Supporting Information:
Calibrating indices of avian density from non-standardized survey data: making the most of a messy situation

Péter Sólymos Steven M. Matsuoka Erin M. Bayne
solymos@ualberta.ca

Subhash R. Lele Patricia Fontaine Steven G. Cumming Diana Stralberg
Fiona K. A. Schmiegelow Samantha J. Song

May 29, 2013

Contents

Chapter 1

Introduction

This document provides Supporting Information for the manuscript entitled "Calibrating indices of avian density from non-standardized survey data: making the most of a messy situation" by Sólymos et al. submitted to the journal *Methods in Ecology and Evolution* (ID MEE-12-11-472).

The purpose of this document is to describe:

1. how to estimate QPAD model parameters;
2. how to retrieve the QPAD model parameter estimates reported in the paper;
3. and how to use these estimates in statistical inference and prediction.

1.1 Software requirements

- R for most of the calculations ((R Core Team, 2012), downloadable from http://www.r-project.org/);
- JAGS for prediction interval calculations using Markov chain Monte Carlo (MCMC) (Plummer, 2012, downloadable from http://mcmc-jags.sourceforge.net/)
- R extension packages: **detect** (Sólymos et al., 2013), **dcmle** (Sólymos, 2010) (can be installed from R console using the `install.packages` function).

1.2 Files

- `BAM_QPAD_coefs_20130226.R`: sourceable text file with estimated model parameters (available from http://dcr.r-forge.r-project.org/qpad/BAM_QPAD_coefs_20130226.R);
- `BAM_QPAD_functions_20130226.R`: sourceable text file with R functions for retrieving the information from the binary R data file (available from http://dcr.r-forge.r-project.org/qpad/BAM_QPAD_functions_20130226.R).

These files can be sourced by using the `load_BAM_QPAD()` function of the **detect** R package. This function allows the user to select versions of the estimates interactively, this way updates to these files are available upon request. Using `load_BAM_QPAD(version=1)` loads the version used in this document.

Chapter 2

Conditional multinomial maximum likelihood estimation

The estimation procedure described in the Appendix is implemented in the `cmulti` function of the **detect** R extension package. Input data specifications are described in the help page of the function (type `?cmulti` into R console).

```
library(detect)

Loading required package:  Formula
Loading required package:  stats4
detect 0.3-0 2013-02-26
```

2.1 Input data

Let us use 100 survey locations (`n`), and generate random values for the covariate `x`:

```
n <- 100
x <- rnorm(n)
X <- cbind(1, x)
```

The function `simfun1` is used to simulate count data (value for `tau` (Effective Detection Radius; EDR) is in units of 100 m:

```
simfun1 <- function(n = 10, phi = 0.1, c = 1, tau = 0.8, type = "rem") {
    if (type == "dis") {
        Dparts <- matrix(c(0.5, 1, NA, 0.5, 1, Inf, 1, Inf, NA),
            3, 3, byrow = TRUE)
        D <- Dparts[sample.int(3, n, replace = TRUE), ]
        CP <- 1 - exp(-(D/tau)^2)
    } else {
        Dparts <- matrix(c(5, 10, NA, 3, 5, 10, 3, 5, NA), 3,
            3, byrow = TRUE)
        D <- Dparts[sample.int(3, n, replace = TRUE), ]
        CP <- 1 - c * exp(-D * phi)
    }
    k <- ncol(D)
```

```
    P <- CP - cbind(0, CP[, -k, drop = FALSE])
    Psum <- rowSums(P, na.rm = TRUE)
    PPsum <- P/Psum
    Pok <- !is.na(PPsum)
    N <- rpois(n, 10)
    Y <- matrix(NA, ncol(PPsum), nrow(PPsum))
    Ypre <- sapply(1:n, function(i) rmultinom(1, N, PPsum[i,
        Pok[i, ]]))
    Y[t(Pok)] <- unlist(Ypre)
    Y <- t(Y)
    list(Y = Y, D = D)
}
```

Now let us simulate counts under the removal model using constant singing rate `phi`. The count matrix `Y` contains the number of unique individuals first observed in time intervals defined in the design matrix `D` which contains the enpoints of the corresponding time intervals in minutes. Note that patterns in `NA` values muxt match between the two matrices:

```
vv <- simfun1(n = n, phi = exp(-1.5))
head(vv$Y)

     [,1] [,2] [,3]
[1,]    7    2    1
[2,]    6    4   NA
[3,]    6    3    1
[4,]    5    1    4
[5,]    5    5   NA
[6,]    6    2    2

head(vv$D)

     [,1] [,2] [,3]
[1,]    3    5   10
[2,]    3    5   NA
[3,]    3    5   10
[4,]    3    5   10
[5,]    5   10   NA
[6,]    3    5   10
```

2.2 Removal sampling

Estimation is done using the `cmulti` function. The left hand side of the formula reads as `count | design` where `count` is a matrix with cell counts, `design` is the matrix describing the interval endpoints for the cells. The right hand side is `1` because we use a constant model, `type="rem"` stands for removal sampling:

```
m1 <- cmulti(vv$Y | vv$D ~ 1, type = "rem")
coef(m1)

log.phi_(Intercept)
             -1.467
```

When covariate `x` affects singing rate, the estimation is as follows:

```
log.phi <- X %*% c(-2, -1)  # log singing rate
vv <- simfun1(n = n, phi = exp(cbind(log.phi, log.phi, log.phi)))
m2 <- cmulti(vv$Y | vv$D ~ x, type = "rem")
coef(m2)

log.phi_(Intercept)           log.phi_x
             -1.943              -0.935
```

2.3 Distance sampling

Simulation and estimation for the distance sampling model with half-normal detection function is similar. We use constant `tau` parameter (EDR, 100 m units, use`Inf` for unlimited distance):

```
vv <- simfun1(n = n, tau = exp(-0.2), type = "dis")
head(vv$Y)

     [,1] [,2] [,3]
[1,]    7    1   NA
[2,]    2    6    0
[3,]    4    4   NA
[4,]    2    6   NA
[5,]    2    6   NA
[6,]    2    6   NA

head(vv$D)

     [,1] [,2] [,3]
[1,]  1.0  Inf   NA
[2,]  0.5    1  Inf
[3,]  0.5    1   NA
[4,]  0.5    1   NA
[5,]  0.5    1   NA
[6,]  0.5    1   NA

m3 <- cmulti(vv$Y | vv$D ~ 1, type = "dis")
coef(m3)

log.tau_(Intercept)
            -0.2287
```

Effect of covariate `x` is estimated as:

```
log.tau <- X %*% c(-0.5, -0.2)  # log EDR
vv <- simfun1(n = n, tau = exp(cbind(log.tau, log.tau, log.tau)),
    type = "dis")
m4 <- cmulti(vv$Y | vv$D ~ x, type = "dis")
coef(m4)

log.tau_(Intercept)           log.tau_x
            -0.4645             -0.1977
```

2.4 Convenience methods

Several methods are defined for the fitted model objects to facilitate statistical inference:

```
summary(m2)


Call:
cmulti(formula = vv$Y | vv$D ~ x, type = "rem")

Removal Sampling (homogeneous singing rate)
Conditional Maximum Likelihood estimates

Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
log.phi_(Intercept)   -1.943      0.143   -13.6  < 2e-16
log.phi_x             -0.935      0.144    -6.5  8.2e-11

log.phi_(Intercept) ***
log.phi_x           ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-likelihood: -198
BIC =  406

summary(m4)


Call:
cmulti(formula = vv$Y | vv$D ~ x, type = "dis")

Distance Sampling (half-normal, circular area)
Conditional Maximum Likelihood estimates

Coefficients:
                     Estimate Std. Error z value Pr(>|z|)
log.tau_(Intercept)  -0.4645     0.0249  -18.66  < 2e-16
log.tau_x            -0.1977     0.0259   -7.64  2.2e-14

log.tau_(Intercept) ***
log.tau_x           ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-likelihood: -156
BIC =  321

coef(m4)

log.tau_(Intercept)           log.tau_x
            -0.4645             -0.1977

vcov(m4)
```

```
                    log.tau_(Intercept)  log.tau_x
log.tau_(Intercept)           6.193e-04 -5.928e-05
log.tau_x                    -5.928e-05  6.701e-04

AIC(m4)

[1] 316.1

confint(m4)

                      2.5 %  97.5 %
log.tau_(Intercept) -0.5133 -0.4157
log.tau_x           -0.2484 -0.1469

logLik(m4)

'log Lik.' -156.1 (df=2)
```

2.5 Joint estimation for removal and distance sampling

The following function simulates counts for the joint estimation of singing rate and distance parameters:

```
simfun12 <- function(n = 10, phi = 0.1, c = 1, tau = 0.8, type = "rem") {
    Flat <- function(x, DIM, dur = TRUE) {
        x <- array(x, DIM)
        if (!dur) {
            x <- aperm(x, c(1, 3, 2))
        }
        dim(x) <- c(DIM[1], DIM[2] * DIM[3])
        x
    }
    Dparts1 <- matrix(c(5, 10, NA, 3, 5, 10, 3, 5, NA), 3, 3,
        byrow = TRUE)
    D1 <- Dparts1[sample.int(3, n, replace = TRUE), ]
    CP1 <- 1 - c * exp(-D1 * phi)
    Dparts2 <- matrix(c(0.5, 1, NA, 0.5, 1, Inf, 1, Inf, NA),
        3, 3, byrow = TRUE)
    D2 <- Dparts2[sample.int(3, n, replace = TRUE), ]
    CP2 <- 1 - exp(-(D2/tau)^2)
    k1 <- ncol(D1)
    k2 <- ncol(D2)
    DIM <- c(n, k1, k2)
    P1 <- CP1 - cbind(0, CP1[, -k1, drop = FALSE])
    P2 <- CP2 - cbind(0, CP2[, -k2, drop = FALSE])
    Psum1 <- rowSums(P1, na.rm = TRUE)
    Psum2 <- rowSums(P2, na.rm = TRUE)
    Pflat <- Flat(P1, DIM, dur = TRUE) * Flat(P2, DIM, dur = FALSE)
    PsumFlat <- Psum1 * Psum2
    PPsumFlat <- Pflat/PsumFlat
```

```
    PokFlat <- !is.na(PPsumFlat)
    N <- rpois(n, 10)
    Yflat <- matrix(NA, ncol(PPsumFlat), nrow(PPsumFlat))
    YpreFlat <- sapply(1:n, function(i) rmultinom(1, N, PPsumFlat[i,
        PokFlat[i, ]]))
    Yflat[t(PokFlat)] <- unlist(YpreFlat)
    Yflat <- t(Yflat)
    Y <- array(Yflat, DIM)
    k1 <- dim(Y)[2]
    k2 <- dim(Y)[3]
    Y1 <- t(sapply(1:n, function(i) {
        count <- rowSums(Y[i, , ], na.rm = TRUE)
        nas <- rowSums(is.na(Y[i, , ]))
        count[nas == k2] <- NA
        count
    }))
    Y2 <- t(sapply(1:n, function(i) {
        count <- colSums(Y[i, , ], na.rm = TRUE)
        nas <- colSums(is.na(Y[i, , ]))
        count[nas == k2] <- NA
        count
    }))
    list(Y = Y, D1 = D1, D2 = D2, Y1 = Y1, Y2 = Y2)
}
```

Joint and independent estimation of constant singing rate and EDR:

```
vv <- simfun12(n = n, phi = exp(-1.5), tau = exp(-0.2))
res <- cmulti2.fit(vv$Y, vv$D1, vv$D2)
res1 <- cmulti.fit(vv$Y1, vv$D1, NULL, "rem")
res2 <- cmulti.fit(vv$Y2, vv$D2, NULL, "dis")
```

Jointly and independently estimated points estimates and standard errors are identical, the two models are orthogonal (correlation is 0):

```
round(cbind(coef.joint = res$coef, coef.indep = c(res1$coef,
    res2$coef)), 4)

     coef.joint coef.indep
[1,]    -1.4178    -1.4178
[2,]    -0.2284    -0.2284

round(cbind(SE.joint = sqrt(diag(res$vcov)), SE.indep = c(sqrt(diag(res1$vcov)),
    sqrt(diag(res2$vcov)))), 4)

     SE.joint SE.indep
[1,]   0.0657   0.0657
[2,]   0.0228   0.0228

ifelse(cov2cor(res$vcov) < 10^-10, 0, cov2cor(res$vcov))

     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Joint and independent estimation of covariate specific singing rate and EDR:

```
vv <- simfun12(n = n, phi = exp(cbind(log.phi, log.phi, log.phi)),
    tau = exp(cbind(log.tau, log.tau, log.tau)))
res <- cmulti2.fit(vv$Y, vv$D1, vv$D2, X1 = X, X2 = X)
res1 <- cmulti.fit(vv$Y1, vv$D1, X, "rem")
res2 <- cmulti.fit(vv$Y2, vv$D2, X, "dis")
```

Jointly and independently estimated points estimates and standard errors are identical, the two models are orthogonal (correlation is 0):

```
round(cbind(coef.joint = res$coef, coef.indep = c(res1$coef,
    res2$coef)), 4)

     coef.joint coef.indep
[1,]    -1.9112    -1.9112
[2,]    -0.8166    -0.8166
[3,]    -0.4914    -0.4914
[4,]    -0.2339    -0.2339

round(cbind(SE.joint = sqrt(diag(res$vcov)), SE.indep = c(sqrt(diag(res1$vcov)),
    sqrt(diag(res2$vcov)))), 4)

     SE.joint SE.indep
[1,]   0.1086   0.1086
[2,]   0.1125   0.1125
[3,]   0.0188   0.0188
[4,]   0.0200   0.0200

round(ifelse(cov2cor(res$vcov) < 10^-10, 0, cov2cor(res$vcov)),
    4)

       [,1]   [,2] [,3] [,4]
[1,] 1.0000 0.7453    0    0
[2,] 0.7453 1.0000    0    0
[3,] 0.0000 0.0000    1    0
[4,] 0.0000 0.0000    0    1
```

Chapter 3

Retrieving QPAD estimates

First we need to load necessary code and data after opening R (commands can be copy-pasted from the document, # marks comments):

```
library(detect)  # load detect package
# source estimates and functions
load_BAM_QPAD(version = 1)

BAM QPAD parameter estimates loaded, version 20130226
BAM QPAD access functions loaded, version 20130226
```

Print out the list of species acronyms (printing out a table linking acronyms to common and scientific names can be done by the `getBAMspeciestable` function):

```
getBAMspecieslist()

 [1] "ALFL" "AMCR" "AMGO" "AMRE" "AMRO" "ATSP" "BAWW" "BBWA"
 [9] "BCCH" "BHCO" "BHVI" "BLBW" "BLJA" "BLPW" "BOCH" "BRCR"
[17] "BTBW" "BTNW" "CAWA" "CCSP" "CEDW" "CHSP" "CMWA" "CONW"
[25] "CORA" "COYE" "CSWA" "DEJU" "EVGR" "FOSP" "GCKI" "GRAJ"
[33] "HAFL" "HETH" "HOWR" "LCSP" "LEFL" "LISP" "MAWA" "MOWA"
[41] "NAWA" "NOPA" "NOWA" "OCWA" "OSFL" "OVEN" "PAWA" "PHVI"
[49] "PISI" "PUFI" "RBGR" "RBNU" "RCKI" "REVI" "RUBL" "RWBL"
[57] "SAVS" "SOSP" "SWSP" "SWTH" "TEWA" "TRES" "VATH" "VEER"
[65] "WAVI" "WCSP" "WETA" "WEWP" "WIWA" "WIWR" "WTSP" "WWCR"
[73] "YBFL" "YRWA" "YWAR"
```

Print out the list of models used for estimating singing rates (`sra`) and effective detection radii (`edr`):

```
getBAMmodellist()

$sra
                                    0
                            "INTERCEPT"
                                    1
                      "INTERCEPT + JDAY"
                                    2
                      "INTERCEPT + TSSR"
```

```
                                        3
                "INTERCEPT + JDAY + JDAY2"
                                        4
                "INTERCEPT + TSSR + TSSR2"
                                        5
                 "INTERCEPT + JDAY + TSSR"
                                        6
         "INTERCEPT + JDAY + JDAY2 + TSSR"
                                        7
         "INTERCEPT + JDAY + TSSR + TSSR2"
                                        8
"INTERCEPT + JDAY + JDAY2 + TSSR + TSSR2"

$edr
                                     0
                             "INTERCEPT"
                                     1
                      "INTERCEPT + TREE"
                                     2
"INTERCEPT + LCC2 + LCC3 + LCC4 + LCC5"
```

Get version info:

```
getBAMversion()

[1] "2"
```

The species acronym `OVEN` stands for Ovenbird (*Seiurus aurocapilla*). This gives the estimated parameters for the Ovenbird from the models best supported by BIC:

```
summaryBAMspecies("OVEN")

BAMcorrection object for species OVEN
model.sra = 1
model.edr = 2
              Estimate Std. Error
sra_INTERCEPT   0.9005       0.20
sra_JDAY       -3.8318       0.45
edr_INTERCEPT  -0.1472       0.01
edr_LCC2       -0.0777       0.01
edr_LCC3        0.0147       0.02
edr_LCC4       -0.0329       0.01
edr_LCC5       -0.0362       0.02
```

The best supported model is returned by:

```
bestmodelBAMspecies("OVEN", type = "BIC")

$sra
[1] "1"

$edr
[1] "2"
```

The type argument can take values `"AIC"`, `"BIC"` or `"multi"` (the latter returns model IDs randomly based on model weights).

It is also possible to print out other model combinations (model IDs can be looked up from the `getBAMmodellist` function):

```
summaryBAMspecies("OVEN", model.sra = 8, model.edr = 1)

BAMcorrection object for species OVEN
model.sra = 8
model.edr = 1
              Estimate Std. Error
sra_INTERCEPT   0.5055       2.85
sra_JDAY       -1.9690      12.69
sra_JDAY2      -2.0578      14.07
sra_TSSR       -0.6820       0.63
sra_TSSR2       3.1629       3.14
edr_INTERCEPT  -0.1694       0.02
edr_TREE       -0.0152       0.03
```

To get all possible models compared, use this:

```
selectmodelBAMspecies("OVEN")

$sra
  model logLik df nobs   AIC   BIC   dAIC   dBIC   weights
0     0  -5178  1 6789 10357 10364 72.467 65.644 8.697e-17
1     1  -5141  2 6789 10285 10299  0.000  0.000 4.736e-01
2     2  -5178  2 6789 10359 10373 74.378 74.378 3.346e-17
3     3  -5141  3 6789 10287 10308  2.210  9.033 1.569e-01
4     4  -5177  3 6789 10361 10381 75.904 82.727 1.560e-17
5     5  -5140  3 6789 10287 10307  1.819  8.642 1.907e-01
6     6  -5140  4 6789 10289 10316  3.895 17.541 6.755e-02
7     7  -5140  4 6789 10288 10316  3.443 17.089 8.469e-02
8     8  -5140  5 6789 10291 10325  5.762 26.231 2.656e-02

$edr
  model logLik df  nobs   AIC   BIC  dAIC  dBIC   weights
0     0 -21791  1 21742 43583 43591 57.12 25.18 3.941e-13
1     1 -21791  2 21742 43585 43601 58.92 34.96 1.603e-13
2     2 -21758  5 21742 43526 43566  0.00  0.00 1.000e+00
```

The column `weights` indicates model weights used by `bestmodelBAMspecies` with argument `type="multi"`.

Chapter 4

Example data analysis

4.1 The Ovenbird data set

We used a data set of Ovenbirds analyzed by Lele et al. (2011) and by Sólymos et al. (2012), available from the **detect** R extension package (data set name: `oven`). The Ovenbird study was conducted in Saskatchewan, Canada. Point counts were sampled according to the standards of the North American Breeding Bird Survey (3 minutes unlimited distance counts).

First, we calculate necessary covariates for the offsets. `JDAY` is Julian day in the unit range (divided by 365), `TSSR` is calculated approximately in hours and divided by the the possible max (24):

```
oven$JDAY <- oven$julian/365
oven$TSSR <- ((oven$timeday/8) - 0.75)/24
oven$xlat <- as.numeric(scale(oven$lat))  # latitude is standardized
oven$xlong <- as.numeric(scale(oven$long))  # longitude is standardized
```

Introduce variables for protocol effects, duration (3 min) and point count radius (note that this has to be in 100 metres for density per ha, use `Inf` for unlimited distance):

```
oven$dur <- 3
oven$dist <- Inf
```

The covariates for distance sampling are derived from the proportion of forest and proportion of deciduous forest:

```
pf <- oven$pforest
pd <- oven$pdecid
pc <- pf - pd
oven$LCC <- factor(5, levels = 1:5)  # 5=OH open habitat
oven$LCC[pf > 0.25 & pc > pd] <- "3"  # 3=SC dense conifer
oven$LCC[pf > 0.25 & pc <= pd] <- "4"  # 4=SD sparse deciduous
oven$LCC[pf > 0.6 & pc > pd] <- "1"  # 1=DC dense conifer
oven$LCC[pf > 0.6 & pc <= pd] <- "2"  # 2=DC dense deciduous
table(oven$LCC)


  1   2   3   4   5
 66 225  25 175 400
```

4.1.1 QPAD offsets

Here is how one can calculate the offsets based on the estimates without covariate effects:

```
bc0 <- with(oven, globalBAMcorrections("OVEN", t = dur, r = dist))
summary(bc0)

       A             p              q
 Min.   :2.2   Min.   :0.73   Min.   :1
 1st Qu.:2.2   1st Qu.:0.73   1st Qu.:1
 Median :2.2   Median :0.73   Median :1
 Mean   :2.2   Mean   :0.73   Mean   :1
 3rd Qu.:2.2   3rd Qu.:0.73   3rd Qu.:1
 Max.   :2.2   Max.   :0.73   Max.   :1
```

The offsets based on possible covariate effects can be calculated as:

```
bm <- bestmodelBAMspecies("OVEN", type = "BIC")
bc <- with(oven, localBAMcorrections("OVEN", t = dur, r = dist,
    jday = JDAY, tssr = TSSR, tree = pforest, lcc = LCC, model.sra = bm$sra,
    model.edr = bm$edr))
summary(bc)

       A              p               q
 Min.   :2.00   Min.   :0.649   Min.   :1
 1st Qu.:2.00   1st Qu.:0.672   1st Qu.:1
 Median :2.18   Median :0.710   Median :1
 Mean   :2.15   Mean   :0.705   Mean   :1
 3rd Qu.:2.19   3rd Qu.:0.733   3rd Qu.:1
 Max.   :2.41   Max.   :0.766   Max.   :1
```

4.2 Inference: point estimation

4.2.1 Poisson GLM

Here is the general way how one can specify the offsets and estimate density, for example here using Poisson generalized linear model (GLM):

```
(mod <- glm(count ~ pforest + xlong, oven, family = poisson("log"),
    offset = corrections2offset(bc)))


Call:  glm(formula = count ~ pforest + xlong, family = poisson("log"),
    data = oven, offset = corrections2offset(bc))

Coefficients:
(Intercept)      pforest        xlong
    -2.5651       2.6349      -0.0715

Degrees of Freedom: 890 Total (i.e. Null);  888 Residual
Null Deviance:     1260
Residual Deviance: 871  AIC: 1510
```

Such models are suitable for calculating point predictions for example for mapping density, but do not take into account the uncertainty associated with the singing rate and EDR estimates underlying the offsets:

```
X <- model.matrix(~pforest + xlong, data.frame(pforest = 10:0/10,
    xlong = 0))
summary(drop(exp(X %*% coef(mod))))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0769  0.1500  0.2870  0.3980  0.5600  1.0700
```

4.2.2 Negative Binomial GLM

The Negative Binomial model is useful in case of overdispersion due to e.g. missing covariate. The Negative Binomial GLM also uses the log link so the usual offset works. Care must be taken because the offset argument of the `glm.nb` function in the **MASS** library (Venables and Ripley, 2002) is not available. So the offset must be defined as part of the formula:

```
library(MASS)
(modNB <- glm.nb(count ~ pforest + xlong + offset(corrections2offset(bc)),
    oven))


Call:  glm.nb(formula = count ~ pforest + xlong + offset(corrections2offset(bc)),
    data = oven, init.theta = 1.974529194, link = log)

Coefficients:
(Intercept)      pforest        xlong
    -2.6161       2.7226      -0.0654

Degrees of Freedom: 890 Total (i.e. Null);  888 Residual
Null Deviance:     986
Residual Deviance: 673  AIC: 1490

round(cbind(Pois = coef(mod), NegBin = coef(modNB)), 3)

              Pois NegBin
(Intercept) -2.565 -2.616
pforest      2.635  2.723
xlong       -0.072 -0.065
```

4.2.3 Poisson–Lognormal mixed effects model

Poisson mixed effects models can be fitted for example via the **lme4** package (Bates et al., 2012) using random intercept for routes:

```
library(lme4)

Loading required package:  Matrix
Loading required package:  lattice
```

```
Attaching package:  'lme4'
The following object is masked from 'package:stats':
       AIC, BIC
```

```
mod4 <- glmer(count ~ pforest + xlong + (1 | route), oven, family = "poisson",
    offset = corrections2offset(bc))
round(cbind(Pois = coef(mod), NegBin = coef(modNB), PLn = fixef(mod4)),
    3)
```

```
             Pois NegBin    PLn
(Intercept) -2.565 -2.616 -2.695
pforest      2.635  2.723  2.587
xlong       -0.072 -0.065 -0.093
```

4.2.4 Logistic regression

For detection/non-detection situations, here is the suggested modification. Note that probability of having 1 means probability of observing non-zero (> 0) counts within the sampling area. using the complementary log–log link is related to density from Poisson GLM with log link:

```
mod01 <- glm(ifelse(count > 0, 1, 0) ~ pforest + xlong, oven,
    family = binomial("cloglog"), offset = corrections2offset(bc))
round(cbind(Pois = coef(mod), NegBin = coef(modNB), PLn = fixef(mod4),
    Bin = coef(mod01)), 3)
```

```
             Pois NegBin    PLn    Bin
(Intercept) -2.565 -2.616 -2.695 -2.818
pforest      2.635  2.723  2.587  2.697
xlong       -0.072 -0.065 -0.093 -0.048
```

4.2.5 Zero-inflated count models

It is possible to use offsets in the count distribution of zero-inflated models, such as zero-inflated Poisson or zero-inflated Negative Binomial model as implemented in the `zeroinfl` function of the **pscl** package (Zeileis et al., 2008):

```
library(pscl)
```

```
Loading required package:  mvtnorm
Loading required package:  coda
```

```
Attaching package:  'coda'
The following object is masked from 'package:lme4':
       HPDinterval
Loading required package:  gam
Loading required package:  splines
Loaded gam 1.06.2
Loading required package:  vcd
Loading required package:  grid
```

```
Loading required package:  colorspace
Classes and Methods for R developed in the
Political Science Computational Laboratory
Department of Political Science
Stanford University
Simon Jackman
hurdle and zeroinfl functions by Achim Zeileis
```

```
modZIP <- zeroinfl(count ~ pforest + xlong | 1, oven, dist = "poisson",
    offset = corrections2offset(bc))
modZINB <- zeroinfl(count ~ pforest + xlong | 1, oven, dist = "negbin",
    offset = corrections2offset(bc))
round(cbind(Pois = c(coef(mod), ZI = NA), NegBin = c(coef(modNB),
    NA), PLn = c(fixef(mod4), NA), Bin = c(coef(mod01), NA),
    ZIP = coef(modZIP), ZINB = coef(modZINB)), 3)
```

```
              Pois NegBin    PLn    Bin    ZIP   ZINB
(Intercept) -2.565 -2.616 -2.695 -2.818 -2.273 -2.350
pforest      2.635  2.723  2.587  2.697  2.612  2.649
xlong       -0.072 -0.065 -0.093 -0.048 -0.049 -0.054
ZI              NA     NA     NA     NA -1.114 -1.368
```

4.2.6 Classification and regression trees (CART)

This is how a CART model can be specified using the offset approach using the **rpart** package (Therneau et al., 2012). Note that the specification has the linear predictor for the Poisson rate on the response scale ($\lambda = f(x)$ when `method = "poisson"`, see package vignette), which needs to be standardized by the correction, and not via offsets (Fig. 4.1):

```
library(rpart)
oven$C <- corrections(bc)
(cart <- rpart((count/C) ~ pforest + xlong, data = oven, method = "poisson"))
```

```
n= 891

node), split, n, deviance, yval
      * denotes terminal node

 1) root 891 836.30 0.35030
   2) pforest< 0.3616 489 197.90 0.08832
     4) pforest< 0.1168 283  43.00 0.02592 *
     5) pforest>=0.1168 206 121.80 0.17730 *
   3) pforest>=0.3616 402 406.50 0.66850
     6) pforest< 0.93 301 278.80 0.54370
      12) xlong< -0.579 72  51.94 0.29850 *
      13) xlong>=-0.579 229 214.60 0.62040
        26) xlong>=0.007684 161 132.40 0.46700 *
        27) xlong< 0.007684 68  63.07 0.96430
          54) pforest< 0.6831 38  37.75 0.64020 *
          55) pforest>=0.6831 30  16.18 1.31400 *
     7) pforest>=0.93 101 102.20 1.02500 *
```

Alternatively, one can use the "anova" with a log transformed response variable. Note that predicted values need to be back-transformed (Fig. 4.1):

```
(cart2 <- rpart(log(count + 0.5) ~ pforest + xlong + offset(corrections2offset(bc)),
    data = oven, method = "anova"))

n= 891

node), split, n, deviance, yval
      * denotes terminal node

 1) root 891 431.50000 -0.68130
   2) pforest< 0.4036 521  89.56000 -0.97700
     4) pforest< 0.2014 373  30.71000 -1.04500 *
     5) pforest>=0.2014 148  52.78000 -0.80550 *
   3) pforest>=0.4036 370 232.30000 -0.26490
     6) pforest< 0.8809 212 119.80000 -0.43000
      12) xlong>=1.673 22   6.92200 -0.87140 *
      13) xlong< 1.673 190 108.10000 -0.37890
        26) xlong< -0.579 50  20.29000 -0.68070 *
        27) xlong>=-0.579 140  81.62000 -0.27110 *
     7) pforest>=0.8809 158  98.94000 -0.04337
      14) xlong>=1.693 23   9.82600 -0.61080
        28) pforest>=0.9159 11   0.08456 -1.06600 *
        29) pforest< 0.9159 12   5.36900 -0.19330 *
      15) xlong< 1.693 135  80.45000  0.05330
        30) xlong< -1.193 55  33.41000 -0.13270
          60) xlong>=-1.402 23  13.86000 -0.51300 *
          61) xlong< -1.402 32  13.83000  0.14070 *
        31) xlong>=-1.193 80  43.83000  0.18120 *

opar <- par(mfrow = c(2, 1), xpd = NA)
plot(cart)
text(cart)
title(main = "poisson")
plot(cart2)
text(cart2)
title(main = "anova")
par(opar)
```

The **rpart** based tree model has issues with handling zero observations, therefore its use is not highly recommended for sparse counts.

4.2.7 Boosted regression trees

This code snipped demonstrates how the offsets can be specified using the **gbm** R package (Ridgeway, 2013) (Fig. 4.2). Of course one might want to use more covariates in such cases, and more advanced settings for determining learning rate e.g. through the **dismo** R package (Hijmans et al., 2013). Note the use of Poisson distribution, where the gradient function is calculated on the log scale ($(\lambda = e^{f(x)})$, see package vignette for specifications), therefore the use of the additive offset is justified:

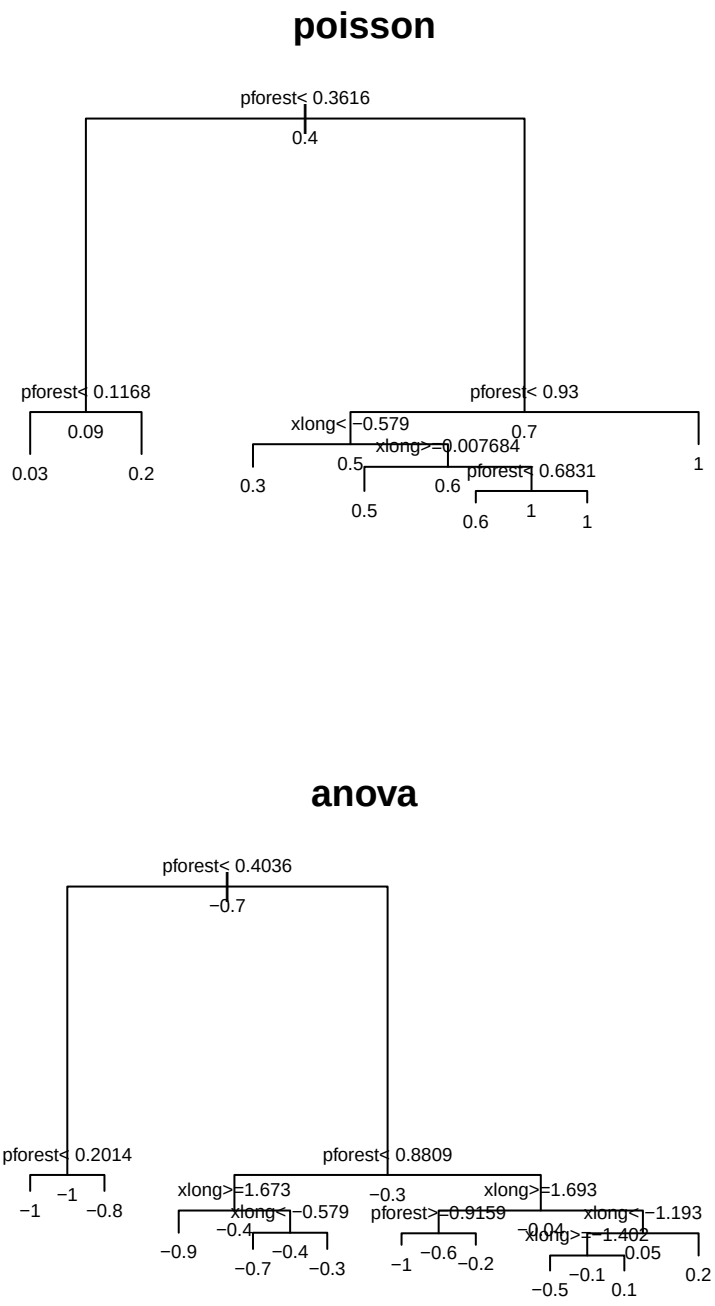

Figure 4.1: Regression tree analysis of the Ovenbird data set using QPAD offsets.

```
library(gbm)

Loading required package:  survival
Loaded gbm 2.0-8

oven$off <- corrections2offset(bc)
(brt <- gbm(count ~ pforest + xlong + offset(off), data = oven,
    distribution = "poisson"))

Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.7366             nan     0.0010    0.0002
     2        1.7357             nan     0.0010    0.0004
     3        1.7349             nan     0.0010    0.0004
     4        1.7344             nan     0.0010    0.0002
     5        1.7334             nan     0.0010    0.0005
     6        1.7327             nan     0.0010    0.0004
     7        1.7321             nan     0.0010    0.0003
     8        1.7312             nan     0.0010    0.0004
     9        1.7305             nan     0.0010    0.0003
    10        1.7300             nan     0.0010    0.0002
    20        1.7219             nan     0.0010    0.0004
    40        1.7052             nan     0.0010    0.0004
    60        1.6907             nan     0.0010    0.0004
    80        1.6766             nan     0.0010    0.0001
   100        1.6634             nan     0.0010    0.0004
gbm(formula = count ~ pforest + xlong + offset(off), distribution = "poisson",
    data = oven)
A gradient boosted model with poisson loss function.
100 iterations were performed.
There were 2 predictors of which 1 had non-zero influence.

plot(brt)
```

4.2.8 Regularization approaches

This example shows the regularized Poisson GLM using the elastic net penalty using the **glmnet** R package (Friedman et al., 2010) (Fig. 4.3):

```
library(glmnet)

Loaded glmnet 1.9-3

enet <- glmnet(model.matrix(mod), mod$y, family = "poisson",
    offset = corrections2offset(bc))
coef(enet)

4 x 55 sparse Matrix of class "dgCMatrix"

  [[ suppressing 55 column names 's0', 's1', 's2' ...  ]]


(Intercept) -1.063 -1.1573 -1.246 -1.3310 -1.411 -1.4876
```

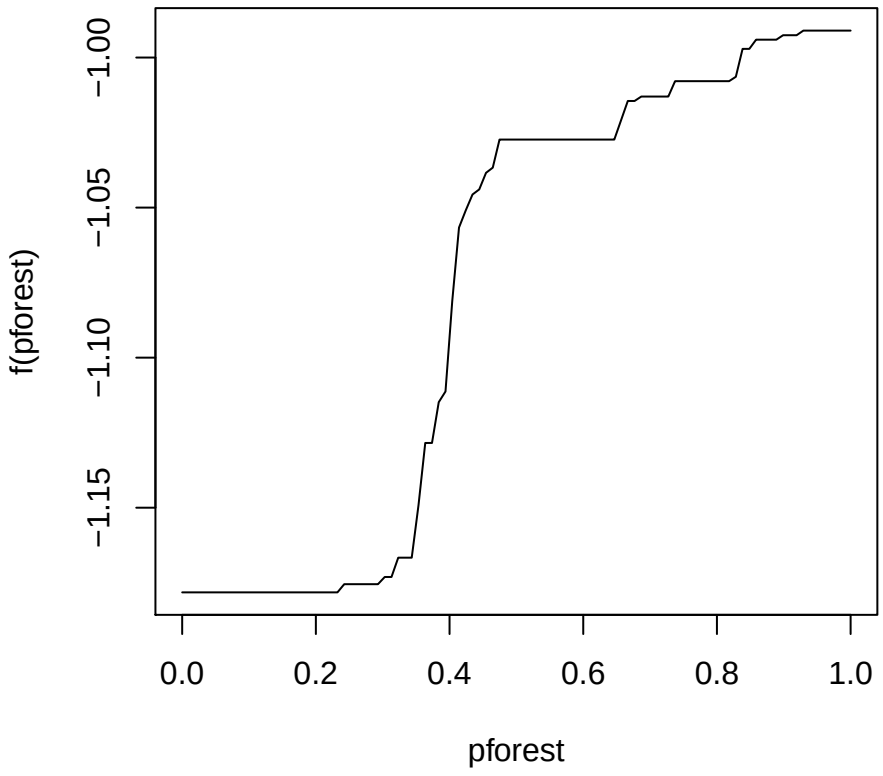

Figure 4.2: Influence plot for the proportion of forest based on the boosted regression tree analysis of the Ovenbird data set using QPAD offsets.

```
(Intercept)  .      .       .      .       .      .
pforest      .      0.2278  0.430  0.6116  0.776  0.9256
xlong        .      .       .      .       .      .

(Intercept) -1.560 -1.629 -1.694 -1.755 -1.813 -1.868
(Intercept)  .      .       .      .       .      .
pforest      1.063  1.188   1.304  1.410   1.509  1.600
xlong        .      .       .      .       .      .

(Intercept) -1.920 -1.968 -2.013 -2.056 -2.096 -2.133
(Intercept)  .      .       .      .       .      .
pforest      1.684  1.761   1.833  1.900   1.961  2.018
xlong        .      .       .      .       .      .

(Intercept) -2.168 -2.200 -2.230 -2.258 -2.284 -2.308
(Intercept)  .      .       .      .       .      .
pforest      2.070  2.119   2.163  2.204   2.242  2.277
xlong        .      .       .      .       .      .

(Intercept) -2.329090 -2.34873 -2.36684 -2.38351 -2.39885
(Intercept)  .         .        .        .        .
pforest      2.308253  2.33641  2.36219  2.38579  2.40738
xlong       -0.004871 -0.01096 -0.01647 -0.02147 -0.02601

(Intercept) -2.41296 -2.42592 -2.43782 -2.44867 -2.45869
(Intercept)  .        .        .        .        .
pforest      2.42714  2.44521  2.46174  2.47676  2.49057
xlong       -0.03013 -0.03387 -0.03726 -0.04035 -0.04315

(Intercept) -2.4679 -2.47627 -2.48397 -2.49102 -2.49747
(Intercept)  .       .        .        .        .
pforest      2.5032  2.51473  2.52527  2.53489  2.54368
xlong       -0.0457 -0.04801 -0.05012 -0.05204 -0.05378

(Intercept) -2.50336 -2.50876 -2.51362 -2.51812 -2.5222
(Intercept)  .        .        .        .        .
pforest      2.55170  2.55902  2.56561  2.57171  2.5773
xlong       -0.05536 -0.05681 -0.05812 -0.05932 -0.0604

(Intercept) -2.52601 -2.5294 -2.53258 -2.53545 -2.53806
(Intercept)  .        .       .        .        .
pforest      2.58237  2.5870  2.59124  2.59510  2.59862
xlong       -0.06139 -0.0623 -0.06312 -0.06386 -0.06454

(Intercept) -2.54037 -2.54255 -2.54454 -2.54636 -2.54802
(Intercept)  .        .        .        .        .
pforest      2.60173  2.60466  2.60734  2.60978  2.61200
xlong       -0.06516 -0.06573 -0.06624 -0.06671 -0.06714

(Intercept) -2.54953
(Intercept)  .
pforest      2.61403
```

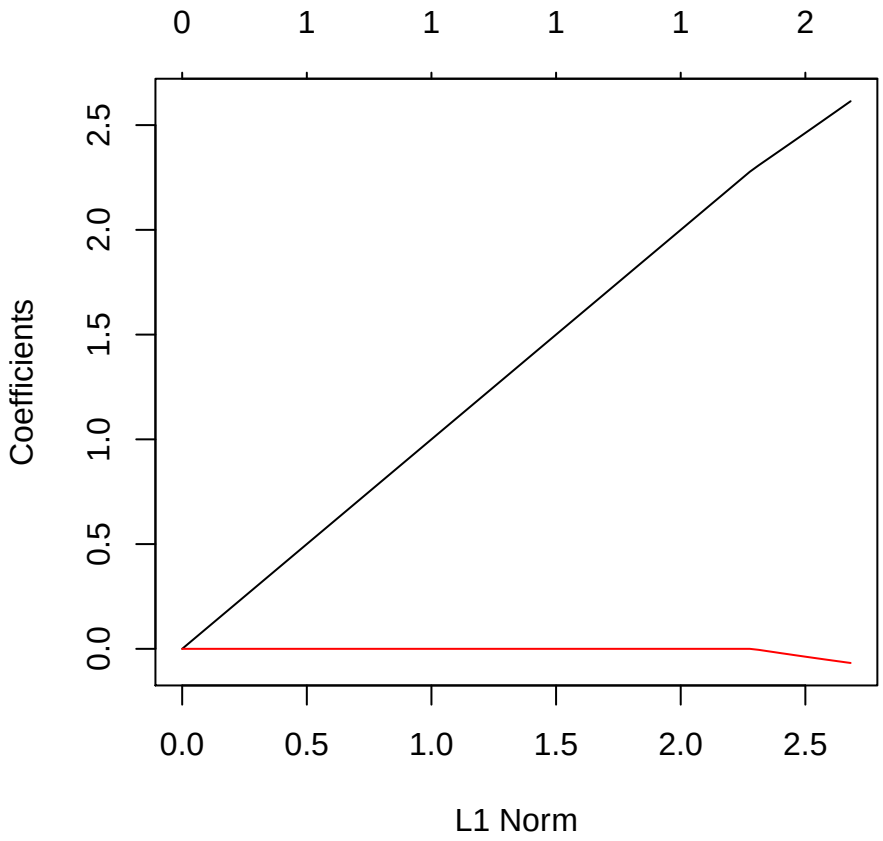

Figure 4.3: Coefficent profile from regularized Poisson GLM analysis of the Ovenbird data set based on QPAD offsets.

```
xlong       -0.06753

plot(enet)
```

4.3 Inference: parameter uncertainty

4.3.1 Non-parametric bootstrap

Here is a simple implementation of the non-parametric bootstrap for Poisson GLM using fixed offset. The `hbootindex` function creates indices for the bootstrap iterations by taking into account the grouped nature of the data set. It resamples routes first, then resamples stops within the resampled routes.

```
set.seed(1234)
Bi <- hbootindex(oven$route, B = 199)
```

```
Loading required package:  pbapply
```

```
modB <- t(sapply(1:length(Bi), function(i) {
    bc <- with(oven, localBAMcorrections("OVEN", t = dur, r = dist,
        jday = JDAY, tssr = TSSR, tree = pforest, lcc = LCC,
        model.sra = bm$sra, model.edr = bm$edr))
    coef(glm(count ~ pforest + xlong, oven[Bi[[i]], ], family = "poisson",
        offset = corrections2offset(bc)[Bi[[i]]]))
}))
```

See how standard errors compare between Wald-type and bootstrap approach:

```
round(cbind(wald_fixed = sqrt(diag(vcov(mod))), boot_fixed = sqrt(diag(cov(modB)))),
    3)
```

```
            wald_fixed boot_fixed
(Intercept)      0.117      0.254
pforest          0.149      0.313
xlong            0.040      0.084
```

To incorporate uncertainty w.r.t. singing rate and distance sampling parameter estimates, one can use nonparametric bootstrap to generate offsets. These offsets will represent the uncertainty in singing rate and distance sampling parameter estimates, so the error can be propagated through the GLM.

For this, we can use the `boot` argument in the functions `globalBAMcorrections` and `localBAMcorrections` making the call `B` times:

```
modB1 <- t(sapply(1:length(Bi), function(i) {
    bc <- with(oven, localBAMcorrections("OVEN", t = dur, r = dist,
        jday = JDAY, tssr = TSSR, tree = pforest, lcc = LCC,
        model.sra = 5, model.edr = 1, boot = TRUE))
    coef(glm(count ~ pforest + xlong, oven[Bi[[i]], ], family = "poisson",
        offset = corrections2offset(bc)[Bi[[i]]]))
}))
```

We can compare standard errors again:

```
round(cbind(wald_fixed = sqrt(diag(vcov(mod))), boot_fixed = sqrt(diag(cov(modB))),
    boot_boot = sqrt(diag(cov(modB1)))), 3)
```

```
            wald_fixed boot_fixed boot_boot
(Intercept)      0.117      0.254     0.257
pforest          0.149      0.313     0.315
xlong            0.040      0.084     0.084
```

The error in offsets can be propagated through other related count models similarly.

It is also possible to take into account model selection uncertainty in the QPAD estimates by randomly choosing among the possible singing rate and distance models based on model weights (with and without the parametric bootstrap procedure in the offsets switched by the `boot` argument):

```
modB2 <- t(sapply(1:length(Bi), function(i) {
    mm <- bestmodelBAMspecies("OVEN", type = "multi")
    bc <- with(oven, localBAMcorrections("OVEN", t = dur, r = dist,
        jday = JDAY, tssr = TSSR, tree = pforest, lcc = LCC,
        model.sra = mm$sra, model.edr = mm$edr, boot = FALSE))
    coef(glm(count ~ pforest + xlong, oven[Bi[[i]], ], family = "poisson",
        offset = corrections2offset(bc)[Bi[[i]]]))
}))
modB3 <- t(sapply(1:length(Bi), function(i) {
    mm <- bestmodelBAMspecies("OVEN", type = "multi")
    bc <- with(oven, localBAMcorrections("OVEN", t = dur, r = dist,
        jday = JDAY, tssr = TSSR, tree = pforest, lcc = LCC,
        model.sra = mm$sra, model.edr = mm$edr, boot = TRUE))
    coef(glm(count ~ pforest + xlong, oven[Bi[[i]], ], family = "poisson",
        offset = corrections2offset(bc)[Bi[[i]]]))
}))
round(cbind(wald_fixed = sqrt(diag(vcov(mod))), boot_fixed = sqrt(diag(cov(modB))),
    boot_boot = sqrt(diag(cov(modB1))), boot_multi = sqrt(diag(cov(modB2))),
    boot_bmulti = sqrt(diag(cov(modB2)))), 3)

            wald_fixed boot_fixed boot_boot boot_multi
(Intercept)      0.117      0.254     0.257      0.254
pforest          0.149      0.313     0.315      0.313
xlong            0.040      0.084     0.084      0.084
            boot_bmulti
(Intercept)       0.254
pforest           0.313
xlong             0.084
```

It is clear that the uncertainty in this case is driven by parameter uncertainty, and not model uncertainty w.r.t. the offsets used.

4.3.2 Bayesian and frequentist approach to hierarchical modeling

In the next example we consider a Poisson-Lognormal generalized linear mixed model with a random intercept for routes, which also incorporates detectability related uncertainty.

This model uses a global maximization technique called data cloning (Lele et al., 2007, 2010), which takes advantage of Bayesian MCMC techniques for maximum likelihood estimation. The software implementation is described in Sólymos (2010). Note that using a single clone is identical to the Bayesian hierarchical modeling.

```
library(dcmle)  # load dcmle package

Loading required package:  dclone
Loading required package:  R2WinBUGS
Loading required package:  parallel
dclone 1.8-1 2012-09-03
Loading required package:  rjags
Linked to JAGS 3.3.0
Loaded modules:  basemod,bugs
dcmle 0.2-2 2012-07-06
```

```
Attaching package:  'dcmle'
The following objects are masked from 'package:coda':
       chanames, crosscorr.plot, gelman.diag,
geweke.diag, heidel.diag, raftery.diag, varnames

load.module("glm")  # load glm module for JAGS

module glm loaded

model <- function() {
    for (i in 1:n) {
        Y[i] ~ dpois(lam[i])
        log(lam[i]) <- inprod(X[i, ], beta) + E[gr[i]] + log(A[i] *
            p[i])
        p[i] <- 1 - exp(-3 * phi[i])
        A[i] <- 3.141593 * tau[i]^2
        log(phi[i]) <- inprod(Z1[i, ], theta01)
        log(tau[i]) <- inprod(Z2[i, ], theta02)
    }
    for (j in 1:m) {
        E[j] ~ dnorm(0, 1/exp(log.sigma)^2)
    }
    for (k in 1:np) {
        beta[k] ~ dnorm(pr[k], 1)
    }
    log.sigma ~ dnorm(-2, 0.01)
    theta01 ~ dmnorm(theta1, Sigma1)
    theta02 ~ dmnorm(theta2, Sigma2)
}
dat <- list(Y = oven$count, X = model.matrix(~pforest + xlong,
    oven), np = 3, n = nrow(oven), m = length(unique(oven$route)),
    gr = dciid(as.integer(as.factor(oven$route))), pr = coef(mod),
    theta1 = coefBAMspecies("OVEN", bm$sra, bm$edr)$sra, Z1 = model.matrix(~JDAY,
        oven), Sigma1 = solve(vcovBAMspecies("OVEN", bm$sra,
        bm$edr)$sra), theta2 = coefBAMspecies("OVEN", bm$sra,
        bm$edr)$edr, Z2 = model.matrix(~LCC, oven), Sigma2 = solve(vcovBAMspecies("OVEN",
        bm$sra, bm$edr)$edr))
dcf <- makeDcFit(model = model, data = dat, params = c("beta",
    "log.sigma"), multiply = c("n", "m"), unchanged = c("np",
    "pr", "theta1", "theta2", "Sigma1", "Sigma2"))
cl <- makePSOCKcluster(3)  # parallel computing for speed up
K <- c(1, 2)  # sequence for the number of clones to use
parLoadModule(cl, "glm")  # load glm module for JAGS on workers

[[1]]
NULL

[[2]]
NULL

[[3]]
NULL
```

```
dcm <- dcmle(dcf, n.clones = K, n.iter = 1000, cl = cl, partype = "parchains")


Fitting model with 1 clone


Parallel computation in progress

Loading required package:  snow

Attaching package:  'snow'
The following objects are masked from 'package:parallel':
       clusterApply, clusterApplyLB, clusterCall,
clusterEvalQ, clusterExport, clusterMap,
clusterSplit, makeCluster, parApply, parCapply,
parLapply, parRapply, parSapply, splitIndices,
stopCluster


Fitting model with 2 clones


Parallel computation in progress

Warning:  chains convergence problem, see R.hat values

stopCluster(cl)  # close cluster
dcm

Call:
dcmle(x = dcf, n.clones = K, cl = cl, n.iter = 1000, partype = "parchains")

Coefficients:
  beta[1]   beta[2]   beta[3] log.sigma
 -2.56782   2.57360  -0.07926  -7.54143
```

The model reveals results similar to the GLM example above, Proportion of forest cover around points had a significant positive effect on density, while longitude had a negative non-significant effect. The estimate of the route level random effect was 0.56 (SE 0.1).

4.4 Prediction

Point predictions (unconditional on the observations) and associated prediction intervals can be calculated by the same MCMC technique:

```
pmodel <- function() {
    for (i in 1:n) {
        Y[i] ~ dpois(lam[i])
        log(lam[i]) <- inprod(X[i, ], tmp[1:np]) + E[i]
        E[i] ~ dnorm(0, 1/exp(tmp[np + 1])^2)
    }
    tmp[1:(np + 1)] ~ dmnorm(cf, Sig)
```

```
}
pf <- 10:0/10
ND <- expand.grid(pforest = pf, xlong = c(-1.8, 0, 1.8))
Xnew <- model.matrix(~pforest + xlong, ND)
## grouping is random (so this is at region and not route
## level)
dat1 <- list(X = Xnew, np = 3, n = nrow(Xnew), cf = coef(dcm),
    Sig = solve(vcov(dcm)))
dcf2p1 <- makeDcFit(model = pmodel, data = dat1, params = c("lam"))
pm1 <- dcmle(dcf2p1, n.clones = 1)

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 330

Initializing model

pm1 <- as.matrix(pm1)
ND$mean <- colMeans(pm1)
ND$cl1 <- apply(pm1, 2, quantile, probs = 0.025)
ND$cl2 <- apply(pm1, 2, quantile, probs = 0.975)
## drawing the figure
op <- par(las = 1)
plot(mean ~ pforest, ND, col = as.integer(as.factor(ND$xlong)),
    ylim = c(0, max(ND$cl2)), type = "n", xlab = "Proportion of forest",
    ylab = "Density (males / ha)", axes = FALSE)
polygon(c(pf, rev(pf)), c(ND$cl1[ND$xlong > 0], rev(ND$cl2[ND$xlong <
    0])), border = NA, col = "grey")
polygon(c(pf, rev(pf)), c(ND$mean[ND$xlong > 0], rev(ND$mean[ND$xlong <
    0])), border = NA, col = "black")
lines(pf, ND$mean[ND$xlong == 0], col = "white", lwd = 2)
box(bty = "l")
axis(1, tck = 0.02)
axis(2, tck = 0.02)
par(op)
```

Predicted density in areas with 100% forest cover was 1.05 males / ha ranging from 0.9 to 1.25 depending on longitude. This prediction almost equals the density estimate of 0.99 (95% confidence limits, 0.85-1.12) by Bayne (2000) based on territory mapping in the same region (Fig. 4.4).

Point predictions conditional on the observations and associated prediction intervals given the observations can be calculated in a similar manner (Fig. 4.5).

```
dat2 <- list(Y = dat$Y, X = dat$X, np = 3, n = nrow(dat$X), cf = coef(dcm),
    Sig = solve(vcov(dcm)))
dcf2p2 <- makeDcFit(model = pmodel, data = dat2, params = c("lam"))
pm2 <- dcmle(dcf2p2, n.clones = 1, n.chains = 1)

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
```

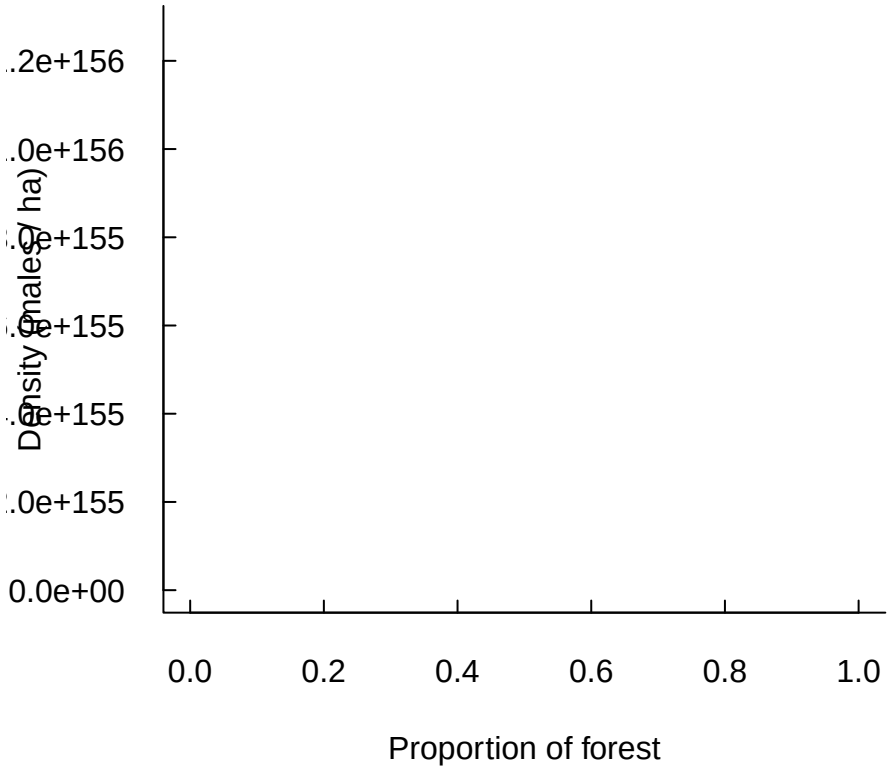

Figure 4.4: Predicted density of Ovenbird as a function of forest cover, based on a generalized linear mixed model (GLMM) fitted to a data set from Saskatchewan, Canada. Grey shade represents 95% prediction intervals incorporating uncertainties related to the random effect and the detectability corrections. Black shade indicates range of variation in mean predictions due to longitude, white line represents the prediction at the centroid of the study area.

```
   Graph Size: 8052

Initializing model

pm2 <- as.matrix(pm2)
pp <- data.frame(mean = colMeans(pm2), cl1 = apply(pm1, 2, quantile,
    probs = 0.025), cl2 = apply(pm1, 2, quantile, probs = 0.975))
summary(pp)

      mean              cl1                cl2
 Min.   :0.0896   Min.   :0.00e+00   Min.   : 1.84e+91
 1st Qu.:0.1207   1st Qu.:0.00e+00   1st Qu.:2.09e+103
 Median :0.2218   Median :0.00e+00   Median :1.09e+113
 Mean   :0.4687   Mean   :2.31e-85   Mean   :3.86e+154
 3rd Qu.:0.7638   3rd Qu.:0.00e+00   3rd Qu.:1.11e+120
 Max.   :1.6202   Max.   :7.63e-84   Max.   :1.27e+156

boxplot(pp$mean ~ dat$Y, xlab = "Observed counts", ylab = "Predicted mean density")
```

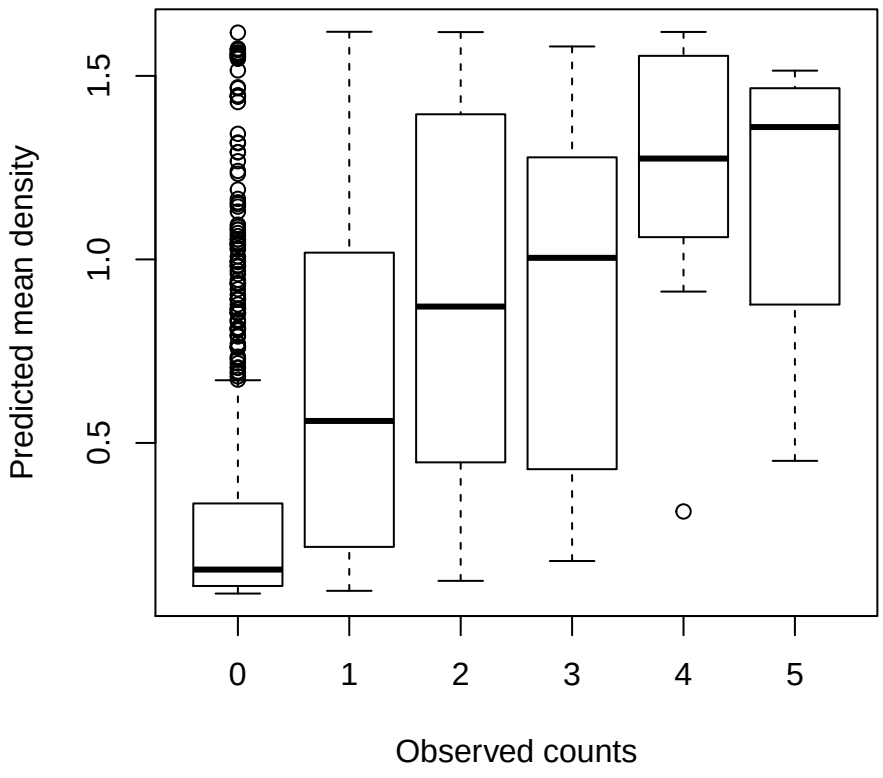

Figure 4.5: The relationship between observed counts and predicted mean density conditional on the observations based on the Ovenbird data set (males / ha).

Chapter 5

Calculating offsets based on independent estimates

Using the estimates derived from our extensive BAM data can help in data deficient situations, but the QPAD approach for using offsets allows the use of custom derived estimates. The only requirement is that estimates need to be consistent with the QPAD approach.

One can create a table with corrections using independently derived point estimates for singing rates (`phi`) and distance parameter (`tau`):

```
customBAMcorrections(r = rep(c(0.5, 1), 2), t = rep(c(5, 10),
    each = 2), phi = seq(0.5, 0.8, len = 4), tau = seq(0.5, 0.8,
    len = 4))

       A      p      q
1 0.7854 0.9179 0.6321
2 3.1416 0.9502 0.3376
3 0.7854 0.9991 0.7833
4 3.1416 0.9997 0.5058
```

Similarly, one can supply a vector of values for `phi` and `tau`:

```
customBAMcorrections(r = rep(c(0.5, 1), 2), t = rep(c(5, 10),
    each = 2), phi = seq(0.5, 0.8, len = 4), tau = seq(0.5, 0.8,
    len = 4))

       A      p      q
1 0.7854 0.9179 0.6321
2 3.1416 0.9502 0.3376
3 0.7854 0.9991 0.7833
4 3.1416 0.9997 0.5058
```

Custom values can be derived from fitted models based on the conditional likelihood estimating procedure:

```
head(customBAMcorrections(r = rep(1, n), t = rep(10, n), phi = fitted(m2),
    tau = fitted(m4)))

      A      p      q
```

```
1 3.142 0.8813 0.4122
2 3.142 0.4927 0.2790
3 3.142 0.8619 0.4030
4 3.142 0.4489 0.2656
5 3.142 0.3200 0.2240
6 3.142 0.9980 0.5465
```

Chapter 6

Further examples

Further examples can be found at the BAM website (http://www.borealbirds.ca) under the Results tab.

These include spatial maps of expected mean abundance for Bird Conservation Regions (BCRs) within provinces and territories of Canada, and relative densities within various land cover classes per spatial units. Information is available for 70 bird species.

For example, the Ovenbird results (density estimates, habitat associations, and more) are available at the BAM website.

Bibliography

D. Bates, M. Maechler, and B. Bolker. *lme4: Linear mixed-effects models using S4 classes*, 2012. URL `http://CRAN.R-project.org/package=lme4`. R package version 0.999999-0.

J. H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2 2010. ISSN 1548-7660. URL `http://www.jstatsoft.org/v33/i01`.

R. J. Hijmans, S. Phillips, J. Leathwick, and J. Elith. *dismo: Species distribution modeling*, 2013. URL `http://CRAN.R-project.org/package=dismo`. R package version 0.8-5.

S. R. Lele, B. Dennis, and F. Lutscher. Data cloning: easy maximum likelihood estimation for complex ecological models using Bayesian Markov chain Monte Carlo methods. *Ecology Letters*, 10:551–563, 2007.

S. R. Lele, K. Nadeem, and B. Schmuland. Estimability and likelihood inference for generalized linear mixed models using data cloning. *Journal of the American Statistical Association*, 105:1617–1625, 2010.

S. R. Lele, M. R. Moreno, and E. Bayne. Dealing with detection error in site occupancy surveys: What can we do with a single survey? *Journal of Plant Ecology*, 5(1):22–31, 2011.

M. Plummer. *JAGS Version 3.3.0 manual (October 2, 2012)*, 2012. URL `http://mcmc-jags.sourceforge.net`.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL `http://www.R-project.org/`. ISBN 3-900051-07-0.

G. Ridgeway. *gbm: Generalized Boosted Regression Models*, 2013. URL `http://CRAN.R-project.org/package=gbm`. R package version 2.0-8.

P. Sólymos. dclone: Data cloning in R. *The R Journal*, 2(2):29–37, 2010. URL `http://journal.r-project.org/`.

P. Sólymos, S. R. Lele, and E. Bayne. Conditional likelihood approach for analyzing single visit abundance survey data in the presence of zero inflation and detection error. *Environmetrics*, 23:197–205, 2012.

P. Sólymos, M. Moreno, and S. R. Lele. *detect: Analyzing Wildlife Data with Detection Error*, 2013. URL `http://dcr.r-forge.r-project.org/,http://www.abmi.ca/`. R package version 0.3-0.

T. Therneau, B. Atkinson, and B. Ripley. *rpart: Recursive Partitioning*, 2012. URL `http://CRAN.R-project.org/package=rpart`. R package version 3.1-55.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL `http://www.stats.ox.ac.uk/pub/MASS4`. ISBN 0-387-95457-0.

A. Zeileis, C. Kleiber, and S. Jackman. Regression models for count data in r. *Journal of Statistical Software*, 27(8):1–25, 7 2008. ISSN 1548-7660. URL `http://www.jstatsoft.org/v27/i08`.