Point count data analysis: How to violate assumptions and get away with it

Peter Solymos

2019-06-21

Contents

List of Tables

List of Figures

Preface

This book provides material for the workshop *Analysis of point-count data in the presence of variable survey methodologies and detection error* at the AOS 2019 conference by Peter Solymos.

The book and related materials in this repository is the basis of a full day workshop (8 hours long with 3 breaks).

Prior exposure to R language is necessary (i.e. basic R object types and their manipulation, such as arrays, data frames, indexing) because this is not covered as part of the course. Check this intro.

About the book and the course

You'll learn

- how to analyze your point count data when it combines different methodologies/protocols/technologies,
- how to violate assumptions and get away with it.

This book/course is aimed towards ornithologists analyzing field observations, who are often faced by data heterogeneities due to field sampling protocols changing from one project to another, or through time over the lifespan of projects, or trying to combine 'legacy' data sets with new data collected by recording units. Such heterogeneities can bias analyses when data sets are integrated inadequately, or can lead to information loss when filtered and standardized to common standards. Accounting for these issues is important for better inference regarding status and trend of bird species and communities.

Analysts of such 'messy' data sets need to feel comfortable with manipulating the data, need a full understanding the mechanics of the models being used (i.e. critically interpreting the results and acknowledging assumptions and limitations), and should be able to make informed choices when faced with methodological challenges.

The course emphasizes critical thinking and active learning. Participants will be asked to take part in the analysis: first hand analytics experience from start to finish. We will use publicly available data sets to demonstrate the data manipulation and analysis. We will use freely available and open-source R packages.

The expected outcome of the course is a solid foundation for further professional development via increased confidence in applying these methods for field observations.

About the author

Peter Solymos is an ecologist (molluscs, birds), he is pretty good at stats (modeling, detectability, data cloning, multivariate), an R programmer (vegan, detect, ResourceSelection, pbapply), sometimes he teaches (like the contents of this book).

Installing R and RStudio

Follow the instructions at the R website to download and install the most up-to-date base R version suitable for your operating system (the latest R version at the time of writing these instructions is 3.6.0).

Having RStudio is not absolutely necessary, but some of our course material will follow a syntax that is close to RStudio's R markdown notation, so having RStudio will make our life easier. RStudio is also available for different operating systems. Pick the open source desktop edition from here (the latest RStudio Desktop version at the time of writing these instructions is 1.2.1335).

Installing required packages

```
pkgs <- c("bookdown", "detect", "devtools", "dismo",
  "Distance", "forecast",
  "glmnet", "gbm", "intrval", "knitr", "lme4", "maptools", "mefa4",
  "mgcv", "MuMIn", "opticut", "partykit", "pscl", "raster",
  "ResourceSelection", "shiny", "sp", "unmarked", "visreg")
to_inst <- setdiff(pkgs, rownames(installed.packages()))
if (length(to_inst))
  install.packages(to_inst, repos="https://cloud.r-project.org/")
devtools::install_github("psolymos/bSims")
devtools::install_github("psolymos/QPAD")
devtools::install_github("borealbirds/paired")
devtools::install_github("borealbirds/lhreg")
still_missing <- setdiff(c(pkgs, "bSims", "paired", "lhreg", "QPAD"),
  rownames(installed.packages()))
if (length(still_missing)) {
  cat("The following packages could not be installed:\n",
    paste("\t-", pkgs, collapse="\n"), "\n")
} else {
  cat("You are all set! See you at the workshop.\n")
}
```

Here is a preprint version of Norman Matloff's *The Art of R Programming* book: http://heather.cs.ucdavis.edu/~matloff/132/NSPpart.pdf. Check out Chapters 1–6 if you want to brush up your R skills.

Installing the book

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")

## clean up
```

```
bookdown::clean_book(TRUE)
## rendering the book
bookdown::render_book('index.Rmd', 'bookdown::gitbook')
bookdown::render_book('index.Rmd', 'bookdown::pdf_book')
bookdown::render_book('index.Rmd', 'bookdown::epub_book')
```

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

How this works


This is an exercise.


This is a note.


This is a warning.

Acknowledgments

List here all the wonderful folks who helped with this book.

Chapter 1

Introduction

All assumptions are violated, but some are more than others

A comparison of apples and oranges occurs when two items or groups of items are compared that cannot be practically compared (Wikipedia). The way we measure things can have a big impact on the outcome of that measurement. For example, you might say that "I saw 5 robins walking down the road", while I might say that "I only saw one robin while sitting on my porch". Who say more robins? If looking at only the numeric results, you saw more robins than me. But this seems like an apples to oranges comparison.

To compare apples to apples, we need to agree on a comparable measurement scheme, or at least figure out how does *effort* affect the observations.

Effort in our example can depend on, e.g. the *area* of the physical space searched, the amount of *time* spent, etc. The outcome might further affected by weather, time of year, time of day, location, experience and skill level of the observer.

All these factors can affect the observed count. Which brings us to the definition of a *point count*: a trained observer records all the birds seen and heard from a point count station for a set period of time within a defined distance radius.

Point count duration and distance have profound effect on the counts, as shown in Figure 1.1 showing that a 10-min unlimited distance count is roughly 300% increased compared to 3-min 50-m counts (averaged across 54 species

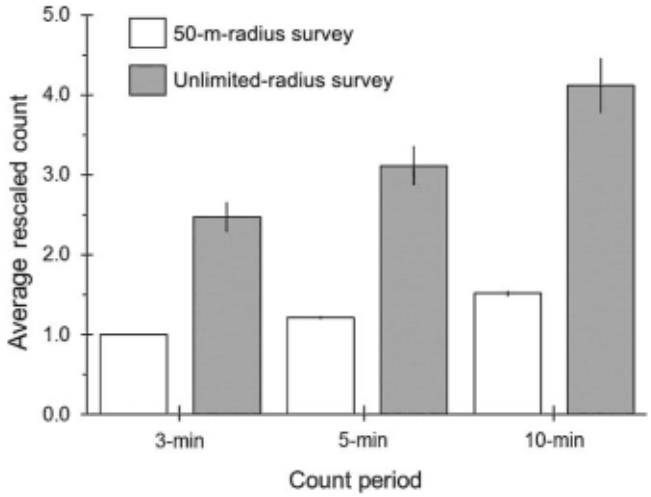


Figure 1.1: Effects of duration and distance on mean counts, from [@matsuoka2014].

of boreal songbirds, (Matsuoka et al., 2014)).

Point counts are commonly used to answer questions like:

- How many? (Abundance, density, population size)
- Is this location part of the range? (0/1)
- How is abundance changing in space? (Distribution)
- How is abundance changing in time? (Trend)
- What is the effect of a treatment on abundance?

1.1 Design-based approaches

Standards and recommendations can maximize efficiency in the numbers of birds and species counted, minimize extraneous variability in the counts.

But programs started to deviate from standards: *"For example, only 3% of 196,000 point counts conducted during the period 1992–2011 across Alaska and Canada followed the standards recommended for the count period and count radius"* ((Matsuoka et al., 2014)). Figure 1.2 show how point count protocol varies across the boreal region of North America.

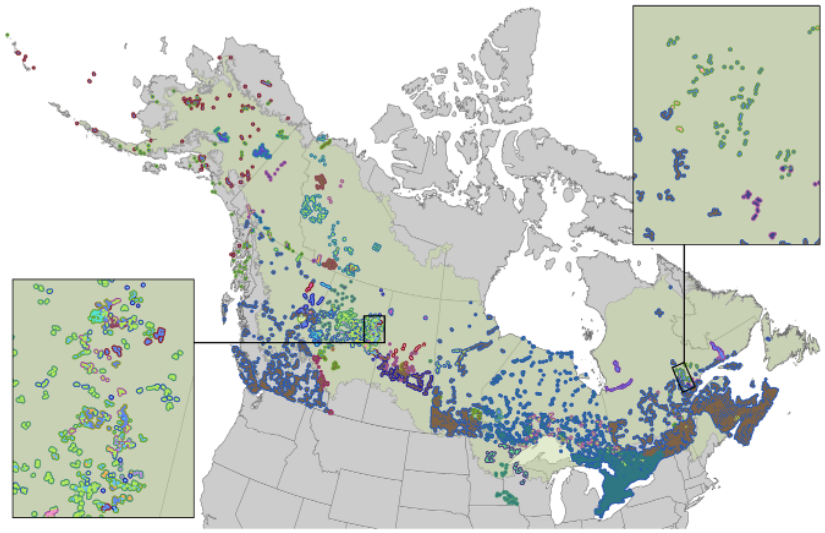


Figure 1.2: Survey methodology variation (colors) among contributed projects in the Boreal Avian Modelling (BAM) data base, from [@barker2015].

Exercise

In what regard can protocols differ?
What might drive protocol variation among projects?
Why have we abandoned following protocols?

1.2 Model-based approaches

Detection probabilities might vary even with fixed effort (we'll cover this more later), and programs might have their own goals and constraints (access, training, etc). These constraints would make it almost impossible, and potentially costly to set up very specific standards.

Labour intensive methods for unmarked populations have come to the forefront, and computing power of personal computers opened the door for model-based approaches, that can accomodate more variation given enough information in the observed data. These methods often rely on ancillary

information and often some sort of replication.

Some of the commonly used model-based approaches are:

- double observer (Nichols et al. 2000),
- distance sampling (Buckland et al. 2001),
- removal sampling (Farnsworth et al. 2002),
- multiple visit occupancy (MacKenzie et al. 2002),
- multiple visit abundance (Royle 2004).

Models come with assumptions, such as:

- population is closed during multiple visits,
- observers are independent,
- all individuals emit cues with identical rates,
- spatial distribution of individuals is uniform,
- etc.

Although assumptions are everywhere, we are really good at ignoring and violating them.

Exercise
Can you mention some assumptions from everyday life?
Can you explain why we neglect/violate assumptions in these situations?

Assumptions are violated, because we seek simplicity. The main question we have to ask: *does it matter in practice* if we violate the assumptions?

1.3 Our approach

In this book and course, we will critically evaluate common assumptions made when analyzing point count data using the following approach:

1. we will introduce a concept,
2. understand how we can infer it from data,
3. then we recreate the situation *in silico*,
4. and see how the outcome changes as we make different assumptions.

It is guaranteed that we will violate every assumption we make. To get away with it, we need to understand how much is too much, and whether it has an impact in practice. If there is a practical consequence, we will look at ways to minimize that effects – so that we can safely ignore the assumption.

Chapter 2

Organizing and Processing Point Count Data

All data are messy, but some are missing

2.1 Introduction

It is often called *data processing*, *data munging*, *data wrangling*, *data cleaning*. None of these expressions capture the dread associated with the actual activity. Luckily, this book and course is not about data manipulation, and we can gladly skip these steps. But keep in mind, in real life, you'll often have to worry about at least four things that can go wrong:

1. space (e.g. wrong UTM zones, errors),
2. time (ISO format please),
3. taxonomy (unknowns, mis-identifications),
4. something else (if there were no errors, check again).

This chapter sets you up for the required data skills for the rest of the book/course.

2.2 Prerequisites

```
library(mefa4)                  # data manipulation
library(raster)                 # reading raster files
library(sp)                     # spatial data manipulation
load("_data/josm/josm.rda")     # load bird data
rr <- stack("_data/josm/landcover-hfi2016.grd") # rasters
```

2.3 R basics

This short document is intended to help you brush up your R skills. If you feel that these R basics are not very familiar, I suggest to take a look at some introductory R books, sich as this preprint version of Norman Matloff's *The Art of R Programming* book: http://heather.cs.ucdavis.edu/~matloff/132/NSPpart.pdf, check out Chapters 1–6.

R is a great calculator:

```
1 + 2
```

Assign a value and print an object using `=` or `<-` (preferred in this book):

```
(x = 2) # shorthand for print
print(x)
x == 2 # logical operator, not assignment
y <- x + 0.5
y # another way to print
```

Logical operators come handy:

```
x == y # equal
x != y # not eaqual
x < y # smaller than
x >= y # greater than or equal
```

Vectors and sequences are created most often by the functions `c`, `:`, `seq`, and `rep`:

```
x <- c(1, 2, 3)
x
1:3
seq(1, 3, by = 1)

rep(1, 5)
rep(1:2, 5)
rep(1:2, each = 5)
```

When doing operations with vectors remember that values of the shorter object are recycled:

```
x + 0.5
x * c(10, 11, 12, 13)
```

Indexing and ordering vectors is a a fundamental skill:

```
x[1]
x[c(1, 1, 1)] # a way of repeatig values
x[1:2]
x[x != 2]
x[x == 2]
x[x > 1 & x < 3]
order(x, decreasing=TRUE)
x[order(x, decreasing=TRUE)]
rev(x) # reverse
```

See how `NA` values can influence sorting character vectors:

```
z <- c("b", "a", "c", NA)
z[z == "a"]
z[!is.na(z) & z == "a"]
z[is.na(z) | z == "a"]
is.na(z)
which(is.na(z))
sort(z)
sort(z, na.last=TRUE)
```

There are a few special values:

```
as.numeric(c("1", "a")) # NA: not available (missing or invalid)
0/0 # NaN: not a number
1/0 # Inf
-1/0 # -Inf
```

Matrices and arrays are vectors with dimensions, elements are in same mode:

```
(m <- matrix(1:12, 4, 3))
matrix(1:12, 4, 3, byrow=TRUE)

array(1:12, c(2, 2, 3))
```

Many objects have attributes:

```
dim(m)
dim(m) <- NULL
m
dim(m) <- c(4, 3)
m
dimnames(m) <- list(letters[1:4], LETTERS[1:3])
m
attributes(m)
```

Matrice and indices:

```
m[1:2,]
m[1,2]
m[,2]
m[,2,drop=FALSE]
m[2]

m[rownames(m) == "c",]
m[rownames(m) != "c",]
m[rownames(m) %in% c("a", "c", "e"),]
m[!(rownames(m) %in% c("a", "c", "e")),]
```

Lists and indexing:

```
l <- list(m = m, x = x, z = z)
l
```

```
l$ddd <- sqrt(l$x)
l[2:3]
l[["ddd"]]
```

Data frames are often required for statistical modeling. A data frame is a list where length of elements match and elements can be in different mode.

```
d <- data.frame(x = x, sqrt_x = sqrt(x))
d
```

Inspect structure of R objects:

```
str(x)
str(z)
str(m)
str(l)
str(d)
str(as.data.frame(m))
str(as.list(d))
```

Get summaries of these objects:

```
summary(x)
summary(z)
summary(m)
summary(l)
summary(d)
```

2.4 JOSM data set

The data is based on the project *Cause-Effect Monitoring Migratory Landbirds at Regional Scales: understand how boreal songbirds are affected by human activity in the oil sands area* (2.1 and 2.2, (Mahon et al., 2016)). JOSM stands for Joint Oil Sands Monitoring. Look at the source code in the `_data/josm` directory of the book if you are interested in data processing details. We skip that for now.

Surveys were spatially replicated because:

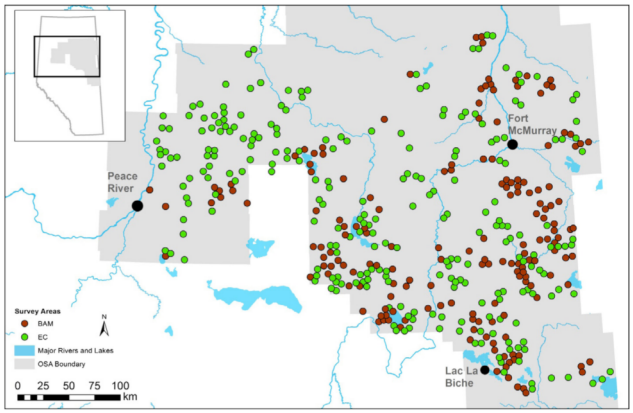


Figure 2.1: JOSM bird data survey locations from [@mahon2016].

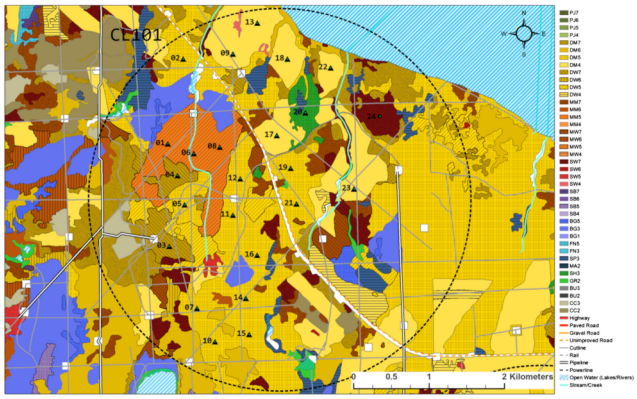


Figure 2.2: Survey area boundary, habitat types and human footprint mapping [@mahon2016].

- we want to make inferences about a population,
- full census is out of reach,
- thus we take a sample of the population
- that is representative and random.
- Ideally, sample size should be as large as possible,
- it reduces variability and
- increases statistical power.

Survey locations were pucked based on various criteria:

- stratification (land cover),
- gradients (disturbance levels),
- random location (control for unmeasured effects),
- take into account historical surveys (avoid, or revisit),
- access, cost (clusters).

The `josm` obejct is a list with 3 elements:

- `surveys`: data frame with survey specific information,
- `species`: lookup table for species,
- `counts`: individual counts by survey and species.

```
names(josm)
```

```
## [1] "surveys" "species" "counts"
```

Species info: species codes, common and scientific names. The table could also contain taxonomic, trait, etc. information as well.

```
head(josm$species)
```

At the survey level, we have coordinates, date/time info, variables capturing survey conditions, and land cover info extracted from 1 km^2 resolution rasters.

```
colnames(josm$surveys)
```

```
##  [1] "SiteID"        "SurveyArea"    "Longitude"     "Latitude"
##  [5] "Date"          "StationID"     "ObserverID"    "TimeStart"
##  [9] "VisitID"       "WindStart"     "PrecipStart"   "TempStart"
## [13] "CloudStart"    "WindEnd"       "PrecipEnd"     "TempEnd"
## [17] "CloudEnd"      "TimeFin"       "Noise"         "OvernightRain"
## [21] "DateTime"      "SunRiseTime"   "SunRiseFrac"   "TSSR"
```

```
## [25] "OrdinalDay"    "DAY"           "Open"          "Water"
## [29] "Agr"           "UrbInd"        "SoftLin"       "Roads"
## [33] "Decid"         "OpenWet"       "Conif"         "ConifWet"
```

The count table contains one row for each unique individual of a species (`SpeciesID` links to the species lookup table) observed during a survey (`StationID` links to the survey attribute table). Check the data dictionary in `_data/josm` folder for a detailed explanation of each column.

```
str(josm$counts)
```

```
## 'data.frame':    52372 obs. of  18 variables:
##  $ ObservationID: Factor w/ 57024 levels "CL10102-130622-001",..: 1 2 3 4 5
##  $ SiteID       : Factor w/ 4569 levels "CL10102","CL10106",..: 1 1 1 1 1 1
##  $ StationID    : Factor w/ 4569 levels "CL10102-1","CL10106-1",..: 1 1 1 1
##  $ TimeInterval : int  1 1 1 1 5 5 1 1 1 1 ...
##  $ Direction    : int  1 2 2 2 1 4 4 4 1 1 ...
##  $ Distance     : int  1 2 2 1 3 3 2 1 1 1 ...
##  $ DetectType1  : Factor w/ 3 levels "C","S","V": 2 2 2 2 1 1 2 2 2 2 ...
##  $ DetectType2  : Factor w/ 3 levels "C","S","V": NA NA NA NA NA NA NA NA N
##  $ DetectType3  : Factor w/ 3 levels "C","S","V": NA NA NA NA NA NA NA NA N
##  $ Sex          : Factor w/ 4 levels "F","M","P","U": 2 2 2 2 4 4 2 2 2 2 .
##  $ Age          : Factor w/ 6 levels "A","F","J","JUV",..: 1 1 1 1 1 1 1 1
##  $ Activity1    : Factor w/ 17 levels "BE","CF","CH",..: 5 5 5 5 NA NA NA 5
##  $ Activity2    : Factor w/ 17 levels "48","BE","CF",..: NA NA NA NA NA NA
##  $ Activity3    : Factor w/ 7 levels "CF","DC","DR",..: NA NA NA NA NA NA N
##  $ ActivityNote : Factor w/ 959 levels "AGITATED","AGITATED CALLING",..: NA
##  $ Dur          : Factor w/ 3 levels "0-3min","3-5min",..: 1 1 1 1 3 3 1 1
##  $ Dis          : Factor w/ 3 levels "0-50m","50-100m",..: 1 2 2 1 3 3 2 1
##  $ SpeciesID    : Factor w/ 150 levels "ALFL","AMBI",..: 107 95 95 107 46 4
```

2.5 Cross tabulating species counts

Take the following dummy data frame (long format):

```
(d <- data.frame(
  sample=factor(paste0("S", c(1,1,1,2,2)), paste0("S", 1:3)),
  species=c("BTNW", "OVEN", "CANG", "AMRO", "CANG"),
```

```
  abundance=c(1, 1, 2, 1, 1),
  behavior=rep(c("heard","seen"), c(4, 1))))
str(d)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ sample   : Factor w/ 3 levels "S1","S2","S3": 1 1 1 2 2
##  $ species  : Factor w/ 4 levels "AMRO","BTNW",..: 2 4 3 1 3
##  $ abundance: num  1 1 2 1 1
##  $ behavior : Factor w/ 2 levels "heard","seen": 1 1 1 1 2
```

We want to add up the `abundance`s for each sample (rows) and species (column):

```
(y <- Xtab(abundance ~ sample + species, d))
```

```
## 3 x 4 sparse Matrix of class "dgCMatrix"
##    AMRO BTNW CANG OVEN
## S1    .    1    2    1
## S2    1    .    1    .
## S3    .    .    .    .
```

`y` is a sparse matrix, that is a very compact representation:

```
object.size(d[,1:3])
```

```
## 2328 bytes
```

```
object.size(y)
```

```
## 2160 bytes
```

Notice that we have 3 rows, but `d$sample` did not have an `S3` value, but it was a level. We can drop such unused levels, but it is generally not recommended, and we need to be careful not to drop samples where no species was detected (this can happen quite often depending on timing of surveys)

```
Xtab(abundance ~ sample + species, d, drop.unused.levels = TRUE)
```

```
## 2 x 4 sparse Matrix of class "dgCMatrix"
##    AMRO BTNW CANG OVEN
## S1    .    1    2    1
## S2    1    .    1    .
```

A sparse matrix can be converted to ordinary matrix

```
as.matrix(y)
```

```
##    AMRO BTNW CANG OVEN
## S1    0    1    2    1
## S2    1    0    1    0
## S3    0    0    0    0
```

The nice thing about this cross tabulation is that we can finter the records without changing the structure (rows, columns) of the table:

```
Xtab(abundance ~ sample + species, d[d$behavior == "heard",])
```

```
## 3 x 4 sparse Matrix of class "dgCMatrix"
##    AMRO BTNW CANG OVEN
## S1    .    1    2    1
## S2    1    .    .    .
## S3    .    .    .    .
```

```
Xtab(abundance ~ sample + species, d[d$behavior == "seen",])
```

```
## 3 x 4 sparse Matrix of class "dgCMatrix"
##    AMRO BTNW CANG OVEN
## S1    .    .    .    .
## S2    .    .    1    .
## S3    .    .    .    .
```

Now let's do this for the real data. We have no abundance column, because each row stands for exactly one individual. We can add a column with 1's, or we can just count the number of rows by using only the right-hand-side of the formula in `Xtab`. `ytot` will be our total count matrix for now.

We also want to filter the records to contain only `S`ongs and `C`alls, without `V`visual detections:

```
table(josm$counts$DetectType1, useNA="always")
```

```
##
##     C     S     V  <NA>
##  9180 41808  1384     0
```

We use `SiteID` for row names, because only 1 station and visit was done at each site:

```
ytot <- Xtab(~ SiteID + SpeciesID, josm$counts[josm$counts$DetectType1 != "V",])
```

See how not storing 0's affect size compared to the long formar and an ordinary wide matrix

```
## 2-column data frame as reference
tmp <- as.numeric(object.size(
  josm$counts[josm$counts$DetectType1 != "V", c("StationID", "SpeciesID")]))
## spare matrix
as.numeric(object.size(ytot)) / tmp
```

```
## [1] 0.1366
```

```
## dense matrix
as.numeric(object.size(as.matrix(ytot))) / tmp
```

```
## [1] 1.106
```

```
## matrix fill
sum(ytot > 0) / prod(dim(ytot))
```

```
## [1] 0.04911
```

Check if counts are as expected:

```
max(ytot) # this is interesting
```

```
## [1] 200
```

```
sort(apply(as.matrix(ytot), 2, max)) # it is CANG
```

```
## BUFF BWTE COGO COHA DCCO GWTE HOLA NHOW NSHO RTHU WWSC CANV NOPI AMBI AMCO
##    0    0    0    0    0    0    0    0    0    0    0    0    1    1
## AMGO BAEA BAOR BEKI BOWA CONI CSWA EAPH GBHE GCTH GGOW GHOW HOWR LEOW MERL
##    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
## NESP NOGO NOHA NSWO PBGR RBGU RTHA SAVS SPSA WBNU BRBL CAGU MYWA SNBU VEER
##    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
## AMKE AMWI BADO BARS BBWO BHCO BLBW BLPW BLTE BWHA COGR DOWO EAKI HAWO KILL
##    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
## LEYE NAWA NOPO OSFL OSPR PIWO PUFI RNDU SORA SSHA COSN AMCR AMRO ATTW BHVI
```

```
##    2    2    2    2    2    2    2    2    2    2    2    3    3    3    3
## BOCH BRCR BTNW CMWA FOSP FRGU GCKI MAWR MOWA NOFL PHVI SACR SOSA SOSP SPGR
##    3    3    3    3    3    3    3    3    3    3    3    3    3    3    3
## TRES WETA WIWA WIWR YBSA FOTE BAWW BBWA BCCH BLJA CAWA CONW COTE GRYE NOWA
##    3    3    3    3    3    3    4    4    4    4    4    4    4    4    4
## NRWS OCWA REVI RNGR RUBL RWBL WAVI WEWP WISN YBFL YWAR ALFL AMRE CHSP CORA
##    4    4    4    4    4    4    4    4    4    4    4    5    5    5    5
## EVGR HETH LCSP RBGR RBNU RCKI SWSP CCSP COYE DEJU LEFL LISP MAWA OVEN RUGR
##    5    5    5    5    5    5    5    6    6    6    6    6    6    6    6
## SWTH BOGU MALL GRAJ PAWA WTSP YRWA COLO TEWA AMPI WWCR CEDW PISI RECR CANG
##    6    7    7    8    8    8    8    9   12   12   20   23   50   51  200
```

```
## lyover (FO) flock (FL) beyond 100m distance
head(josm$counts[
  josm$counts$SiteID == rownames(ytot)[which(ytot[,"CANG"] == 200)] &
  josm$counts$SpeciesID == "CANG",])
```

We can check overall mean counts:

```
round(sort(colMeans(ytot)), 4)
```

```
##   BUFF   BWTE   COGO   COHA   DCCO   GWTE   HOLA   NHOW   NSHO   RTHU
## 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##   WWSC   CANV   NOPI   GBHE   GCTH   GHOW   LEOW   NOHA   RBGU   BRBL
## 0.0000 0.0000 0.0000 0.0002 0.0002 0.0002 0.0002 0.0002 0.0002 0.0002
##   CAGU   AMCO   BAEA   BARS   NESP   NOGO   NOPO   NSWO   RNDU   SNBU
## 0.0002 0.0004 0.0004 0.0004 0.0004 0.0004 0.0004 0.0004 0.0004 0.0004
##   VEER   BEKI   CSWA   MERL   SAVS   SSHA   MYWA   AMKE   BAOR   OSPR
## 0.0004 0.0007 0.0007 0.0007 0.0007 0.0007 0.0007 0.0009 0.0009 0.0009
##   SPGR   WBNU   AMGO   AMWI   BOWA   CONI   EAPH   HOWR   NRWS   BLTE
## 0.0009 0.0009 0.0011 0.0011 0.0011 0.0011 0.0011 0.0011 0.0011 0.0013
##   COGR   EAKI   GGOW   NAWA   COSN   COTE   FRGU   MAWR   FOTE   KILL
## 0.0013 0.0013 0.0013 0.0013 0.0013 0.0015 0.0015 0.0015 0.0015 0.0018
##   RTHA   BADO   BLBW   AMBI   PBGR   SPSA   AMPI   BHCO   BWHA   SOSP
## 0.0020 0.0024 0.0024 0.0028 0.0028 0.0028 0.0028 0.0031 0.0037 0.0042
##   RUBL   MALL   PUFI   DOWO   SORA   LEYE   ATTW   HAWO   RNGR   BBWO
## 0.0044 0.0046 0.0048 0.0059 0.0068 0.0094 0.0096 0.0101 0.0101 0.0107
##   BLJA   BOGU   AMCR   EVGR   RWBL   OSFL   LCSP   TRES   FOSP   WEWP
## 0.0134 0.0140 0.0166 0.0169 0.0169 0.0186 0.0193 0.0201 0.0217 0.0232
```

```
##   WIWA   PIWO   RECR   SOSA   YWAR   GCKI   BLPW   CAWA   SACR   BTNW
## 0.0236 0.0256 0.0269 0.0269 0.0291 0.0304 0.0306 0.0315 0.0322 0.0335
##   NOWA   OCWA   BRCR   CCSP   COLO   PHVI   CONW   CEDW   RUGR   MOWA
## 0.0341 0.0359 0.0381 0.0385 0.0387 0.0394 0.0429 0.0449 0.0475 0.0477
##   WAVI   BCCH   BOCH   NOFL   SWSP   GRYE   WWCR   AMRO   RBNU   BBWA
## 0.0582 0.0593 0.0593 0.0622 0.0659 0.0685 0.0751 0.0757 0.0766 0.0810
##   CMWA   BHVI   COYE   YBFL   YBSA   AMRE   BAWW   LEFL   WETA   WISN
## 0.0812 0.0814 0.0814 0.0873 0.0878 0.0889 0.0963 0.0974 0.1086 0.1280
##   CORA   WIWR   ALFL   MAWA   PISI   RBGR   LISP   DEJU   GRAJ   CANG
## 0.1401 0.1466 0.1582 0.1727 0.1775 0.1832 0.2169 0.2725 0.2898 0.3018
##   PAWA   REVI   RCKI   HETH   CHSP   SWTH   WTSP   OVEN   YRWA   TEWA
## 0.3053 0.3344 0.3898 0.4344 0.4460 0.7402 0.8091 0.8831 0.8934 1.2221
```

2.6 Joining species data with predictors

Let's join the species counts with the survey attributes. This is how we can prepare the input data for regression analysis.

```
spp <- "OVEN" # which species
josm$species[spp,]

compare_sets(rownames(josm$surveys),rownames(ytot))
```

```
##        xlength ylength intersect union xbutnoty ybutnotx
## labels    4569    4569      4569  4569        0        0
## unique    4569    4569      4569  4569        0        0
```

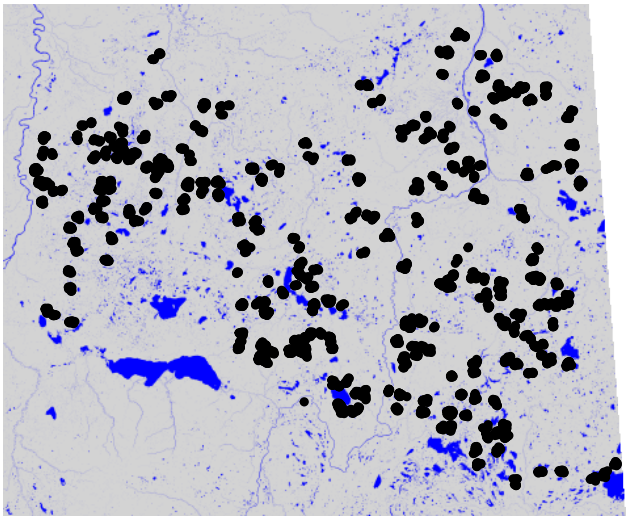
```
x <- josm$surveys
x$y <- as.numeric(ytot[rownames(x), spp])
```

2.7 Explore predictor variables

Put the locations on the map:

```
xy <- x[,c("Longitude", "Latitude")]
coordinates(xy) <- ~ Longitude + Latitude
```

```
proj4string(xy) <- "+proj=longlat +ellps=GRS80 +datum=NAD83 +no_defs"
xy <- spTransform(xy, proj4string(rr))
col <- colorRampPalette(c("lightgrey", "blue"))(100)
plot(rr[["Water"]], col=col, axes=FALSE, box=FALSE, legend=FALSE)
plot(xy, add=TRUE, pch=19, cex=0.5)
```



Exercise
Explore the data to understand the distributions and associations. Use `summary`, `table`, `hist`, `plot` (bivariate, scatterplot matrix), etc.

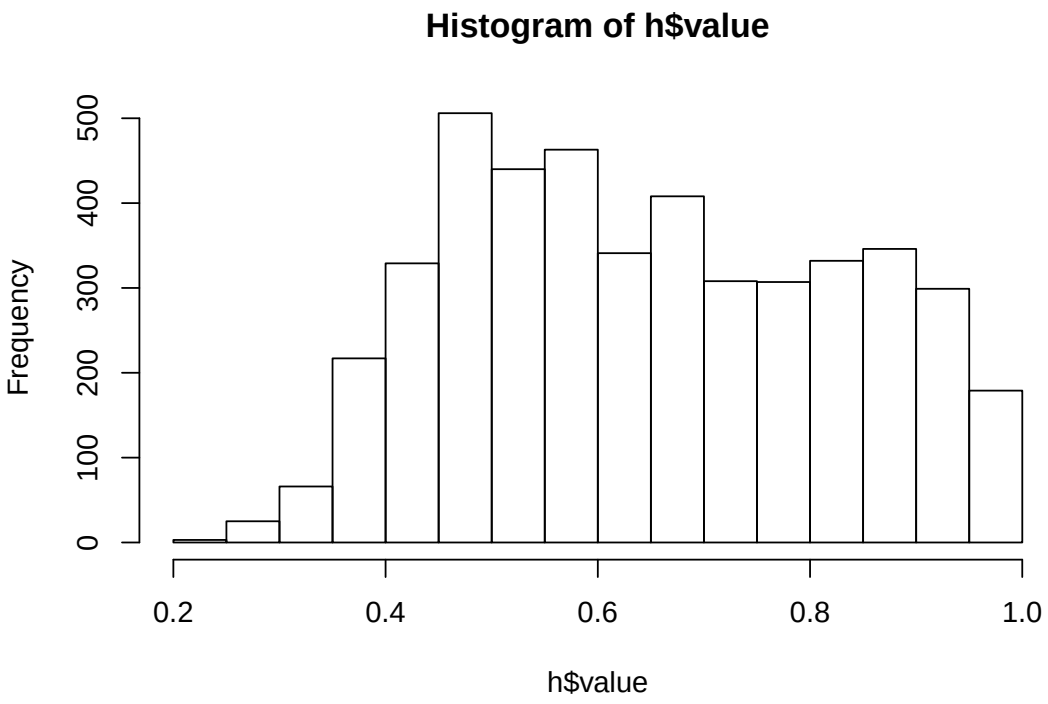
2.8 Derived variables

Add up some of the compositional variables into meaningful units:

```
x$FOR <- x$Decid + x$Conif+ x$ConifWet # forest
x$AHF <- x$Agr + x$UrbInd + x$Roads # 'alienating' human footprint
x$WET <- x$OpenWet + x$ConifWet + x$Water # wet + water
```

Classify surveys locations based on dominant land cover type:

```
cn <- c("Open", "Water", "Agr", "UrbInd", "SoftLin", "Roads", "Decid",
  "OpenWet", "Conif", "ConifWet")
h <- find_max(x[,cn])
hist(h$value)
```




```
table(h$index)
```

```
##
##     Open    Water      Agr   UrbInd  SoftLin    Roads    Decid  OpenWet
##       12       10        4       14        0        2     2084      160
##    Conif ConifWet
##      745     1538
```

```
x$HAB <- droplevels(h$index) # drop empty levels
```

Chapter 3

A Primer in Regression Techniques

> All models are wrong, but some are useful – Box

3.1 Introduction

This chapter will provide all the foundations we need for the coming chapters. It is not intended as a general and all-exhaustive introduction to regression techniques, but rather the minimum requirement moving forwards. We will also hone our data processing and plotting skills.

3.2 Prerequisites

```
library(mefa4)                  # data manipulation
library(mgcv)                   # GAMs
library(pscl)                   # zero-inflated models
library(lme4)                   # GLMMs
library(MASS)                   # Negative Binomial GLM
library(partykit)               # regression trees
library(intrval)                # interval magic
```

```
library(opticut)                # optimal partitioning
library(visreg)                 # regression visualization
library(ResourceSelection)      # marginal effects
library(MuMIn)                  # multi-model inference
source("functions.R")           # some useful stuff
load("_data/josm/josm.rda") # JOSM data
```

Let's pick a species, Ovenbird (OVEN), that is quite common and abundant in the data set. We put together a little data set to work with:

```
spp <- "OVEN"

ytot <- Xtab(~ SiteID + SpeciesID, josm$counts[josm$counts$DetectType1 != "V",
ytot <- ytot[,colSums(ytot > 0) > 0]
x <- data.frame(
  josm$surveys,
  y=as.numeric(ytot[rownames(josm$surveys), spp]))
x$FOR <- x$Decid + x$Conif+ x$ConifWet # forest
x$AHF <- x$Agr + x$UrbInd + x$Roads # 'alienating' human footprint
x$WET <- x$OpenWet + x$ConifWet + x$Water # wet + water
cn <- c("Open", "Water", "Agr", "UrbInd", "SoftLin", "Roads", "Decid",
  "OpenWet", "Conif", "ConifWet")
x$HAB <- droplevels(find_max(x[,cn])$index) # drop empty levels
x$DEC <- ifelse(x$HAB == "Decid", 1, 0)

table(x$y)
```

```
##
##    0    1    2    3    4    5    6
## 2493  883  656  363  132   29   13
```

3.3 Poisson null model

The null model states that the expected values of the count at all locations are identical: $E[Y_i] = \lambda$ $(i = 1, ..., n)$, where Y_i is a random variable that follows a Poisson distribution with mean λ: $(Y_i \mid \lambda) \sim Poisson(\lambda)$. The observation (y_i) is a realization of the random variables Y at site i, these observations are

independent and identically distributed (i.i.d.), and we have n observations in total.

Saying the the distribution is Poisson is an assumption in itself. For example we assume that the variance equals the mean ($V(\mu) = \mu$).

```
mP0 <- glm(y ~ 1, data=x, family=poisson)
mean(x$y)
```

```
## [1] 0.8831
```

```
mean(fitted(mP0))
```

```
## [1] 0.8831
```

```
exp(coef(mP0))
```

```
## (Intercept)
##      0.8831
```

```
summary(mP0)
```

```
##
## Call:
## glm(formula = y ~ 1, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
##  -1.33   -1.33   -1.33    1.02    3.57
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.1243     0.0157   -7.89  2.9e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 7424.8  on 4568  degrees of freedom
## Residual deviance: 7424.8  on 4568  degrees of freedom
## AIC: 12573
```

```
##
## Number of Fisher Scoring iterations: 6
```

The `family=poisson` specification implicitly assumes that we use a logarithmic link functions, that is to say that $log(\lambda) = \beta_0$, or equivalently: $\lambda = e^{\beta_0}$. The mean of the observations equal the mean of the fitted values, as expected.

The logarithmic function is called the link function, its inverse, the exponential function is called the inverse link function. The model family has these convenently stored for us:

```
mP0$family
```

```
##
## Family: poisson
## Link function: log
```

```
mP0$family$linkfun
```

```
## function (mu)
## log(mu)
## <environment: namespace:stats>
```

```
mP0$family$linkinv
```

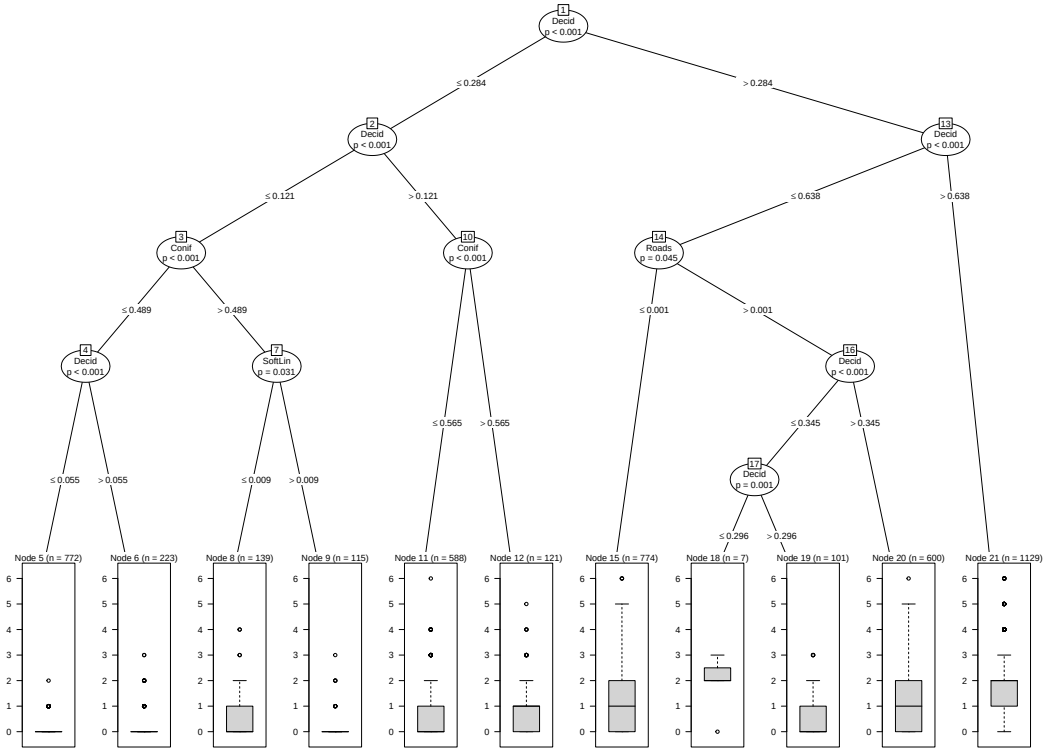
```
## function (eta)
## pmax(exp(eta), .Machine$double.eps)
## <environment: namespace:stats>
```

3.4 Exploring covariates

Now, in the absence of info about species biology, we are looking at a blank page. How should we proceed? What kind of covariate (linear predictor) should we use? We can do a quick and dirty exploration to see what are the likely candidates. We use a regression tree (`ctree` refers to conditional trees). It is a nonparametric method based on binary recursive partitioning in a conditional inference framework. This means that binary splits are made along the predictor variables, and the explanatory power of the split is assessed based on how it maximized difference between the splits and minimized

the difference inside the splits. It is called conditional, because every new split is conditional on the previous splits (difference can be measured in many different ways, think e.g. sum of squares). The stopping rule in this implementation is based on permutation tests (see `?ctree` or details and references).

```
mCT <- ctree(y ~ Open + Water + Agr + UrbInd + SoftLin + Roads +
  Decid + OpenWet + Conif + ConifWet, data=x)
plot(mCT)
```



The model can be seen as a piecewise constant regression, where each bucket (defined by the splits along the tree) yields a constant predictions based on the mean of the observations in the bucket. Any new data classified into the same bucket will get the same value. There is no notion of uncertainty (confidence or prediction intervals) in this nonparameric model.

But we see something very useful: the proportion of deciduous forest in the landscape seems to be vary influential for Ovenbird abundance.

3.5 Single covariate

With this new found knowledge, let's fit a parametric (Poisson) linear model using `Decid` as a predictor:

```
mP1 <- glm(y ~ Decid, data=x, family=poisson)
mean(x$y)
```

```
## [1] 0.8831
```

```
mean(fitted(mP0))
```

```
## [1] 0.8831
```

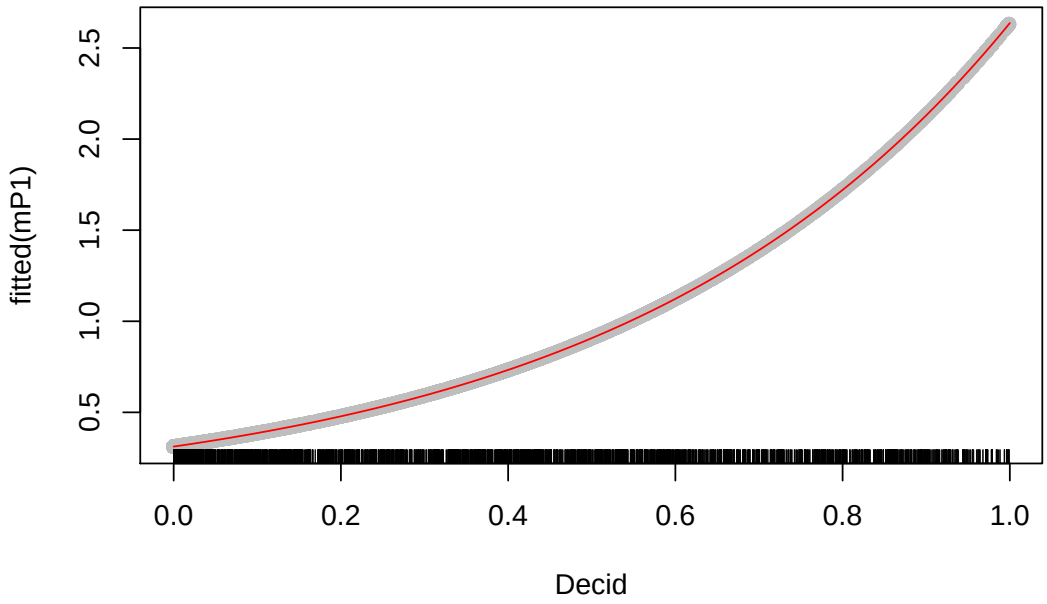
```
coef(mP1)
```

```
## (Intercept)       Decid
##      -1.164       2.134
```

Same as before, the mean of the observations equal the mean of the fitted values. But instead of only the intercapt, now we have 2 coefficients estimated. Our linear predictor thus looks like: $log(\lambda_i) = \beta_0 + \beta_1 x_{1i}$. This means that expected abundance is e^{β_0} where `Decid`=0, $e^{\beta_0} e^{\beta_1}$ where `Decid`=1, and $e^{\beta_0 + \beta_1 x_1}$ in between.

The relationship can be visualized by plotting the fitted values against the predictor, or using the coefficients to make predictions using our formula:

```
dec <- seq(0, 1, 0.01)
lam <- exp(coef(mP1)[1] + coef(mP1)[2] * dec)
plot(fitted(mP1) ~ Decid, x, pch=19, col="grey")
lines(lam ~ dec, col=2)
rug(x$Decid)
```



The model summary tells us that resudials are not quite right (we would expect 0 median and symmertic tails), in line with residual deviance being much higher than residual degrees of freedom (these should be close if the Poisson assumption holds). But, the `Decid` effect is significant (meaning that the effect size is large compared to the standard error):

```
summary(mP1)
```

```
##
## Call:
## glm(formula = y ~ Decid, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.291  -0.977  -0.790   0.469   4.197
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.1643     0.0352   -33.1   <2e-16 ***
## Decid         2.1338     0.0537    39.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 7424.8  on 4568  degrees of freedom
## Residual deviance: 5736.9  on 4567  degrees of freedom
## AIC: 10887
##
## Number of Fisher Scoring iterations: 6
```

We can compare this model to the null (constant, intercept-only) model:

```
AIC(mP0, mP1)
BIC(mP0, mP1)
model.sel(mP0, mP1)
```

```
## Model selection table
##     (Intrc) Decid df logLik  AICc delta weight
## mP1 -1.1640 2.134  2  -5442 10887     0      1
## mP0 -0.1243        1  -6285 12573  1686      0
## Models ranked by AICc(x)
```

```
R2dev(mP0, mP1)
```

```
##          R2   R2adj Deviance    Dev0    DevR     df0     dfR p_value
## mP0    0.00    0.00     0.00 7424.78 7424.78 4568.00 4568.00  <2e-16 ***
## mP1    0.23    0.23  1687.87 7424.78 5736.91 4568.00 4567.00  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

AIC uses the negative log likelihood and the number of parameters as penalty. Smaller value indicate a model that is closer to the (unknowable) true model (caveat: this statement is true only asymptotically, i.e. it holds for very large sample sizes). For small samples, we of ten use BIC (more penalty for complex models when sample size is small), or AICc (as in `MuMIn::model.sel`).

The other little table returned by `R2dev` shows deviance based (quasi) R^2 and adjusted R^2 for some GLM classes, just for the sake of completeness. The Chi-squared based test indicates good fit when the p-value is high (probability of being distributed according the Poisson).

None of these two models is a particularly good fit in terms of the parametric

distribution. This, however does not mean these models are not useful for making inferential statements about ovenbirds. How useful these statements are, that is another question. Let's dive into cinfidence and prediction intervals a bit.

```
B <- 2000
alpha <- 0.05

xnew <- data.frame(Decid=seq(0, 1, 0.01))
CI0 <- predict_sim(mP0, xnew, interval="confidence", level=1-alpha, B=B)
PI0 <- predict_sim(mP0, xnew, interval="prediction", level=1-alpha, B=B)
CI1 <- predict_sim(mP1, xnew, interval="confidence", level=1-alpha, B=B)
PI1 <- predict_sim(mP1, xnew, interval="prediction", level=1-alpha, B=B)

## nominal coverage is 95%
sum(x$y %[]% predict_sim(mP0, interval="prediction", level=1-alpha, B=B)[,c("lwr", "u
```

```
## [1] 0.9619
```

```
sum(x$y %[]% predict_sim(mP1, interval="prediction", level=1-alpha, B=B)[,c("lwr", "u
```

```
## [1] 0.9711
```

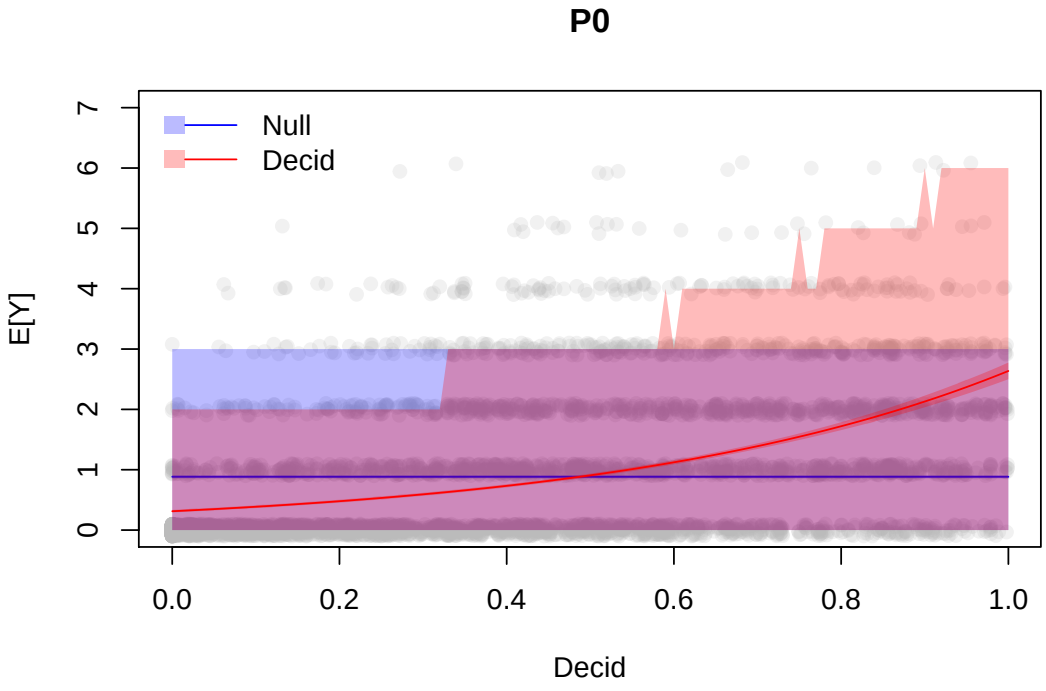
A model is said to have good *coverage* when the prediction intervals encompass the right amount of the observations. When the nominal level is 95% ($100 \times (1-\alpha)$, where α is Type I. error rate), we expect 95% of the observations fall within the 95% *prediction interval*. The prediction interval includes the uncertainty around the coefficients (confidence intervals, uncertainty in $\hat{\lambda}$) and the stochasticity coming from the Poisson distribution ($Y_i \sim Poisson(\hat{\lambda})$).

The code above calculate the confidence and prediction intervals for the two models. We also compared the prediction intervals and the nomial levels, and those were quite close (ours being a bit more conservative), hinting that maybe the Poisson distributional assumption is not very bad after all, but we'll come back to this later.

Let's see our confidence and prediction intervals for the two models:

```
yj <- jitter(x$y, 0.5)

plot(yj ~ Decid, x, xlab="Decid", ylab="E[Y]",
```

```
  ylim=c(0, max(PI1$upr)+1), pch=19, col="#bbbbbb33", main="P0")

polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(PI0$lwr, rev(PI0$upr)), border=NA, col="#0000ff44")
polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(CI0$lwr, rev(CI0$upr)), border=NA, col="#0000ff44")
lines(CI0$fit ~ xnew$Decid, lty=1, col=4)

polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(PI1$lwr, rev(PI1$upr)), border=NA, col="#ff000044")
polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(CI1$lwr, rev(CI1$upr)), border=NA, col="#ff000044")
lines(CI1$fit ~ xnew$Decid, lty=1, col=2)

legend("topleft", bty="n", fill=c("#0000ff44", "#ff000044"), lty=1, col=c(4,2)
  border=NA, c("Null", "Decid"))
```



Exercise
What can we conclude from this plot?
Coverage is comparable, so what is the difference then?
Which model should I use for prediction and why? (Hint: look at the non overlapping regions.)

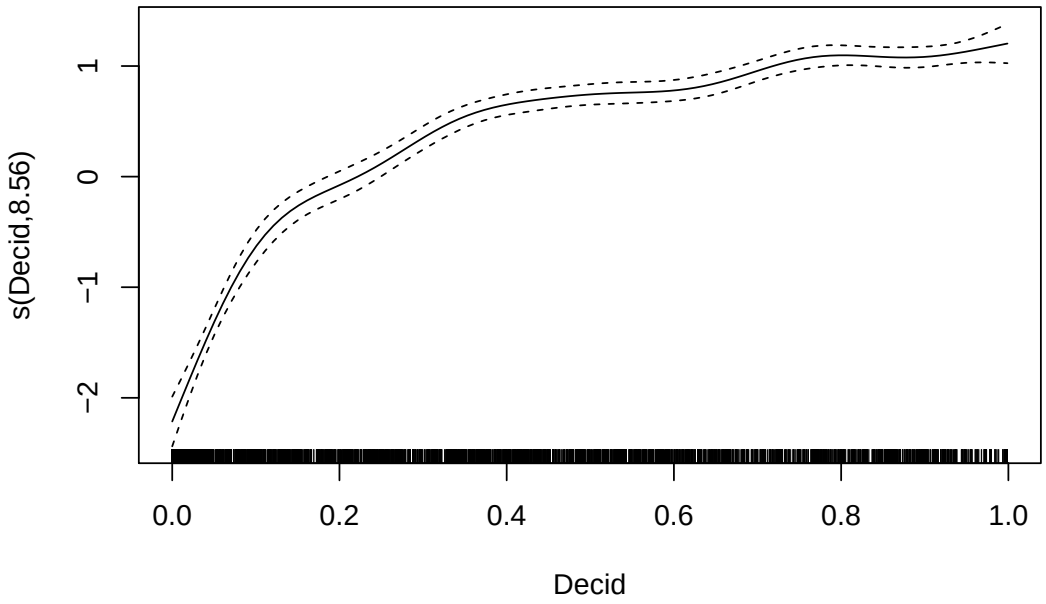
3.6 Additive model

Generalized additive models (GAMs) are semiparametric, meaning that parametric assumptions apply, but responses are modelled more flexibly.

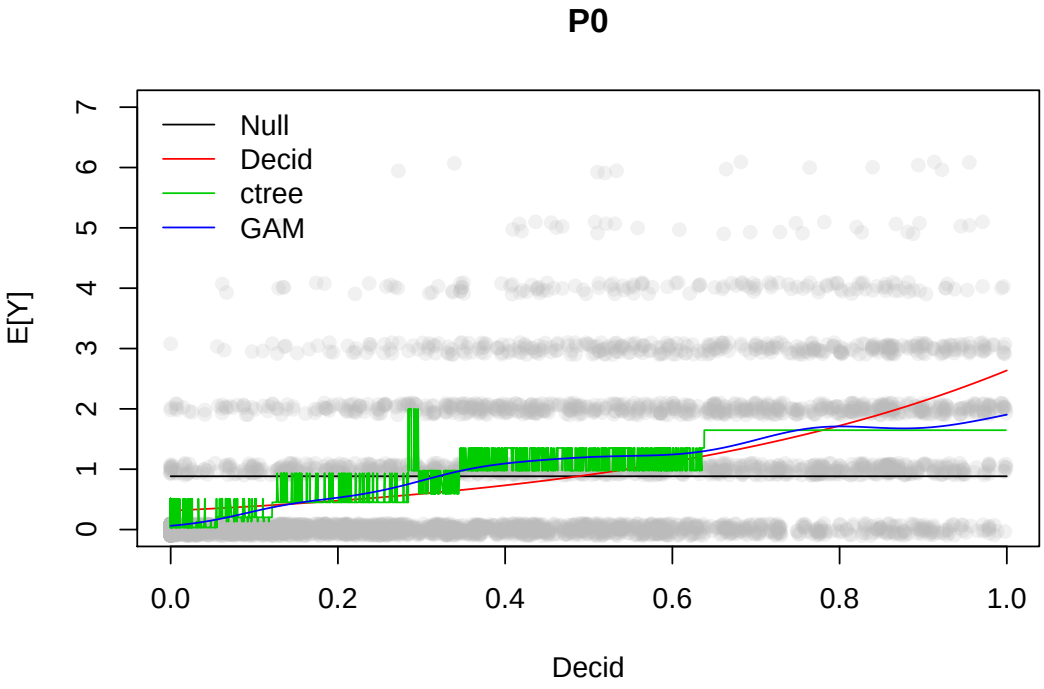
```
mGAM <- mgcv::gam(y ~ s(Decid), x, family=poisson)
summary(mGAM)
```

```
##
## Family: poisson
## Link function: log
##
## Formula:
## y ~ s(Decid)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.5606     0.0283   -19.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##           edf Ref.df Chi.sq p-value
## s(Decid) 8.56   8.94   1193  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.239   Deviance explained =   29%
## UBRE = 0.15808  Scale est. = 1         n = 4569
```

```
plot(mGAM)
```




```
fitCT <- predict(mCT, x[order(x$Decid),])
fitGAM <- predict(mGAM, xnew, type="response")

plot(yj ~ Decid, x, xlab="Decid", ylab="E[Y]",
  ylim=c(0, max(PI1$upr)+1), pch=19, col="#bbbbbb33", main="P0")
lines(CI0$fit ~ xnew$Decid, lty=1, col=1)
lines(CI1$fit ~ xnew$Decid, lty=1, col=2)
lines(fitCT ~ x$Decid[order(x$Decid)], lty=1, col=3)
lines(fitGAM ~ xnew$Decid, lty=1, col=4)
legend("topleft", bty="n", lty=1, col=1:4,
  legend=c("Null", "Decid", "ctree", "GAM"))
```



Exercise
Play with GAM and other variables to understand response curves:
`plot(mgcv::gam(y ~ s(<variable_name>), data=x, family=poisson))`

3.7 Nonlinear terms

We can use polynomial terms to approximate the GAM fit:

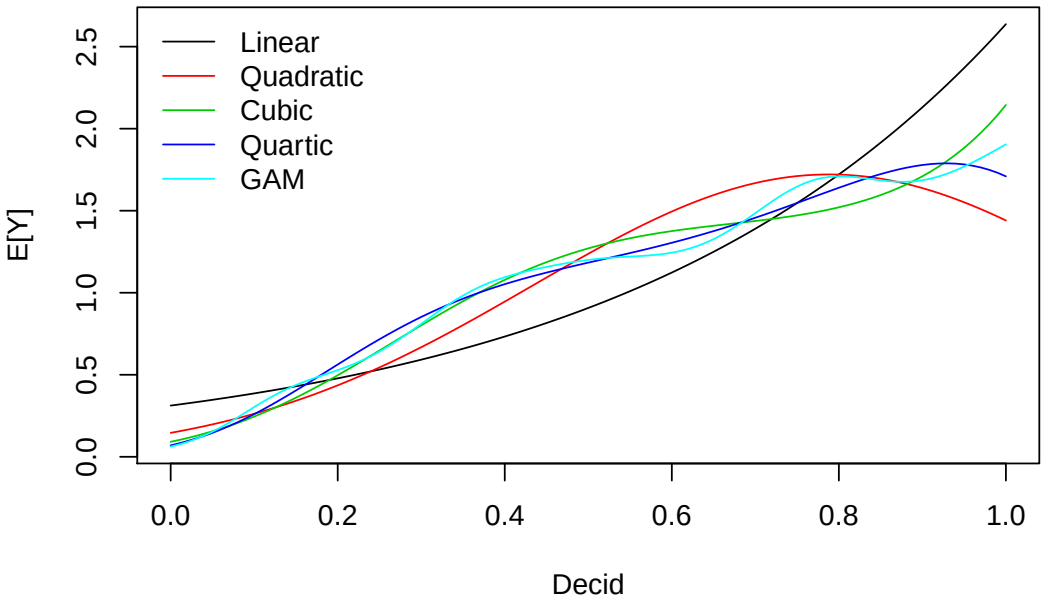
```
mP12 <- glm(y ~ Decid + I(Decid^2), data=x, family=poisson)
mP13 <- glm(y ~ Decid + I(Decid^2) + I(Decid^3), data=x, family=poisson)
mP14 <- glm(y ~ Decid + I(Decid^2) + I(Decid^3) + I(Decid^4), data=x, family=poisson)
model.sel(mP1, mP12, mP13, mP14, mGAM)
```

```
## Model selection table
##        (Int)    Dcd  Dcd^2  Dcd^3  Dcd^4 s(Dcd) class df logLik  AICc
```

```
## mGAM -0.5606                                      +   gam  9  -5209 10438
## mP14 -2.6640 16.640 -38.60 41.470 -16.31              glm  5  -5215 10441
## mP13 -2.3910 11.400 -16.31  8.066                     glm  4  -5226 10461
## mP12 -1.9240  6.259  -3.97                            glm  3  -5269 10544
## mP1  -1.1640  2.134                                   glm  2  -5442 10887
##       delta weight
## mGAM   0.00   0.84
## mP14   3.31   0.16
## mP13  23.42   0.00
## mP12 106.04   0.00
## mP1  449.60   0.00
## Models ranked by AICc(x)
```

Not a surprise that the most complex model won. GAM was more complex than that.

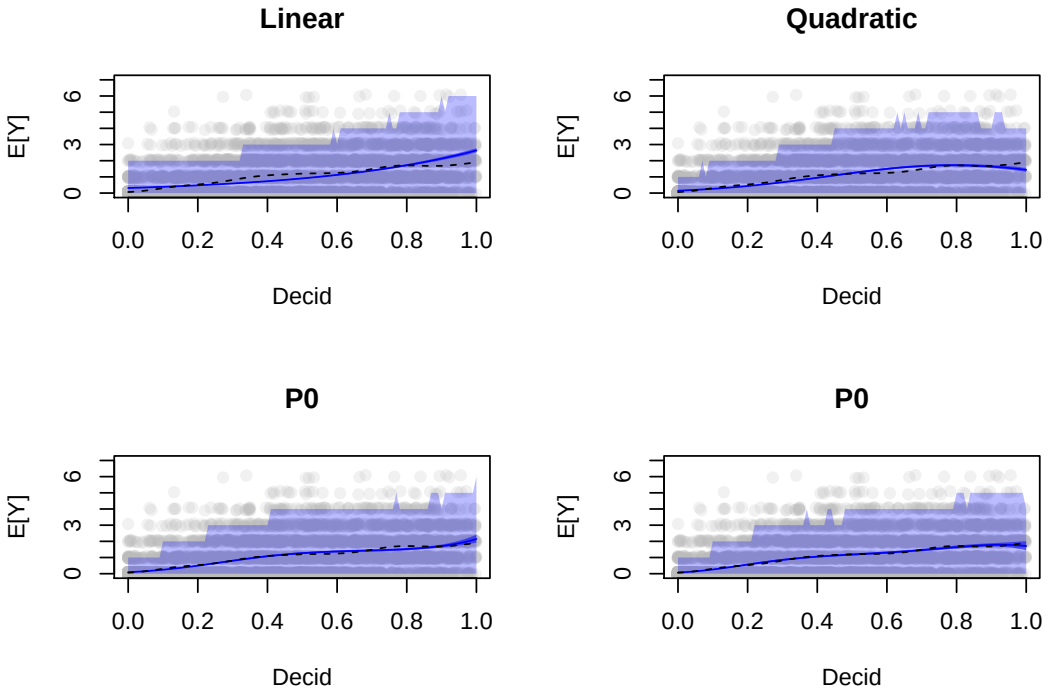
```
pr <- cbind(
  predict(mP1, xnew, type="response"),
  predict(mP12, xnew, type="response"),
  predict(mP13, xnew, type="response"),
  predict(mP14, xnew, type="response"),
  fitGAM)
matplot(xnew$Decid, pr, lty=1, type="l",
  xlab="Decid", ylab="E[Y]")
legend("topleft", lty=1, col=1:5, bty="n",
  legend=c("Linear", "Quadratic", "Cubic", "Quartic", "GAM"))
```



Let's see how these affect our prediction intervals:

```
CI12 <- predict_sim(mP12, xnew, interval="confidence", level=1-alpha, B=B)
PI12 <- predict_sim(mP12, xnew, interval="prediction", level=1-alpha, B=B)
CI13 <- predict_sim(mP13, xnew, interval="confidence", level=1-alpha, B=B)
PI13 <- predict_sim(mP13, xnew, interval="prediction", level=1-alpha, B=B)
CI14 <- predict_sim(mP14, xnew, interval="confidence", level=1-alpha, B=B)
PI14 <- predict_sim(mP14, xnew, interval="prediction", level=1-alpha, B=B)

op <- par(mfrow=c(2,2))
plot(yj ~ Decid, x, xlab="Decid", ylab="E[Y]",
  ylim=c(0, max(PI1$upr)+1), pch=19, col="#bbbbbb33", main="Linear")
polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(PI1$lwr, rev(PI1$upr)), border=NA, col="#0000ff44")
polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(CI1$lwr, rev(CI1$upr)), border=NA, col="#0000ff88")
lines(CI1$fit ~ xnew$Decid, lty=1, col=4)
lines(fitGAM ~ xnew$Decid, lty=2, col=1)

plot(yj ~ Decid, x, xlab="Decid", ylab="E[Y]",
  ylim=c(0, max(PI1$upr)+1), pch=19, col="#bbbbbb33", main="Quadratic")
polygon(c(xnew$Decid, rev(xnew$Decid)),
```

```
  c(PI12$lwr, rev(PI12$upr)), border=NA, col="#0000ff44")
polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(CI12$lwr, rev(CI12$upr)), border=NA, col="#0000ff88")
lines(CI12$fit ~ xnew$Decid, lty=1, col=4)
lines(fitGAM ~ xnew$Decid, lty=2, col=1)

plot(yj ~ Decid, x, xlab="Decid", ylab="E[Y]",
  ylim=c(0, max(PI1$upr)+1), pch=19, col="#bbbbbb33", main="P0")
polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(PI13$lwr, rev(PI13$upr)), border=NA, col="#0000ff44")
polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(CI13$lwr, rev(CI13$upr)), border=NA, col="#0000ff88")
lines(CI13$fit ~ xnew$Decid, lty=1, col=4)
lines(fitGAM ~ xnew$Decid, lty=2, col=1)

plot(yj ~ Decid, x, xlab="Decid", ylab="E[Y]",
  ylim=c(0, max(PI1$upr)+1), pch=19, col="#bbbbbb33", main="P0")
polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(PI14$lwr, rev(PI14$upr)), border=NA, col="#0000ff44")
polygon(c(xnew$Decid, rev(xnew$Decid)),
  c(CI14$lwr, rev(CI14$upr)), border=NA, col="#0000ff88")
lines(CI14$fit ~ xnew$Decid, lty=1, col=4)
lines(fitGAM ~ xnew$Decid, lty=2, col=1)
```




```
par(op)
```

3.8 Categorical variables

Categorical variables are expanded into a *model matrix* before estimation. The model matrix usually contains indicator variables for each level (value 1 when factor value equals a particular label, 0 otherwise) except for the *reference category* (check `relevel` if you want to change the reference category).

The estimate for the reference category comes from the intercept, the rest of the estimates are relative to the reference category. In the log-linear model example this means a ratio.

```
head(model.matrix(~DEC, x))
```

```
##         (Intercept) DEC
## CL10102           1   1
## CL10106           1   0
## CL10108           1   0
```

```
## CL10109           1   1
## CL10111           1   1
## CL10112           1   1
```

```
mP2 <- glm(y ~ DEC, data=x, family=poisson)
summary(mP2)
```

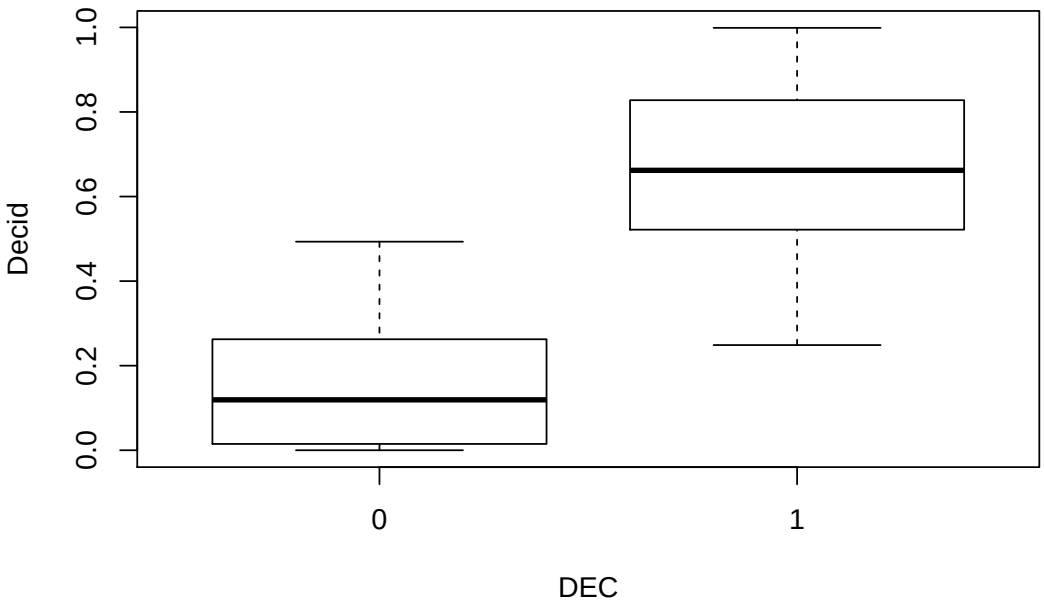
```
##
## Call:
## glm(formula = y ~ DEC, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.691  -0.921  -0.921   0.449   4.543
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.8577     0.0308   -27.8   <2e-16 ***
## DEC           1.2156     0.0358    33.9   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 7424.8  on 4568  degrees of freedom
## Residual deviance: 6095.5  on 4567  degrees of freedom
## AIC: 11246
##
## Number of Fisher Scoring iterations: 6
```

```
coef(mP2)
```

```
## (Intercept)         DEC
##     -0.8577      1.2156
```

The estimate for a non-deciduous landscape is e^{β_0}, and it is $e^{\beta_0}e^{\beta_1}$ for deciduous landscapes. Of course such binary classification at the landscape (1 km^2) level doesn't really makes sense for various reasons:

```
boxplot(Decid ~ DEC, x)
```




```
model.sel(mP1, mP2)
```

```
## Model selection table
##     (Intrc) Decid   DEC df logLik  AICc delta weight
## mP1 -1.1640 2.134         2  -5442 10887   0.0      1
## mP2 -0.8577       1.216   2  -5621 11246 358.6      0
## Models ranked by AICc(x)
```

```
R2dev(mP1, mP2)
```

```
##          R2   R2adj Deviance    Dev0    DevR     df0     dfR p_value
## mP1    0.23    0.23  1687.87 7424.78 5736.91 4568.00 4567.00  <2e-16 ***
## mP2    0.18    0.18  1329.23 7424.78 6095.55 4568.00 4567.00  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Having estimates for each land cover type improves the model, but the model using continuous variable is still better

```
mP3 <- glm(y ~ HAB, data=x, family=poisson)
summary(mP3)
```

```
##
## Call:
## glm(formula = y ~ HAB, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.691  -0.873  -0.817   0.449   4.832
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.386      0.577   -2.40   0.0163 *
## HABWater       1.030      0.690    1.49   0.1357
## HABAgr         0.693      0.913    0.76   0.4477
## HABUrbInd      0.134      0.764    0.17   0.8612
## HABRoads     -10.916    201.285   -0.05   0.9567
## HABDecid       1.744      0.578    3.02   0.0025 **
## HABOpenWet     0.422      0.591    0.71   0.4755
## HABConif       0.913      0.579    1.58   0.1150
## HABConifWet    0.288      0.579    0.50   0.6185
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 7424.8  on 4568  degrees of freedom
## Residual deviance: 5997.2  on 4560  degrees of freedom
## AIC: 11161
##
## Number of Fisher Scoring iterations: 10
```

```
model.sel(mP1, mP2, mP3)
```

```
## Model selection table
##     (Intrc) Decid   DEC HAB df logLik  AICc delta weight
## mP1 -1.1640 2.134            2  -5442 10887   0.0      1
## mP3 -1.3860               +  9  -5572 11162 274.4      0
## mP2 -0.8577       1.216      2  -5621 11246 358.6      0
## Models ranked by AICc(x)
```

```
R2dev(mP1, mP2, mP3)
```

```
##          R2   R2adj Deviance    Dev0    DevR     df0     dfR p_value
## mP1    0.23    0.23  1687.87 7424.78 5736.91 4568.00 4567.00  <2e-16 ***
## mP2    0.18    0.18  1329.23 7424.78 6095.55 4568.00 4567.00  <2e-16 ***
## mP3    0.19    0.19  1427.55 7424.78 5997.23 4568.00 4560.00  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The prediction in this case would look like: $log(\lambda_i) = \beta_0 + \sum_{j=1}^{k-1} \beta_j x_{ji}$, where we have k factor levels (and $k-1$ indicator variables besides the intercept).

Here is a general way of calculating fitted values or making predictions based on the design matrix (`X`) and the coefficients (`b`) (column ordering in `X` must match the elements in `b`) given a parametric log-linear model `object` and data frame `df`:

```
b <- coef(object)
X <- model.matrix(formula(object), df)
exp(X %*% b)
```

3.9 Multiple main effects

We can keep adding variables to the model in a forwards-selection fashion. `add1` adds variables one at a time, selecting from the scope defined by the formula:

```
scope <- as.formula(paste("~ FOR + WET + AHF +",paste(cn, collapse="+")))
tmp <- add1(mP1, scope)
tmp$AIC_drop <- tmp$AIC-tmp$AIC[1] # current model
tmp[order(tmp$AIC),]
```

```
## Single term additions
##
## Model:
## y ~ Decid
##          Df Deviance   AIC AIC_drop
## ConifWet  1     5638 10791    -96.5
```

```
## Conif     1     5685 10838    -49.4
## WET       1     5687 10839    -48.1
## Water     1     5721 10873    -13.7
## FOR       1     5724 10876    -11.0
## OpenWet   1     5728 10880     -6.9
## Open      1     5730 10882     -4.7
## Roads     1     5733 10885     -1.9
## AHF       1     5734 10886     -0.7
## <none>          5737 10887      0.0
## Agr       1     5736 10888      1.2
## UrbInd    1     5736 10889      1.5
## SoftLin   1     5737 10889      1.6
```

It looks like `ConifWet` is the best covariate to add next because it leads to the biggest drop in AIC, and both effects are significant.

```
mP4 <- glm(y ~ Decid + ConifWet, data=x, family=poisson)
summary(mP4)
```

```
##
## Call:
## glm(formula = y ~ Decid + ConifWet, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.237  -0.996  -0.679   0.447   4.439
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.7014     0.0556  -12.61   <2e-16 ***
## Decid         1.6224     0.0719   22.57   <2e-16 ***
## ConifWet     -0.9785     0.0993   -9.86   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 7424.8  on 4568  degrees of freedom
## Residual deviance: 5638.4  on 4566  degrees of freedom
```

```
## AIC: 10791
##
## Number of Fisher Scoring iterations: 6
```

`drop1` is the function opposite of `add1`, it assesses which term should be dropped from a more saturated model:

```
formula_all <- y ~ Open + Agr + UrbInd + SoftLin + Roads +
  Decid + OpenWet + Conif + ConifWet +
  OvernightRain + TSSR + DAY + Longitude + Latitude

tmp <- drop1(glm(formula_all, data=x, family=poisson))
tmp$AIC_drop <- tmp$AIC-tmp$AIC[1] # current model
tmp[order(tmp$AIC),]
```

```
## Single term deletions
##
## Model:
## y ~ Open + Agr + UrbInd + SoftLin + Roads + Decid + OpenWet +
##     Conif + ConifWet + OvernightRain + TSSR + DAY + Longitude +
##     Latitude
##               Df Deviance   AIC AIC_drop
## OvernightRain  1     5500 10674     -2.0
## Roads          1     5500 10674     -1.9
## SoftLin        1     5500 10675     -1.6
## Agr            1     5501 10675     -1.4
## <none>               5500 10676      0.0
## Decid          1     5505 10679      3.0
## OpenWet        1     5505 10679      3.1
## Conif          1     5508 10682      6.0
## UrbInd         1     5511 10685      8.7
## Longitude      1     5519 10693     16.5
## TSSR           1     5524 10698     21.8
## ConifWet       1     5528 10703     26.4
## DAY            1     5529 10703     26.7
## Open           1     5531 10705     28.7
## Latitude       1     5580 10754     78.2
```

The `step` function can be used to automatically select the best model based

on adding/dropping terms:

```
mPstep <- step(glm(formula_all, data=x, family=poisson),
  trace=0) # use trace=1 to see all the steps
summary(mPstep)
```

```
##
## Call:
## glm(formula = y ~ Open + UrbInd + Decid + OpenWet + Conif + ConifWet +
##     TSSR + DAY + Longitude + Latitude, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.763  -0.986  -0.674   0.451   4.624
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -5.88293    1.30223   -4.52  6.3e-06 ***
## Open        -3.47428    0.65867   -5.27  1.3e-07 ***
## UrbInd      -1.66883    0.54216   -3.08  0.00208 **
## Decid        0.83372    0.25957    3.21  0.00132 **
## OpenWet     -0.74076    0.30238   -2.45  0.01430 *
## Conif       -0.88558    0.26566   -3.33  0.00086 ***
## ConifWet    -1.89423    0.27170   -6.97  3.1e-12 ***
## TSSR        -1.23416    0.24984   -4.94  7.8e-07 ***
## DAY         -2.87970    0.52686   -5.47  4.6e-08 ***
## Longitude    0.03831    0.00877    4.37  1.2e-05 ***
## Latitude     0.20930    0.02309    9.06  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 7424.8  on 4568  degrees of freedom
## Residual deviance: 5501.1  on 4558  degrees of freedom
## AIC: 10669
##
## Number of Fisher Scoring iterations: 6
```

3.10 Interaction

When we consider interactions between two variables (say x_1 and x_2), we really referring to adding another variable to the model matrix that is a product of the two variables ($x_{12} = x_1 x_2$):

```
head(model.matrix(~x1 * x2, data.frame(x1=1:4, x2=10:7)))
```

```
##   (Intercept) x1 x2 x1:x2
## 1           1  1 10    10
## 2           1  2  9    18
## 3           1  3  8    24
## 4           1  4  7    28
```

Let's consider interaction between our two predictors from before:

```
mP5 <- glm(y ~ Decid * ConifWet, data=x, family=poisson)
summary(mP5)
```

```
##
## Call:
## glm(formula = y ~ Decid * ConifWet, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.081  -1.022  -0.484   0.374   4.321
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -0.5604     0.0566    -9.9   <2e-16 ***
## Decid            1.2125     0.0782    15.5   <2e-16 ***
## ConifWet        -2.3124     0.1490   -15.5   <2e-16 ***
## Decid:ConifWet   5.3461     0.3566    15.0   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 7424.8  on 4568  degrees of freedom
```

```
## Residual deviance: 5395.2  on 4565  degrees of freedom
## AIC: 10549
##
## Number of Fisher Scoring iterations: 6
```

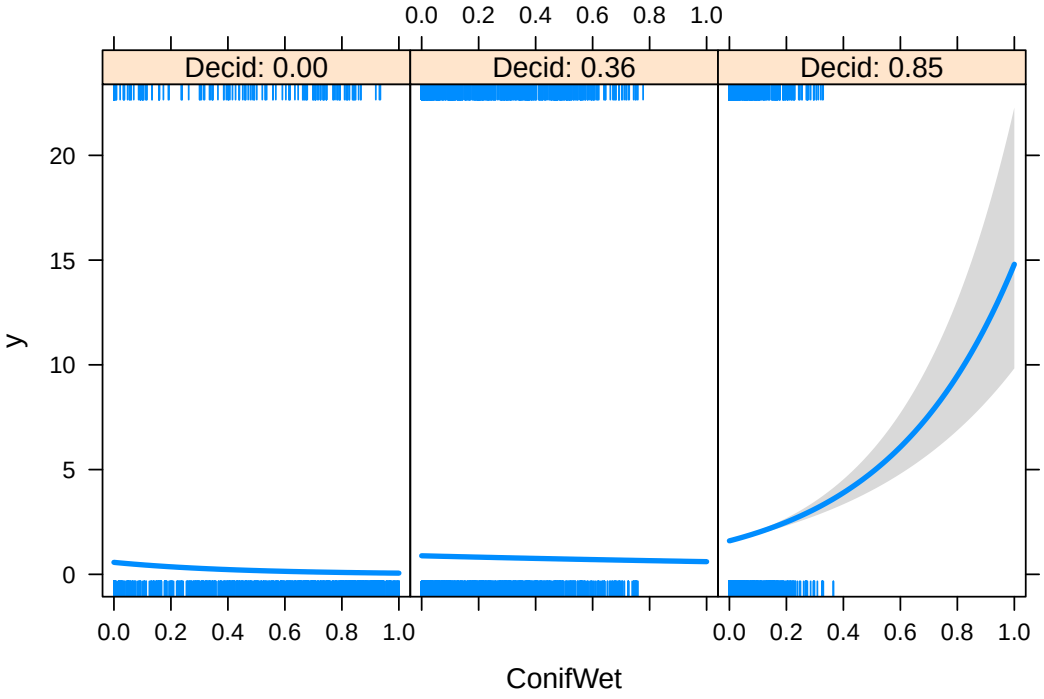
```
model.sel(mP0, mP1, mP4, mP5)
```

```
## Model selection table
##       (Int)   Dcd     CnW CnW:Dcd df logLik  AICc  delta weight
## mP5 -0.5604 1.213 -2.3120   5.346  4  -5271 10549    0.0      1
## mP4 -0.7014 1.622 -0.9785          3  -5392 10791  241.2      0
## mP1 -1.1640 2.134                  2  -5442 10887  337.7      0
## mP0 -0.1243                        1  -6285 12573 2023.6      0
## Models ranked by AICc(x)
```

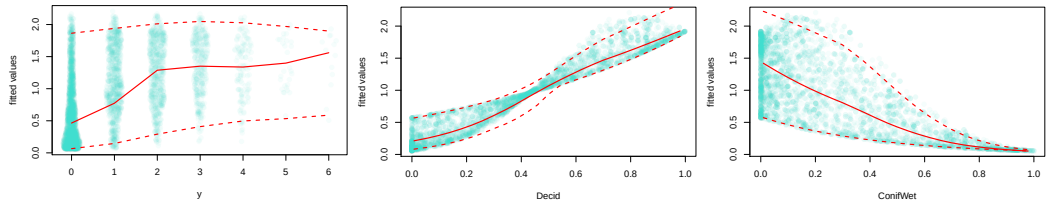
The model with the interaction is best supported, but how do we make sense of this relationship? We can't easily visualize it in a single plot. We can either

1. fix all variables (at their mean/meadian) and see how the response is changing along a single variable: this is called a *conditional* effect (conditional on fixing other variables), this is what `visreg::visreg` does;
2. or plot the fitted values against the predictor variables, this is called a *marginal* effects, and this is what `ResourceSelection::mep` does.

```
visreg(mP5, scale="response", xvar="ConifWet", by="Decid")
```

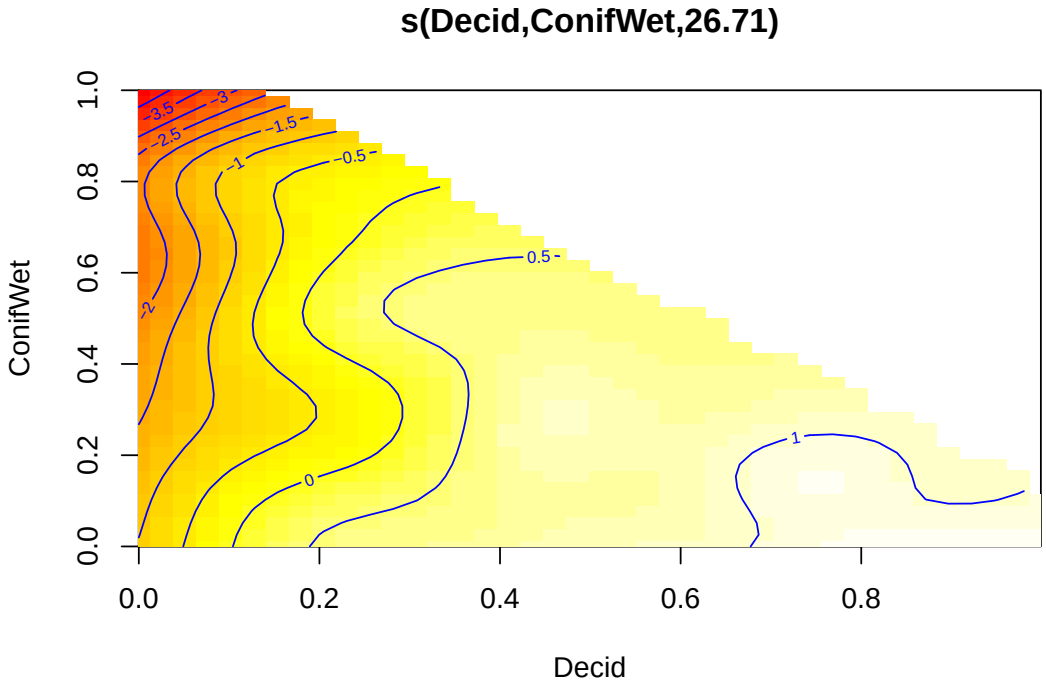



```
mep(mP5)
```



Let's use GAM to fit a bivariate spline:

```
mGAM2 <- mgcv::gam(y ~ s(Decid, ConifWet), data=x, family=poisson)
plot(mGAM2, scheme=2, rug=FALSE)
```



Final battle of Poisson models:

```
model.sel(mP0, mP1, mP12, mP13, mP14, mP2, mP3, mP4, mP5, mGAM, mGAM2)
```

```
## Model selection table
##         (Int)    Dcd  Dcd^2  Dcd^3  Dcd^4   DEC HAB     CnW CnW:Dcd s(Dcd)
## mGAM2 -0.6251                                                            
## mGAM  -0.5606                                                          +
## mP14  -2.6640 16.640 -38.60 41.470 -16.31                              
## mP13  -2.3910 11.400 -16.31  8.066                                     
## mP12  -1.9240  6.259  -3.97                                           
## mP5   -0.5604  1.213                                -2.3120   5.346     
## mP4   -0.7014  1.622                                -0.9785           
## mP1   -1.1640  2.134                                                  
## mP3   -1.3860                                    +                     
## mP2   -0.8577                              1.216                       
## mP0   -0.1243                                                          
##       s(Dcd,CnW) class df logLik  AICc   delta weight
## mGAM2          +   gam 27  -5160 10376    0.00      1
## mGAM               gam  9  -5209 10438   61.11      0
```

```
## mP14                glm  5  -5215 10441   64.42      0
## mP13                glm  4  -5226 10461   84.53      0
## mP12                glm  3  -5269 10544  167.14      0
## mP5                 glm  4  -5271 10549  172.99      0
## mP4                 glm  3  -5392 10791  414.18      0
## mP1                 glm  2  -5442 10887  510.71      0
## mP3                 glm  9  -5572 11162  785.06      0
## mP2                 glm  2  -5621 11246  869.35      0
## mP0                 glm  1  -6285 12573 2196.58      0
## Models ranked by AICc(x)
```

```
R2dev(mP0, mP1, mP12, mP13, mP14, mP2, mP3, mP4, mP5, mGAM, mGAM2)
```

```
##             R2   R2adj Deviance    Dev0    DevR     df0     dfR p_value
## mP0       0.00    0.00     0.00 7424.78 7424.78 4568.00 4568.00 < 2e-16 ***
## mP1       0.23    0.23  1687.87 7424.78 5736.91 4568.00 4567.00 < 2e-16 ***
## mP12      0.27    0.27  2033.44 7424.78 5391.34 4568.00 4566.00 < 2e-16 ***
## mP13      0.29    0.28  2118.06 7424.78 5306.72 4568.00 4565.00 7.6e-14 ***
## mP14      0.29    0.29  2140.17 7424.78 5284.61 4568.00 4564.00 3.4e-13 ***
## mP2       0.18    0.18  1329.23 7424.78 6095.55 4568.00 4567.00 < 2e-16 ***
## mP3       0.19    0.19  1427.55 7424.78 5997.23 4568.00 4560.00 < 2e-16 ***
## mP4       0.24    0.24  1786.40 7424.78 5638.38 4568.00 4566.00 < 2e-16 ***
## mP5       0.27    0.27  2029.60 7424.78 5395.18 4568.00 4565.00 < 2e-16 ***
## mGAM      0.29    0.29  2152.63 7424.78 5272.15 4568.00 4559.00 5.5e-13 ***
## mGAM2     0.30    0.30  2250.35 7424.78 5174.43 4568.00 4539.00 8.4e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Of course, the most complex model wins but the Chi-square test is still significant (indicating lack of fit). Let's try different error distribution.

3.11 Different error distributions

We will use the 2-variable model with interaction:

```
mP <- glm(y ~ Decid * ConifWet, data=x, family=poisson)
```

Let us try the Negative Binomial distribution first. This distribution is related

to Binomial experiments (number of trials required to get a fixed number of successes given a binomial probability). It can also be derived as a mixture of Poisson and Gamma distributions (see Wikipedia), which is a kind of hierarchical model. In this case, the Gamma distribution acts as an i.i.d. random effect for the intercept: $Y_i \sim Poisson(\lambda_i)$, $\lambda_i \sim Gamma(e^{\beta_0+\beta_1 x_{1i}}, \gamma)$, where γ is the Gamma variance.

The Negative Binomial variance (using the parametrization common in R functions) is a function of the mean and the scale: $V(\mu) = \mu + \mu^2/\theta$.

```
mNB <- glm.nb(y ~ Decid * ConifWet, data=x)
summary(mNB)
```

```
##
## Call:
## glm.nb(formula = y ~ Decid * ConifWet, data = x, init.theta = 3.5900635,
##     link = log)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.860  -0.985  -0.451   0.317   3.803
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -0.5905     0.0630   -9.38   <2e-16 ***
## Decid            1.2459     0.0892   13.97   <2e-16 ***
## ConifWet        -2.3545     0.1605  -14.67   <2e-16 ***
## Decid:ConifWet   5.6945     0.4009   14.20   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(3.59) family taken to be 1)
##
##     Null deviance: 6089.3  on 4568  degrees of freedom
## Residual deviance: 4387.7  on 4565  degrees of freedom
## AIC: 10440
##
## Number of Fisher Scoring iterations: 1
##
```

```
##
##                Theta:  3.590
##            Std. Err.:  0.425
##
##  2 x log-likelihood:  -10430.448
```

Next, we look at zero-inflated models. In this case, the mixture distribution is a Bernoulli distribution and a count distribution (Poisson or Negative Binomial, for example). The 0's can come from both the zero and the count distributions, whereas the >0 values can only come from the count distribution: $A_i \sim Bernoulli(\varphi)$, $Y_i \sim Poisson(A_i\lambda_i)$.

The zero part of the zero-inflated models are often parametrized as probability of zero $(1 - \varphi)$, as in the `pscl::zeroinfl` function:

```
## Zero-inflated Poisson
mZIP <- zeroinfl(y ~ Decid * ConifWet | 1, x, dist="poisson")
summary(mZIP)
```

```
##
## Call:
## zeroinfl(formula = y ~ Decid * ConifWet | 1, data = x, dist = "poisson")
##
## Pearson residuals:
##    Min     1Q Median     3Q    Max
## -1.218 -0.700 -0.339  0.378  8.951
##
## Count model coefficients (poisson with log link):
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -0.3241     0.0651   -4.98  6.5e-07 ***
## Decid            1.0700     0.0860   12.44  < 2e-16 ***
## ConifWet        -2.4407     0.1564  -15.60  < 2e-16 ***
## Decid:ConifWet   5.9373     0.3840   15.46  < 2e-16 ***
##
## Zero-inflation model coefficients (binomial with logit link):
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.613      0.103   -15.6   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Number of iterations in BFGS optimization: 12
## Log-likelihood: -5.21e+03 on 5 Df
```

```
## Zero-inflated Negative Binomial
mZINB <- zeroinfl(y ~ Decid * ConifWet | 1, x, dist="negbin")
summary(mZINB)
```

```
##
## Call:
## zeroinfl(formula = y ~ Decid * ConifWet | 1, data = x, dist = "negbin")
##
## Pearson residuals:
##    Min     1Q Median     3Q    Max
## -1.190 -0.689 -0.338  0.361  8.956
##
## Count model coefficients (negbin with log link):
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -0.3873     0.0732   -5.29  1.2e-07 ***
## Decid            1.1195     0.0911   12.29  < 2e-16 ***
## ConifWet        -2.4230     0.1589  -15.25  < 2e-16 ***
## Decid:ConifWet   5.9227     0.3963   14.94  < 2e-16 ***
## Log(theta)       2.6504     0.5305    5.00  5.9e-07 ***
##
## Zero-inflation model coefficients (binomial with logit link):
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.861      0.193   -9.64   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Theta = 14.159
## Number of iterations in BFGS optimization: 22
## Log-likelihood: -5.2e+03 on 6 Df
```

Now we compare the four different parametric models:

```
AIC(mP, mNB, mZIP, mZINB)
```

Our best model is the Zero-inflated Negative Binomial. The probability of observing a zero as part of the zero distribution is back transformed from the

zero coefficient using the inverse logit function:

```
unname(plogis(coef(mZINB, "zero"))) # P of 0
```

```
## [1] 0.1346
```

Now we use the scale parameter to visualize the variance functions for the Negative Binomial models (the 1:1 line is the Poisson model):

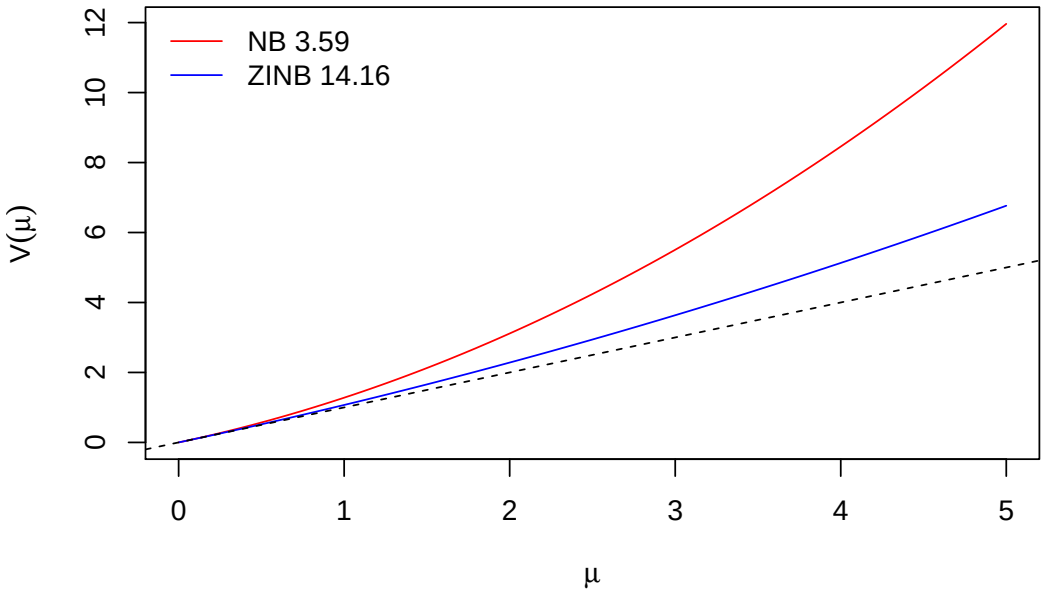
```
mNB$theta
```

```
## [1] 3.59
```

```
mZINB$theta
```

```
## [1] 14.16
```

```
mu <- seq(0, 5, 0.01)
plot(mu, mu + mu^2/mNB$theta, type="l", col=2,
  ylab=expression(V(mu)), xlab=expression(mu))
lines(mu, mu + mu^2/mZINB$theta, type="l", col=4)
abline(0,1, lty=2)
legend("topleft", bty="n", lty=1, col=c(2,4),
  legend=paste(c("NB", "ZINB"), round(c(mNB$theta, mZINB$theta), 2)))
```



Exercise
How can we interpret these different kinds of overdispersion (zero-inflation and higher than Poisson variance)?
What are some of the biological mechanisms that can contribute to the overdispersion?

It is also common practice to consider generalized linear mixed models (GLMMs) for count data. These mixed models are usually considered as Poisson-Lognormal mixtures. The simplest, so called i.i.d., case is similar to the Negative Binomial, but instead of Gamma, we have Lognormal distribution: $Y_i \sim Poisson(\lambda_i)$, $log(\lambda_i) = \beta_0 + \beta_1 x_{1i} + \epsilon_i$, $\epsilon_i \sim Normal(0, \sigma^2)$, where σ^2 is the Lognormal variance on the log scale.

We can use the `lme4::glmer` function: use `SiteID` as random effect (we have exactly n random effects).

```
mPLN1 <- glmer(y ~ Decid * ConifWet + (1 | SiteID), data=x, family=poisson)
summary(mPLN1)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: y ~ Decid * ConifWet + (1 | SiteID)
##    Data: x
##
##      AIC      BIC   logLik deviance df.resid
##    10423    10455    -5206    10413     4564
##
## Scaled residuals:
##    Min     1Q Median     3Q    Max
## -1.150 -0.629 -0.288  0.418  5.469
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  SiteID (Intercept) 0.294    0.542
## Number of obs: 4569, groups:  SiteID, 4569
##
```

```
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -0.7518     0.0675   -11.1   <2e-16 ***
## Decid            1.2847     0.0920    14.0   <2e-16 ***
## ConifWet        -2.3380     0.1625   -14.4   <2e-16 ***
## Decid:ConifWet   5.6326     0.4118    13.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) Decid  ConfWt
## Decid       -0.895
## ConifWet    -0.622  0.644
## Decid:CnfWt  0.176 -0.379 -0.700
```

Note
The number of unknowns we have to somehow estimate is now more than the number of observations we have. How is that possible?

Alternatively, we can use `SurveyArea` as a grouping variable. We have now $m < n$ random effects, and survey areas can be seen as larger landscapes within which the sites are clustered: $Y_i j \sim Poisson(\lambda_i j)$, $log(\lambda_i j) = \beta_0 + \beta_1 x_{1ij} + \epsilon_i$, $\epsilon_i \sim Normal(0, \sigma^2)$. The index i $(i = 1, ..., m)$ defines the cluster (survey area), the j $(j = 1, ..., n_i)$ defines the sites within survey area i $(n = \sum_{i=1}^{m} n_i)$.

```
mPLN2 <- glmer(y ~ Decid * ConifWet + (1 | SurveyArea), data=x, family=poisson)
summary(mPLN2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: y ~ Decid * ConifWet + (1 | SurveyArea)
##    Data: x
##
##      AIC      BIC   logLik deviance df.resid
##    10021    10053    -5006    10011     4564
```

```
##
## Scaled residuals:
##    Min     1Q Median     3Q    Max
## -1.739 -0.643 -0.320  0.355  6.535
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  SurveyArea (Intercept) 0.295    0.543
## Number of obs: 4569, groups:  SurveyArea, 271
##
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -0.7459     0.0783   -9.53   <2e-16 ***
## Decid            1.1967     0.0984   12.16   <2e-16 ***
## ConifWet        -2.3213     0.1686  -13.77   <2e-16 ***
## Decid:ConifWet   5.5346     0.3977   13.92   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) Decid  ConfWt
## Decid       -0.808
## ConifWet    -0.610  0.628
## Decid:CnfWt  0.162 -0.325 -0.670
```

In the battle of distributions (keeping the linear predictor part the same) the clustered GLMM was best supported:

```
tmp <- AIC(mP, mNB, mZIP, mZINB, mPLN1, mPLN2)
tmp$delta_AIC <- tmp$AIC - min(tmp$AIC)
tmp[order(tmp$AIC),]
```

Exercise
What are some of the biological mechanisms that can lead to the clustered GLMM bi be the best model?

3.12 Count duration effects

Let's change gears a bit now, and steer closer to the main focus of this book. We want to account for methodological differences among samples. One aspect of mathodologies involve variation in total counting duration. We'll now inspect what that does to our observations.

First, we create a list of matrices where counts are tabulated by surveys and time intervals for each species:

```
ydur <- Xtab(~ SiteID + Dur + SpeciesID ,
  josm$counts[josm$counts$DetectType1 != "V",])
```

We use the same species (spp) as before and create a data frame indluring the cumulative counts during 3, 5, and 10 minutes:

```
y <- as.matrix(ydur[[spp]])
head(y)
```

```
##         0-3min 3-5min 5-10min
## CL10102      3      0       0
## CL10106      0      0       0
## CL10108      0      0       0
## CL10109      2      0       1
## CL10111      2      0       0
## CL10112      2      0       0
```

```
colMeans(y) # mean count of new individuals
```

```
##  0-3min  3-5min 5-10min
## 0.67367 0.09346 0.11600
```

```
cumsum(colMeans(y)) # cumulative counts
```

```
##  0-3min  3-5min 5-10min
##  0.6737  0.7671  0.8831
```

```
x <- data.frame(
  josm$surveys,
  y3=y[,"0-3min"],
  y5=y[,"0-3min"]+y[,"3-5min"],
```

```
  y10=rowSums(y))

table(x$y3)
```

```
##
##    0    1    2    3    4    5    6
## 2768  922  576  226   61   14    2
```

```
table(x$y5)
```

```
##
##    0    1    2    3    4    5    6
## 2643  894  632  285   87   24    4
```

```
table(x$y10)
```

```
##
##    0    1    2    3    4    5    6
## 2493  883  656  363  132   29   13
```

If we fit single-predictor GLMs to these 3 responses, we get different fitted values, consistent with our mean counts:

```
m3 <- glm(y3 ~ Decid, data=x, family=poisson)
m5 <- glm(y5 ~ Decid, data=x, family=poisson)
m10 <- glm(y10 ~ Decid, data=x, family=poisson)
mean(fitted(m3))
```

```
## [1] 0.6737
```

```
mean(fitted(m5))
```

```
## [1] 0.7671
```

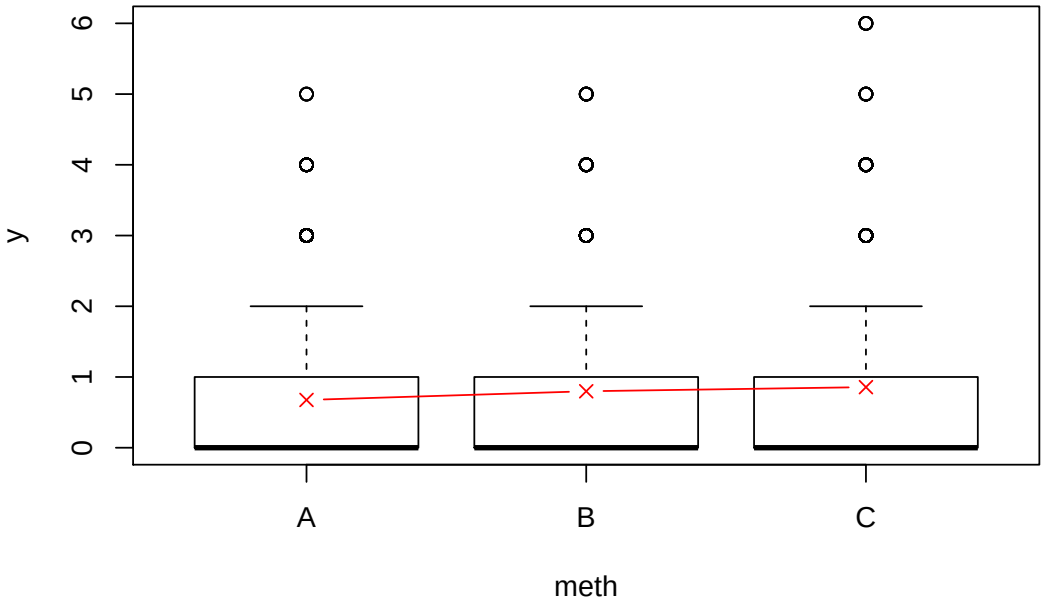
```
mean(fitted(m10))
```

```
## [1] 0.8831
```

Using the multiple time interval data, we can pretend that we have a mix of methodologies with respect to count duration:

```
set.seed(1)
x$meth <- as.factor(sample(c("A", "B", "C"), nrow(x), replace=TRUE))
```

```
x$y <- x$y3
x$y[x$meth == "B"] <- x$y5[x$meth == "B"]
x$y[x$meth == "C"] <- x$y10[x$meth == "C"]
boxplot(y ~ meth, x)
sb <- sum_by(x$y, x$meth)
points(1:3, sb[,1]/sb[,2], col=2, type="b", pch=4)
```



We can estimate the effect of the methodology:

```
mm <- glm(y ~ meth - 1, data=x, family=poisson)
summary(mm)
```

```
##
## Call:
## glm(formula = y ~ meth - 1, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.309  -1.263  -1.162   0.369   3.616
##
## Coefficients:
##       Estimate Std. Error z value Pr(>|z|)
```

```
## methA  -0.3929     0.0309  -12.70  < 2e-16 ***
## methB  -0.2255     0.0289   -7.79  6.6e-15 ***
## methC  -0.1550     0.0277   -5.60  2.1e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 7225.2  on 4569  degrees of freedom
## Residual deviance: 6941.8  on 4566  degrees of freedom
## AIC: 11657
##
## Number of Fisher Scoring iterations: 6
```

```
exp(coef(mm))
```

```
##  methA  methB  methC
## 0.6751 0.7981 0.8564
```

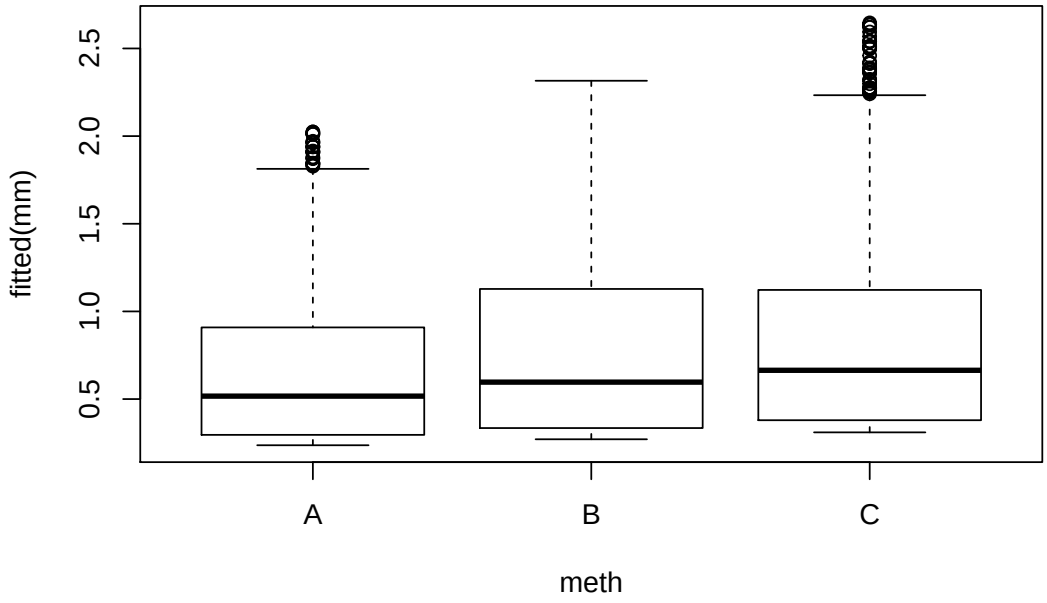
Or the effect of the continuous predictor and the method (discrete):

```
mm <- glm(y ~ Decid + meth, data=x, family=poisson)
summary(mm)
```

```
##
## Call:
## glm(formula = y ~ Decid + meth, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.278  -0.939  -0.736   0.457   4.201
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.4416     0.0457  -31.56  < 2e-16 ***
## Decid         2.1490     0.0574   37.43  < 2e-16 ***
## methB         0.1347     0.0424    3.18   0.0015 **
## methC         0.2705     0.0415    6.51  7.3e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 6976.3  on 4568  degrees of freedom
## Residual deviance: 5442.7  on 4565  degrees of freedom
## AIC: 10159
##
## Number of Fisher Scoring iterations: 6
```

```
boxplot(fitted(mm) ~ meth, x)
```




```
exp(coef(mm))
```

```
## (Intercept)       Decid       methB       methC
##      0.2365      8.5766      1.1442      1.3106
```

The fixed effects adjusts the means well:

```
cumsum(colMeans(y))
```

```
##  0-3min  3-5min 5-10min
##  0.6737  0.7671  0.8831
```

```
mean(y[,1]) * c(1, exp(coef(mm))[3:4])
```

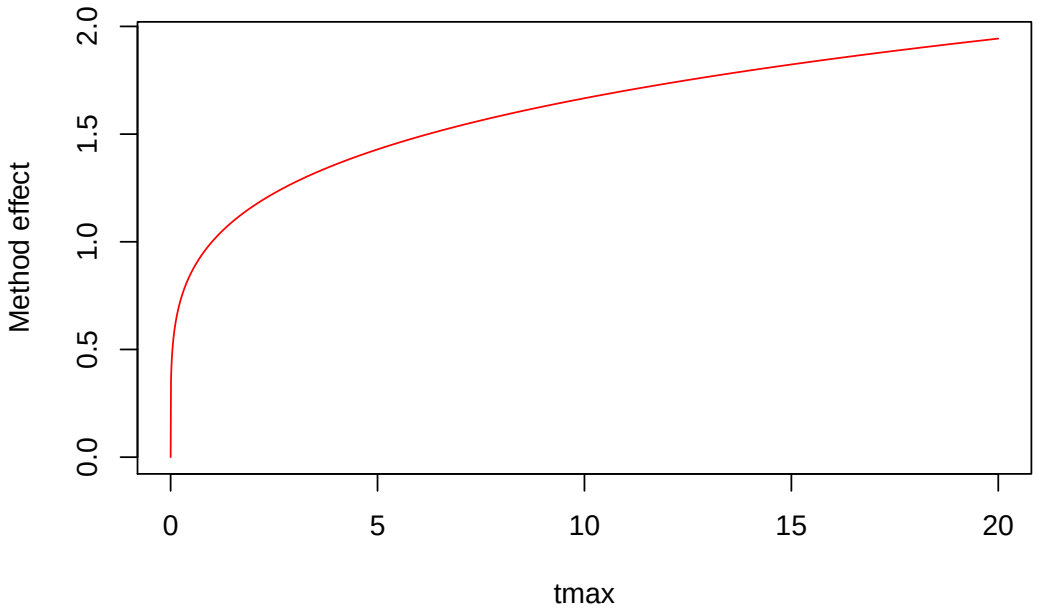
```
##         methB  methC
## 0.6737 0.7708 0.8829
```

But it is all relative, depends on reference methodology/protocol. The problem is, we can't easily extrapolate to a methodology with count duration of 12 minutes, or interpolate to a mathodology with count duration of 2 or 8 minutes. We need somehow to express time expediture in minutes to make that work. Let's try something else:

```
x$tmax <- c(3, 5, 10)[as.integer(x$meth)]
mm <- glm(y ~ Decid + I(log(tmax)), data=x, family=poisson)
summary(mm)
```

```
##
## Call:
## glm(formula = y ~ Decid + I(log(tmax)), family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.284  -0.939  -0.731   0.453   4.195
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.6777     0.0702  -23.91  < 2e-16 ***
## Decid          2.1504     0.0574   37.48  < 2e-16 ***
## I(log(tmax))   0.2218     0.0340    6.53  6.7e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 6976.3  on 4568  degrees of freedom
## Residual deviance: 5443.0  on 4566  degrees of freedom
## AIC: 10158
##
## Number of Fisher Scoring iterations: 6
```

```
tmax <- seq(0, 20, 0.01)
plot(tmax, exp(log(tmax) * coef(mm)[3]), type="l",
  ylab="Method effect", col=2)
```



Now we are getting somewhere. But still, this function keep increasing monotonically.

Exercise
What kind of function would we need and why?
What is the underlying biological mechanism?

3.13 Count radius effects

Before solving the count duration issue, let us look at the effect of survey area. We get a similar count breakdown, but now by distance band:

```
ydis <- Xtab(~ SiteID + Dis + SpeciesID ,
  josm$counts[josm$counts$DetectType1 != "V",])
```

```
y <- as.matrix(ydis[[spp]])
head(y)
```

```
##         0-50m 50-100m 100+m
## CL10102     1       2     0
## CL10106     0       0     0
## CL10108     0       0     0
## CL10109     1       2     0
## CL10111     1       0     1
## CL10112     0       2     0
```

```
colMeans(y) # mean count of new individuals
```

```
##   0-50m 50-100m   100+m
## 0.29241 0.49223 0.09849
```

```
cumsum(colMeans(y)) # cumulative counts
```

```
##   0-50m 50-100m   100+m
##  0.2924  0.7846  0.8831
```

```
x <- data.frame(
  josm$surveys,
  y50=y[,"0-50m"],
  y100=y[,"0-50m"]+y[,"50-100m"])

table(x$y50)
```

```
##
##    0    1    2    3    4    5
## 3521  792  228   25    2    1
```

```
table(x$y100)
```

```
##
##    0    1    2    3    4    5    6
## 2654  833  647  316   92   20    7
```

We don't consider the unlimited distance case, because the survey area there is unknown (although we will ultimately address this problem mater). We

compare the counts within the 0-50 and 0-100 m circles:

```
m50 <- glm(y50 ~ Decid, data=x, family=poisson)
m100 <- glm(y100 ~ Decid, data=x, family=poisson)
mean(fitted(m50))
```

```
## [1] 0.2924
```

```
mean(fitted(m100))
```

```
## [1] 0.7846
```

```
coef(m50)
```

```
## (Intercept)       Decid
##      -2.265       2.126
```

```
coef(m100)
```

```
## (Intercept)       Decid
##      -1.327       2.209
```

3.14 Offsets

Offsets are constant terms in the linear predictor, e.g. $log(\lambda_i) = \beta_0 + \beta_1 x_{1i} + o_i$, where o_i is an offset. In the survey area case, an offset might be the log of area surveyed. The logic for this is based on point processes: intensity is a linear function of area under a homogeneous Poisson point process. So we can assume that $o_i = log(A_i)$, where A stands for area.

Let's see if using area as offset makes our models comparable:

```
m50 <- glm(y50 ~ Decid, data=x, family=poisson,
  offset=rep(log(0.5^2*pi), nrow(x)))
m100 <- glm(y100 ~ Decid, data=x, family=poisson,
  offset=rep(log(1^2*pi), nrow(x)))
coef(m50)
```

```
## (Intercept)       Decid
##      -2.024       2.126
```

```
coef(m100)
```

```
## (Intercept)       Decid
##      -2.471       2.209
```

```
mean(exp(model.matrix(m50) %*% coef(m50)))
```

```
## [1] 0.3723
```

```
mean(exp(model.matrix(m100) %*% coef(m100)))
```

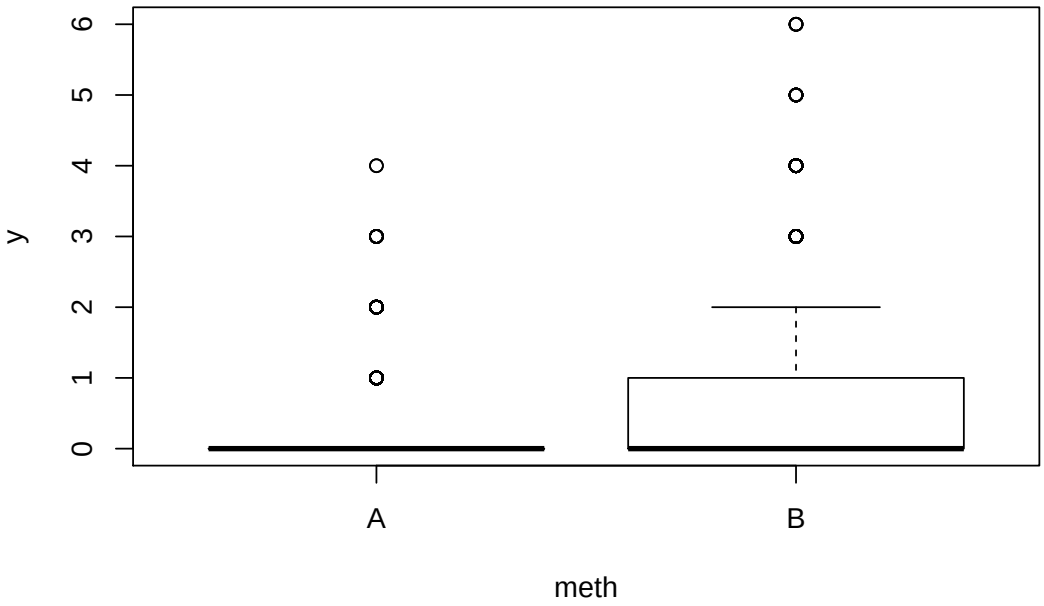
```
## [1] 0.2498
```

These coefficients and mean predictions are much closer to each other, but something else is going on.

Exercise
Can you guess why we cannot make abundances comparable using log area as as offset?

We pretend again, that survey area varies in our data set:

```
set.seed(1)
x$meth <- as.factor(sample(c("A", "B"), nrow(x), replace=TRUE))
x$y <- x$y50
x$y[x$meth == "B"] <- x$y100[x$meth == "B"]
boxplot(y ~ meth, x)
```



Methodology effect:

```
mm <- glm(y ~ meth - 1, data=x, family=poisson)
summary(mm)
```

```
##
## Call:
## glm(formula = y ~ meth - 1, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.256  -1.256  -0.775   0.228   3.731
##
## Coefficients:
##       Estimate Std. Error z value Pr(>|z|)
## methA  -1.2040     0.0375  -32.10   <2e-16 ***
## methB  -0.2370     0.0240   -9.87   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
```

```
##     Null deviance: 7299.4  on 4569  degrees of freedom
## Residual deviance: 5587.9  on 4567  degrees of freedom
## AIC: 9066
##
## Number of Fisher Scoring iterations: 6
```

```
exp(coef(mm))
```

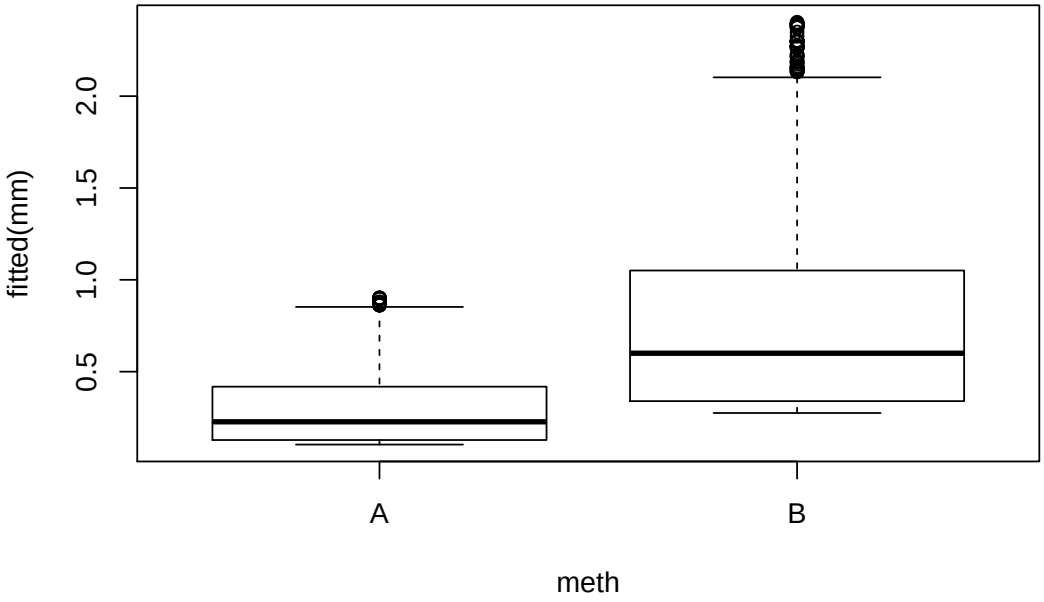
```
## methA methB
## 0.300 0.789
```

Predictor and method effects:

```
mm <- glm(y ~ Decid + meth, data=x, family=poisson)
summary(mm)
```

```
##
## Call:
## glm(formula = y ~ Decid + meth, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.185  -0.847  -0.584   0.274   4.347
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.2719     0.0554   -41.0   <2e-16 ***
## Decid         2.1706     0.0690    31.4   <2e-16 ***
## methB         0.9804     0.0445    22.0   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 6110.2  on 4568  degrees of freedom
## Residual deviance: 4531.0  on 4566  degrees of freedom
## AIC: 8011
##
## Number of Fisher Scoring iterations: 6
```

```
boxplot(fitted(mm) ~ meth, x)
```




```
exp(coef(mm))
```

```
## (Intercept)       Decid       methB
##      0.1031      8.7632      2.6654
```

```
cumsum(colMeans(y))[1:2]
```

```
##   0-50m 50-100m
##  0.2924  0.7846
```

```
mean(y[,1]) * c(1, exp(coef(mm))[3])
```

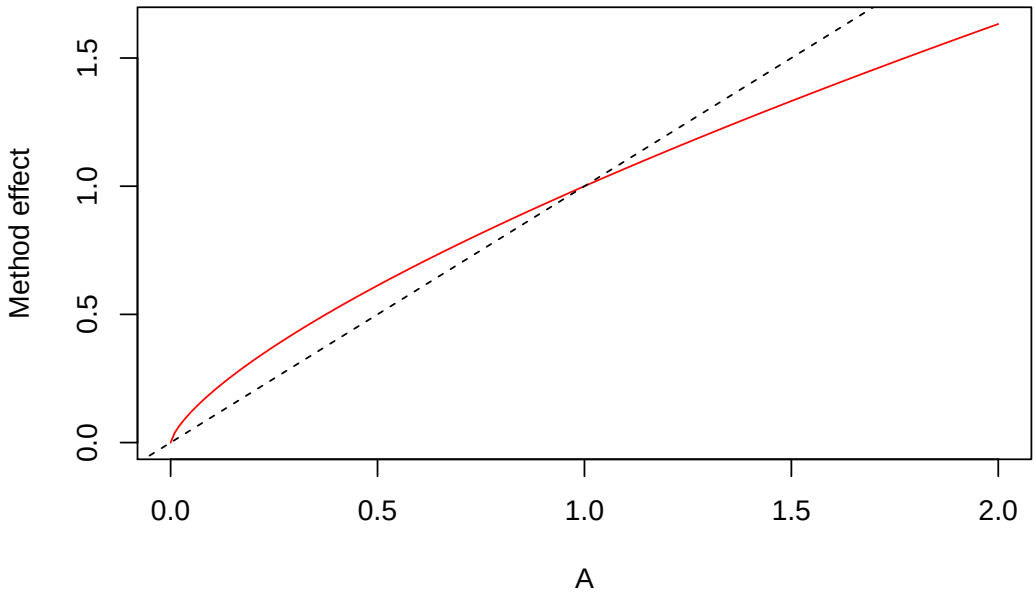
```
##         methB
## 0.2924 0.7794
```

Use log area as continuous predictor: we would expect a close to 1:1 relationship on the abundance scale.

```
x$logA <- log(ifelse(x$meth == "A", 0.5, 1)^2*pi)
mm <- glm(y ~ Decid + logA, data=x, family=poisson)
summary(mm)
```

```
##
## Call:
## glm(formula = y ~ Decid + logA, family = poisson, data = x)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.185  -0.847  -0.584   0.274   4.347
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.1011     0.0513   -40.9   <2e-16 ***
## Decid         2.1706     0.0690    31.4   <2e-16 ***
## logA          0.7072     0.0321    22.0   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 6110.2  on 4568  degrees of freedom
## Residual deviance: 4531.0  on 4566  degrees of freedom
## AIC: 8011
##
## Number of Fisher Scoring iterations: 6
```

```
A <- seq(0, 2, 0.01) # in ha
plot(A, exp(log(A) * coef(mm)[3]), type="l",
  ylab="Method effect", col=2)
abline(0, 1, lty=2)
```



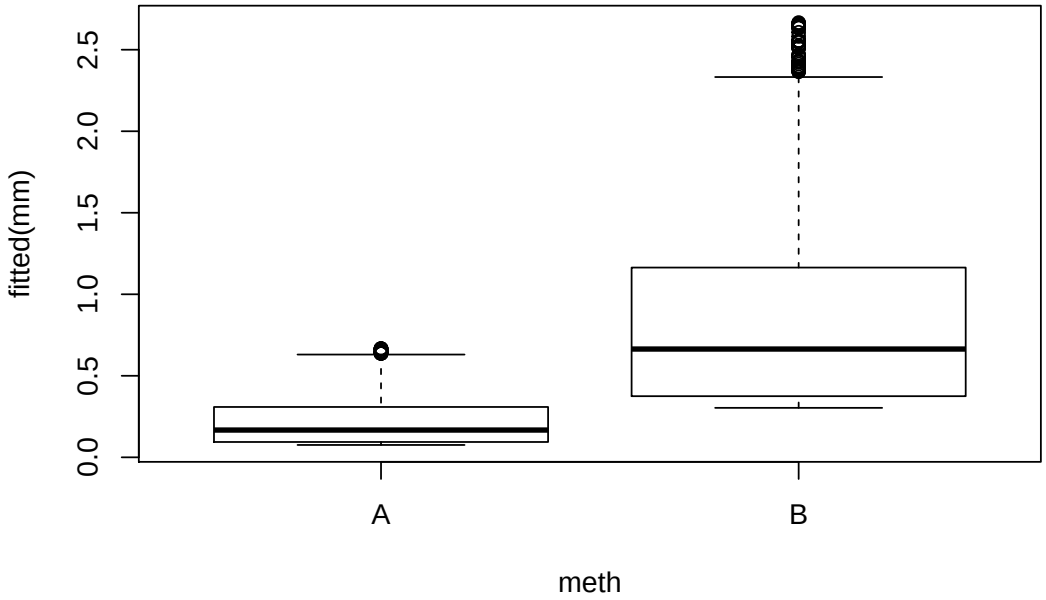
The offset forces the relationship to be 1:1 (it is like fixing the `logA` coefficient to be 1):

```
mm <- glm(y ~ Decid, data=x, family=poisson, offset=x$logA)
summary(mm)
```

```
##
## Call:
## glm(formula = y ~ Decid, family = poisson, data = x, offset = x$logA)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.302  -0.836  -0.512   0.260   4.219
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.3374     0.0453   -51.6   <2e-16 ***
## Decid         2.1758     0.0690    31.5   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
```

```
##
##     Null deviance: 5671.1  on 4568  degrees of freedom
## Residual deviance: 4609.2  on 4567  degrees of freedom
## AIC: 8087
##
## Number of Fisher Scoring iterations: 6
```

```
boxplot(fitted(mm) ~ meth, x)
```




```
cumsum(colMeans(y))[1:2]
```

```
##   0-50m 50-100m
##  0.2924  0.7846
```

```
c(0.5, 1)^2*pi * mean(exp(model.matrix(mm) %*% coef(mm))) # /ha
```

```
## [1] 0.2200 0.8798
```

Exercise
Why did we get a `logA` coefficient that was less than 1 when theoretically we should have gotten 1?

Predictions using offsets in `glm` can be tricky. The safest way is to use the matrix product (`exp(model.matrix(mm) %*% coef(mm) + <offset>)`). We can often omit the offset, e.g. in the log area case we can express the prediction per unit area. If the unit is 1 ha, as in our case, log(1)=0, which means the mean abundance per unit area can be calculated by omitting the offsets all together.

Chapter 4

Behavioral Complexities

4.1 Introduction

We have reviewed so far how to fit *naive* models to estimate the expected value of the observed counts, λ. So what is this λ? Here are some deifinitions for further discussion:

- **relative abundance**: λ without any reference to nuisance variables, but possibly standardized by design, or nuisance variables used as fixed effects,
- **abundance**: $N = \lambda/C$, C is a correction factor and N refers to the number of individuals within the area surveyed – the problem is that we cannot measure this directly (this is a latent variable), moreover the survey area is also often unknown (i.e. for unlimited distance counts),
- **occupancy**: the probability that the survey area is occupied, this is really equivalent to the indicator function $N > 0$,
- **density** $D = N/A = \lambda/AC$, abundance per unit area – same problems as above: both N and A are unknowns.

Our objective in the following chapters is to work out the details of estimating abundance and density in some clever ways through learning about the nature of the mechanisms contributing to C.

4.2 Prerequisites

```
library(bSims)                  # simulations
library(detect)                 # multinomial models
load("_data/josm/josm.rda")     # JOSM data
```

4.3 Birds in the forest

Build a landscape: extent is given in 100 m units

```
(l <- bsims_init(extent=10))
```

```
## bSims landscape
##   1 km x 1 km
##   stratification: H
```

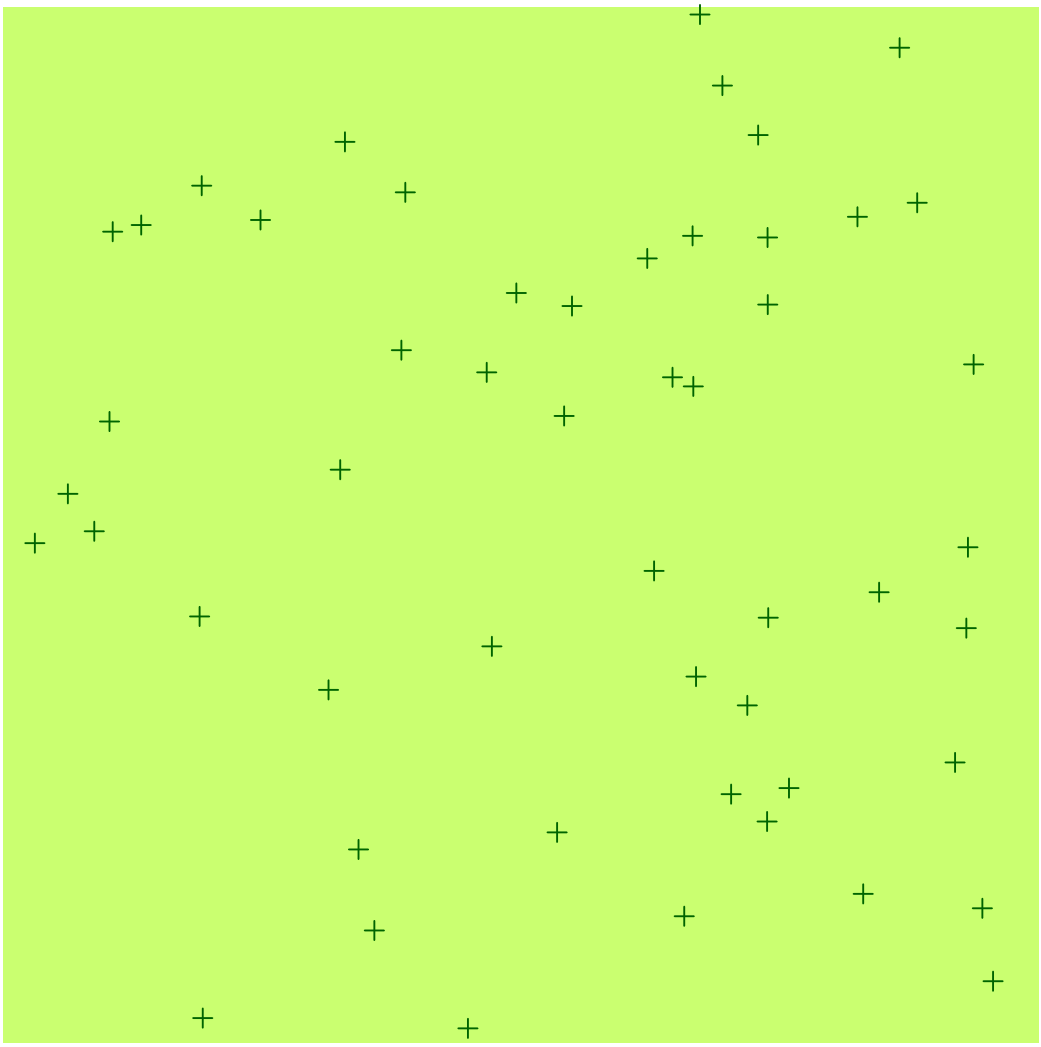
```
plot(l)
```

We have a 100 ha landscape that we populate with birds, 1 bird / ha using a Poisson spatial point process. As a result, we have N birds in the landscape, $N \sim Poisson(\lambda)$, $\lambda = DA$:

```
set.seed(1)
(a <- bsims_populate(l, density=0.5))
```

```
## bSims population
##   1 km x 1 km
##   stratification: H
##   total abundance: 52
```

```
plot(a)
```



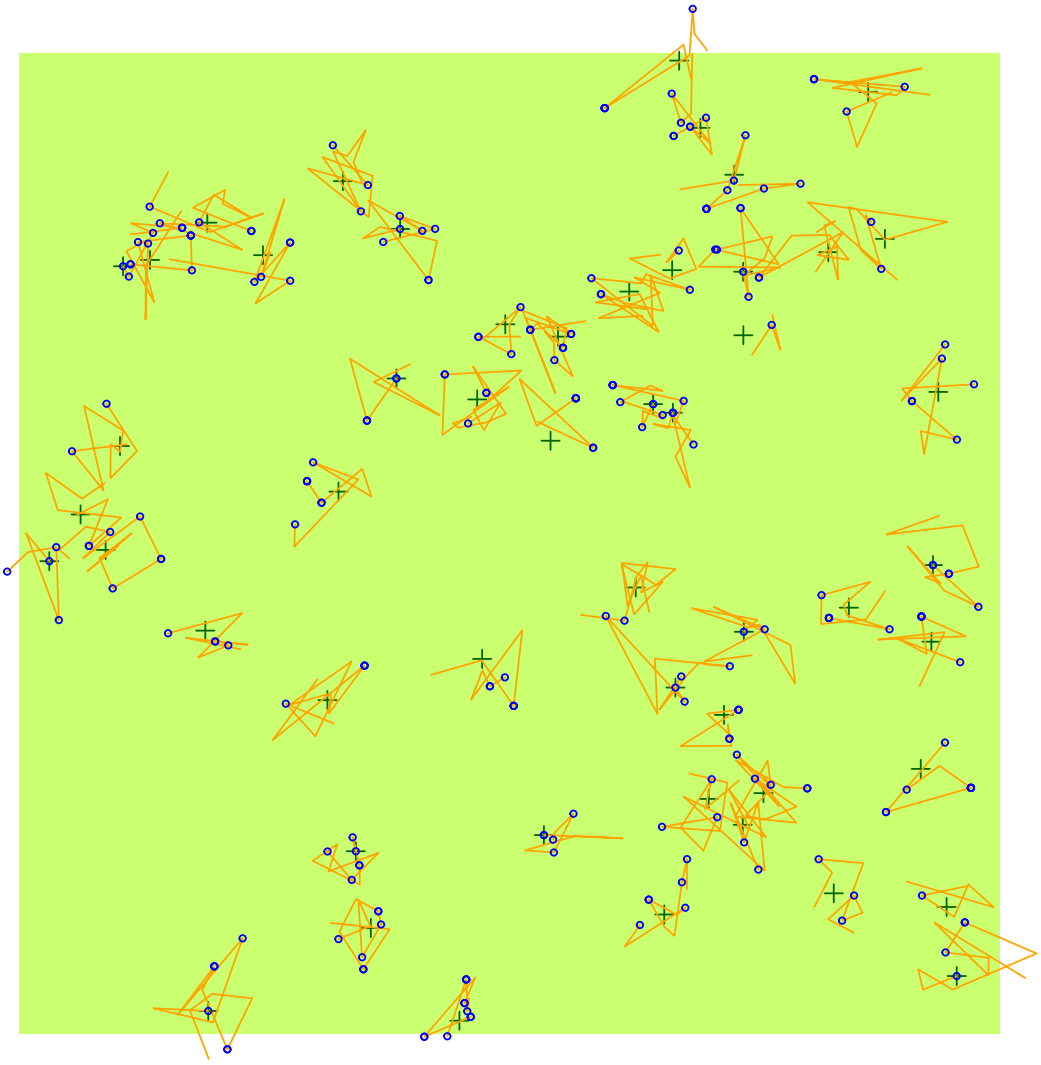
The locations can be seen as nest locations (`a$nests` stores the locations). But birds don't just stay put in one place. They move and vocalize:

```
(b <- bsims_animate(a,
  vocal_rate=0.5, duration=10,
  move_rate=1, movement=0.25))
```

```
## bSims events
```

```
##   1 km x 1 km
##   stratification: H
##   total abundance: 52
##   duration: 10 min
```

```
plot(b)
```



The `get_events` function, as the name implies, extracts the events: movements (`$v` is 0) and vocalizations (`$v` is 1) alike, unless filtered for vocalization events only. Besides the coordinates, we also have the time of event (`$t`) and

the individual identifier (`$i` linking to the rows of the `b$nests` table):

```
e <- get_events(b, event_type="both")
head(e)
v <- get_events(b, event_type="vocal")
head(v)
```

4.4 Survival model

Survival models assess time-to-event data which is often censored (some event has not occurred at the time the data collection ended).

Event time (T) is a continuous random variable. In the simplest case, its probability density function is the Exponential distribution: $f(t) = \phi e^{-t\phi}$. The corresponding cumulative distribution function is: $F(t) = \int_0^t f(t)dt = 1 - e^{-t\phi}$, giving the probability that the event has occurred by duration t and we will refer to this probability as p_t. The parameter ϕ is the rate of the Exponential distribution with mean $1/\phi$ and variance $1/\phi^2$.

In survival model, the complement of $F(t)$ is called the *survival function* ($S(t) = 1 - F(t)$, $S(0) = 1$), which gives the probability that the event has not occurred by duration t. The the *hazard function* ($\lambda(t) = f(t)/S(t)$) which defines the instantaneous rate of occurrence of the event (the density of events at t divided by the probability of surviving). The cumulative hazard (cumulative risk) the sum of the risks between doration 0 and t ($\Lambda(t) = \int_0^t \lambda(t)dt$).

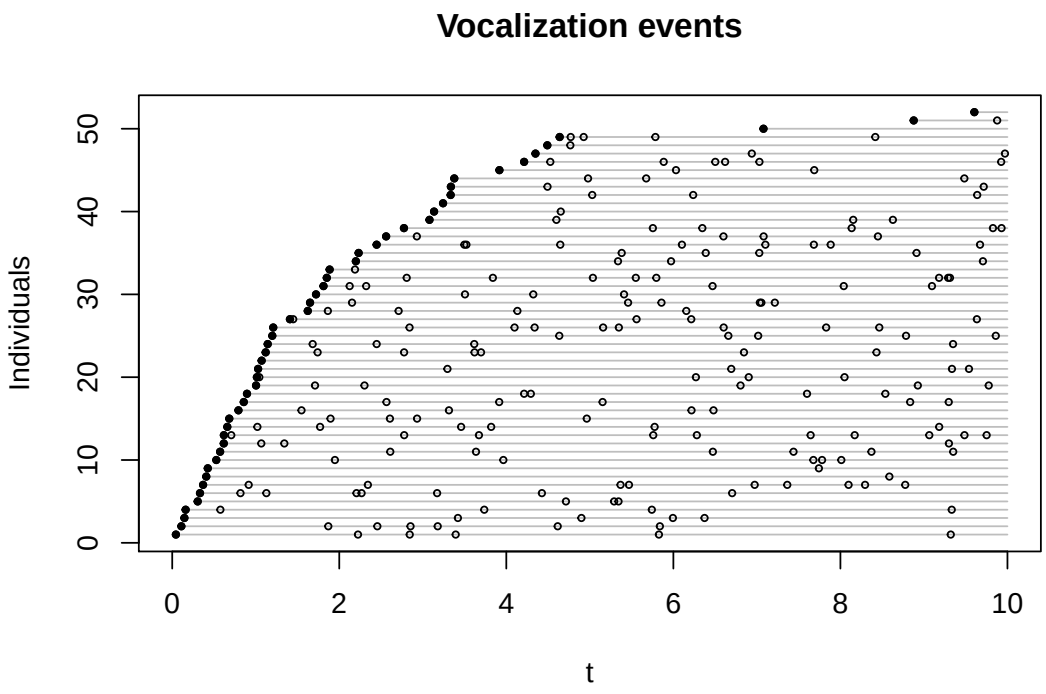
The simplest survival distribution assumes constant risk over time ($\lambda(t) = \phi$), which corresponds to the Exponential distribution. The Exponential distribution also happens to describe the lengths of the inter-event times in a homogeneous Poisson process (events are independent, 'memory-less' process).

4.5 Vocalization events

Event times in our bSims example follow a Poisson process with rate ϕ (`vocal_rate`) within `duration` $t = 10$ minutes.

Let's subset the vocalization events to include the time of first detections for each individual (`v1`). The estimated rate should match our setting, the plot shows the Exponential probability density function on top of the event times:

```
v1 <- v[!duplicated(v$i),]

tmp <- v1
tmp$o <- seq_len(nrow(v1))
plot(o ~ t, tmp, type="n", ylab="Individuals",
  main="Vocalization events",
  ylim=c(1, nrow(b$nests)), xlim=c(0,10))
for (i in tmp$o) {
  tmp2 <- v[v$i == v1$i[i],]
  lines(c(tmp2$t[1], 10), c(i,i), col="grey")
  points(tmp2$t, rep(i, nrow(tmp2)), cex=0.5)
  points(tmp2$t[1], i, pch=19, cex=0.5)
}
```



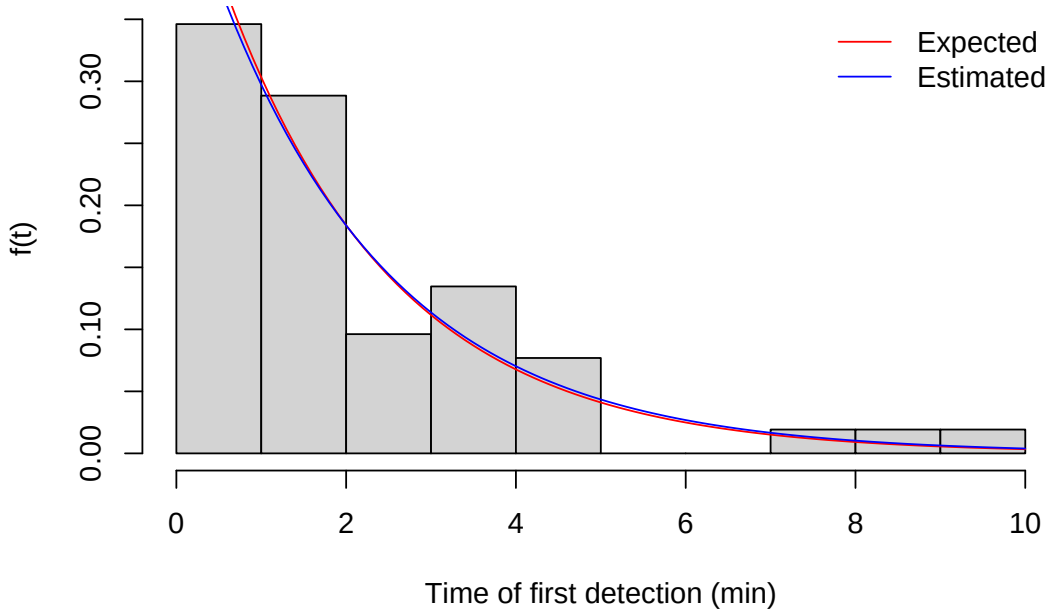

```
(phi <- b$vocal_rate[1])
```

```
## [1] 0.5
```

```
(phi_hat <- fitdistr(v1$t, "exponential")$estimate)
```

```
##   rate
## 0.4808
```

```
hist(v1$t, xlab="Time of first detection (min)", freq=FALSE, main="",
  col="lightgrey", ylab="f(t)")
curve(dexp(x, phi), add=TRUE, col=2)
curve(dexp(x, phi_hat), add=TRUE, col=4)
legend("topright", bty="n", lty=1, col=c(2,4),
  legend=c("Expected", "Estimated"))
```

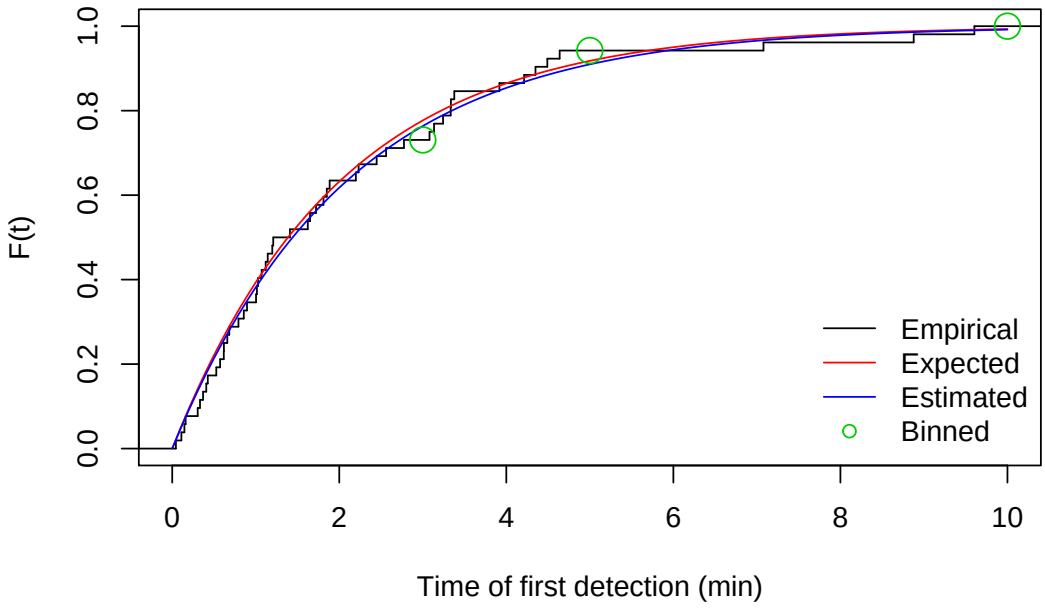


Now let's visualize the corresponding cumulative distribution function. We also bin the events into time intervals defined by interval end times in the vector `br` (breaks to be used with `cut`):

```
br <- c(3, 5, 10)
i <- cut(v1$t, c(0, br), include.lowest = TRUE)
table(i)
```

```
## i
##  [0,3]  (3,5] (5,10]
##     38     11      3
```

```
plot(stepfun(v1$t, (0:nrow(v1))/nrow(v1)), do.points=FALSE, xlim=c(0,10),
  xlab="Time of first detection (min)", ylab="F(t)", main="")
curve(1-exp(-phi*x), add=TRUE, col=2)
curve(1-exp(-phi_hat*x), add=TRUE, col=4)
legend("bottomright", bty="n", lty=c(1,1,1,NA),
  col=c(1,2,4,3), pch=c(NA,NA,NA,21),
  legend=c("Empirical", "Expected", "Estimated", "Binned"))
points(br, cumsum(table(i))/sum(table(i)), cex=2, col=3, pch=21)
```



4.6 Removal model

The time-removal model, originally developed for estimating wildlife and fish abundances from mark-recapture studies, was later reformulated for avian surveys with the goal of improving estimates of bird abundance by accounting for the availability bias inherent in point-count data. The removal model applied to point-count surveys estimates the probability that a bird is available for detection as a function of the average number of detectable cues that an individual bird gives per minute (singing rate, ϕ), and the known count duration (t).

Time-removal models are based on a removal experiment whereby animals are trapped and thereby removed from the closed population of animals being sampled. When applying a removal model to avian point-count surveys, the counts of singing birds $(Y_{ij}, \ldots, Y_{iJ})$ within a given point-count survey i $(i = 1, \ldots, n)$ are tallied relative to when each bird is first detected in multiple and consecutive time intervals, with the survey start time $t_{i0} = 0$, the end times of the time intervals t_{ij} $(j = 1, 2, \ldots, J)$, and the total count duration of the survey

$$t_{iJ}$$

. We count each individual bird once, so individuals are 'mentally removed' from a closed population of undetected birds by the surveyor.

The continuous-time formulation of the removal model is identical to the Exponential survival model formulation with respect to the cumulative density function, which defines probability of availability for sampling given the occurrence of the species. The response variable in the removal model follows multinomial distribution with cell probabilities derived from the cumulative probability function.

We will use the `detect::cmulti` function to fit multinomial models using conditional maximum likelihood procedure (the conditioning means that we only use observations where the total count is not 0, i.e. the species was present). The `Y` matrix lists the number of new individuals counted in each time interval, the `D` matrix gives the interval end times. (We use the `detect::cmulti.fit` function to be able to fit the model to a single survey.)

```
(y <- matrix(as.numeric(table(i)), nrow=1))
```

```
##      [,1] [,2] [,3]
## [1,]   38   11    3
```

```
(d <- matrix(br, nrow=1))
```

```
##      [,1] [,2] [,3]
## [1,]    3    5   10
```

```
(phi_hat1 <- exp(cmulti.fit(y, d, type="rem")$coef))
```

```
## [1] 0.4683
```

```
phi # setting
```

```
## [1] 0.5
```

```
phi_hat # from time-to-event data
```

```
##   rate
## 0.4808
```

4.6.1 Real data

Let's pick a species from the JOSM data set. For predictors, we will use a variable capturing date (`DAY`; standardized ordinal day of the year) and an other one capturing time of day (`TSSR`; time since local sunrise). The data frame `X` contains the predictors. The matrix `Y` contains the counts of newly counted individuals binned into consecutive time intervals: cell values are the Y_{ij}'s. The `D` object is another matrix mirroring the structure of `Y` but instead of counts, it contains the interval end times: cell values are the t_{ij}'s.

```
yall <- Xtab(~ SiteID + Dur + SpeciesID,
  josm$counts[josm$counts$DetectType1 != "V",])
yall <- yall[sapply(yall, function(z) sum(rowSums(z) > 0)) > 100]

spp <- "TEWA"

Y <- as.matrix(yall[[spp]])
D <- matrix(c(3, 5, 10), nrow(Y), 3, byrow=TRUE,
  dimnames=dimnames(Y))
X <- josm$surveys[rownames(Y), c("DAY", "TSSR")]
head(Y[rowSums(Y) > 0,])
```

```
##         0-3min 3-5min 5-10min
## CL10106      4      0       0
## CL10112      2      0       0
## CL10120      1      1       0
## CL10170      1      0       0
## CL10172      0      0       2
## CL10181      0      0       1
```

```
head(D)
```

```
##         0-3min 3-5min 5-10min
## CL10102      3      5      10
## CL10106      3      5      10
## CL10108      3      5      10
## CL10109      3      5      10
## CL10111      3      5      10
## CL10112      3      5      10
```

```
summary(X)
```

```
##       DAY              TSSR
##  Min.   :0.392   Min.   :-0.0285
##  1st Qu.:0.422   1st Qu.: 0.0506
##  Median :0.452   Median : 0.1041
##  Mean   :0.450   Mean   : 0.1040
##  3rd Qu.:0.474   3rd Qu.: 0.1568
##  Max.   :0.504   Max.   : 0.2357
```

The `D` matrix can take different methodologies for each row. The leftover values in each row must be filled with `NA`s and the pattern of `NA`s must match between the `Y` and `D` matrices (i.e. you should't have observation in a non-existing time interval). Integrating data becomes really easy this way, for example:

```
matrix(c(3, 5, 10, NA, NA, 1:5, 4, 8, NA, NA, NA), 3, byrow=TRUE)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    3    5   10   NA   NA
## [2,]    1    2    3    4    5
## [3,]    4    8   NA   NA   NA
```

4.6.2 Time-invariant conventional model

Time-invariant means that the rate is constant over time (i.e. no difference between morning and midnight), while conventional refers to the assumption that all individuals share the same rate (their behaviour is identical in this regard).

In the time-invariant conventional removal model (`Me0`), the individuals of a species at a given location and time are assumed to be homogeneous in their singing rates. The time to first detection follows the Exponential distribution, and the cumulative density function of times to first detection in time interval $(0, t_{iJ})$ gives us the probability that a bird sings at least once during the point count as $p(t_{iJ}) = 1 - exp(-t_{iJ}\phi)$.

We fit this model by specifying intercep-only in the right hand side of the formula, and `type="rem"` as part of the `cmulti` call:

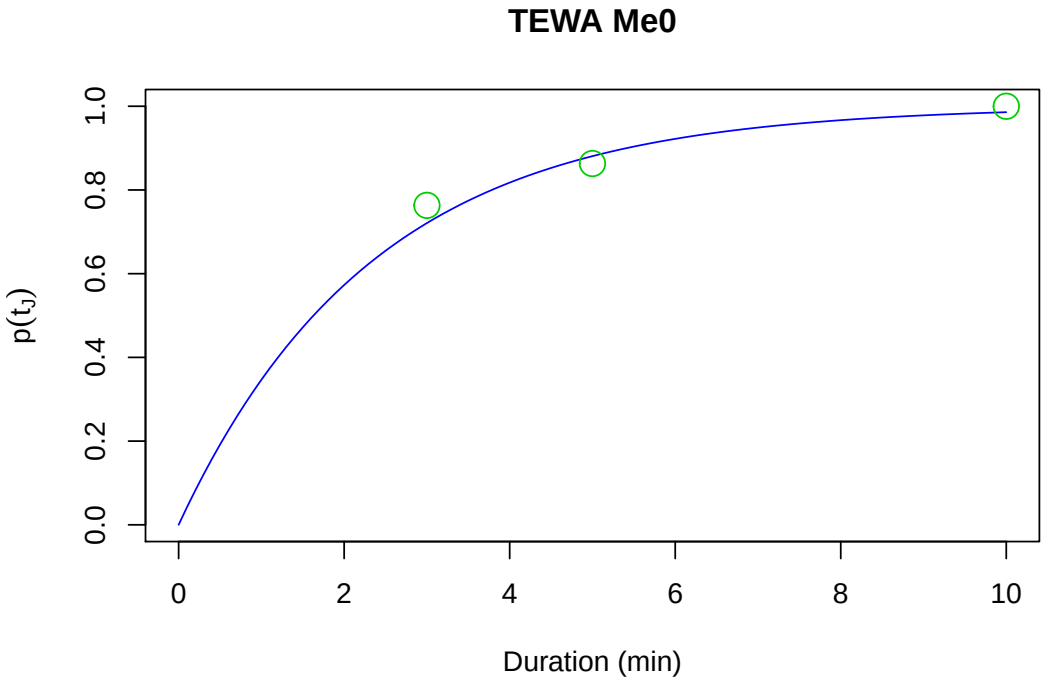
```
Me0 <- cmulti(Y | D ~ 1, type="rem")
summary(Me0)
```

```
##
## Call:
## cmulti(formula = Y | D ~ 1, type = "rem")
##
## Removal Sampling (homogeneous singing rate)
## Conditional Maximum Likelihood estimates
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## log.phi_(Intercept)  -0.8547     0.0174   -49.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood: -3.2e+03
## BIC = 6.42e+03
```

```
(phi_Me0 <- exp(coef(Me0)))
```

```
## log.phi_(Intercept)
##              0.4254
```

```
curve(1-exp(-x*phi_Me0), xlim=c(0, 10), ylim=c(0, 1), col=4,
  xlab="Duration (min)", ylab=expression(p(t[J])),
  main=paste(spp, "Me0"))
points(D[1,], cumsum(colSums(Y))/sum(Y), cex=2, col=3, pch=21)
```



4.6.3 Time-varying conventional removal model

Singing rates of birds vary with time of day, time of year, breeding status, and stage of the nesting cycle. Thus, removal model estimates of availability may be improved by accounting for variation in singing rates using covariates for day of year and time of day. In this case $p(t_{iJ}) = 1 - e^{-t_{iJ}\phi_i}$ and $log(\phi_i) = \beta_0 + \sum_{k=1}^{K} \beta_k x_{ik}$ is the linear predictor with K covariates and the corresponding unknown coefficients (β_k, $k = 0, \ldots, K$).

Let's fit a couple of time-varying models using `DAY` and `TSSR` as covariates:

```
Me1 <- cmulti(Y | D ~ DAY, X, type="rem")
Me2 <- cmulti(Y | D ~ TSSR, X, type="rem")
```

Now compare the three conventional models based on AIC and inspect the summary for the best supported model with the `JDAY` effect.

```
Me_AIC <- AIC(Me0, Me1, Me2)
Me_AIC$delta_AIC <- Me_AIC$AIC - min(Me_AIC$AIC)
Me_AIC[order(Me_AIC$AIC),]
```

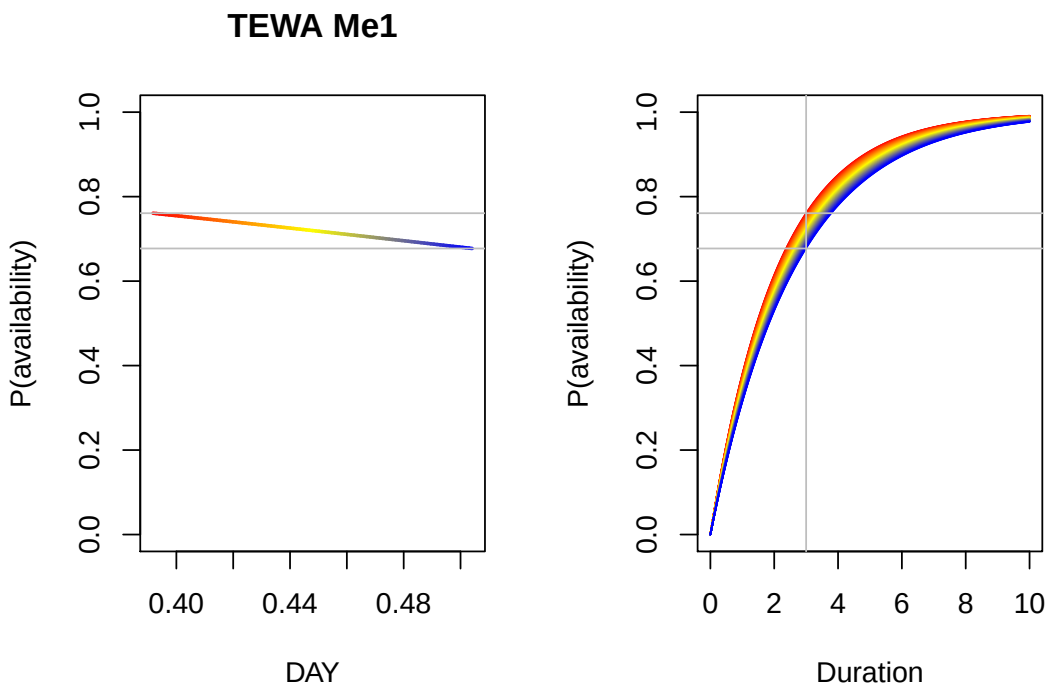
```
Me_Best <- get(rownames(Me_AIC)[Me_AIC$delta_AIC == 0])
summary(Me_Best)
```

```
##
## Call:
## cmulti(formula = Y | D ~ DAY, data = X, type = "rem")
##
## Removal Sampling (homogeneous singing rate)
## Conditional Maximum Likelihood estimates
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## log.phi_(Intercept)   0.0784     0.2615    0.30  0.76427
## log.phi_DAY          -2.0910     0.5866   -3.56  0.00036 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood: -3.2e+03
## BIC = 6.41e+03
```

To visually capture the time-varying effects, we make some plots using base graphics, colors matching the time-varying predictor. This way we can not only assess how availability probability (given a fixed time interval) is changing with the values of the predictor, but also how the cumulative distribution changes with time.

```
b <- coef(Me_Best)

n <- 100
DAY <- seq(min(X$DAY), max(X$DAY), length.out=n+1)
TSSR <- seq(min(X$TSSR), max(X$TSSR), length.out=n+1)
Duration <- seq(0, 10, length.out=n)
col <- colorRampPalette(c("red", "yellow", "blue"))(n)

op <- par(mfrow=c(1,2))
p1 <- 1-exp(-3*exp(b[1]+b[2]*DAY))
plot(DAY, p1, ylim=c(0,1), type="n",
    main=paste(spp, rownames(Me_AIC)[Me_AIC$delta_AIC == 0]),
```

```
    ylab="P(availability)")
for (i in seq_len(n)) {
    lines(DAY[c(i,i+1)], p1[c(i,i+1)], col=col[i], lwd=2)
}
abline(h=range(p1), col="grey")

plot(Duration, Duration, type="n", ylim=c(0,1),
    ylab="P(availability)")
for (i in seq_len(n)) {
    p2 <- 1-exp(-Duration*exp(b[1]+b[2]*DAY[i]))
    lines(Duration, p2, col=col[i])
}
abline(v=3, h=range(p1), col="grey")
```




```
par(op)
```

4.7 Finite mixtures

Let's relax the assumption that all individuals vocalize at the same rate. We can think about this as different groups in the population. The individuals within the groups have homogenerous rates, but the group level rates are different. We can introduce such heterogeneity into our bSims world by specifying the group level rates (`phi` vector) and the proportion of individuals belonging to the groups (`mix`).

```
phi <- c(10, 0.5)
mix <- c(0.25, 0.75)

set.seed(1)
(a2 <- bsims_populate(l, density=1)) # increase density
```

```
## bSims population
##   1 km x 1 km
##   stratification: H
##   total abundance: 104
```

```
(b2 <- bsims_animate(a2, vocal_rate=phi, mixture=mix))
```

```
## bSims events
##   1 km x 1 km
##   stratification: H
##   total abundance: 104
##   mixture, duration: 10 min
```

```
b2$vocal_rate
```

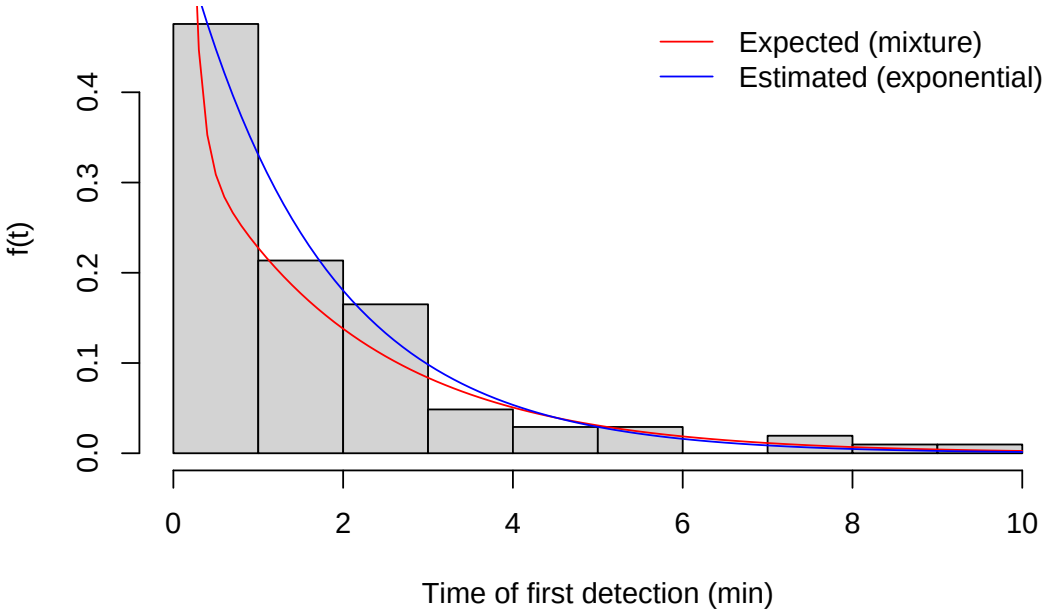
```
##   G1  G2
## H 10 0.5
## E 10 0.5
## R 10 0.5
```

If we plot the time to first detection data, we can see how expected distribution (red) is different from the fitted Exponential distribution assuming homogeneity:

```
v <- get_events(b2, event_type="vocal")
v1 <- v[!duplicated(v$i),]
(phi_hat <- fitdistr(v1$t, "exponential")$estimate)
```

```
##   rate
## 0.6068
```

```
hist(v1$t, xlab="Time of first detection (min)", freq=FALSE, main="",
  col="lightgrey", ylab="f(t)")
curve(mix[1]*dexp(x, phi[1])+mix[2]*dexp(x, phi[2]), add=TRUE, col=2)
curve(dexp(x, phi_hat), add=TRUE, col=4)
legend("topright", bty="n", lty=1, col=c(2,4),
  legend=c("Expected (mixture)", "Estimated (exponential)"))
```

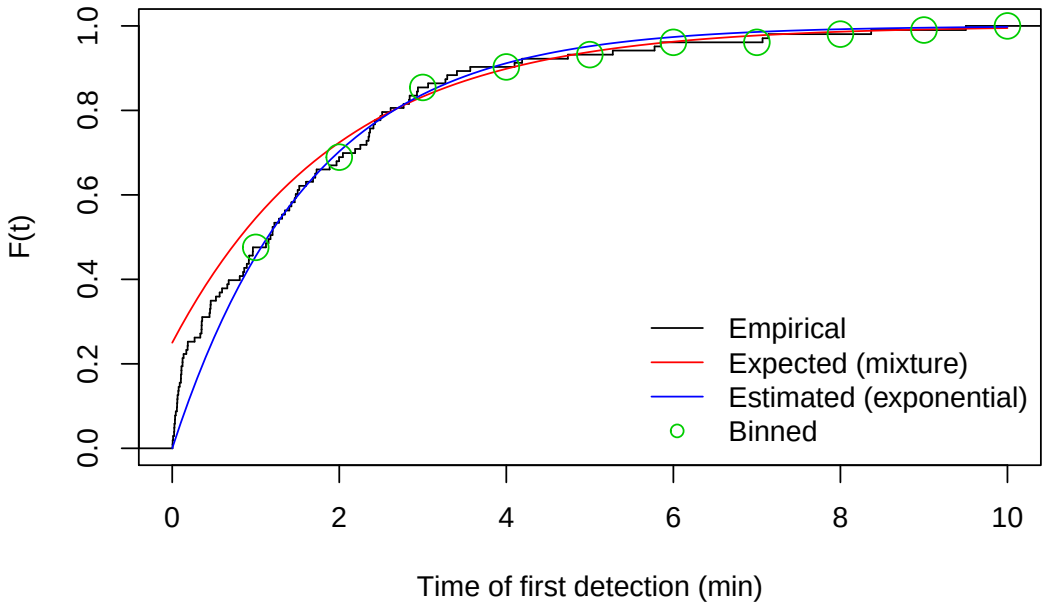


Now let's visualize the corresponding cumulative distribution function:

```
br <- 1:10
i <- cut(v1$t, c(0, br), include.lowest = TRUE)
table(i)
```

```
## i
##  [0,1]  (1,2]  (2,3]  (3,4]  (4,5]  (5,6]  (6,7]  (7,8]  (8,9] (9,10]
##     49     22     17      5      3      3      0      2      1      1
```

```
plot(stepfun(v1$t, (0:nrow(v1))/nrow(v1)), do.points=FALSE, xlim=c(0,10),
  xlab="Time of first detection (min)", ylab="F(t)", main="")
curve(1-mix[2]*exp(-phi[2]*x), add=TRUE, col=2)
curve(1-exp(-phi_hat*x), add=TRUE, col=4)
legend("bottomright", bty="n", lty=c(1,1,1,NA),
  col=c(1,2,4,3), pch=c(NA,NA,NA,21),
  legend=c("Empirical", "Expected (mixture)", "Estimated (exponential)", "Binn
points(br, cumsum(table(i))/sum(table(i)), cex=2, col=3, pch=21)
```



We use the `detect::cmulti` function to fit the finite mixture model:

```
(y <- matrix(as.numeric(table(i)), nrow=1))
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]   49   22   17    5    3    3    0    2    1     1
```

```
(d <- matrix(br, nrow=1))
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    2    3    4    5    6    7    8    9    10
```

```
cf <- cmulti.fit(y, d, type="fmix")$coef # log.phi, logit.c
```

```
c(phi=phi[2], c=mix[2]) # setting
```

```
##  phi    c
## 0.50 0.75
```

```
c(phi_hat=exp(cf[1]), c_hat=plogis(cf[2])) # estimate
```

```
## phi_hat   c_hat
##  0.5176  0.8837
```

4.7.1 Time-invariant finite mixture removal model

The removal model can accommodate behavioral heterogeneity in singing by subdividing the sampled population for a species at a given point into a finite mixture of birds with low and high singing rates, which requires the additional estimation of the proportion of birds in the sampled population with low singing rates.

In the continuous-time formulation of the finite mixture (or two-point mixture) removal model, the cumulative density function during a point count is given by $p(t_{iJ}) = (1-c)1 + c(1-e^{-t_{iJ}\phi}) = 1 - ce^{-t_{iJ}\phi}$, where ϕ is the singing rate for the group of infrequently singing birds, and c is the proportion of birds during the point count that are infrequent singers. The remaining proportions $(1-c$; the intercept of the cumulative density function) of the frequent singers are assumed to be detected instantaneously at the start of the first time interval. In the simplest form of the finite mixture model, the proportion and singing rate of birds that sing infrequently is homogeneous across all times and locations (model `Mf0`). We are using the `type = "fmix"` for finite mixture removal models.

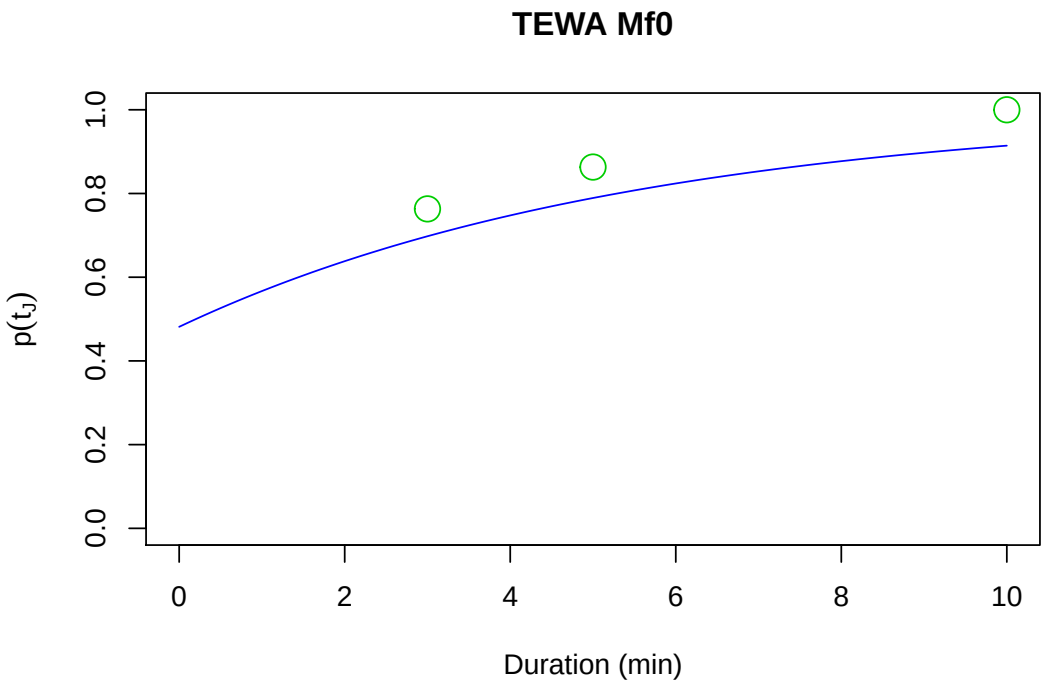
Here, for the read bird data set:

```
Mf0 <- cmulti(Y | D ~ 1, type="fmix")
summary(Mf0)
```

```
##
## Call:
## cmulti(formula = Y | D ~ 1, type = "fmix")
##
```

```
## Removal Sampling (heterogeneous singing rate)
## Conditional Maximum Likelihood estimates
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## log.phi_(Intercept)  -1.7146     0.0970  -17.68   <2e-16 ***
## logit.c               0.0742     0.0598    1.24     0.21
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood: -3.1e+03
## BIC = 6.22e+03
```

```
cf_Mf0 <- coef(Mf0)

curve(1-plogis(cf_Mf0[2]) * exp(-x*exp(cf_Mf0[1])),
  xlim=c(0, 10), ylim=c(0, 1), col=4, main=paste(spp, "Mf0"),
  xlab="Duration (min)", ylab=expression(p(t[J])))
points(D[1,], cumsum(colSums(Y))/sum(Y), cex=2, col=3, pch=21)
```



4.7.2 Time-varying finite mixture removal models

Previously, researchers have applied covariate effects on the parameter ϕ_i of the finite mixture model, similarly to how we modeled these effects in conventional models. This model assumes that the parameter c is constant irrespective of time and location (i.e. only the infrequent singer group changes its singing behavior).

We can fit finite mixture models with `DAY` and `TSSR` as covariates on ϕ. In this case $p(t_{iJ}) = 1 - ce^{-t_{iJ}\phi_i}$ and $log(\phi_i) = \beta_0 + \sum_{k=1}^{K} \beta_k x_{ik}$ is the linear predictor with K covariates and the corresponding unknown coefficients (β_k, $k = 0, \ldots, K$).

```
Mf1 <- cmulti(Y | D ~ DAY, X, type="fmix")
Mf2 <- cmulti(Y | D ~ TSSR, X, type="fmix")
```

Compare the three finite mixture models based on AIC and inspect the summary for the best supported model:

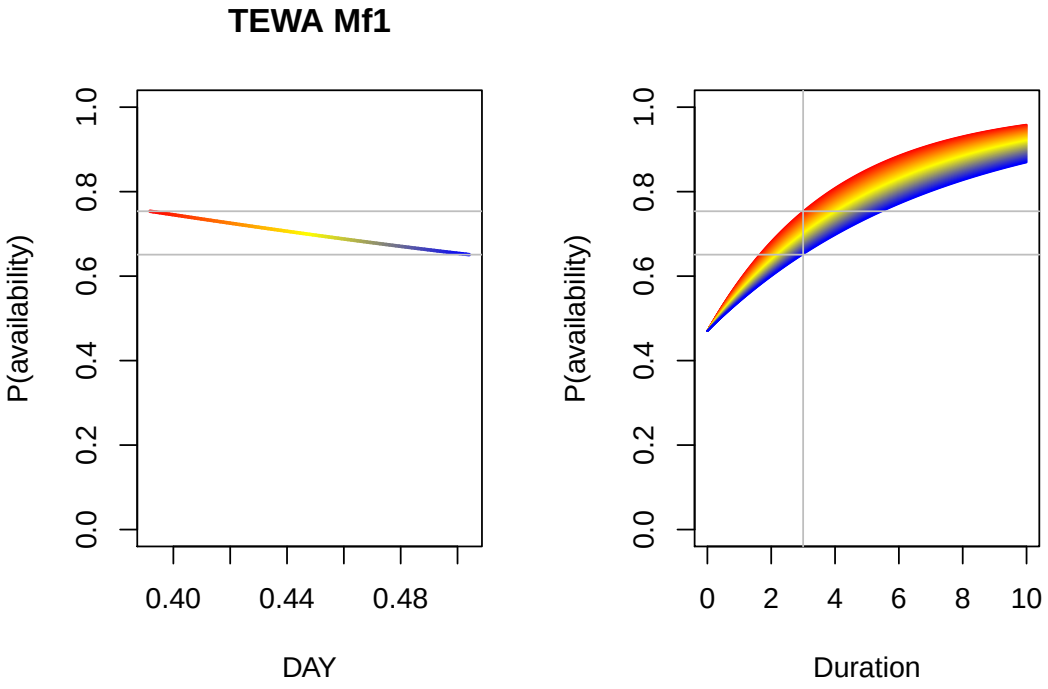
```
Mf_AIC <- AIC(Mf0, Mf1, Mf2)
Mf_AIC$delta_AIC <- Mf_AIC$AIC - min(Mf_AIC$AIC)

Mf_Best <- get(rownames(Mf_AIC)[Mf_AIC$delta_AIC == 0])
Mf_AIC[order(Mf_AIC$AIC),]

summary(Mf_Best)
```

```
##
## Call:
## cmulti(formula = Y | D ~ DAY, data = X, type = "fmix")
##
## Removal Sampling (heterogeneous singing rate)
## Conditional Maximum Likelihood estimates
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## log.phi_(Intercept)     0.754      0.848    0.89   0.3739
## log.phi_DAY            -5.412      1.938   -2.79   0.0052 **
## logit.c                 0.119      0.062    1.92   0.0548 .
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood: -3.1e+03
## BIC = 6.22e+03
```

We produce a similar plot as before.

```
b <- coef(Mf_Best)

op <- par(mfrow=c(1,2))
p1 <- 1-plogis(b[3])*exp(-3*exp(b[1]+b[2]*DAY))
plot(DAY, p1, ylim=c(0,1), type="n",
    main=paste(spp, rownames(Mf_AIC)[Mf_AIC$delta_AIC == 0]),
    ylab="P(availability)")
for (i in seq_len(n)) {
    lines(DAY[c(i,i+1)], p1[c(i,i+1)], col=col[i], lwd=2)
}
abline(h=range(p1), col="grey")

plot(Duration, Duration, type="n", ylim=c(0,1),
    ylab="P(availability)")
for (i in seq_len(n)) {
    p2 <- 1-plogis(b[3])*exp(-Duration*exp(b[1]+b[2]*DAY[i]))
    lines(Duration, p2, col=col[i])
}
abline(v=3, h=range(p1), col="grey")
```




```
par(op)
```

An alternative parametrization is that c_i rather than ϕ be the time-varying parameter, allowing the individuals to switch between the frequent and infrequent group depending on covariates. We can fit this class of finite mixture model with `DAY` and `TSSR` as covariates on c using `type = "mix"` (instead of `"fmix"`). In this case $p(t_{iJ}) = 1 - c_i e^{-t_{iJ}\phi}$ and $logit(c_i) = \beta_0 + \sum_{k=1}^{K} \beta_k x_{ik}$ is the linear predictor with K covariates and the corresponding unknown coefficients $(\beta_k, k = 0, \ldots, K)$. Because c_i is a proportion, we model it on the logit scale.

```
Mm1 <- cmulti(Y | D ~ DAY, X, type="mix")
Mm2 <- cmulti(Y | D ~ TSSR, X, type="mix")
```

We did not fit a null model for this parametrization, because it is identical to the `Mf0` model, so that model `Mf0` is what we use to compare AIC values and inspect the summary for the best supported model:

```
Mm_AIC <- AIC(Mf0, Mm1, Mm2)
Mm_AIC$delta_AIC <- Mm_AIC$AIC - min(Mm_AIC$AIC)
```

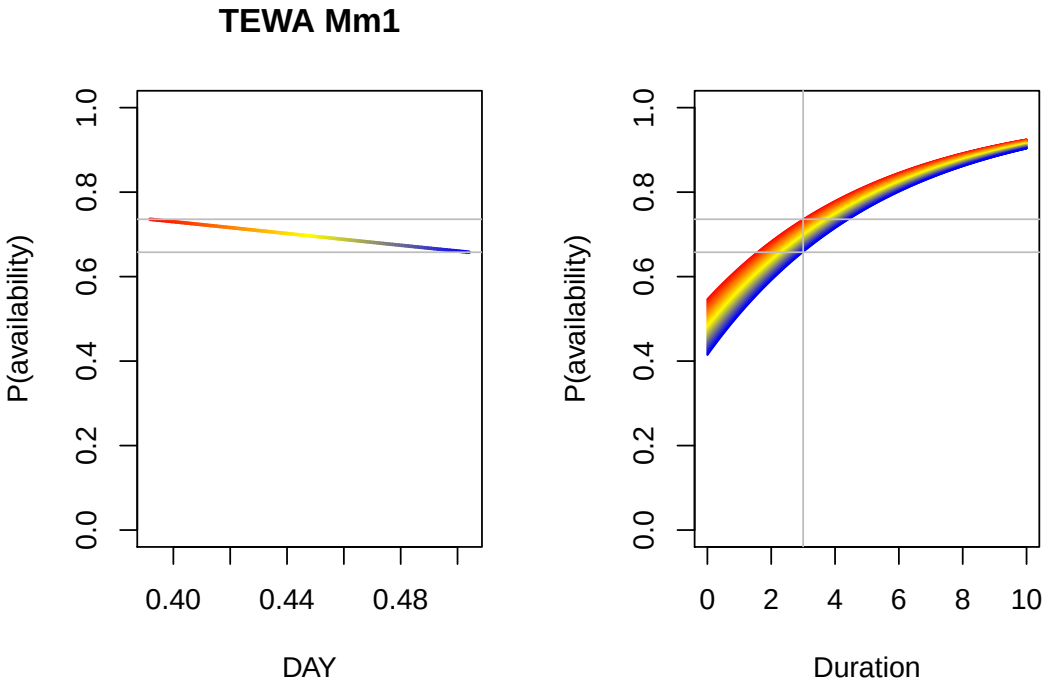
```
Mm_Best <- get(rownames(Mm_AIC)[Mm_AIC$delta_AIC == 0])
Mm_AIC[order(Mm_AIC$AIC),]

summary(Mm_Best)
```

```
##
## Call:
## cmulti(formula = Y | D ~ DAY, data = X, type = "mix")
##
## Removal Sampling (heterogeneous singing rate)
## Conditional Maximum Likelihood estimates
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## log.phi                -1.716      0.097  -17.69   <2e-16 ***
## logit.c_(Intercept)    -2.070      0.692   -2.99   0.0028 **
## logit.c_DAY             4.804      1.558    3.08   0.0020 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood: -3.1e+03
## BIC = 6.22e+03
```

We produce a similar plot as before:

```
b <- coef(Mm_Best)

op <- par(mfrow=c(1,2))
p1 <- 1-plogis(b[2]+b[3]*DAY)*exp(-3*exp(b[1]))
plot(DAY, p1, ylim=c(0,1), type="n",
    main=paste(spp, rownames(Mm_AIC)[Mm_AIC$delta_AIC == 0]),
    ylab="P(availability)")
for (i in seq_len(n)) {
    lines(DAY[c(i,i+1)], p1[c(i,i+1)], col=col[i], lwd=2)
}
abline(h=range(p1), col="grey")

plot(Duration, Duration, type="n", ylim=c(0,1),
```

```
    ylab="P(availability)")
for (i in seq_len(n)) {
    p2 <- 1-plogis(b[2]+b[3]*DAY[i])*exp(-Duration*exp(b[1]))
    lines(Duration, p2, col=col[i])
}
abline(v=3, h=range(p1), col="grey")
```




```
par(op)
```

4.8 Let the best model win

So which of the 3 parametrizations proved to be best for our data? It was the finite mixture with time-varying proportion of infrequent singers. Second was the other finite mixture model, while the conventional model was lagging behind.

```
M_AIC <- AIC(Me_Best, Mf_Best, Mm_Best)
M_AIC$delta_AIC <- M_AIC$AIC - min(M_AIC$AIC)
```

```
M_AIC[order(M_AIC$AIC),]
```

Finite mixture models provide some really nice insight into how singing behavior changes over time and, due to more parameters, they provide a better fit and thus minimize bias in population size estimates. But all this improvement comes with a price: sample size requirements (or more precisely, the number of detections required) are really high. To have all the benefits with reduced variance, one needs about 1000 non-zero observations to fit finite mixture models, 20 times more than needed to reliably fit conventional removal models. This is much higher than previously suggested minimum sample sizes.

Our findings also indicate that lengthening the count duration from 3 minutes to 5–10 minutes is an important consideration when designing field surveys to increase the accuracy and precision of population estimates. Well-informed survey design combined with various forms of removal sampling are useful in accounting for availability bias in point counts, thereby improving population estimates, and allowing for better integration of disparate studies at larger spatial scales.

Exercise
Compare different durations, numbers and lengths of time intervals when estimating vocalization rates.
Estimate vocalization rates for other species (e.g. rare species, specias with less frequent vocalizations).
Compare linear and polynomial `DAY` effects for migratory and resident species (e.g. BCCH, BOCH, BRCR, CORA, GRAJ, RBNU).

4.9 Estimating abundance

Let us use the bSims approach to see how well we can estimate abundance after accounting for availability. We set `Den` as density (D), and because area is $A = 100$ ha by default, the expected value of the abundance (λ) bacomes

AD, while the actual abundance (N) is a realization of that based on Poisson distribution ($N \sim Poisson(\lambda)$):

```
phi <- 0.5
Den <- 1

set.seed(1)
l <- bsims_init()
a <- bsims_populate(l, density=Den)
(b <- bsims_animate(a, vocal_rate=phi, move_rate=0))
```

```
## bSims events
##   1 km x 1 km
##   stratification: H
##   total abundance: 104
##   duration: 10 min
```

The next function we use is `bsims_transcribe` which takes the events data and bins it according to time intervals, `tint` defines the end times of each interval:

```
tint <- c(1, 2, 3, 4, 5)
(tr <- bsims_transcribe(b, tint=tint))
```

```
## bSims transcript
##   1 km x 1 km
##   stratification: H
##   total abundance: 104
##   duration: 10 min
##   detected: 104 heard
##   1st event detected by bins:
##     [0-1, 1-2, 2-3, 3-4, 4-5 min]
##     [0+ m]
```

```
tr$removal # binned new individuals
```

```
##     0-1min 1-2min 2-3min 3-4min 4-5min
## 0+m     35     28     16     12      6
```

```
(Y <- sum(tr$removal)) # detected in 0-3 min
```

```
## [1] 97
```

After `max(tint)` duration, we detected Y individuals. Because $E[Y] = NC$, we only have to estimate the correction factor C, that happens to be $C = p$ in this case because our bSims world ignored the observation process so far. p is estimated based on ϕ:

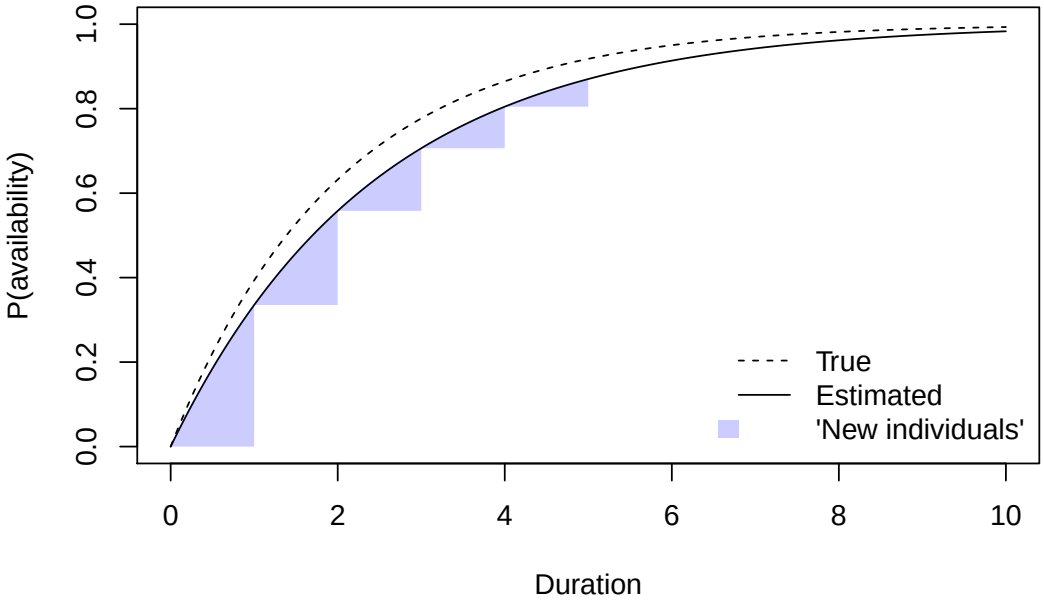
```
fit <- cmulti.fit(tr$removal, matrix(tint, nrow=1), type="rem")
c(true=phi, estimate=exp(fit$coef))
```

```
##     true estimate
##   0.5000   0.4083
```

```
(p <- 1-exp(-max(tint)*exp(fit$coef)))
```

```
## [1] 0.8702
```

```
tt <- seq(0, 10, 0.01)
plot(tt, 1-exp(-tt*phi), type="l", ylim=c(0, 1),
  ylab="P(availability)", xlab="Duration", lty=2)
lines(tt, 1-exp(-tt*exp(fit$coef)))
for (i in seq_len(length(tint))) {
  ii <- c(0, tint)[c(i, i+1)]
  ss <- tt >= ii[1] & tt <= ii[2]
  xi <- tt[ss]
  yi <- 1-exp(-xi*exp(fit$coef))
  polygon(c(xi, xi[length(xi)]), c(yi, yi[1]),
    border=NA, col="#0000ff33")
}
legend("bottomright", bty="n", lty=c(2, 1, NA),
  fill=c(NA, NA, "#0000ff33"), border=NA,
  legend=c("True", "Estimated", "'New individuals'"))
```



Our estimate of N becomes $Y/C = Y/p$:

```
N <- sum(a$abundance)
Nhat <- Y/p
c(true=N, estimate=Nhat)
```

```
##     true estimate
##    104.0    111.5
```

In this case, area is known, so density becomes:

```
A <- sum(a$area)
c(true=N / A, estimate=Nhat / A)
```

```
##     true estimate
##    1.040    1.115
```

Next we use the `Best` model from our real JOSM bird data analysis:

```
spp <- "TEWA"

Y <- as.matrix(yall[[spp]])
D <- matrix(c(3, 5, 10), nrow(Y), 3, byrow=TRUE,
  dimnames=dimnames(Y))
X <- josm$surveys[rownames(Y), c("DAY", "TSSR")]
```

```
Best <- get(rownames(M_AIC)[M_AIC$delta_AIC == 0])
summary(Best)
```

```
##
## Call:
## cmulti(formula = Y | D ~ DAY, data = X, type = "mix")
##
## Removal Sampling (heterogeneous singing rate)
## Conditional Maximum Likelihood estimates
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## log.phi                -1.716      0.097  -17.69   <2e-16 ***
## logit.c_(Intercept)    -2.070      0.692   -2.99   0.0028 **
## logit.c_DAY             4.804      1.558    3.08   0.0020 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood: -3.1e+03
## BIC = 6.22e+03
```

In this case, availability varies due to `DAY`. Our estimate of N_i becomes $Y_i/C_i = Y_i/p_i$:

```
p <- 1 - plogis(model.matrix(Best) %*% coef(Best)[-1]) *
  exp(-10 * exp(coef(Best)[1]))
summary(p)
```

```
##        V1
##  Min.   :0.903
##  1st Qu.:0.909
##  Median :0.913
##  Mean   :0.914
##  3rd Qu.:0.919
##  Max.   :0.925
```

We can now calculate mean abundance, where `ytot` tallies up the counts across the 3 time intervals:

```
ytot <- rowSums(Y)
table(ytot)
```

```
## ytot
##    0    1    2    3    4    5    6    7    8   12
## 1782 1151  887  466  188   71   20    2    1    1
```

```
mean(ytot / p)
```

```
## [1] 1.337
```

Alternatively, we can fit a GLM and use `log(p)` as an offset:

```
mod <- glm(ytot ~ 1, family=poisson, offset=log(p))
summary(mod)
```

```
##
## Call:
## glm(formula = ytot ~ 1, family = poisson, offset = log(p))
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.573  -1.560  -0.207   0.645   5.777
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.2910     0.0134    21.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 7087.6  on 4568  degrees of freedom
## Residual deviance: 7087.6  on 4568  degrees of freedom
## AIC: 14054
##
## Number of Fisher Scoring iterations: 5
```

The GLM based estimate comes from the intercept, because $E[Y_i] = N_i C_i$ is equivalent to $\lambda_i = e^{\beta_0} e^{o_i}$, this $\hat{N}_i = e^{\hat{\beta}_0}$:

```
exp(coef(mod))
```

```
## (Intercept)
##       1.338
```

This result tells us mean abundance after correcting for availability bias, but we don't know what area was effectively sampled, and detection of individuals given availability is probably less than 1 because this happens to be a real data set and it is guaranteed that humans in the forest cannot detect birds that are very far (say > 500 m away). We shall address these problem in the next chapter.

Chapter 5

The Detection Process

5.1 Introduction

As part of the detection process, a skilled observer counts individual birds at a count station. New individuals are assigned to time and distance categories, the type of behavior also registered. During this process, auditory cues travel through the distance between the bird and the observer. As the pover of the sound fades away, the chanches of being detected also decreases. If the detection process is based on visual detections, vegetation can block line of sight, etc. In this chapter, we scrutinize how this detection process contributes to the factor C.

5.2 Prerequisites

```
library(bSims)                # simulations
library(detect)               # multinomial models
library(Distance)             # distance sampling
load("_data/josm/josm.rda")   # JOSM data
source("functions.R")         # some useful stuff
```

5.3 Distance functions

The distance function ($g(d)$ describes the probability of detecting an individual given the distance between the observer and the individual (d). The detection itself is often triggered by visual or auditory cues, and thus depend on the individuals being available for detection (and of course being present in the survey area).

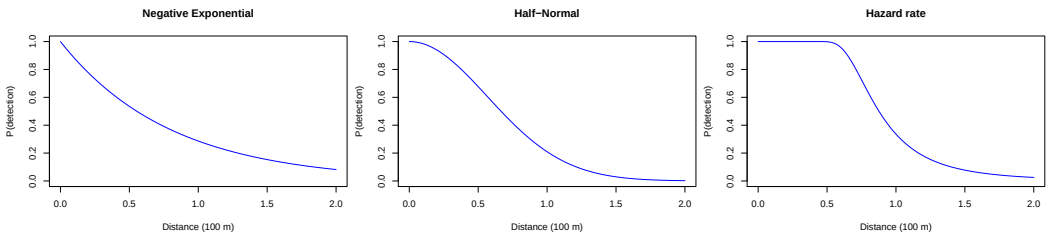
Distance functions have some characteristics:

- It is a monotonic decreasing function of distance,
- $g(0) = 1$: detection at 0 distance is perfect.

Here are some common distance function and rationale for their use (i.e. mechanisms leading to such distance shapes):

1. Negative Exponential: a one-parameter function ($g(d) = e^{-d/\tau}$), probability quickly decreases with distance, this mirrors sound attenuation under spherical spreading, so might be a suitable form for acoustic recoding devices (we will revisit this later), but not a very useful form for human based counts, as explained below;
2. Half-Normal: this is also a one-parameter function ($g(d) = e^{-(d/\tau)^2}$) where probability initially remain high (the *shoulder*), reflecting an increased chance of detecting individuals closer to the observer, this form has also sone practical advantages that we will discuss shortly (τ^2 is variance of the unfolded Normal distribution, $\tau^2/2$ is the variance of the Half-Normal distribution – both the Negative Exponential and the Half-Normal being special cases of $g(d) = e^{-(d/\tau)^b}$ that have the parameter b [$b > 0$] affecting the shoulder);
3. Hazard rate: this is a two-parameter model ($g(d) = 1 - e^{-(d/\tau)^{-b}}$) that have the parameter b ($b > 0$) affecting the more pronounced and sharp shoulder.

```
d <- seq(0, 2, 0.01)
plot(d, exp(-d/0.8), type="l", col=4, ylim=c(0,1),
  xlab="Distance (100 m)", ylab="P(detection)", main="Negative Exponential")
plot(d, exp(-(d/0.8)^2), type="l", col=4, ylim=c(0,1),
  xlab="Distance (100 m)", ylab="P(detection)", main="Half-Normal")
plot(d, 1-exp(-(d/0.8)^-4), type="l", col=4, ylim=c(0,1),
  xlab="Distance (100 m)", ylab="P(detection)", main="Hazard rate")
```



Exercise

Try different values of b to explore the different shapes of the Hazard rate function.

Write your own code (`plot(d, exp(-(d/<tau>)^<b>), type="l", ylim=c(0,1))`), or run `shiny::runApp("_shiny/distancefun.R")`.

We will apply this new found knowledge to our bSims world: the observer is in the middle of the landscape, and each vocalization event is aither detected or not, depending on the distance. Units of `tau` are given on 100 m units, so that corresponding density estimates will refer to ha as the unit area.

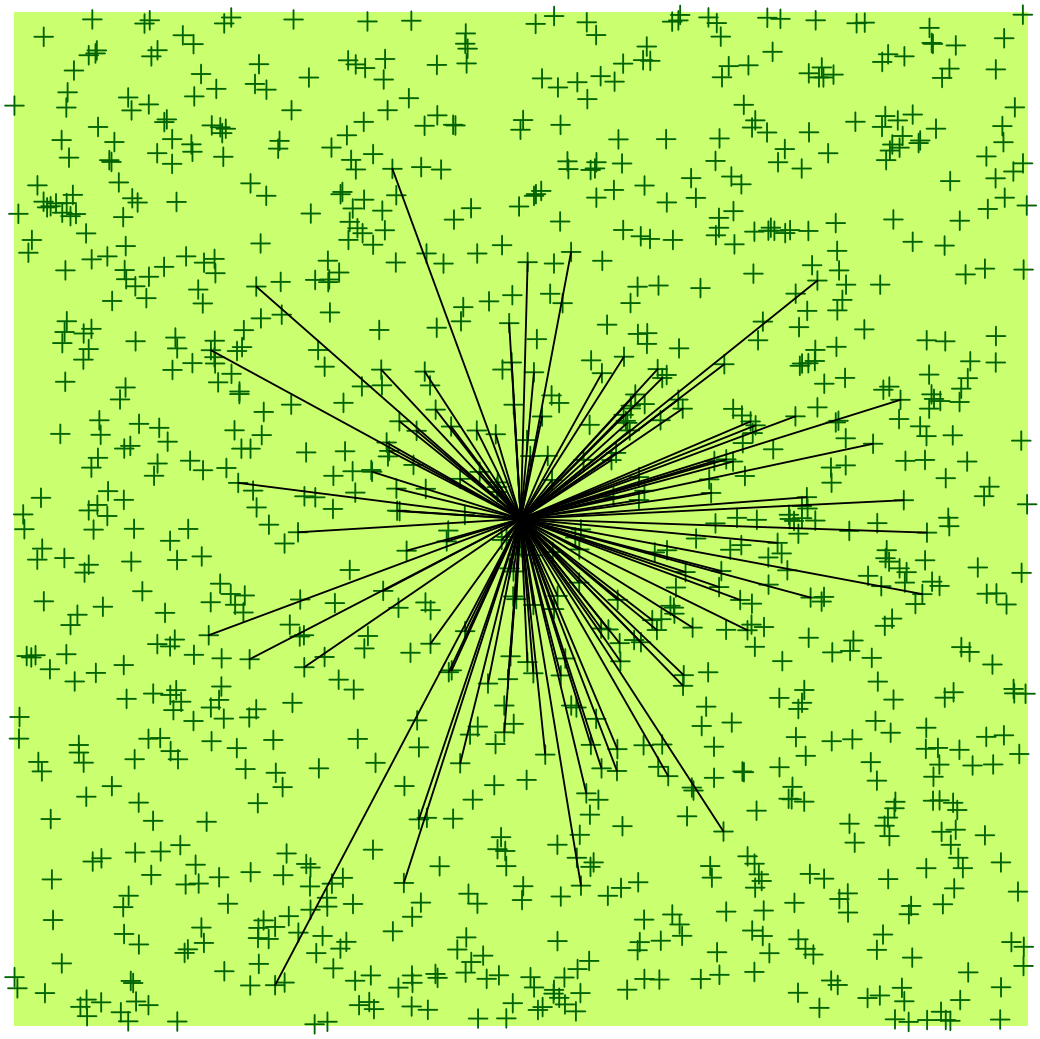
In this example, we want all individuals to be equally available, so we are going to override all behavioral aspects of the simulations by the `initial_location` argument when calling `bsims_animate`. We set `density` and `tau` high enough to detections in this example.

```
tau <- 2

set.seed(123)
l <- bsims_init()
a <- bsims_populate(l, density=10)
b <- bsims_animate(a, initial_location=TRUE)

(o <- bsims_detect(b, tau=tau))
```

```
## bSims detections
##   1 km x 1 km
##   stratification: H
```

```
##   total abundance: 1013
##   no events, duration: 10 min
##   detected: 128 seen/heard
```

```
plot(o)
```

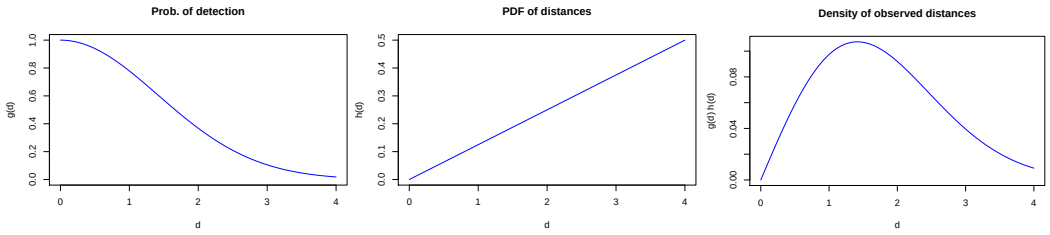


5.4 Distance sampling

The distribution of the *observed distances* is a product of detectability and the distribution of the individuals with respect to the point where the observer is located. For point counts, area increases linearly with radial distance, implying a triangular distribution with respect to the point ($h(d) = \pi 2d/A = \pi 2d/\pi r_{max}^2 = 2d/r_{max}^2$, where A is a circular survey area with truncation distance r_{max}). The product $g(d)h(d)$ gives the density function of the observed distances.

```
g <- function(d, tau, b=2, hazard=FALSE)
  if (hazard)
    1-exp(-(d/tau)^-b) else exp(-(d/tau)^b)
h <- function(d, rmax)
  2*d/rmax^2
```

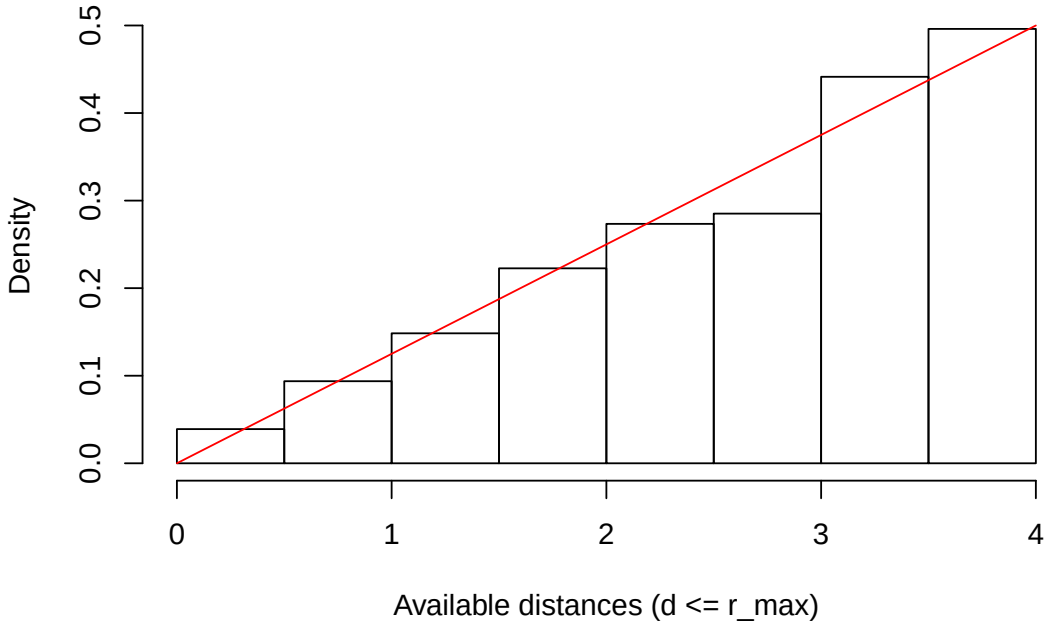
```
rmax <- 4

d <- seq(0, rmax, 0.01)
plot(d, g(d, tau), type="l", col=4, ylim=c(0,1),
  xlab="d", ylab="g(d)", main="Prob. of detection")
plot(d, h(d, rmax), type="l", col=4,
  xlab="d", ylab="h(d)", main="PDF of distances")
plot(d, g(d, tau) * h(d, rmax), type="l", col=4,
  xlab="d", ylab="g(d) h(d)", main="Density of observed distances")
```



The object `da` contains the distances to all the nests based on our bSims object, we use this to display the distribution of available distances:

```
da <- sqrt(rowSums(a$nests[,c("x", "y")]^2))

hist(da[da <= rmax], freq=FALSE, xlim=c(0, rmax),
```

```
  xlab="Available distances (d <= r_max)", main="")
curve(2*x/rmax^2, add=TRUE, col=2)
```

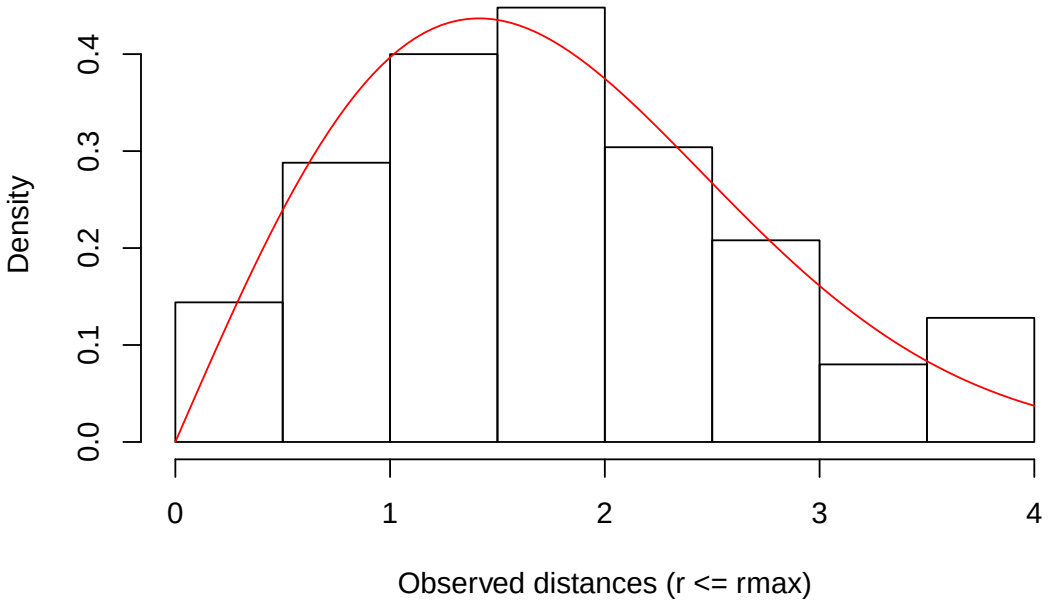


The `get_detections` function returns a data frame with the detected events (in our case just the nest locations): `$d` is the distance, `$a` is the angle (in degrees, counter clock-wise from positive x axis).

```
head(dt <- get_detections(o))
```

The following code plots the probability density of the observed distances within the truncation distance r_{max}, thus we need to standardize the $g(r)h(r)$ function by the integral sum:

```
f <- function(d, tau, b=2, hazard=FALSE, rmax=1)
  g(d, tau, b, hazard) * h(d, rmax)
tot <- integrate(f, lower=0, upper=rmax, tau=tau, rmax=rmax)$value

hist(dt$d[dt$d <= rmax], freq=FALSE, xlim=c(0, rmax),
  xlab="Observed distances (r <= rmax)", main="")
curve(f(x, tau=tau, rmax=rmax) / tot, add=TRUE, col=2)
```



In case of the Half-Normal, we can linearize the relationship by taking the log of the distance function: $log(g(d)) = log(e^{-(d/\tau)^2}) = -(d/\tau)^2 = x\frac{1}{\tau^2} = 0 + x\beta$. Consequently, we can use GLM to fit a model with $x = -d^2$ as predictor and no intercept, and estimate $\hat{\beta}$ and $\hat{\tau} = \sqrt{1/\hat{\beta}}$.

For this method to work, we need to know the observed and unobserved distances as well, which makes this approach of low utility in practice when location of unobserved individuals is unknown. But we can at least check our bSims data:

```
dat <- data.frame(
  distance=da,
  x=-da^2,
  detected=ifelse(rownames(o$nests) %in% dt$i, 1, 0))
summary(dat)
```

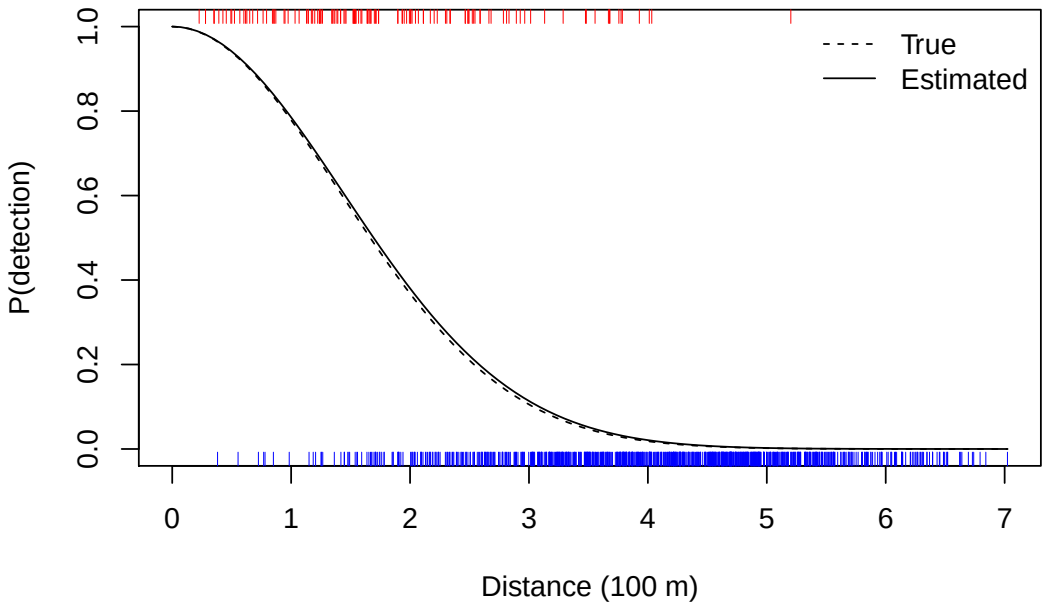
```
##     distance           x             detected
##  Min.   :0.226   Min.   :-49.33   Min.   :0.000
##  1st Qu.:2.876   1st Qu.:-23.74   1st Qu.:0.000
##  Median :3.965   Median :-15.72   Median :0.000
##  Mean   :3.827   Mean   :-16.69   Mean   :0.126
##  3rd Qu.:4.872   3rd Qu.: -8.27   3rd Qu.:0.000
```

```
##  Max.   :7.024   Max.   : -0.05   Max.   :1.000
```

```
mod <- glm(detected ~ x - 1, data=dat, family=binomial(link="log"))
c(true=tau, estimate=sqrt(1/coef(mod)))
```

```
##       true estimate.x
##      2.000      2.034
```

```
curve(exp(-(x/sqrt(1/coef(mod)))^2),
  xlim=c(0,max(dat$distance)), ylim=c(0,1),
  xlab="Distance (100 m)", ylab="P(detection)")
curve(exp(-(x/tau)^2), lty=2, add=TRUE)
rug(dat$distance[dat$detected == 0], side=1, col=4)
rug(dat$distance[dat$detected == 1], side=3, col=2)
legend("topright", bty="n", lty=c(2,1),
  legend=c("True", "Estimated"))
```



The Distance package offers various tools to fit models to observed distance data. See here for a tutorial. The following script fits the Half-Normal (`key = "hn"`) without ajustments (`adjustment=NULL`) to observed distance data from truncated point transect. It estimates $\sigma = \sqrt{\tau}$:

```
dd <- ds(dt$d, truncation = rmax, transect="point",
  key = "hn", adjustment=NULL)
```

```
## Fitting half-normal key function
```

```
## Key only model: not constraining for monotonicity.
```

```
## AIC= 315.502
```

```
## No survey area information supplied, only estimating detection function.
```

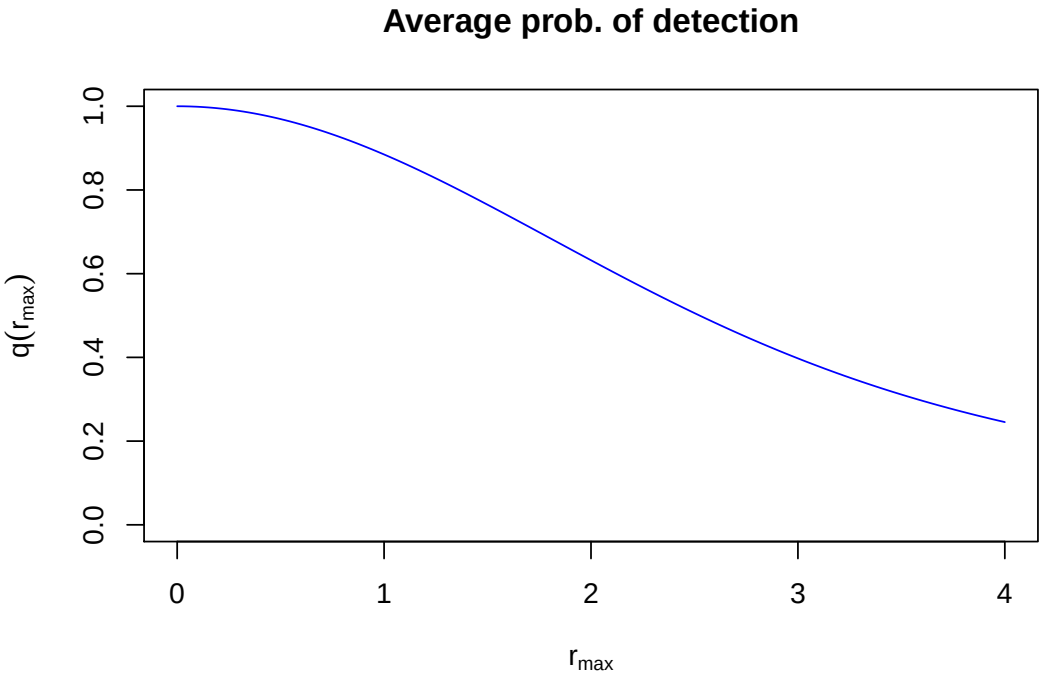
```
c(true=tau, estimate=exp(dd$ddf$par)^2)
```

```
##     true estimate
##    2.000    2.176
```

5.5 Average detection

To calculate the average probability of detecting individuals within a circle with truncation distance r_{max}, we need to integrate over the product of $g(r)$ and $h(r)$: $q(r_{max}) = \int_0^{r_{max}} g(d)h(d)dd$. This gives the volume of pie dough cut at r_{max}, compared to the volume of the cookie cutter (πr_{max}^2).

```
q <- sapply(d[d > 0], function(z)
  integrate(f, lower=0, upper=z, tau=tau, rmax=z)$value)

plot(d, c(1, q), type="l", col=4, ylim=c(0,1),
  xlab=expression(r[max]), ylab=expression(q(r[max])),
  main="Average prob. of detection")
```

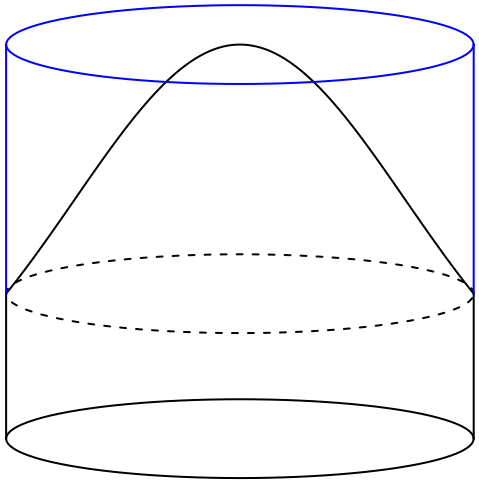


For the Half-Normal detection function, the analytical solution for the average probability is $\pi\tau^2[1 - exp(-d^2/\tau^2)]/(\pi r_{max}^2)$, where the denominator is a normalizing constant representing the volume of a cylinder of perfect detectability.

To visualize this, here is the pie analogy for $\tau = 2$ and $r_{max} = 2$:

```
tau <- 2
rmax <- 2
w <- 0.1
m <- 2
plot(0, type="n", xlim=m*c(-rmax, rmax), ylim=c(-w, 1+w),
  axes=FALSE, ann=FALSE)
yh <- g(rmax, tau=tau)
lines(seq(-rmax, rmax, rmax/100),
  g(abs(seq(-rmax, rmax, rmax/100)), tau=tau))
draw_ellipse(0, yh, rmax, w, lty=2)
lines(-c(rmax, rmax), c(0, yh))
lines(c(rmax, rmax), c(0, yh))
draw_ellipse(0, 0, rmax, w)
draw_ellipse(0, 1, rmax, w, border=4)
```

```
lines(-c(rmax, rmax), c(yh, 1), col=4)
lines(c(rmax, rmax), c(yh, 1), col=4)
```

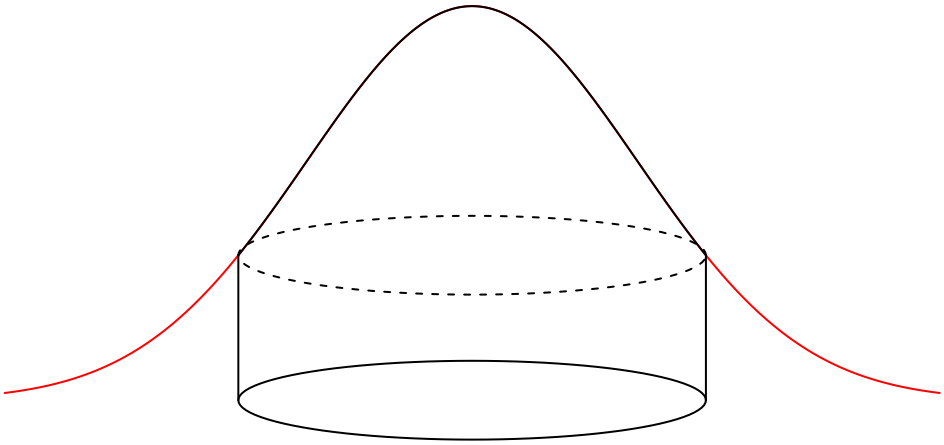


5.6 Binned distances

The cumulative density function for the Half-Normal distribution ($\pi(r) = 1 - e^{-(r/\tau)^2}$) is used to calculate cell probabilities for binned distance data (the normalizing constant is the area of the integral $\pi\tau^2$, instead of πr_{max}^2). It captures the proportion of the observed distances relative to the whole volume of the observed distance density. In the pie analogy, this is the dough volume inside the cookie cutter, compared to the dough volume inside and outside of the cutter (that happens to be $\pi\tau^2$ for the Half-Normal):

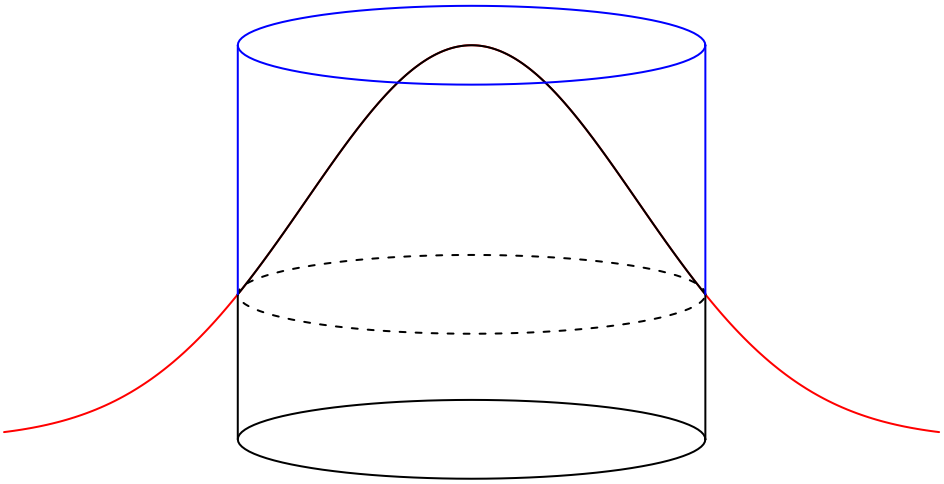
```
plot(0, type="n", xlim=m*c(-rmax, rmax), ylim=c(-w, 1+w),
  axes=FALSE, ann=FALSE)
yh <- g(rmax, tau=tau)
lines(seq(-m*rmax, m*rmax, rmax/(m*100)),
  g(seq(-m*rmax, m*rmax, rmax/(m*100)), tau=tau),
  col=2)
lines(seq(-rmax, rmax, rmax/100),
  g(abs(seq(-rmax, rmax, rmax/100)), tau=tau))
draw_ellipse(0, yh, rmax, w, lty=2)
lines(-c(rmax, rmax), c(0, yh))
```

```
lines(c(rmax, rmax), c(0, yh))
draw_ellipse(0, 0, rmax, w)
```



In case of the Half-Normal distance function, τ is the *effective detection radius* (EDR). The effective detection radius is the distance from observer where the number of individuals missed within EDR (volume of 'air' in the cookie cutter above the dough) equals the number of individuals detected outside of EDR (dough volume outside the cookie cutter), EDR is the radius r_e where $q(r_e) = \pi(r_e)$:

```
plot(0, type="n", xlim=m*c(-rmax, rmax), ylim=c(-w, 1+w),
  axes=FALSE, ann=FALSE)
yh <- g(rmax, tau=tau)
lines(seq(-m*rmax, m*rmax, rmax/(m*100)),
  g(seq(-m*rmax, m*rmax, rmax/(m*100)), tau=tau),
  col=2)
lines(seq(-rmax, rmax, rmax/100),
  g(abs(seq(-rmax, rmax, rmax/100)), tau=tau))
draw_ellipse(0, yh, rmax, w, lty=2)
lines(-c(rmax, rmax), c(0, yh))
lines(c(rmax, rmax), c(0, yh))
draw_ellipse(0, 0, rmax, w)
draw_ellipse(0, 1, rmax, w, border=4)
lines(-c(rmax, rmax), c(yh, 1), col=4)
lines(c(rmax, rmax), c(yh, 1), col=4)
```



Exercise

What would be a computational algorithm to calculate EDR for any distance function and truncation distance?
Trye to explain how the code below is working.
Why are EDRs different for different truncation distances?

```
find_edr <- function(dist_fun, ..., rmax=Inf) {
  ## integral function
  f <- function(d, ...)
    dist_fun(d, ...) * 2*d*pi
  ## volume under dist_fun
  V <- integrate(f, lower=0, upper=rmax, ...)$value
  u <- function(edr)
    V - edr^2*pi
  uniroot(u, c(0, 1000))$root
}

find_edr(g, tau=1)
```

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

```
find_edr(g, tau=10)
```

```
## [1] 10
```

```
find_edr(g, tau=1, b=1)
```

```
## [1] 1.414
```

```
find_edr(g, tau=1, b=4, hazard=TRUE)
```

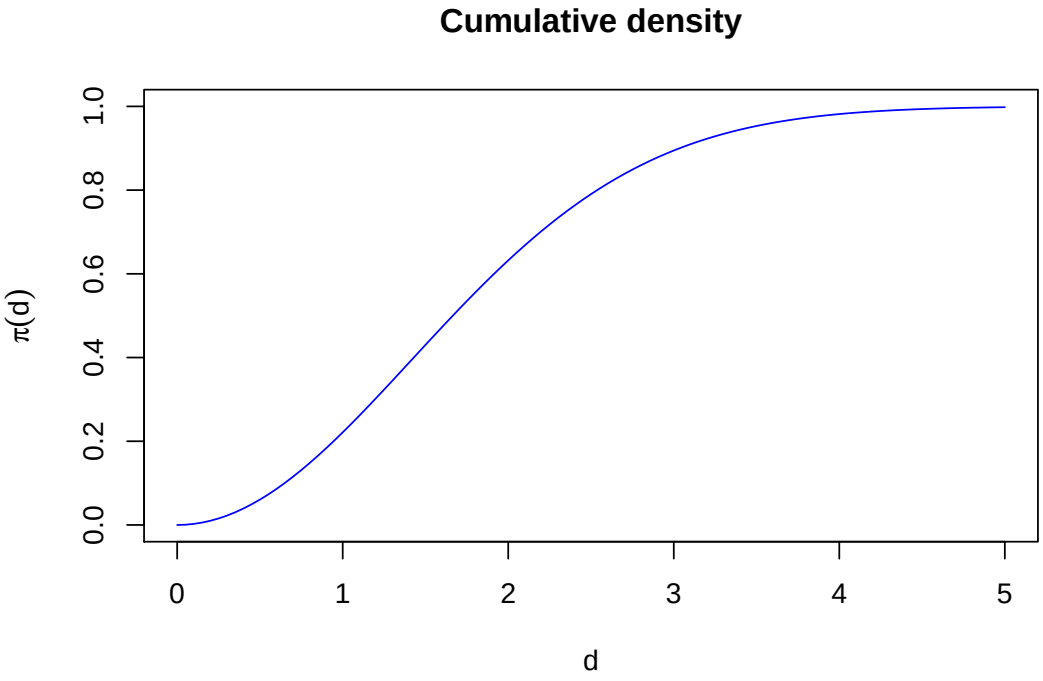
```
## [1] 1.331
```

```
find_edr(g, tau=1, rmax=1)
```

```
## [1] 0.7951
```

The function $\pi(r)$ increases monotonically from 0 to 1:

```
curve(1-exp(-(x/tau)^2), xlim=c(0, 5), ylim=c(0,1), col=4,
  ylab=expression(pi(d)), xlab=expression(d),
  main="Cumulative density")
```



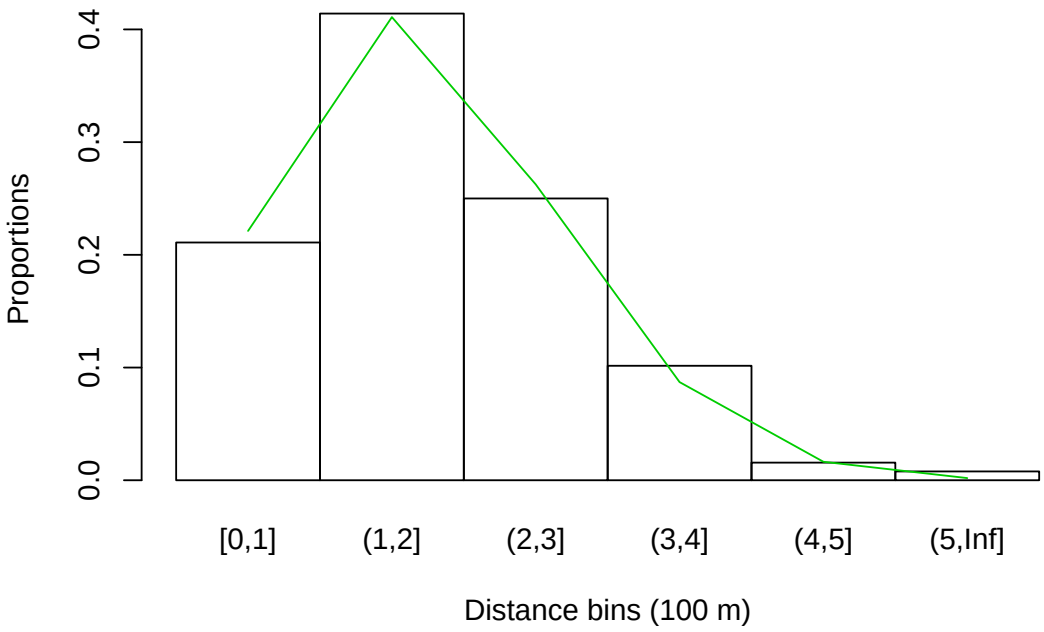
Here are binned distances for the bSims data, with expected proportions based on $\pi()$ cell probabilities (differences within the distance bins). The nice thing about this cumulative density formulation is that it applies equally to truncated and unlimited (not truncated) distance data, and the radius end

point for a bin (stored in `br`) can be infinite:

```
br <- c(1, 2, 3, 4, 5, Inf)
dat$bin <- cut(da, c(0, br), include.lowest = TRUE)
(counts <- with(dat, table(bin, detected)))
```

```
##          detected
## bin         0   1
##   [0,1]     7  27
##   (1,2]    42  53
##   (2,3]   111  32
##   (3,4]   227  13
##   (4,5]   287   2
##   (5,Inf] 211   1
```

```
pi_br <- 1-exp(-(br/tau)^2)

barplot(counts[,"1"]/sum(counts[,"1"]), space=0, col=NA,
  xlab="Distance bins (100 m)", ylab="Proportions")
lines(seq_len(length(br))-0.5, diff(c(0, pi_br)), col=3)
```



We can use the `bsims_transcribe` function for the same effect, and estimate

$\hat{\tau}$ based on the binned data:

```
(tr <- bsims_transcribe(o, rint=br))
```

```
## bSims transcript
##   1 km x 1 km
##   stratification: H
##   total abundance: 1013
##   no events, duration: 10 min
##   detected: 128 seen/heard
##   1st event detected by bins:
##     [0-10 min]
##     [0-100, 100-200, 200-300, 300-400, 400-500, 500+ m]
```

```
tr$removal
```

```
##          0-10min
## 0-100m        27
## 100-200m      53
## 200-300m      32
## 300-400m      13
## 400-500m       2
## 500+m          1
```

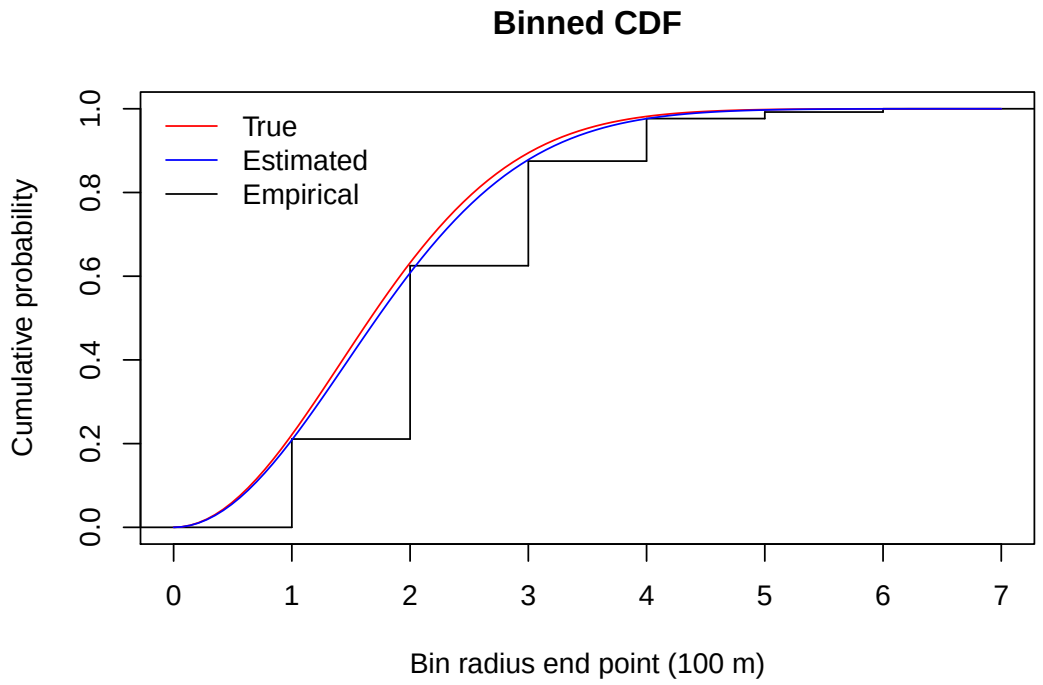
```
Y <- matrix(drop(tr$removal), nrow=1)
D <- matrix(br, nrow=1)

tauhat <- exp(cmulti.fit(Y, D, type="dis")$coef)

c(true=tau, estimate=tauhat)
```

```
##     true estimate
##    2.000    2.067
```

Here are cumulative counts and the true end expected cumulative cell probabilities:

```
plot(stepfun(1:6, c(0, cumsum(counts[,"1"])/sum(counts[,"1"]))),
  do.points=FALSE, main="Binned CDF",
  ylab="Cumulative probability",
  xlab="Bin radius end point (100 m)")
```

```
curve(1-exp(-(x/tau)^2), col=2, add=TRUE)
curve(1-exp(-(x/tauhat)^2), col=4, add=TRUE)
legend("topleft", bty="n", lty=1, col=c(2, 4, 1),
  legend=c("True", "Estimated", "Empirical"))
```



5.7 Availability bias

We have ignored availability so far when working with bSims, but can't continue like that for real data. What this means, is that $g(0) < 1$, so detecting an individual 0 distance from the observer depends on an event (visual or auditory) that would trigger the detection. For example, if a perfecly camouflaged birds sits in silence, detection might be difficult. Movement, or a vocalization can, however, reveal the individual and its location.

The `phi` and `tau` values are at the high end of plausible values for songbirds. The `Density` value is exaggerated, but this way we will have enough counts to prove our points using bSims:

```
phi <- 0.5
tau <- 2
Den <- 10
```

Now we go through the layers of our bSims world:

1. initiating the landscape,
2. populating the landscape by individuals,
3. breath life into the virtual birds and let them sing,
4. put in an observer and let the observation process begin.

```
set.seed(2)
l <- bsims_init()
a <- bsims_populate(l, density=Den)
b <- bsims_animate(a, vocal_rate=phi)
o <- bsims_detect(b, tau=tau)
```

Transcription is the process of turning the detections into a table showing new individuals detected by time intervals and distance bands, as defined by the `tint` and `rint` arguments, respectively.

```
tint <- c(1, 2, 3, 4, 5)
rint <- c(0.5, 1, 1.5, 2) # truncated at 200 m
(tr <- bsims_transcribe(o, tint=tint, rint=rint))
```

```
## bSims transcript
##   1 km x 1 km
##   stratification: H
##   total abundance: 957
##   duration: 10 min
##   detected: 282 heard
##   1st event detected by bins:
##     [0-1, 1-2, 2-3, 3-4, 4-5 min]
##     [0-50, 50-100, 100-150, 150-200 m]
```

```
(rem <- tr$removal) # binned new individuals
```

```
##          0-1min 1-2min 2-3min 3-4min 4-5min
## 0-50m         1      4      0      2      0
## 50-100m       4      6      2      4      1
```

```
## 100-150m     13      5      6      4      2
## 150-200m     13      5      2      7      3
```

```
colSums(rem)
```

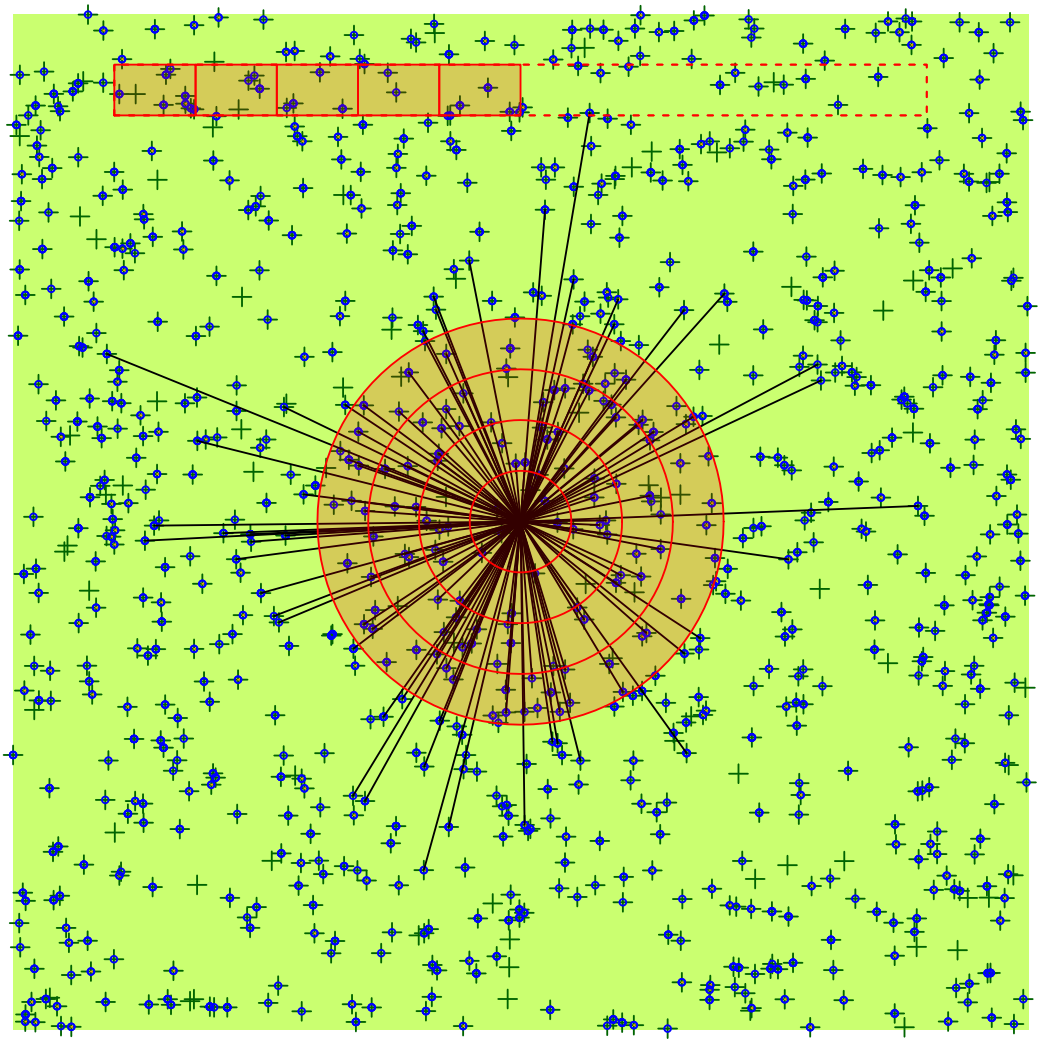
```
## 0-1min 1-2min 2-3min 3-4min 4-5min
##     31     20     10     17      6
```

```
rowSums(rem)
```

```
##    0-50m  50-100m 100-150m 150-200m
##        7       17       30       30
```

The plot method displays the detections presented as part of the `tr` object.

```
plot(tr)
```



The detection process and the transcription (following a prescribed protocol) is inseparable in the field. However, recordings made in the field can be processed by a number of different ways. Separating these processed gives the ability to make these conparisons on the exact same set of detections.

5.8 Estimating density with truncation

We now fit the removal model to the data pooled by time intervals. `p` is the cumulative probability of availability for the total duration:

```
fitp <- cmulti.fit(matrix(colSums(rem), 1), matrix(tint, 1), type="rem")
phihat <- exp(fitp$coef)
c(true=phi, estimate=exp(fitp$coef))
```

```
##     true estimate
##   0.5000   0.3301
```

```
(p <- 1-exp(-max(tint)*phihat))
```

```
## [1] 0.8081
```

The distance sampling model uses the distance binned counts, and a Half-Normal detection function, `q` is the cumulative probability of perceptibility within the area of truncation distance `rmax`:

```
fitq <- cmulti.fit(matrix(rowSums(rem), 1), matrix(rint, 1), type="dis")
tauhat <- exp(fitq$coef)
c(true=tau, estimate=tauhat)
```

```
##     true estimate
##    2.000    2.659
```

```
rmax <- max(rint)
(q <- (tauhat^2/rmax^2) * (1-exp(-(rmax/tauhat)^2)))
```

```
## [1] 0.7638
```

The known `Area`, `p`, and `q` makes up the correction factor, which is used to estimate density based on $\hat{D} = Y/(A\hat{p}\hat{q})$:

```
(A <- pi * rmax^2)
```

```
## [1] 12.57
```

```
Dhat <- sum(rem) / (A * p * q)
c(true=Den, estimate=Dhat)
```

```
##     true estimate
```

```
##    10.00    10.83
```

5.9 Unlimited distance

We now change the distance bins to include the area outside of the previous `rmax` distance, making the counts unlimited distance counts:

```
rint <- c(0.5, 1, 1.5, 2, Inf) # unlimited

(tr <- bsims_transcribe(o, tint=tint, rint=rint))
```

```
## bSims transcript
##   1 km x 1 km
##   stratification: H
##   total abundance: 957
##   duration: 10 min
##   detected: 282 heard
##   1st event detected by bins:
##     [0-1, 1-2, 2-3, 3-4, 4-5 min]
##     [0-50, 50-100, 100-150, 150-200, 200+ m]
```

```
(rem <- tr$removal) # binned new individuals
```

```
##          0-1min 1-2min 2-3min 3-4min 4-5min
## 0-50m         1      4      0      2      0
## 50-100m       4      6      2      4      1
## 100-150m     13      5      6      4      2
## 150-200m     13      5      2      7      3
## 200+m        23     13      7      3      2
```

```
colSums(rem)
```

```
## 0-1min 1-2min 2-3min 3-4min 4-5min
##     54     33     17     20      8
```

```
rowSums(rem)
```

```
##    0-50m  50-100m 100-150m 150-200m    200+m
##        7       17       30       30       48
```

The removal model is basically the same, the only difference is that the counts can be higher due to detecting over larger area and thus potentially detecting more individuals:

```
fitp <- cmulti.fit(matrix(colSums(rem), 1), matrix(tint, 1), type="rem")
phihat <- exp(fitp$coef)
c(true=phi, estimate=phihat)
```

```
##     true estimate
##   0.5000   0.4285
```

```
(p <- 1-exp(-max(tint)*phihat))
```

```
## [1] 0.8826
```

The diatance sampling model also takes the extended data set.

```
fitq <- cmulti.fit(matrix(rowSums(rem), 1), matrix(rint, 1), type="dis")
tauhat <- exp(fitq$coef)
c(true=tau, estimate=tauhat)
```

```
##     true estimate
##    2.000    2.021
```

The problem is that our truncation distance is infinite, thus the area that we are sampling is also infinite. This does not make too much sense, and not at all hepful in estimating density (anything divided by infinity is 0). So we use EDR (`tauhat` for Half-Normal) and calculate the estimated effective area sampled (`Ahat`; $\hat{A} = \pi\hat{\tau}^2$). We also set `q` to be 1, because the logic behind EDR is that its volume equals the volume of the integral, in other words, it is an area that would give on average same count under perfect detection Finally, we estimate density using $\hat{D} = Y/(\hat{A}\hat{p}1)$

```
(Ahat <- pi * tauhat^2)
```

```
## [1] 12.83
```

```
q <- 1

Dhat <- sum(rem) / (Ahat * p * q)
c(true=Den, estimate=Dhat)
```

```
##     true estimate
##    10.00    11.66
```

5.10 Replicating landscapes

Remember, that we have used so far a single location. We set the density unreasonably high to have enough counts for a reasonable estimate. We can independently replicate the simulation for multiple landscapes and analyze the results to give justice to bSims under idealized conditions:

```
phi <- 0.5
tau <- 1
Den <- 1

tint <- c(3, 5, 10)
rint <- c(0.5, 1, 1.5, Inf)

sim_fun <- function() {
  l <- bsims_init()
  a <- bsims_populate(l, density=Den)
  b <- bsims_animate(a, vocal_rate=phi)
  o <- bsims_detect(b, tau=tau)
  bsims_transcribe(o, tint=tint, rint=rint)$rem
}

B <- 200
set.seed(123)
res <- pbapply::pbreplicate(B, sim_fun(), simplify=FALSE)

Ddur <- matrix(tint, B, length(tint), byrow=TRUE)
Ydur1 <- t(sapply(res, function(z) colSums(z)))
Ydur2 <- t(sapply(res, function(z) colSums(z[-nrow(z),])))
colSums(Ydur1) / sum(Ydur1)
colSums(Ydur2) / sum(Ydur2)
fitp1 <- cmulti(Ydur1 | Ddur ~ 1, type="rem")
fitp2 <- cmulti(Ydur2 | Ddur ~ 1, type="rem")
```

```
phihat1 <- unname(exp(coef(fitp1)))
phihat2 <- unname(exp(coef(fitp2)))

Ddis1 <- matrix(rint, B, length(rint), byrow=TRUE)
Ddis2 <- matrix(rint[-length(rint)], B, length(rint)-1, byrow=TRUE)
Ydis1 <- t(sapply(res, function(z) rowSums(z)))
Ydis2 <- t(sapply(res, function(z) rowSums(z)[-length(rint)]))
colSums(Ydis1) / sum(Ydis1)
colSums(Ydis2) / sum(Ydis2)
fitq1 <- cmulti(Ydis1 | Ddis1 ~ 1, type="dis")
fitq2 <- cmulti(Ydis2 | Ddis2 ~ 1, type="dis")
tauhat1 <- unname(exp(fitq1$coef))
tauhat2 <- unname(exp(fitq2$coef))

## unlimited correction
Apq1 <- pi * tauhat1^2 * (1-exp(-max(tint)*phihat1)) * 1
rmax <- max(rint[is.finite(rint)])
## truncated correction
Apq2 <- pi * rmax^2 *
  (1-exp(-max(tint)*phihat2)) *
  (tauhat2^2/rmax^2) * (1-exp(-(rmax/tauhat2)^2))

round(rbind(
  phi=c(true=phi, unlimited=phihat1, truncated=phihat2),
  tau=c(true=tau, unlimited=tauhat1, truncated=tauhat2),
  D=c(Den, unlimited=mean(rowSums(Ydis1))/Apq1,
      truncated=mean(rowSums(Ydis2))/Apq2)), 4)
##     true unlimited truncated
## phi  0.5    0.4835    0.4852
## tau  1.0    1.0002    0.9681
## D    1.0    1.0601    1.1048
```

Exercise
If time permits, try different settings and time/distance intervals.

5.11 JOSM data

Quickly organize the JOSM data:

```
## predictors
x <- josm$surveys
x$FOR <- x$Decid + x$Conif+ x$ConifWet # forest
x$AHF <- x$Agr + x$UrbInd + x$Roads # 'alienating' human footprint
x$WET <- x$OpenWet + x$ConifWet + x$Water # wet + water
cn <- c("Open", "Water", "Agr", "UrbInd", "SoftLin", "Roads", "Decid",
  "OpenWet", "Conif", "ConifWet")
x$HAB <- droplevels(find_max(x[,cn])$index) # drop empty levels
levels(x$HAB)[levels(x$HAB) %in%
  c("OpenWet", "Water", "Open", "Agr", "UrbInd", "Roads")] <- "Open"
levels(x$HAB)[levels(x$HAB) %in%
  c("Conif", "ConifWet")] <- "Conif"
x$OBS <- as.factor(x$ObserverID)

## time intervals
yall_dur <- Xtab(~ SiteID + Dur + SpeciesID,
  josm$counts[josm$counts$DetectType1 != "V",])
yall_dur <- yall_dur[sapply(yall_dur, function(z) sum(rowSums(z) > 0)) > 100]

## distance intervals
yall_dis <- Xtab(~ SiteID + Dis + SpeciesID,
  josm$counts[josm$counts$DetectType1 != "V",])
yall_dis <- yall_dis[sapply(yall_dis, function(z) sum(rowSums(z) > 0)) > 100]
```

Pick our most abundant species again, and organize the data:

```
spp <- "TEWA"

Ydur <- as.matrix(yall_dur[[spp]])
Ddur <- matrix(c(3, 5, 10), nrow(Ydur), 3, byrow=TRUE,
  dimnames=dimnames(Ydur))
stopifnot(all(rownames(x) == rownames(Ydur)))

Ydis <- as.matrix(yall_dis[[spp]])
```

```
Ddis <- matrix(c(0.5, 1, Inf), nrow(Ydis), 3, byrow=TRUE,
  dimnames=dimnames(Ydis))
stopifnot(all(rownames(x) == rownames(Ydis)))

colSums(Ydur)
```

```
##  0-3min  3-5min 5-10min
##    4262     558     764
```

```
colSums(Ydis)
```

```
##   0-50m 50-100m   100+m
##    2703    2470     411
```

We pick a removal models with `DAY` as covariate, and calculate $p(t)$:

```
Mdur <- cmulti(Ydur | Ddur ~ DAY, x, type="rem")
summary(Mdur)
```

```
##
## Call:
## cmulti(formula = Ydur | Ddur ~ DAY, data = x, type = "rem")
##
## Removal Sampling (homogeneous singing rate)
## Conditional Maximum Likelihood estimates
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## log.phi_(Intercept)   0.0784     0.2615    0.30  0.76427
## log.phi_DAY          -2.0910     0.5866   -3.56  0.00036 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood: -3.2e+03
## BIC = 6.41e+03
```

```
phi <- exp(model.matrix(Mdur) %*% coef(Mdur))
summary(phi)
```

```
##        V1
```

```
##  Min.   :0.377
##  1st Qu.:0.402
##  Median :0.420
##  Mean   :0.423
##  3rd Qu.:0.448
##  Max.   :0.477
```

```
p <- 1-exp(-10*phi)
```

We fit the intercept only distance sampling model next:

```
Mdis0 <- cmulti(Ydis | Ddis ~ 1, x, type="dis")
summary(Mdis0)
```

```
##
## Call:
## cmulti(formula = Ydis | Ddis ~ 1, data = x, type = "dis")
##
## Distance Sampling (half-normal, circular area)
## Conditional Maximum Likelihood estimates
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## log.tau_(Intercept) -0.48272    0.00753   -64.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood: -3.82e+03
## BIC = 7.65e+03
```

Let's try a few covariates:

- continuous FORest cover covariate: sound attenuation increases with forest cover;
- discrete HABitat has 3 levels: open, deciduous forest, and coniferous forest (based on dominant land cover), because broad leaves and needles affect sound attenuation;
- finally, we use observer ID as categorical variable: observers might have different hearing abilities, training/experiance levels, good times, bad times, etc.

```
Mdis1 <- cmulti(Ydis | Ddis ~ FOR, x, type="dis")
Mdis2 <- cmulti(Ydis | Ddis ~ HAB, x, type="dis")
```

We can look at AIC to find the best supported model:

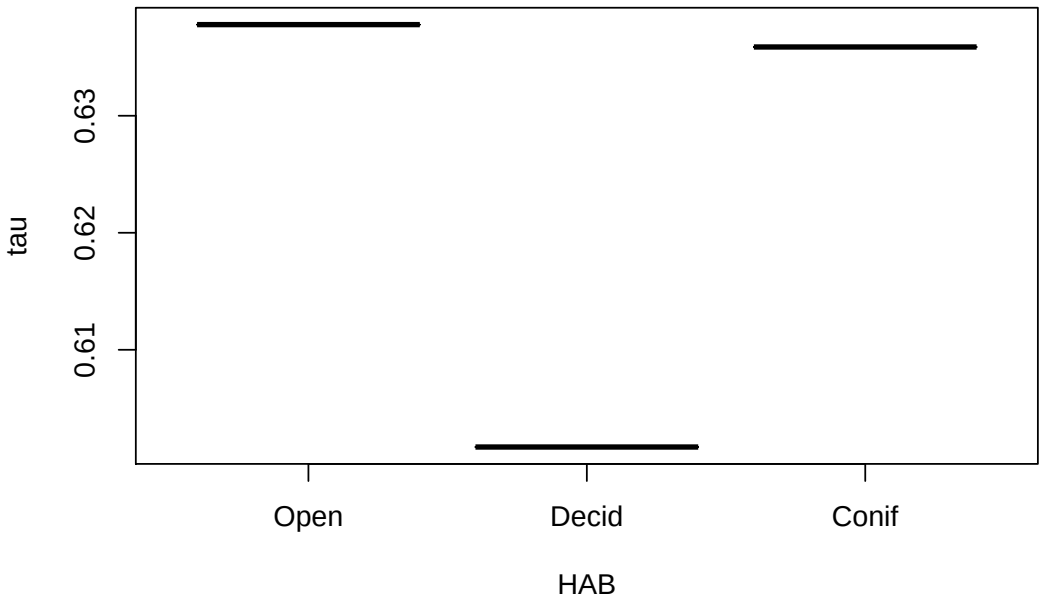
```
aic <- AIC(Mdis0, Mdis1, Mdis2)
aic$delta_AIC <- aic$AIC - min(aic$AIC)
aic[order(aic$AIC),]

Mdis <- get(rownames(aic)[aic$delta_AIC == 0])
summary(Mdis)
```

```
##
## Call:
## cmulti(formula = Ydis | Ddis ~ HAB, data = x, type = "dis")
##
## Distance Sampling (half-normal, circular area)
## Conditional Maximum Likelihood estimates
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## log.tau_(Intercept)  -0.4497     0.0321  -14.02   <2e-16 ***
## log.tau_HABDecid     -0.0583     0.0336   -1.73    0.083 .
## log.tau_HABConif     -0.0030     0.0343   -0.09    0.930
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-likelihood: -3.82e+03
## BIC = 7.66e+03
```

Exercise
Use `OBS` as predictor for `tau` and look at predicted EDRs.
What is the practical issue with using observer as predictor?

After finding the best model, we predict `tau`:

```
tau <- exp(model.matrix(Mdis) %*% coef(Mdis))
boxplot(tau ~ HAB, x)
```



Finally, we calculate the correction factor for unlimited distances, and predict mean density:

```
Apq <- pi * tau^2 * p * 1
x$ytot <- rowSums(Ydur)
mean(x$ytot / Apq)
```

```
## [1] 1.039
```

Alternatively, we can use the log of the correction as an offset in log-linear models. This offset is called the QPAD offset:

```
off <- log(Apq)
m <- glm(ytot ~ 1, data=x, offset=off, family=poisson)
exp(coef(m))
```

```
## (Intercept)
##       1.025
```

Exercise
Try distance sampling and density estimation for another species. Fit multiple GLMs with QPAD offsets and covariates affecting density, interpret the results and the visualize responses.

Sometimes, a recording is made at the survey location that is listened to and transcribed in the lab using headphones and possibly a computer screen. This presents new challenges, and also new opportunities for analysis of count data – and is the topic of the next chapter.

Chapter 6

Dealing with Recordings

integration challenges

calibration (exponential/cloglog approximation)

fixed effects

paired

sensor sensitivity - EDR

Chapter 7

A Closer Look at Assumptions

break them assumptions

Compare how does movement based detection influences estimates (when % increases)

36% of records in BAM is 'heard & seen' and I want to understand how likely the following 2 scenarios are:

- 'seen and heard' means that visual and auditory detections are independent processes, i.e. if it often happens that you spot a bird by just visually, that would distort availability.
- or it means that visual and auditory detections are NOT independent processes, i.e after a vocalization the observer also gets a visual.

Both happen, depending to some extent on the species, but more frequently the second scenario.

If you hear an individual vocalizing, it is much easier to also get a visual. But you do sometimes just see an individual moving and not vocalizing (more often females). How likely you are to spot those individuals can depend on the species, e.g., some species forage in ways that make them more visible.

Andy: My experience is that in forests it's almost entirely auditory, and if you see a bird it's almost certainly after it called. The rare exception might be birds foraging on the ground (e.g. thrushes kicking up leaves), or woodpeckers, but again these are usually heard before being seen even though the auditory

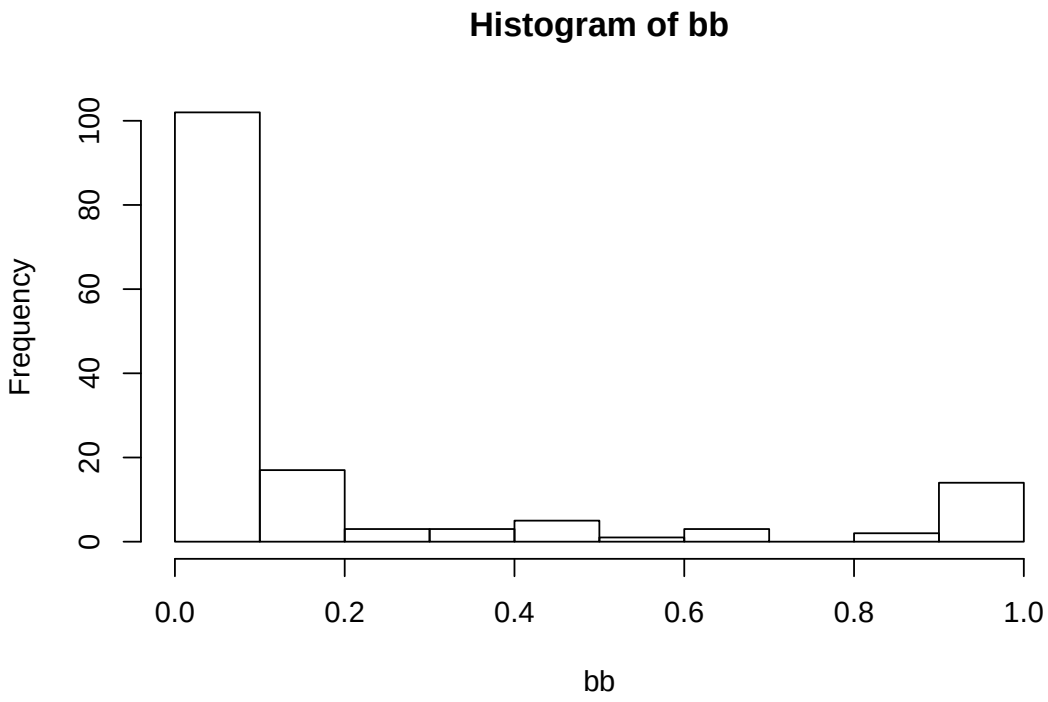
signal isn't a song or call. In grasslands, however, it's pretty common to see a bird before hearing it.

JT: I agree with you Andy, but I was thinking of species like American Redstarts that have a fairly flashy and obvious form of feeding as they chase and flycatch insects - it isn't uncommon to see these individuals before hearing them. Ultimately, to go back to Peter's concern, I do not think that the auditory and visual processes can be considered independent.

```
load("_data/josm/josm.rda") # JOSM data

100*table(josm$counts$DetectType1)/sum(table(josm$counts$DetectType1))
```

```
##
##      C      S      V
## 17.528 79.829  2.643
```

```
aa <- table(josm$counts$SpeciesID, josm$counts$DetectType1)
bb <- aa[,"V"]/rowSums(aa)
hist(bb)
```



```r
sort(bb[bb > 0.2])
```

```
##   ATTW   EAKI   TRES   BAEA   OSPR   BOGU   AMWI   MALL   MERL   RBGU
## 0.2414 0.2500 0.2520 0.3333 0.3333 0.3663 0.4444 0.4615 0.5000 0.5000
##   SPGR   SSHA   AMKE   BLBW   CAGU   RNDU   NOHA   BLTE   BUFF   BWTE
## 0.5000 0.5714 0.6364 0.6452 0.6667 0.8462 0.8571 0.9310 1.0000 1.0000
##   COGO   COHA   DCCO   GWTE   HOLA   NHOW   NSHO   RTHU   WWSC   CANV
## 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
##   NOPI
## 1.0000
```

Chapter 8

Understanding Roadside Surveys

directional diff in signal transmission

Chapter 9

Miscellaneous Topics

model selection and conditional likelihood

variance/bias trade off

error propagation

MCMC?

N-mixture ideas

phylogenetic and life history/trait stuff

PIF methods

These are just reminders, to be deleted later

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 1. If you do not manually label them, there will be automatic labels anyway.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

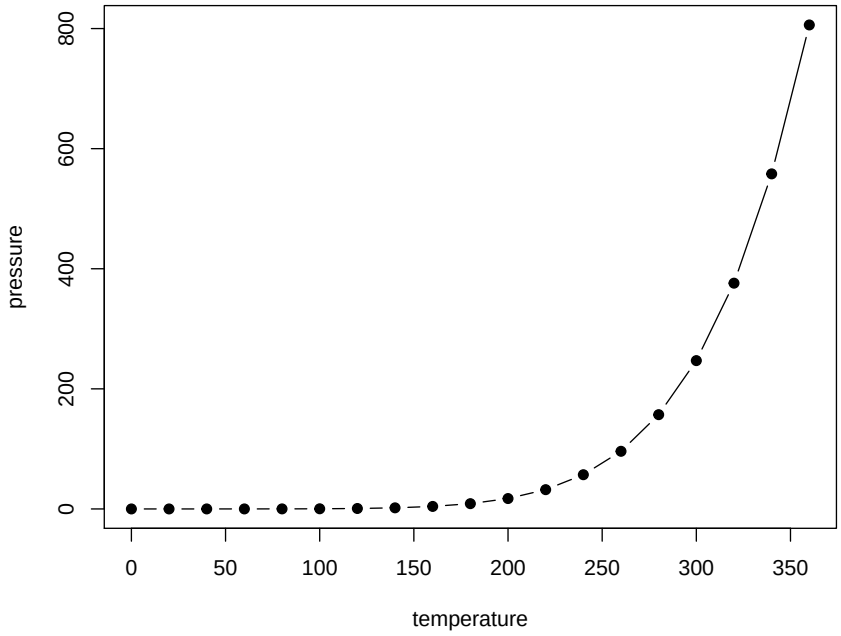


Figure 9.1: Here is a nice figure!

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 9.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 9.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

9.1 Binomial model and censoring

Try cloglog with a rare species, like BOCH

```
#spp <- "OVEN" # which species
spp <- "BOCH" # which species
#spp <- "CAWA" # which species
```

Table 9.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

```
x <- data.frame(
  josm$surveys,
  y=as.numeric(ytot[rownames(x), spp]))
x$y01 <- ifelse(x$y > 0, 1, 0)

table(x$y)


mP <- glm(y ~ Decid * ConifWet, x, family=poisson)
mBc <- glm(y01 ~ Decid * ConifWet, x, family=binomial("cloglog"))
mBl <- glm(y01 ~ Decid * ConifWet, x, family=binomial("logit"))
```

```
coef(mP)
coef(mBc)
coef(mBl)

plot(fitted(mBc) ~ fitted(mP), col=4,
  ylim=c(0, max(fitted(mP))), xlim=c(0, max(fitted(mP))))
points(exp(model.matrix(mBc) %*% coef(mBc)) ~ fitted(mP), col=2)
abline(0,1)
```

9.2 Optimal partitioning

```
oc <- opticut(as.matrix(ytot) ~ 1, strata = x$HAB, dist="poisson")
plot(oc)
```

9.3 Optilevels

When we have categorical or compositional (when e.g. proportions add up to 1, also called the unit sum constraint) data, we often want to simplify and merge classes or add up columns. We can do this based on the structural understanding of these land cover classes (call all treed classes Forest, like what we did for `FOR`, `WET` and `AHF`).

Alternatively, we can let the data (the birds) tell us how to merge the classes. The algorithm does the following:

1. fit model with all classes,
2. order estimates for each class from smallest to largest,
3. merge classes that are near each others, 2 at a time, moving from smallest to largest,
4. compare ΔAIC or ΔBIC values for the merged models and pick the smallest,
5. treat this best merged model as an input in step 1 and star over until Δ is negative (no improvement).

Here is the code for simplifying categories using the `opticut::optilevels` function:

```
M <- model.matrix(~HAB-1, x)
colnames(M) <- levels(x$HAB)
ol1 <- optilevels(x$y, M, dist="poisson")
sort(exp(coef(bestmodel(ol1))))
## estimates
exp(ol1$coef)
## optimal classification
ol1$rank
data.frame(combined_levels=ol1$levels[[length(ol1$levels)]])
```

Here is the code for simplifying compositional data:

```
ol2 <- optilevels(x$y, x[,cn], dist="poisson")
sort(exp(coef(bestmodel(ol2))))
## estimates
exp(ol2$coef)
## optimal classification
ol2$rank
head(groupSums(as.matrix(x[,cn]), 2, ol2$levels[[length(ol2$levels)]]))
```

9.4 N-mixture models

9.5 Estimating abundance

Exponential model, bSims data

Note: `$visits` is not yet exposed

```
set.seed(1)
phi <- 0.5
Den <- 1
l <- bsims_init()
a <- bsims_populate(l, density=Den)
b <- bsims_animate(a, vocal_rate=phi)
```

```
tint <- 1:5
(tr <- bsims_transcribe(b, tint=tint))
```

Multiple-visit stuff for bSims: the counting of new individuals resets for each interval (also: needs equal intervals)

```
tr$visits

v <- get_events(b, vocal_only=TRUE)
v <- v[v$t <= max(tint),]
v1 <- v[!duplicated(v$i),]

tmp <- v1
tmp$o <- seq_len(nrow(v1))
plot(o ~ t, tmp, type="n", ylab="Individuals",
  main="Vocalization events",
  ylim=c(1, nrow(b$nests)), xlim=c(0,max(tint)))
for (i in tmp$o) {
  tmp2 <- v[v$i == v1$i[i],]
  lines(c(tmp2$t[1], max(tint)), c(i,i), col="grey")
  points(tmp2$t, rep(i, nrow(tmp2)), cex=0.5)
  points(tmp2$t[1], i, pch=19, cex=0.5)
}


plot(o ~ t, tmp, type="n", ylab="Individuals",
  main="Vocalization events",
  ylim=c(1, nrow(b$nests)), xlim=c(0,max(tint)))
for (j in seq_along(tint)) {
  ii <- if (j == 1)
    c(0, tint[j]) else c(tint[j-1], tint[j])
  vv <- v[v$t > ii[1] & v$t <= ii[2],]
  tmp <- vv[!duplicated(vv$i),]
  tmp$o <- seq_len(nrow(tmp))
  if (nrow(tmp)) {
    for (i in tmp$o) {
      tmp2 <- vv[vv$i == tmp$i[i],]
```

```
      lines(c(tmp2$t[1], ii[2]), c(i,i), col="grey")
      points(tmp2$t, rep(i, nrow(tmp2)), cex=0.5)
      points(tmp2$t[1], i, pch=19, cex=0.5)
    }
  }
}

library(unmarked)

f <- function() {
  a <- bsims_populate(l, density=Den)
  b <- bsims_animate(a, vocal_rate=phi, move_rate=0)
  tr <- bsims_transcribe(b, tint=tint)
  drop(tr$visits)
}

Den <- 0.01

#ymx <- tr$visits
(ymx <- t(replicate(10, f())))

## highly dependent on K when Den is higher
nmix <- pcount(~1 ~1, unmarkedFramePCount(y=ymx), K=1000)
coef(nmix)
plogis(coef(nmix)[2])
exp(coef(nmix)[1])
Den * 100
```

Bibliography

Mahon, C. L., Holloway, G., Sólymos, P., Cumming, S. G., Bayne, E. M., Schmiegelow, F. K. A., and Song, S. J. (2016). Community structure and niche characteristics of upland and lowland western boreal birds at multiple spatial scales. *Forest Ecology and Management*, 361:99–116.

Matsuoka, S. M., Mahon, C. L., Handel, C. M., Sólymos, P., Bayne, E. M., Fontaine, P. C., and Ralph, C. J. (2014). Reviving common standards in point-count surveys for broad inference across studies. *Condor*, 116:599–608.