Assessing acoustic tag and receiver performance: An approach for visualizing and analyzing passive detection range studies in R

Workshop Handout - GLATOS, February 28, 2019

Todd Hayden (haydento@msu.edu), Tom Binder, Mike Lowe

Updated: 2019-11-12

Contents

1 Introduction

This vignette presents one approach for evaluating tag and receiver performance from a passive detection range study. Passive detection range studies consist of a network of multiple tags or receivers such that each tag transmission can be detected on multiple receivers deployed at different distances from the tag. All tags and receivers are moored at the same location during the study, resulting in a time series of tag detections at different distances. Range testing studies conducted prior to deployment of tagged fish allow a researcher to evaluate tag performance at their study site and determine appropriate receiver spacing that maximizes receiver network performance and supports study objectives. As well, deployment of range testing tags concurrently with tagged fish provides a mechanism to identify periods of time when performance of acoustic networks may have been poor and resulted in fish passing through the array undetected.

In this vignette, we present the code for several plots that we have found useful for summarizing and visualizing results of detection range studies. As well, we include code for calculating detection range curves over a user-defined time period. Detection range curves represent the probability of detecting a

tag transmission during a time interval at different distances between tag and receiver. Depending on study goals, researchers may desire to fit statistical models to empirical detection range curves in order to generalize and summarize changes in detection range attenuation through time. Detection of acoustic tags in aquatic environment is mediated by complex biotic and abiotic variables that are highly site-specific and fitting statistical models to detection range curves is project specific and beyond the scope of this vignette. However, in the last part of this vignette, we will demonstrate how empirically determined detection range curves may be used to simulate the probability that a fish moving through a line of receivers or a grid of receivers is detected.

Please note that the code and ideas presented in this vignette are examples of how data from common passive detection range testing studies using Vemco tags and receivers may be analyzed and do not necessarily represent best practices. Successful evaluation of detection range depends on specific research objectives and experimental design and does not easily conform to a "one-size fits all" analysis.

2 Data

Example detection data discussed in this vignette was from a passive range test in a small, shallow bay (<3 meters) in Lake Superior. The study was conducted by Bay Mills Indian Community [1] with the goal of evaluating tag performance in a shallow nearshore environment with extensive emergent vegetation. Vemco VR2Tx (69kHz) acoustic receivers were used exclusively in this study. In addition to detecting and recording tag transmissions, the VR2Tx receiver is outfitted with an user-adjustable integrated acoustic tag that can be programmed for different transmission strength and nominal tag delay intervals. These acoustic receivers have the ability to transmit an acoustic signal (tag) and detect and record acoustic transmissions. As well, these receivers log all tag transmissions such that the total number of tag transmissions in a time interval are known and enable passive range testing without independent tags and receivers.

The example range test discussed in this vignette consisted of four tags deployed 0-600 meters from a total of 9 acoustic receivers (Figure 1). The internal transmitter on 2 VR2Tx receivers were programmed to randomly transmit a 160 dB coded tag with an average nominal delay of 10 minutes (540-660 minute range) and two internal transmitters of 2 VR2Tx receivers were programmed to transmit a 154 dB transmission at a nominal delay of 10 minutes (540-660 minute range). The remaining 5 VR2Tx receivers did not transmit and were deployed at various distances resulting in a range of tag-receiver distances ranging from 0-600 meters. All equipment was deployed in October, 2018 and recovered in November 2018.

Currently, range testing data may be submitted to GLATOS by setting up a project and completing a data submission. Details about how to format and submit range testing data to the GLATOS database is beyond the scope of this vignette. Please contact the GLATOS Data Steward for more information about uploading range testing data and data export.[2]

In the first part of this vignette, we provide R code for loading and cleaning up the detection data. In subsequent sections, we present a figure to visualize raw detection history of each receiver, code to calculate detection probability, extract detection range curves, and a figure to visualize detection range curves.

The powerful R package data.table is used extensively in this vignette. Package vignettes included with the data.table package are well done and are an excellent resource for getting started with data.table. If you are not familiar with data.table, we highly recommend reviewing the *Introduction to data.table*, *Reference semantics*, and *Efficient reshaping using data.table* vignettes. [3]

[1] contact- Frank Zomar, email- fzomer@baymills.org

[2] GLATOS Data Steward- Nancy Nate, email: nnate@usgs.gov

[3] Additional information about data.table for R, including vignettes: https://github.com/Rdatatable/data.table/wiki

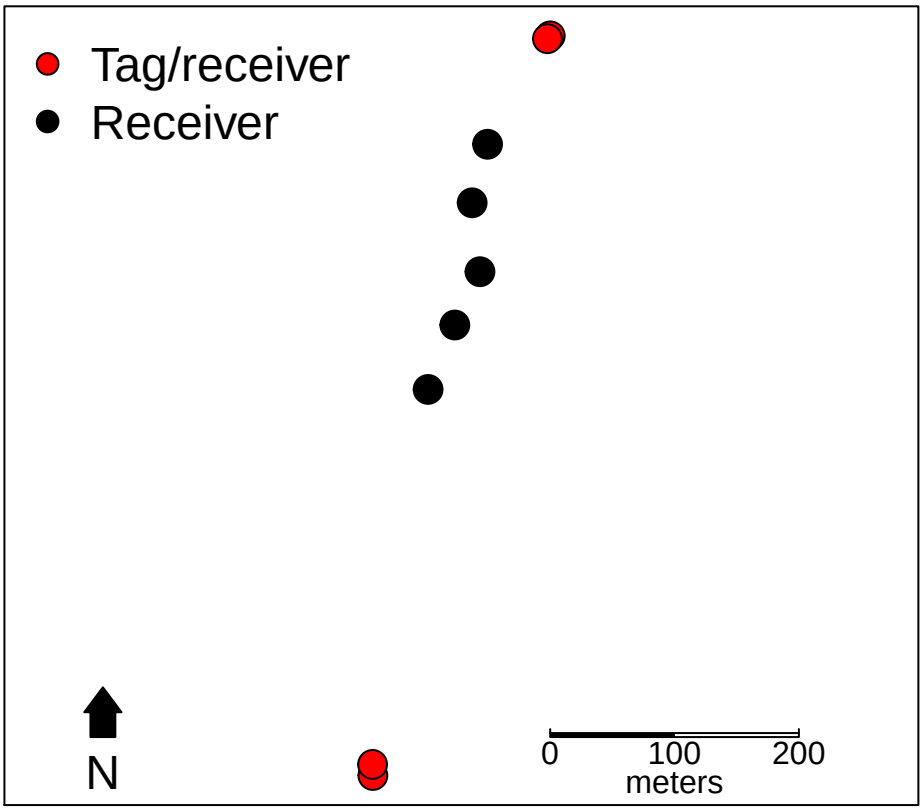


Figure 1: Study map of acoustic tags and receivers. Black circles are receivers and red circles are receivers with active integrated tags. Two tags with different transmission levels were deployed at each end of receiver line.

3 Import data and clean up

First, we need to load libraries, load the detection data file, and do a little clean-up. Example data used in this vignette are included with the GLATOS R package and will be loaded below using `data(range_detection)`, but note that this data object was created using the GLATOS `read_glatos_detections` function to load data from the original GLATOS detection export CSV file (e.g., named *WBCCM_detectionsWithLocs_20181115_162626.csv*). The most recent version of the GLATOS package and installation instructions can be obtained at https://gitlab.oceantrack.org/GreatLakes/glatos.

```
# load libraries
library(glatos)
library(data.table)
library(geosphere)
library(ggplot2)
library(reshape2)

# load detection data from glatos package
data(range_detection)
dtc <- range_detection

# convert dtc to data.table
setDT(dtc)
```

The resulting `dtc` object contains 30 columns and is identical to the standard export for all glatos detection data. Please see the data loading vignette (link) and data.dictionary vignette (link) included in `glatos` for complete information about data structure and contents of columns.

Many of the columns in the dtc object are not needed for this analysis. In the next code chunk, we will extract only the necessary columns from the `dtc` object and calculate the geographic distance between each receiver using the `geosphere::distm` function. The internal tag on 4 receivers was used in this study so distances among receivers is the same as distances between each tag and receiver.

```
# extract only needed columns
dtc <- dtc[, c("detection_timestamp_utc", "transmitter_id", "receiver_sn",
               "deploy_lat", "deploy_long")]

# extract unique combinations of receiver_sn, deploy_lat, and deploy_long
rec_dist <- unique(dtc[,c("receiver_sn", "deploy_lat", "deploy_long")])

# calculate distance matrix
dist_matrix <- distm(rec_dist[, c("deploy_long", "deploy_lat")],
                     rec_dist[, c("deploy_long", "deploy_lat")])

# rename columns and rows of matrix
colnames(dist_matrix) <- rec_dist[,get("receiver_sn")]
rownames(dist_matrix) <- paste0("6", substr(rec_dist[,get("receiver_sn")], 3,6))

# convert matrix into 3-column data using data.table::melt
dist_matrix <- as.data.table(reshape2::melt(dist_matrix,
                                            id.vars = c("columns", "rows"),
                                            value.name = "dist_m"))

# change names
setnames(dist_matrix, c("Var1", "Var2"),
```

```
        c("transmitter_id", "receiver_sn"))

# "Transmitter_id" and "receiver_sn" columns are integer type but we need character.
# Next line of code converts both to character
dist_matrix[,c("transmitter_id", "receiver_sn") := list(as.character(transmitter_id),
                                                        as.character(receiver_sn))]

# Join "dist_matrix" object with "dtc" object using "transmitter_id" and
# "receiver_sn" as keys. Keep all records of dtc.
dtc <- dist_matrix[dtc, on = c("transmitter_id", "receiver_sn")]
```

In the last line, we used a `data.table` join to merge the `dist_matrix` object with `dtc` on key columns common to both objects.[4]. At this point, the `dtc` object contains an additional column (`dist_m`) that specifies the distance between the transmitter and the receiver that detected the transmission.

4 Visualize raw detection data

In the next code chunk, we create a multi-panel plot of detections (Figure 2). Each panel represents a different tag and the Y-axis is the distance between tag and receiver in meters. The x-axis represents time in each panel. Tags 482223 and 482221 were programmed to transmit a 154 dB signal and tags 482222 and 482215 transmitted a 158 dB signal.

```
# set base plot parameters
par(mfrow=c(1,4), mar=c(3,2,1,0), oma=c(2,3,0,0), las=1, bty="l")

# extract vector of transmitter ids
groups <- unique(dtc$transmitter_id)

# loop through each transmitter and create plot
for(i in 1:length(groups)){
 dtc.i <- dtc[transmitter_id == groups[i],]
    plot(dtc.i$detection_timestamp_utc, dtc.i$dist_m, pch = 16, col = "red",
         las = 1, ylab = NA, xlab = NA,
         main = paste0("tag ", groups[i]), ylim = c(0,615),
         xlim = c(as.POSIXct("2018-10-09 00:00:00", tz = "UTC"),
                  as.POSIXct("2018-11-09 00:00:00", tz = "UTC")),
         axes = FALSE)

  # create axes
  if(i %in% c(1)){axis(2, at = seq(0,600, by = 100), labels = TRUE)}
  if(i %in% c(2,3,4)){axis(2, at = seq(0,600, by = 100), labels = FALSE)}
  axis.POSIXct(1, at = seq(as.POSIXct("2018-10-09 00:00:00", tz = "UTC"),
                           as.POSIXct("2018-11-10 00:00:00", tz = "UTC"),
                           by = "10 days"), labels = TRUE, format = "%m-%d")

  # add axis labels
  mtext("tag-receiver distance", outer = TRUE, side = 2, las = 0, line = 1, cex = 0.6)
  mtext("date (month-day)", outer = TRUE, side = 1, cex = 0.6)
```

[4]The `data.table` package provides a simple framework for complex merge functionality that we use extensively in this vignette. Additional examples of data.table merges or "joins" are included in the data.table package help, accessed by typing `?data.table` in R. Base function aggregate and ddply in the plyr packages provide similar functionality

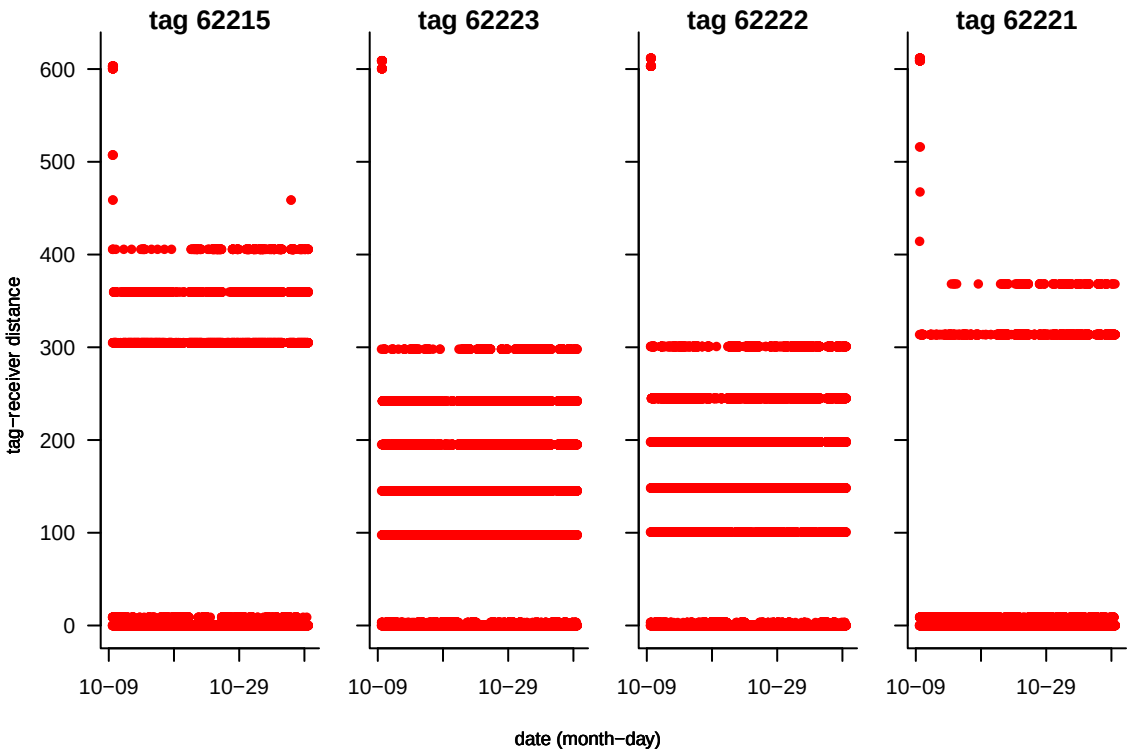


Figure 2: Detections (red circles) of acoustic tags over time as a function of distance between tag and receiver. All tags were programmed to broadcast with nominal delay of 10 minutes. Output of tags 482223 and 482221 was 154 dB and output of tags 482215 and 482222 was 158 dB

```
  box()
}
```

Notice the sharp decline in the number of detections recorded at tag-receiver distances greater than about 400 m. (Figure 2)

5 Calculate detection probability

One advantage of using VR2Tx receivers for range testing is that each receiver with an active integrated tag logs each tag transmission. This allows us to determine exactly how many tag transmissions occurred in a given time interval (remember, tags were programmed to randomly transmit a signal on average every 10 minutes). When the integrated tag is active, VR2Tx receivers record a tag transmission from the integrated tag as the last 4 digits of the receiver serial number with a "6" prefix. For example, the tag id code of the integrated tag for receiver 482215 is 62215. This coding scheme enables us to determine which detections were actually tag transmissions from the on-board tag for each receiver and by counting records from the integrated tag on the receiver, we can determine the number of tag transmissions that occurred in a time interval. In the next column, we are going to create new tag and receiver id columns where the "6" and "48" prefixes are removed. We can then use the new columns to identify and subset records of tag transmissions by subsetting records where the tag and receiver id are the same.

```r
# use substring function to simplify tag and receiver ids and
# add short_trans_id and short_rec_id columns to dtc
dtc[, c("short_trans_id", "short_rec_id") := list(substr(dtc$transmitter_id, 2,5),
                                                  substr(dtc$receiver_sn, 3,6))]
# subset detections from same tag and receiver
# these represent actual transmissions from each tag
trans_log <- dtc[short_trans_id == short_rec_id]
```

The `trans_log` object contains timestamps for all integrated tag transmissions. By simply counting detections in the `trans_log` object, we can determine the maximum number of times a tag could have been detected during the study. To calculate detection probability for a receiver, we need to divide the total number of times a tag could have been detected by the total number of times the tag was detected. However, before we do this, we need to synchronize receivers and tags to account for any clock drift. Detection data exports from GLATOS are corrected for potential receiver clock drift owing to environmental conditions (i.e., temperature) and mechanical characteristics of digital clocks. This correction is based on an assumption that receiver clocks will drift linearly while deployed and uses the difference between receiver time and computer time when the receiver is initiated and downloaded to correct for receiver clock drift on detection timestamps. The time correction for clock drift on receivers is implemented in Vemco VUE software and although the simple linear correction eliminates most clock drift, slight differences between tag transmission time and receiver detection time may be present in the detection data owing to non-linear clock drift and travel time of the acoustic signal between tag and receiver. However, because the VR2Tx receiver logs the actual time when each tag transmitted a signal, we are able to account for tag transmission travel time and standardize all detection timestamps to the tag transmission timestamp. We can do this by joining the `trans_log` object extracted in the previous code chunk with the `dtc` object using a `data.table` non-equi join. We essentially create a time window for each detection timestamps that include the logged transmission timestamp and then match each detection with transmission timestamp that falls within the window. This is completed in the next code chunk and the tag transmission timestamp identified for each detection is added to the `dtc` object as a new column `log_timestamp`. The last line of code in the next chunk simply calculates the difference between the transmission timestamp and detection timestamp for a sanity check to make sure difference between these two timestamps is small.

```r
# All detection timestamp within 10 seconds (+- 5 seconds) of tag transmission
# were identified as same transmission
dtc[, c("start", "end") := list(detection_timestamp_utc - 5,
                                detection_timestamp_utc + 5)]

# join detections with tag transmissions with data.table non-equi join
dtc <- trans_log[dtc, .(detection_timestamp_utc = i.detection_timestamp_utc,
                        transmitter_id = i.transmitter_id,
                        receiver_sn = i.receiver_sn,
                        log_timestamp = x.detection_timestamp_utc,
                        short_trans_id = i.short_trans_id,
                        short_rec_id = i.short_rec_id,
                        deploy_lat = i.deploy_lat,
                        deploy_long = i.deploy_long),
                 on = .(transmitter_id, detection_timestamp_utc >= start,
                        detection_timestamp_utc <= end)]

# calculate difference between when tag was transmitted
# and actual detection timestamp
dtc[, diff := abs(detection_timestamp_utc - log_timestamp)]
```

In the dtc object, we now have added two columns. The `log_timestamp` column is the time that the tag transmitted. We would expect the time difference between the recorded timestamp on the receiver and the

actual transmission to be the same if receiver clocks were perfectly synchronized at the beginning of the study and if clock drift did not occur. If timestamps were reported at millisecond resolution, differences between transmission time and detection time would also reflect travel time of the signal. The `diff` column in the dtc object is the difference between the time of tag transmission and timestamp when tag was detected. In this study, differences were less than 1 second for all all detections suggesting the time correction was successful and receiver clocks were synchronized. However we did have a couple of detections that were greater than +- 5 seconds from the tag transmission time at the beginning of the study. These occurred during the time period when receivers and tags were actively being deployed in the water and may represent wierdness that occurred while equipment was deployed. We remove these in the next code chunk.

Also in the next chunk, unique vectors of transmitter id numbers and receiver serial numbers are extracted, and a vector of equally spaced timestamps (4 hour intervals) starting on the first day of the study and ending on the last day of the study is created. Eventually, we will calculate the number of detections in each interval. The time interval chosen is user defined and can be adjusted to meet study objectives.[5] Calculating detection probability at long time intervals will smooth variability in detection probability estimates over time and short time intervals will exhibit more noise.

```
# remove un-matched detections
dtc <- dtc[!is.na(log_timestamp)]

# extract vector of all transmitters and receivers
trans_vec <- unique(dtc$transmitter_id)
rec_vec <- unique(dtc$receiver_sn)

# create time sequence by day (can be adjusted to whatever time interval by changing
# "by" argument (seconds)
tseq <- seq(from = min(dtc$log_timestamp), to = max(dtc$log_timestamp), by = 3600*4)

# bin all detections by time sequence and add bins as new column
dtc[, tbin := tseq[findInterval(dtc$log_timestamp, tseq)]]
```

`dtc` contains a new timestamp column named `tbin` that specifies the time bin that each detection falls in based on its timestamp. In the next step, we will count the number of known transmissions in each time bin for each tag.

```
# Notice "short_trans_id == short_rec_id". This line counts the number of
# actual acoustic transmissions for each tag.
# see the "Introduction to data.table" vignette for more info about grouping with data.table
exp <- dtc[short_trans_id == short_rec_id, .(exp = .N),
           by = .(tbin, transmitter_id)]
```

We calculate the number of tag transmissions that were detected in each time bin for each tag and receiver in the next code chunk. The `dtc` object only contains detections for receivers during time bins that detected a tag. We have to include all tags and receiver combinations in our analysis, even if some tag-receiver combinations were not detected in order to calculate detection probability for each tag and receiver. We accomplish this by creating an object that contains all combinations of tags, receivers, and time bins and then merging this object with the dtc object. We are essentially copying all actual detections into the table that contains all combinations of tag, time bin, and receiver combination and adding zero if a tag, time receiver was not detected. This is easily accomplished using a `data.table` join.

[5] Overall detection probability for each receiver during the trial may be calculated by specifying one interval that includes the entire duration of the study

```
# Find all combinations of tag, time bin, and receiver vectors
# Use the data.table "CJ" cross-join function to do this.
# The base function "expand.grid" provides similar functionality
# to the "CJ" function
cross_join <- CJ( transmitter_id = trans_vec, receiver_sn= rec_vec, tbin = tseq)

# count all observed detections for each time bin
obs_count <- dtc[, .(obs = .N), by = .(tbin, receiver_sn, transmitter_id)]

# join "obs_count" with "cross_join" and keep all cross_join and add NA if not detected
obs <- obs_count[cross_join, on = c("transmitter_id", "tbin", "receiver_sn")]

# change NA to 0 in "obs" object
obs[is.na(obs), obs := 0]
```

Now we simply join the observed with the expected using `tbin` and `transmitter_id` columns as keys. Detection probability is calculated for each time bin by dividing the `obs` column by the `exp` column. The `dp` column in the `dp` object is detection probability for each time interval. Also in this chunk, we added the tag-receiver distance that we calculated earlier to `dp` using `transmitter_id` and `receiver_sn` as keys in a `data.table` join.

```
# merge observed and expected by tbin and transmitter_id
dp <- exp[obs, on = c("tbin", "transmitter_id")]

# calculate detection probability and save to a new column named "dp"
dp[, dp := obs/exp]

# merge tag-receiver dist with dp.
dp <- dist_matrix[dp, on = c("transmitter_id", "receiver_sn")]
```

We have now calculated detection probability for each tag, time interval and receiver and added a column with tag-receiver distances. In the next sections, we will visualize patterns in detection probability over time and extract detection range curves.

6 Stacked plots to visualize detection probability over time

In the next section, we plot detection probability on each tag for each receiver over time and tag-receiver distance. We can then use a loess smoothing function to summarize and extract patterns in detection probability over time. This type of plot is really easy using the `ggplot2` package. The code in the next chunk will produce a stacked plot for each tag.[6]

```
# round distances for plot labels
dp[, dist_rnd := round(dist_m)]

# turn distance into a factor and sort so tag-receiver distances
# increases from small to large.
dp[, dist_f := factor(dist_rnd, levels = sort(unique(dist_rnd), decreasing = TRUE))]

# define tag groups
tag_grp <- unique(dp$transmitter_id)
```

[6]The stacked plot for only tag 62215 is included in this vignette.

```
# loop through each tag and extract data for one tag at a time
for(i in 1:length(tag_grp)){
  dp.i <- dp[transmitter_id == tag_grp[i]]

  # remove rows where dist_rnd equals 0.
  dp.i <- dp.i[dist_rnd != 0]

  # create plot of detection probability for each receiver
  p <- ggplot(data = dp.i, aes(x = tbin, y = dp)) +
    ylim(0,1) +
    geom_line() +
    facet_grid(dist_f ~ .) +
    labs(x = "time", y = "detection probability") +
    geom_smooth(method = "loess", formula = y ~ x, col = "red", span = 0.10, se = FALSE) +
    ggtitle(paste0("tag ", dp.i$transmitter_id[1]))
  print(p)
}
```

As you can see in the stacked plot, detection range decreased as distance between the receiver and tag increased (Figure 3). Also notice how detection range fluctuated through time and a lack of detections at distances greater than or equal to about 450 m. Interestingly, periods of time with good detection probability at the closest receiver to the tag (9 m) corresponded to poor detection probability on receivers located 305 – 400 meters away from the tag. Periods of time when detection probability was high on close receivers but low on receiver further away may represent time periods when poor environmental conditions resulted in rapid attenuation of the signal such that only the closest receivers were detected. Likewise, periods of high detection probability at further receivers and low detection probability at closer receivers may have occurred when signal power exceeded the detection threshold of receivers in close proximity but attenuation of acoustic signal in water allowed the signal to be detected on receivers further way.

7 Visualize detection range curves

In the next code chunk, we will extract and plot detection range curves for each time interval.[7] Detection range curves are another way to visualize detection range as a function of distance between the tag and receiver during a specific time interval. Detection range curve is an powerful tool for determining appropriate receiver spacing and an important input for simulating the probability of detecting a fish on a receiver line or in a grid of receivers.

```
# add tag-date name
dp[, id := paste(transmitter_id, tbin, sep = ", ")]

# initialize loop to plot detection probability for each tag
groups <- unique(dp$id)

# order by tags and time
setorder(dp, transmitter_id, dist_m)

par(bty="l")
for(i in 1:length(groups)){
  dp.i <- dp[id == groups[i],]
```

[7] We present one example in this vignette

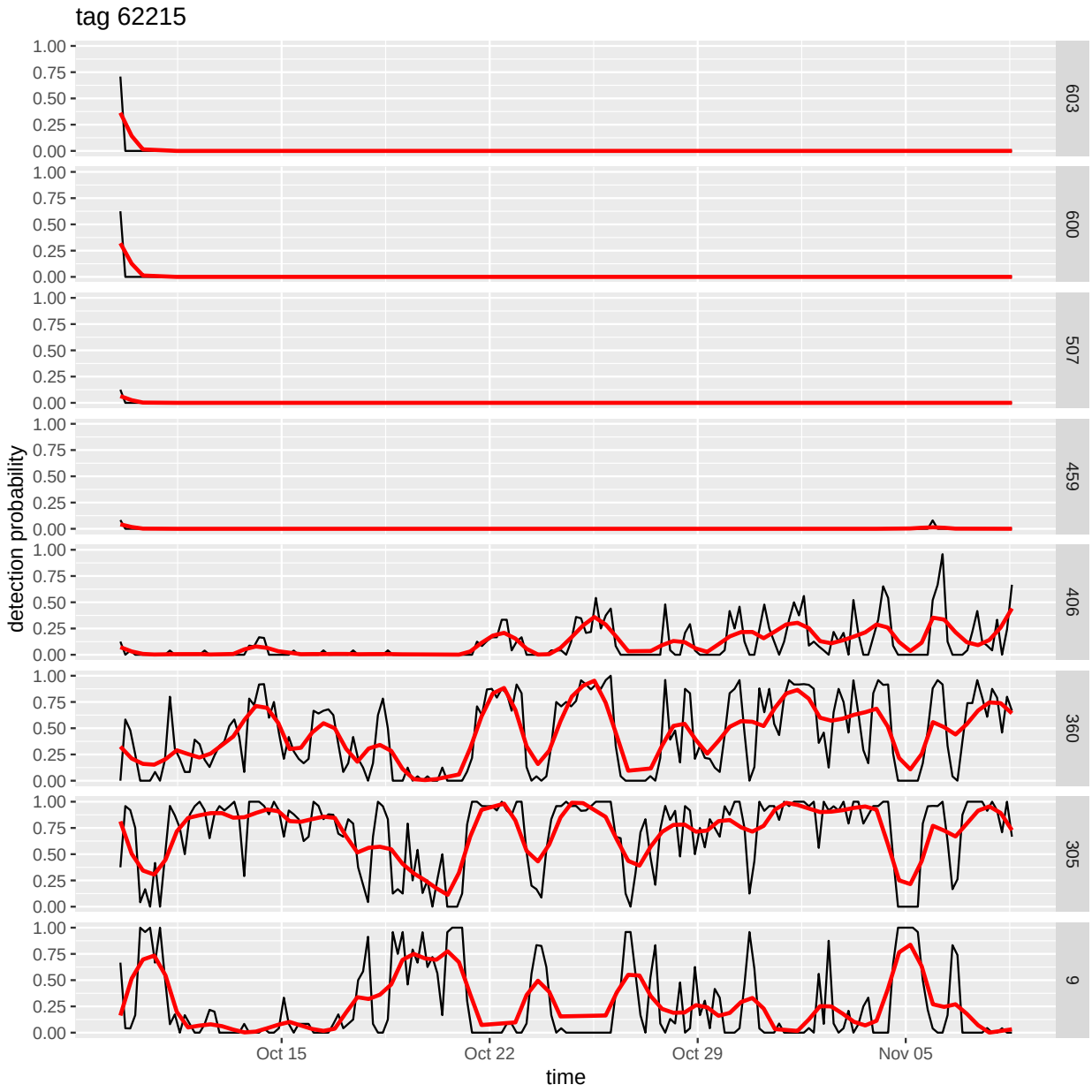


Figure 3: Probability of detecting tag 62215 during successive daily time intervals on 8 receivers located 9-603 meters from tag (right y-axis labels). Black lines in each panel is detection probability and red line is a loess smoother.

```
  # remove rows where dist_rnd equals 0.
  dp.i <- dp.i[dist_rnd != 0]

  plot(dp.i$dist_m, dp.i$dp, ylim = c(0,1), xlim = c(0,650),
       axes = FALSE, main =  paste0(dp.i$tbin[1], ", tag: ",
                                    dp.i$transmitter_id[1]),
       ylab = "detection probability",
       xlab = "tag-receiver distance (m)", pch = 16, col = "red", cex = 2, type = "o")
  axis(2, at = seq(0, 1, 0.2), las = 1)
  axis(1, at = seq(0, 1000, 200))
  axis(1, at = seq(100, 900, 200), labels = NA)
  box()
}
```

As you can see in this example, detection probability range drops dramatically after about 400 m (Figure 4). Detection range curves may be generalized with statistical approaches such as logistic regression models or generalized additive models. Up to this point, our examples have focused on estimating the probability of detecting a tag transmission and summarizing and visualizing tag detection data. However, an important consideration when analyzing telemetry data or developing a new telemetry project is to estimate the probability that a fish is detected when within range of a receiver. Some of the most critical decisions that we make about telemetry system design or interpretation of data, are not observed. Simulations can be useful for exploring the relative performance of telemetry system characteristics (e.g., receiver spacing, transmitter power, transmitter delay) or the potential consequences of unanticipated outcomes (e.g., lost receivers).

8 Predicting receiver line performance with simulation

Simulation can be a powerful tool for decision making and the `glatos` package contains several simulation-based functions. When considering simulation, it is important to keep in mind that simulations are a simplification of reality and that results are only as good as the assumptions and variables going in (i.e., 'garbage in, garbage out'). Fortunately, many processes that influence telemetry systems are fairly well understood. Our approach to these types of simulations is generally to be cautious: make conservative assumptions to yield conservative inferences.

In our next example, we will incorporate observed detection data (i.e., detection range curve) into a simulation to predict the probability that an acoustic-tagged fish will be detected on a receiver line, given constant fish velocity (ground speed), receiver spacing, number of receivers, and detection range curve. Although range detection studies are important for estimating the probability of detecting a tag transmission, they do not estimate the probability of detecting a fish that is moving through a line of multiple receivers that may detect a tag transmission.

Here, we will use the `receiver_line_det_sim()` function in the `glatos` package to determine the smallest spacing between receivers in a receiver line that will ensure that all fish are detected by that line. The function `receiver_line_det_sim()` swims a virtual tagged fish perpendicularly through a virtual receiver line to estimate the probability of detecting the fish on that receiver line, given constant fish movement rate, receiver spacing and number of receivers for a specified detection range curve. Detection of each transmission on all receivers is simulated by draws from a Bernoulli distribution using the distance between the tag and each receiver in the line based on the user-provided range detection curve. A fish is only considered detected on the line if detected more than once on a given receiver in the simulation, as single detections would not pass a false detection filter. The `receiver_line_det_sim()` requires a function that specifies a detection range curve for the `rngFun` argument. We will use empirical data to define a detection range curve from the previous example but any other function that describes the shape of the detection range curve (i.e., logistic) can be used as long as the function supplies a numeric vector of detection probabilities for an input vector of distances. Other examples of detection range curves are available in the help document for

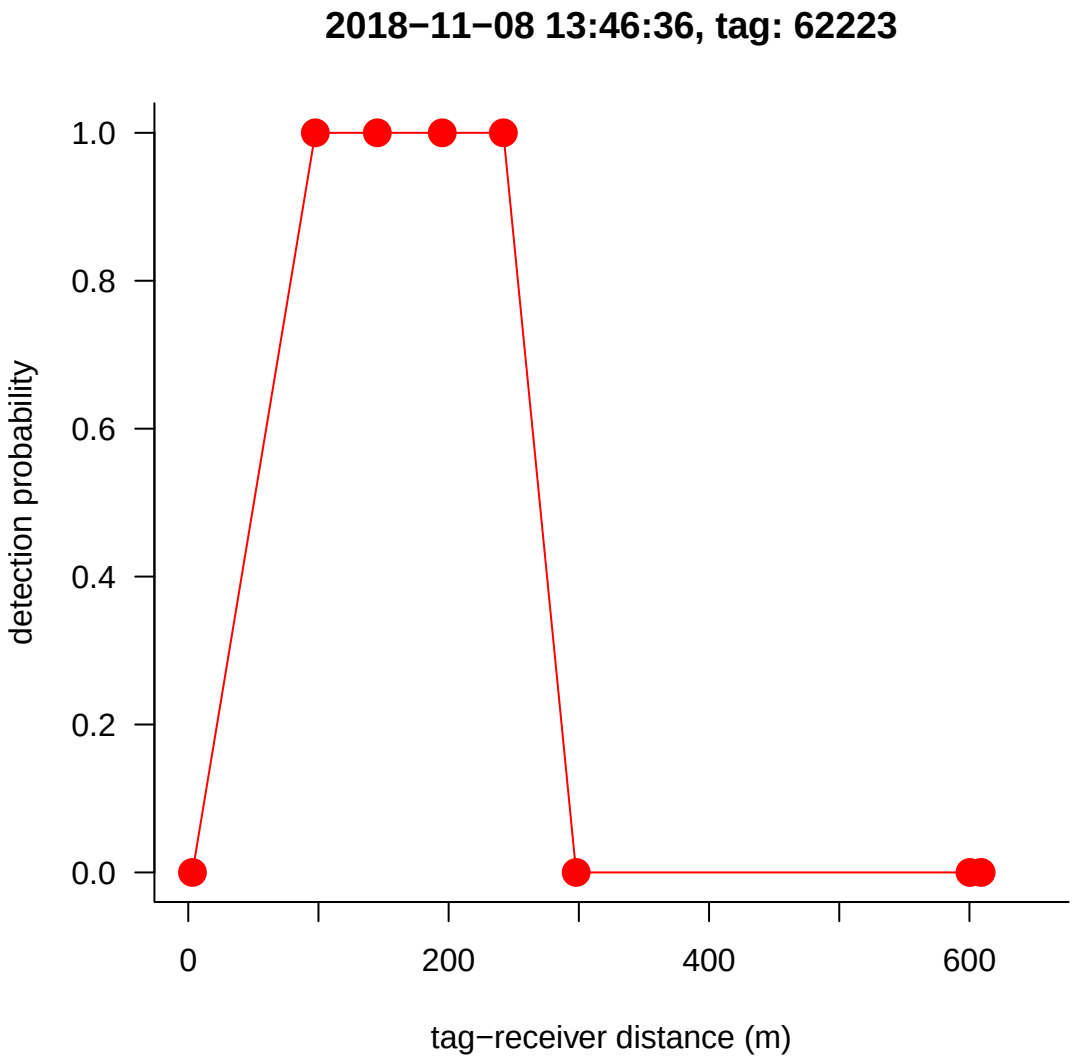


Figure 4: Example of detection range curve. Detection probability is on the y-axis and distance between tag and receiver is on the x-axis.

`receiver_line_det_sim()`. The remaining arguments for `receiver_line_det_sim()` are: `vel`, constant fish velocity; `delayRng`, a numeric vector with min and max delay for the tag; `burstDur`, duration of each coded burst; `recSpc`, a numeric vector with distances in meters between receivers; `maxDist`, max distance between fish and receiver; `outerLim`, maximum distance simulated fish are allowed to pass to the left and right of the receiver line; `nsim`, number of fish to simulate; `showPlot`, a switch specifying whether a plot of fish paths is to be returned.

Let's assume that we have detection range data (for `rngFun`) from a range test we conducted at the new receiver line location, fish swim speed (`vel`) from the literature, tag delay we want to use based on consideration of collision rates and tag life (for `delayRng`), and the burst duration of the tag from the manufacturer (for `burstDur`). We can use these parameters to estimate the probability that a fish will be detected on the receiver line, given those parameters, using the `receiver_line_det_sim()` function.

Now, let's setup and run a simulation. First, we need to extract detection range curve for one time interval created in the previous code chunk to define `rngFun`.

```
# Define detection range function (to pass as rngFun) that returns detection
# probability for given distance assume logistic form of detection range
# curve where dm=distance in meters b=intercept and slope.

# extract detection range data for tag id "62215, 2018-10-26"
# this tag id and date was chosen randomly for this example
edr <- dp[id == groups[401]]

# remove dist==0 because these are detections logged by on-board transmitter, not actual detection
edr <- edr[dist_m != 0]

# Use base::approx function to return the detection probability based on distance and linear interpolat
edrf <- function(dm, my.edr=edr){
  p <- approx(x=my.edr$dist_m, y=my.edr$dp, xout=dm, rule=2)$y
  return(p)
}

# create sequence at 50m intervals
dist_seq <- seq(0,650,by = 25)

# interpolate detection probability using the edrf function
interpolated <- edrf(dist_seq)
```

Before we run the simulation, let's preview the range curve created by `edrf`. In the previous chunk, we passed a vector of distances from 0 to 650 separated by 25 meters.

Lets take a look at the original detection range curve and interpolated range curve.

```
# plot original detection range curve
plot(edr$dist_m, edr$dp, pch = 16, col = "red", ylim = c(0,1), las = 1, cex = 1.5, ylab = "tag transmis

# add interpolated detection probabilities
lines(dist_seq, interpolated, type = "o", col = "black", pch = 16, cex = 1)
legend("topright", legend  = c("original", "interpolated"), col = c("red", "black"), lty = c(NA,1), pch
```

Now that the range detection curve is set up, we can set up the simulation. `pdrf` is supplied to `receiver_line_det_sim()` as the `rngFun` argument. For our first simulation, let's assume that we have five receivers for our new line, intended to span 1 km between an island and mainland shoreline.

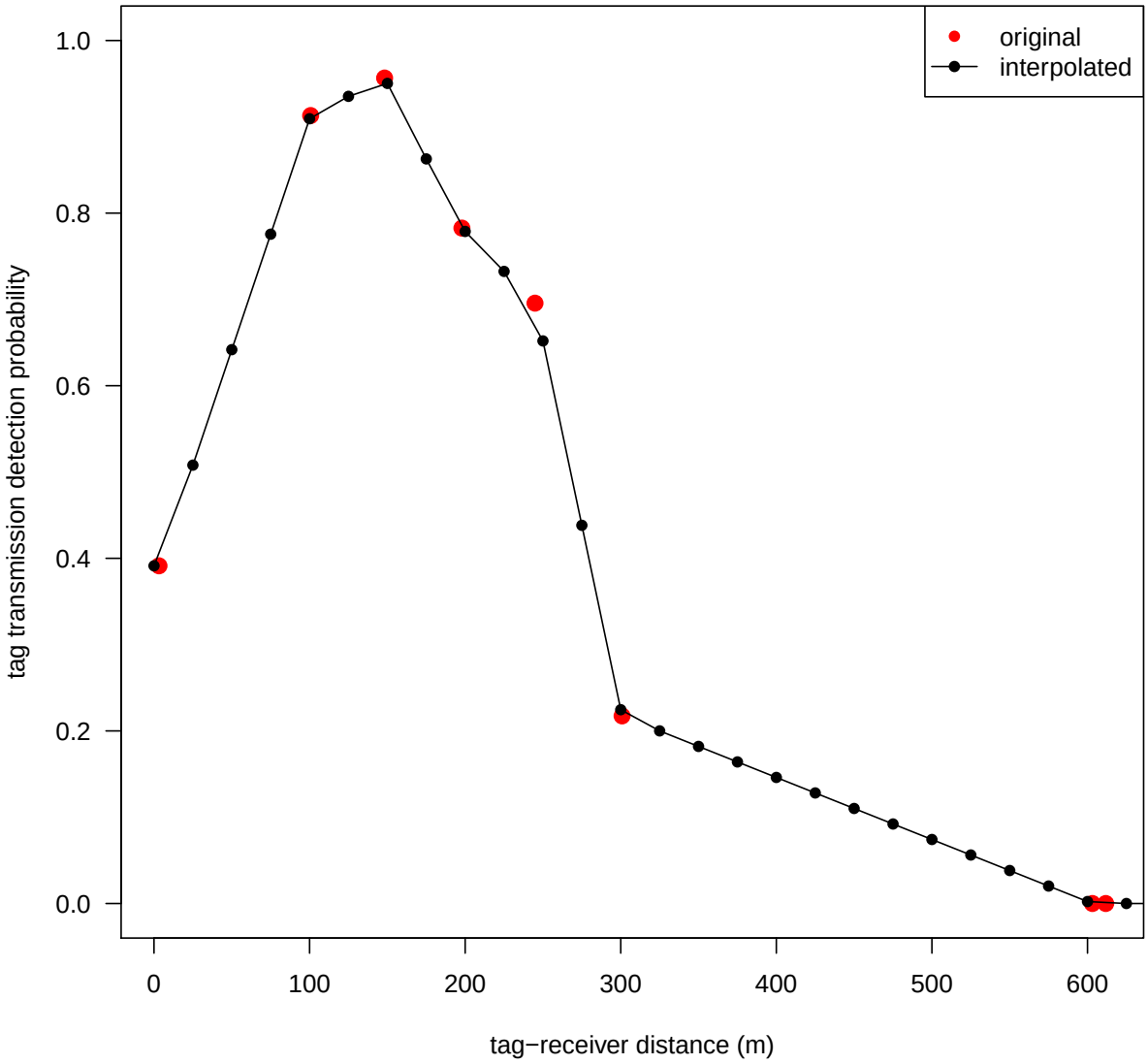


Figure 5: Detection range curve (red circles) with interpolated points (black circles and line). Detection probability is on the y-axis and distance between tag and receiver is on the x-axis.

Let's run the simulation with our five receivers spaced at 200 m (`recSpc`) and the first and last receivers 100 m from shore (`outerLim`) so that fish are allowed to pass left and right of virtual line up to 100 m. First, we will run the function with `showPlot=T` and just ten virtual fish so we can see what it's doing. Then we will run it with 10000 fish (with `showPlot=F`) to get a more robust result.

```
# Five receivers and allow fish to pass to left and right of line.
sim <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(200,5),
    outerLim=c(100, 100), nsim=10, showPlot=T)
```

```
# The `receiver_line_det_sim()` estimates line detection probability for the
# specified combination of receiver line layout and fish transmitter
# specifications. The value returned is the portion of simulated fish that
# were detected at least twice passing the receiver line (i.e., they would
# pass a false detection filter).
```

It looks like about 100% of virtual fish were detected. Now, let's repeat it with 10000 fish, but without the plot.

```
# Again but with 10000 fish and no plot.
sim <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(200,5),
    outerLim=c(100, 100), nsim=10000)
```

Results suggest that 92% of all fish should be detected under the conditions simulated. However, we might want to determine how the proportion of fish detected changes if we use more or less receivers. In order to conserve resources (i.e., receivers), we want to identify the smallest receiver spacing that would still detect about the same number of fish. We'll simply loop through a vector of receiver spacing, get the detection probability estimate for each, and then use that to guide our decision.

```
# Preallocate a vector to store estimates.
sim_out <- rep(NA, 10)

# One receivers with 500 m in between.
sim_out[1] <- receiver_line_det_sim(rngFun=edrf, recSpc=500,
 outerLim=c(500, 500), nsim=10000)

# Two receivers with 500 m in between and 250 m on each end.
sim_out[2] <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(500,2),
 outerLim=c(250, 250), nsim=10000)

# Three receivers with 333 m in between and 167 m on each end.
sim_out[3] <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(333,3),
 outerLim=c(167,167 ), nsim=10000)

# Four receivers with 250 m in between and 125 m on each end.
sim_out[4] <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(250,4),
 outerLim=c(125, 125), nsim=10000)

# Five receivers with 200 m in between and 100 m on each end.
sim_out[5] <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(200,5),
 outerLim=c(100, 100), nsim=10000)

# Six receivers at 167 m with 83.5 m on each side.
```

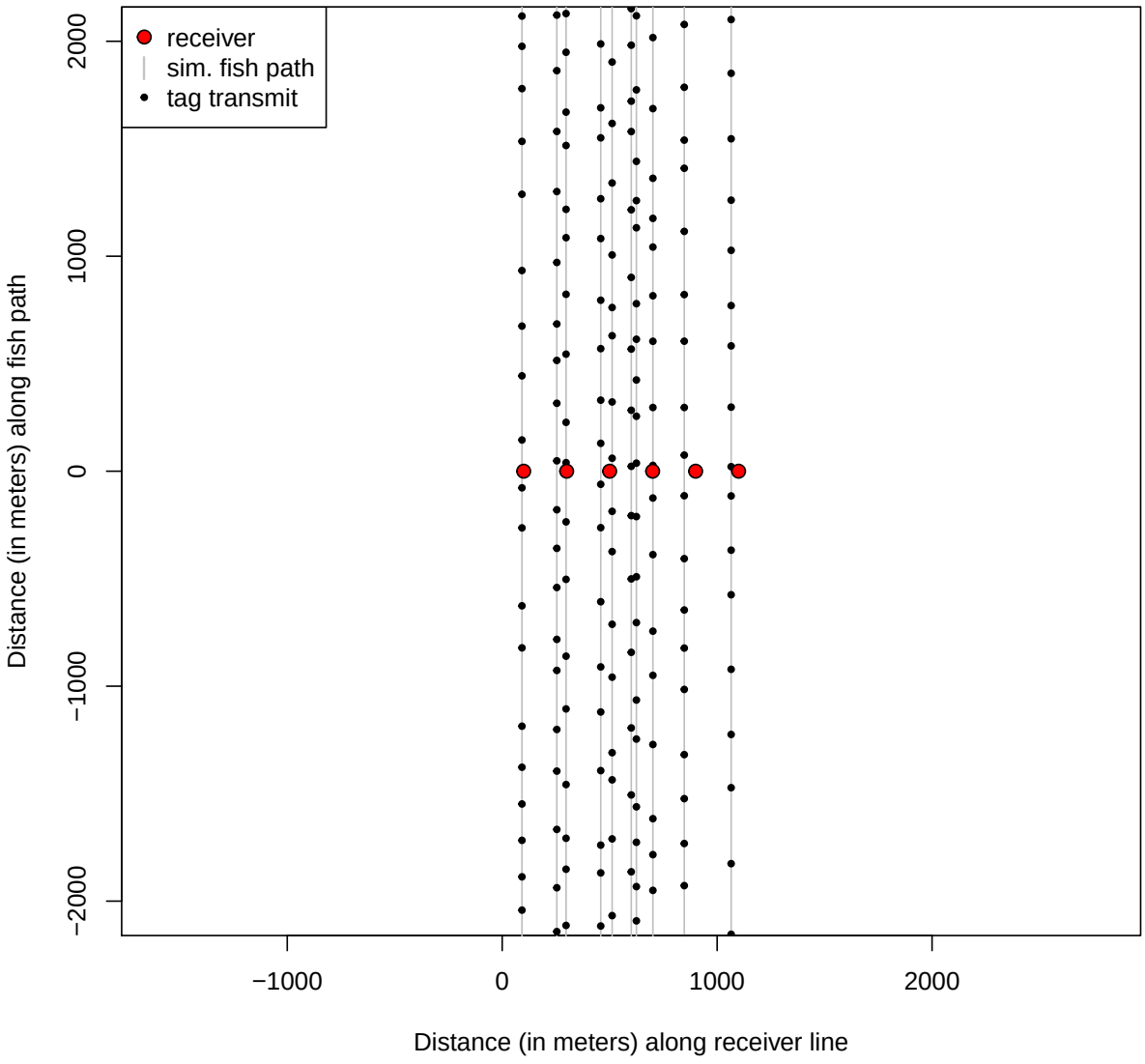
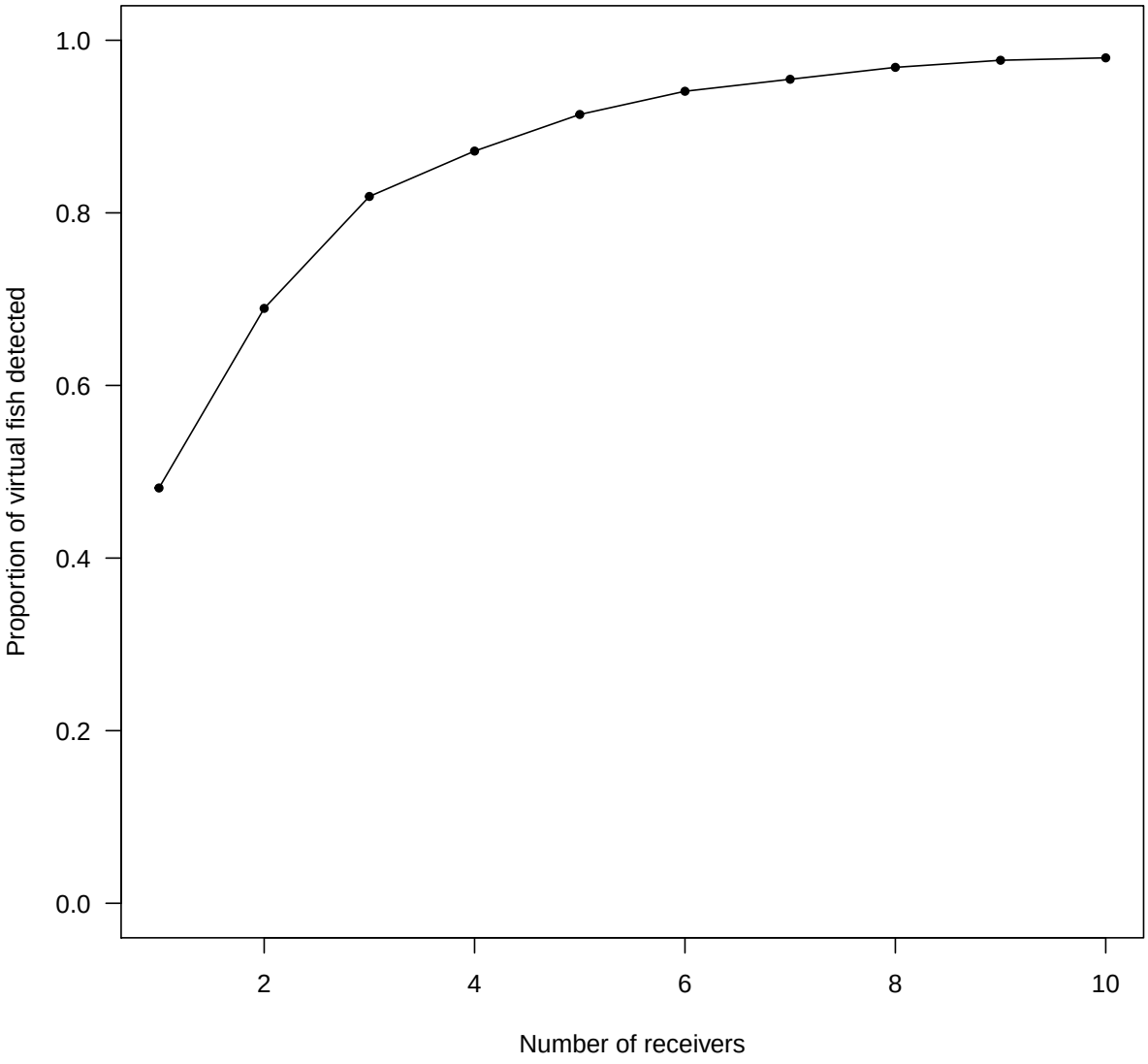


Figure 6: Paths of 10 virtual fish through receiver line

```
sim_out[6] <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(167,6),
                                    outerLim=c(83.5, 83.5), nsim=10000)

# Seven receivers at 143 m with 71.5 m on each side.
sim_out[7] <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(143,7),
                                    outerLim=c(71.5, 71.5), nsim=10000)

# Eight receivers with 125 m between and 62.5 m on each end.
sim_out[8] <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(125,8),
 outerLim=c(62.5, 62.5), nsim=10000)

# Nine receivers with 111 m between and 55.5 m on each end.
sim_out[9] <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(111,9), outerLim=c(55.5, 55.5), nsim=10000)

# Ten receivers with 100 m between and 50 m on each end.
sim_out[10] <- receiver_line_det_sim(rngFun=edrf, recSpc=rep(100,10), outerLim=c(50, 50), nsim=10000)
```

Next, we Plot the results.

```
# Plot number of receivers vs proportion of virtual fish detected.
plot(1:10, sim_out, type='o', ylim=c(0,1),
 xlab='Number of receivers',
 ylab='Proportion of virtual fish detected',
 pch=20, las = 1)
```



These results suggest that the proportion of fish detected reaches a plateau at around 5 receivers. Deploying a line with more than five receivers does not drastically increase the proportion of fish that are detected with the input parameters simulated. If we deployed 5 receivers, the percent of fish detected is 91%, but before we make our final decision, let's consider what might happen if we lost a receiver during our study, so that our line ended up having a hole in it. Again, we will simulate just ten fish with a plot to see what this looks like and then we'll run the simulation with 10000 virtual fish.

```
# Four receivers with 200, 400, 200, 2000 m in between and 100 m on each end.
dph <- receiver_line_det_sim(rngFun=edrf, recSpc=c(200,400,200,200),
                             outerLim=c(100, 100), nsim=10000)
```

Estimated detection probability dropped to about 87.7% of fish after losing a receiver, so our final decision might depend on how likely we are to lose a receiver during the study. Similar to the way we've used `receiver_line_det_sim()` here, we could simulate detection probabilities over a range of fish swim speed, tag delay, or even detection range curves to understand how results might be sensitive to our assumptions

or decisions regarding those parameters.

9 Simulation of 2-D movements with grids

In addition to the one-dimensional receiver line simulation presented above, the `glatos` package contains a set of functions that can be used to simulate fish movement and detection on receivers in a virtual two-dimensional arena.

In the final example of this vignette, we will use `crw_in_polygon`, `transmit_along_path`, and `detect_transmissions` functions to explore system performance of two different grid spacings on detection of virtual fish in Lake Huron. Although we will simulate system performance of different receiver grid configurations, these functions can be used to explore, compare, and contrast theoretical performance of a wide range of transmitter and receiver network designs. The `crw_in_polygon` function is used to simulate a random walk as a series of equal length steps with turning angles drawn from a normal distribution inside of a polygon. After random walk movement paths for virtual fish are created, the `transmit_along_path()` function is used to simulate tag signal transmissions along each path based on constant movement velocity, transmitter delay range, and duration of signal. Last, `detect_transmissions()` is used to simulate detection of transmitter signals in a receiver network based on detection range curve, location of transmitter (calculated by `transmit_along_path`), and locations of receivers.

In the first code chunk, we load an outline of the Great Lakes included in `glatos`. The outline is a SpatialPolygonsDataFrame but we are going to convert it to a simple feature object using functions in the `sf` package. [8] We then transform greatLakesPoly from a cartesian projection (WGS84- lat/lon) to a projected map using a map coordinate system for the American Great lakes region. This step simplifies the data manipulation process. Finally, in this code chunk, we extract Lake Huron from greatLakesPoly.

```
# read in polygon
data(greatLakesPoly)

# convert to simple features object
poly <- st_as_sf(greatLakesPoly)

# set attribute-geometry relationship
st_agr(poly) <- "constant"

# transform to projected map
# more info about map projection can be found at
# https://epsg.io/3175
poly <- st_transform(poly, crs = 3175)

# crop out Lake Huron
# see map coordinate conversion tool at https://epsg.io
# tool is useful for woking out bounding box in epsg 3175 coordinate
# system (NAD83/Great Lakes and St. Lawrence River)
LH <- st_crop(poly, xmin = 879714.45,
              xmax = 1263717.46, ymin = 692997.10,
              ymax = 1120461.83)

# set agr for sf
st_agr(LH) <- "constant"
```

[8] Simple features is a new R package that provides consistent and standardized functions for working with spatial data. Please see https://r-spatial.github.io/sf/ for more information.

The simulation functions from `glatos` requires unprojected (WGS 84) latitude and longitude coordinate system and SpatialPolygon objects in `sp` format . We create a new object and convert the Lake Huron polygon shapefile back to WGS 84 and `sp` in the next step.

```
# convert back to WGS84-lat/lon
LH_latlon <- st_transform(LH, crs = 4326)

# convert from sf to sp object
LH_latlon <- as(LH_latlon, "Spatial")
```

Next we set up a receiver grid in Lake Huron for use in the simulation and create a grid of virtual receivers that are spaced 10 km apart. After making the grid, we convert back to a cartesian coordinate system (i.e., WGS 84, latitude and longitude).

```
# make grid in LH polygon
# cellsize is in meters...
grid_10km <- st_intersection(LH, st_make_grid(LH, cellsize = 10000, what = "centers"))

# convert back to WGS84-lat/lon
grid_10km <- st_transform(grid_10km, crs = 4326)

# convert coordinates to sp for future use
grid_10km_sp <- as(grid_10km, "Spatial")
```

In addition to a object of receiver locations, we also need to simulate fish movements in Lake Huron. In the next chunk, we use the `crw_in_polygon` function to simulate a virtual fish movement starting from Saginaw Bay. Theta controls the turn angles that fish exhibit and tweaking can produce a fish that changes directions a lot or tend to move in a straight line. We also provide coordinates for a start point so our virtual fish starts in Saginaw Bay and a location is calculated every 10 km. We calculate positions for our virtual fish for 30 steps.

```
# do random walk in LH polygon starting in Saginaw Bay
path <- crw_in_polygon(LH_latlon, theta = c(0,25), stepLen= 10000, initHeading = 0, nsteps = 30, initPos
```

Now it is time to bring together fish track, grid, and Lake Huron outline to make sure everything looks good. This is a good time to remember that our virtual fish likely does not behave like real fish!

```
# plot path, grid and points
sp::plot(LH_latlon, col = "lightgrey", border = "grey")
points(path, type = "o", pch = 20, col = "red")
plot(st_geometry(grid_10km), pch = 20, col = "black", add = TRUE)
```

In the next step, we use `transmit_along_path` to determine where along the path the tag transmits, given the virtual fish movement rate and tag transmission specifications. For our example, we use the default values (movement velocity = 0.5 meters per second, and tag delay range from 60–180 seconds).

```
tag_trans <- transmit_along_path(path, vel= 0.5, delayRng = c(60, 180) )
```

The final step is to determine which transmissions are detected. This is done by the `transmit_along_path` function. This function uses the virtual receiver locations, tag transmission information and a detection range function to simulate detection of the fish. For this example, we are going to use the same detection range curve function used in the previous example of the line detection probability simulation. Recall that the `edrf` object in the previous line detection range simulation described the detection range curve. We will leave it up to the reader to review how this object was created.

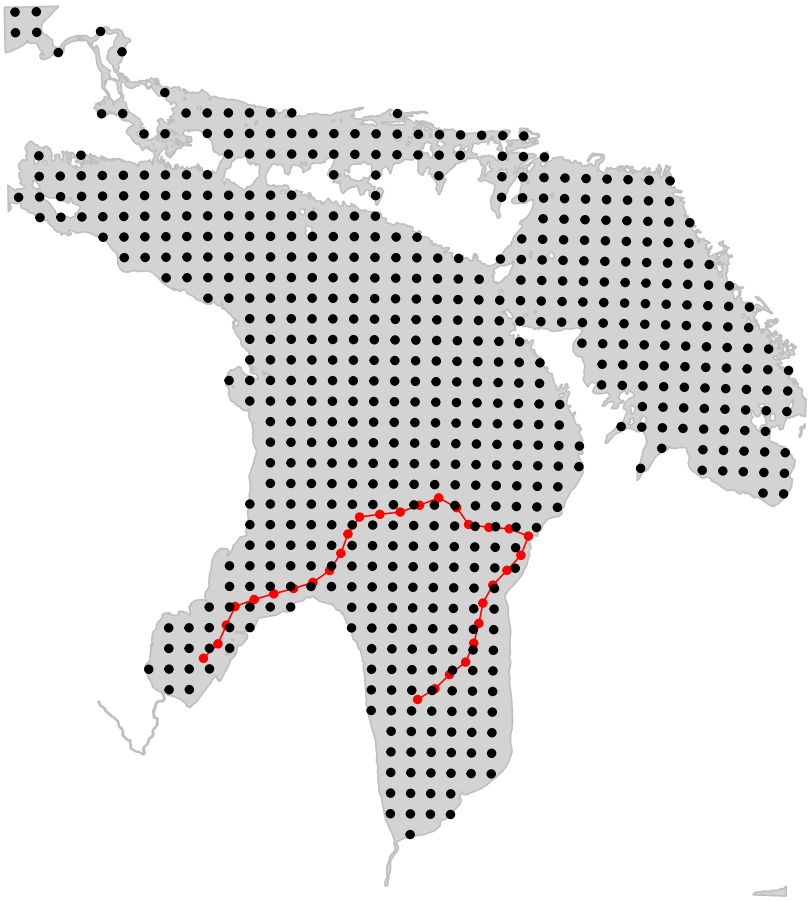


Figure 7: Plot of receiver network grid (black circles) and movement track of virtual fish (red circle and line) in Lake Huron

```
mydtc <- detect_transmissions(trnsLoc = tag_trans, recLoc = grid_10km_sp,
                              detRngFun = edrf, show_progress = FALSE )

# calculate the mean elapsed time (seconds) between detections
sim <- as.data.table(mydtc)
detinterval <- round(mean(diff(sim$etime))/60,0)

# calculate mean elapsed time difference between first detections on successive receivers
foo <- sim[, .SD[1,"etime"],  by = recv_id]
f_int <- mean(diff(foo$etime))
```

The output from the simulation consists of a spatial object that contains all locations where the fish was detected and coordinates for the receivers that detected the transmission. We can see that this virtual fish was detected 7 times on 3 receivers. The mean time interval between successive detections was 1165 minutes and the mean elapsed time difference between first detections on successive receivers was 3489 minutes.

Let's visualize our simulation. We will create a plot of the virtual fish track, receiver grid and add the points where the fish was detected.

```
sp::plot(LH_latlon, col = "lightgrey", border = "grey")
points(path, type = "o", pch = 20, col = "red")
plot(st_geometry(grid_10km), pch = 20, col = "black", add = TRUE)
points(mydtc, col = "blue", cex = 2, pch=20)
```

At this point, we have simulated detections for one virtual fish. In the next section we are going to simulate multiple fish (n=100) and compare the performance of a 5 km grid with a 10 km grid. We will keep the tag specifications, movement rates, and turn angles the same for both grid configurations. Comparison of different grid spacing is an important consideration when determing how to allocate limited number of receivers but the influence of many other parameters may also be critical for designing a grid study that meets project objectives.[9] In the first chunk, we make the 10 km and 5 km grids by adapting the code from the previous example.

```
# make 10 km grid
grid_10km <- st_intersection(LH, st_make_grid(LH, cellsize = 10000, what = "centers"))
grid_10km <- st_transform(grid_10km, crs = 4326)
grid_10km_sp <- as(grid_10km, "Spatial")

# make 5 km grid
grid_5km <- st_intersection(LH, st_make_grid(LH, cellsize = 5000, what = "centers"))
grid_5km <- st_transform(grid_5km, crs = 4326)
grid_5km_sp <- as(grid_5km, "Spatial")
```

Next, we create 100 virtual fish paths, starting in Saginaw Bay. We store each virtual fish path as an object in a list. Notice that we produce a data.frame with x and y coordinates as output for each virtual fish.

```
# make empty list
paths <- vector("list", length = 100)

# loop through and create fish paths, store each fish as item in list.
for(i in 1:100){
```

[9] see Kraus, R.T., Holbrook, C.M., Vandergoot, C.S. , Stewart, T.R., Faust, M.D. , Watkinson, D.A., Charles, C., Pegg, M., Enders, E.C., and Krueger, C.C. (2018) Evaluation of acoustic telemetry grids for determining aquatic animal movement and survival. Methods in Ecology and Evolution 2018:1-14, DOI: 10.1111/2041-210X.12996.

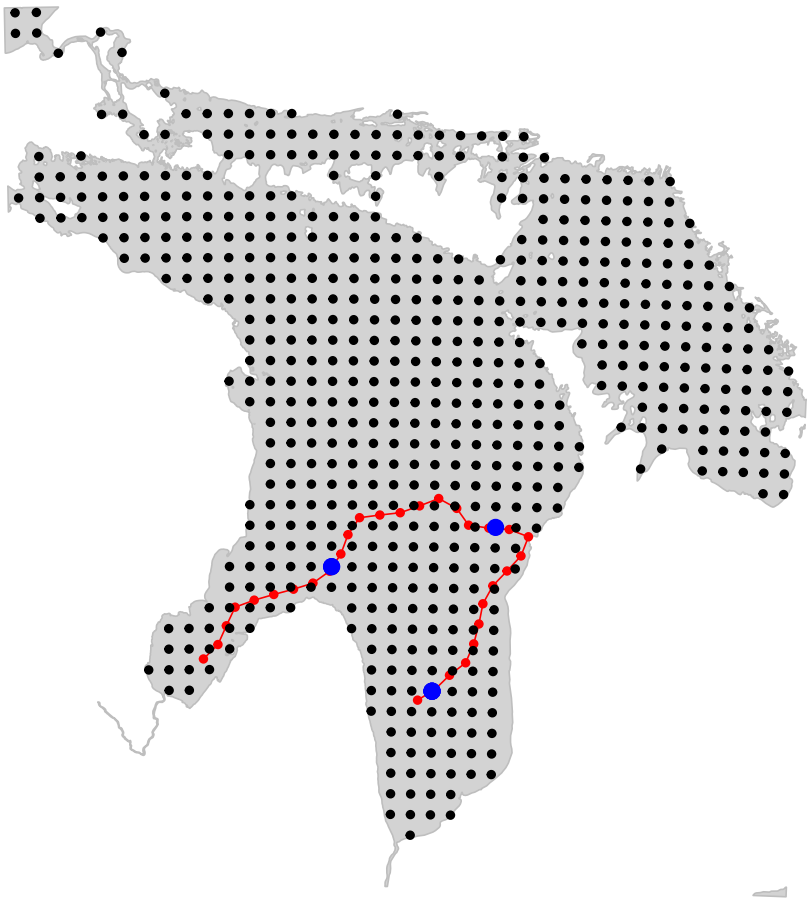


Figure 8: Plot of receiver network grid (black circles), movement track of virtual fish (red circle and line), and locations where virtual fish was detected (blue circles) in Lake Huron

```
# do random walk in LH polygon of virtual fish starting in Saginaw Bay
  paths[[i]] <- crw_in_polygon(LH_latlon, theta = c(0,25),
                               stepLen= 10000, initHeading = 0,
                               nsteps = 30, initPos = c(-83.6, 43.8), sp_out = TRUE,
                               show_progress = FALSE)
}
```

Now we are going to collapse the list into a single data.table and add a ID column for each fish.

```
# make empty list to hold results
trans <- vector("list", length = 100)

# loop through and calculate transmissions along each path
for(j in 1:100){
  trans[[j]] <- transmit_along_path(paths[[j]], vel = 0.5, delayRng = c(60,180))
}

# make empty list to hold results
sim <- vector("list", length = 100)

# loop through and calculate transmissions along each path
for(k in 1:100){
  sim[[k]] <- detect_transmissions(trans[[k]], recLoc = grid_10km_sp,
                                   detRngFun = edrf, show_progress = FALSE)

}

# convert each simulated object into a data.frame
sim <- lapply(sim, as.data.frame)

# combine list into a single data.table object
sim <- rbindlist(sim, fill = TRUE, idcol = "virt_fish")

# calculate the mean elapsed time (seconds) between detections
det_interval_sim1 <- mean(diff(sim$etime))

# calculate mean elapsed time difference between
# first detections on successive receivers by fish
out <- sim[, .SD[1,"etime"],  by = c("recv_id", "virt_fish")]
out[, t_diff := etime - data.table::shift(etime), by = virt_fish]
rec_by_dtc_sim1 <- mean(out$t_diff, na.rm = TRUE)
```

Now that we've calculated the average elapsed time difference between first detections on successive receivers for all simulated fish on a 10 km grid (2516), we are going to repeat the simulation with a 5 km grid in the next chunk.

```
# make empty list to hold results
sim2 <- vector("list", length = 100)

# loop through and calculate transmissions along each path
for(k in 1:100){
  sim2[[k]] <- detect_transmissions(trans[[k]], recLoc = grid_5km_sp,
```

```
                                    detRngFun = edrf, show_progress = FALSE)

}

# convert each simulated object into a data.frame
sim2 <- lapply(sim2, as.data.frame)

# combine list into a single data.table object
sim2 <- rbindlist(sim2, fill = TRUE, idcol = "virt_fish")

# calculate the mean elapsed time (seconds) between detections
det_interval_sim2 <- mean(diff(sim2$etime))

# calculate mean elapsed time difference between
# first detections on successive receivers by fish
sim2_out <- sim2[, .SD[1,"etime"],  by = c("recv_id", "virt_fish")]
sim2_out[, t_diff := etime - data.table::shift(etime), by = virt_fish]
rec_by_dtc_sim2 <- mean(sim2_out$t_diff, na.rm = TRUE)
```

From our simulations, we estimated the time interval between first detections on successive receivers was 2516 minutes for a 10 km grid and 822 minutes for a 5 km grid. Likewise, the mean time between detections was 3 minutes for the 10 km grid and 2 minutes for the 5 km grid. As you can see, the 5 km grid detected fish more often the the 10 km grid at the cost of many more receivers. This is one example of how simulation may be used to estimate the theoretical performance of telemetry arrays. Additionally, simulation may be a helpful tool for post-hoc evaluation of telemetry array performance.