Worked Example for Offshore Redistribution Impact Data

Using the `MRSea` Package

Monique Mackenzie
Eric Rexstad
Lindesay Scott-Hayward
Cornelia Oedekoven

Centre for Research into Ecological and Environmental Modelling University of St. Andrews

Table of contents

Contents

1 Introduction **2**

2 Two-Stage Modelling of Distance Sampling Data **3**

2.1 A brief introduction 3

3 Preamble **3**

4 Data Requirements **4**

4.1 Data requirements: columns 5

4.2 Data Requirements: rows 6

5 Distance Analysis **7**

5.1 Fitting detection functions 7

5.2 Model Selection 8

5.3 Goodness of fit 9

5.4 Adjusting counts for imperfect detection 11

5.5 Creating Count Data from Distance Data 11

6 Introduction to Spatial Modelling **12**

7 Loading the Data **12**

7.1 Checking for Co-linearity 16

8 Fitting a Model **16**

8.1 Fitting a Smooth Term 17

8.2 Checking for Correlation 18

8.3 Model Selection 22

8.4 Checking p-values 29

8.5 Assessing covariate relationships 32

9 Diagnostics **34**

9.1 Cumulative Residuals and runs profiles 37

9.2 COVRATIO and PRESS statistics 39

9.3 Raw Residuals 42

10 Prediction and Inference **44**

10.1 Data requirements for predicting 44

10.2 Making predictions 44

10.3 Visualising the redistribution using predictions 45

11 Bootstrap Confidence Intervals **46**

11.1 Visualising Bootstrap Confidence Intervals 48

12 Significant Differences **49**

12.1 Visualising significant differences 49

13 Comparison to the Truth **51**

13.1 Detection function 51

13.2 Overdispersion and correlation 51

13.3 Type of impact 52

1 Introduction

This document takes you through the process of fitting a detection function model to distance sampling data and then fitting spatial models using the CReSS method in a GEE framework with SALSA for model selection. The last section uses the fitted count model to make predictions across the entire study area - including those areas not surveyed - which we use to draw inference from our study. We use simulated segmented line transect data as our case study. Here, the type of impact was a redistribution of animals within the study area.

2 Two-Stage Modelling of Distance Sampling Data

2.1 A brief introduction

For distance sampling data, we recognise that the number of observed animals along transect lines is a result of two underlying processes:

1. Observation process: not all animals in the covered area are detected
2. Distribution of animals

These two processes are modelled in two stages. In a first stage we fit a detection function to the observed distances - in case of line transect data these are the perpendicular distances from the line to the detection. This detection function enables us to obtain an estimate of the average detection probability in the covered area which is used to adjust observed counts for imperfect detection. We describe how to use the code from this package to fit detection functions in section 5.1, how to select between contending models in section 5.2 and to assess models in section 5.3.
These adjusted counts are then modelled in a second stage count model using covariates that relate to how animals distribute themselves in the study area.
Hence, the interest generally lies in the second item listed here, while the first item involves nuisance parameters that need to be taken into account by estimating the average detection probability in the covered area in a first stage detection model.
We do this for two main reasons:

- The observed counts need to be adjusted for imperfect detection
- To estimate the uncertainty associated with the observation process

The latter requires non-parametric bootstrapping for propagation of uncertainty from the first-state detection model into second stage count model. Non-parametric bootstrapping for distance sampling data is explained in section 11.

3 Preamble

Before we start, we load the `MRSea` package and its dependencies. This may require installing the following packages if they are not already installed on your computer:

- mrds, lawstat, car, mvtnorm, splines, geepack, ggplot2, calibrate, Matrix and fields.

After installing these packages, the following command will load package MRSea and these packages into the active workspace.

```
require(MRSea)
```

This contains the data you will need to follow this example and all of the functions.

To find what data sets are available for testing use the following command:

```
data(package="MRSea")
```

```
Data sets in package MRSea:

dis.data.de                Line transect data with decrease post-impact
dis.data.no                Line transect data with no post-impact consequence
dis.data.re                Line transect data with redistribution post-impact
knotgrid.ns                Knot grid data for nearshore example
knotgrid.off               Knot grid data for offshore example
ns.data.de                 Nearshore data with decrease post-impact
ns.data.no                 Nearshore data with no effect of impact
ns.data.re                 Nearshore data with redistribution post-impact
ns.predict.data.de         Prediction grid data for nearshore post-impact decrease
ns.predict.data.no         Prediction grid data for nearshore no post-impact consequence
ns.predict.data.re         Prediction grid data for nearshore post-impact redistribution
predict.data.de            Prediction grid data for post-impact decrease
predict.data.no            Prediction grid data for no post-impact consequence
predict.data.re            Prediction grid data for post-impact redistribution
```

4 Data Requirements

Distance sampling data generally contains three types of information on the observed animals. Our example consists of segmented line transect data, i.e. line transect data where the lines were divided into small segments. Here the three types of information on the observed animals are:

1. Observed perpendicular distances for each detection
2. Total number of detections for each segment
3. Cluster size for each detection

Hence, we begin with a data frame that contains all the information necessary. In terms of columns we can divide the information into a minimum of three levels, the observation, segment and transect level.

4.1 Data requirements: columns

Observation level

- *object* Object number, no repeats allowed
- *distance* Perpendicular distance in metres for line transects, required if distances were recorded as exact
- *distbegin* Left cutpoint of distance interval in metres, required if data were collected in distance intervals (see function *which.bin* for adding this column)
- *distend* Right cutpoint of distance interval in metres, required if data were collected in distance intervals (see function *which.bin* for adding this column)
- *size* If the data are clustered objects, the data frame should also contain this field that gives the observed number in the cluster
- Optional covariates for detection function modelling

Segment level

- *segment.id* Segment ids, no repeats of the same segment ids for different transects or different visits to the same segments allowed
- *segment.label* Segment label
- *length* Length of segment in km
- Optional covariates for detection function and count modelling

Transect level

- *transect.id* Transect id

- Optional covariates for detection function and count modelling

4.2 Data Requirements: rows

- One record for each detection
- One record for each visit to a segment in case no detections were made

We note that information entered at the respective levels is assumed to be the same for each entry for the respective *object*, *segment.id* or *transect.id*. Information entered at the observation level may vary from one record to the next. Information entered at the segment level needs to be the same for all records with the same *segment.id*. Information entered at the transect level needs to be the same for all records with the same *transect.id*.
To illustrate this, we have a look at our case study. We begin with loading the data.

```
# Loading the data
data(dis.data.re)
dis.data<-dis.data.de
head(dis.data)
```

```
  transect.id transect.label season impact segment.id
1           1              1      1      0          1
2           1              1      1      0          2
3           1              1      1      0          3
4           1              1      1      0          4
5           1              1      1      0          5
6           1              1      1      0          6
  segment.label length  x.pos   y.pos   depth object distance
1           1-1  0.306 656250 6043750 -27.359     NA       NA
2           1-2  0.500 656250 6044250 -27.561     NA       NA
3           1-3  0.500 656250 6044750 -28.608     NA       NA
4           1-4  0.500 656250 6045250 -27.999     NA       NA
5           1-5  0.500 656250 6045750 -27.519     NA       NA
6           1-6  0.500 656250 6046250 -27.223     NA       NA
```

We note that the first six records, contain the information for six different *segment.id*s where no detections were made. Hence, we have one record for each visit to a segment and *NA*'s in the columns pertaining to the observation level.

```
dis.data[214:218,]
```

```
    transect.id transect.label season impact segment.id
214           6              6      1      0        183
```

```
215           6              6      1      0        183
216           6              6      1      0        183
217           6              6      1      0        184
218           6              6      1      0        185
    segment.label length  x.pos   y.pos   depth object  distance
214          6-22    0.5 666250 6046250 -11.302     41  74.13493
215          6-22    0.5 666250 6046250 -11.302     42 154.87536
216          6-22    0.5 666250 6046250 -11.302     43 192.34939
217          6-23    0.5 666250 6046750  -9.283     NA        NA
218          6-24    0.5 666250 6047250  -6.870     NA        NA
```

From these records we can see that for *segment.id* 183, three detections were made, hence we have three records, one for each detection.

5 Distance Analysis

5.1 Fitting detection functions

We begin by fitting a global half-normal detection function to our data. In the next step we will consider other models and compare the relative fit of these models using AIC.

```
# Fitting a half-normal detection function
result <- ddf(dsmodel=~mcds(key="hn", formula=~1),
     data = dis.data, method="ds", meta.data=list(width=250))
```

The ddf function

Here used for fitting a single observer model with a global half-normal detection function and a truncation distance of 250m to our data. A summary of the object created by *ddf* can be obtained using:

```
summary(result)
```

```
Summary for ds object
Number of observations :  2373
Distance range         :  0  -  250
AIC                    :  25446.68

Detection function:
 Half-normal key function

Detection function parameters
Scale Coefficients:
            estimate         se
(Intercept) 4.754715 0.02068034
```

```
                        Estimate           SE         CV
Average p              0.5639511  0.009623902 0.01706513
N in covered region 4207.8115128 91.704527557 0.02179388
```

Conclusion

We conclude that the average detection probability in the covered region was 0.56 with a coefficient of variation of 1.7%.

We can also use the ddf object for plotting our detection function with a histogram of the detections.

```
plot(result,showpoints=F,breaks=seq(0,250,10),
     main="All data combined")
```

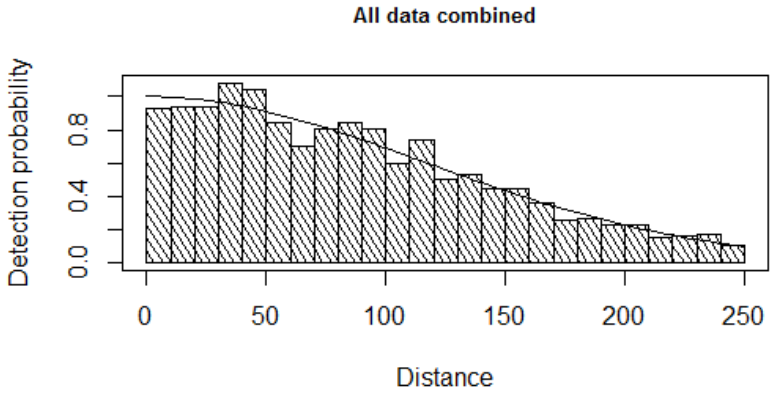


Figure 1: Half-normal detection function

5.2 Model Selection

We are now going to fit a couple more detection models and determine if any of these provide a better relative fit to the data based on minimum AIC.

```
# Using a hazard-rate detection function
result.hr <- ddf(dsmodel=~mcds(key="hr", formula=~1),
     data = dis.data, method="ds", meta.data=list(width=250))
# Using a half-normal function with impact as a covariate
result.imp <- ddf(dsmodel=~mcds(key="hn", formula=~1+impact),
     data = dis.data, method="ds", meta.data=list(width=250))
```

The ddf function

Changing the key function from half-normal to hazard-rate is done using the argument *key*.
Adding covariates to the model is done by altering the argument *formula*.

Using minimum AIC for model selection
We can obtain the AIC value for each detection model using the *summary* function. However, these may also be extracted from the *ddf* object in this manner:

```
# Half-normal model
result$criterion
[1] 25446.68
# Hazard-rate model
result.hr$criterion
[1] 25454.93
# Half-normal model with impact as a covariate
result.imp$criterion
[1] 25448.21
```

Conclusion

The half-normal model has the lowest AIC. Results are slightly ambiguous as the difference in AIC is less than 2 when compared to the *impact*-model.

5.3 Goodness of fit

There are a few goodness of fit tests for detection function models included in the `mrds` package. We show how to use them in this section.

The χ^2 - Test

A chi-sqare test can be done with the *ddf.gof* function. The object this function returns contains various results that can be extracted individually:

```
fit.test <- ddf.gof(result)

# Chi-square statistic, p-value and degrees of freedom
fit.test$chisquare$chi1$chisq
[1] 41.64964
fit.test$chisquare$chi1$p
[1] 0.6931487
fit.test$chisquare$chi1$df
[1] 47
```

Conclusion

The large p-value provides no evidence against the null hypothesis that the model is correct.

QQ-plot

In addition, we may wish to assess the model fit using a QQ-plot. This plot allows identifying potential problems related to our distance data such as rounding of distances to preferred values or overdispersed distance data.

The qqplot.ddf function

This function plots the observed distribution of the distance data vs. the fitted distribution of the distance data.
In addition, it cunducts the Kolmogorov-Smirnov and Cramér-von Mises tests.

```
qq.result <- qqplot.ddf(result)
```

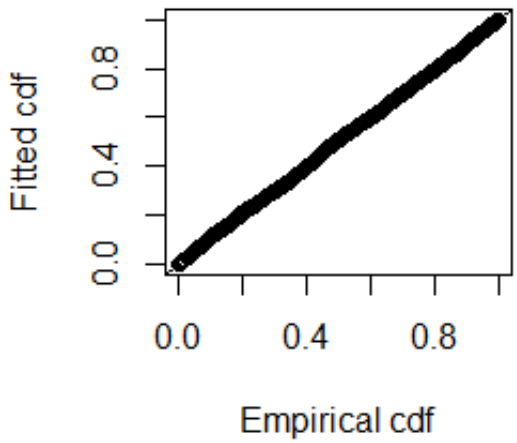


Figure 2: QQ plot for half-normal detection function fitted to our data

Conclusion

The points lie on the diagonal line, hence we conclude that the fit of our half-normal model is adequate.

K-S and C-v-M tests

The Kolmogorov-Smirnov and Cramér-von Mises test results are extracted using:

```
# Kolmogorov-Smirnov statistic and p-value
qq.result$ks$Dn
[1] 0.01503729
qq.result$ks$p
[1] 0.6566394
# Cramér-von Mises statistic and p-value
qq.result$CvM$W
[1] 0.07091705
qq.result$CvM$p
[1] 0.7459479
```

Conclusion

Both p-values are large, hence we conclude that our model is correct.

5.4 Adjusting counts for imperfect detection

In this section, we use the *create.NHAT* function to add two new columns to our distance data: *NHAT* and *area*. *NHAT* is the estimated number of animals (as opposed to the observed number of animals) for each detection and *area* is the size of the covered area for the respective *segment.id*.

```
dis.data <- create.NHAT(dis.data,result)
```

The create.NHAT function

We inflate our number of detected animals by dividing the size of each detection by its probability of being detected which was estimated by the *ddf* function. We also calculate the covered area in km^2 for each *segment.id* by multiplying the length of the segment with the truncation distance and with 2 (for two-sided transects).

5.5 Creating Count Data from Distance Data

In this next step, we create a new data frame in which we sum up the *NHATs* for each *segment.id* and discard the columns from the observation layer.

```
count.data <- create.count.data(dis.data)
```

The create.count.data function

We aggregate the data by *segment.id*. This reduces the number of records in the count data to one for each visit to a segment. The *NHATs* from the distance data are summed up for each segment id.

Next step

This completes the first-stage analysis of distance sampling data. In the next steps we will use our *count.data* to construct spatial models of animal distribution.

6 Introduction to Spatial Modelling

Introduction

Aim

Produce a density surface map along with estimates of uncertainty and identify any significant effect of an impact event (e.g. offshore renewable installation)

The following sections take you through:

1. model fitting
2. checking diagnostics
3. making predictions and inference

Note: This example assumes that any distance sampling analysis has already taken place and we begin with the adjusted counts in each segment.

7 Loading the Data

Loading the Data

Either continue from where you left off in the previous section with `count.data` or, load `count.data` if it was saved as an object.

Requirements:

The coordinates must be labelled `x.pos` and `y.pos`, there must be a column for segment area, labelled `area` and the estimated counts from the detection process must be labelled `NHAT`

```
# make effort column segment length * 0.5 (width of
# transect) - twice truncation distance
attach(count.data)
head(count.data)
```

```
  transect.id transect.label season impact segment.id
1           1              1      1      0          1
2           1              1      1      0          2
3           1              1      1      0          3
4           1              1      1      0          4
5           1              1      1      0          5
6           1              1      1      0          6
  segment.label length  x.pos   y.pos  depth  area NHAT
1           1-1  0.306 656250 6043750 -27.36 0.153    0
2           1-2  0.500 656250 6044250 -27.56 0.250    0
3           1-3  0.500 656250 6044750 -28.61 0.250    0
4           1-4  0.500 656250 6045250 -28.00 0.250    0
5           1-5  0.500 656250 6045750 -27.52 0.250    0
6           1-6  0.500 656250 6046250 -27.22 0.250    0
```

Exploratory Data Analysis

First let's assess the data inputs for the modelling process.

- Check estimates from distance sampling process
- Check for strange covariate values
- Identify possible relationships between covariates and the response (animal counts)

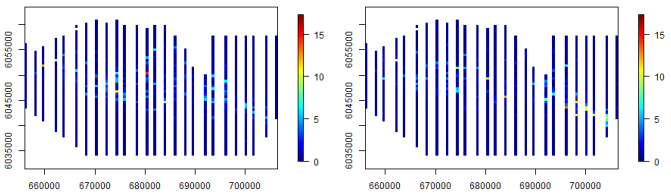


Figure 3: Estimated bird counts before (left) and after (right) an impact event. Each cell is 0.5 km^2 and the colour represents mean animal counts across four seasons.

Assessing the relationships of available covariates and our response (animal counts).

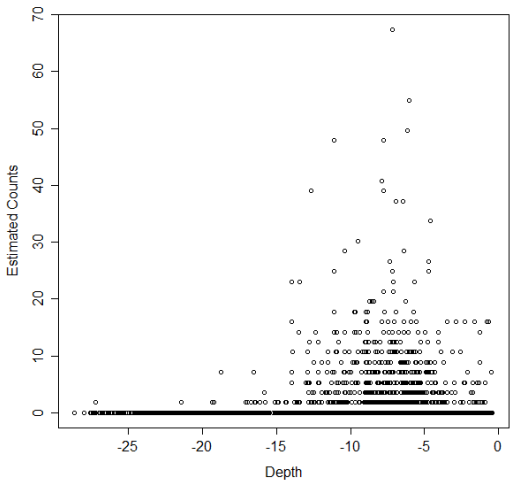


Figure 4: Plot of depth against the estimated bird counts.

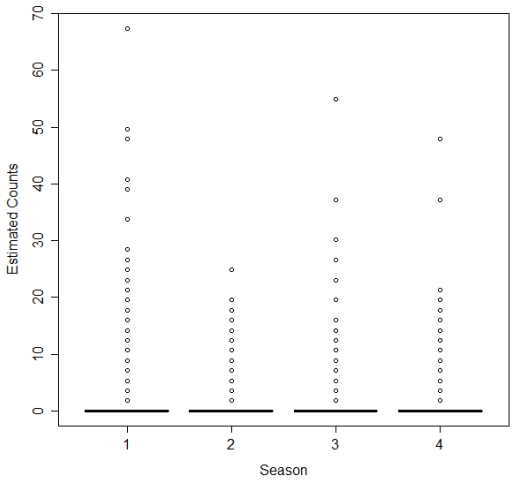


Figure 5: Plot of Season against the estimated bird counts.

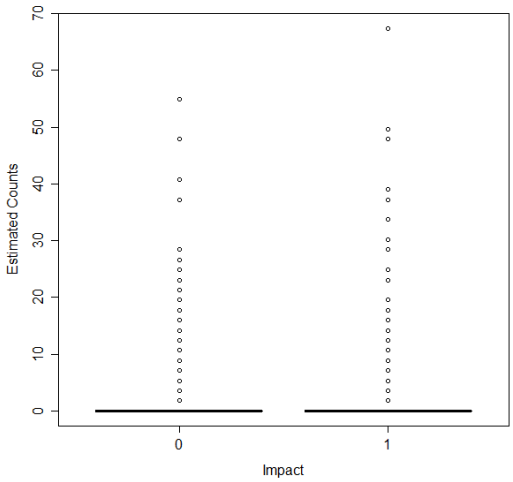


Figure 6: Plot of Impact against the estimated bird counts. Zero is pre impact and one is post impact.

EDA

- Birds were seen predominantly in shallow waters (Figure 4).
- Few birds were seen in waters deeper than 15m.
- Non-linear relationship between depth and bird counts.
- Difficult to identify any relationship between season/impact and bird counts due to the large number of zeros in the data (Figures 5 & 6).

Checking for Collinear variables

Variance Inflation Factors (VIFs)

To assess collinearity between covariates and tell us by how much the standard error is inflated by the other variables in the model.

- Generalised VIFs (GVIFs) are calculated, because the covariates have more than one degree of freedom, and adjusted ($\text{GVIF}_{\text{adj}} = \text{GVIF}^{1/2*Df}$) for the number of degrees of freedom.
- GVIF_{adj} gives us the decrease in precision of estimation due to collinearity (equivalent to $\sqrt{VIF}$).
- For example, a GVIF_{adj} of 2 means that the confidence intervals are twice as wide as they would be for uncorrelated predictors.

7.1 Checking for Co-linearity

Checking for Co-linearity

```
fullModel.linear <- glm(NHAT ~ as.factor(season) + as.factor(impact) +
    depth + x.pos + y.pos, family = poisson, data =count.data)
vif(fullModel.linear)
```

```
                   GVIF Df GVIF^(1/(2*Df))
as.factor(season) 1.000  3           1.000
as.factor(impact) 1.000  1           1.000
depth             4.745  1           2.178
x.pos             1.494  1           1.222
y.pos             5.309  1           2.304
```

```
[1] "Maximum VIF is: 2.3"
```

The maximum value for `depth` and `y.pos` suggests the confidence intervals are twice as wide as they should be for these covariates. One could be removed, however, `y.pos` will not enter our model linearly. Suggest check again when spatial smooth fitted

Checking factor covariates

Each level of a factor based covariate must have some non-zero entries, otherwise the will be problems in the model fitting process

```
# check all factor levels have counts
checkfactorlevelcounts(factorlist=c("season", "impact"), count.data,
                       count.data$NHAT)
```

```
[1] "season will be fitted as a factor variable;
there are non-zero counts for all levels"
[1] "impact will be fitted as a factor variable;
there are non-zero counts for all levels"
```

Conclusion: season and impact are fine for use in the model.

8 Fitting a Model

Here we are going to fit a GEE-CReSS model with SALSA for knot selection. Recap:

Generalised Estimating Equations (GEE)

Framework to allow for correlated errors.

Complex Region Spatial Smoother (CReSS)

Flexible spatial smoothing method.

Spatially Adaptive Local Smoothing Algorithm (SALSA)

Automated knot selection procedure for both one-dimensional (e.g. depth) and two-dimensional (e.g. geographic space) covariates. The knots are sources of flexibility in the surface that can raise or lower the surface.

8.1 Fitting a Smooth Term

A smooth of depth

First we set up an object (`splineParams`) that contains the information required by SALSA for adaptive knot placement.

- Each covariate that might be considered smooth is a list entry in `splineParams`
- The list contains the covariate name, data, initial knot location (one knot at the mean), the boundary knots (greatest range of prediction data and `count.data`) and the degree of the smooth.
- Note: The smooth one dimensional covariates appear in the `splineParams` object starting at slot `[[2]]`. Slot `[[1]]` is reserved for the spatial term.

```
# load prediction data
data(predict.data.re)
predictData<-predict.data.re

# make splineParams object
splineParams<-makesplineParams(data=count.data, varlist=c('depth'),
         predictionData=predictData)
str(splineParams)

List of 2
 $ : list()
 $ :List of 5
  ..$ covar      : chr "depth"
```

```
  ..$ explanatory: num [1:9232] -27.4 -27.6 -28.6 -28 -27.5 ...
  ..$ knots      : num -12.4
  ..$ bd         : num [1:2] -28.6 -0.2
  ..$ degree     : num 2
```

```
fullModel <- glm(NHAT ~ as.factor(season) + as.factor(impact) +
    bs(depth, knots = splineParams[[2]]$knots) + x.pos + y.pos, family =
quasipoisson,  data = count.data)
```

8.2 Checking for Correlation

Checking for Correlation

We fit a model containing all covariates of interest (`fullModel` above) and carry out some **runs tests** to check for **correlated residuals** (the model assumption is uncorrelated residuals).

Runs Test

This is a test for randomness and allows us to determine if we have correlation in our model residuals. We will see a large p-value (H_0: uncorrelated residuals) and good mixing of the profile plot if there is no correlation.

The `runs.test` function can be found in the `lawstat` library

```
runs.test(residuals(fullModel, type = "pearson"),
alternative = c("two.sided"))
```

```
Runs Test - Two sided
data:  residuals(fullModel, type = "pearson")
Standardized Runs Statistic = -67.28, p-value <
2.2e-16
```

- The small p-value ($p << 0.05$) indicates that there is an issue with correlation in the residuals
- The test statistic is negative, which indicates that there are fewer runs of residuals than would be expected if there were un-correlated residuals. We have positive correlation.

- This result is also shown on the runs profile plot (Figure 7).

Plotting runs profile

```
plotRunsProfile(fullModel,  varlist = c("depth"))

[1] "Calculating runs test and plotting profile"
```

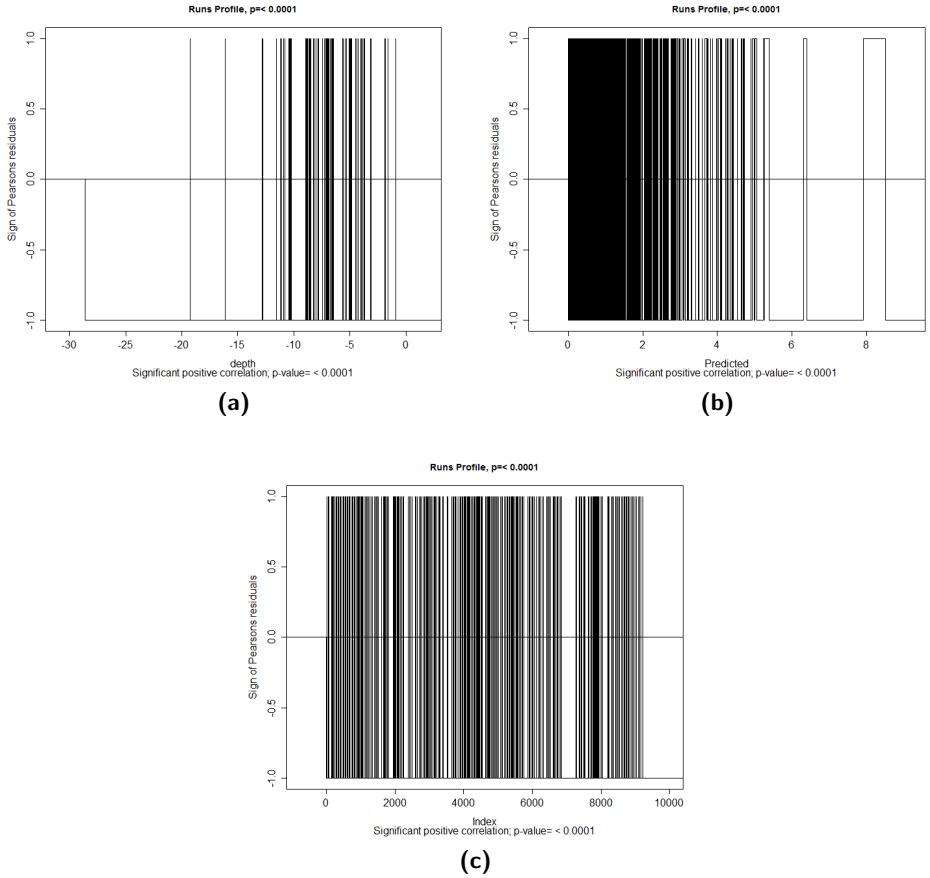


Figure 7: Runs profiles for residuals ordered by (a) depth, (b) predicted value and (c) temporally (by observation index). The p-values and text presented on each plot indicate if there is correlation present in the residuals. The lines are the strings of sequences of positive and negative residuals. A vertical line is the switch between a positive and negative run (or vice versa).

Checking for Correlation

- Runs test and plots show an issue with correlated residuals
- We have positive correlation in the residuals no matter how they are ordered.
- Conclusion:
  - We have correlation that must be accounted for

Choosing a blocking structure to model correlation.

- Correlation within blocks should decline to approximately zero
- Between blocks residuals should be uncorrelated
- Blocks are usually determined by the sampling design
- Here we use the unique transect identifier: Each transect is independent but residuals may be correlated within a transect.
- There are 26 transects repeated 8 times (4 seasons before and 4 after the impact event) giving us 208 blocks.

Autocorrelation function plot

We can check the correlation declines within our blocks by plotting the autocorrelation of the model residuals by block (Figure 8). First we must make a column in our data (if it does not already exist) that represents the blocking structure we wish to use.

```
count.data$blockid <- paste(data$transect.id, data$season, data$impact,
    sep = "")
runACF(count.data$blockid, fullModel, store = F)
```

- Some blocks have little correlation, whilst others have high correlation (0.25) at a lag of 10.
- Correlation in all blocks declines to approximately zero, which we want to see if our blocking is appropriate
- If the correlation in *all* blocks declined very quickly to zero (after one lag) we need not model the correlation
- Conclusion: Our blocking structure is suitable

Autocorrelation function plot

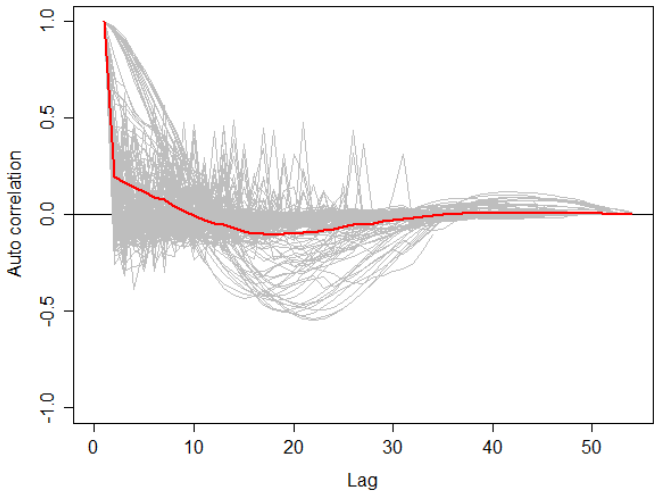


Figure 8: Plot of the correlation in residuals for each block (grey lines). The mean correlation at each lag is indicated in red.

8.3 Model Selection

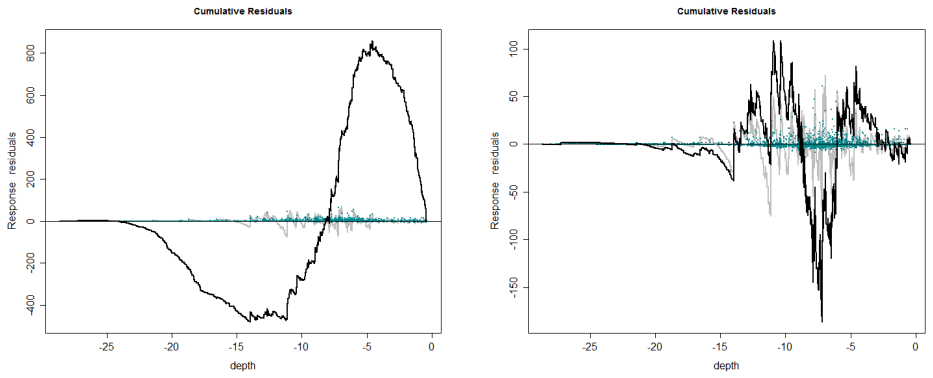
First we select what one-dimensional covariates we want in our model and use SALSA to determine the knot locations of those that are continuous - `depth`.

We can use cumulative residual plots to check for **appropriately modelled covariates**.

- These plots show systematic over- or under- prediction.
- We expect to see good mixing (lots of peaks and troughs) and deviation from this may indicate the covariate is not modelled appropriately.
- Figure 9 shows cumulative residual plots from a model with depth modelled as a linear term and with one knot at the mean.

Plotting Cumulative Residuals

```
# plotting cumulative residuals for the model with depth as a smooth term
plotCumRes(fullModel, varlist= c("depth"), splineParams)


"Calculating cumulative residuals"
```



(a) **(b)**

Figure 9: Cumulative residual plots residuals ordered by depth. (a) depth modelled as a linear term and (b) depth modelled with one knot at the mean depth. The blue points are the residual values, the black line represents the cumulative residuals. The grey line in the background is what we would expect the cumulative residuals to be if depth was modelled correctly.

Cumulative Residuals

- Depth as a linear term is not appropriate (Figure 9a).
- The black line (our model) does not correspond well with the expected line (grey) and shows systematic over prediction at deeper depths and under prediction in shallower waters.
- Depth as a smooth term with one knot at the mean is much better but there is still some evidence of systematic over/under prediction (Figure 9b).
- Conclusion:
  - we might want to consider more flexibility for `depth` by using SALSA.

Setting up the model for SALSA

- There must be a column called `response` in the data, which is the response variable used in the initial model to be fitted.
- The object `salsa1dlist` contains parameters for the `runSALSA1D` function.
  - `fitnessMeasure`. The criterion for selecting the 'best' model. Available options: AIC, AIC_c, BIC, QIC_b.
  - `minKnots_1d`. Minimum number of knots to be tried.
  - `maxKnots_1d`. Maximum number of knots to be tried.
  - `startKnots_1d`. Starting number of knots (spaced at quantiles of the data).
  - `degree`. The degree of the B-spline. Does not need to be specified if `splineParams` is a parameter in `runSALSA1D`.
  - `maxIterations`. The exchange/improve steps will terminate after maxIterations if still running.
  - `gaps`. The minimum gap between knots (in unit of measurement of explanatory).
- The initial model contains all the factor level covariates and any covariates of interest that are not specified in the `varlist` argument of `runSALSA1D`.

```
# info for SALSA
data$response <- data$NHAT

salsa1dlist <- list(fitnessMeasure = "QICb", minKnots_1d = 2,
                    maxKnots_1d = 20, startKnots_1d = 2, degree = 2,
                    maxIterations = 10, gaps = c(1))

# set initial model without the spline terms in there (so
# all other non-spline terms)
initialModel <- glm(response ~ as.factor(season) + as.factor(impact) +
    offset(log(area)), family = "quasipoisson", data = count.data)
```

```
# run SALSA
salsa1dOutput <- runSALSA1D(initialModel, salsa1dlist, varlist=c("depth"),
    factorlist=c("Season", "Impact"), predictData, splineParams=splineParams)
```

Let's look at the structure of the output:

- `bestModel`. The model object for the best fitted model.
- `modelFits1D`. Each slot in the list shows the term fitted, the fit statistic resulting from that term, the knots used and finally the overall formula. If `varlist` is more than one covariate, this output shows how each covariate was retained and what knots were finalised.
- `splineParams`. The spline parameter object is updated with the new knot numbers and locations of the covariates from `varlist`.
- `fitStat`. The fit statistic of the best model.

```
str(salsa1dOutput, max.level = 1)


List of 4
 $ bestModel   :List of 30
  ..- attr(*, "class")= chr [1:2] "glm" "lm"
 $ modelFits1D :List of 2
 $ splineParams:List of 2
 $ fitStat     : num 8464
```

```
# knots chosen for depth
salsa1dOutput$splineParams[[2]]$knots


[1] -17.184  -7.457
```

- Initial model - one knot at the mean depth (-12.4m)
- SALSA result - two knots placed either side of the mean value
- The result of SALSA is additional flexibility in the relationship between bird counts and depth.

Spatial component

- Next we add a two dimensional CReSS smooth of geographic coordinates (s(`x.pos`, `y.pos`))
- To test for a redistribution of animals with an impact effect we fit an interaction term between the smooth of coordinates and `impact`.
- SALSA is used to determine spatially adaptive knot locations for this smooth term.

SALSA 2D requirements

- A grid of knot locations
- Matrix of knot to knot distances
- Matrix of data to knot distances
- Vector of range parameters for CReSS (determines the range of effectiveness of each knot basis). See (**?**) for more details.
- Minimum, maximum and starting number of knots

The next section assumes that the knot grid has been defined. See section **??** for some guidance to do this.

Spatial component

```
# load pre-made knotgrid (regular grid containing NA's for invalid knot
# locations (e.g. on land or outside study region))
data(knotgrid.off)
knotgrid <- knotgrid.off
```

Figure 10 shows the locations of the data points and the locations of the knot points in `knotgrid`. The transects in this data are spaced 2km apart. If knots were chosen at the same y-location on adjacent transects there is no data between them for support. Therefore, we use a gap parameter of 4000m (`x.pos` and `y.pos` are in meters) to prevent adjacent knots.

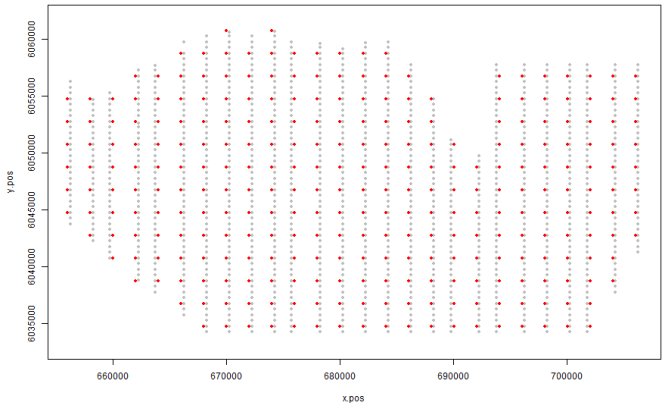


Figure 10: Data and knot locations. The data is in grey and the knot locations in red. Units are in meters.

Next make the distance matrices that give the distance between the data and the knots in one matrix and the second matrix is the knot to knot distances. The function `makeDists` calculates Euclidean distance, however, the `runSALSA2D` function may also take geodesic distance matrices (as the fish swims rather than as the crow flies). See **?** for more details on geodesic distances.

```
# make distance matrices for datatoknots and knottoknots
distMats <- makeDists(cbind(count.data$x.pos, count.data$y.pos),
                      na.omit(knotgrid))
str(distMats)


List of 2
 $ dataDist: num [1:9232, 1:290] 14820 15129 15448 15776 16113 ...
 $ knotDist: num [1:290, 1:290] 0 2000 4000 6000 8000 10000 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:290] "7" "8" "9" "10" ...
  .. ..$ : chr [1:290] "7" "8" "9" "10" ...
```

A sequence of range parameters is required for the CReSS smooth.

- The range parameter determines the influence of each selected knot.
- Small numbers are for a local influence and large ones a global influence.
- Once knot locations are selected (using the mid value in the range sequence), SALSA selects the appropriate range parameter (from the sequence given) for each knot.

```
# choose sequence of radii
r_seq <- getRadiiChoices(numberofradii=8, distMats$dataDist)
```

Setting up the spatial SALSA components

- `fitnessMeasure`. The fitness measures available are the same as for `runSALSA1D`.
- `knotgrid`. $(k \times 2)$ matrix of knot coordinates. Rows of `NA`'s identify illegal knot locations
- `startKnots`. Number of space-filled knots to start with (between minKnots and maxKnots)
- `minKnots`. Minimum number of knots to fit
- `maxKnots`. Maximum number of knots to fit
- `r_seq`. Sequence of range parameters for the CReSS basis.
- `gap`. Minimum gap between knots (in unit of measurement of `x.pos` and `y.pos`)
- `interactionTerm`. Specifies which term in the model the spatial smooth will interact with. If NULL no interaction term is fitted.

```
# make parameter set for running salsa2d
salsa2dlist <- list(fitnessMeasure = "QICb", knotgrid = knotgrid,
    startKnots = 6, minKnots = 4, maxKnots = 20, r_seq = r_seq,
    gap = 4000, interactionTerm = "as.factor(impact)")
```

The initial model is the best model from the one-dimensional SALSA results.

```
# splineParams must be an object in workspace
# update splineParams with the SALSA1D results
splineParams <- salsa1dOutput$splineParams
salsa2dOutput_k6 <- runSALSA2D(salsa1dOutput$bestModel, salsa2dlist,
    distMats$dataDist, distMats$knotDist,
    splineParams = splineParams)
```

Multiple SALSA runs

The example above uses 6 starting knot locations. There is a risk that the SALSA algorithm may get stuck in local minima or maxima and so we recommend that a variety of starting knot numbers are used. Here we try 6, 8, 10 and 12 (Table 2).

We use Cross-Validation (CV) as a method for selecting between models with a variety of starting knots.

k-fold Cross-Validation (CV)

- Method for assessing model fit where the data is split into k sets and the model fitted to $k_{(-i)}$ sets and predicted to the k_i^{th} set.
- Mean Squared Error (MSE) is calculated for each of the k_i prediction sets and a mean taken to get the CV score.
- The smaller the CV score the better the model.

Example:

```
count.data$foldid <- getCVids(count.data, folds = 5, block = "blockid")
```

```
cv1 <- getCV_CReSS(count.data, salsa1dOutput$bestModel,
           salsa1dOutput$splineParams)


[1] 6.087
```

Choosing a model

Table 1: Table of CV scores for a variety of starting knot numbers for the spatial smooth.

Model type	Start knots	End knots	CV
1D terms only	-	-	6.0865
1D/2D terms	6	5	5.6508
1D/2D terms	8	8	5.8286
1D/2D terms	10	10	5.7556
1D/2D terms	12	11	5.7275

The best model chosen using CV score is the model that uses a spatial smooth with 6 spatial knots.

```
# having chosen the 2d interaction model, save the model object
baseModel <- salsa2dOutput_k6$bestModel
# update spline parameter object
splineParams <- salsa2dOutput_k6$splineParams
```

Chosen Model - recheck for collinearity

```
vif(baseModel)
```

```
                                    GVIF Df GVIF^(1/(2*Df))
as.factor(season)                  1.000  3           1.000
as.factor(impact)                  2.016  1           1.420
s(depth)                           5.078  4           1.225
s(x.pos, y.pos)                    163.5  6           1.529
s(x.pos, y.pos):as.factor(impact) 96.45  6           1.4633
```

Maximum adjusted GVIF is less than $\sqrt{5}$ (one of the cut-off's often seen in the literature) so we are happy there is no issue with collinearity.

8.4 Checking p-values

Re-assessing runs test for best model

- Our best model based on CV selection is the model with the interaction term.
- We could also use p-value selection, though the model must be fitted as a GEE to model the correlation.

We must account for the correlation in our residuals, which we found earlier but should also check the current model.

```
runs.test(residuals(baseModel, type = "pearson"))


Runs Test - Two sided
data:  residuals(baseModel, type = "pearson")
Standardized Runs Statistic = -66.57, p-value <2.2e-16
```

GEE framework

There is significant positive correlation ($p << 0.05$ and test statistic is negative) so we re-fit the model as a GEE.

Note: It is possible to do this at this stage as we are modelling correlation using empirical standard errors and not a specific correlation structure.

The current model:

```
glm(formula = response ~ as.factor(season) + as.factor(impact) +
 bs(depth, knots = splineParams[[2]]$knots, degree =
 splineParams[[2]]$degree, Boundary.knots = splineParams[[2]]$bd) +
 LocalRadialFunction(radiusIndices,dists,radii,aR) +
 as.factor(impact):LocalRadialFunction(radiusIndices,dists,radii,aR) +
 offset(log(area)), family = quasipoisson, data = count.data)
```

```
# N.B. for the GEE formula, the data must be ordered by block (which this is)
# and the blockid must be numeric
# specify parameters for local radial:
radiusIndices <- splineParams[[1]]$radiusIndices
dists <- splineParams[[1]]$dist
radii <- splineParams[[1]]$radii
aR <- splineParams[[1]]$invInd[splineParams[[1]]$knotPos]

# update model in workspace with parameters for spatial smooth (above)
baseModel <- update(baseModel, . ~ .)

# Re-fit the chosen model as a GEE (based on SALSA knot placement) and
# GEE p-values
geeModel <- geeglm(formula(baseModel), data = count.data, family = poisson,
                   id = blockid)
```

Checking p-values

```
# table of p-values
# specifying varlist and factorlist makes shorter variable names
getPvalues(geeModel, varlist = c("depth"), factorlist = c("season",
    "impact"))


[1] "Getting marginal p-values"
                Variable p-value
1                 season <0.0001
2                 impact 0.5398
3                  depth <0.0001
4        s(x.pos, y.pos) <0.0001
5 s(x.pos, y.pos):impact 0.0014
```

Do we remove the main `impact` effect?

Either:

- Keep the main effect for `impact` since it is part of the interaction term and is difficult to really interpret what this p-value actually means. or
- Remove the main effect for impact as it is not significant ($p >> 0.05$).

Here we chose the first option, but below is an example of how to remove the `impact` term using the `update` function.

Removing the main `impact` effect

```
# how to remove impact
model <- update(geeModel, . ~ . - as.factor(impact))
# reshow p-values
getPvalues(model, varlist = c("depth"), factorlist = c("season",
    "impact"))


[1] "Getting marginal p-values"
                Variable p-value
1                 season <0.0001
2                  depth <0.0001
3        s(x.pos, y.pos) <0.0001
4 s(x.pos, y.pos):impact <0.0001
```

8.5 Assessing covariate relationships

Partial Plots

Visually examine the one-dimensional covariates in the model (depth, season, impact). Check smooth/linear terms are specified correctly.

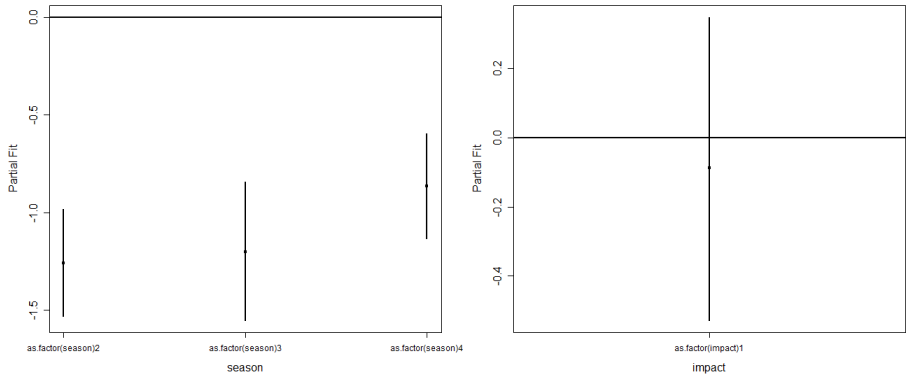
```
runPartialPlots(geeModel, count.data, varlist = c("depth"),
     factorlist =c("season", "impact"))


[1] "Making partial plots"
```

Assessing covariate relationships

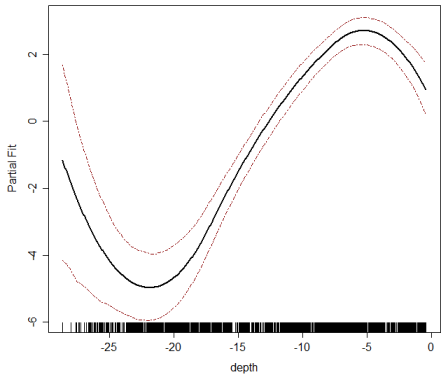
Partial Plots

- Season: predictions for each of the three seasons (2, 3, 4) are, on average, of fewer animals than for the baseline (season 1).
- GEE based confidence intervals show little difference between the three levels and the baseline level
- Impact: confirms the ANOVA p-value and indicates no change in bird numbers post impact.
- Depth: a declining non-linear relationship with increased depth (fewer birds in deeper waters).
- Most birds are estimated in waters 3-8m deep.
- Small confidence interval indicates a precisely estimated relationship.

Note: if the confidence interval for depth was very wide it might indicate that depth as a linear term is more appropriate.



(a) **(b)**



(c)

Figure 11: Partial plots for (a) season, (b) impact and (c) depth.

9 Diagnostics

Diagnostics

We make multiple plots to assess the fit of our model:

- Observed vs Fitted
- Fitted vs residuals
- Cumulative residuals
- Runs sequence
- COVRATIO statistics
- PRESS statistics
- Raw residuals

Observed vs fitted Plot

Observed vs Fitted

Indication of fit to the data. All the values would be on the 45^o line for a perfectly fitting model.

The agreement between the input data and the model undergoing evaluation can be quantified using numerical measures; Marginal R-squared and Concordance Correlation.

These measures are an output on the observed vs fitted plot.

Marginal R-squared value

It is widely used to assess models fitted to both un-correlated and correlated data and is approximately between 0 and 1. Values closer to 1 indicate better fit to the data.

Concordance Correlation

It is guaranteed to return values between zero and one. Values closer to 1 indicate better fit to the data.

```
# create observed vs fitted and fitted vs residual plots
runDiagnostics(geeModel)


[1] "Assessing predictive power"
```

Observed vs fitted

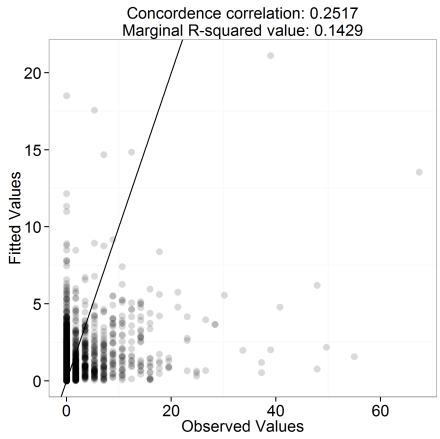


Figure 12: Diagnostic plots of observed vs fitted values, where the diagonal line indicates where data should lie for a perfect fit.

Observed vs fitted

- High observed values are under-predicted
- Observed zeros tend to be over-predicted
- Marginal R-squared and concordance correlation are low
- Conclusion: poor model fit

Fitted values vs scaled Pearsons residuals Plot

Fitted values vs residuals

Indication of correct mean-variance relationship (model assumption). Expect to see no pattern in the plot.

- For a Poisson model the size of residuals is expected to increase with increasing fitted values
- For over-dispersed data the size of residuals increases at a faster rate than the strict Poisson relationship
- To get a plot with an expectation of no pattern we use Pearsons residuals to account for the former and scaled these by the dispersion parameter for the latter
- Thus we plot Fitted values against scaled Pearsons residuals

Fitted values vs scaled Pearsons residuals

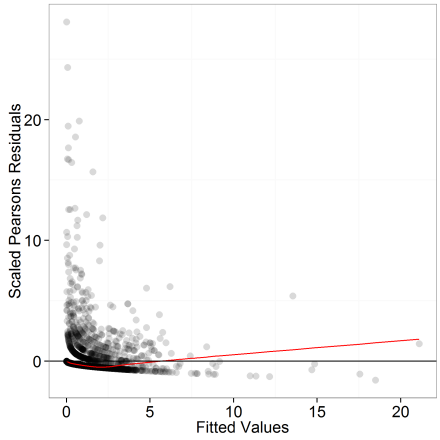


Figure 13: Diagnostic plot of fitted values vs scaled Pearsons residuals, where the red line is a locally weighted least squares regression to indicate pattern in the plot, which might otherwise be hidden due to over-plotting.

- Possible pattern in residuals but hard to tell due to over-plotting
- Locally weighted least squares regression line does not indicate an unusual pattern
- Conclusion: no issue with model assumption (mean-variance relationship)

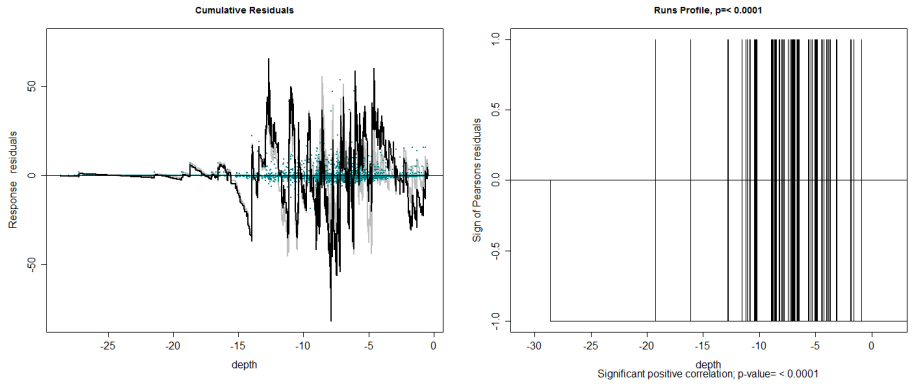
9.1 Cumulative Residuals and runs profiles

Cumulative residuals and runs profiles

- Assessment of systematic over- or under- prediction using cumulative residuals.
- Assessment of the correlated nature of residuals (how random they are) given that the residuals are ordered by covariate value, predicted value or temporally.

```
plotCumRes(geeModel, varlist=c('depth'), splineParams=splineParams, d2k=dists)
plotRunsProfile(geeModel, varlist=c('depth'))
```

Cumulative residuals and runs profiles for depth



(a) **(b)**

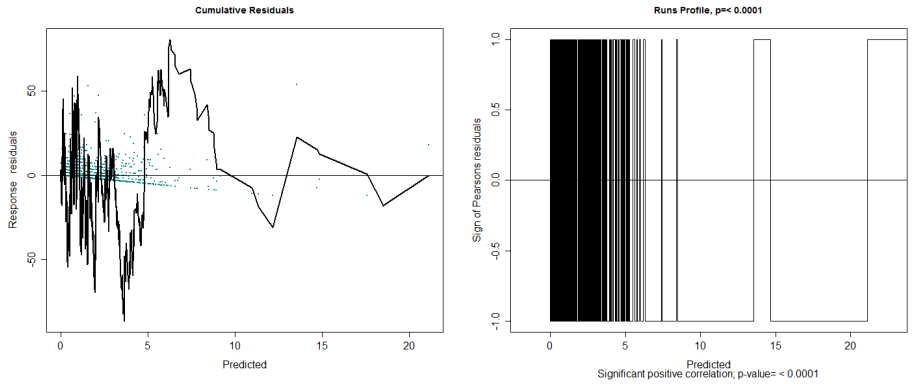
Figure 14: Cumulative residual plot (a) and runs profile (b) for residuals ordered by depth. The blue points are the residual values, the black line represents the cumulative residuals. The grey line in the background is what we would expect the cumulative residuals to be if depth was modelled correctly.

Ordered by **depth**

- No systematic over or under prediction
- Variable modelled appropriately (similar to expected (grey line) and better than Figure 9)

- Fewer runs than would be expected if residuals were random ($p < 0.05$)
- Significant positive correlation between residuals when ordered by depth

Cumulative residuals and runs profiles ordered by predicted value



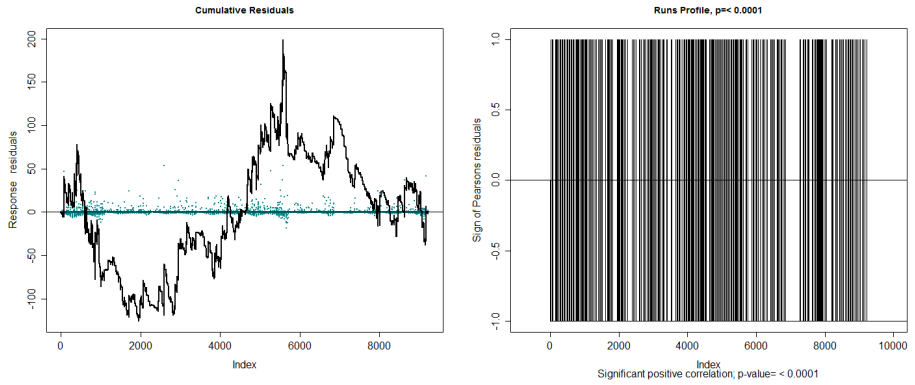
(a) **(b)**

Figure 15: Cumulative residual plot (a) and runs profile (b) for residuals ordered by the predicted value. The blue points are the residual values and the black line represents the cumulative residuals.

Ordered by **prediction** value

- Little systematic over or under prediction at predicted counts < 5
- Good mixing at small predicted values
- But, fewer runs than would be expected if residuals were random ($p < 0.05$)
- Significant positive correlation between residuals when ordered by predicted value

Cumulative residuals and runs profiles ordered temporally



(a) **(b)**

Figure 16: Cumulative residual plot (a) and runs profile (b) for residuals ordered by the index of observations (temporally). The blue points are the residual values and the black line represents the cumulative residuals.

Ordered by **index** (temporally)

- Early observations (before impact) are over predicted and the later (after impact) are under predicted
- Best mixing of residuals seen in the runs profile
- But, still fewer runs than would be expected if residuals were random ($p < 0.05$)
- Significant positive correlation between residuals when ordered temporally

Cumulative residuals and runs profiles

- Despite the inclusion of temporal covariates (season and impact) we still have some unmodelled correlation
- We have modelled this correlation using GEEs so no need for concern.
- p-values from ANOVA (used for covariate selection earlier) are reliable due to modelled correlation

9.2 COVRATIO and PRESS statistics

The PRESS and COVRATIO statistics are relative measures that assess how aspects of the model change when individual blocks are removed from the analysis.

COVRATIO statistic

Signals the change in the **precision of the parameter estimates** when each block is omitted.

- Values greater than 1 signal removing the block inflates model standard errors
- values less than 1 signal standard errors are smaller when that block is excluded

PRESS statistic

Quantifies the sensitivity of **model predictions** to removing each block.

- Relatively large values signal the model is sensitive to these subjects.
- Model coefficients are re-estimated when each block is omitted (one-by one) and the sum of the squared differences between the response data and the predicted values (when that subject is removed) are found.

If model predictions or measures of precision appear particularly sensitive to omitted blocks, examine model conclusions based on models with and without the potentially problematic blocks.

COVRATIO and PRESS statistics

```
timeInfluenceCheck(geeModel, count.data$blockid, dists, splineParams)
```

```
[1] "Calculating the influence measures will take approximately 7 minutes"
```

```
# influence plots (covratio and press statistics)
influence<-runInfluence(geeModel, count.data$blockid, dists, splineParams)
```

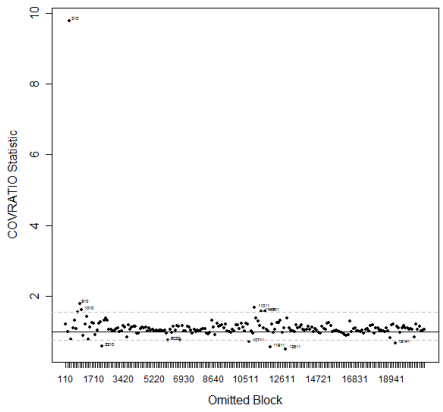


Figure 17: Plot of COVRATIO statistics; the dashed grey lines indicate the lower 2.5% and upper 97.5% quantiles of the statistics.

COVRATIO statistics

- there will always be blocks outside the dashed lines (they are quantiles)
- As it is a relative measure, blocks far away may be of concern
- Blocks with COVRATIO statistic < 1 are of most concern (decrease in precision when removed)
- Here, block 310 is far away but results in an increase in standard errors when removed
- Conclusion: No issue with blocks unduly influencing the precision of parameter estimates

PRESS statistics

- there will always be blocks above the dashed line (it is a quantile)
- As it is a relative measure, blocks far away may be of concern
- Here, block 12611 is far away. This is the right-hand most transect in season 1 before the impact event and contains only one segment with a bird count.

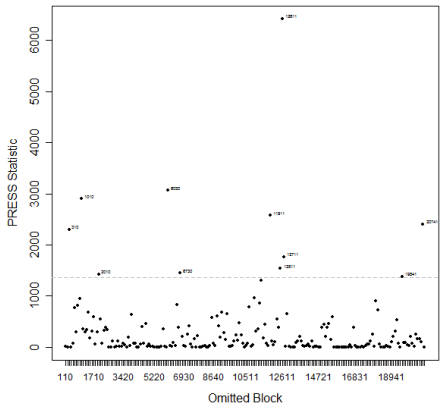


Figure 18: Plot of PRESS statistics; 95% of the statistics fall below the dashed grey lines. Labelled points on both plots are outside the grey dashed line(s) and are labelled with the identifier for the block that has been removed to create the statistic (not an observation number).

- Conclusion: we may choose to fit a model with and without this block to assess the change in predictions

9.3 Raw Residuals

Raw residuals

Raw residuals (fitted - observed values) can be represented spatially to ascertain if some areas are over/under predicted.

- Highest residuals in central area down to right hand side (in cells where birds were detected)
- No systematic over/under prediction

- Conclusion: no spatial issue with raw residuals

Raw residuals

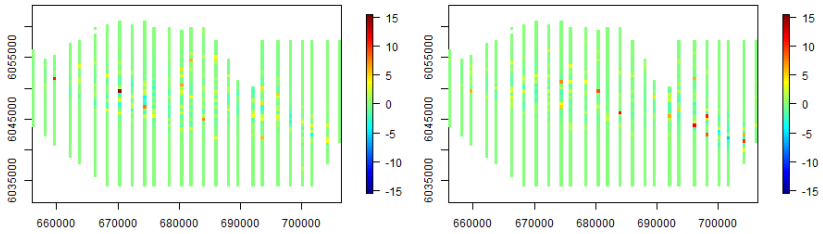


Figure 19: Raw residuals before impact (left) and after impact (right). These residuals are fitted values - observed values (mean* birds/km^2). * mean density because the predictions for each cell are for each season.

```
# residual plot
resids <- fitted(geeModel) - count.data$NHAT
dims <- getPlotdimensions(count.data$x.pos, count.data$y.pos,
    segmentWidth=500, segmentLength=500)

par(mfrow = c(1, 2), mar = c(3, 3, 3, 5))
quilt.plot(count.data$x.pos[count.data$impact == 0], count.data$y.pos[
    count.data$impact == 0], resids[count.data$impact == 0], asp = 1,
    ncol = dims[2], nrow = dims[1], zlim = c(-15.5, 15.5))
quilt.plot(count.data$x.pos[count.data$impact == 1], count.data$y.pos[
    count.data$impact == 1], resids[count.data$impact == 1], asp = 1,
    ncol = dims[2], nrow = dims[1], zlim = c(-15.5, 15.5))
```

How did we do?

Table 2: Table of potential modelling problems, what method was used to assess the problem, was the potential problem an issue in reality and if yes then what was the solution. Solutions in red were not done and are possibilities for the future.

	Method	Issue (Y/N)	Solution
Collinearity	VIF	N	-
Over-dispersion	-	Y	estimated by GEE
Model Fit	Observed vs Fitted	Y	use more covariates
mean-variance relationship	Fitted vs residuals	N	-
Correlated Residuals	runs test	Y	modelled by GEE
Correlated Residuals	Runs profile	Y	modelled by GEE
Covariate specification	cumulative residuals	N	-
systematic over/under prediction	cumulative residuals	N	-
spatial systematic over/under prediction	raw residual plot	N	-
removing blocks	COVRATIO statistics	N	-
removing blocks	PRESS statistics	N	-

10 Prediction and Inference

10.1 Data requirements for predicting

For making predictions, we require a data frame containing grid data with the records we wish to make predictions for. Again, this requires certain columns and records which can be summarised as:

- **Columns:** each covariate retained in the best fitting model with exactly matching column names
- **Columns:** *x.pos* and *y.pos* provide geographic position information used for calculating distance matrices for CReSS/SALSA
- **Columns:** *area* provides the area of each grid cell.
- **Rows:** 1 record for each grid cell and for each date and time a prediction is required.

Using our case study of offshore data, we have a look at our prediction data using the `head` function after we load the data.

```
# If not loaded already, load the prediction grid data
data(predict.data.re)
predictData <- predict.data.re
head(predictData)
```

```
   area  x.pos   y.pos   depth segment.id season impact   truth
1 0.006 655444 6044450 -27.701          1      1      0 0.001429259
2 0.179 655250 6044750 -28.163          2      1      0 0.051732441
3 0.250 655250 6045250 -28.512          3      1      0 0.084503893
4 0.250 655250 6045750 -28.139          4      1      0 0.069776928
5 0.250 655250 6046250 -27.830          5      1      0 0.059117647
6 0.250 655250 6046750 -27.466          6      1      0 0.048559153
```

10.2 Making predictions

We are now going to use these data to make predictions using our best fitting CReSS/Salsa model from the previous sections. This requires three steps:

1. Creating a matrix of distances from prediction data to knots
2. Making predictions to the prediction data using the best fitting model
3. Converting the predictions back to the response scale

```
# create the distance matrix for predictions
dists <- makeDists(cbind(predictData$x.pos, predictData$y.pos),
    na.omit(knotgrid), knotmat = FALSE)$dataDist
# use baseModel to make predictions to avoid a warning from
# using geeModel (same answers though)
predslink <- predict(baseModel, predictData, type = "link")
# reversing the log-link to convert predictions back to the response scale
preds <- exp(predslink)
```

The object `preds` contains the predictions for each cell in our prediction grid data.

10.3 Visualising the redistribution using predictions

We know that our model identified a redistribution of animals within the study area by the significant interaction term between impact and the two-dimensional smooth of *x.pos* and *y.pos*. However, from the model coefficients it is not obvious where the redistribution has occurred. Hence, we are going to investigate this by visualising the redistribution using our predictions.

The following is code for plotting the model predictions in Figure 20.

```
# plotting the predictions for before and after impact
par(mfrow=c(1,2), mar=c(3,3,3,5))
quilt.plot(predictData$x.pos[predictData$impact==0],
predictData$y.pos[predictData$impact==0],
preds[predictData$impact==0], asp=1, nrow=104, ncol=54,
zlim=c(0, maxlim))
quilt.plot(predictData$x.pos[predictData$impact==1],
predictData$y.pos[predictData$impact==1], preds[predictData$impact==1],
asp=1,nrow=104, ncol=54, zlim=c(0, maxlim))
```

Note: the value for `maxlim` can be determined by plotting both plots without the `zlim` argument first from which you can determine what the maximum value in the scale of either plot is.

The `quilt.plot` function

- Plots averages of the predictions for each grid cell
- Subsetting the data using the square brackets restricts the input data to before (left) and after impact (right plot)
- Using the `zlim` argument allows ensuring that left and right plot have the same scale colour scheme

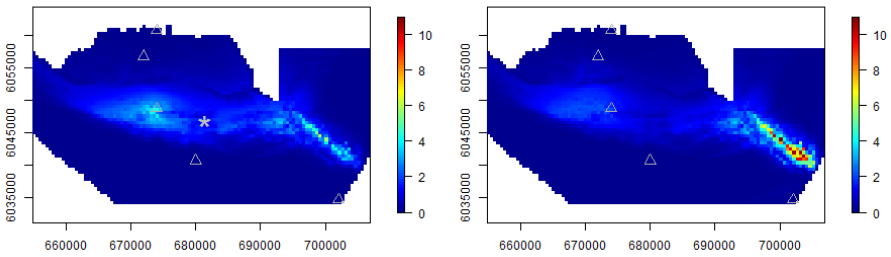


Figure 20: Predictions of bird density (mean birds/km^2) from the fitted model for before (left) and after (right) an impact event. The * indicated the location of the impact; the △ indicate the knot locations.

Conclusion

- Decline in bird density in the central region
- Increase in density in the south east of the study area
- The central region is where a known impact has taken place and the results suggest that birds have moved from this region to the south east after the impact event

Next step

- Is this difference real or simply random noise?
- We need to classify if the differences are significant or due to sampling variation
- We do this by constructing bootstrap confidence intervals for each prediction in the prediction grid data.

11 Bootstrap Confidence Intervals

Following the predictions from section 10.2, we now create a set of bootstrap predictions and use these to construct percentile confidence intervals around each prediction from our prediction grid data. We run B iterations where for each iteration the following steps are completed:

- resample the original distance data with replacement

- re-fit the detection function to the bootstrapped data and re-estimate the *NHATs* (estimated number of animals)
- re-fit the count model to the bootstrapped data
- using the model coefficients and covariance matrix to sample new coefficients from a multivariate Normal distribution
- make predictions to the study area using these coefficients

This approach incorporates the uncertainty associated with both modelling stages, the detection model and the count model.

The `do.bootstrap.cress` function

This function performs the number of bootstrap iterations specified with the argument *B* where the fitting model for the second stage is CReSS.
If a `ddf.obj` is specified, this function automatically applies the 'distance'-approach which includes re-fitting the detection model as described above.
Since the prediction object can be quite large, it is not returned but saved into the current working directory.

```
# do the bootstrap
dis.data$seasonimpact <- paste(dis.data$season, dis.data$impact)
do.bootstrap.cress(dis.data, predictData, ddf.obj, baseModel,
    splineParams, dists, resample = "transect.id",
    rename = "segment.id", stratum = "seasonimpact", B = 250)
```

In the next step we will use these bootstrap predictions to construct our 95% confidence intervals for each prediction.

The `makeBootCIs` function

Using the `B` bootstrap predictions from a call to the `do.bootstrap.cress` function, the `makeBootCIs` function creates lower and upper limits for 95% confidence intervals using the percentile method for each record in the prediction data.

```
# read in bootstrap predictions
load(predictionboot.RData)
cis <- makeBootCIs(bootPreds)
```

11.1 Visualising Bootstrap Confidence Intervals

In this section we will investigate whether looking at the lower and upper confidence limits from the 95% confidence intervals will give us a similar picture as looking at the predictions.

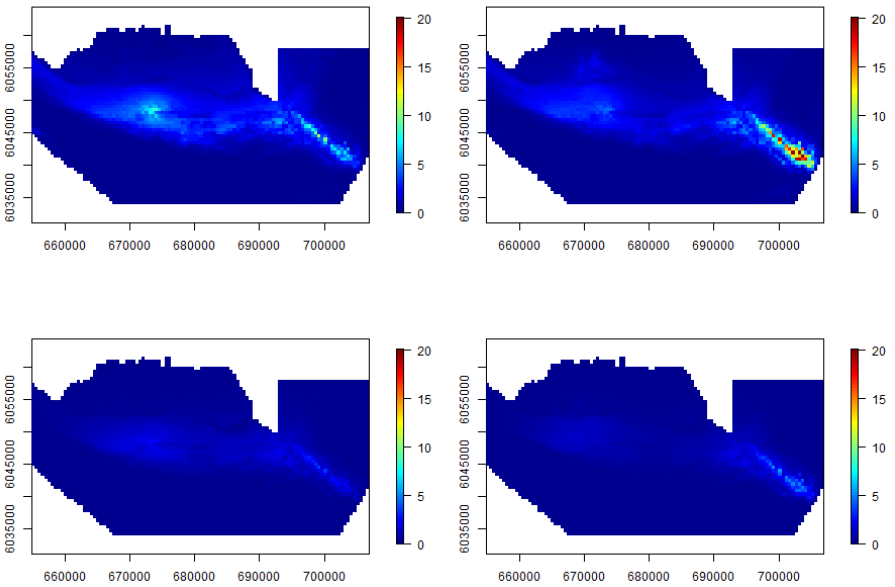


Figure 21: Upper (top) and lower (bottom) 95% confidence intervals of bird density (mean birds/km^2) from the fitted model from before (left) and after (right) impact.

Conclusion

- Lower and upper intervals before impact (left) show a higher density of birds in the central and south east region.
- After impact (right), both lower and upper intervals show a decline in density in the central region and an increase in the area to the south east.
- It is difficult to see where in the study region any significant differences may occur.

Next step

- To see where the redistribution may have occurred we proceed to the next step where we calculate the differences in densities before and after impact for each corresponding pair of predictions.

12 Significant Differences

In this section we will estimate the differences in predictions for each corresponding pair of records from our prediction grid data. The first half of our prediction grid data consists of records from before impact while the second half consists of records from after impact.

Note that the before and after data need to be ordered in the same manner and of the same length.

The `getDifferences` function

For each bootstrap iteration the differences in predictions from before and after impact are calculated (Density$_{after}$ - Density$_{before}$) and 95% confidence intervals calculated using the percentile method.

If these intervals contain the value zero, a '0' is recorded. If they do not contain the value zero, '1' is recorded if the lower limit is above zero as an indication that the difference is significantly positive (increase in animal densities after impact). A '-1' is recorded if the upper limit is below zero, indicating that the difference is significantly negative (decrease in animal densities after impact).

The function returns a list of two objects:

- The median for each difference
- The marker for significant positive and negative differences

```
differences <- getDifferences(beforePreds =
        bootPreds[predictData$impact == 0, ],
        afterPreds = bootPreds[predictData$impact == 1, ])


NOTE: 1 bootstrap(s) removed due to infinite values
```

12.1 Visualising significant differences

To illustrate the differences in animal densities after impact, we plot the calculated differences (Density$_{after}$ - Density$_{before}$) using the `quilt.plot` function.

The locations of the significant differences are added to the same plot using the '○' and '+' symbols.

```
# The median for each after - before difference
meandiff <- differences$meandiff
# The marker for each after - before difference:
# positive ('1') and negative ('-') significant differences
marker <- differences$significanceMarker
par(mfrow = c(1, 1))
quilt.plot(predictData$x.pos[predictData$impact == 0],
predictData$y.pos[predictData$impact ==  0],
meandiff, asp = 1, nrow = 104, ncol = 55)
# add + or - depending on significance of cells.  Just
# requires one significance out of all to be allocated
points(predictData$x.pos[predictData$impact == 0][marker == 1],
    predictData$y.pos[predictData$impact == 0][marker == 1],
    pch = "+", col = "darkgrey", cex = 0.75)
points(predictData$x.pos[predictData$impact == 0][marker == (-1)],
    predictData$y.pos[predictData$impact == 0][marker == (-1)],
    col = "darkgrey", cex = 0.75)
points(681417.3, 6046910, cex = 3, pch = "*", lwd = 1, col = "grey")
```

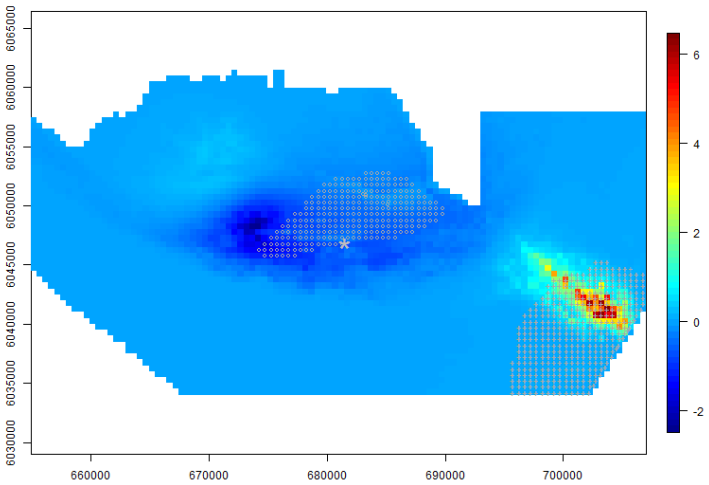


Figure 22: Mean differences in predicted bird density (mean birds/km^2) before and after impact. Positive values indicate more birds post impact and negative values fewer birds post impact. Significant differences were calculated using percentile confidence intervals: '+' indicates a significant positive difference and '-' a significant negative one. The grey star is the site of the impact event.

Conclusion

- There was a significant decline in animals around the impact site and in the North of the study area

- There was also a significant increase in animals in the south east of the study area

13 Comparison to the Truth

Here we will finalise our analysis by comparing our results with the truth. This is possible as our data was simulated and hence the parameter values known.

13.1 Detection function

We created detections from a half-normal detection function with a known scale parameter. Our model selection procedure identified the half-normal model as the one with the best fit.

	True value	**Maximum likelihood estimate**	**95% CI**
Scale parameter	120	116.1	(112.7, 122.4)

Conclusion

- We were able to identify the type of model correctly
- Our best guess of the true value for the scale parameter based on the data was 116.1.
- We were 95% sure the true scale parameter for the half-normal detection function was between 112.7 and 122.4. The true value for this parameter was captured by this interval.

13.2 Overdispersion and correlation

We created data that were both overdispersed and positively correlated within any `transect.id` (i.e. observed detections from the same transect repeat).

Conclusion

We conclude that our proposed methods are capable of identifying overdispersion and correlation in the data.

Truth	Model
Overdispersed data	estimated overdispersion parameter was greater than 1
Positively correlated data	accounted for using GEE-based *p*-values

13.3 Type of impact

For this particular data set, the total number of animals before and after impact did not change. The type of impact that we implemented was a redistribution from the area surrounding the impact into the south east of the study area.

Truth	Our model
Redistribution within study area	identified with a significant interaction term
Overall abundance remained the same	identified with the non-significant main effect for impact

Conclusion

We conclude that our model was capable of correctly identifying the redistribution effect and removed and reallocated birds to the correct locations despite highly correlated and overdispersed data.