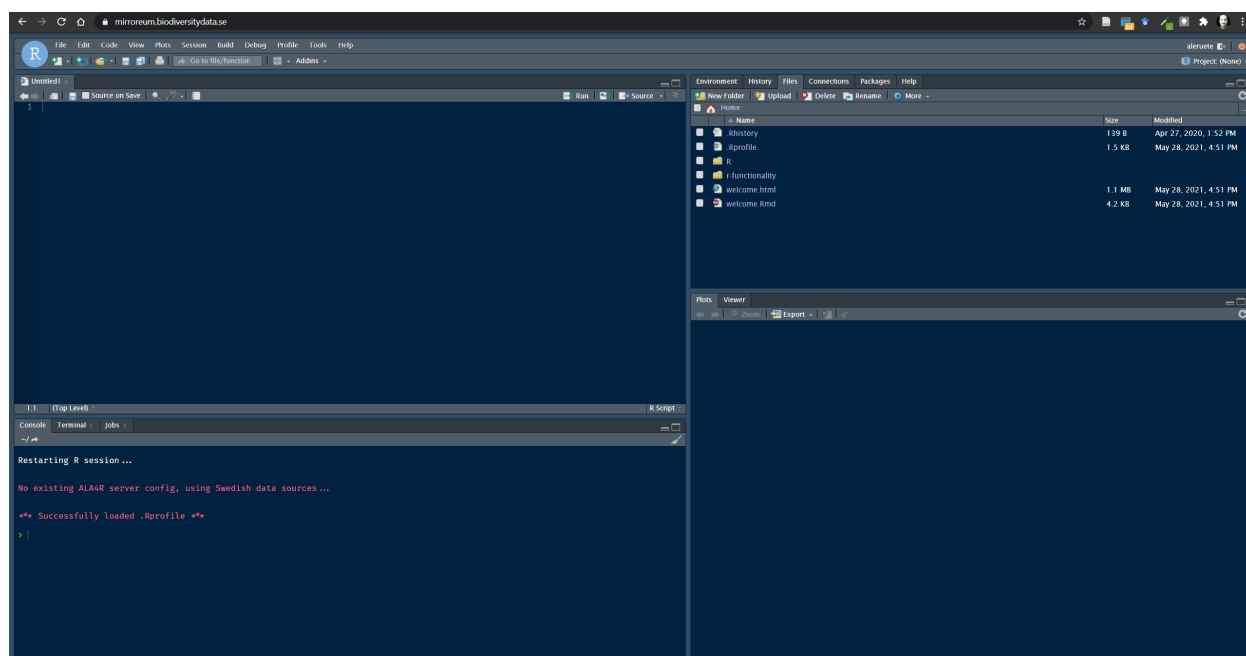
Biodiversity resources are increasingly international. The SBDI has made an effort to canalize biodiversity data and resources to help the research community access and analyze Swedish primary biodiversity data. Each research question draws its own challenges which are unique in themselves. Our aim here is to provide a few examples that prompt questions that may be asked at different stages of the process. The validity and appropriateness of a particular method depends on the individual researcher(s). For a comprehensive workflow on how to treat and analyze PBD please refer to our tutorial on [biodiversity analysis tool](#) where we go through the complete workflow Data -> Cleaning -> Fitness evaluation -> Analysis

## R and Mirroreum

The present tutorial is focused on the statistical programming language R. R is a free software environment for statistical computing and graphics that is widely used within the scientific community and where the complete analysis workflow can be documented in a fully reproducible way.

At SBDI we provide access for researchers and students to [Mirroreum](#) – an online web-based environment for Reproducible Open Research in the area of biodiversity analysis. Mirroreum is based on a Free and Open Source stack of software. Logging in, you immediately get access to a web-based version of R Studio with a large number of pre-installed packages such as all the packages offered from ROpenSci and more.

Compared to running R Studio on your own machine, Mirroreum offers more computational resources and a standardized environment where you can rely on all the relevant packages being installed and the configuration parameters being set appropriately. To know more about Mirroreum or to request an account please visit the [SBDI documentation site](#)



## SBDI4R - an R to search an access data

The SBDI4R package enables the R community to directly access data and resources hosted by SBDI. The goal is to enable observations of species to be queried and output in a range of standard formats. It includes some filter functions that allow you to filter prior to download. It also includes some simple summary functions, and some function for some simple data exploration. The examples included in this tutorial also show you how you can continue exploring and analyzing using other R package.

Please refer to the [package documentation](#) for details on how to install it. Once installed the SBDI4R package must be loaded for each new R session:

```
## Warning: package 'sp' was built under R version 4.0.4
```

## Customizing SBDI4R

Various aspects of the SBDI4R package can be customized.

**Caching** SBDI4R can cache most results to local files. This means that if the same code is run multiple times, the second and subsequent iterations will be faster. This will also reduce load on the web servers. By default, this caching is session-based, meaning that the local files are stored in a temporary directory that is automatically deleted when the R session is ended. This behaviour can be altered so that caching is permanent, by setting the caching directory to a non-temporary location. For example, under Windows, use something like:

```
sbdi_config(cache_directory = file.path("c:", "mydata", "sbdi_cache")) ## Windows
```

or for Linux:

```
sbdi_config(cache_directory = "~/mydata/sbdi_cache") ## Linux
```

Note that this directory must exist (you need to create it yourself).

All results will be stored in that cache directory and will be used from one session to the next. They won't be re-downloaded from the server unless the user specifically deletes those files or changes the caching setting to “refresh”.

If you change the `cache_directory` to a permanent location, you may wish to add something like this to your .Rprofile file, so that it happens automatically each time the SBDI4R package is loaded:

```
setHook(packageEvent("SBDI4R", "onLoad"),  
  function(...) sbdi_config(cache_directory=file.path("~", "mydata", "sbdi_cache")))
```

Caching can also be turned off entirely by:

```
sbdi_config(caching="off")
```

or set to “refresh”, meaning that the cached results will re-downloaded from the SBDI servers and the cache updated. (This will happen for as long as caching is set to “refresh” — so you may wish to switch back to normal “on” caching behavior once you have updated your cache with the data you are working on).

**E-mail address** Each download request to SBDI servers is also accompanied by an “e-mail address” string that identifies the user making the request. You will need to provide an email address registered with the SBDI. You can create an account [here](#). Once an email is registered with the SBDI, it should be stored in the config:

```
sbdi_config(email="your.valid@emailaddress.com")
```

Else you can provide this e-mail address as a parameter directly to each call of the function occurrences().

**Setting the download reason** SBDI requires that you provide a reason when downloading occurrence data (via the SBDI4R `occurrences()` function). You can provide this as a parameter directly to each call of `occurrences()`, or you can set it once per session using:

```
sbdi_config(download_reason_id = "your_reason_id")
```

(See `sbdi_reasons()` for valid download reasons, e.g. \* 3 for “education”, \* 7 for “ecological research”, \* 8 for “systematic research/taxonomy”, \* 10 for “testing”)

**0.0.0.1 Privacy** *NO* other personal identification information is sent. You can see all configuration settings, including the the user-agent string that is being used, with the command:

```
sbdi_config()
```

**Other options** If you make a request that returns an empty result set (e.g. an un-matched name), by default you will simply get an empty data structure returned to you without any special notification. If you would like to be warned about empty result sets, you can use:

```
sbdi_config(warn_on_empty=TRUE)
```

## Other packages needed

Some additional packages are needed for these examples. Install them if necessary with the following script.

```
to_install <- c("BIRDS", "colorRamps", "cowplot", "dplyr", "ggplot2",  
               "leaflet", "maps", "mapdata", "maptools", "sf", "sp",  
               "rgeos", "tidyr", "xts")  
to_install <- to_install[!sapply(to_install,  
                                requireNamespace,  
                                quietly=TRUE)]  
  
if(length(to_install)>0)  
  install.packages(to_install,  
                  repos="http://cran.us.r-project.org")
```

## Your collaboration is appreciated

Open Source also means that you can contribute. You don't need to know how to program but every input is appreciated. Did you find something that is not working? Have suggestions for examples or text? you can always

1. Reach to us via the [support center](#)
2. Submit and issue to the GitHub code repository [see how](#)
3. Or contribute with your code or documents modifications by “forking” the code and submitting a “pull request”

The repositories you can contribute to are:

- Mirroreum <https://github.com/mskyttner/mirroreum>
- SBDI4R <https://github.com/biodiversitydata-se/SBDI4R> (NOTE: we may not develop this package but instead move to a new one)
- the general analysis workflows <https://github.com/biodiversitydata-se/biodiversity-analysis-tools>
- these tutorial <https://github.com/biodiversitydata-se/r-tools-tutorial>

# 1 Example with fish data from SERS

In this example we are interested in exploring data from a specific data resource – Swedish Electrofishing Registry - SERS (Institutionen för akvatiska resurser, SLU). This data base has 2.8 M observations starting in the 1950's.

As you may already know, SBDI is a collection of many biodiversity databases. We start by searching for the data resource we are interested in using the function `pick_filter()`. This is an interactive query guiding you through the many resources available to filtering your query (data resources, spatial layers, and curated species lists).

```
library(SBDI4R)
fq_str <- pick_filter("resource")
# follow the instructions
```

Follow the instruction. Your choices here would have been “in3” -> “dr10”. Your variable `fq_str` will now contain a string “data\_resource\_uid:dr10”.

But we are not interested in the complete database, but on the last 10 years of data. for this we concatenate (add to a vector) another filter string. These will be treated as AND factors.

```
y1 <- 2008
y2 <- 2012
fq_str <- c(fq_str, paste0("year:[", y1, " TO ", y2,"]"))
# Note the square brackets are hard limits
```

For references on how to use the filters see SBDI APIS [documentation](#).

Using the function `occurrences()` we can the query for the observations fulfilling our filter. If you haven't specified that in the `sbdi_config()` before, you need to pass your email and the download reason.

```
xf <- SBDI4R::occurrences(fq = fq_str,
                          email = "sbdi4r-test@biodiversitydata.se",
                          download_reason_id = 10)

# Remove what is not a species
xf$data <- xf$data[xf$data$rank %in% c("species", "subspecies", "variety", "form", "cultivar"),]

# Simply summarise all records by data source
table(xf$data$dataResourceName, xf$data$dataResourceID)
```

```
##
##
##      SLU Aqua  Institute of Freshwater Research Swedish Electrofishing Registry - SERS 93205
```

```
table(xf$data$dataResourceID)
```

```
##
##      dr10
##      93205
```

## 1.1 Plotting data on a map

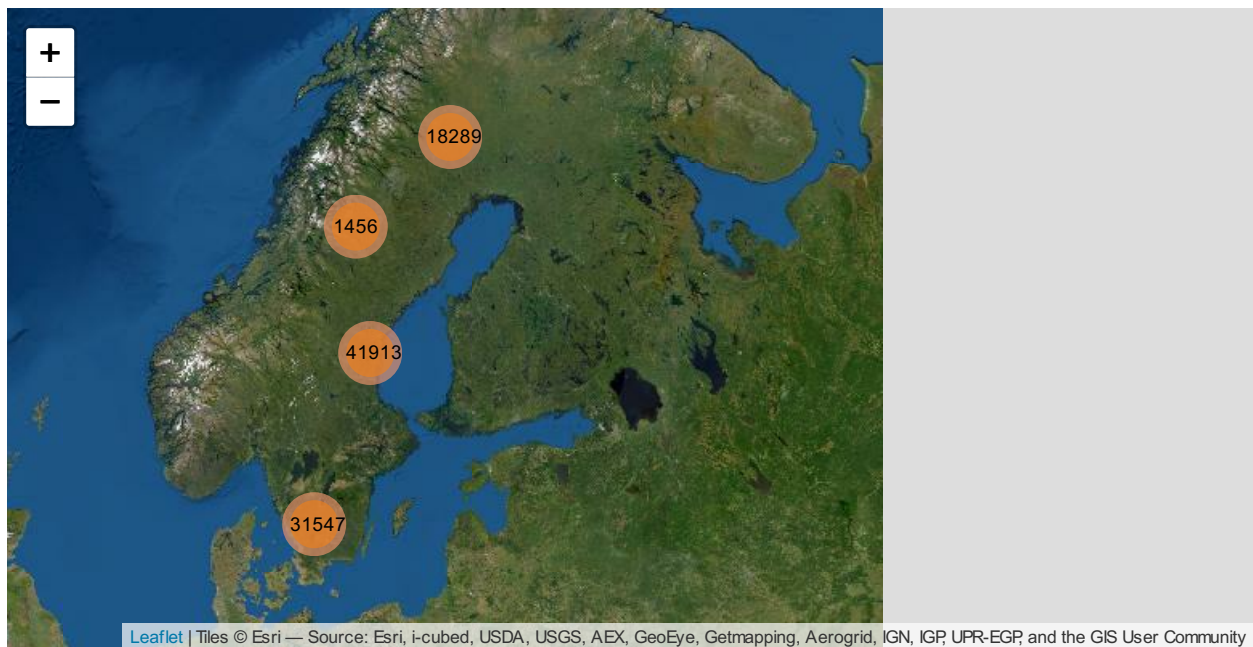
You can quickly plot all the observations as a PDF file with the function `occurrence_plot()`, one page per species:

```
occurrences_plot(xf, "obsPlot.pdf",  
                 grouped=FALSE,  
                 taxon_level="species",  
                 pch='.')
```

Note that the plot is saved to a pdf file in the current working directory. You can find that with `getwd()`.

**1.1.0.1 Leaflet** There are many other ways of producing spatial plots in R. The leaflet package provides a simple method of producing browser-based maps with panning, zooming, and background layers:

```
library(leaflet)  
# drop any records with missing lat/lon values  
xfl <- xf$data[!is.na(xf$data$longitude) | !is.na(xf$data$latitude),]  
marker_colour <- rep("#d95f02", nrow(xfl))  
# blank map, with imagery background  
leaflet(width = "100%") %>%  
  addProviderTiles("Esri.WorldImagery") %>%  
  # add markers  
  addCircleMarkers(xfl$longitude, xfl$latitude,  
                   radius = 1,  
                   fillOpacity = .5,  
                   opacity = 1,  
                   col=marker_colour,  
                   clusterOptions = markerClusterOptions())
```



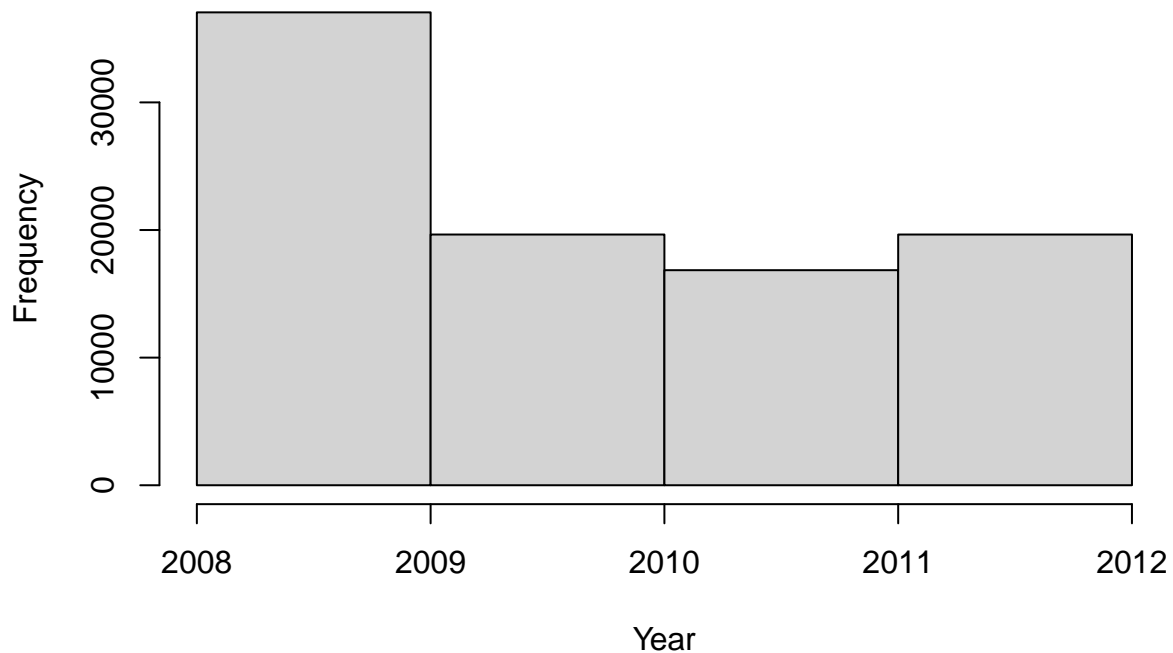
## 1.2 Temporal summary

A quick summary over the years reveal a drop in number of records over time.

```
table(xf$data$year)
```

```
##
## 2008 2009 2010 2011 2012
## 17757 19300 19648 16853 19647
```

```
hist(xf$data$year,
     breaks = seq(y1, y2),
     xlab = "Year",
     main = "")
```



### 1.3 Species summary

In the same way we can summaries the number of observations for each species, by common or scientific name.

```
sppTab <- table(xf$data$commonName)
sppDF <- as.data.frame(sppTab)
colnames(sppDF)[1] <- "species"
head(sppDF)
```

```
##           species Freq
## 1                66
## 2 Alpine bullhead 4615
## 3 American burbot 7081
## 4           Aral asp    6
## 5       Arctic char   46
## 6   aurora trout  856
```

```
sppTab <- table(xf$data$scientificName)
sppDF <- as.data.frame(sppTab)
colnames(sppDF)[1] <- "species"
head(sppDF)
```

```
##           species Freq
```



```
## 1      Abramis brama (Linnaeus, 1758)    61
## 2    Alburnus alburnus (Linnaeus, 1758) 660
## 3    Anguilla anguilla (Linnaeus, 1758) 2140
## 4      Astacus astacus (Linnaeus, 1758) 618
## 5 Barbatula barbatula (Linnaeus, 1758) 620
## 6      Blicca bjoerkna (Linnaeus, 1758)  74
```

Perhaps, you need to send this table as a .CSV file to a colleague.

```
write.csv(sppDF, "SERS_species_summary.csv")
# NOTE: again this will be saved on your working directory
```

## 1.4 Spatial biodiversity analysis

Let's now ask: how does the species richness vary across Sweden? In this case we want to summarise occurrences species-wise over a defined grid instead of plotting every observation point. First we need to overlay the observations with a grid. In this case, the standard Swedish grids at 50, 25, 10 and 5 km are provided as data in the SBDI4R package (with Coordinate Reference System = WGS84, EPSG:4326).

```
library(sp) # the function coordinates() and proj4string() are in sp
library(rgeos) # the function over() is in package rgeos
# load some shapes over Sweden's political borders
data("swe_wgs84", package="SBDI4R", envir=environment())
# A standard 50km grid
data("Sweden_Grid_50km_Wgs84", package="SBDI4R", envir=environment())

grid <- Sweden_Grid_50km_Wgs84

# make the observations spatial
# NOTE: make sure there are no NAs on either column defining the coordinates
# xf$data[!is.na(xf$data$longitude) | !is.na(xf$data$latitude),]

obs <- as.data.frame(xf$data)
coordinates(obs) <- obs[,c("longitude", "latitude")]
wkt <- sf::st_crs(4326)[[2]]
proj4string(obs) <- sp::CRS(wkt) #CRS("+init=epsg:4326")

nObs <- nrow(obs)

# overlay the data with the grid
ObsInGridList <- over(grid, obs, returnList=TRUE)
wNonEmpty <- unname( which( unlist(lapply(ObsInGridList, nrow)) != 0) )
if(length(wNonEmpty)==0) message("Observations don't overlap any grid cell.")
```

The result `ObsInGridList` is a list object with a subset of the data on each grid. Now summarise occurrences within grid cells:

```
# check n the total number of observations
sum(unlist(lapply(ObsInGridList, nrow)))
```

```
## [1] 93205
```

```

# apply a summary over the grid cells
nCells <- length(ObsInGridList)

res <- data.frame("nObs"=as.numeric(rep(NA,nCells)),
                 "nYears"=as.numeric(rep(NA,nCells)),
                 "nSpp"=as.numeric(rep(NA,nCells)),
                 row.names = row.names(grid),
                 stringsAsFactors = FALSE)

cols2use <- c("scientificName", "year")

dataRes <- lapply(ObsInGridList[wNonEmpty],
                 function(x){
                   x <- x[,cols2use]
                   colnames(x) <- c("scientificName", "year")
                   return(c("nObs" = length(x[, "scientificName"]),
                           "nYears" = length(unique(x[, "year"])),
                           "nSpp" = length(unique(x[, "scientificName"])))
                   )
                 })

dataRes <- as.data.frame(dplyr::bind_rows(dataRes, .id = "gridID"))

res[wNonEmpty,] <- dataRes[,-1]

resSp <- sp::SpatialPolygonsDataFrame(grid, res)

```

And finally plot the grid summary as a map:

```

palBW <- leaflet::colorNumeric(c("white", "navyblue"),
                              c(0, max(resSp@data$nSpp, na.rm = TRUE)),
                              na.color = "transparent")

oldpar <- par()
par(mar = c(1,1,0,0))
plot(resSp, col=palBW(resSp@data$nSpp), border = NA)
plot(swe_wgs84$Border, border=1, lwd=1, add=T)
legend("bottomleft",
      legend = round(seq(0, max(resSp@data$nSpp, na.rm = TRUE), length.out = 5)),
      col = palBW(seq(0, max(resSp@data$nSpp, na.rm = TRUE), length.out = 5)),
      title = "Number of \nspecies", pch = 15, bty="n")
par(oldpar)

```

We can go further by gathering the observations by latitude.

```

library(dplyr)
library(tidyr)
xgridded <- xf$data %>%
  mutate(longitude = round(longitude * 4)/4,
         latitude = round(latitude * 4)/4) %>%
  group_by(longitude,latitude) %>%
  ## subset to vars of interest

```

```

select(longitude, latitude, species) %>%
  ## take one row per cell per species (presence)
  distinct() %>%
  ## calculate species richness
  mutate(richness=n()) %>%
  ## convert to wide format (sites by species)
  mutate(present=1) %>%
  do(tidyr::pivot_wider(data=.,
                        names_from=species,
                        values_from=present,
                        values_fill=0)) %>%

  ungroup()
## where a species was not present, it will have NA: convert these to 0
sppcols <- setdiff(names(xgridded),
                   c("longitude", "latitude", "richness"))
xgridded <- xgridded %>%
  mutate_at(sppcols, function(z) ifelse(is.na(z), 0, z))

```

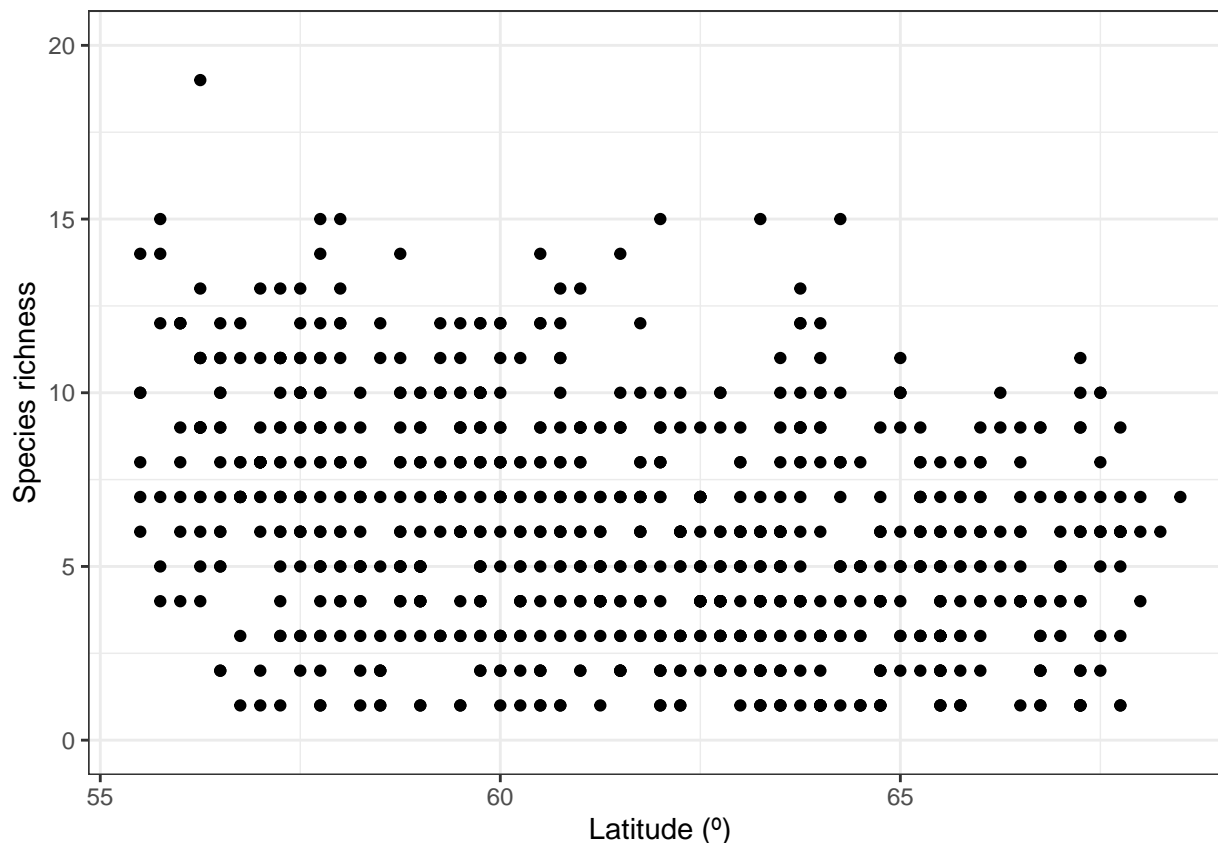
And plot it accordingly

```

library(ggplot2)

ggplot(xgridded, aes(latitude, richness)) +
  labs(x = "Latitude (°)",
       y = "Species richness") +
  lims(y = c(0,20)) +
  geom_point() +
  theme_bw()

```



## 2 Example with opportunistic data on Dragonflies

In this example we are interested in exploring opportunistically collected data from the Swedish citizen science observation data portal - Artportalen.

### 2.1 Name searching

To begin, we want be sure there is an unequivocal way to find the species within the order Odonata and nothing else, so let's search for it:

```
sx <- search_fulltext("odonata")
sx$data[, c("guid", "scientificName", "rank", "occurrenceCount")]
```

##	guid	scientificName	rank	occurrenceCount
## 1	9829523	Odonata associated gemycircularvirus 1	species	0
## 2	10072832	Odonata associated gemycircularvirus 2	species	0
## 3	8062407	Bdellodes odonata Wallace & Mahon, 1976	species	0
## 4	789	Odonata	order	207680
## 5	7367071	Ramalina fastigiata var. odonata Hue	variety	0

We quickly see there that other taxonomic definitions appear too, but there is only one order. Let's refine the search. To know which search fields we can use to filter the search we use the function `sbdi_fields(fields_type = "general")`. The search field we are looking for is "order\_s".

```

sx <- search_fulltext(fq="order_s:Odonata", page_size = 10)
sx$data[, c("scientificName", "rank", "occurrenceCount")]

```

```

##      guid                scientificName    rank occurrenceCount
## 1  9829523  Odonata associated gemycircularvirus 1 species          0
## 2 10072832  Odonata associated gemycircularvirus 2 species          0
## 3  8062407  Bdellodes odonata Wallace & Mahon, 1976 species          0
## 4      789                Odonata    order        207680
## 5  7367071  Ramalina fastigiata var. odonata Hue variety          0

```

Now we can download the taxonomic data (note that the search is case-sensitive):

```

tx <- taxinfo_download("order_s:Odonata",
  fields = c("guid", "order_s", "genus_s", "specificEpithet_s",
    "scientificName", "canonicalName_s", "rank"),
  verbose = FALSE)
tx <- tx[tx$rank == "species" & tx$genusS != "",] ## restrict to species and not hybrids

```

Then you can save tx as the complete species list for later use.

## 2.2 Filter the search to get the observations

As usual we start by searching for the data resource we are interested in using the function `pick_filter()`. This is an interactive query guiding you through the many resources available to filtering your query (data resources, spatial layers, and curated species lists).

```

# follow the instructions
fq_str <- pick_filter("resource")

```

Follow the instruction. Your choices here would have been “in3” → “dr5”. Your variable `fq_str` will now contain a string “data\_resource\_uid:dr5”.

We only need data from 2000 to 2010

```

y1 <- 2000
y2 <- 2010
fq_str <- c(fq_str, paste0("year:[", y1, " TO ", y2, "]"))
# Note the square brackets are hard limits

```

We also need to filter spatially for Southern Sweden ([Götaland](#)).

Vector spatial layers (eg. polygons) can be imported in a number of different ways. SBDI APIs take as search input polygons in the s.k. WKT [Well Known Text](#) format. So the first step is to load a vector layer and transform it into a WKT string. You could instead use the data we kindly provided in the SBDI4R package `data("swe")`.

```

data("swe")
wGotaland <- swe$Counties$LnNamn %in% c("Blekinge", "Gotlands", "Hallands",
  "Jönköpings", "Kalmar", "Kronobergs",
  "Östergötlands", "Skåne", "Västra Götalands")
gotaland_c <- swe$Counties[wGotaland,]

```

Unfortunately, in this instance this gives a WKT string that is too long and won't be accepted by the web service. Also, the shapefile we just got is projected in the coordinate system SWEREF99 TM, and the web service only accepts coordinates in a geodesic coordinate system WGS84. Instead, let's construct the WKT string directly, which gives us a little more control over its format:

```
gotaland_c <- sf::as_Spatial(
  sf::st_transform(
    sf::st_as_sf(gotaland_c),
    crs = sf::st_crs(4326)$wkt) )

gotaland <- rgeos::gUnaryUnion(gotaland_c)

# extract the polygons coordinates
nPol <- length(gotaland@polygons[[1]]@Polygons)
lonlat <- list()
for(p in seq(nPol)){
  lonlat[[p]] <- gotaland@polygons[[1]]@Polygons[[p]]@coords
}
lonlat <- do.call(rbind, lonlat)

# create a convex hull of the polygon to reduce the length of the WKT string
gotaland_ch <- chull(lonlat)
lonlat <- lonlat[c(gotaland_ch, gotaland_ch[1]), ]

# create WKT string
# first join each lon-lat coordinate pair
wkt_temp <- apply(lonlat, 1, function(z) paste(round(z,4), collapse=" "))
# now build the WKT string
wkt <- paste("MULTIPOLYGON((( ", paste(wkt_temp, collapse=","), ")))", sep="")
# NOTE: as of today, the SBDI APIs will only work properly if the polygon is
# submitted as a MULTIPOLYGON
```

We download the observations using the command `occurrences()`, but be aware that the search fields may not be the same as those use to search for taxa. We therefore recommend using the function `sbdi_fields("occurrence")`. Here see that the field we need this time is "order".

```
xf <- SBDI4R::occurrences(taxon = "order:Odonata",
  fq = fq_str,
  wkt = wkt,
  extra = "collector",
  email = "sbdi4r-test@biodiversitydata.se",
  download_reason_id = 10)
```

We have now downloaded the data locally and depending on your configuration this will be cached on your computer. However, as the search and download could take long, we recommend to anyhow save the data locally.

```
save(xf, file = "an_appropriated_name.rdata")
load(file = "an_appropriated_name.rdata")
```

## 2.3 Quality and fit-for-use check

Before we can use the observation records we need to know if the observation effort has varied over time and in space. For this we need to define field visits i.e. occasions at which an observer has sampled observations,

and reconstruct them (that is, assign each observation a visitUID). We do this using the package [BIRDS](#). We even want the data to be summarized over a grid of 25 km (provided by the SBDI4R package). The following functions will perform many different summaries at the same time. Please refer to the package documentation for more detail.

```
library(BIRDS)
OB <- organiseBirds(xf$data, sppCol = "species" ,
  # We only want observations identified at the species level
  taxonRankCol = "rank", taxonRank = "species",
  # the visits are defined by collector and named locality
  idCols = c("locality", "collector"),
  timeCols = c("year", "month", "day"),
  xyCols = c("longitude", "latitude") )

## 252 observations did not match with the specified taxon rank and were removed.

# We don't need the whole grid, just the piece that overlaps our searching polygon
gotaland_grid25 <- raster::intersect(gotaland, Sweden_Grid_25km_Wgs84)

# This is another way of doing it.
# gotaland_grid25 <- gIntersection(gotaland,
#                               spTransform(Sweden_Grid_25km_Wgs84,
#                               CRSobj = CRS(sf::st_crs(4326)$wkt)))

SB <- summariseBirds(OB, grid = gotaland_grid25, spillOver = "unique")
```

## 1664 observations did not overlap with the grid and will be discarded.

## 0.009 % of the visits spill over neighbouring grid cells.

Once summarized, we can see over space and for a few selected year how was the sampling effort (in this case number of observations) distributed.

```
maxC <- max(SB$spatial@data$nObs, na.rm = TRUE)
palBW <- leaflet::colorNumeric(c("white", "navyblue"),
  c(0, maxC),
  na.color = "transparent")

oldpar <- par()
par(mar = c(4,0,4,0), mfrow=c(1,3))
plot(SB$spatial, col=palBW(SB$spatial@data$nObs),
  border = "grey", main="All years") ## with palette
legend("topleft", inset = c(0,0.05),
  legend = round(seq(0, maxC, length.out = 5)),
  col = palBW(seq(0, maxC, length.out = 5)),
  title = "Number of \nobservations", pch = 15, bty="n")

## or export other combinations, e.g. one map per observed year
yearlySp <- exportBirds(SB,
  dimension = "spatial",
  timeRes = "yearly",
  variable = "nObs",
  method = "sum")
```

```

maxC <- max(yearlySp@data$'2005', na.rm = TRUE)
palBW <- leaflet::colorNumeric(c("white", "navyblue"),
                              c(0, maxC),
                              na.color = "transparent")

plot(yearlySp["2005"], col=palBW(yearlySp@data$'2005'),
     border = "grey",main="2005")
legend("topleft", inset = c(0,0.05),
      legend = round(seq(0, maxC, length.out = 5)),
      col = palBW(seq(0, maxC, length.out = 5)),
      border = "grey",
      title = "Number of \nobservations", pch = 15, bty="n")

maxC <- max(yearlySp@data$'2010', na.rm = TRUE)
palBW <- leaflet::colorNumeric(c("white", "navyblue"),
                              c(0, maxC),
                              na.color = "transparent")

plot(yearlySp["2010"], col=palBW(yearlySp@data$'2010'),
     border = "grey",main="2010")
legend("topleft", inset = c(0,0.05),
      legend = round(seq(0, maxC, length.out = 5)),
      col = palBW(seq(0, maxC, length.out = 5)),
      border = "grey",
      title = "Number of \nobservations", pch = 15, bty="n")
par(oldpar)

```

There are other ways to plot spatial data, here there is another example this time using the package `sf` instead of `sp` and using number of visits as the measure for sampling effort.

```

library(sf)
library(cowplot)
library(ggplot2)
library(colorRamps)
library(gridExtra)

spatial_sf <- st_as_sf(SB$spatial)

vis <- ggplot(data = spatial_sf, aes( fill = nVis)) +
  geom_sf() +
  ggtitle("Visits") +
  scale_fill_gradient(low = "#56B1F7",
                     high = "#132B43",
                     na.value = NA) +
  theme(plot.margin = margin(1, 1, 1, 1, "pt")) +
  theme_cowplot()

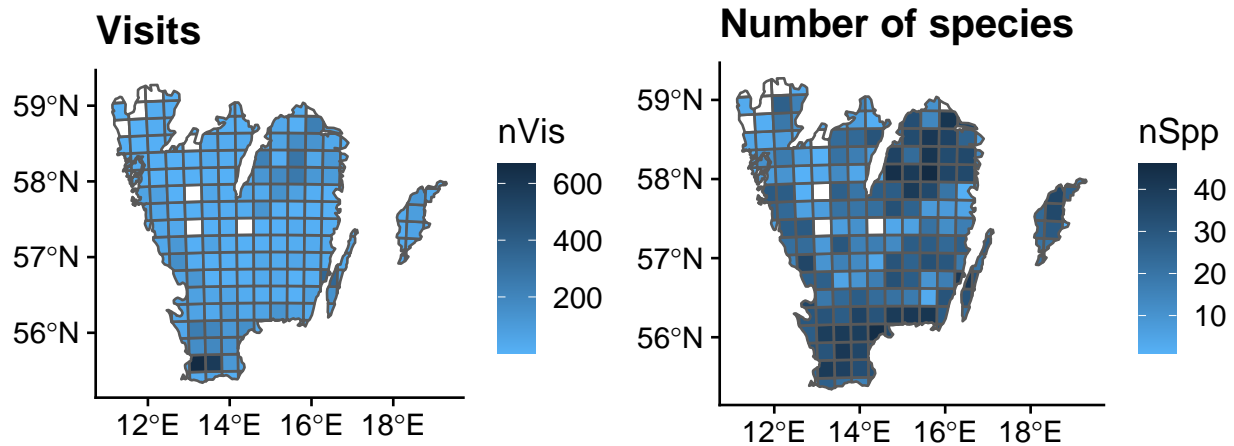
spp <- ggplot(data = spatial_sf ,aes( fill = nSpp))+
  geom_sf()+
  ggtitle("Number of species")+
  scale_fill_gradient(low = "#56B1F7",
                     high = "#132B43",
                     na.value = NA) +

```



```
theme(plot.margin = margin(1, 1, 1, 1, "pt")) +
theme_cowplot()

grid.arrange(vis, spp, ncol=2)
```



How has observation effort (frequency of visits) varied over time and space? – 1) show maps as in Example 7 (all years, year 2000, 2002, 2004, 2006, 2008, 2010), 2) make also a time line plot with no. visits against years, no. of gridcells with visits against years.

we see that SB contains an element called `SB$temporal` that contains a daily time series with time specific rows when there is information. `xts` also supports time, but dating below day resolution is not yet implemented in the BIRDS package.

```
sb.xts <- SB$temporal
dim(sb.xts)
```

```
## [1] 1118    3
```

```
head(sb.xts, 20)
```

```
##           nObs nVis nSpp
## 2000-03-24     1     1     1
## 2000-04-05     4     3     3
## 2000-04-06    11     6     3
## 2000-04-10     1     1     1
## 2000-04-12     3     3     1
## 2000-04-13     8     5     2
```

```
## 2000-04-20    1    1    1
## 2000-04-21    5    4    2
## 2000-04-23    5    2    3
## 2000-04-24    7    5    2
## 2000-04-26    1    1    1
## 2000-04-27    7    6    3
## 2000-04-28    9    7    3
## 2000-04-29    6    3    3
## 2000-05-27    1    1    1
## 2000-06-03    1    1    1
## 2000-07-30    1    1    1
## 2000-08-03    1    1    1
## 2000-08-05    5    2    5
## 2000-08-06    3    1    3
```

Sub-setting is convenient in `xts` as you can do it with its dates and with a `/` for a range of dates.

```
sb.xts["2010-09-07"] #a specific day
```

```
##           nObs nVis nSpp
## 2010-09-07     9     7     5
```

```
sb.xts["2010-09-01/2010-09-15"] #for a period
```

```
##           nObs nVis nSpp
## 2010-09-01    38    15    14
## 2010-09-02    26    12    12
## 2010-09-03    20     9    10
## 2010-09-04    63    19    18
## 2010-09-05    71    25    12
## 2010-09-06    16     4     9
## 2010-09-07     9     7     5
## 2010-09-08    13     6     8
## 2010-09-09    32    12    14
## 2010-09-10     1     1     1
## 2010-09-11    15     8     8
## 2010-09-12    15     7     8
## 2010-09-13    14     5     9
## 2010-09-14     1     1     1
## 2010-09-15     3     3     2
```

```
sb.xts["2010-09"] #a specific month
```

```
##           nObs nVis nSpp
## 2010-09-01    38    15    14
## 2010-09-02    26    12    12
## 2010-09-03    20     9    10
## 2010-09-04    63    19    18
## 2010-09-05    71    25    12
## 2010-09-06    16     4     9
## 2010-09-07     9     7     5
```

```
## 2010-09-08 13 6 8
## 2010-09-09 32 12 14
## 2010-09-10 1 1 1
## 2010-09-11 15 8 8
## 2010-09-12 15 7 8
## 2010-09-13 14 5 9
## 2010-09-14 1 1 1
## 2010-09-15 3 3 2
## 2010-09-17 3 2 3
## 2010-09-18 9 5 5
## 2010-09-19 12 7 5
## 2010-09-21 3 2 3
## 2010-09-22 4 4 2
## 2010-09-23 3 3 2
## 2010-09-24 10 5 5
## 2010-09-25 6 3 6
## 2010-09-26 7 6 2
## 2010-09-28 2 2 2
## 2010-09-29 5 3 4
## 2010-09-30 2 2 2
```

The package `xts` has several tools for converting to different periods. Here we will use `to.monthly`. This provides, the first, min, max, and last of the data. We can plot the daily maximum number of observations. The plot command with an `xts` object provides a TON of features. This makes it fairly easy to customize your plots. Read more in `?plot.xts`.

```
library(xts)
obs.m <- to.monthly(sb.xts$nObs)
obs.m["2007-04"]
```

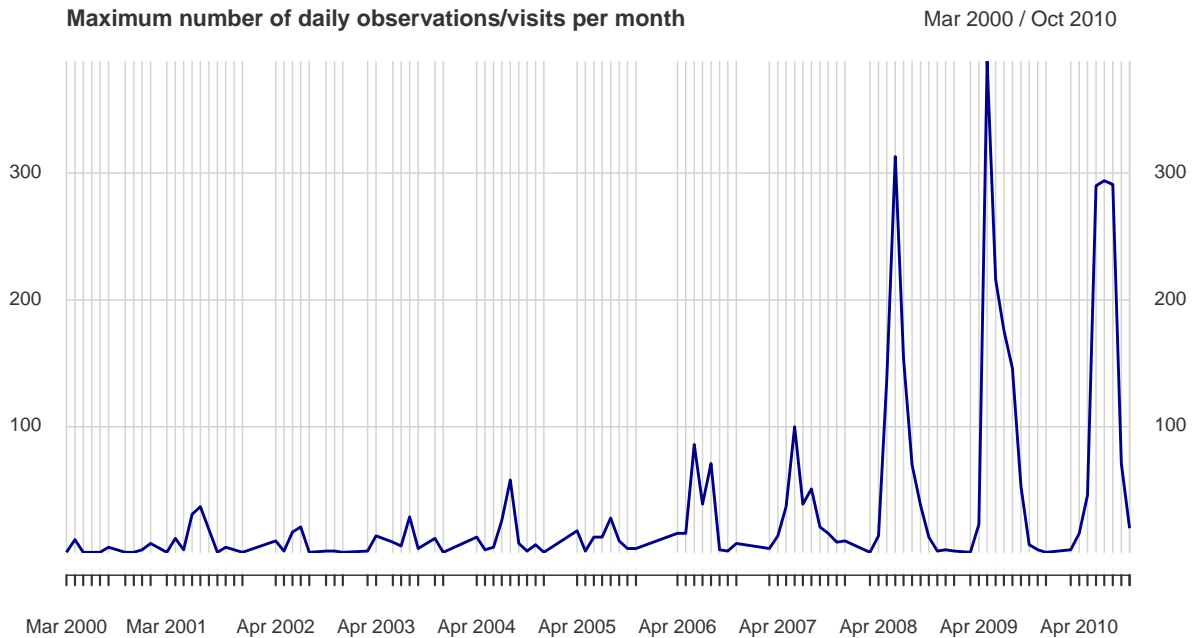
```
##          sb.xts$nObs.Open sb.xts$nObs.High sb.xts$nObs.Low sb.xts$nObs.Close
## Apr 2007                7              14                1                2
```

```
sb.xts["2007-04"]
```

```
##          nObs nVis nSpp
## 2007-04-02    7    4    4
## 2007-04-11   14    7    3
## 2007-04-12    8    6    4
## 2007-04-13    1    1    1
## 2007-04-15    1    1    1
## 2007-04-17    6    4    3
## 2007-04-18    1    1    1
## 2007-04-21    1    1    1
## 2007-04-23    1    1    1
## 2007-04-27   11    6    4
## 2007-04-28    4    4    3
## 2007-04-30    2    2    2
```

```
plot(obs.m["2000/2010",2],
     col = "darkblue",
     grid.ticks.on = "month",
```

```
major.ticks = "month",
grid.col = "lightgrey",
main = "Maximum number of daily observations/visits per month")
```



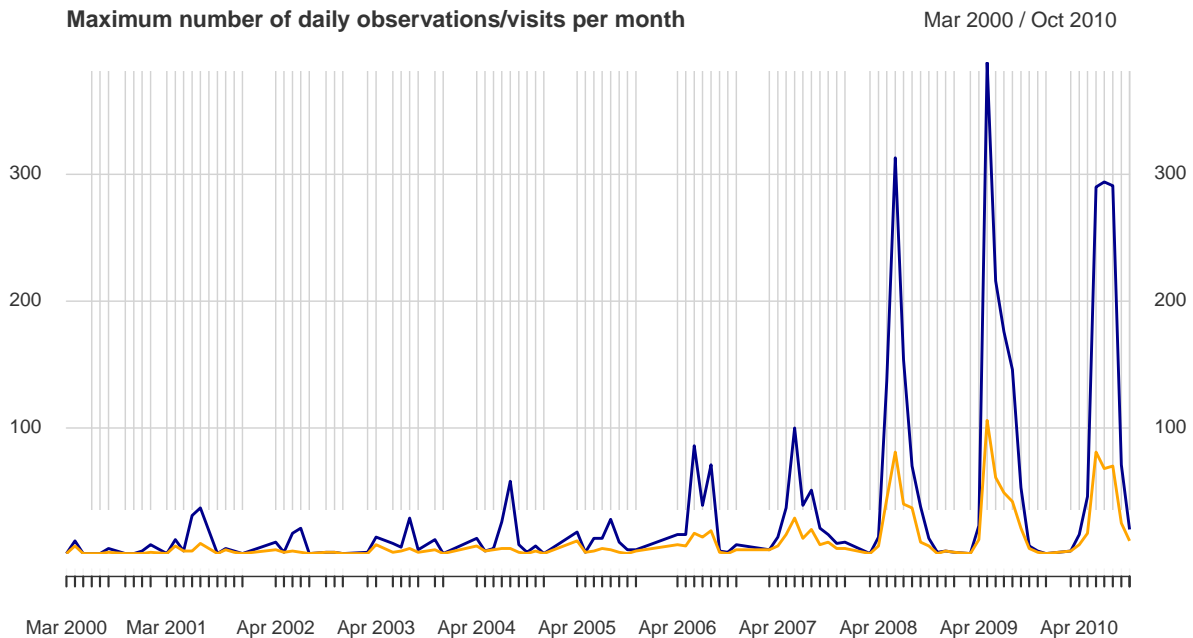
```
vis.m <- to.monthly(sb.xts$nVis)
vis.m["2007-04"]
```

```
##          sb.xts$nVis.Open sb.xts$nVis.High sb.xts$nVis.Low sb.xts$nVis.Close
## Apr 2007                4                7                1                2
```

```
sb.xts["2007-04"]
```

```
##          nObs nVis nSpp
## 2007-04-02    7    4    4
## 2007-04-11   14    7    3
## 2007-04-12    8    6    4
## 2007-04-13    1    1    1
## 2007-04-15    1    1    1
## 2007-04-17    6    4    3
## 2007-04-18    1    1    1
## 2007-04-21    1    1    1
## 2007-04-23    1    1    1
## 2007-04-27   11    6    4
## 2007-04-28    4    4    3
## 2007-04-30    2    2    2
```

```
lines(vis.m["2000/2010",2], col = "orange", lwd=2)
```



```
# major.ticks = "month", grid.col = "lightgrey",
# main = "Maximum number of daily visits per month")
```

We can now look at some particular species and ask whether this has changed in occurrence over time: Plot no. records of species x and no. visits all species over years (we simply explore by comparing records for a species with no visits, can assume that species has increased of stronger positive trend than for no. visits)

Plot no. gridcells with visits for species x and no. gridcells with visits for all species over years (we simply explore by comparing records for a species with no visits, can assume that species has increased of stronger positive trend than for no. visits) (species x: Tvåfläckad trollslända *Epitheca bimaculata*)