Vignette for the `MRSea` Package (v0.2.0)

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

February 6, 2015

Lindesay Scott-Hayward
Cornelia Oedekoven
Monique Mackenzie
Cameron Walker

This vignette constitutes work carried out at the Centre for Research into Ecological and Environmental Modelling (CREEM) at the University of St. Andrews.

Please reference this document as: Scott-Hayward, L.A.S., Oedekoven, C.S., Mackenzie, M.L. and Walker, C.G. (2015). Vignette for the MRSea Package v0.2.0: Statistical Modelling of bird and cetacean distributions in offshore renewables development areas. Centre for Research into Ecological and Environmental Modelling, University of St Andrews.

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 vignette gives an updated example of the code for version 0.2.0. For additional information regarding methods, see Mackenzie, et al. (2013) and Scott-Hayward, et al. (2013). The user should be familiar with generalised linear models and their assumptions and model selection. The `MRSea` package primarily allows spatially adaptive model selection for both one and two dimensional covariates using the functions `runSALSA1D_withremoval` and `runSALSA2D`, which implement the methods of Walker, et al. (2010) and Scott-Hayward, et al. (2013). Other functions include diagnostics (to assess residual correlation: `runACF`, smooth relationships: `runPartialPlots` and model selection (ANOVA) for a Generalised Estimating Equation used when residual correlation is present: `getPvalues`) and inference (`do.bootstrap.cress`).

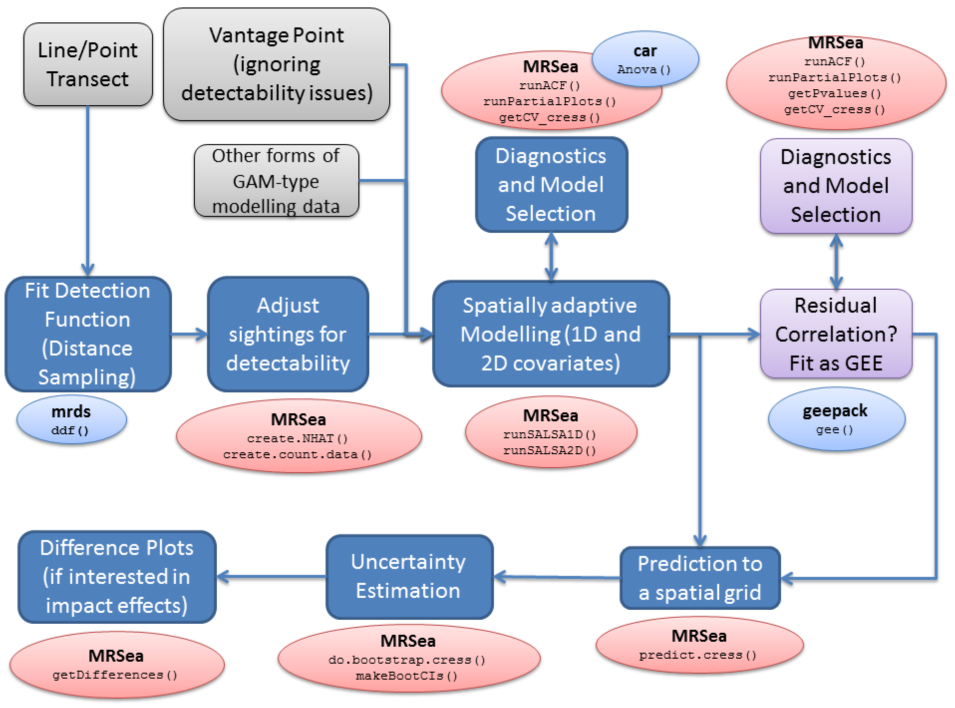

Line/Point Transect
Vantage Point (ignoring detectability issues)
Other forms of GAM-type modelling data
MRSea runACF() runPartialPlots() getCV_cress()
car Anova()
Diagnostics and Model Selection
MRSea runACF() runPartialPlots() getPvalues() getCV_cress()
Diagnostics and Model Selection
Fit Detection Function (Distance Sampling)
mrds ddf()
Adjust sightings for detectability
MRSea create.NHAT() create.count.data()
Spatially adaptive Modelling (1D and 2D covariates)
MRSea runSALSA1D() runSALSA2D()
Residual Correlation? Fit as GEE
geepack gee()
Difference Plots (if interested in impact effects)
MRSea getDifferences()
Uncertainty Estimation
MRSea do.bootstrap.cress() makeBootCIs()
Prediction to a spatial grid
MRSea predict.cress()

Figure 1: Example of the modelling process using MRSea. Packages with functions to run certain parts are given in oval boxes. To complete the modelling process, other packages may be used at certain stages Hojsgaard, et al. (2006); Laake, et al. (2014); Fox, et al. (2011). These are coded light blue, whilst MRSea functions are in red. GEE (far right), stands for Generalised Estimating Equations.

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.aspxhttp://creem2.st-and.ac.uk/software.aspx. The manual and this document use version 0.2.0 of MRSea.

2 Distance sampling using the `mrds` library

1. Load data and fit detection function (Distance Sampling)

```
require(MRSea)
# we will use the dataset with a known re-distribution of animals
data(dis.data.re)
dis.data<-dis.data.re
require(mrds) # distance sampling package
result <- ddf(dsmodel=~mcds(key="hn", formula=~1),
              data = dis.data, method="ds",
              meta.data=list(width=250))
```

2. Adjust sightings for detectability

```
# create.NHAT and create.count.data are MRSea functions to adjust the
# sightings for the detection function estimated above.
dis.data <- create.NHAT(dis.data,result)
count.data <- create.count.data(dis.data)
```

3. Try a simple model

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

4. Try a model with a smooth term for depth

```
#knots <- mean(depth) # must be specified as an object
require(splines)
fullModel <- glm(response ~ as.factor(season) + as.factor(impact) +
                   bs(depth, knots = mean(depth)) + x.pos + y.pos,
                 family = poisson,data = data)
```

5. SALSA1D requires that `foldid` is a column in the data set so that k-fold Cross-Validation (CV) may be used. The user may specify how many folds (5 or 10 is usual) and whether or not the data has a blocking structure. If the data are correlated then when selecting folds for the CV, the blocks must not be split up.

   For correlated data:

```
# for correlated data:
data$blockid <- paste(data$transect.id, data$season, data$impact,sep = "")
data$foldid <- getCVids(data = data, folds = 5, block = 'blockid')
```

or uncorrelated data:

```
# for uncorrelated data@
data$foldid<- getCVids(data=data, folds=5)
```

3 Selection of 1D Covariates

Run SALSA1D to select what covariates are included and whether or not they are smooth. SALSA selects the smoothness of each term (number and location of knots) and CV is used to choose between the best smooth term, a linear term or no term at all. To not allow the removal process the user may set `removal = FALSE` as a parameter in the function `runSALSA1D_withremoval`.

6. Specify the parameters required:

```
salsa1dlist <- list(fitnessMeasure = "AICh", minKnots_1d = 2,maxKnots_1d = 5,
                    startKnots_1d = 1, degree = 2, maxIterations = 10,
                    gaps = c(1))
```

7. If you wish to make predictions once the model is fitted, then a prediction grid should be created and specified. This is because the splines fitted here (B-splines) are unable to make predictions outside of the range they were created. For example, if the data range for depth is smaller than the range of depths in the prediction data, predictions cannot be made.

```
data(predict.data.re)  # contains predict.data
# This is a spatial grid for making predictions.  All covariates in
# final model must be in this data frame and the naming must be the
# same as for the data
predictData <- predict.data.re
range(data$depth)

## [1] -28.608  -0.411

range(predictData$depth)

## [1] -28.61  -0.20
```

Here the range of the predictions is slightly wider than the range of the data, so we will specify `predictData` when running SALSA.

8. Set up the initial model with factor covariates and the offset term (if required), and run SALSA.

```
initialModel <- glm(response ~ as.factor(season) + as.factor(impact)
                    + offset(log(area)), family = "quasipoisson",
                    data = data)
```

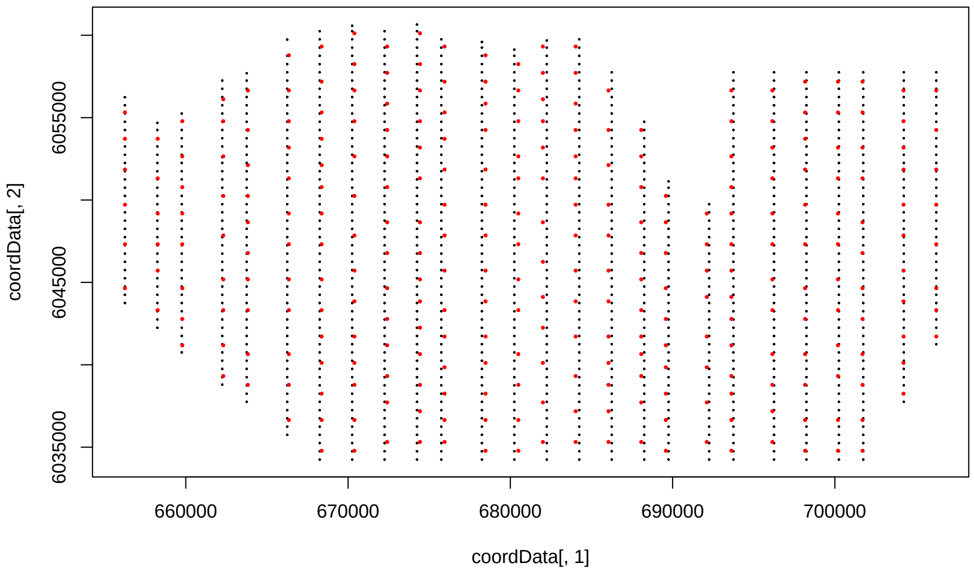
```
# run SALSA
salsa1dOutput <- runSALSA1D_withremoval(initialModel, salsa1dlist, c("depth"),
                      predictionData=predictData, datain=data, removal=TRUE)
```

```
# How many knots were chosen for depth?
salsa1dOutput$splineParams[[2]]$knots

## [1] -16.62

splineParams<-salsa1dOutput$splineParams
# ~~~~~~~~~~~~~~~~~~~~~~~
```

4 Selection of flexibility for 2D smooth term

9. Create a grid of knots that will be used as possible knot locations. This may take while and could be different every time you run it so I suggest saving the knotgrid as a file.

```
knotgrid<- getKnotgrid(coordData = cbind(data$x.pos, data$y.pos))
#
# write.csv(knotgrid, file='knotgrid_fullanalysis.csv', row.names=F)
# ~~~~~~~~~~~~~~~~~~~~~~~
```



The black points in the figure are the data and the red points, the candidate knot locations.

10. Set up parameters for SALSA2D. Distance matrices and the range parameter. The default setting of 10 radii should be fine for general use. Choose a fit statistic, min, max and start knots.

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

r_seq <- getRadiiChoices(numberofradii=10, distMatrix=distMats$dataDist)

# ~~~~~~~~~~~~~~~~~~~~~~~

# make parameter set for running salsa2d
salsa2dlist<-list(fitnessMeasure = 'AICh', knotgrid = knotgrid,
                  knotdim=c(100,100), startKnots=10, minKnots=4,
                  maxKnots=12, r_seq=r_seq, gap=4000,
                  interactionTerm="as.factor(impact)")
```

11. Run SALSA2D to find the appropriate number and location of knots for the 2D smooth term of `x.pos` and `y.pos`.

```
salsa2dOutput_k6<-runSALSA2D(salsa1dOutput$bestModel, salsa2dlist,
                             d2k=distMats$dataDist, k2k=distMats$knotDist,
                             splineParams=splineParams)
```

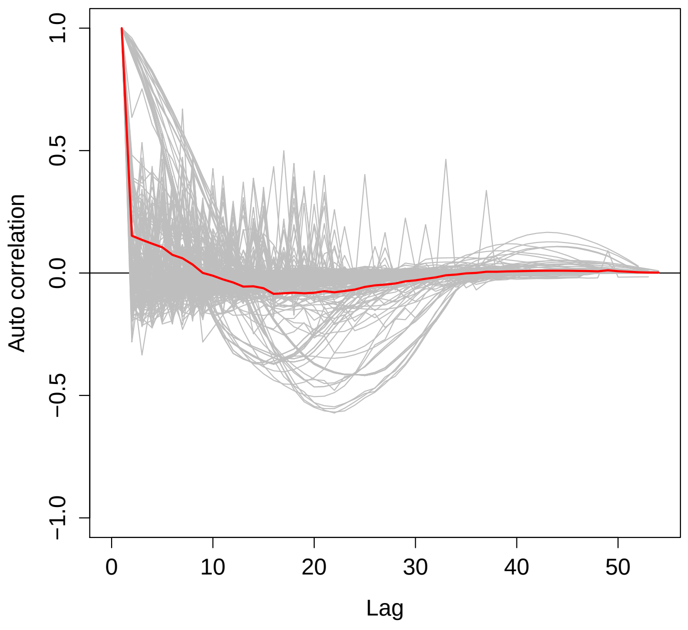


Figure 2: ACF plot showing correlation in each block (grey lines), and the mean correlation by lag across blocks (red line).

12. Update relevent SALSA parameters in your workspace (this is important for updating the model or making predictions).

```r
splineParams<-salsa2dOutput_k6$splineParams
# specify parameters for local radial function:
radiusIndices <- splineParams[[1]]$radiusIndices
dists <- splineParams[[1]]$dist
radii <- splineParams[[1]]$radii
aR <- splineParams[[1]]$invInd[splineParams[[1]]$knotPos]
```

13. Are the residuals correlated? Make a suitable blocking structure, within which residuals are expected to be correlated but between which they are independent. Use `runACF` to assess the blocking structure.

```r
data$blockid<-paste(data$transect.id, data$season, data$impact, sep='')
runACF(block = data$blockid, model = salsa2dOutput_k6$bestModel)
```

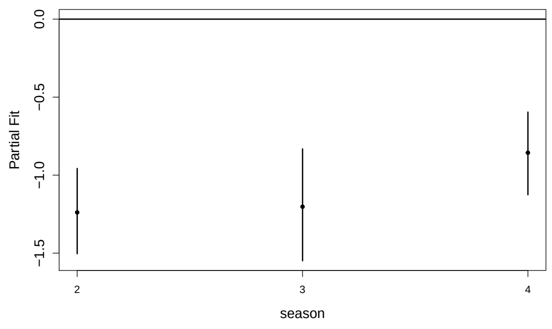
Update the model to run a GEE to account for the correlation in the residuals (seen in the ACF plot). The point estimates do not change as an independent working correlation structure is used, however, the standard errors are now appropriate and may be used for inference.

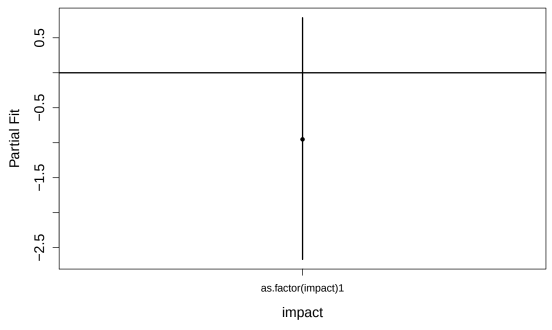
```
# Re-fit the chosen model as a GEE (based on SALSA knot placement) and
# GEE p-values
require(geepack)
geeModel<- geeglm(formula(salsa2dOutput_k6$bestModel), data=data,
                  family=poisson, id=blockid)
```

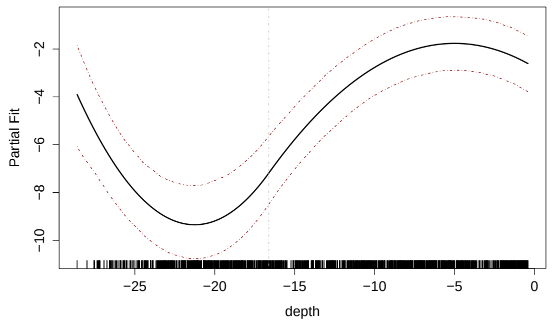
14. Check for model selection

```
getPvalues(model = geeModel, varlist = 'depth',
           factorlist = c('season', 'impact'))

## [1] "Getting marginal p-values"
##                  Variable  p-value
## 1                  season  <0.0001
## 2                  impact 0.523364
## 3                   depth  <0.0001
## 4          s(x.pos, y.pos)  <0.0001
## 5 s(x.pos, y.pos):impact  0.00323
```

```
par(mfrow=c(2,2))
runPartialPlots(model = geeModel, data = data, factorlist =
                  c('season', 'impact'), varlist = 'depth', showKnots = T)

## [1] "Making partial plots"
```





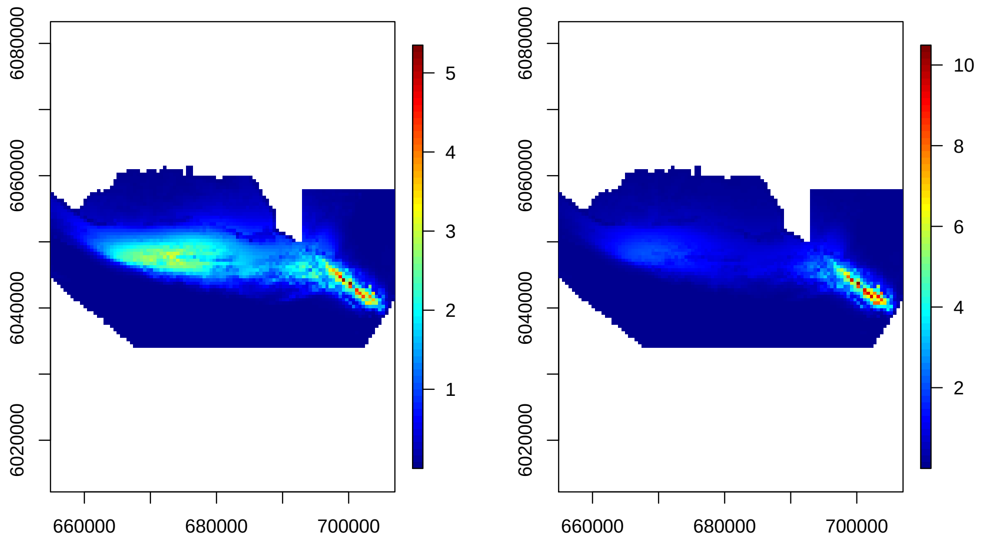


5 Making Predictions

```
dists<-makeDists(cbind(predictData$x.pos, predictData$y.pos),
                 na.omit(knotgrid),knotmat=FALSE)$dataDist

# make predictions on response scale
preds<-predict.cress(predictData, splineParams, dists, geeModel)
```

Plotting the predictions pre and post impact:

```
par(mfrow=c(1,2))
quilt.plot(predictData$x.pos[predictData$impact==0],
           predictData$y.pos[predictData$impact==0],
           preds[predictData$impact==0], nrow=104, ncol=55, asp=1)

quilt.plot(predictData$x.pos[predictData$impact==1],
           predictData$y.pos[predictData$impact==1],
           preds[predictData$impact==1], nrow=104, ncol=55, asp=1)
```



6 Bootstrapped Confidence Intervals and Difference Surfaces

15. The coding in this section has not changed from the original user guide.

16. Bootstrap to include parameter estimation uncertainty in the detection function and parameter estimation in the spatial model. (Note: If no detection function estimated, then the bootstrap is just on the parameters of the spatial model.)

```
dis.data$seasonimpact <- paste(dis.data$season, dis.data$impact)
do.bootstrap.cress(dis.data, predict.data=predictData, result, geeModel,
                   splineParams, dists, resample = "transect.id",
                   rename = "segment.id", stratum = "seasonimpact", B = 100)
```

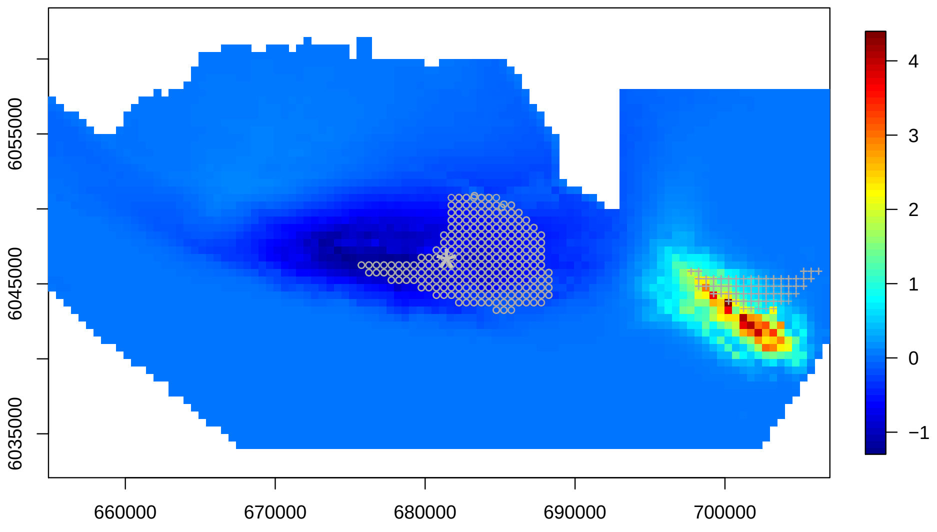
```
load('predictionboot.RData')
cis <- makeBootCIs(bootPreds)
```

17. Calculate the differences before and after across all bootstraps

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

18. Plot differences and indicate where significant positive/negative differences lie.

```
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")
```



7 References: