

# User-Defined Likelihood Functions in Rdistance

*Trent L. McDonald and Jason D. Carlisle*

*August 6, 2018*

## Introduction

Detection functions are often fit using a small set of likelihood functions that lend themselves to the distance-sampling data (e.g., half-normal, hazard rate, etc.). **Rdistance** has many of these traditional likelihood functions built in (see help documentation for **F.dfunc.estim**), but the package also allows the user to specify a custom function. Here, we give an example of how to specify a user-defined likelihood function when fitting a detection function in **Rdistance**. This tutorial is current as of version 2.1.2 of **Rdistance**.

## Details

Assuming that the user-defined likelihood is named *form*, estimation of a user-defined likelihood requires that the following two functions be defined:

1. **form.like** - The likelihood: This function defines the user-defined likelihood and should take the following inputs, in this order:
  - **a** = the parameter vector for the likelihood.
  - **dist** = the vector of observed distances.
  - **covars** = a matrix of covariates, one line per detection. If no covariates are included, set **covars** = **NULL**.
  - **pointSurvey** = a logical scalar equal to **TRUE** if distances are radial from a point, **FALSE** if distances are perpendicular off-transect.
  - **w.lo** = the left truncation (minimum distance).
  - **w.hi** = the right truncation (maximum distance).
  - **series** = the name of the expansion series. If the likelihood does not use a series, this function still requires a parameter named **series**.
  - **expansions** = the number of expansions. If the likelihood does not use expansions, this function still needs a parameter named **expansions**.
  - **scale** = a logical scalar. If **TRUE**, the likelihood should be scaled to integrate to 1.0. If **FALSE**, the user defined likelihood does not need to integrate to 1.0. See the help documentation for **uniform.like** for an example of how this parameter should be used.

The likelihood function should return a vector the same length as **dist** (the vector of distances) containing the likelihood values. That is, the *i*-th element of the output vector should be the likelihood of observing **dist[i]**.

2. **form.start.limits** - This function provides the starting values, limits, and names of parameters in the likelihood and should take the following inputs (described above): **dist**, **expansions**, **w.lo**, and **w.hi**.

This function should return a list containing the following components:

- `start` = a vector of length  $p$  of starting values for the parameters of the likelihood, assuming there are  $p$  parameters in the likelihood.
- `lowlimit` = a vector of length  $p$  of lower limits for parameters of the likelihood.
- `highlimit` = a vector of length  $p$  of upper limits for parameters of the likelihood.
- `names` = a vector of length  $p$  of names for the parameters of the likelihood.

## Example

A user-defined likelihood function: the triangular distribution on  $[0, b]$ .

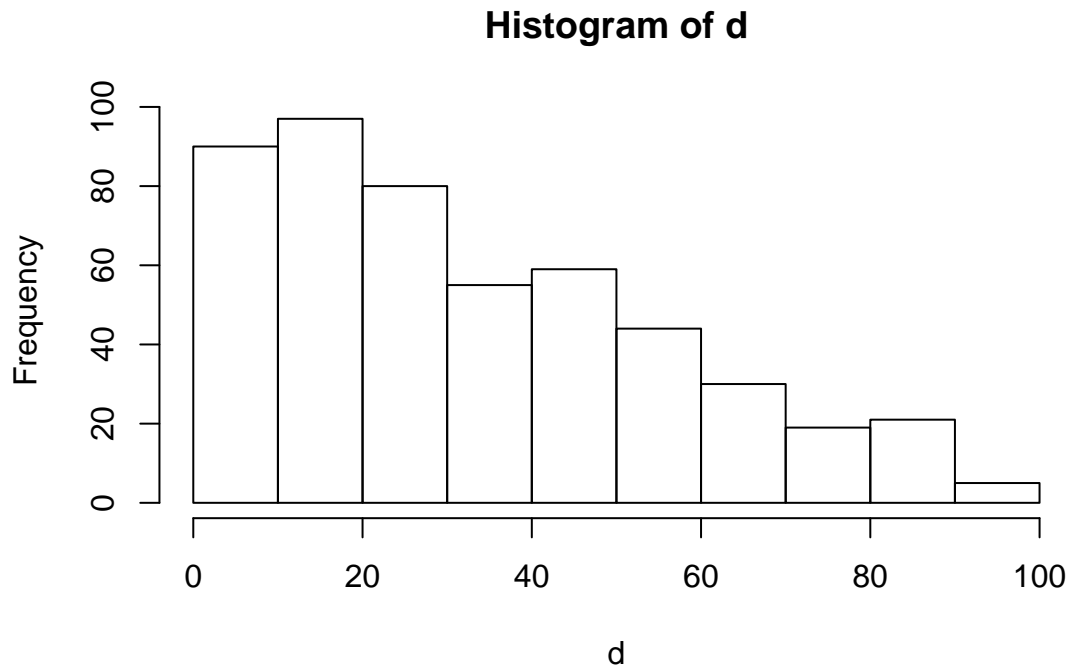
```
# Part 1: The likelihood function
triangular.like <- function(a, dist, covars=NULL,
                           pointSurvey=FALSE, w.lo=0, w.hi,
                           series="", expansions=0, scale=TRUE){
  L <- (2/a)*(1 - dist/a)
  L[ L < 0 ] <- 0
  L
}

# Part 2: The starting values, limits, and names of parameters in the likelihood
triangular.start.limits <- function(dist, expansions, w.lo, w.hi){
  list(start=max(dist)*.75,
       lowlimit=w.lo,
       highlimit=w.hi,
       names="Max")
}
```

To demonstrate, generate some data and fit the triangular distance function.

```
# A function to generate triangular random deviates
rtriang <- function(n, b){
  x <- seq(0, b, length=500)
  CDF <- 2*x/b - (x/b)^2
  u <- runif(n)
  r <- approx( CDF, x, xout=u )$y
}

# Simulated vector of distances
set.seed(123)
d <- rtriang(500, 100) # true b = 100
hist(d)
```



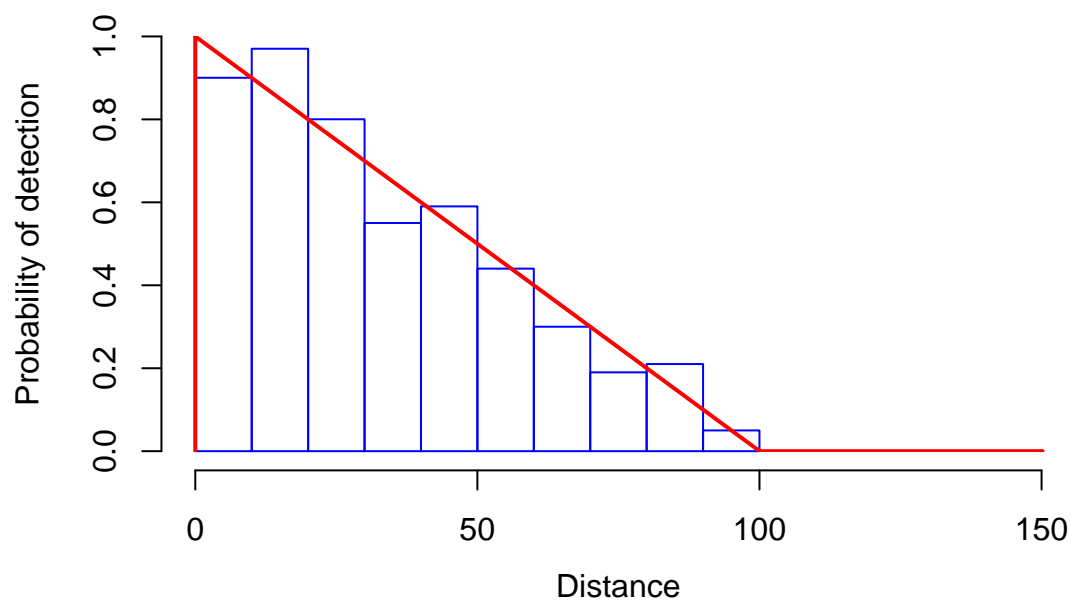
```
# Fit detection function with user-defined "triangular" likelihood
# Requires the F.dfunc.estim function from Rdistance
require(Rdistance)
tri.dfunc <- dfuncEstim( d~1, likelihood="triangular", w.hi=150 )
tri.dfunc
```

```
## Call: dfuncEstim(formula = d ~ 1, likelihood = "triangular", w.hi
##      = 150)
## Coefficients:
##      Estimate SE          z      p(>|z|)
## Max 100.0356 1.657778 60.34319 0
##
## Convergence: Success
## Function: TRIANGULAR
## Strip: 0 to 150
## Effective strip width (ESW): 50.01839
## Probability of detection: 0.3334559
## Scaling: g(0) = 1
## Log likelihood: 2204.624
## AICc: 4411.256
```

All dfunc methods (e.g., plot, predict, etc.) should work with custom distance functions.

```
plot(tri.dfunc)
```

## triangular, 0 expansions



```
AIC(tri.dfunc)
```

```
## [1] 4411.256
## attr("criterion")
## [1] "AICc"
```

The effective strip width of a distance function is simply the area under the distance function after scaling. In `Rdistance`, scaling is accomplished by setting `x.scl` and `g.x.scl` in the call to `dfuncEstim`. `dfuncEstim` scales the distance function such that  $g(x.scl) = g.x.scl$ . The default is `x.scl = 0` and `g.x.scl = 1`.

For the triangular case,  $g(0) = 1$  and the true effective strip width is simply the area of a right triangle located at the origin. The true effective strip width of the triangular distance function is:

```
tri.dfunc$g.x.scl*tri.dfunc$param / 2
```

```
##      Max
## 50.01781
```

The effective strip width calculated using ESW differs only slightly due to numerical integration error.

```
ESW(tri.dfunc)
```

```
## [1] 50.01839
```