Using the MRSea Package

Statistical Modelling of bird and cetacean distributions in offshore renewables development areas

October 21, 2013

Lindesay Scott-Hayward
Cornelia Oedekoven
Monique Mackenzie
Eric Rexstad

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

Contents

Chapter 1

Introduction

The MRSea package was developed for analysing data that was collected for assessing potential impacts of renewable developments on marine wildlife, although the methods are applicable to other studies as well. This user guide consists of three sections following this introduction.

We present two worked examples of which one is segmented line transect data (offshore scenario, Chapter 2) and one is grid count data (nearshore scenario, Chapter 3). Both examples are simulated data based on actual studies. The type of impact artificially imposed for these data sets was a redistribution of animals within the study area and the data are included in the package. The package also includes additional datasets simulated under no impact and overall decrease scenarios. All the coding necessary to reproduce the worked examples is given in the respective sections.

Following these examples, Chapter 4 provides extra tips and tricks for coding in R that might be useful in case the user needs to adjust the code for their own purposes.

A full description of each of the functions within the MRSea package can be found in the reference manual at: http://creem2.st-and.ac.uk/software.aspx. The manual and this document use version 0.1.1 of MRSea.

Please reference this document as:

Scott-Hayward, L.A.S., Oedekoven, C.S., Mackenzie, M.L. and Rexstad E. (2013). User Guide for the MRSea Package: Statistical Modelling of bird and cetacean distributions in offshore renewables development areas. University of St. Andrews contract for Marine Scotland; SB9 (CR/2012/05).

Chapter 2

Worked Example with Distance Sampling

2.1 Introduction

This chapter takes you through the process of fitting a detection function model to distance sampling data (section 2.5) and then fitting spatial models using the CReSS method in a GEE framework with SALSA for model selection (section 2.8). Section 2.10 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. The type of impact was a redistribution of animals within the study area.

2.2 Two-Stage Modelling of Distance Sampling Data

2.2.1 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 perpendicular distances from the transect to the detection. This detection function enables us to estimate the average detection probability in the covered area used to adjust observed counts for imperfect detection. We describe how to use the code from this package to fit detection functions in section 2.5.1, how to select between contending models in section 2.5.2 and to assess models in section 2.5.3.

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.

Interest generally lies in the count model, while the detection function model 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 count model. Non-parametric bootstrapping for distance sampling data is explained in section 2.11.

2.3 Preparing to conduct an analysis with MRSea

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)
```

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

```r
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
```

2.4 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 segments. The three types of information on the observed animals are:

- Observed perpendicular distances for each detection
- Total number of detections for each segment
- Cluster size for each detection

We begin with a data frame that contains all the information necessary. Columns can be divided into three levels, the observation, segment and transect level.

2.4.1 Data requirements: columns

Observation level

- *object* Object number, no repeats allowed

- *distance* Perpendicular distance in metres (for this example) for line transects, required if distances were recorded as exact
- 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

2.4.2 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. 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 we load the data.

```
# Loading the data
data(dis.data.re)
head(dis.data.re)


  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.re[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.

2.5 Distance Analysis

2.5.1 Fitting detection functions

We begin by fitting a half-normal detection function to our data[1]. We will consider other models and compare the relative fit of these models using AIC.

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

The ddf function

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.

[1]The ddf function is from the mrds package. See the reference material for the mrds package for more details on distance sampling with R

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

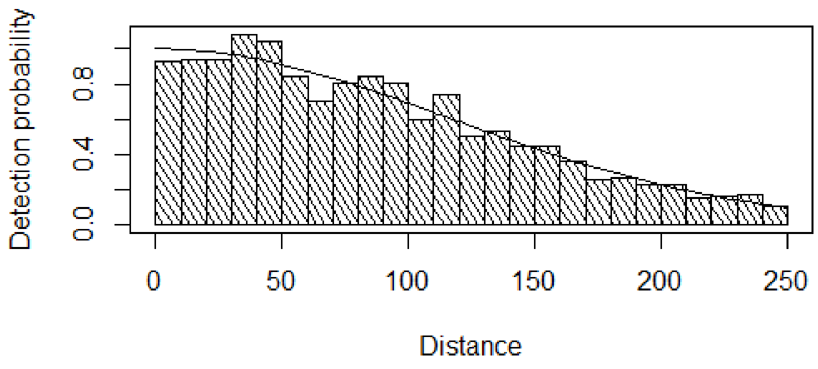


Figure 2.1: Half-normal detection function

2.5.2 Model Selection

More detection models can be fitted to determine if any of these provide a better relative fit to the data.

```
# Using a hazard-rate detection function
result.hr <- ddf(dsmodel=~mcds(key="hr", formula=~1),
     data = dis.data.re, 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.re, 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

The AIC value for each detection model can be printed using the *summary* function. 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.

2.5.3 Goodness of fit

There are several 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-square test is performed by 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 fits the data.

QQ-plot

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 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 performs the Kolmogorov-Smirnov and Cramér-von Mises tests.

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

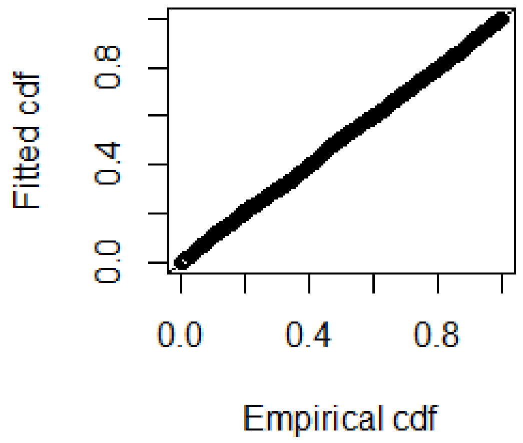


Figure 2.2: QQ plot for half-normal detection function fitted to our data

Conclusion

The points lie on the diagonal line, 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 fits the data.

2.5.4 Adjusting counts for imperfect detection

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 by twice the truncation distance(for two-sided transects).

2.5.5 Creating Count Data from Distance Data

We create a new data frame in which we sum the *NHATs* for each *segment.id* and discard the columns from the observation layer because the observations themselves are not modelled with the count modelling to follow.

```
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.

2.6 Introduction to Spatial Modelling

Introduction

Aim

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

The following sections detail the following:

1. model fitting

2. checking diagnostics
3. making predictions and inference

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

2.7 Loading the Data

Loading the Data

Either continue from the previous section with `count.data` or, load `count.data` if it was saved.

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`. Both are manufactured by `create.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
```

2.7.1 Exploratory Data Analysis (EDA)

Assess the data inputs for the modelling process.

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

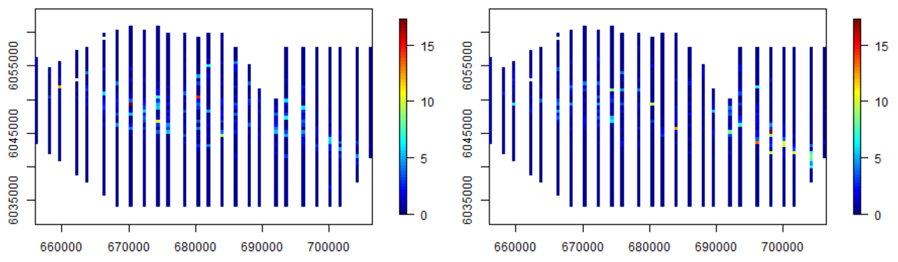


Figure 2.3: Estimated bird counts before (left) and after (right) an impact event. Each cell is 0.5 km^2 and the colour represents mean animal counts across four seasons.

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

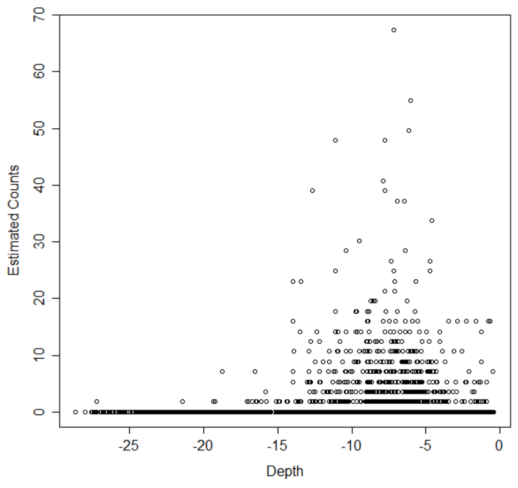


Figure 2.4: Plot of depth against the estimated bird counts.

EDA

- Birds were seen predominantly in shallow waters (Figure 2.4).
- Few birds were seen in waters deeper than 15m.

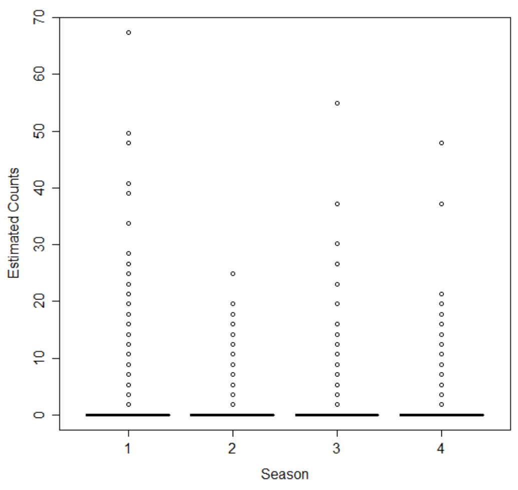


Figure 2.5: Plot of Season against the estimated bird counts.

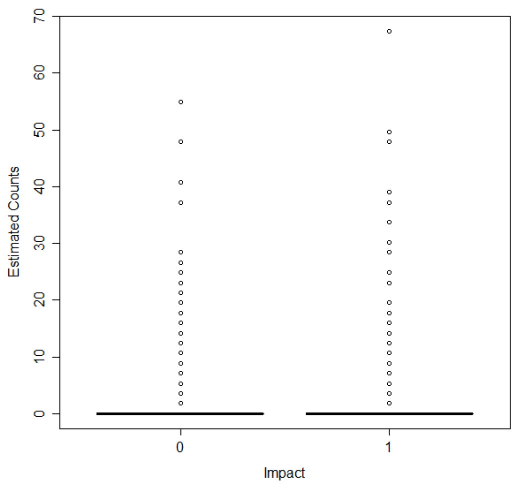


Figure 2.6: Plot of Impact against the estimated bird counts. Zero is pre impact and one is post impact.

- 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 2.5 & 2.6).

2.7.2 Checking for Collinearity

Variance Inflation Factors (VIFs)

These are used to assess collinearity between covariates and to tell us by how much the standard error is inflated by the other variables in the model (Fox and Weisberg, 2002).

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

Checking for Collinearity

```
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"
```

Conclusion:

The large values 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 for the response variable, otherwise there 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.

2.8 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. For more details see Hardin and Hilbe (2002).

Complex Region Spatial Smoother (CReSS)

Flexible spatial smoothing method. For more details see Scott-Hayward et al. (2013).

Spatially Adaptive Local Smoothing Algorithm (SALSA)

Automated knot selection procedure for both one-dimensional (e.g. depth) and two-dimensional (e.g. spatial) covariates. The knots are sources of flexibility in the surface that can raise or lower the surface. For more details see Walker et al. (2010).

2.8.1 Fitting a Smooth Term

A smooth of depth

Construct an object (`splineParams`) that contains the information required by SALSA for adaptive knot placement.

- Each covariate considered smooth is a list entry in `splineParams` and identified in `varlist`

- Other inputs include the data and, optionally, a grid of prediction data.
- The output list contains the covariate name, data, initial knot location (one knot at the mean), the boundary knots (greatest range of the 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)
# re-named for easier writing
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
```

2.8.2 Checking for Correlation

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

We fit a model containing all covariates of interest (`fullModel` above) and carry out some **runs tests** to check for **correlated residuals** (model assumes 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 is 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 2.7).

Plotting runs profile

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


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

- 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

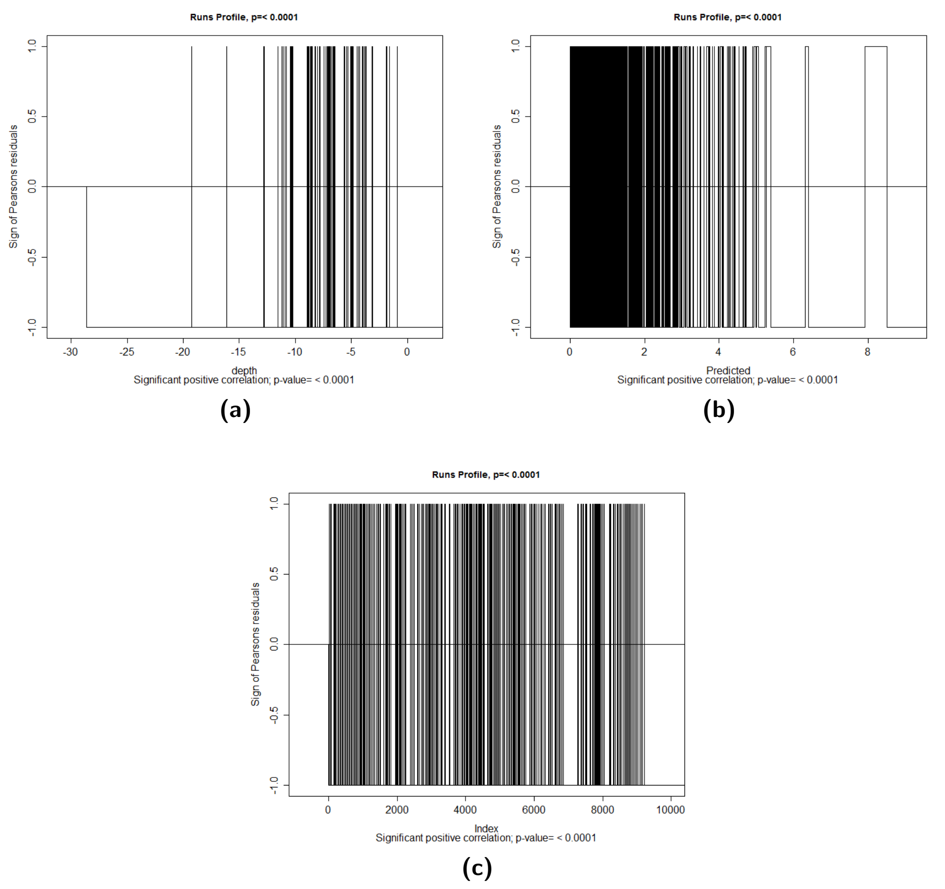


Figure 2.7: Runs profiles for residuals ordered by (a) depth, (b) predicted value and (c) temporally (by observation index). The p-values and text presented on each plot indicate if there is correlation present in the residuals. The lines are the strings of sequences of positive and negative residuals. A vertical line is the switch between a positive and negative run (or vice versa).

- 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 2.8). First make a column in the data (if it does not already exist) that represents the blocking structure.

```
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 (10 segments, therefore at a distance of 5km).
- 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

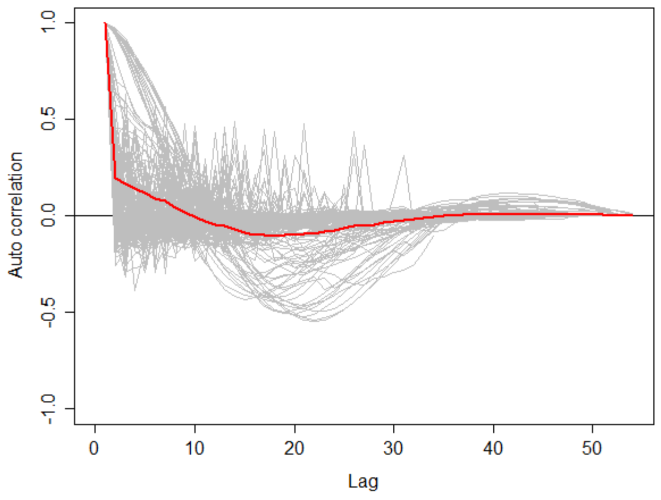


Figure 2.8: Plot of the correlation in residuals for each block (grey lines). The mean correlation at each lag is indicated in red.

2.8.3 Model Selection

Select what one-dimensional covariates to include in the 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**.

- Cumulative residuals are sequentially summed raw residuals on the scale of the response.
- 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 2.9 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"
```

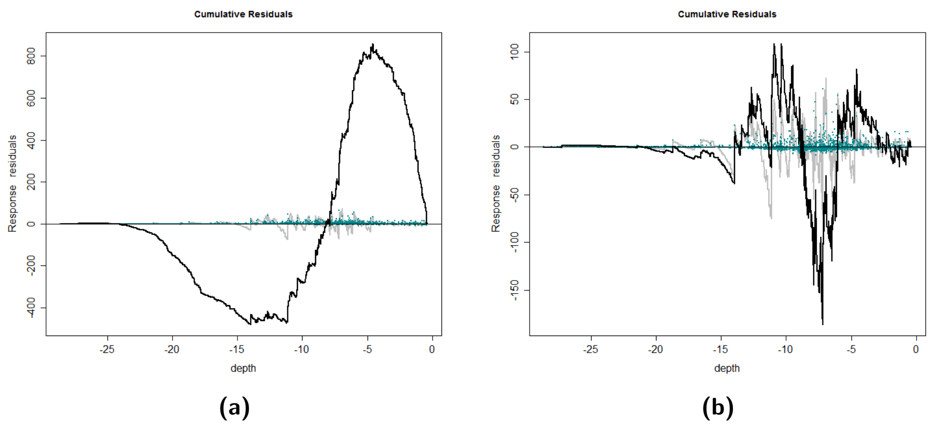


Figure 2.9: Cumulative residual plots residuals ordered by depth. (a) depth modelled as a linear term and (b) depth modelled with one knot at the mean depth. The blue points are the residual values, the black line represents the cumulative residuals. The grey line in the background is what we would expect the cumulative residuals to be if depth was modelled correctly.

Cumulative Residuals

- Depth as a linear term is not appropriate (Figure 2.9a).
- 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 2.9b).
- 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 argument `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

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

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

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

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 (see output above)
- 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)`) and an interaction between this term 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 (Scott-Hayward et al., 2013) for more details.
- Minimum, maximum and starting number of knots

The next section assumes that the knot grid has been defined. See section 4.2.2 for guidance to do this.

Spatial component

```
# load 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 2.10 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 are 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.

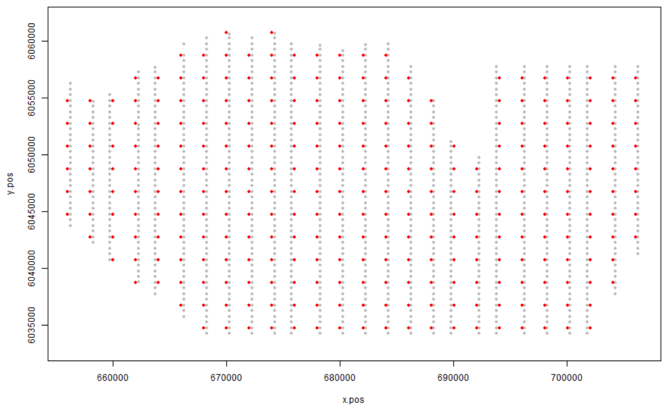


Figure 2.10: Data and knot locations. The data is in grey and the knot locations in red. Units are in meters.

The distance matrices 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. The `runSALSA2D` function may also take geodesic distance matrices (as the fish swims rather than as the crow flies). See Scott-Hayward et al. (2013) for more details on using geodesic distances and CReSS.

```
# 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 SALSA chosen 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 created below) for each knot.

- `getRadiiChoices` selects a number of radii (default $= 8$) based on the distances in the data to knot distance matrix created above.

```
# create 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,
    d2k = distMats$dataDist, k2k = distMats$knotDist, splineParams = splineParams)
```

The structure of the output:

- `bestModel`. The model object for the best fitted model.

- `splineParams`. The first list entry in the spline parameter object is updated with all the information used for fitting the spatial smooth.
- `fitStat`. The fit statistic of the best model.

```
str(salsa2dOutput_k6, max.level=1)


List of 3
 $ bestModel   :List of 30
  ..- attr(*, "class")= chr [1:2] "glm" "lm"
 $ splineParams:List of 2
 $ fitStat     : num 5456
```

```
str(salsa2dOutput_k6$splineParams, max.level=2)


List of 2
 $ :List of 14
  ..$ knotDist     : num [1:290, 1:290] 0 2000 4000 6000 8000 10000 12000 ...
  .. ..- attr(*, "dimnames")=List of 2
  ..$ dist         : num [1:9232, 1:290] 14820 15129 15448 15776 16113 ...
  ..$ gridResp     : int [1:364] NA NA NA NA NA NA 668024 670024 672024 ...
  ..$ grid         :'data.frame':       364 obs. of  2 variables:
  .. ..- attr(*, "out.attrs")=List of 2
  ..$ response     : num [1:9232] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ knotgrid     :'data.frame':       364 obs. of  2 variables:
  ..$ datacoords   : num [1:9232, 1:2] 656250 656250 656250 656250 656250 ...
  ..$ radii        : num [1:8] 136 370 1010 2756 7518 ...
  ..$ minKnots     : num 4
  ..$ maxKnots     : num 20
  ..$ gap          : num 4000
  ..$ knotPos      : int [1:5] 348 24 91 192 295
  ..$ radiusIndices: num [1:5] 8 3 1 3 2
  ..$ invInd       : num [1:364] 0 0 0 0 0 0 1 2 3 4 ...
 $ :List of 5
  ..$ covar      : chr "depth"
  ..$ explanatory: num [1:9232] -27.4 -27.6 -28.6 -28 -27.5 ...
  ..$ knots      : num [1:2] -17.18 -7.53
  ..$ bd         : num [1:2] -28.6 -0.2
  ..$ degree     : num 2
```

`knotPos` gives the index of chosen knot locations. The index identifies which rows of the knot grid were selected by SALSA. For an explanation of the other entries in the spatial part of the `splineParams` object refer to the help file for `runSALSA2D`.

Multiple SALSA runs

The example uses 6 initial knots. There is a risk that the SALSA algorithm may get stuck in local minima or maxima and so we recommend that different numbers of initial knots are considered. Here we try 6, 8, 10 and 12 (Table 3.2).

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

Example:

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

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


[1] 6.087
```

Choosing a model

Table 2.1: 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 5 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 2.24 ($\sqrt{5}$; a threshold often seen in the literature) so we are happy there is no issue with collinearity.

2.8.4 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, . ~ .)
```

The baseModel is updated with the parameters returned by SALSA2D using the update function[2]. This makes sure that the baseModel object is made using the parameters currently in the workspace.

```
# 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)
```

Note: The family specified for the GEE model is Poisson because the dispersion parameter is automatically estimated using geeglm. Therefore, we need not (and cannot) specify the quasi-Poisson family.

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 0.5376
3                  depth <0.0001
4         s(x.pos, y.pos) <0.0001
5 s(x.pos, y.pos):impact 0.0006
```

Do we remove the main impact effect?

[2]The update function is part of the basic functionality in R and can be used to quickly re-fit a model removing or adding covariates, changing the family or data set or in this case refreshing the parameters used in LocalRadialFunction.

- Keep the main effect for `impact` because it is part of the interaction term and is difficult to interpret.
- Fit a model without the interaction term to test for a significant increase/decrease pre and post impact.
- If the interaction is significant but the `impact` term in a model without the interaction is not significant then there has been a re-distribution of animals but no overall change.
- This is the case for this model. p¿¿0.05 for `impact` in a model with no interaction term.

Below is an example of how to remove the interaction term using the `update` function.

Removing the main `impact` effect

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


[1] "Getting marginal p-values"
               Variable p-value
1                season <0.0001
2                impact  0.5376
3                depth  <0.0001
4        s(x.pos, y.pos) <0.0001
```

2.8.5 Assessing covariate relationships

Partial Residual Plots[3]

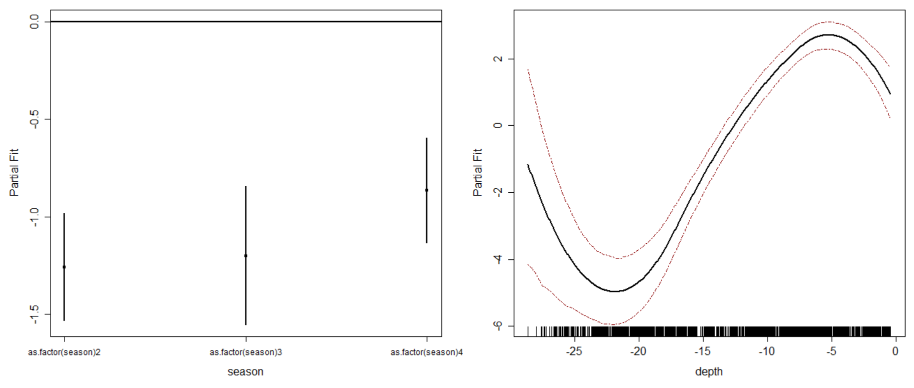
These plots allow visual examination the one-dimensional covariates in the model (depth, season, impact) and to check that smooth/linear terms are specified correctly.

[3]see 'component-plus-residual' plots in Fox and Weisberg (2002) for more details.

The partial residuals, e_{ij}, for predictor x_j are formed by adding the fitted linear component in this predictor to the least squares residuals[4]. The partial residuals are then plotted against x_j

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


[1] "Making partial plots"
```

Assessing covariate relationships



(a) **(b)**

Figure 2.11: Partial plots for (a) season and (b) depth.

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
- Depth: a declining non-linear relationship with increased depth (fewer birds in deeper waters).
- Most birds are estimated in waters 3-8m deep.

[4] $e_{ij} = e_i + b_j x_{ij}$

- Small confidence interval surrounding the predicted relationship of birds with depth indicates a precisely estimated relationship.

Note: if the confidence intervals for depth were very wide it would be reasonable to assume depth could be adequately modelled as a linear term.

2.9 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 created using the `runDiagnostics` function.

Marginal R-squared value

It is widely used to assess models fitted to both uncorrelated 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

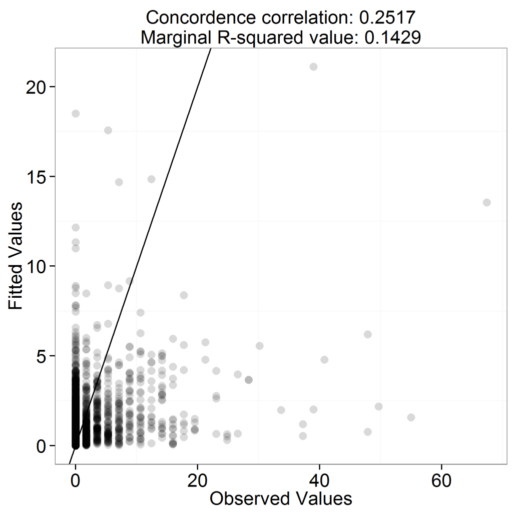


Figure 2.12: Diagnostic plots of observed vs fitted values, where the diagonal line indicates where data should lie for a perfect fit.

Observed vs fitted

- High observed values are under-predicted (Figure 2.12) as is common for very noisy data
- Observed zeros tend to be over-predicted (as is likely given they cannot be under-predicted)

- Marginal R-squared and concordance correlation are low

- Conclusion: poor model fit

Fitted values vs scaled Pearsons residuals Plot

The plot gives an indication of the correct mean-variance relationship assumed under the model. We 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 (estimated by `geeglm`) for the latter
- Thus we plot Fitted values against scaled Pearsons residuals

Fitted values vs scaled Pearsons residuals

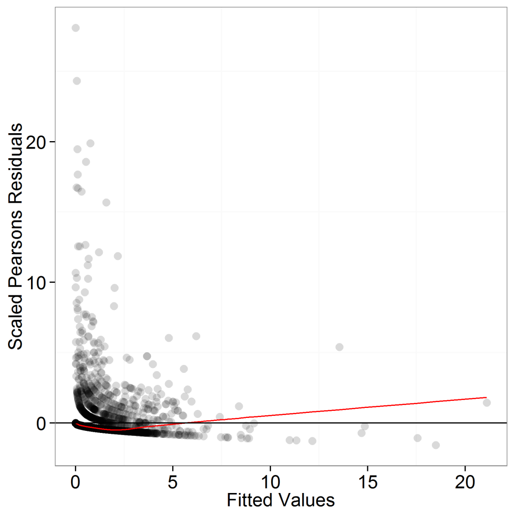


Figure 2.13: Diagnostic plot of fitted values vs scaled Pearsons residuals, where the red line is a locally weighted least squares regression to indicate pattern in the plot, which might otherwise be hidden due to over-plotting.

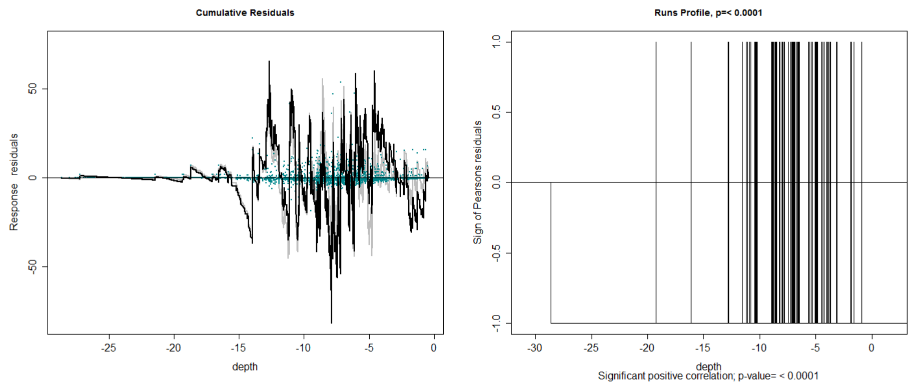
- Possible pattern in residuals but hard to tell due to over-plotting (Figure 2.13)
- Locally weighted least squares regression line does not indicate an unusual pattern
- Conclusion: no issue with model assumption (mean-variance relationship)

2.9.1 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'))
```

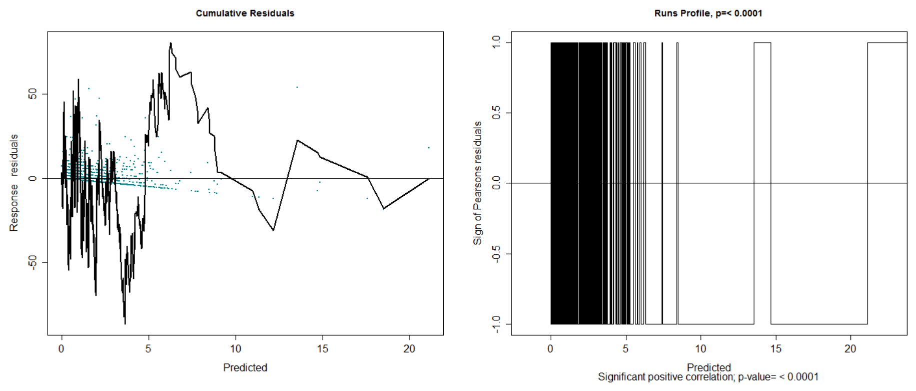


(a) (b)

Figure 2.14: 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 (Figure 2.14)
- Variable modelled appropriately (similar to expected (grey line) and better than Figure 2.9)
- Fewer runs than would be expected if residuals were random ($p < 0.05$)
- Significant positive correlation between residuals when ordered by depth



(a) **(b)**

Figure 2.15: 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 (Figure 2.15)
- 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

Ordered by **index** (temporally)

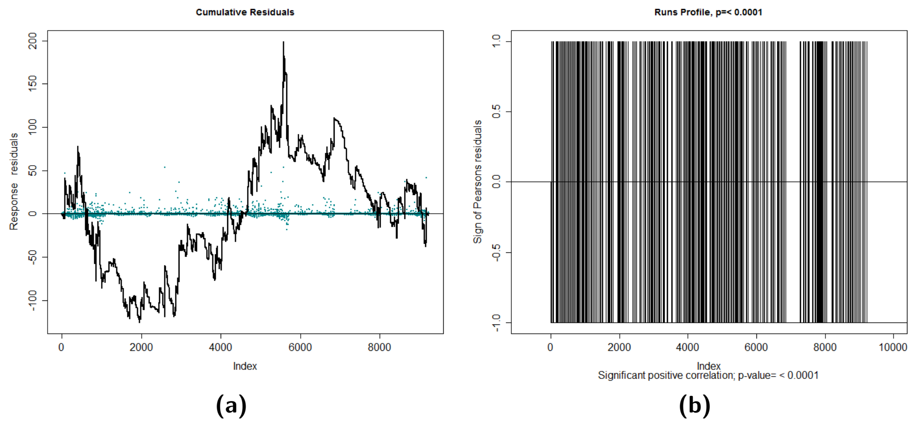


Figure 2.16: Cumulative residual plot (a) and runs profile (b) for residuals ordered by the index of observations (temporally). The blue points are the residual values and the black line represents the cumulative residuals.

- Early observations (before impact) are over predicted and the later (after impact) are under predicted (Figure 2.16)
- 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

2.9.2 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 block 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.

```
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)
```

COVRATIO statistics

- there will always be blocks outside the dashed lines (they are quantiles)
- As it is a relative measure, blocks with statistics far away from the majority of the statistics may be of concern
- Blocks with COVRATIO statistic < 1 are of most concern (decrease in precision when removed)
- There is a marked decrease in standard errors when blocks 2411, 12811 and 11911 are removed.

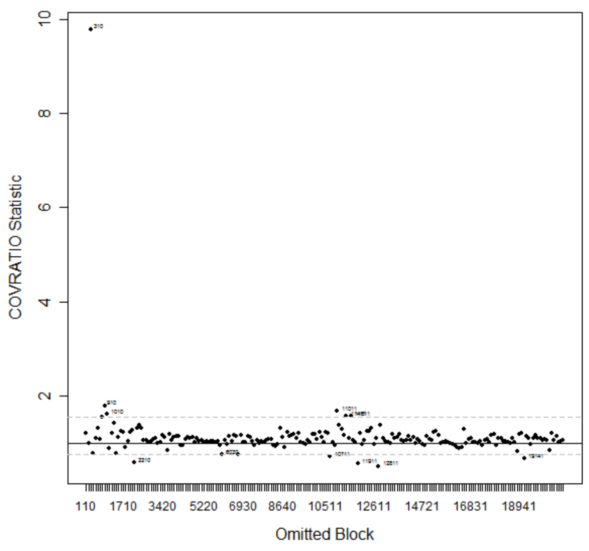


Figure 2.17: Plot of COVRATIO statistics; the dashed grey lines indicate the lower 2.5% and upper 97.5% quantiles of the statistics.

- These blocks contain high animal counts so naturally removing the blocks would lower the variance.
- Block 310 is far away but results in an increase in standard errors when removed (Figure 2.17)
- 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 blocks with statistics far away from the majority of the statistics 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 2.18).
- Conclusion: we may choose to fit a model with and without this block to assess the change in predictions

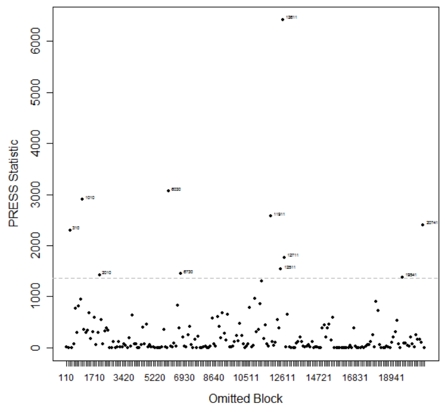


Figure 2.18: Plot of PRESS statistics; 95% of the statistics fall below the dashed grey lines. Labelled points on both plots are outside the grey dashed line(s) and are labelled with the identifier for the block that has been removed to create the statistic (not an observation number).

2.9.3 Raw Residuals

Raw residuals (observed - fitted 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; Figure 2.19)
- No systematic over/under prediction
- Conclusion: no spatial issue with raw residuals

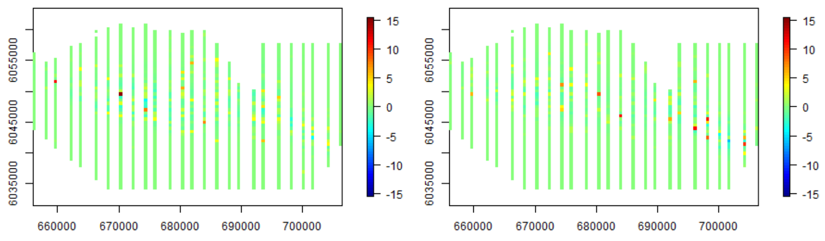


Figure 2.19: Raw residuals before impact (left) and after impact (right). These residuals are fitted values - observed values (mean* birds/km^2). * mean density because the predictions for each cell are for each season.

```r
# 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 2.2: 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. The solution in red is a suggestion for a real analysis. Here we modelled the covariates that were used to generate the data so we know the lack of fit was down to the noise in the data, not to missing covariates.

	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	-

2.10 Prediction and Inference

2.10.1 Data requirements for predicting

For making predictions, we require a data frame containing grid data with the records for which to make predictions. 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
```

Note: the `truth` column is an artefact of this being simulated data. Ordinarily the user would not know this.

2.10.2 Making predictions

We 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.

2.10.3 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*. From the model coefficients it is not obvious where the redistribution has occurred. We are going to investigate this by visualising the redistribution using our predictions.

The following is code for plotting the model predictions in Figure 2.20.

```
# plotting the predictions for before and after impact
# get the plot dimensions. We know each cell is 500x500m
dims<-ceiling(getPlotdimensions(x.pos=predictData$x.pos, predictData$y.pos,
   segmentWidth=500, segmentLength=500))
# round up dimensions if necessary - in this case our grid is not quite regular.
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$impact==0], asp=1, nrow=dims[1], ncol=dims[2],
    zlim=c(0, maxlim))
quilt.plot(predictData$x.pos[predictData$impact==1],
    predictData$y.pos[predictData$impact==1], preds[predictData$impact==1],
    asp=1,nrow=dims[1], ncol=dims[2], 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 the maximum value in the scale of either plot.

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
- The dimensions of the plot (`nrow` and `ncol`) determine the size of each grid cell. Since we know our grid to be 500m by 500m we can use `getPlotDimensions` to find the values for these arguments.

Conclusion

- Decline in bird density in the central region
- Increase in density in the south east of the study area

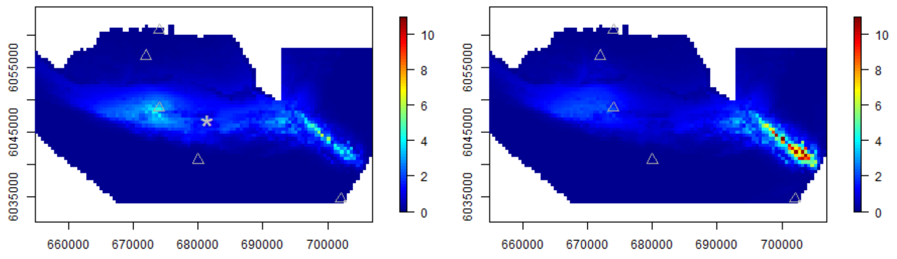


Figure 2.20: Predictions of bird density (mean birds/km^2) from the fitted model for before (left) and after (right) an impact event. The * indicated the location of the impact; the $\triangle$ indicate the knot locations.

- 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.

2.11 Bootstrap Confidence Intervals

Following the predictions from section 2.10.2, we now create replicate predictions and use these to construct percentile confidence intervals for each prediction grid cell. We run B iterations performing the following:

- resample the original transect data with replacement
- refit the detection function to the bootstrapped data and re-estimate the *NHATs* (estimated number of animals)
- refit 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 refitting the detection model as described above.

Because the prediction object can be quite large, it is not returned but saved to file in the current working directory with the file name 'predictionboot.RData'.

```
# 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.

An optional argument `quants` can be used to specify an interval other than 95%. The default is `quants = c(0.025, 0.975)`

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

2.11.1 Visualising Bootstrap Confidence Intervals

Figure 2.21 depicts the range of uncertainty in the spatial estimates of density. These can be contrasted with the point estimates found in Figure 2.20.

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.

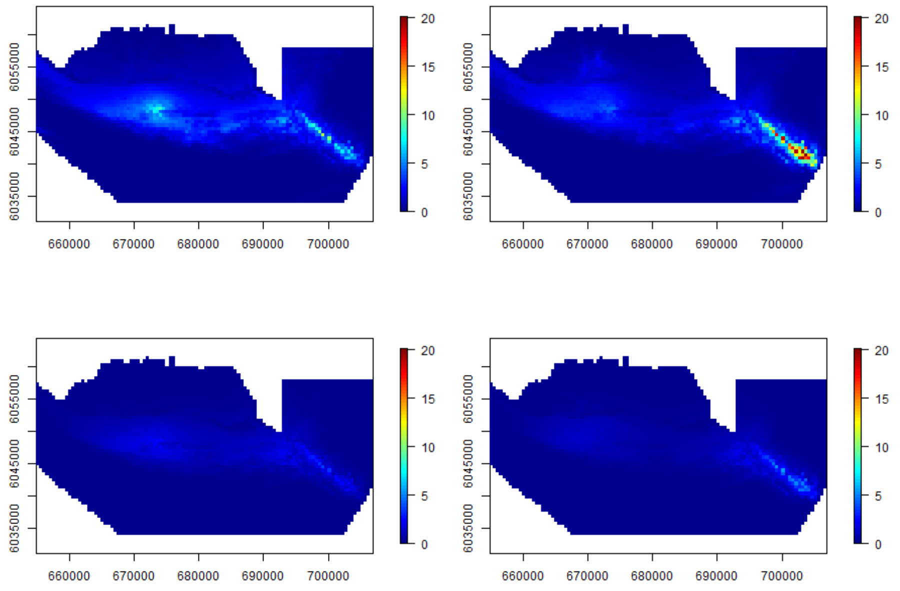


Figure 2.21: Upper (top) and lower (bottom) 95% confidence intervals of bird density (mean birds/km^2) from the fitted model from before (left) and after (right) impact.

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.

2.12 Identifying Differences

In this section we estimate the differences in predictions for each corresponding pair of records from our prediction grid. The first half of our prediction grid 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
```

2.12.1 Visualising 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 differences are added to the same plot using the '$\circ$' and '$+$' symbols.

```
# The median for each after - before difference
mediandiff <- differences$mediandiff
# 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$impact ==  0],
    mediandiff, asp = 1, nrow = 104, ncol = 55)
# add + or - depending on significance of cells.  Just
# requires one significance out of all to be allocated
points(predictData$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")
```

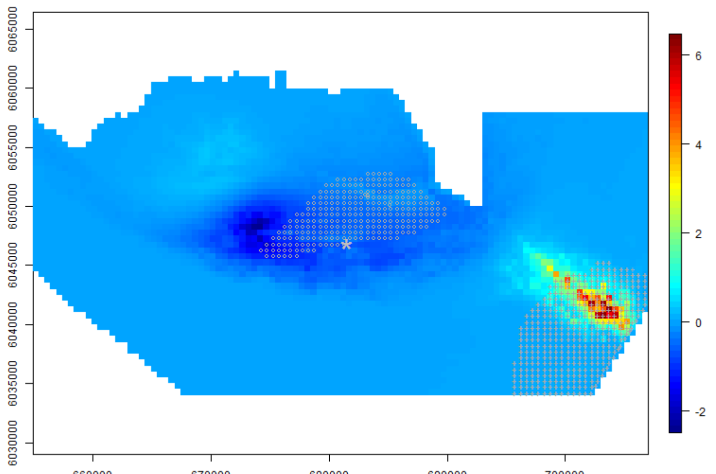


Figure 2.22: Mean differences in predicted bird density (mean birds/km^2) before and after impact. Positive values indicate more birds post impact and negative values fewer birds post impact. Considerable differences were calculated using percentile confidence intervals: '+' indicates a large positive difference and '-' a large negative one. The grey star is the site of the impact event.

Conclusion

- There was a large decline in animals around the impact site (Figure 2.22).
- There was also a significant increase in animals in the south east of the study area

2.13 Comparison to the Truth

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

2.13.1 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.

2.13.2 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).

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

Conclusion

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

2.13.3 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 method was capable of correctly identifying the redistribution effect and reallocated birds to the correct locations despite highly correlated and overdispersed data.

Chapter 3

Worked Example without Distance Sampling

3.1 Introduction

This chapter takes you through the process of fitting spatial models using the CReSS method in a GEE framework with SALSA for model selection (section 3.4). We use simulated vantage point data as our case study. Section 3.6 uses the fitted count model to make predictions across the entire study area, which we use to draw inference from our study. We use simulated vantage point data as our case study. The type of impact was a redistribution of animals within the study area.

Aim

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

Note: No distance sampling analysis can be conducted as there is no information available to do so.

The following sections take you through:

- model fitting
- checking diagnostics
- making predictions and inference

3.2 Preparing to conduct an analysis with MRSea

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.

See section 2.3 for information on the datasets this package holds.

3.3 Loading the Data

3.3.1 Data Requirements

There are a few requirements for analysing data using the `MRSea` package.

- The geographic coordinates must be labelled `x.pos` and `y.pos`
- There must be a column labelled `area` representing the effort associated with each coordinate

```
# Loading the data
data(ns.data.re)
data <- ns.data.re
attach(data)
head(data, n=3)


  x.pos y.pos   area  floodebb observationhour GridCode Year
1  1500 -4500 0.3853       EBB              12      a11    9
2  1500 -4500 0.3853     FLOOD               8      a11    9
3  1500 -4500 0.3853     FLOOD               9      a11    9

  DayOfMonth MonthOfYear impact  birds cellid
1         13           3      0      0      1
2         16           3      0      0      2
3         16           3      0      0      3
```

3.3.2 Exploratory Data Analysis (EDA)

Assess the data inputs for the modelling process.

- Check for unusual covariate values
- Identify possible relationships between covariates and the response (animal counts)
- This data was collected by a cliff-top observer (grey dot, Figure 3.1)

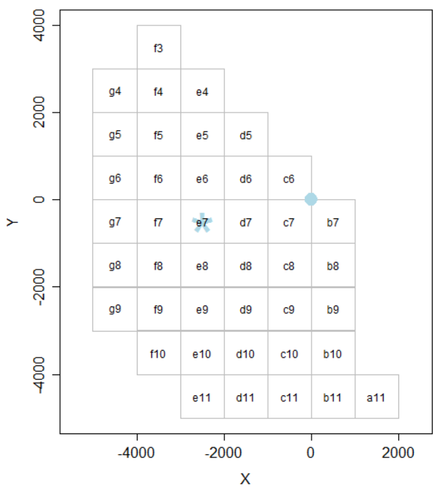


Figure 3.1: Grid cell identifiers for the study region. The grey circle is the location of a cliff-top observer and the star the site of the impact event.

- Most animals are seen near to the observer (Figure 3.2)
- Few are seen to the far right and far left of the observer

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

EDA

- Birds were seen predominantly in the morning hours (7-12).
- Few birds were seen very early and very late.
- Non-linear relationship between observation hour and bird counts.

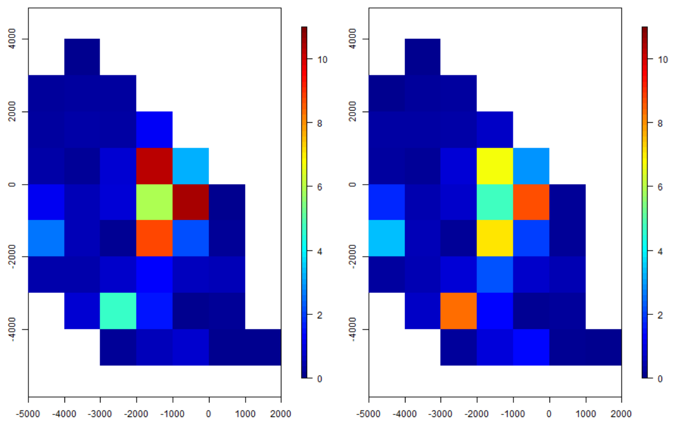


Figure 3.2: Estimated bird counts before (left) and after (right) an impact event. Each cell is 1 km^2 and the colour represents bird count.

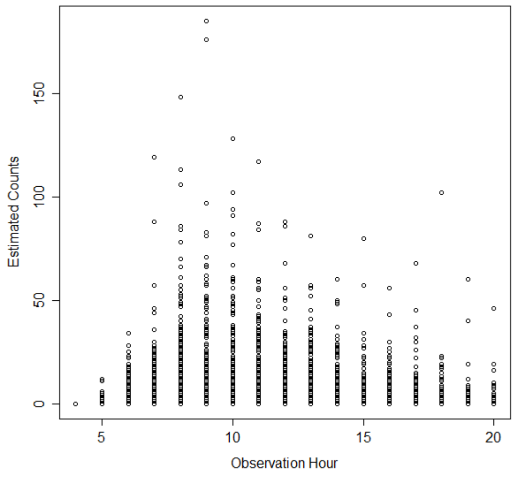


Figure 3.3: Plot of observation hour against the estimated bird counts.

- Difficult to identify any relationship between tide state/impact and bird counts due to the large number of zeros in the data (Figures 3.4 & 3.5)..

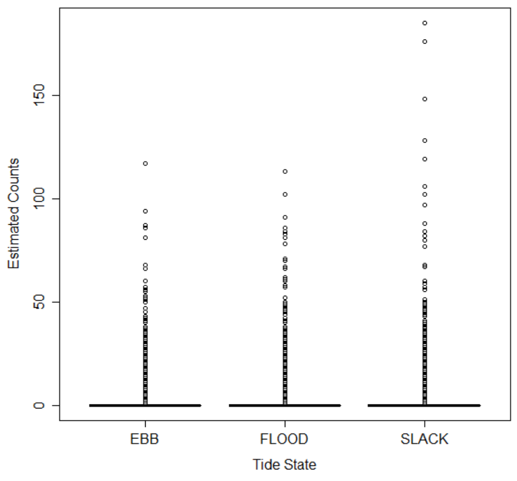


Figure 3.4: Plot of tide state against the estimated bird counts.

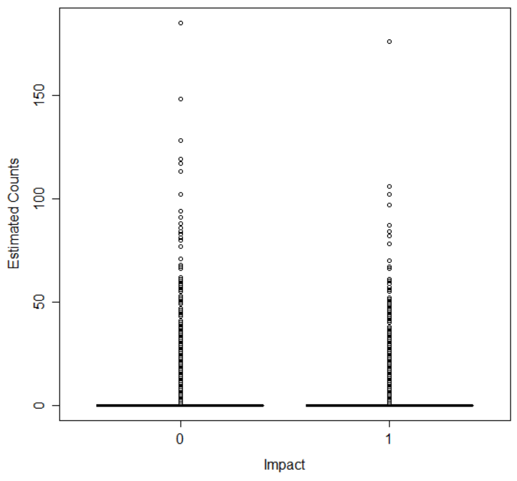


Figure 3.5: Plot of Impact against the estimated bird counts. Zero is pre impact and one is post impact.

3.3.3 Checking for Collinearity

Variance Inflation Factors (VIFs)

These are used to assess collinearity between covariates and to tell us by how much the standard error is inflated by the other variables in the model. For more details see section 2.7.2 or Fox and Weisberg (2002).

```
fullModel <- glm(birds ~ as.factor(floodebb) + as.factor(impact) +
    observationhour + x.pos + y.pos, family = poisson, data = data)
vif(fullModel)


                     GVIF Df GVIF^(1/(2*Df))
as.factor(floodebb) 1.006  2           1.002
as.factor(impact)   1.000  1           1.000
observationhour     1.006  1           1.003
x.pos               1.323  1           1.150
y.pos               1.323  1           1.150
```

```
[1] "Maximum VIF is: 1.15"
```

Conclusion:

The maximum VIF is approximately 1, which means that there is no issue with collinearity (no inflation of standard errors).

Checking Factor Covariates

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

```
# check all factor levels have counts
checkfactorlevelcounts(factorlist=c("floodebb", "impact"), data, data$birds)


[1] "floodebb 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: tide state and impact are fine for use in the model.

3.4 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. For more details see Hardin and Hilbe (2002).

Complex Region Spatial Smoother (CReSS)

Flexible spatial smoothing method. For more details see Scott-Hayward et al. (2013).

Spatially Adaptive Local Smoothing Algorithm (SALSA)

Automated knot selection procedure for both one-dimensional (e.g. depth) and two-dimensional (e.g. spatial) covariates. The knots are sources of flexibility in the surface that can raise or lower the surface. For more details see Walker et al. (2010).

3.4.1 Fitting a Smooth Term

A smooth of observation hour

Construct an object (`splineParams`) that contains the information required by SALSA for adaptive knot placement.

- Each covariate considered smooth is a list entry in `splineParams` and identified in `varlist`
- Other inputs include the data and, optionally, a grid of prediction data.
- The list contains the covariate name, data, initial knot location (one knot at the mean), the boundary knots (greatest range of the prediction data and `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(ns.predict.data.re)
# re-named for easier writing
```

```
predictData<-ns.predict.data.re

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

List of 2
 $ : list()
 $ :List of 5
  ..$ covar      : chr "observationhour"
  ..$ explanatory: int [1:27798] 12 8 9 10 11 12 13 14 15 8 ...
  ..$ knots      : num 12
  ..$ bd         : num [1:2] 4 20
  ..$ degree     : num 2
```

3.4.2 Checking for Correlation

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

We fit a model containing all covariates of interest (`fullModel` above) and carry out some **runs tests** to check for **correlated residuals** (model assumes 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 = -71.59, 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 3.6).

Plotting runs profile

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

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

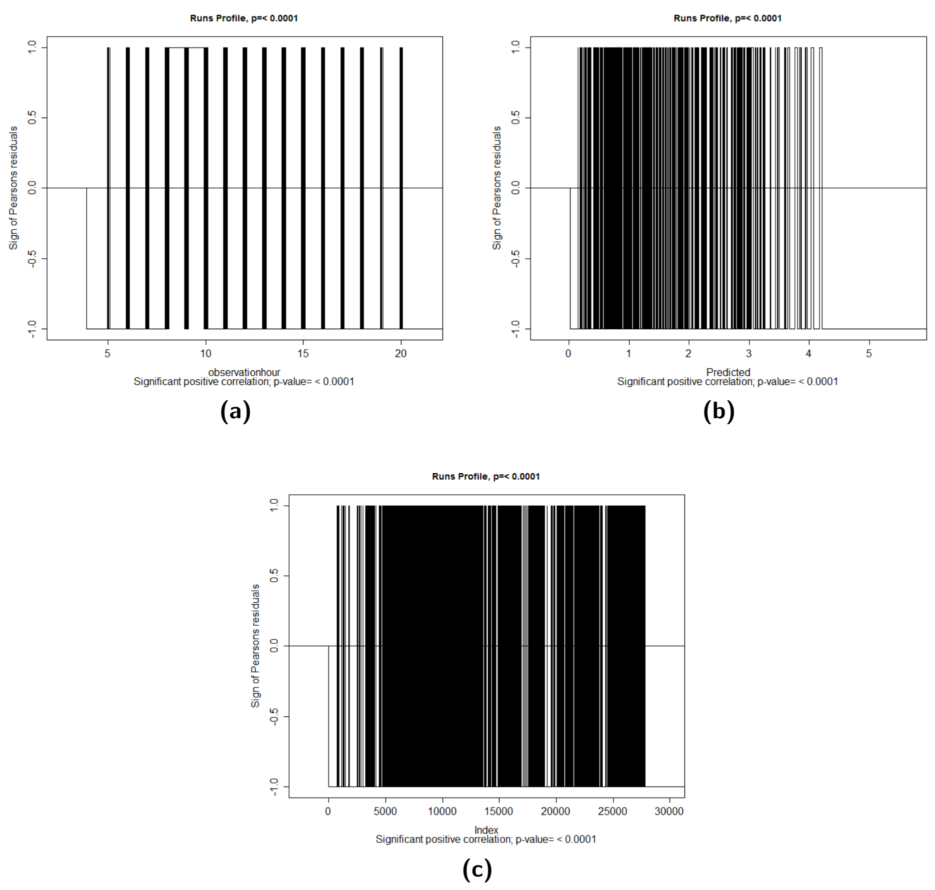


Figure 3.6: Runs profiles for residuals ordered by (a) observation hour, (b) predicted value and (c) temporally (by observation index). The p-values and text presented on each plot indicate if there is correlation present in the residuals. The lines are the strings of sequences of positive and negative residuals. A vertical line is the switch between a positive and negative run (or vice versa).

- 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 independent
- Blocks are usually determined by looking at the sampling design
- Here we use the unique cell identifier within each day of observation: Each grid cell is considered independent but residuals may be non-independent within a block.
- There are 41 grid cells repeated several days a month, for 12 months over four years times, giving us 5576 blocks.

Autocorrelation function plot

We can check the correlation declines within our blocks by plotting the autocorrelation of the model residuals by block (Figure 3.7). First make a column in the data (if it does not already exist) that represents the blocking structure.

```
data$blockid <- paste(data$GridCode, data$Year, data$MonthOfYear,
    data$DayOfMonth, sep = "")
# **** blockid must be a factor or numeric, not a character ****
data$blockid <- as.factor(data$blockid)
 runACF(data$blockid, fullModel, store = F)
```

- Some blocks have little correlation, whilst others have high correlation (0.5) at a lag of 6 (data points are at hourly intervals so a lag of 6 is 6 hours.)
- It is these blocks with the higher correlation that must be accounted for
- Conclusion: Our blocking structure is suitable

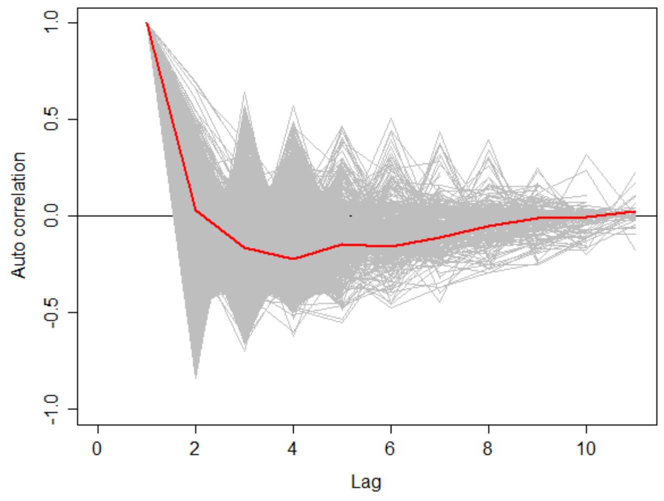


Figure 3.7: Plot of the correlation in residuals for each block (grey lines). The mean correlation at each lag is indicated in red.

3.4.3 Model Selection

Select what one-dimensional covariates to include in the model and use SALSA to determine the knot locations of those that are continuous: `observation hour`.

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

- Cumulative residuals are sequentially summed raw residuals on the scale of the response.
- 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 3.8 shows cumulative residual plots from a model with depth modelled as a linear term and with one knot at the mean.

Plotting Cumulative Residuals

```
# plot cumulative residuals for model with observationhour as smooth term
plotCumRes(fullModel, varlist= c("observationhour"), splineParams)
```

```
"Calculating cumulative residuals"
```

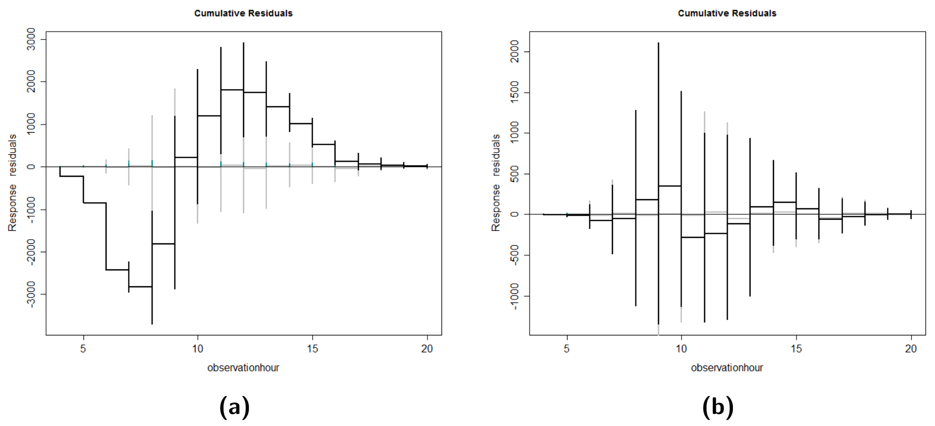


Figure 3.8: Cumulative residual plots residuals ordered by observation hour. (a) observation hour modelled as a linear term and (b) depth modelled with one knot at the mean observation hour. The blue points are the residual values, the black line represents the cumulative residuals. The grey line in the background is what we would expect the cumulative residuals to be if observation hour was modelled correctly.

Cumulative Residuals

- Observation hour as a linear term is not appropriate (Figure 3.8a).
- The black line (our model) does not correspond well with the expected line (grey) and shows systematic over prediction at early hours and under prediction in the early afternoon.
- Observation hour as a smooth term with one knot at the mean is much better but the black line may mask the expected line in places due to the discrete nature of the variable (Figure 3.8b).
- Conclusion:
  - we might want to consider more flexibility for `observationhour` 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 argument `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$response <- data$birds

# set initial model without the spline terms
initialModel <- glm(response ~ as.factor(floodebb) + as.factor(impact) +
    offset(log(area)), family = "quasipoisson", data = data)


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

```
# run SALSA
salsa1dOutput <- runSALSA1D(initialModel, salsa1dlist, varlist=
    c("observationhour"), factorlist=c("floodebb", "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 34341
```

```
# knots chosen for observation hour
salsa1dOutput$splineParams[[2]]$knots


[1]  9 15 17
```

- Initial model - one knot at the mean observation hour (12pm)
- SALSA result - three knots selected (see output above)
- The result of SALSA is additional flexibility added to the model for the relationship between bird counts and observation hour.

Spatial component

- Next we add a two dimensional 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 (Scott-Hayward et al., 2013) for more details.
- Minimum, maximum and starting number of knots

The next section assumes that the knot grid has been defined. See section 4.2.2 for some guidance to do this.

Spatial component

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

The knot points in `knotgrid` are located at the centre of each of the grid cells containing data.

The distance matrices 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 Scott-Hayward et al. (2013) for more details on geodesic distances.

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


List of 2
 $ dataDist: num [1:27798, 1:41] 4000 4000 4000 4000 4000 4000 4000 ...
 $ knotDist: num [1:41, 1:41] 0 1000 2000 3000 4000 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:41] "3" "4" "5" "6" ...
  .. ..$ : chr [1:41] "3" "4" "5" "6" ...
```

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 created below) for each knot.
- getRadiiChoices selects a number of radii (default $= 8$) based on the distances in the data to knot distance matrix created above.

```
# create 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 = 1, 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, 12, 14 and 16 (Table 3.1).

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

Example:

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

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


[1] 33.91
```

Choosing a model

Table 3.1: 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	-	-	33.9058
1D/2D terms	6	6	28.6426
1D/2D terms	8	8	28.1481
1D/2D terms	10	10	27.9571
1D/2D terms	12	12	27.0373
1D/2D terms	14	14	27.0474
1D/2D terms	16	16	27.0367

The best model chosen using CV score is the model that uses a spatial smooth with 16 spatial knots.

```
# having chosen the 2d interaction model, save the model object
baseModel <- salsa2dOutput_k16$bestModel
# update spline parameter object
splineParams <- salsa2dOutput_k16$splineParams
```

3.4.4 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 was 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 = -88.67, p-value < 2.2e-16
```

GEE framework

There is significant positive residual 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(floodebb) + as.factor(impact) +
    bs(observationhour, 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 = 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, . ~ .)
```

The `baseModel` is updated with the parameters returned by SALSA2D using the `update` function[1]. This makes sure that the `baseModel` object is made using the parameters currently in the workspace.

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

Note: The family specified for the GEE model is Poisson because the dispersion parameter is automatically estimated using `geeglm`. Therefore, we need not (and cannot) specify the quasi-Poisson family.

Checking p-values

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


[1] "Getting marginal p-values"
                Variable  p-value
1               floodebb  <0.0001
2                 impact   0.5615
3        observationhour  <0.0001
4        s(x.pos, y.pos)  <0.0001
5 s(x.pos, y.pos):impact   0.0241
```

Do we remove the main `impact` effect?

- Keep the main effect for `impact` because it is part of the interaction term and is difficult to interpret.
- Fit a model without the interaction term to test for a significant increase/decrease pre and post impact.
- If the interaction is significant but the `impact` term in a model without the interaction is not significant then there has been a re-distribution of animals but no overall change.

[1]The update function is part of the basic functionality in `R` and can be used to quickly re-fit a model removing or adding covariates, changing the family or data set or in this case refreshing the parameters used in `LocalRadialFunction`.

- This is the case for this model. $p¿¿0.05$ for `impact` in a model with no interaction term.

Below is an example of how to remove the interaction term using the `update` function.

Removing the main `impact` effect

```
# how to remove impact
noint.model<-update(geeModel, .~. - as.factor(impact):
                     LocalRadialFunction(radiusIndices, dists, radii, aR))
# reshow p-values
getPvalues(noint.model, varlist = c("observationhour"), factorlist = c("floodebb",
    "impact"))


[1] "Getting marginal p-values"
            Variable  p-value
1           floodebb  <0.0001
2             impact   0.5615
3    observationhour  <0.0001
4    s(x.pos, y.pos)  <0.0001
```

3.4.5 Assessing covariate relationships

Partial Residual Plots[2]

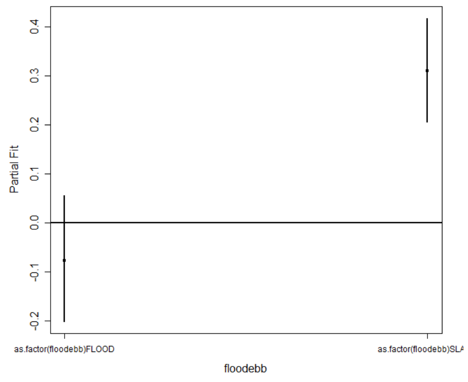
These plots allow visual examination the one-dimensional covariates in the model (observation hour, tide state and impact) and to check that smooth/linear terms are specified correctly.

The partial residuals, e_{ij}, for predictor x_j are formed by adding the fitted linear component in this predictor to the least squares residuals[3]. The partial residuals are then plotted against x_j

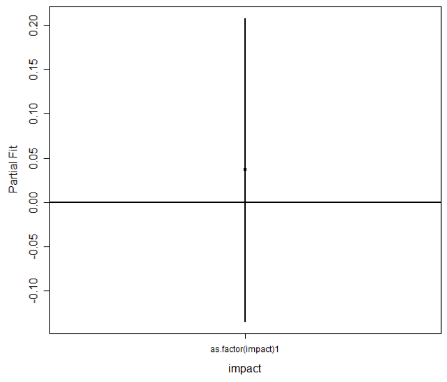
```
runPartialPlots(geeModel, data, factorlist = c("floodebb", "impact"),
    varlist = c("observationhour"))

[1] "Making partial plots"
```

[2]see 'component-plus-residual' plots in Fox and Weisberg (2002) for more details.
[3]$e_{ij} = e_i + b_j x_{ij}$

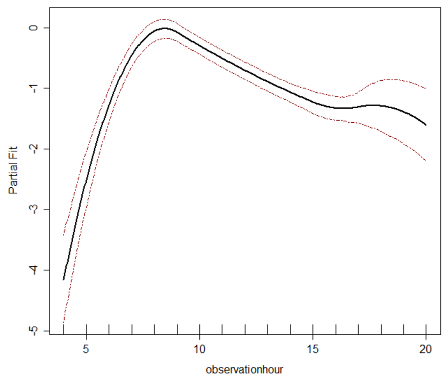
Assessing covariate relationships



(a)



(b)



(c)

Figure 3.9: Partial plots for (a) tide state, (b) impact and (c) observation hour.

Partial Plots

- Tide state: predictions for flood are not significantly different to ebb (baseline)
- There are, on average, more animals seen during slack tide than for the baseline (ebb).
- Impact: confirms the ANOVA p-value (seen earlier; $p >> 0.05$) and indicates no change in bird numbers post impact.

- Observation hour: a peaked non-linear relationship with maximum birds seen between 8 and 11am.
- Small confidence interval indicates a precisely estimated relationship.

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

3.5 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 measure are output on the observed vs fitted plot created using the `runDiagnostics` function.

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"
```

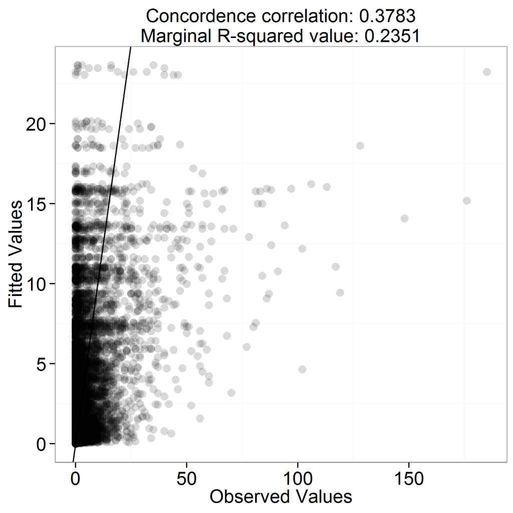


Figure 3.10: Diagnostic plots of observed vs fitted values, where the diagonal line indicates where data should lie for a perfect fit.

Observed vs fitted

- High observed values are under-predicted (Figure 3.10)
- Observed zeros tend to be over-predicted (as is likely given they cannot be under-predicted)
- Marginal R-squared and concordance correlation are low

- Conclusion: poor model fit

Fitted values vs scaled Pearsons residuals Plot

The plot gives an indication of the correct mean-variance relationship assumed under the model. We 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 overdispersed 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 (estimated by `geeglm`) for the latter
- Thus we plot Fitted values against scaled Pearsons residuals

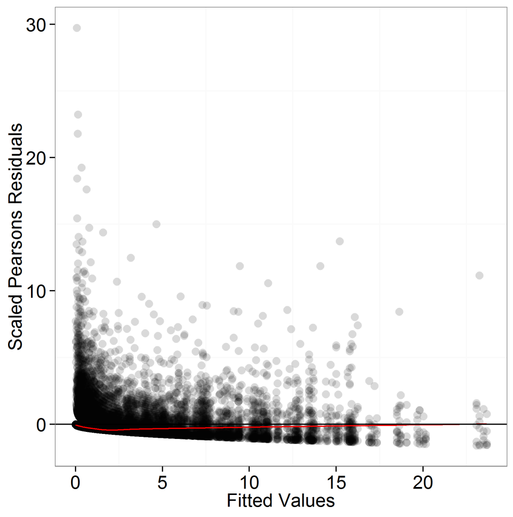


Figure 3.11: Diagnostic plot of fitted values vs scaled Pearsons residuals, where the red line is a locally weighted least squares regression to indicate pattern in the plot, which might otherwise be hidden due to over-plotting.

- Possible pattern in residuals but hard to tell due to overplotting (Figure 3.11)
- Locally weighted least squares regression line does not indicate an unusual pattern
- Conclusion: no issue with model assumption (mean-variance relationship)

3.5.1 Cumulative Residuals and runs profiles

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

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

Cumulative residuals and runs profiles for observation hour

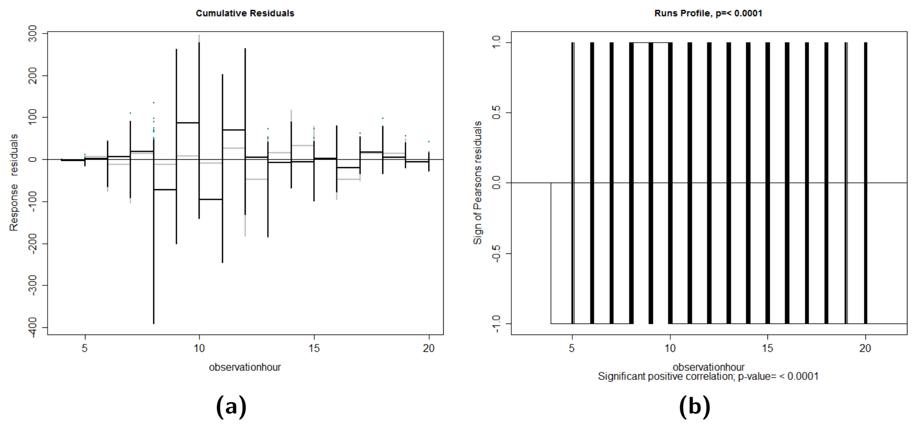


Figure 3.12: Cumulative residual plot (a) and runs profile (b) for residuals ordered by observation hour. The blue points are the residual values, the black line represents the cumulative residuals. The grey line in the background is what we would expect the cumulative residuals to be if observation hour was modelled correctly.

Ordered by **observation hour**

- No systematic over or under prediction (Figure 3.12)
- Variable modelled appropriately:
  - similar to expected relationship (grey line) and better than Figure 3.8
  - good mixing about the zero cumulative residual line
- Discrete nature of variable makes inference (visual) difficult
- Visually good mixing, but fewer runs than would be expected if residuals were random ($p << 0.05$)
- Significant positive correlation between residuals when ordered by observation hour

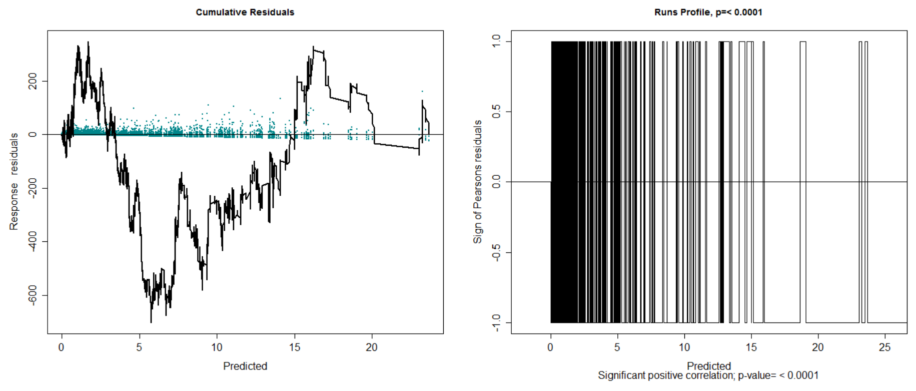


Figure 3.13: Cumulative residual plot (a) and runs profile (b) for residuals ordered by the predicted value. The blue points are the residual values and the black line represents the cumulative residuals.

Cumulative residuals and runs profiles ordered by predicted value

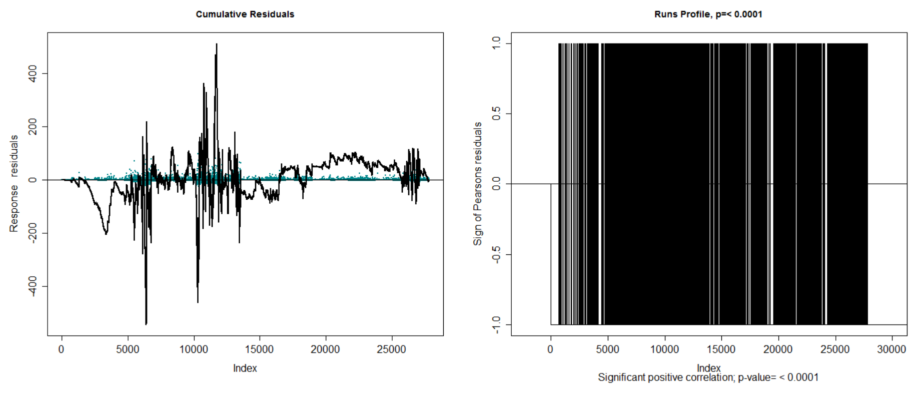
Ordered by **prediction** value

- Under prediction at predicted density < 3 (Figure 3.13)
- Over-predicted up to about 12 birds/km^2
- Thereafter some over/under prediction
- 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

Ordered by **index** (temporally)

- Lot of mixing about zero cumulative residual line (Figure 3.14)
- 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



(a) **(b)**

Figure 3.14: 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.

Cumulative residuals and Runs Profiles

- Despite the inclusion of temporal covariates (tidal state and impact) we still have some unmodelled correlation
- We model this correlation using GEEs
- p-values from ANOVA (fitted earlier to decide which covariates to keep) are reliable due to modelled correlation

3.5.2 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.

PRESS statistic

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

See section 2.9.2 for more details on COVRATIO and PRESS statistics.

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

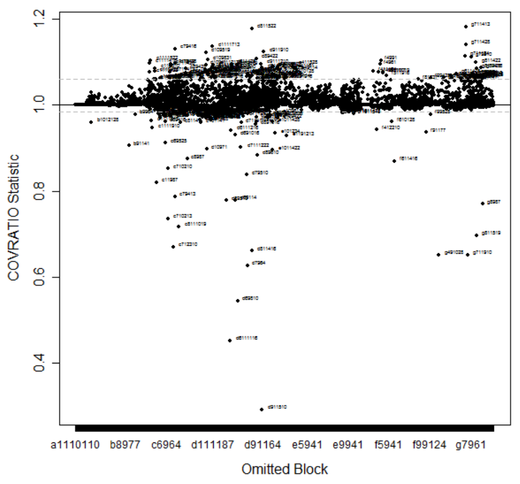


Figure 3.15: Plot of COVRATIO statistics; the dashed grey lines indicate the lower 2.5% and upper 97.5% quantiles of the statistics.

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, blocks `d611810` and `d6111116` are far away and result in an decrease in standard errors when removed (Figure 3.15)
- These blocks are both grid code `D6` in the year 2011 (first of two years after impact) and the former in month 8 and the 10^{th} day of the month.
- Both blocks have particularly high bird counts compared with most blocks and so adds to the variability in the data.

- Conclusion: Variability increases with higher counts for Poisson data and so it is fine for these blocks to remain.

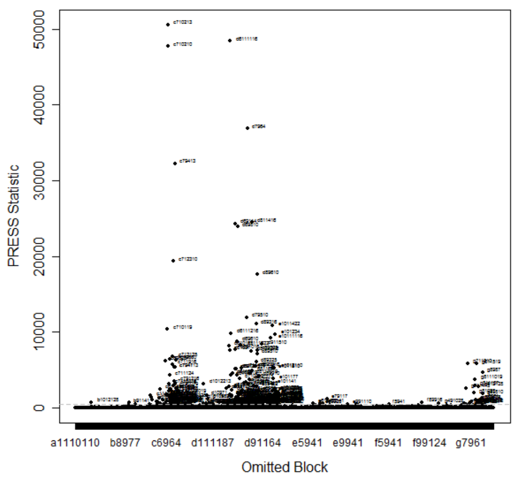


Figure 3.16: Plot of PRESS statistics; 95% of the statistics fall below the dashed grey lines. Labelled points on both plots are outside the grey dashed line(s) and are labelled with the identifier for the block that has been removed to create the statistic (not an observation number).

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
- Predictions are sensitive to several blocks: c710213 c710210 and d6111116 (Figure 3.16).
- As above, these blocks all contain very high bird counts

- Conclusion: Variability increases with higher counts for Poisson data and so it is fine for these blocks to remain.

3.5.3 Raw Residuals

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

- No real spatial pattern to the residuals (Figure 3.17)
- No systematic over/under prediction

- Conclusion: No spatial bias in model

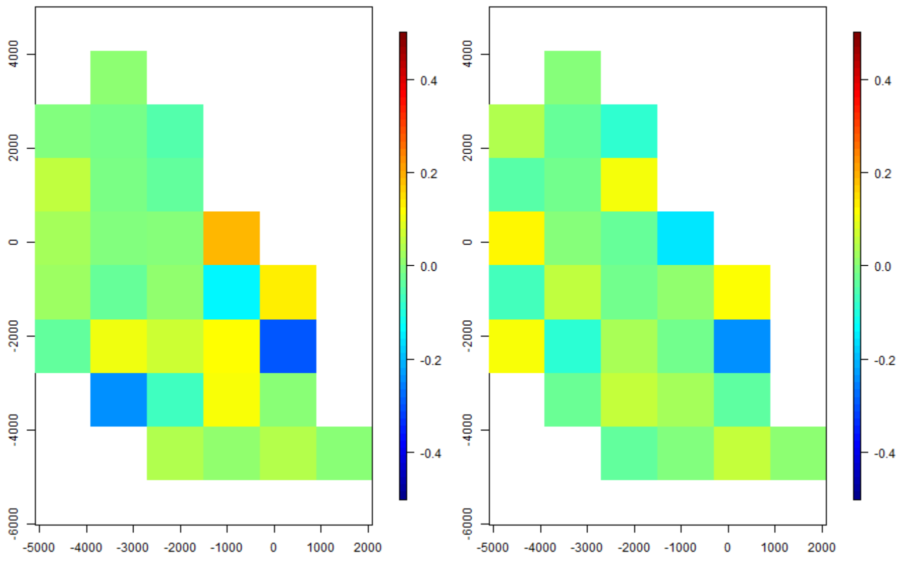


Figure 3.17: Raw residuals before impact (left) and after impact (right). These residuals are fitted values - observed values (mean* birds/km^2). * mean density because the predictions are for several observation hours.

```
# residual plot
resids <- fitted(geeModel) - data$birds
dims <- getPlotdimensions(data$x.pos, data$y.pos, 1000, 1000)

par(mfrow = c(1, 2), mar = c(3, 3, 3, 5))
quilt.plot(data$x.pos[data$impact == 0], data$y.pos[data$impact ==
    0], resids[data$impact == 0], asp = 1, ncol = dims[2], nrow = dims[1],
    zlim = c(-2.2, 2.2))
quilt.plot(data$x.pos[data$impact == 1], data$y.pos[data$impact ==
    1], resids[data$impact == 1], asp = 1, ncol = dims[2], nrow = dims[1],
    zlim = c(-2.2, 2.2))
```

How did we do?

Table 3.2: 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	-

3.6 Prediction and Inference

3.6.1 Data requirements for predicting

For making predictions, we require a data frame containing grid data with records for which to make predictions. This requires certain columns and records these 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 made for

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

```r
# loading the prediction grid data
data(ns.predict.data.re)
predictData <- ns.predict.data.re
head(predictData)
```

```
  x.pos y.pos     area floodebb observationhour GridCode Year DayOfMonth
1  1500 -4500 0.385253      EBB              12      a11    9         13
2  1500 -4500 0.385253    FLOOD               8      a11    9         16
3  1500 -4500 0.385253    FLOOD               9      a11    9         16
4  1500 -4500 0.385253    FLOOD              10      a11    9         16
5  1500 -4500 0.385253    FLOOD              11      a11    9         16
6  1500 -4500 0.385253    FLOOD              12      a11    9         16
  MonthOfYear impact        truth
1           3      0 0.0003871928
2           3      0 0.0006163433
3           3      0 0.0006064698
4           3      0 0.0005072918
5           3      0 0.0004220902
6           3      0 0.0003570533
```

Note: the `truth` column is an artefact of this being simulated data. Ordinarily the user would not know this.

3.6.2 Making predictions

We make predictions using our best fitting CReSS/SALSA model from the previous sections. This requires three steps:

1. Creating a matrix of distances from the 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 record in our prediction grid.

3.6.3 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`. From the model coefficients it is not obvious where the redistribution has occurred. We are going to investigate this by visualising the redistribution using our predictions.

The following is code for plotting the model predictions in Figure 3.18.

```
# plotting the predictions for before and after impact
# get the plot dimensions. We know each cell is 1000x1000m
dims<-getPlotdimensions(x.pos=predict.data.re$x.pos, predict.data.re$y.pos,
   segmentWidth=1000, segmentLength=1000)
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$impact==0], asp=1, nrow=dims[1], ncol=dims[2],
    zlim=c(0, maxlim))
quilt.plot(predictData$x.pos[predictData$impact==1],
    predictData$y.pos[predictData$impact==1], preds[predictData$impact==1],
    asp=1,nrow=dims[1], ncol=dims[2], zlim=c(0, maxlim))
```

Note that 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.
- The dimensions of the plot (`nrow` and `ncol`) determine the size of each grid cell. Since we know our grid to be 1000m by 1000m we can use `getPlotDimensions` to find the values for these arguments.

Conclusion

- Decline in bird density in the central eastern region
- Increase in density in the south of the study region
- 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 after the impact event

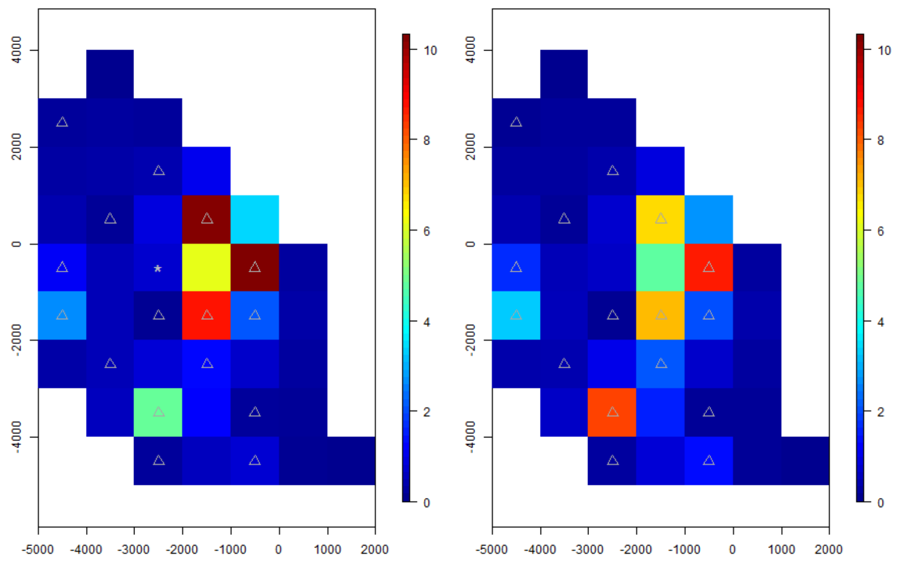


Figure 3.18: Predictions of bird density (mean birds/km^2) from the fitted model for before (left) and after (right) an impact event. The * indicated the location of the impact; the $\triangle$ indicate the knot locations.

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.

3.7 Bootstrap confidence intervals

Following the predictions from section 3.6.2, we now create replicate predictions and use these to construct percentile confidence intervals around each prediction grid cell. We run `B` iterations performing the following:

- Input best model from fitting to the original observed data.
- Use 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 the count model coefficients.

The `do.bootstrap.cress` function

This function performs the number of bootstrap iterations specified with the argument `B` where the fitting model is CReSS. The recommended number of bootstrap iterations is 999.

If `ddf.obj` is set to `NULL`, this function automatically applies the 'no distance sampling'-approach with the three steps described above.

Since the prediction object can be quite large, it is not returned but saved to file in the current working directory with the file name ‘predictionboot.RData’.

```
# do the bootstrap
do.bootstrap.cress(data, predictData, ddf.obj=NULL, baseModel, splineParams,
                   dists, B=250)
```

In the next step we 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.

An optional argument `quants` can be used to specify an interval other than 95%. The default is `quants = c(0.025, 0.975)`

```
# read in bootstrap predictions
load("predictionboot.RData")
# make percentile confidence intervals
cis <- makeBootCIs(bootPreds)
```

3.7.1 Visualising bootstrap confidence intervals

Figure 3.19 depicts the range of uncertainty in the spatial estimates of density. These can be contrasted with the point estimates found in Figure 3.18.

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.

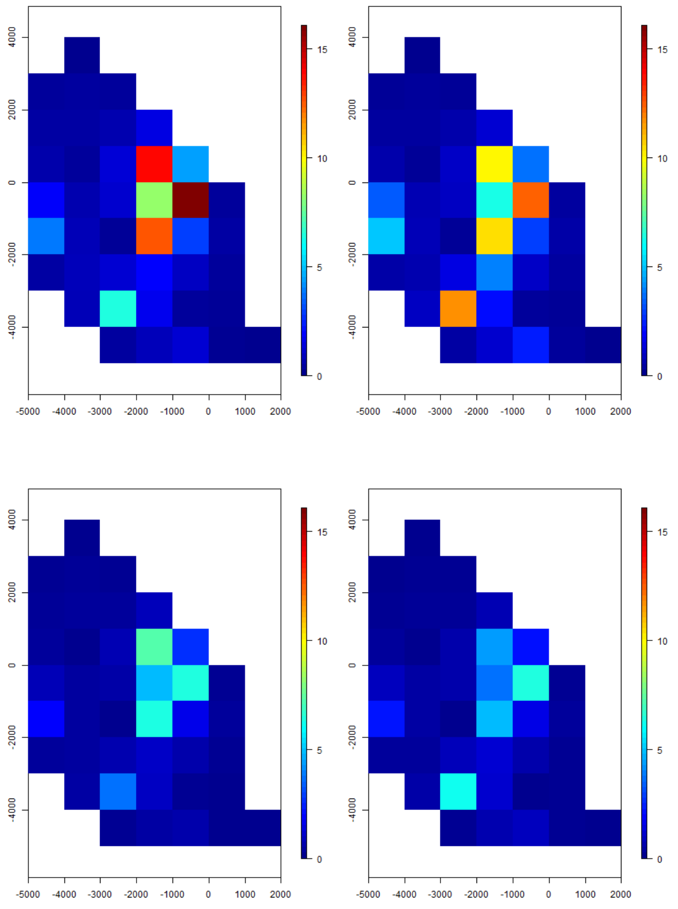


Figure 3.19: Upper (top) and lower (bottom) 95% confidence intervals of bird density (birds/km^2) from the fitted model from before (left) and after (right) impact.

Conclusion

- Lower and upper intervals before impact (left) show a higher density of birds in the central eastern region.

- After impact (right), both lower and upper intervals show a decline in density in the central eastern region and an increase in the area to the south.
- It is difficult to see where in the study region any significant differences may have occurred.

3.8 Identifying differences

In this section we estimate the differences in predictions for each corresponding pair of records from our prediction grid. The first half of our prediction grid 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 for the difference 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, ])
```

3.8.1 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 differences are added to the same plot using the '$\circ$' and '$+$' symbols.

```
# The median for each after - before difference
mediandiff <- differences$mediandiff
# The marker for each after - before difference:
positive ('1') and negative ('-1') significant differences
marker <- differences$significanceMarker
par(mfrow = c(1, 1))
quilt.plot(predictData$x.pos[predictData$impact == 0],
    predictData$y.pos[predictData$impact ==  0],
    mediandiff, asp = 1, nrow = 7, ncol = 9)
# 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=2)
points(predictData$x.pos[predictData$impact==0][marker==(-1)],
    predictData$y.pos[predictData$impact==0][marker==(-1)], col="darkgrey",
    cex=3)
points(0,0,pch=20, col="grey",, cex=3)
points(data$x.pos[which(data$GridCode=='e7')[1]],
    data$y.pos[which(data$GridCode=='e7')[1]],cex=2, pch="*", lwd=2,
    col="grey")
```

Conclusion

- There was a significant decline in animals around the impact site and closest to the cliff-top observer (Figure 3.20).
- There was also a significant increase in animals in the south of the study area.

3.9 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.

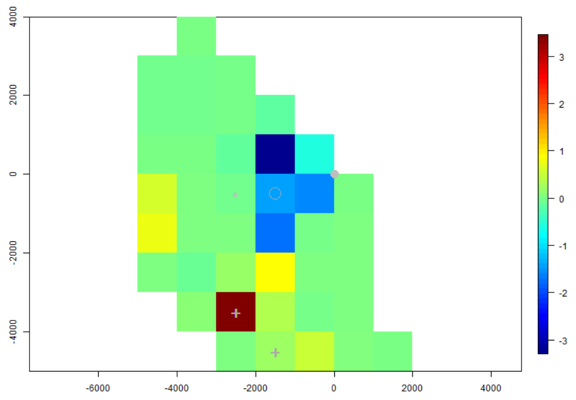


Figure 3.20: Mean differences in predicted bird density (birds/km^2) before and after impact. Positive values indicate more birds post impact and negative values fewer birds post impact. Significant differences were calculated using percentile confidence intervals: '+' indicates a significant positive difference and '○' a significant negative one. The grey star is the site of the impact event.

3.9.1 Overdispersion and correlation

We created data that were both overdispersed and positively correlated within any grid cell-day (i.e. observed counts from the same grid cell within the same day).

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

Conclusion

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

3.9.2 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 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 method was capable of correctly identifying the redistribution effect and reallocated birds to the correct locations despite highly correlated and overdispersed data.

Chapter 4

Tips and Tricks for analyses in R related to the MRSea Package

4.1 Distance Tips and Tricks

4.1.1 Analysing binned distance data

If we decide to bin the exact distance data into intervals and fit a detection function to interval distance data we can do this by first adding the cut points of the respective interval for each detection to the data using the function `which.bin` and then setting the argument `binned` within `meta.data` to `TRUE`. Note that for the new version of `mrds`, the cut points of the intervals also need to be defined with the argument `breaks` when running the function.

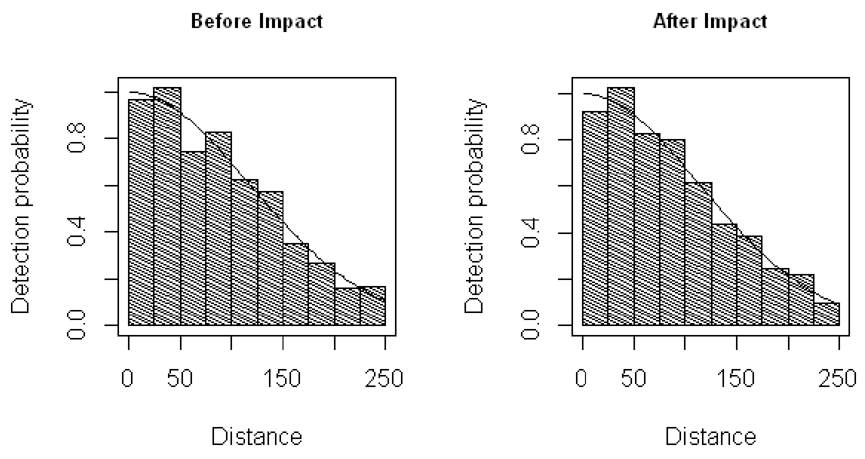
```
data(dis.data.re)
dis.data.re<-which.bin(dis.data.re, cutpoints=c(0,50,100,150,200,250))
result.int<-ddf(dsmodel=~mcds(key="hn",formula=~1),data=dis.data.re,
    method="ds", meta.data=list(width=250, binned=T,
     breaks=c(0,50,100,150,200,250)))
```

The AIC value from this model may not be compared with the AIC from a model which uses exact distance data from the offshore worked example due to the fact that the data have changed by binning the distances.

4.1.2 Plotting side by side detection function plots for different factor covariate levels

We begin with the example from the worked example above where we fit a covariate model to the exact distance data. We use the following `ddf` call which includes the two level factor covariate `impact` in the detection function model. We then plot the two detection functions for the respective levels side by side:

```
data(dis.data.re)
result.imp<-ddf(dsmodel=~mcds(key="hn", formula=~1+impact),
     data=dis.data, method="ds", meta.data=list(width=250))
par(mfrow=c(1,2))
plot(result.imp, showpoints=F, breaks=seq(0,250,25), subset=impact==0,
    main="Before Impact")
plot(result.imp, showpoints=F, breaks=seq(0,250,25), subset=impact==1,
    main="After Impact")
```



4.2 Spatial Modelling Tips and Tricks

4.2.1 General

Use of `grep` function

the `grep` function is used in the following `MRSea` functions: `getPvalues`, `plotCumRes`, `plotRunsProfile`, `runPartialPlots` and `runSALSA1D` . This will cause the user some issues if there are nested covariate names in the model.

For example:

- wavefront and wavefrontheight are not suitable as wavefront is nested,

- but wavefront and waveheight are fine.

Save option for plotting

All of the plotting functions in MRSea have an option for saving a 'png' image straight to your working directory. See the help files of specific functions for details.

For example:

```
plotCumRes(fullModel, varlist, splineParams, save=T, label='_glmk1')
```

This code will save cumulative residual plots for residuals ordered by all variables specified in `varlist` and by predicted value and temporal order into the working directory. The label allows the user to make the file name specific to the model. In this case, one of the plots is labelled 'CumRes_observationhour_glmk1.png'.

4.2.2 SALSA

SALSA1D knots

When smoothing biological data many people will restrict the degrees of freedom to reflect the unlikely nature of more complex smooths. For this reason, and our experience gained from using SALSA on biological data, we recommend that the maximum number of knots for a one-dimensional smooth is between 3 and 5.

SALSA2D knots

Requirements for the SALSA2D knot grid:

- `(k x 2)` data frame
- regular grid covering whole study region
  - `c(NA, NA)` in rows where the knot location is invalid
  - Invalid is on land
  - or very far away from data

Make a grid that covers the study region

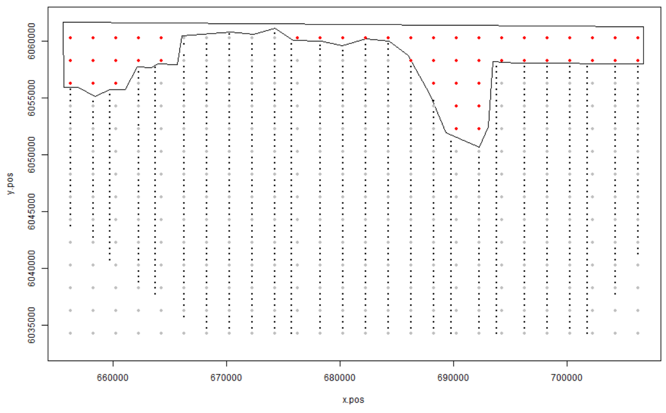


Figure 4.1: Figure identifying knot locations inside a boundary (red). The grey dots are knot locations outside the boundary and the black dots are the data locations.

```
# load data
data(dis.data.re)
data(predict.data.re)

# sequence of x and y values spaced by 2000 (roughly the spacing of the
     transects)
x<-seq(min(dis.data.re$x.pos), max(dis.data.re$x.pos), by=2000)
y<-seq(min(dis.data.re$y.pos), max(dis.data.re$y.pos), by=2000)
grid<-expand.grid(x.pos=x, y.pos=y)

plot(dis.data.re$x.pos, dis.data.re$y.pos, pch=20, cex=0.5, asp=1,
     xlab='x.pos', ylab='y.pos')
points(grid, pch=20, col='red')

# draw round boundary as no boundary polygon available
bnd<-locator()
# i have drawn the coastline to the top for the boundary
polymap(bnd, add=T)
```

First find the knot points that are on land (Figure 4.1).

```
marker<-rep(0, nrow(grid))
id<-which(inout(grid, bnd)==TRUE)
# place a 1 in the marker vector for knots we do not want
marker[id]<-1

# make knotgrid object and make unwanted knot locations NA
knotgrid<-grid
knotgrid[marker==1,]<-c(NA,NA)
```

There are still knots in the open water area to the bottom of the survey. The Euclidean distance between knots and data is used to identify knots

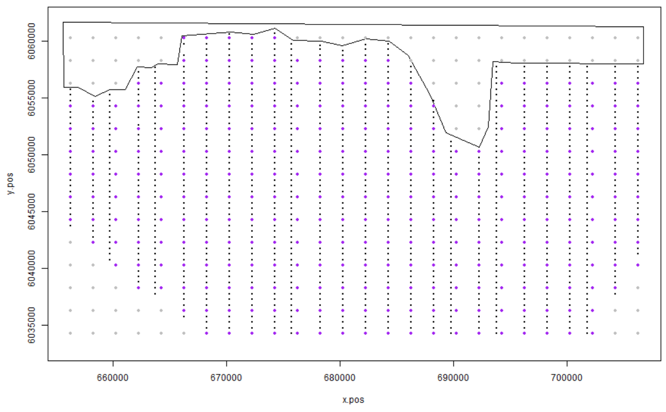


Figure 4.2: Figure identifying legal (purple) and illegal (grey) knot locations. The black dots are the data locations.

that are located a specified distance from any data. These knot locations are identified as illegal by placing an NA row in the knot grid object. The final knot grid is given in Figure 4.2.

```
distcheck<-makeDists(datacoords=cbind(dis.data.re$x.pos, dis.data.re$y.pos),
     knotcoords=na.omit(knotgrid))

naid<-which(is.na(knotgrid[,1]))
# knots further tham 1km from data are identified with a 1
marker[-naid][which(apply(distcheck$dataDist, 2, min)>1000)]<-1
knotgrid[marker==1,]<-c(NA,NA)
```

gap **parameter**

The gap parameter was discussed briefly in section 2.8.3 (page 28) to prevent adjacent knots when there was no data between transects. Whilst the example was for SALSA2D, gap is also an argument for SALSA1D. Occasionally, knots may be placed too close causing prediction error. The user may specify a gap in the `salsa1dlist` or `salsa2dlist` arguments to prevent this. The specification of a gap may also reduce computational time, since there are fewer legal moves for a knot.

Interaction Terms

More than one interaction term may be included, however the code for SALSA2D may only be run on one interaction at a time. Furthermore, the knot locations selected for the spatial smooth in one interaction term will also be used for any other interaction terms.

4.2.3 SALSA1D with more than one smooth covariate

The examples presented in chapters 2 and 3 fitted only one smooth covariate. The following code describes how two or more smooth covariates may be added to the model using `runSALSA1D` to select the smoothness (knot locations) for each.

There are two ways to run models with more than one smooth covariate:

1. No removal of terms from the model. Useful if SALSA1D is only required for knot selection and the user decides on variable selection.
2. Variable selection included. The resulting model may have terms removed, be linear or smooth.

Both methods are set up in the same way:

Start by building the spline parameters object and specifying more than one covariate in `varlist`.

```
# make splineParams object
splineParams<-makesplineParams(data=data, varlist=c('observationhour',
    'DayOfMonth'), predictionData=predictData)
str(splineParams)

List of 3
 $ : list()
 $ :List of 5
  ..$ covar      : chr "observationhour"
  ..$ explanatory: int [1:27798] 12 8 9 10 11 12 13 14 15 8 ...
  ..$ knots      : num 12
  ..$ bd         : int [1:2] 4 20
  ..$ degree     : num 2
 $ :List of 5
  ..$ covar      : chr "DayOfMonth"
  ..$ explanatory: int [1:27798] 13 16 16 16 16 16 19 19 19 25 ...
  ..$ knots      : num 13.9
  ..$ bd         : int [1:2] 1 31
  ..$ degree     : num 2
```

To fit a simple model with the default knot values (one knot at the mean) simply repeat the `bs()` term, used in the previous examples, one for each covariate.

```r
# fitting a model with two smooth terms
fullModel<-glm(birds ~ as.factor(floodebb) + as.factor(impact) +
    bs(observationhour, knots=splineParams[[2]]$knots) +
    bs(DayOfMonth, knots=splineParams[[3]]$knots) +
    x.pos + y.pos, family=quasipoisson, data=data)
```

To run SALSA with more than one covariate there are changes in the `salsa1dlist` object and in the `varlist` argument of `runSALSA1D`. Each covariate must be specified in `varlist` and `minKnots_1d`, `maxKnots_1d`, `startKnots_1d`, `degree` and `gaps` must all be vectors the same length as `varlist`. This allows different parameters for each of the covariates.

Below is an example of the two ways to run SALSA when you have more than one smooth covariate:

1. No removal of terms

Each covariate is added to the model as a smooth term with the knots specified in the spline parameter object (default knot at the mean if from `makesplineParams`) and then SALSA checks each one in turn for an improvement in knot number and location. If there is an improvement then the spline parameter object is updated with the new number and location.

Note: There must be a `foldid` column in the data so that cross-validation can be used for selection.

```r
data$response<- data$birds
# set initial model without the spline terms in there
initialModel <- glm(response ~ as.factor(floodebb) + as.factor(impact) +
    offset(log(area)), family = "quasipoisson", data = data)

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

# run SALSA
salsa1dOutput <- runSALSA1D(initialModel, salsa1dlist, varlist=
    c("observationhour", "DayOfMonth"), factorlist=c("floodebb", "impact"),
    predictData,  splineParams=splineParams)
```

The structure of the output is the same as before but with more information in `modelFits1D` regarding what terms were fitted and what knots were chosen.

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

List of 4
 $ bestModel   :List of 30
  ..- attr(*, "class")= chr [1:2] "glm" "lm"
 $ modelFits1D :List of 3
 $ splineParams:List of 3
 $ fitStat     : num 31627
```

We can also look directly at the SALSA selected knots by looking at the spline parameter object. In this example, three knots were chosen for observation hour, whilst the knot location for day of the month was unchanged from the default value (the mean). One must now consider whether or not day of the month should be a model covariate at all by fitting a model with and without it or using the `runSALSA1D_withremoval` code instead (below).

```
# knots chosen for observation hour
salsa1dOutput$splineParams[[2]]$knots

[1]  9 15 17


# knots chosen for day of month
salsa1dOutput$splineParams[[3]]$knots

[1]  13.86
```

2. Removal of terms

Each covariate is added to the model as a smooth term with the knots specified in the spline parameter object (default knot at the mean if from `makesplineParams`) and then SALSA checks each one in turn for an improvement in knot number and location. If there is an improvement then the spline parameter object is updated with the new number and location. If there is no improvement over the single mean knot model then the term is tested as linear or with the term removed. The model with the best CV score is returned as `salsa1dOutput$bestModel` (in the example below).

```
data$response<- data$birds
data$blockid<-pastedata$transect.id, data$season, data$impact, sep="")
```

```r
data$foldid<-getCVids(data, folds=5, block='blockid')
# set initial model without the spline terms in there
initialModel <- glm(response ~ as.factor(floodebb) + as.factor(impact) +
    offset(log(area)), family = "quasipoisson", data = data)

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

# run SALSA
salsa1dOutput <- runSALSA1D_withremoval(initialModel, salsa1dlist, varlist=
    c("observationhour", "DayOfMonth"), factorlist=c("floodebb", "impact"),
    predictData,  splineParams=splineParams)
```

The structure of the output is the same as before but with more information in `modelFits1D` regarding what terms were kept and if knots were chosen and an additional entry, `keptvarlist`, listing the covariates retained in the best model.

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

List of 5
 $ bestModel   :List of 30
  ..- attr(*, "class")= chr [1:2] "glm" "lm"
 $ modelFits1D :List of 3
 $ splineParams:List of 3
 $ fitStat     : num 31951
 $ keptvarlist : chr "observationhour"
```

The model selected has removed DayOfMonth and selects 3 knots for observationhour. To see what happened during the process look at the `modelFits1D` object in the output. The first step (list entry 1) fitted a model with the knot values in `splineParams`. In this case that is one knot at the mean for each covariate.

```
> salsa1dOutput$modelFits1D
[[1]]
[[1]]$term
[1] "startmodel"

[[1]]$kept
NULL

[[1]]$basemodelformula
glm(formula = response ~ as.factor(floodebb) + as.factor(impact) +
   offset(log(area)) + bs(observationhour, knots = splineParams[[2]]$knots,
   degree = splineParams[[2]]$degree, Boundary.knots = splineParams[[2]]$bd)
   + bs(DayOfMonth, knots = splineParams[[3]]$knots, degree =
   splineParams[[3]]$degree, Boundary.knots = splineParams[[3]]$bd),
   family = quasipoisson,  data = data)

[[1]]$knotsSelected
NULL

[[1]]$tempfits
         CV     fitStat
   33.89111 32564.32819
```

The second step (list entry 2) investigates new knot locations and numbers for observationhour. Three knots were selected by SALSA and kept in the model due to an improvement in CV score from the step above (33.89 to 33.83). The `baseModelFits` (model returned from this step) and `modelfits` (model with knots chosen by SALSA) are identical as the new knots are retained.

```
[[2]]
[[2]]$term
[1] "bs(observationhour, knots = splineParams[[2]]$knots,
      degree=splineParams[[2]]$degree, Boundary.knots=splineParams[[2]]$bd)"

[[2]]$kept
[1] "YES - new knots"

[[2]]$basemodelformula
glm(formula = response ~ as.factor(floodebb) + as.factor(impact) +
    bs(DayOfMonth, knots = splineParams[[3]]$knots, degree =
    splineParams[[3]]$degree, Boundary.knots = splineParams[[3]]$bd)
   + bs(observationhour, knots = splineParams[[2]]$knots, degree =
    splineParams[[2]]$degree, Boundary.knots = splineParams[[2]]$bd)
   + offset(log(area)), family = quasipoisson, data = data)

[[2]]$knotsSelected
[1]  9 15 16

[[2]]$baseModelFits
         CV     fitStat
   33.83714 31627.38018

[[2]]$modelfits
         CV     fitStat
   33.83714 31627.38018
```

The last step (list entry 3 in this example) investigates new knot locations and numbers for DayOfMonth. A smooth term is not retained as the CV score is worse for the selected knots (33.84) and improves over the model from the previous step when the term is removed (33.83 to 33.82).

```
[[3]]
```

```
[[3]]$term
[1] "bs(DayOfMonth, knots = splineParams[[3]]$knots,
    degree=splineParams[[3]]$degree, Boundary.knots=splineParams[[3]]$bd)"

[[3]]$kept
[1] "NO"

[[3]]$basemodelformula
glm(formula = response ~ as.factor(floodebb) + as.factor(impact) +
    bs(observationhour, knots = splineParams[[2]]$knots, degree =
    splineParams[[2]]$degree, Boundary.knots = splineParams[[2]]$bd)
   + offset(log(area)), family = quasipoisson, data = data)

[[3]]$knotsSelected
[1] "NA"

[[3]]$baseModelFits
         CV     fitStat
   33.82023 31950.97759

[[3]]$modelfits
        CV    fitStat
   33.8485 32397.2881
```

The splineParams object has been updated and also forms part of the output.

4.2.4 Bootstrapping

Bootstrap data created outside the function

If the detection function process is complicated owing to multiple data sources, for example, the bootstrap data from the detection function process may be created outside the `do.bootstrap.cress` function. Use the argument `nhats` to specify a matrix where each column is a replicate and each row is a data point, which must be in the same order as the original data. The specification for `B` must not be greater than the number of columns in `nhats`.

Bootstrapping in parallel

The argument `nCores` is set to 1 by default but if you are using a Windows machine, increasing the number of cores used (using `nCores`) will speed up the bootstrap process. Note that the package `parallel` is required for this.

See `?do.bootstrap.cress` for more information.

Bibliography

Fox, J. and Weisberg, S. (2002). *An R companion to applied regression*. Sage, 2^{nd} edition.

Hardin, J. and Hilbe, J. (2002). *Generalized Estimating Equations*. Chapman & Hall/CRC.

Scott-Hayward, L., Mackenzie, M. L., Donovan, C. R., Walker, C. G., and Ashe, E. (2013). Complex Region Spatial Smoother (CReSS). *Journal of Computational and Graphical Statistics* .

Walker, C., Mackenzie, M., Donovan, C., and O'Sullivan, M. (2010). SALSA - a Spatially Adaptive Local Smoothing Algorithm. *Journal of Statistical Computation and Simulation* **81,** 179–191.