Beginner Point Transects Tutorial

Michael Kleinsasser, Jason D. Carlisle and Trent L. McDonald

May 22, 2018

Introduction

This tutorial is a beginner's guide to doing point transect distance-sampling analysis using `Rdistance`. Topics covered include input data requirements, fitting a detection function, estimating abundance (or density), and selecting the best fit detection function using AICc. We use the internal datasets `thrasherDetectionData` and `thrasherSiteData` (point transect surveys of brown thrashers). This tutorial is current as of version 2.1.0 of `Rdistance`.

1: Install and load Rdistance

If you haven't already done so, install the latest version of `Rdistance`. In the R console, issue `install.packages("Rdistance")`. After the package is installed, it can be loaded into the current session as follows:

```
require(Rdistance)
```

```
## Loading required package: Rdistance
```

```
## Rdistance (version 2.1.0)
```

2: Read in the input data

To complete this tutorial, we use the datasets `thrasherDetectionData.rda` and `thrasherSiteData.rda` (point transect surveys of brown thrashers) from the package `Rdistance`.

The first required dataset is a detection data.frame, with a row for each transect surveyed and columns named:

- `siteID` = Factor, the site or transect surveyed
- `groupsize` = Numeric, the number of individuals within the detected group.
- `dist` = Numeric, the perpendicular distance (also known as off-transect distance) from the point transect to the detected group.

Load the example dataset of thrasher detections and observed distances (`thrasherDetectionData`) using the following commands:

```
load("thrasherDetectionData.rda")
head(thrasherDetectionData)
```

```
##   siteID groupsize dist
## 1  C1X01         1   11
## 2  C1X01         1  183
## 3  C1X02         1   58
## 4  C1X04         1   89
## 5  C1X05         1   83
## 6  C1X06         1   95
```

The second required dataset is a transect data.frame, with a row for each transect surveyed, and the following required columns, named as follows:

- `siteID` = Factor, the site or transect surveyed.
- `...` = Any additional transect-level covariate columns (these will be ignored).

Load the example dataset of thrasher transects surveyed (`thrasherSiteData`) using the following commands:

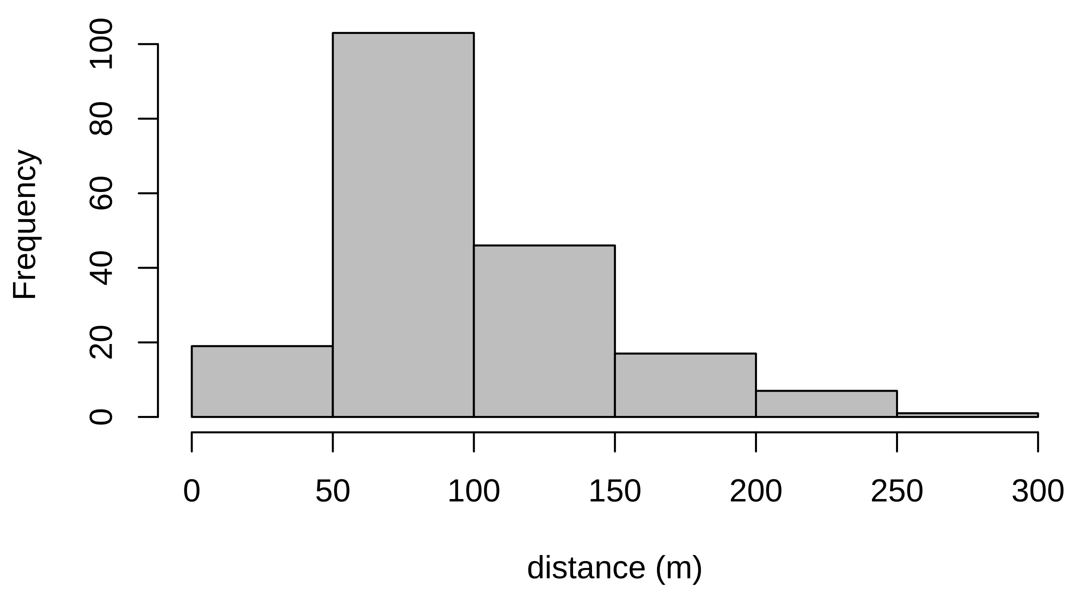
```
load("thrasherSiteData.rda")
head(thrasherSiteData)
```

```
##   siteID observer bare herb shrub height
## 1  C1X01     obs5 45.8 19.5  18.7   23.7
## 2  C1X02     obs5 43.4 20.2  20.0   23.6
## 3  C1X03     obs5 44.1 18.8  19.4   23.7
## 4  C1X04     obs5 38.3 22.5  23.5   34.3
## 5  C1X05     obs5 41.5 20.5  20.6   26.8
## 6  C1X06     obs5 43.7 18.6  20.0   23.8
```

3: Fit a detection function

Once the data are imported, the first step is to fit a detection function. Before we do so, explore the distribution of the distances:

```
hist(thrasherDetectionData$dist, col="grey", main="", xlab="distance (m)")
```



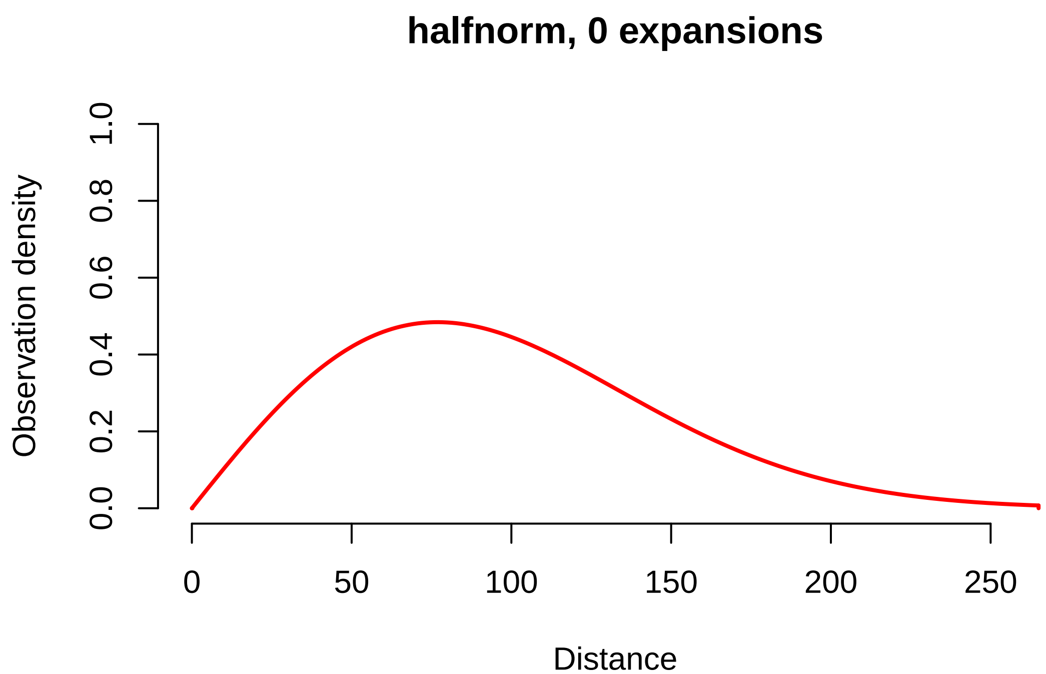

```
summary(thrasherDetectionData$dist)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   11.00   63.00   86.00   97.16  123.00  265.00
```

Next, fit a detection function (plotted as a red line) using `dfuncEstim`. Point transect is an option in `dfuncEstim` by selecting `PointSurvey = TRUE`. Our illustration uses the half-normal likelihood as the detection function. In section 5, we demonstrate an automated process that fits multiple detection functions and compares them using AICc.

```
dfunc <- dfuncEstim(formula = thrasherDetectionData$dist ~ 1,
                    detectionData = thrasherDetectionData,
                    siteData = thrasherSiteData,
                    pointSurvey = TRUE,
                    pointID = thrasherDetectionData$siteID,
                    likelihood = "halfnorm")

plot(dfunc)
```




```
dfunc
```

```
## Call: dfuncEstim(formula = thrasherDetectionData$dist ~ 1, detectionData = thrasherDetectionData,
##
## Coefficients:
##        Estimate  SE      z         p(>|z|)
## Sigma  76.88769  2.9063  26.45552  3.152313e-154
##
## Convergence: Success
## Function: HALFNORM
## Strip: 0 to 265
## Effective detection radius (EDR): 108.5909
## Probability of detection: 0.1679173
## Scaling: g(0) = 1
## Log likelihood: 1004.254
## AICc: 2010.53
```

The effective detection radius (EDR) is the essential information from the detection function that will be used to estimate abundance in section 4. The EDR is calculated by integrating over the detection function, resulting in the area under the curve of the detection function. See the help documentation for `EDR` for details.

4: Estimate abundance given the detection function

Estimating abundance requires the additional information contained in the the thrasher site dataset, described in section 2, where each row represents one transect. Load the example dataset of surveyed thrasher transects from the package.

```
data("thrasherSiteData")
head(thrasherSiteData)
```

```
##   siteID observer bare herb shrub height
## 1  C1X01     obs5 45.8 19.5  18.7   23.7
## 2  C1X02     obs5 43.4 20.2  20.0   23.6
## 3  C1X03     obs5 44.1 18.8  19.4   23.7
## 4  C1X04     obs5 38.3 22.5  23.5   34.3
## 5  C1X05     obs5 41.5 20.5  20.6   26.8
## 6  C1X06     obs5 43.7 18.6  20.0   23.8
```

Next, estimate abundance (or density in this case) using `abundEstim`. If `area` = 1, then density is given in the squared units of the distance measurements — in this case, thrashers per square meter. Instead, we set `area` = 10000 in order to convert to thrasher per hectare (1 ha == 10,000 m^2). The equation used to calculate the abundance estimate is detailed in the help documentation for `abundEstim`.

Confidence intervals for abundance are calculated using a bias-corrected bootstrapping method (see `abundEstim`), and the detection function fit in each iteration of the bootstrap is calculated. Note that, as with all bootstrapping procedures, there may be slight differences in the confidence intervals between runs due to so-called 'simulation slop'. Increasing the number of bootstrap iterations (`R` = 100 used here) may be necessary to stabilize CI estimates.

```
# Estimate Abundance - Density; fatalities per m2
fit <- abundEstim(dfunc         = dfunc,
                  detectionData = thrasherDetectionData,
                  siteData      = thrasherSiteData,
                  area           = 10000,       # For density
                  R              = 100,
                  ci             = 0.95)
```

```
fit
```

```
## Call: dfuncEstim(formula = thrasherDetectionData$dist ~ 1, detectionData = thrasherDetectionData,
##
## Coefficients:
##        Estimate  SE      z         p(>|z|)
## Sigma  76.88769  2.9063  26.45552  3.152313e-154
##
## Convergence: Success
## Function: HALFNORM
## Strip: 0 to 265
## Effective detection radius (EDR): 108.5909
## Probability of detection: 0.1679173
## Scaling: g(0) = 1
## Log likelihood: 1004.254
## AICc: 2010.53
```

```
##
## Abundance estimate:  0.4408976 ;  95% CI=( 0.4112426 to 0.4723903 )
```

The abundance estimate can be extracted from the `fit` object.

```
fit$n.hat
```

```
## [1] 0.4408976
```

The confidence interval (in this case 95%) can be extracted from the `fit` object.

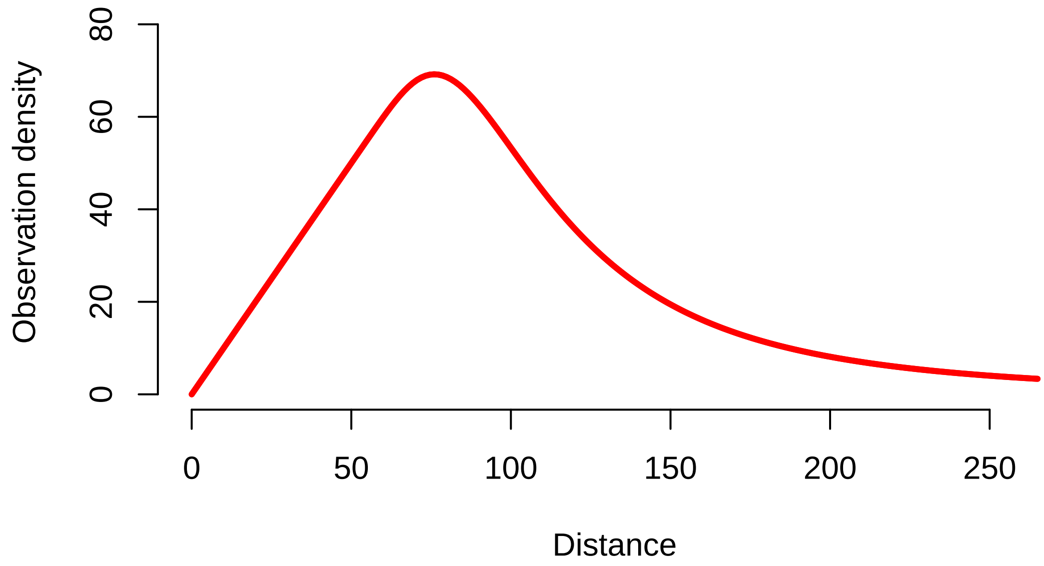
```
fit$ci
```

```
## 4.865615% 98.81483%
## 0.4112426 0.4723903
```

5: Use AICc to select a detection function and estimate abundance

Fitting of a detection function and estimating abundance (sections 3 and 4) can be automated using the function `autoDistSamp`. The function attempts to fit multiple detection functions, uses AICc (by default, but see help documentation for `autoDistSamp` under `criterion` for other options) to find the 'best' detection function, then proceeds by estimating abundance using the best fit detection function (the estimated function with lowest AICc). By default, `autoDistSamp` tries a large subset of `Rdistance`'s built-in detection functions, but you can control exactly which detection functions are attempted (see help documentation for `autoDistSamp`). Specifying `plot=TRUE` returns a plot of each detection function. In this example, we attempt to fit the likelihoods half-normal, hazard rate, exponential, and "uniform" with no expansion terms, and we don't plot each function (`plot=FALSE`).

```
# Automated Fit - Conduct the fit automated but also choose the best model based on AIC
auto <- autoDistSamp(formula       = thrasherDetectionData$dist ~ 1,
                     detectionData = thrasherDetectionData,
                     siteData      = thrasherSiteData,
                     pointSurvey   = TRUE,
                     expansions    = c(0),
                     likelihoods   = c("halfnorm", "hazrate", "negexp", "uniform"),
                     plot          = FALSE,    # Would plot ALL models if TRUE
                     area          = 10000,
                     R             = 100,
                     ci            = 0.95,
                     plot.bs       = TRUE,
                     w.hi          = 265)
```




```
auto
```

```
## Call: dfuncEstim(formula = formula, detectionData = detectionData,     siteData = siteData, likeliho
##
## Coefficients:
##        Estimate   SE        z         p(>|z|)
## Sigma  93.729609  5.871345  15.96391  2.280016e-57
## Beta    4.199521  0.397081  10.57598  3.851349e-26
##
## Convergence: Success
## Function: HAZRATE
## Strip: 0 to 265
## Effective detection radius (EDR): 118.6223
## Probability of detection: 0.2003737
## Scaling: g(0) = 1
## Log likelihood: 999.0199
## AICc: 2002.103
##
## Abundance estimate:  0.3694814 ;  95% CI=( 0.3449149 to 0.3976385 )
```

The detection function with the lowest AICc value (and thus selected as the 'best') is the hazard rate likelihood with 0 cosine expansion terms.

Conclusion

In sections 3 and 4, we fitted a half-normal detection function and used that function to estimate thrasher density. Our estimate was 0.44 thrashers per ha (95% CI = 0.41 - 0.47). In section 5, we used AICc to determine the best-fitting detection function and used the function with the lowest AICc to estimate thrasher density. The thrasher density estimate was 0.37 thrashers per ha (95% CI = 0.34 - 0.4). (Note, estimates may vary slightly from these due to minor 'simulation slop' inherent in bootstrapping methods).

That concludes this `Rdistance` tutorial. You are now ready to read in your own data, fit a detection function, and estimate abundance for point transects.