Fitting Hurdle Models with the dshm R-Package

Filippo Franchini

2020-04-23

Summary

In this tutorial you will learn how to use the `dshm` R-package to:

- Fit Hurdle models.
- Explore estimated relationships.
- Run diagnostics tool on fitted models.
- Plot model predictions.
- Bootstrap confidence intervals.
- Investigate spatial correlation.

Fitting Hurdle Models

Observation dataset

The dataset `obsdata` is an observation dataset containing information for dolphin sightings. Each row represents a sighting with information about `size` (i.e. number of dolphins); `distance` (i.e. the perpendicular distance in km from the transect line); `Latitude` and `Longitude` in degrees (i.e. coordinates); and finally `Sample.Label`, `Transect.Label` and `Region.Label` (i.e. the segment, transect and stratum where the sighting is located).

You can explore the `obsdata` dataset by typing:

```
head(obsdata, 6)
```

This will print the first 6 rows of the dataset:

	Region.Label	Area	Transect.Label	Sample.Label	Latitude	Longitude	distance	size
135	7	244.1659	1049107	4	-52.24967	-59.09806	0.2999543	4
136	7	244.1659	1049107	4	-52.24675	-59.09098	0.3596699	5
137	7	244.1659	1049108	1	-52.24953	-59.07130	0.0520945	1
138	7	244.1659	1049109	2	-52.30924	-59.03370	0.1285575	4
139	7	244.1659	1049109	3	-52.29406	-59.03097	0.0125000	5
140	7	244.1659	1049109	3	-52.29114	-59.03045	0.3758770	4

Detection function

In distance sampling the probability of observing an animal away from the transect line decreases with distance. When estimating abundance we have thus to take into account for this imperfect detection away from the transect line. This can be easily done by fitting a **detection function** to the `obsdata` dataset as follows:

```
mod_hr <- Distance::ds(data = obsdata, transect = "line",
          key = "hr", adjustment = NULL, truncation = max(obsdata$distance))
```

The code you just used creates a `mod_hr` object by fitting a hazard-rate detection function to the distances in `obsdata`. The detection function (i.e. the object `mod_hr`) is used in the function `dshm_fit` to correct animal abundance for imperfect detection away from the transect line. You can plot the fitted detection function:

```
plot(mod_hr, showpoints = FALSE)
```

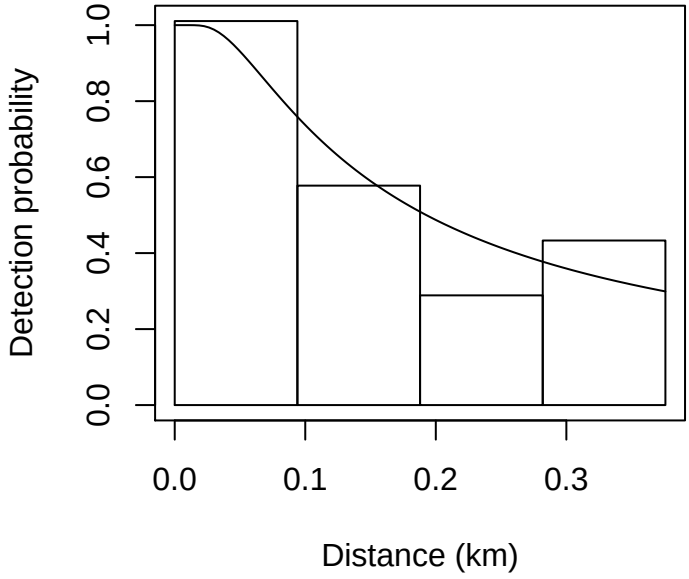


Figure 1: Fitted hazard-rate detection function.

As you can see the detection probability decreases with distance.

Specifiying submodel variants

Hurdle models are two-stage models where a presence-absence submodel (i.e. `pa`) accounts for dolphins' probability of presence while an abundance-given-presence submodel (i.e. `ab`) accounts for the number of dolphins. Submodels can be specified in many different ways that we call **submodel variants**.

Before fitting the Hurdle model using the function `dshm_fit`, we need to specify the variants for each of the two Hurdle submodels (presence-absence and abundance-given-presence):

```
variants.pa <- c("s(DC,bs='cs')",
                 "s(DR,bs='cs')",
                 "s(depth,bs='cs')")

variants.ab <- c("s(DC,bs='cs')",
                 "s(DR,bs='cs')",
                 "s(depth,bs='cs')")
```

The two objects `variants.pa` and `variants.ab` are the variants for the presence-absence (`pa`) and abundance-given-presence (`ab`) submodels, respectively. You can specify any type of structure as long as it respects the `mgcv` R-package syntax (i.e. GAM framework).

Hurdle model fit

After fitting the **detection function** and having specified the **submodel variants** we can fit the Hurdle model using the function `dshm_fit`:

```
H_m<-dshm_fit(det.fn = mod_hr,
              obsdata = obsdata,
              segdata = cor_seg_final@data,
              grid = grid_final@data,
              effects.pa = variants.pa,
              effects.ab = variants.ab)
```

The function prints automatically a model selection table for both presence-absence and abundance submodels. It also prints the selected variant IDs for each submodel. In case that multiple submodel variants are selected, the function calculates a submodel average using selected variant weights. All the relevant information is accessible through the `H_m` object. All fitted presence-absence and abundance submodel variants can be accessed through `H_m$models$pa` and `H_m$models$ab`, respectively. Selected submodel variant IDs are accessible through `H_m$info$ID.pa` and `H_m$info$ID.ab`. If we thus want pick up the best presence-absence submodel variant we can type `H_m$models$pa[[H_m$info$ID.pa[1]]]`.

Model Relationships

You can easily explore the relationships of both dolphin probability of presence and dolphin abundance with covariates by typing:

```
plot(H_m$models$pa[[H_m$info$ID.pa[1]]],select=1,shade=TRUE)
plot(H_m$models$ab[[H_m$info$ID.ab[1]]],select=1,shade=TRUE)
```

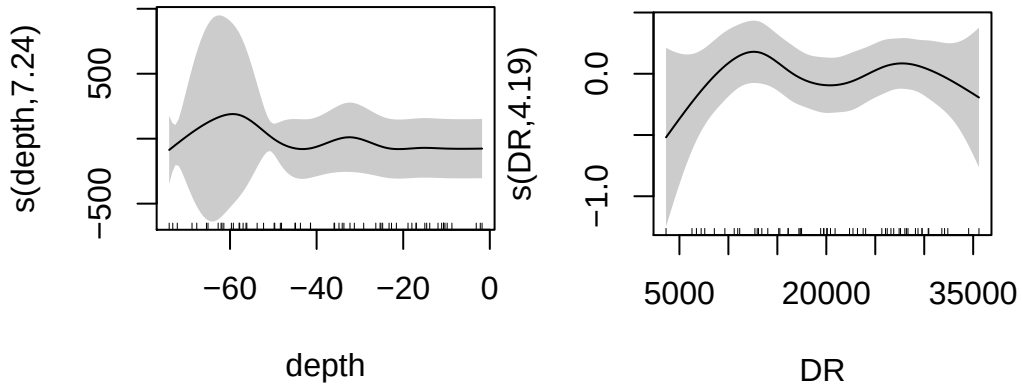


Figure 2: (*left*) Selected presence-absence submodel variant showing the probability of presence vs. depth (m). (*right*) Selected abundace submodel vairant showing the dolphin abundance vs. distance to river (m). Note that y-axis scale is on the linear predictor scale.

Model Diagnostics

You can run the model diagnostics as follows:

```
dshm_diagnostics(model = H_m,
                 plot = TRUE,
                 plot.n = 2)
```

The function `dshm_diagnostics` yields two plots:

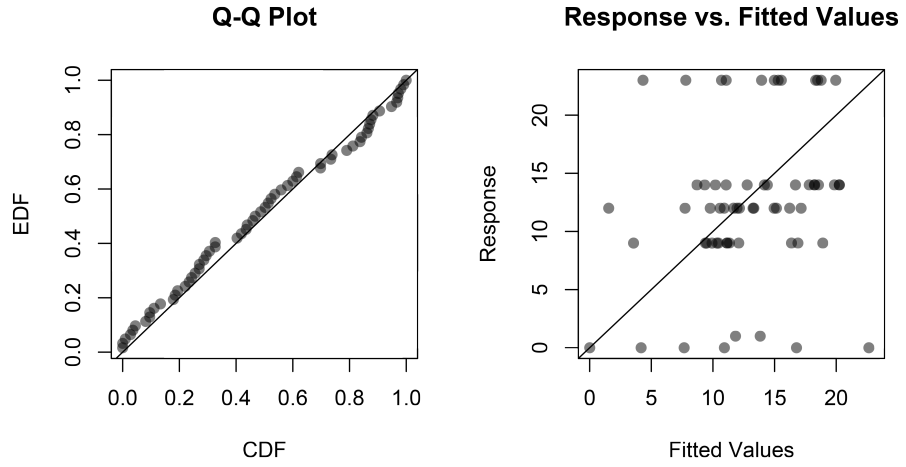


Figure 3: (*left*) EDF vs. CDF plot. (*right*) Observed values vs. Fitted values plot.

The two plots are:

- EDF (empirical distribution function) vs. CDF (cumulative distribution function). For a good fit the points on the EDF vs. CDF plot should be on the diagonal.
- Fitted vs. observed values.

The function also calculates Kolmogorov-Smirnov (KS) statistics, where KS p-value should be above 0.1 for a good fit.

Model Predictions

You can then plot the Hurdle model predictions on the prediction grid using the function `dshm_plot`:

```
dshm_plot(prediction = H_m$grid_data$H,
      grid = grid_final,
      cex = 4,
      scale_col = c("lightblue","blue","yellow","orange","red","red"),
      scale_val = c(0,1,1.5,2,3.5,4))
```

You can also decide to plot the `pa` and `ab` submodel predictions (real scale) using `H_m$grid_data$pa` and `H_m$grid_data$ab`, respectively.

The function `dshm_plot` allows to quickly explore the Hurdle model predictions. However, note that the plot does not respect the projection and real dimensions, the function is only intended for a quick exploration. You can quickly change the color scale and intervals for each color using the arguments `scale_col` and `scale_val`. In this example I chose 5 colors and intervals:

- more than 0 is lightblue
- more than 1 is blue
- more than 1.5 is yellow
- more than 2 is orange
- between 3.5 and 4 is red

Note that predictions that are > 4 they will remain uncolored. So it is important that you put a value that is bigger than your maximum.

After exploring different colors and intervals you can decide to save the grid as a raster image by setting the `saveRaster = TRUE` and by specifying the `raster_name`. Let's say we decide `raster_name = "Hurdle dolphin"`, this will save the raster as a .tif image with the name `Hurdle dolphin.tif` in your working directory.

You can then plot the raster image by typing:

```
raster::plot(raster::raster("Hurdle dolphin.tif"),
        col=colorRampPalette(c("lightblue","blue","yellow","red"))(20))
raster::plot(land_crop, add = TRUE, col = "grey")
```

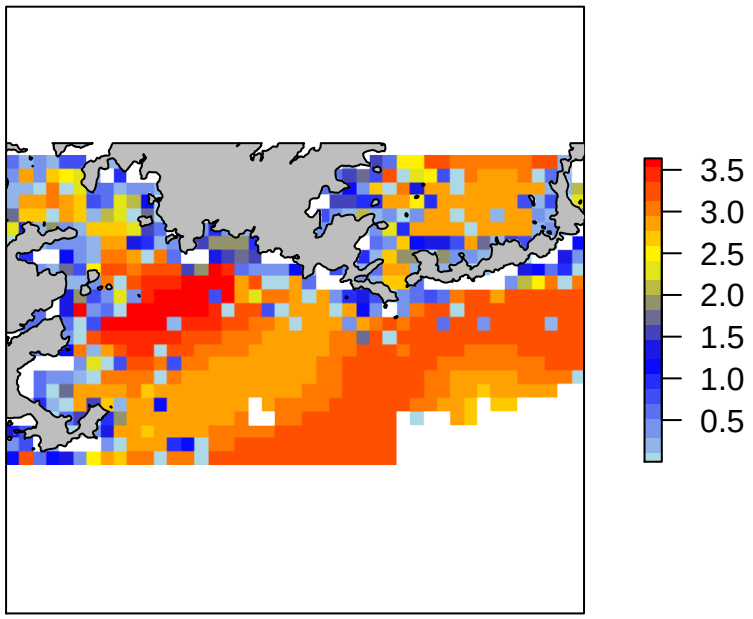


Figure 4: Hurdle model predictions (number of dolphin per grid cell)

Compared to `dshm_plot`, plotting the predictions as a raster image will preserve projection and real dimensions. Moreover, you can easily import the raster in other GIS software such as ArcGIS and QGIS.

Confidence Intervals

You can estimate prediction grid confidence intervals using the function `dshm_boot`. The function is based on a non-parametric bootstrap and uses the same submodel variant weights and knot locations stored in the object `H_m` (i.e the Hurdle model previously fitted). It is strongly recommended to run the function in parallel. Depending on the number variants for each submodel and the size of the prediction grid, the code will require from 30 minutes to a few hours to do 1000 simulations. On my computer it tool 13 minutes on 7 cores. You can run the bootstrap by typing:

```
boot<-dshm_boot(det.fn.par=list(transect="line",key="hr",
                            adjustment=NULL,formula=~1,
                            truncation = max(obsdata$distance)),
                effects.pa = variants.pa,
                effects.ab = variants.ab,
                distdata = obsdata,
                obsdata = obsdata,
                segdata = cor_seg_final@data,
                grid = grid_final@data,
                model_fit = H_m,
                nsim = 1000,
                parallel = TRUE,
                ncores = 7)
```

The function uses a resampling with replacement and as a consequence if the survey area is big (not this case) this might lead to spatial inconsistency. For big areas it is possible to restrict the resampling to transect line or stratum by setting `stratification = "transect"` or `stratification = "stratum"`.

Obviously, not all the 1000 simulations will be successful due to the fact that the model fit may fail with some dataset generated after resampling. After finishing all simulation the function will print the number of available simulations and display the message:

Would you like to continue (Yes/No)?

If the number of available simulations is very low it is recommended to type `No` and repeat the process increasing `nsim`. It is also recommended that before putting a very high `nsim`, try with just 20-30 simulations to see if the code works with your data. If you accept the available simulations the function will ask:

Would you like to choose a limit for the amount of animals per grid cell (Yes/No)?

Here you can decide if you would like to drop simulations with huge numbers. I encourage to always choose `No` unless you have clear and strong biological / statistical reasons.

After answering to the second question the `boot` object is saved on your workspace. My `boot` object contained 492 simulations, so approximately 50% of the simulations failed. For each simulation the object contains the prediction grid (`boot$sim_grids`) as well as fitted values and observations (`boot$obs_fit`).

We can write a function to calculate lower and upper confidence intervals:

```
ci <- function(x,type){
        if (type == "lwr") {
          ci<-quantile(x,probs = 0.025,na.rm = TRUE)
        } else {
          ci<-quantile(x,probs = 0.975,na.rm = TRUE)
        }
      }
```

We can then use the `ci` function to calculate lower and upper CIs as follows:

```
lwr.int<-apply(boot$sim_grids,1,ci,type="lwr")
upr.int<-apply(boot$sim_grids,1,ci,type="upr")
```

Then we use the `dshm_plot` function to plot the upper and lower CIs and to save them as raster images.

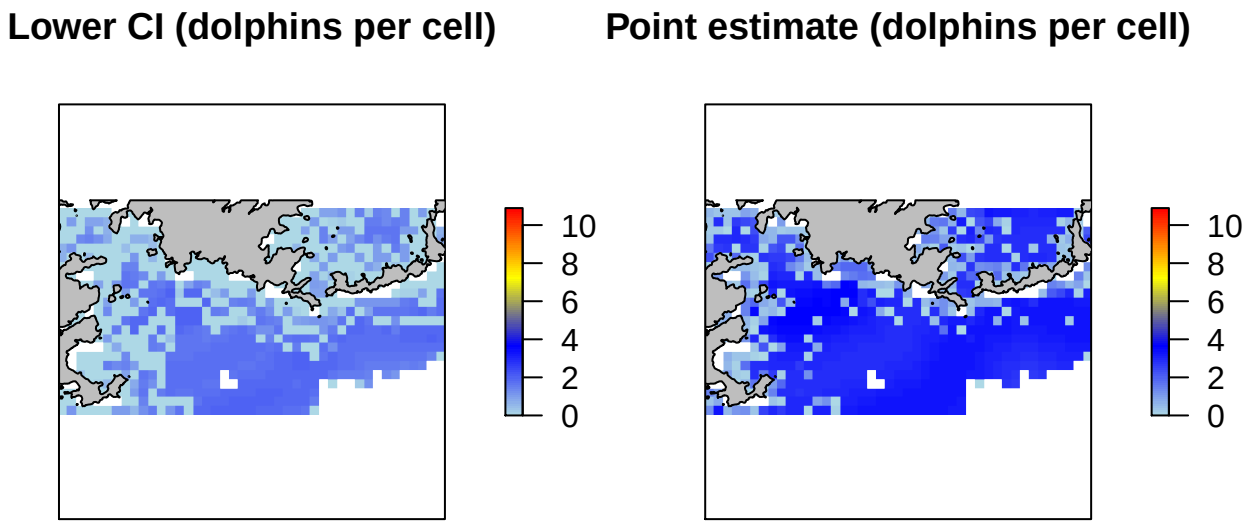
```
dshm_plot(prediction = lwr.int,
      grid = grid_final,
      cex = 4,
      scale_col = c("lightblue","blue","yellow","orange","red","red"),
      scale_val = c(0,1,1.5,2,3.5,4),
      saveRaster = TRUE,
      raster_name = "lwr_Dolphin")
```

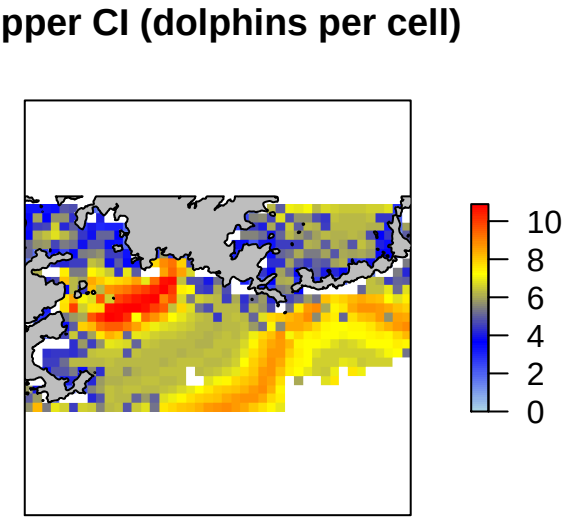
You can then repeat the same code for `upr.int` and call the raster `upr_Dolphin`. Now you saved all raster as .tif images in your working directory. You can load them in your workspace using the following code:

```
pred <- raster::raster("Hurdle dolphin.tif")
pred_lwr <- raster::raster("lwr_Dolphin.tif")
pred_upr <- raster::raster("upr_Dolphin.tif")
```

You can plot all three raster images for model predictions together with lower and upper CIs:

```
raster::plot(pred_lwr,
        main="Lower CI (dolphins per cell)",
        axes=FALSE,col=colorRampPalette(c("lightblue","blue","yellow","red"))(100),
        zlim=c(0,max(na.omit(pred_upr[]))))
raster::plot(land_crop,col="grey",add=TRUE)
raster::plot(pred,
        main="Point estimate (dolphins per cell)",
        axes=FALSE,col=colorRampPalette(c("lightblue","blue","yellow","red"))(100),
        zlim=c(0,max(na.omit(pred_upr[]))))
raster::plot(land_crop,col="grey",add=TRUE)
raster::plot(pred_upr,
        main="Upper CI (dolphins per cell)",
        axes=FALSE,col=colorRampPalette(c("lightblue","blue","yellow","red"))(100),
        zlim=c(0,max(na.omit(pred_upr[]))))
raster::plot(land_crop,col="grey",add=TRUE)
```





Note that all plots are on the same colour scale. You can easily set the same color scale also using `dshm_plot`.

Spatial Correlation

You can calculate spatial correlation with distance using the function `dshm_spatial_correlation` as follows:

```
dshm_spatial_correlation(coord = sp::coordinates(cor_seg_final),
z = H_m$residuals, xlab = "Distance (km)",
ylab = "Correlation", lims = c(-0.2,0.2))
```

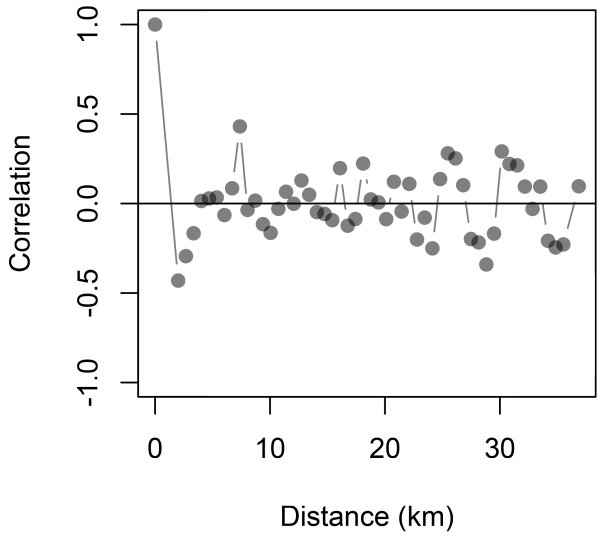


Figure 5: Spatial correlation with distance in km.

The function yields a plot for correlation coefficient vs. distance in km and calculates the percentage of data that is between a minimum and maximum correlation value defined by the user through the argument `lims`.