Seed dispersal by mallards on spring migration can speed up climate-driven plant range shifts

Erik Kleyheeg, Wolfgang Fiedler, Kamran Safi, Jonas Waldenström, Martin Wikelski, and Mariëlle L. van Toor

November 29, 2018

Explanation

Here we have prepared commented R-code that we used for preparing and running the analyses used in the study "Seed dispersal by mallards on spring migration can speed up climate-driven plant range shifts". As the original code was extensive with more than 4,500 lines of code, we decided to break it down and only provide the parts relevant to replicating the preparation of movement data for the mallard migration simulator, running the simulations, and super-imposing gut retention on the simulated trajectories. To keep this file compact and digestiable, we decided to leave out parts of code that are more specific to the study system. We therefore decided not to include the sampling tool for destination locations, the identification of stopover locations, or the dispersal into wetlands.

We performed the simulations using the R-library *batchtools*, a library providing a framework to manage R jobs and statistical experiments and their results on batch computing systems. We did not include our implementation of *batchtools*, but we encourage interested readers to take a look at the package, as we think it is a convenient and powerful tool for large simulation experiments.

A note on our use of the empirical Random Trajectory Generator: At the time of writing the paper, this movement model by Technitis *et al.* has not been published yet, and so we cannot release the code for this documentation at this point. An R-package for the eRTG and the accompanying paper are in preparation. That means that at this point, the code provided in this documentation is only partly functional, as it depends on the functions that are part of the *eRTG*-package. To allow for a comprehensive documentation of our analyses, we have thus included a dataset of 1,000 randomized trajectories produced with the eRTG for working with this script in the supplementary information (file: "data/sim.tracjectories.RData"). As soon as the paper and package for the eRTG are released, the step lengths, turning angles, etc. should be possible to use with the package for simulating own trajectories.

Required packages

We have listed all packages with the version used in this document here:

```
library(momentuHMM)
library(plyr)
library(maptools)
library(ggplot2)
library(RColorBrewer)
library(lubridate)
library(geosphere)
library(circular)
library(psych)
library(MASS)
library(raster)
library(rgdal)
```

If you wish to install the packages required to run the code in this document, you can use the installation script below. It installs all packages not present on your system, with the exception of the package *rgdal*, as this package has external dependencies you might have to install first. This will strongly depend on your operating system.

```
check.packages <- c('momentuHMM', # Hidden Markov Models for animal movement data
                    'plyr', # data handling
                    'maptools', # calculating solar noon
                    'ggplot2', # plotting library
                    'lubridate', # time operations
                    'geosphere', # spherical trigonometry
                    'circular', # circular statistics (for turning angles)
                    'psych') # to include time of day in the HMM

not.installed <- !check.packages %in% rownames(installed.packages())

for(i in not.installed){
  install.packages(i)
}
```

Importing and preparing the tracking data

The first step is to import the tracking data into your R-session and prepare it for running the Hidden Markov Model using the package *momentuHMM*. The data available with the study, "mlrd.migrations.csv", has already been cleaned, i.e. we removed duplicate timestamps, non-migratory individuals, and has the missed fixes included in the data. Ground speed was provided by the tags used to track the mallards and thus did not need to be computed separately. We calculated step length (using the package *geosphere*, function *distGeo()*) and turning angle (using package *momentuHMM*, function *momentuHMM:::turnAngle*) prior to including the missed fixes in the data.

Here, we will prepare the data for applying hidden Markov models with the package *momentuHMM*. Specifically, we will demonstrate how we calculated the time of day relative to solar noon for each location.

```
# import data
df <- read.delim('data/mlrd.migrations.csv', header=T, sep=',', as.is=TRUE)

# convert column containing the timestamp to POSIX-format (all timestamps are in UTC)
df$timestamp <- as.POSIXct(df$timestamp, tz='UTC')

#  calculating time of day from longitude, latitude and timestamp

df <- ddply(df, 'ID', function(id){ # apply this function to each individual
  id$tod <- unlist(lapply(1:nrow(id), function(j){ # apply this function for every row j

    if(is.na(id$location.long[j])){return(NA)} # return NA for missed fixes

    # convert location in row j to "SpatialPoints"
    loc <- SpatialPoints(id[j,c('location.long', 'location.lat')],
                         proj4string=CRS('+proj=longlat +datum=WGS84'))

    # calculate time of solar noon for the given location
    noon <- solarnoon(loc, id$timestamp[j], POSIXct.out=TRUE)$time

     # calculate time difference between time at location and solar noon
```

```
    tod <- as.numeric(difftime(id$timestamp[j], noon, units='hours'))+12

     # calculate time of day relative to midnight
    tod <- ifelse(tod<24, tod, tod-24)
    return(tod)
  }))
  return(id)
})

# Covariates of the model (here: time of day) cannot have missing data
# we will approximate time of day for missed fixes
# to do so, we calculate the expected timestamp at the missed fix (last timestamp + 1 h)
# and we will use the the location of the previous fix
df <- ddply(df, 'ID', function(id){
  for(j in 2:nrow(id)){
    if(is.na(id$tod[j])){

      # "try" to turn the object into a SpatialPoints-object
      sp <- try(SpatialPoints(as.data.frame(id[j-1,c('location.long', 'location.lat')]),
                            proj4string=CRS('+proj=longlat +datum=WGS84')))

      # if it doesn't work (it's a missed fix), try to find the closest location
      n <- 1
      while(class(sp)=='try-error'){
        a <- j-(n+1)
        sp <- try(SpatialPoints(as.data.frame(id[a,c('location.long', 'location.lat')]),
                                proj4string=CRS('+proj=longlat +datum=WGS84')))
        n <- n+1
      }

      # once the closest location is found, calculate time of day
      noon <- solarnoon(sp, id$timestamp[j], POSIXct.out=TRUE)$time
      tod <- as.numeric(difftime(id$timestamp[j], noon, units='hours'))+12
      id$tod[j] <- ifelse(tod<24, tod,tod-24)
    }
  }
  return(id)
})

# check for missing entries after this approximation of time of day for missed fixes:
any(is.na(df$tod))
```

```
## [1] TRUE
```

```
# there still is on e missing tod for individual JC75963
# this is the very first location for this individual, so we remove it from the data:
df <- df[!is.na(df$tod),]


# convert dataframe to "momentuHMMData" using the momentuHMM conversion function
df <- momentuHMM:::momentuHMMData(df)
#df$sex <- factor(df$sex)
```

Applying Hidden Markov Models using momentuHMM

Now that we have included the time of day in the dataset, we can define the starting parameters for the Hidden Markov model, and apply the model to the data. Here, we use the starting parameters that we used for our final model. Again, in our final model we used three data streams (ground speed, turning angle, and step length), and used time of a day as a covariate for transition probabilities (ducks often start migrating at dusk/during the night). We previously determined heuristically that the best fitting distributions for the data streams were:

- Ground speed (in m/s): Gamma distribution
- Turning angle (in radian): Wrapped Cauchy distribution
- Step length (in km): Weibull distribution

```
# starting parameters for ground speed:
par.speed <- c(0.01, 2.00, 20.00, # mean ground speed for states 1, 2, and 3
               0.10, 1.00, 5.00, # s.d. of ground speed for states 1, 2, and 3
               0.10, 0.00, 0.00) # zero-mass

par.angle <- c(pi, pi, pi, # mean angle for states 1, 2, and 3
               0.5, 0.5, 0.95) # concentration for states 1, 2, and 3

par.step <- c(0.10, 1.00, 2.50, # shape parameter for states 1, 2, and 3
              0.01, 5.00, 70.00, # scale parameter for states 1, 2, and 3
              0.10, 0.00, 0.00) # zero-mass


# prepare design matrix using
par.final <- getParDM(data=df, # the data to which to apply the HMM
                      nbStates=3, # number of states
                      dist=list(step='weibull', # distributions for the three data streams
                                speed='gamma',
                                angle='wrpcauchy'),
                      Par=list(step=par.step, # starting parameters
                               speed=par.speed,
                               angle=par.angle),
                      estAngleMean=(list(angle=T))) # estimate the mean turning angle

# run Hidden Markov Model (this can take a while)
hmm.final <- fitHMM(data=df,
                    nbStates=3,
                    dist=list(step='weibull', speed='gamma', angle='wrpcauchy'),
                    Par=par.final,
                    estAngleMean=(list(angle=T)),
                    formula=~cosinor(tod, period=24)) # use time of day as covariate
```

```
## ============================================================================
## Fitting HMM with 3 states and 3 data streams
## ----------------------------------------------------------------------------
##  step ~ weibull(shape=~1, scale=~1, zeromass=~1)
##  speed ~ gamma(mean=~1, sd=~1, zeromass=~1)
##  angle ~ wrpcauchy(mean=~1, concentration=~1)
##
```

```
##  Transition probability matrix formula: ~cosinor(tod, period = 24)
##
##  Initial distribution formula: ~1
## ============================================================================
## DONE
```

```
# print model summary
#print(hmm.final)

# plot the model
#plot(hmm.final)

# plot stationary state probabilities as a function of time of day:
#plotStationary(hmm.final, plotCI=T)

# you can also plot the individual tracks annotated with state identity
#plot(hmm.final, plotTracks=T, animals=unique(df$ID))
```

The model above might differ slightly form the final model we used for our study. We have thus included our final model in the supplementary dat folder as well (file: "data/20180605_hmm_final.RData"), and will continue this example with the model we used, rather than the model presented above.

Preparing the tracking data for the eRTG

With the finished HMM, we can inform our tracking data with the most likely sequence of states, for which we used the Viterbi algorithm included in the *momentuHMM* package. Assuming that state identity does not change within a GPS-burst, we will use the state identities from the model data (which was subsampled from the full, bursted data) to assign state identity to the full, bursted data. We will then retain only bursts assigned with state 3, corresponding to flight, to compute step lengths and turning angles and prepare the data for informing an initial eRTG at the highest sampling rate of 1 second.

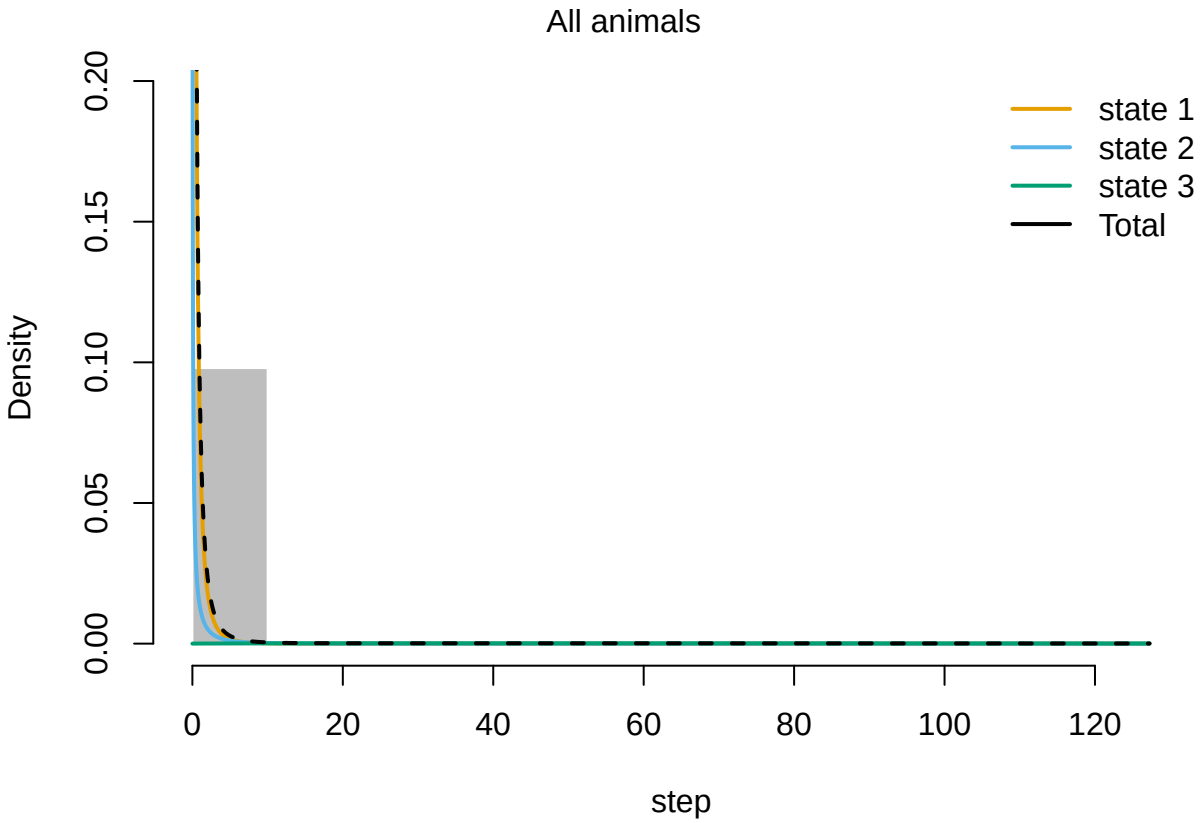
```
# import model used in the study
# the file contains the model, as well as the data and parameters used for the model
load('data/20180605_hmm_final.RData')

print(hmm.final)
```

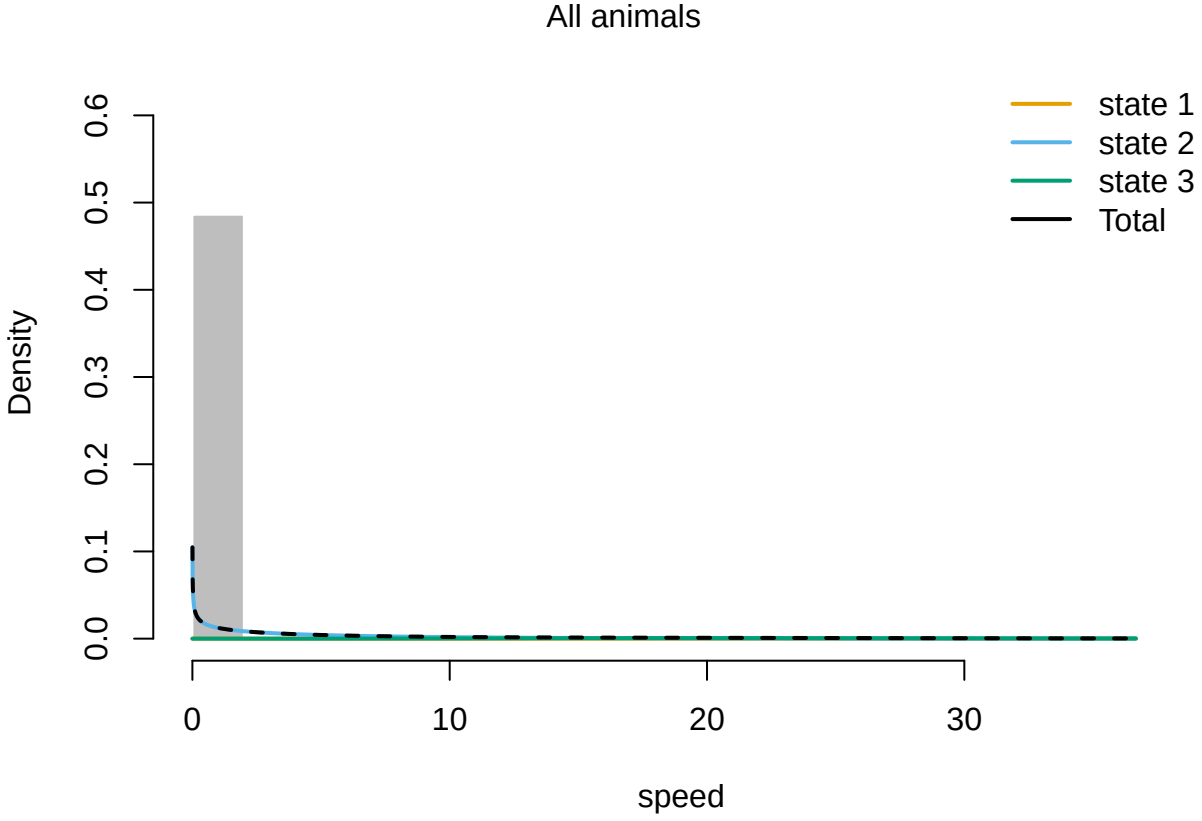
```
## Value of the maximum log-likelihood: 5203.765
##
##
## step parameters:
## ----------------
##               state 1      state 2      state 3
## shape    4.580594e-01 5.069955e-01 1.458548e+00
## scale    1.133653e-01 5.542915e-01 8.699548e+01
## zeromass 4.303736e-08 9.987223e-09 9.997041e-09
##
## speed parameters:
## -----------------
##               state 1      state 2      state 3
## mean     2.777778e-01 5.064855e+00 24.751876548
## sd       1.674406e-09 6.322630e+00 10.316327709
## zeromass 8.474216e-01 1.012314e-08  0.001792409
```

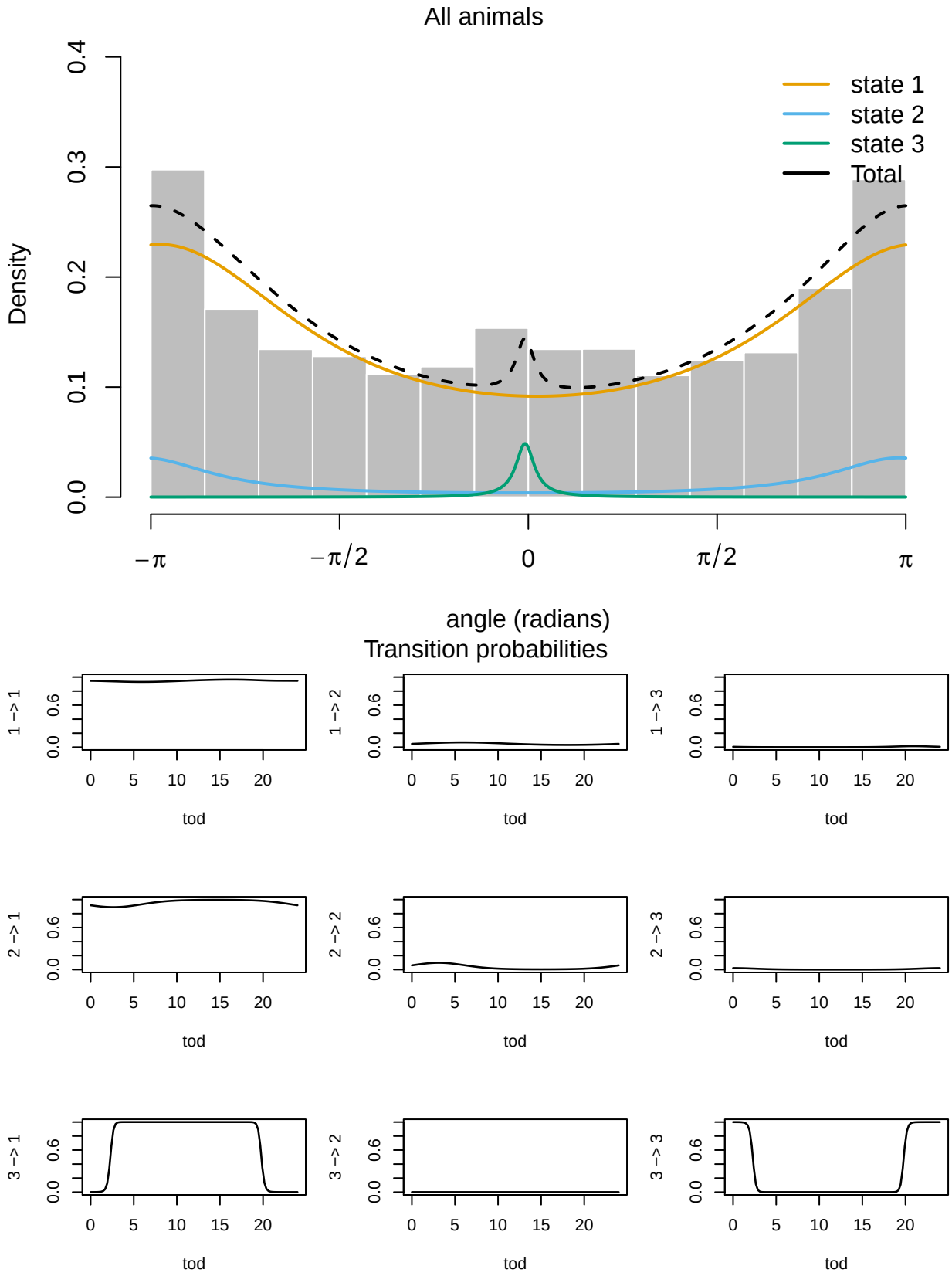
```
##
## angle parameters:
## -----------------
##                   state 1  state 2     state 3
## mean          -3.0669764 3.084454 -0.02738578
## concentration  0.2256364 0.503641  0.91061213
##
## Regression coeffs for the transition probabilities:
## ---------------------------------------------------
##                                    1 -> 2    1 -> 3    2 -> 1    2 -> 3
## (Intercept)                   -2.999637116 -7.385839  3.873347 -2.175060
## cosinorCos(tod, period = 24)  0.004752597  2.219863 -1.147285  1.130765
## cosinorSin(tod, period = 24)  0.390429225 -2.274796 -1.193073 -1.102154
##                                  3 -> 1    3 -> 2
## (Intercept)                    18.66926 -6.340627
## cosinorCos(tod, period = 24) -27.36672 -8.854909
## cosinorSin(tod, period = 24)   6.97844  4.506875
##
## Transition probability matrix (based on mean covariate values):
## ---------------------------------------------------------------
##           state 1      state 2      state 3
## state 1 0.9544144 4.550410e-02 8.153494e-05
## state 2 0.9939012 5.857734e-03 2.411027e-04
## state 3 1.0000000 1.761394e-19 2.334770e-20
##
## Initial distribution:
## ---------------------
##                 state 1   state 2      state 3
## ID:JC74440 0.000116785 0.9998832 4.170746e-18
## ID:JC75963 0.000116785 0.9998832 4.170746e-18
## ID:JC79575 0.000116785 0.9998832 4.170746e-18
## ID:JC79706 0.000116785 0.9998832 4.170746e-18
## ID:JC79712 0.000116785 0.9998832 4.170746e-18
## ID:JC79719 0.000116785 0.9998832 4.170746e-18
## ID:JC79727 0.000116785 0.9998832 4.170746e-18
## ID:JC79736 0.000116785 0.9998832 4.170746e-18
```

```
plot(hmm.final)
```

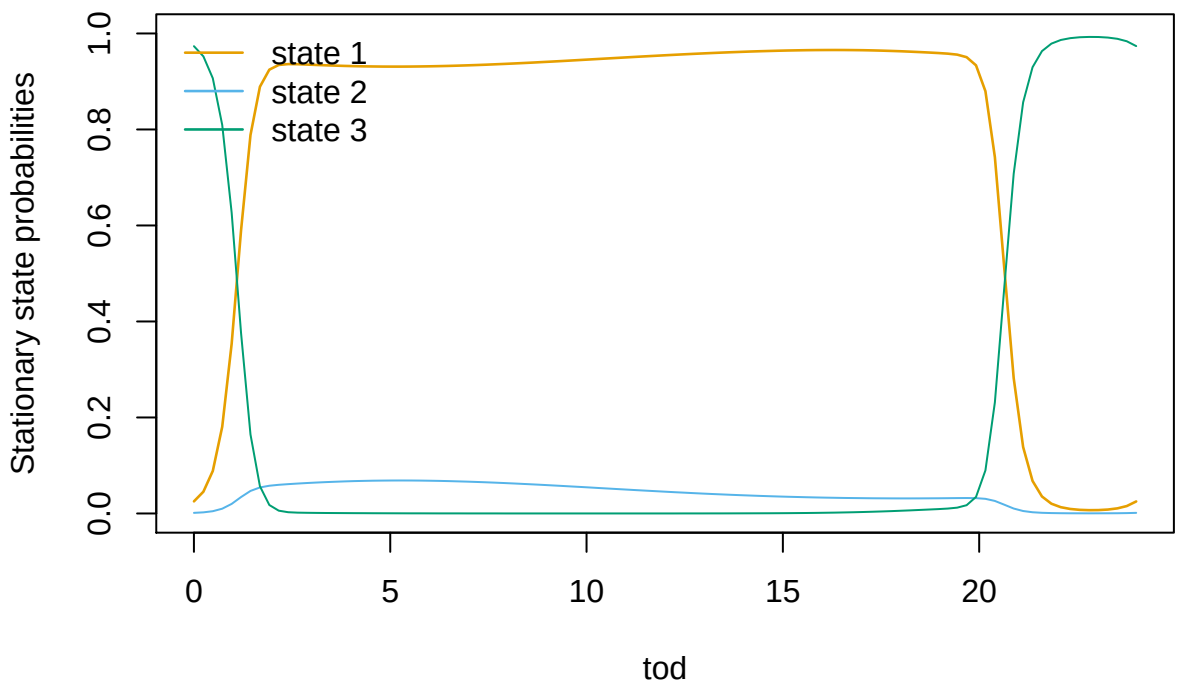
```
## Decoding state sequence... DONE
```


All animals
Density
0.00
0.05
0.10
0.15
0.20
0
20
40
60
80
100
120
step
state 1
state 2
state 3
Total


All animals
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
10
20
30
speed
state 1
state 2
state 3
Total




```
plotStationary(hmm.final, plotCI=T)
```




```
# import full data set containing the bursted data
load('data/mlrd.clean.RData')

# use Viterbi algorithm to decode most likely state sequence
hmm$state.identity <- viterbi(hmm.final)

# because each of the GPS-bursts has a unique identifier, we can transfer state identities
# of bursts to the full, bursted data set
# only keep individuals present in the model
mlrd <- mlrd[mlrd$individual.local.identifier %in% unique(hmm$individual.local.identifier),]
mlrd$state <- NA

# apply the following function to each individual:
mlrd.state <- ddply(mlrd, 'individual.local.identifier', function(id){
  ddply(id, 'burst', function(b){ # apply this function to each burst
    d.id <- unique(b$individual.local.identifier) # individual identifier
    b.id <- unique(b$burst) # burst identifier

    # find the state identity for the corresponding individual & burst
    state <- hmm$state[hmm$individual.local.identifier==d.id & hmm$burst==b.id]
    if(length(state)==1){
      b$state <- state # assign state identity to burst
    }
    return(b)
  })
})

# we will now calculate a new data set containing:
```

```r
# distance between subsequent locations (as predecessor to step length)
# time elapsed between subsequent locations
# turning angles between subsequent locations
# all of these will be calculated separately for each individual & burst

# the following function is applied to each individual separately
step.turn <- ddply(mlrd.state, 'individual.local.identifier', function(id){
  print(unique(id$individual.local.identifier)) # print individual identifier

  if(all(is.na(id$state))){return(NULL)} # this should not apply, but just in case

  # check whether the individual tracking data contains any bursts identified as state 3
  if(any(id$state==3)){

    # only retain bursts identified as state 3
    short <- id[id$state==3 & !is.na(id$state),]

    # apply the following function separately for each burst:
    ddply(short, 'burst', function(b){
      # we need a minimum of 3 consecutive fixes to calculate turning angles:
      if(nrow(b>3)){

        # calculate time difference between fixes (in seconds)
        tdiff <- c(NA, unlist(lapply(2:nrow(b), function(j){
          difftime(b$timestamp[j], b$timestamp[j-1], unit='secs')
        })))

        # calculate spatial distance between fixes (in meters)
        dist <- c(NA, unlist(lapply(2:nrow(b), function(j){
          distGeo(b[j,c('location.long', 'location.lat')],
                  b[j-1,c('location.long', 'location.lat')])
        })))

        # calculate turning angles (in radians)
        turn <- c(NA, unlist(lapply(2:(nrow(b)-1), function(j){
          momentuHMM:::turnAngle(b[j-1,c('location.long', 'location.lat')],
                                 b[j,c('location.long', 'location.lat')],
                                 b[j+1,c('location.long', 'location.lat')],
                                 type='LL', angleCov=F)
        })), NA)

        # combine in a new data.frame and return from function
        return(data.frame(tdiff=tdiff, dist=dist, turn=turn))
      }
    })
  }
})
```

```
## [1] "JC74440"
## [1] "JC75963"
## [1] "JC79575"
## [1] "JC79706"
## [1] "JC79712"
## [1] "JC79719"
```

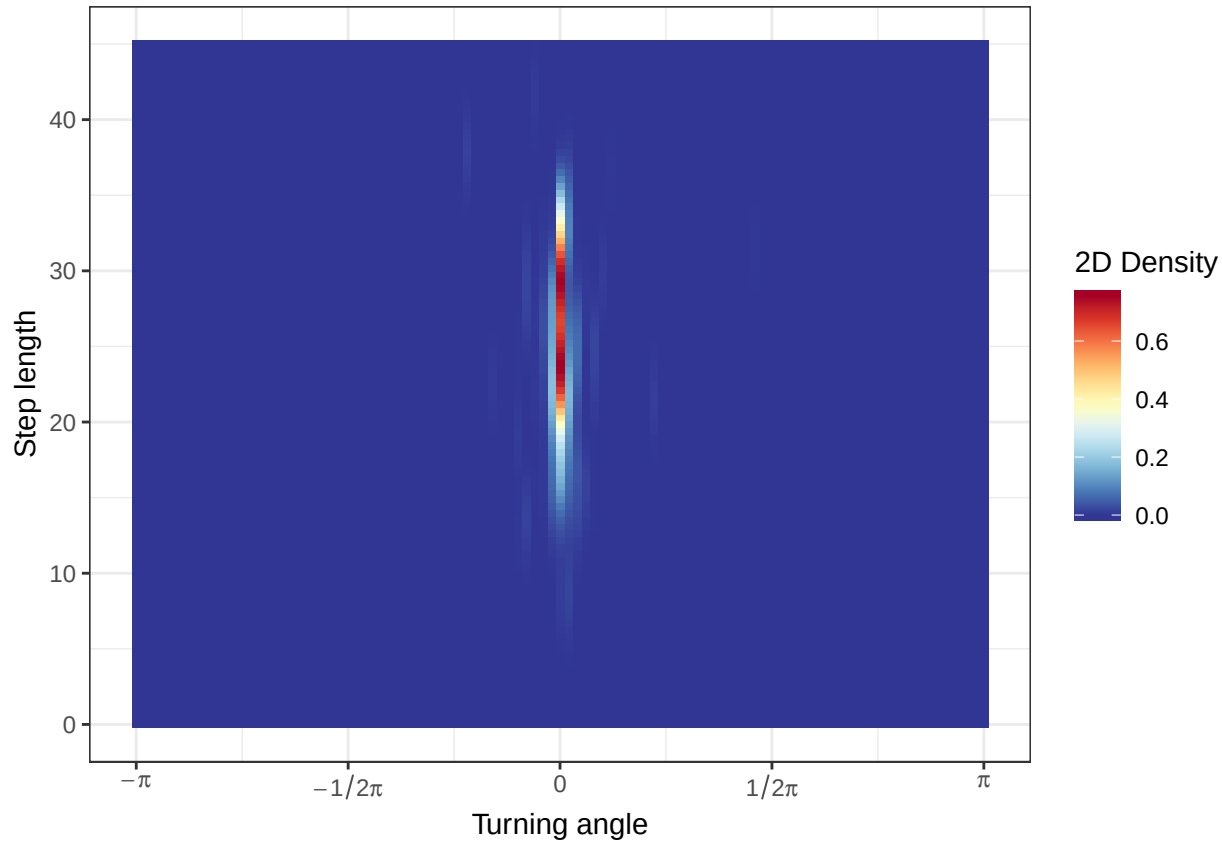
```
## [1] "JC79727"
## [1] "JC79736"
```

```
# now we calculate autocorrelation of step length and turning angle at a lag of 1 step
# (i.e., how do step length and turning angle change from one location to the next?)

# apply function separately to each individual
step.turn <- ddply(step.turn, 'individual.local.identifier', function(id){

  # apply function separately to each burst
  ddply(id, 'burst', function(b){

    # calculate step length as distance per unit time
    b$step <- b$dist/b$tdiff

    # calculate lag at 1 step for step length and turning angle
    b$step.lag <- c(NA, diff(b$step, lag=1))
    b$turn.lag <- c(NA, diff(b$turn, lag=1))
    return(b)
  })
})
```

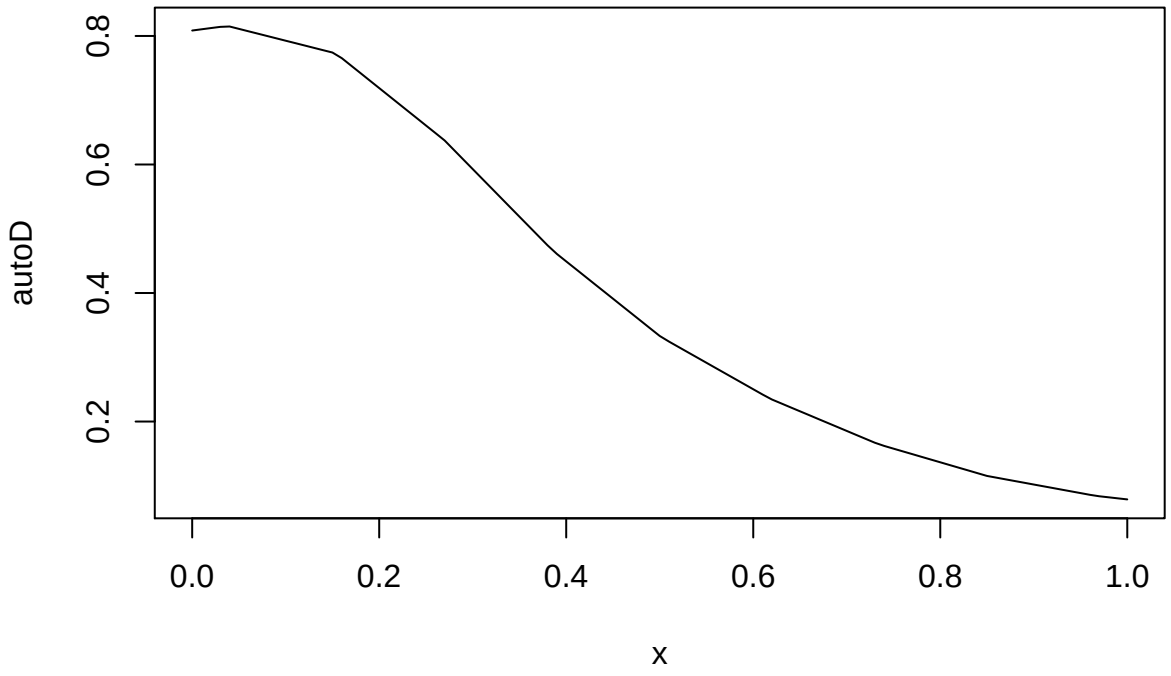
Initial eRTG with 1 second sampling rate

We used the step.turn data.frame as derived above to derive the initial version of the eRTG:

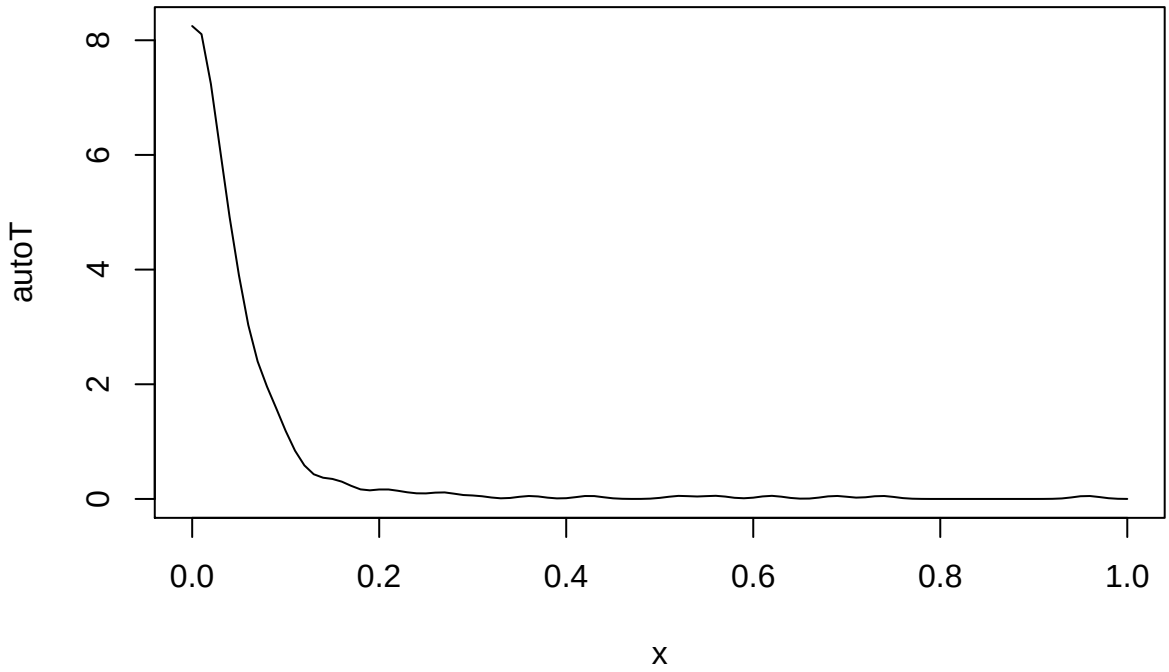
```
# only return complete cases
step.turn <- step.turn[complete.cases(step.turn),]

# we made the simple assumption that mallards fly at speeds >1.5 m/s
step.turn <- step.turn[step.turn$step>1.5,]

# this function is part of the eRTG, and thus not yet available:
#rasterDT <- StepITurnHist(x=step.turn$turn, y=step.turn$step)
#image(rasterDT)

# for visualisation purposes, we'll approximate what the function as follows:
# compute a 2D-kernel density estimator for step length and turning angle
st <- kde2d(x=step.turn$turn, y=step.turn$step, lims=c(-pi, pi, 0, 45), n=101)
st <- raster(st)
st.df <- data.frame(x=coordinates(st)[,1], y=coordinates(st)[,2], value=values(st))
ggplot(st.df, aes(x=x, y=y, fill=value)) +
  geom_raster() +
  scale_fill_gradientn(name='2D Density', colours=rev(brewer.pal(11, 'RdYlBu'))) +
  labs(x='Turning angle', y='Step length') + theme_bw() +
  scale_x_continuous(breaks=c(-pi, -pi/2, 0, pi/2, pi),
                     labels=expression(-pi, paste(-1/2,pi), 0, paste(1/2, pi), pi))
```




```
# approximate the autocorrelation in step length & turning angle:
autoD <- approxfun(density.default(step.turn$step.lag))
autoT <- approxfun(density.default(step.turn$turn.lag))
plot(autoD)
```



```r
plot(autoT)
```




```r
# combine objcts into initial 1-s mallard flight eRTG
#st.1s <- list(dtRaster=rasterDT, autoD=autoD, autoT=autoT)

# running an initial, unconditional trajectory:
#sim.even <- simm.uncond(250000, start=c(0,0), a0=runif(1, -1*pi, pi), densities=st.1s)
```

Thinning the initial eRTG

The movement model we used for this study was the **empirical Random Trajectory Generator**, developed by Technitis et al. This movement model is a conditional model similar to a biased correlated random walk that can be best described as a mean-reverting Ornstein-Uhlenbeck process. As mentioned in the explanation above, the code for this model is, at this date, not yet available, but will be published as an R-package with accompanying paper. We here provide part of the long trajectory we simulated with the unconditional eRTG (we provide only a subset as the original file was 118 MB), and thin this to a reduced sampling rate of 300 seconds. We calculate step lengths, turning angles, etc. for the eRTG with a sampling rate of 300 seconds.

```r
# load parameters required for the eRTG
# derived from the step length and turning angles calculated above
load('data/sim.long.1s.subset.RData')

# world azimuthal equidistant projection
# for calculating distances from long unconditional trajectory
proj.az <- CRS('+proj=aeqd +lat_0=0 +lon_0=0 +x_0=0 +y_0=0
               +ellps=WGS84 +datum=WGS84 +units=m +no_defs')

# now we subsample this trajectory by thinning it to every 300th entry
# to make sure that this process isn't sensitive to the starting location,
# we randomised the starting location several times, and subsample from there
rndm <- rbind.fill(lapply(1:100, function(j){ # randomise starting location 100 times
  if(j %in% seq(0,100,10)){print(j)}
```

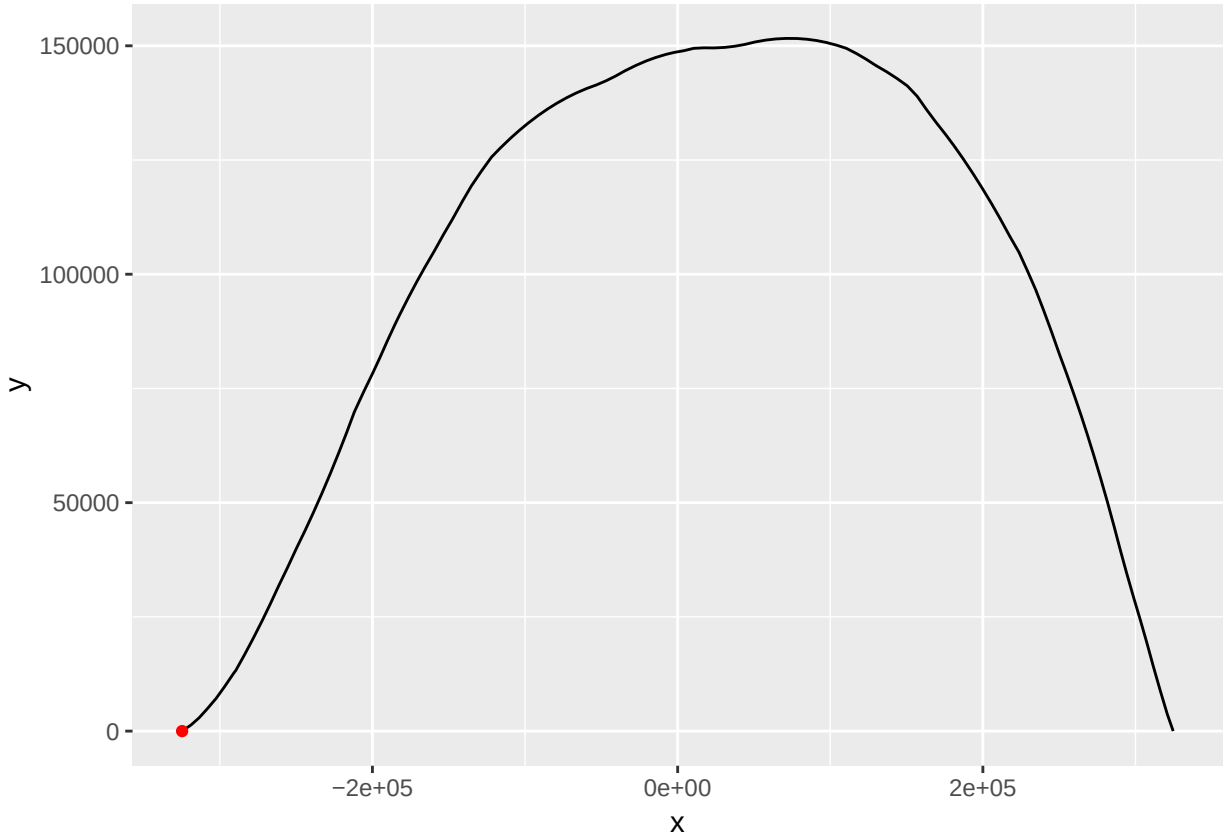
```
  # pick a random starting location out of the first 300 locations
  smpl <- rep(F, 300)
  smpl[sample(length(smpl), 1)] <- T

  # then pick every 300-th location following the randomised starting location
  short <- sim[rep(smpl, length.out=nrow(sim)),]

  # recalculate distances & turning angles
  po <- spTransform(SpatialPoints(short[,c('x', 'y')],
                    proj4string=proj.az), CRSobj=CRS('+proj=longlat +datum=WGS84'))
  short$d <- c(NA, unlist(lapply(2:nrow(short), function(j){
    distGeo(po[j,], po[j-1,])
  })))
  short$a <- c(NA, unlist(lapply(2:nrow(short), function(j){
    bearing(po[j,], po[j-1,])
  }))) * pi/180
  short$t <- c(NA, wrap(diff(short$a * pi/180)))
  short$lag.d <- c(NA, diff(short$d, lag=1))
  short$lag.t <- c(NA, diff(short$t, lag=1))
  short <- short[-c(1:2),]
  short$repl <- j
  return(short)
}))

# prepare eRTG with sampling rate of 300 seconds:
# make SteplTurnHist
rasterDT <- SteplTurnHist2(x=short$t, y=short$d) # function from eRTG
autoD <- approxfun(density.default(short$lag.d))
autoT <- approxfun(density.default(short$lag.t))
st.300 <- list(dtRaster=rasterDT, autoD=autoD, autoT=autoT)

# run a long unconditional track for the 5-min-eRTG
#sim.300 <- simm.uncond(250000, start=c(0,0), a0=runif(1, -1*pi, pi), densities=st.300)
```

Simulating random trajectories using the eRTG

As we cannot demonstrate the actual simulation of randomised trajectories using the eRTG, we have included a data set containing 1,000 randomised trajectories from our full simulation data set for exploration:

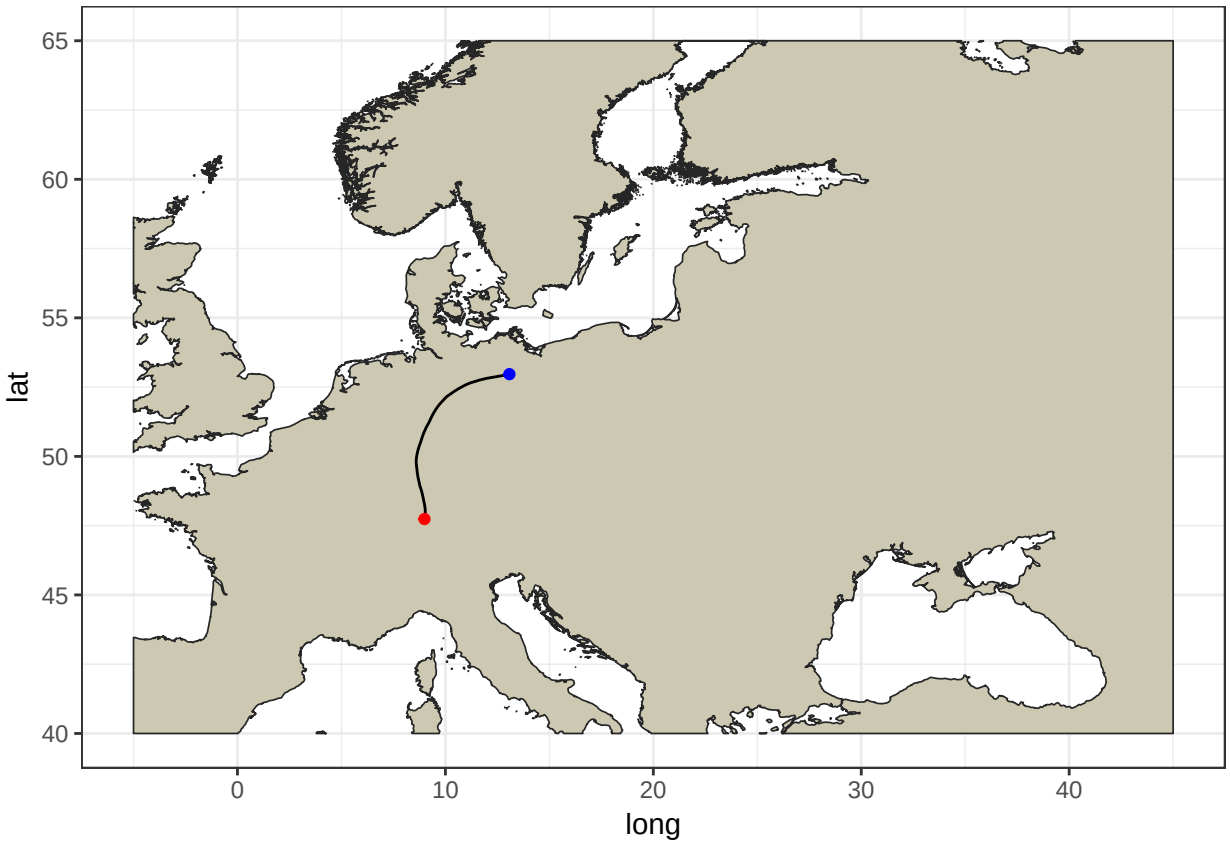
```
# tracks were simulated between two fixed locations:
# Lake Constance (starting location)
# breeding area, sampled with destination sampling tool

# prior to each simulation, we projected the corresponding location pair
# to a two-point equidistant projection:
# proj.tpe <- CRS(paste0("+proj=tpeqd +lat_1=", start@coords[,2], " +lon_1=", start@coords[,1],
#                        " +lat_2=", destination@coords[,2], " +lon_2=", destination@coords[,1],
#                        " +x_0=0 +y_0=0 +a=6371000 +b=6371000 +units=m +no_defs"))

# load random tracks:
load('data/1000randomtracks.RData')
# this object is a list containing 1,000 data.frames, each corresponding to a random track
```

```
# you can plot single trajectories as:

i <- 12
ggplot(tracks[[i]]$track, aes(x=x, y=y)) + geom_path() +
  geom_point(data=tracks[[i]]$track[1,], colour='red') # mark starting location
```




```
# all tracks are projected (species two-point equidistant projections)
# we can re-project all of them to long-lat:

tracks <- lapply(1:length(tracks), function(i){
  t <- tracks[[i]]$track
  p <- tracks[[i]]$proj
  po <- SpatialPoints(t[,c('x', 'y')], proj4string=p)
  po <- spTransform(po, CRSobj=CRS('+proj=longlat +datum=WGS84'))
  t$long <- coordinates(po)[,1]
  t$lat <- coordinates(po)[,2]
  return(t)
})

# now we can plot the tracks with a map as reference:
load('data/base.map.RData')
i <- 12
ggplot(map) +
  geom_polygon(aes(x=long, y=lat, group=group), fill='cornsilk3', colour='grey15', size=0.3) +
  geom_path(data=tracks[[i]], aes(x=long, y=lat)) +
  geom_point(data=tracks[[i]][1,], aes(x=long, y=lat), colour='red') +
  geom_point(data=tracks[[i]][nrow(tracks[[i]]),], aes(x=long, y=lat), colour='blue') +
```

```
  theme_bw()
```



Super-imposing gut retention on simulated trajectories

We calculated probabability of dispersal of each seeds using the gut retention curves for small (short retention time) and large seeds (long retention time) as presented in the paper. We used integration to calculate cumulative probabilities, i.e. at each location, we integreated the probability of a seed leaving the duck's gut over five minute intervals (from 5 minutes prior to a location to the time of sampling)

```
# we can also include fasting time prior to migration
# just change the number for the object fast.time below

fast.time <- 0 # time in seconds

# apply this function to each simulated trajectory
tracks <- lapply(1:length(tracks), function(i){

  track <- tracks[[i]]

  # calculate time at each location relative to start of migration (if fast.time == 0)
  # if fast.time is > 0, this time is relative to seed ingestion
  track$time <- seq(0, (nrow(track)-1)*300, 300)/3600 + fast.time

  # compute cumulative extretion probability for small seeds
  track$cum.small <- c(0, unlist(lapply(2:nrow(track), function(j){
    integrate(f=dgamma, lower=track$time[j-1], upper=track$time[j], shape=2.7, rate=0.63)$value
```

```
  }))

  # compute cumulative extretion probability for large seeds
  track$cum.large <- c(0, unlist(lapply(2:nrow(track), function(j){
    integrate(f=dgamma, lower=track$time[j-1], upper=track$time[j], shape=2.7, rate=0.44)$value
  })))

  return(track)
})

# plot trajectories with super-imposed cumulative excretion probability:
# the code below plots excretion probability for small seeds
# exchange "cum.small" for "cum.large" if you like to plot excretion prob. for large seeds
i <- 12
ggplot(map) +
  geom_polygon(aes(x=long, y=lat, group=group), fill='cornsilk3', colour='grey15', size=0.3) +
  geom_path(data=tracks[[i]], aes(x=long, y=lat, colour=cum.small)) +
  scale_colour_viridis_c(name="Excretion probability") +
  geom_point(data=tracks[[i]][1,], aes(x=long, y=lat), colour='red') +
  geom_point(data=tracks[[i]][nrow(tracks[[i]]),], aes(x=long, y=lat), colour='blue') +
  theme(legend.position='bottom')
```